Targeting the Poly (ADP-Ribose) Polymerase-1 Catalytic Pocket Using AutoGrow4, a Genetic Algorithm for *De Novo* Design

by

Jacob Oscar Spiegel

Bachelor of Engineering in Biomedical Engineering, State University of New York at Stony Brook, 2013

Submitted to the Graduate Faculty of the

Dietrich School of Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2020

UNIVERSITY OF PITTSBURGH

DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Jacob Oscar Spiegel

It was defended on

March 10, 2020

and approved by

Dr. Andrew VanDemark, Associate Professor, Department of Biological Sciences

Dr. Jeffrey Lawrence, Professor and Chair, Department of Biological Sciences

Dr. Bennett Van Houten, Professor, Department of Pharmacology and Chemical Biology

Dissertation Director: Dr. Jacob Durrant, Assistant Professor, Department of Biological Sciences

Targeting the Poly (ADP-Ribose) Polymerase-1 Catalytic Pocket Using AutoGrow4, a Genetic Algorithm for *De Novo* Design

Jacob Oscar Spiegel, Ph.D.

University of Pittsburgh, 2020

AutoGrow4 is a free and open-source program for *de novo* drug design that uses a genetic algorithm (GA) to create novel predicted small-molecule ligands for a given protein target without the constraints of a finite, pre-defined virtual library. By leveraging recent computational and cheminformatic advancements, AutoGrow4 is faster, more stable, and more modular than previous versions. Features such as docking-software compatibility, chemical filters, multithreading options, and selection methods have been expanded to support a wide range of user needs. This dissertation will cover the development and validation of AutoGrow4, as well as its application to poly (ADP-ribose) polymerase-1 (PARP-1).

PARP-1 is a well-characterized DNA-damage recognition protein, and PARP-1 inhibition is an effective treatment for ovarian and breast cancers that are homologous-recombination (HR) deficient[1–5]. As a well-studied protein, PARP-1 is also an excellent drug target with which to validate AutoGrow4. Multiple crystallographic structures of PARP-1 bound to various PARP-1 inhibitors (PARPi) serve as positive controls for assessing the quality of AutoGrow4-generated compounds in terms of predicted binding affinity, chemical structure, and predicted protein-ligand interactions.

This dissertation describes how I (1) generated novel potential PARPi with predicted binding affinities that surpass those of known PARPi; (2) validated AutoGrow4 as a tool for *de novo* drug design, lead optimization, and hypothesis generation, using PARP-1 as a test target; (3) contributed support to the growing notion that there is a need for HR-deficient cancer chemotherapies that do not rely on the same set of protein-ligand interactions typical of current PARPi; (4) generated novel potential PARPi that are predicted to bind to PARP-1 independent of a post-translational modification that is known to cause PARPi resistance; and (5) generated novel potential PARPi that are predicted to bind a secondary PARP-1 pocket that is distant from the primary catalytic site.

Table of Contents

List of Tables

List of Figures

List of Equations

List of Appendix JSON Files

List of Abbreviations

ABC	ATP-Binding Cassette
AD	Automodification Domain
ADME-PK	Absorption, Distribution, Metabolism, Excretion, and Pharmacokinetics
ADP	Adenosine Diphosphate
alt-NHEJ	Alternative Non-Homologous End Joining
APLF	Aprataxin and PNK-Like Factor
ART	ADP-Ribosyl Transferase Subdomain
ASL	αD-Active Site Loop
ATM	Ataxia Telangiectasia Mutated
ATP	Adenosine Triphosphate
BAD	Benzamide Adenine Dinucleotide
BER	Base Excision Repair
BRCT	BRCA C-Terminus Subdomain
BRICS	Breaking of Retrosynthetically Interesting Chemical Substructures
CADD	Computer-Aided Drug Design
CAT	Catalytic Domain
c-NHEJ	Canonical Non-Homologous End Joining
CPU	Computer Processing Unit(s)
CRC	University of Pittsburgh's Center for Research Computing

Da	Dalton
DBD	DNA-Binding Domain
DNA-PK	DNA-Dependent Protein Kinase
DSB	Double-Strand DNA Break(s)
ETKDG	Experimental Torsion with Knowledge Distance Geometry
GA	Genetic Algorithm(s)
GPU	Graphics Processing Unit(s)
HD	Helical Subdomain
HR	Homologous Recombination
LBDD	Ligand-Based Drug Design
logP	Octanol–Water Partition Coefficient
MARylation	Mono-ADP-Ribosylation
MMEJ	Microhomology-Mediated End Joining
MPI	Message Passing Interface
MW	Molecular Weight
NAD+	Nicotinamide Adenine Dinucleotide
NHEJ	Non-Homologous End Joining
PAR	Poly (ADP-Ribose)
PARG	Poly (ADP-Ribose) Glycohydrase
PARP	Poly (ADP-Ribose) Polymerase
PARP-1	Poly (ADP-Ribose) Polymerase-1
PARPi	Poly (ADP-Ribose) Polymerase Inhibitor(s)

PDB	Protein Data Bank
PTEN	Phosphatase and Tensin Homolog
pY907	Phosphorylated Y907
QED	Quantitative Estimate of Drug-Likeness
QM	Quantum Mechanics
QSAR	Quantitative Structure-Activity Relationship
QVina2	QuickVina (version 2.1)
RMSD	Root-Mean-Square Deviation
RPA	Replication Protein A
SA_Score	Synthetic Accessibility Score
SBDD	Structure-Based Drug Design
SMI	SMILES File
SMP	Symmetric Multiprocessing
SSB	Single-Strand DNA Break(s)
TOP2	Topoisomerase 2-Alpha
UFF	Universal Force Field
Vina	AutoDock Vina (version 1.1.2)
VS	Virtual Screen
WGR	Tryptophan-Glycine–Arginine-Rich Subdomain
XRCC	X-Ray Repair Cross Complementing Protein
Zn1, Zn2, and Zn3	Zinc Fingers 1, 2, and 3

Preface

Many friends, family members, colleagues, and mentors have contributed, directly or indirectly, to this dissertation.

Over my six-and-a-half years of graduate school I have been blessed with three great research mentors. The first, Dr. Roger Hendrix, was my first thesis advisor. It was a privilege and an honor to know and learn from Roger. While writing this dissertation, I often wondered how Roger would turn a given phrase. I miss him and am grateful to have known him. The second, Dr. Robert Duda, trained and coached me through my years in the Hendrix lab. Nobody has taught me more about phage biology and experimentation than Bob. His patience, instruction, and daily discussions refined my thinking and guided my research. The third, Dr. Jacob Durrant, gave me a second chance when Roger passed. I am eternally grateful to Jacob for his support, kindness, and mentorship. Without these three brilliant men I would not be the person I am today.

Thank you to my committee members, Dr. Jeffrey Lawrence, Dr. Andrew VanDemark, and Dr. Bennett Van Houten. Your counsel and encouragement have been invaluable. Thank you to Dr. Bennett Van Houten for your suggestion to apply AutoGrow4 to PARP-1.

Thank you also to the wonderful friends and teachers who carried me through my years in the Hendrix lab: Dr. Bonnie Oh, Dr. Jianfei Hua, Dr. Patricia Campbell, Dr. Suvrajit Maji, Aletheia Atzinger, and Joshua Maurer. Thank you all for being wonderful lab mates. Additionally, thank you to Dr. Craig Peebles and Dr. Susan Godfrey for your thorough advice and brilliant conversations over the years.

The Durrant lab has been a place of learning and growth for me. I am incredibly grateful to the members of the Durrant lab: Jennifer Walker, Patrick Ropp, Erich Hellemann, Yuri Kochnev, and Kevin Cassidy. Thank you all for being amazing friends and colleagues.

To the teaching minor program and especially my teaching mentor, Dr. Xiaodong Zhu: I thoroughly enjoyed teaching with you. I learned so much from your feedback.

To Kevin Yeh, Jean Olivier Brutus, Abir Deb, and Nic Acton, thank you for your support, friendship, and hours of whiteboarding.

I would not be a fraction of who I am today without the love and support of my family. To my parents, Stacy and Alan, thank you for raising me in a home that values learning, and thank you for encouraging me at an early age to pursue my interests. To my brilliant siblings Jamie, Joshua, and Jonathan, thank you for dinnertime debates and endless laughter. May you all be called doctor soon!

To my lovely wife Pauline, thank you for your love, support, and edits. Your patience, your counsel, and your constant humor made this all possible. This manuscript is dedicated to you.

Dedication

"Standing on the moon with nothing left to do

A lovely view of heaven

But I'd rather be with you."

~Robert Hunter

This dissertation is dedicated to my best friend and wife Pauline. Your love, support, and humor carried me through the ordeals of graduate school. You sing music into my life and make existence sweet. For all that has happened and all that is to come, I am blessed to have you in my life.

1.0 Introduction

This chapter introduces the biological and computational background relevant to AutoGrow4 and its application to poly (ADP-ribose) polymerase-1 (PARP-1). It also describes the current state of the field and the motivation for designing AutoGrow4. It contains expanded adaptations of the work published in the AutoGrow4 manuscript.

AutoGrow4 was published under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium."[6] Several sections presented here are adapted and reprinted with rights and permission:

> **Jacob O Spiegel**†, & Jacob D Durrant. (2020) AutoGrow4: An open-source genetic algorithm for *de novo* drug design and lead optimization. *Journal of Cheminformatics,* 12, 25. https://doi.org/10.1186/s13321-020-00429-4.
>
> † Jacob O. Spiegel should be regarded as first author.

I am the first author of the published manuscript, having written the entire AutoGrow4 codebase, performed most of the experiments discussed in the paper, and analyzed the data. I designed the layout for all figures in the paper with Dr. Jacob Durrant. Dr. Durrant refined and generated the high-quality images for publication. Dr. Durrant also provided guidance and insight as described in the acknowledgement section. All writing in this chapter is original content written by Jacob O. Spiegel.

1.1 Biological Background

DNA is under continuous threat of damage[7]. To respond to different types of damage, cells have evolved several pathways for DNA repair[8,9]. Cancer cells commonly have defects in DNA repair pathways and so are prone to acquiring additional mutations[10–14].

This dissertation focuses on rational drug design targeting the protein PARP-1, which is involved in DNA-damage repair[15,16]. I chose PARP-1 because (1) PARP-1 is a proven cancer drug target[1–5]; (2) its catalytic domain has a well-studied druggable pocket[17–19]; and (3) its multiple biologically confirmed inhibitors serve as positive controls and as source compounds for rational drug design[1–5]. This section begins with a review of three DNA repair pathways that interact with PARP-1: the base excision repair (BER), non-homologous end joining (NHEJ), and homologous recombination (HR) pathways. I then describe how PARP-1 binds to DNA and recruits repair proteins. Finally, I detail the pharmacology of PARP-1, including its role in HR-deficient cancer, current pharmaceutical options for its inhibition, and identified PARP-1 inhibitor (PARPi) resistance mechanisms.

1.1.1 DNA Damage Repair in Humans

I will begin by discussing three PARP-1-implicated DNA repair pathways. This discussion will overview each pathway's target DNA-damage type, key proteins, and general mechanisms for DNA-repair. Collectively, these pathways are critical because DNA is under continuous threat of damage due to exposure to environmental agents, reactive oxygen species, UV-radiation, and

errors made while replicating[7]. As DNA is a complex biological molecule, the possible types of DNA-damage vary greatly. Such damage types include abasic sites, double-strand DNA breaks, and single-strand DNA breaks. Cells therefore require multiple strategies for identifying different types of DNA damage and either repairing or terminating the cell[8,9].

1.1.1.1 Base Excision Repair (BER)

PARP-1 recruits BER repair proteins to sites of DNA damage[20]. The BER mechanism repairs many types of DNA lesions including abasic sites, single-strand DNA breaks (SSB), and various base alterations such as lesions caused by oxidation, alkylation, or deamination[20–22]. These types of DNA damage can stall transcription and translation[23], cause genomic instability, progress into double-strand DNA breaks (DSB)[24], and result in cell death[23]. As its name implies, BER excises small regions of DNA damage and creates an SSB as an intermediate[22]. This "intentionally" formed SSB is usually passed from one protein to the next in the BER pathway to prevent the break from becoming a DSB and to ensure that it is repaired[21,22]. BER also repairs "unintentional" SSB such as those that are caused by reactive oxygen species[21,22].

There are several variations of the BER pathway that share many of the same key proteins. All variations follow five steps[21]. (1) Detect and excise the damaged DNA base, resulting in an apurinic or apyrimidinic site (AP site), also referred to as an abasic site[21]. (2) Incise the AP site (creating an SSB) using an AP endonuclease or AP lyase[21]. (3) Clean up the terminal ends of the sugar backbone[21] using a lyase or a phosphodiesterase[21]. (4) Replace the missing region with

complementary DNA[21] using a DNA polymerase[21]. (5) Ligate the DNA to remove nicks[21] using a DNA ligase[21].

1.1.1.2 Non-Homologous End Joining (NHEJ)

PARP-1 binding and activation can also stimulate a type of non-homologous end joining (NHEJ)[25]. NHEJ is one of the most active repair pathways for DSB and rejoins ends of DSB by direct ligation[25]. However, NHEJ is error prone because the rejoined ends may share little to no homology to guide the repair. A common consequence of NHEJ is loss of DNA bases at the joining site[25]. NHEJ comprises two main sub-pathways: the well-studied canonical NHEJ (c-NHEJ) and the more recently discovered alternative NHEJ (alt-NHEJ)[25]. Alt-NHEJ has also been referred to as "backup NHEJ" and "microhomology-mediated end joining" (MMEJ)[25].

c-NHEJ is activated when the DNA binding proteins Ku70 and Ku80 (Ku proteins) bind to the sites of DSB[24], whereas alt-NHEJ is stimulated by PARP-1 binding and activation[25]. Ku and DNA-dependent protein kinase (DNA-PK) bind to chains of adenosine diphosphate ribose (ADP-ribose) that are produced by PARP-1, and so some models propose that PARP-1 recruits c-NHEJ proteins to the damage sites[24–26]. Contrastingly, PARP-1 and Ku also compete for DSB binding, so PARP-1 may inhibit c-NHEJ function[27]. Ku's stronger affinity and more frequent DSB binding support the view that PARP-1-initiated alt-NHEJ is a backup to Ku-initiated c-NHEJ[27], but these conflicting observations create uncertainty about PARP-1's role in NHEJ repair.

1.1.1.3 Homologous Recombination (HR)

Although PARP-1 does not directly stimulate the HR pathway, cells that are HR deficient (e.g., many breast-cancer cells[9,11,14,28]) tend to be hypersensitive to PARP-1 pharmacological inhibition[1–4]. In fact, drugs that inhibit PARP-1 have been successfully used to treat patients with HR-deficient cancers[1–4]. Thus, understanding HR repair is crucial to understanding PARP-1's impact on many cancer types.

HR uses homologous DNA as a template for DSB repair[24,29]. Unlike NHEJ, which remains active throughout the cell cycle, HR activity is mostly limited to S and G2 phases[9,14,29]. Additionally, HR repair is less error prone than NHEJ[24,29] and is therefore critically important for genome stability and repair fidelity, particularly during DNA replication[14].

Deficiencies in the proteins of the HR pathway are common in many cancers[9,11,14,28]. The breast cancer tumor suppressor proteins BRCA1 and BRCA2 are two frequent examples[9,14]. These proteins are involved in transcription regulation[11], end resection[14,28], filament promotion[14,28], DNA-strand protection at stalled forks[14], cell-cycle checkpoint[11], and cellular apoptosis[11]. HR-deficient cells must rely more heavily on alternative repair pathways such as BER or NHEJ or risk genomic instability[9,28,30]. Therefore, HR-deficient cancers are less able to survive further loss of non-HR repair pathways than are HR-proficient cells[9,28,30].

1.1.2 PARP-1 Recruits BER and Alt-NHEJ With Poly (ADP)-Ribose Chains

In this section, I will discuss the mechanisms by which PARP-1 recruits DNA damage repair proteins to sites of DNA damage. This discussion will include a review of the role that ADP-ribosylation plays in recruiting repair proteins, how PARP-1 identifies DNA damage sites, and the communication between different PARP-1 domains required to coordinate its DNA binding and catalytic activity.

1.1.2.1 ADP-Ribosylation as a Form of DNA Repair Signaling

Nicotinamide adenine dinucleotide (NAD+) is an organic small molecule common to all living organisms[15]. It is composed of an adenosine 5'-monophosphate and a nicotinamide mononucleotide that are covalently connected by a phosphodiester bond[15] (Figure 1). Among its many biological roles, NAD+ can be catabolized into ADP-ribose and nicotinamide by cleaving a N-glycosidic bond[15] (Figure 1). ADP-ribosyltransferases are proteins that catalyze this reaction and attach the resulting ADP-ribose molecule(s) onto acceptor protein(s)[15]. This post translational modification is referred to as mono-ADP-ribosylation (MARylation) when a single ADP-ribose is attached, or poly-ADP-ribosylation (PARylation) when multiple ADP-ribose molecules are attached[15]. Poly (ADP-ribose) (PAR) chains can be connected either as a linear chain or as branching chains[15] (Figure 1B).

Poly (ADP-ribose) polymerases (PARPs) are a family of ADP-ribosyltransferases found in eukaryotes. PARP proteins can transfer ADP-ribose to glutamic-acid, aspartic-acid, cysteine,

lysine, and serine acceptor sites[31–34]. Among the 17 PARP proteins in humans, PARP-1 was the first identified and is the most studied[35,36]. Multiple members of the PARP family, including PARP-1, can act both as ADP-ribosyltransferases and PARylation acceptors[15].

PAR chains are thought to act as a scaffold for recruiting DNA repair proteins to sites of DNA damage[24,37]. Many DNA repair proteins, including p53, DNA ligase III, X-ray repair cross complementing protein 1 (XRCC1), DNA-PK(CS), and Ku70, contain a PAR-binding motif that promotes non-covalent interactions with PAR-chains[38]. These PAR-binding motifs are typically 20 amino acids long and consist of two conserved regions: a region dense in basic amino acids, and a region consisting of both hydrophobic and basic amino acids[38]. The location of PAR-binding motifs coincides with five different functional domains that are involved in nuclear localization, protein degradation, nuclear export, protein-protein interactions, and DNA binding[38]. Several other structures also facilitate protein-PAR noncovalent interactions, including the PAR-binding zinc fingers found in the NHEJ protein aprataxin and PNK-like factor (APLF)[15,39], tryptophan and glutamic-acid rich WWE domains found in E3 ligases[15,40], and macro domains found in PAR glycohydrase[15,41,42].

When attached to PARP-1, the negatively charged PAR chains may also help PARP-1 dissociate from negatively charged DNA[28], enabling DNA-repair-protein access to the damage site[43]. These PAR chains are rapidly degraded by PAR glycohydrase or PAR hydrolase[28,33]. Both hyperactivation of PARP-1 and inhibition of PAR glycohydrase can cause adenosine triphosphate (ATP) and NAD+ depletion, leading to a form of metabolic death known as parthanatos[12,26,28,44].

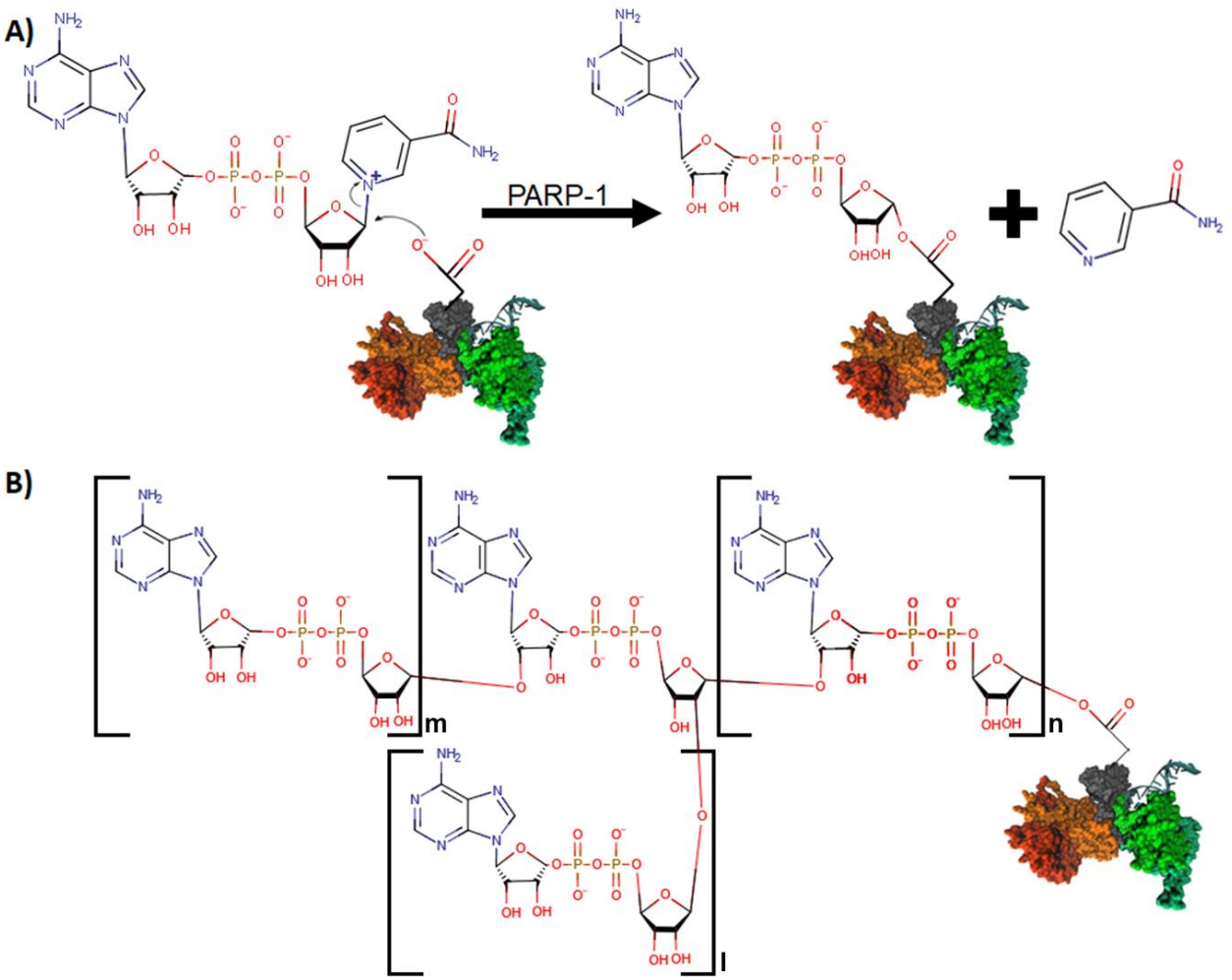


Figure 1. The mechanism of PARylation.

A) The first ADP-ribose modification attached to an aspartic-acid sidechain by PARP-1. Nicotinamide is a byproduct of this reaction. B) The attachment and elongation of the PAR chain, both as a linear chain and as branching chains. Square brackets with the letters l, m, and n indicate repeatable units that could be extended to form a longer or more branched PAR chain. The acceptor protein shown in this figure is derived from PDB 4DQY[45] and can be viewed in larger and labeled scale in Figure 2B (p.11). This figure is adapted with permissions from figures presented in Lin (2015)[15] and Alemasova and Lavrik (2019)[16].

1.1.2.2 PARP-1 DNA Damage Recognition and Binding

Understanding PARP-1/DNA interactions is critical given that one of PARP-1's most important functions is to identify and bind to DNA damage sites. The PARP-1 N-terminal DNA-binding domain (DBD) is responsible for PARP-1's DNA binding activity. The DBD includes three DNA-binding zinc fingers (Zn1, Zn2, and Zn3)[17,45,46] (Figure 2) that differ from many other DNA-binding zinc fingers in that they recognize DNA structures rather than specific DNA sequences[33,47]. Zn1 and Zn2 identify many types of DNA lesions with high affinity[15,45,47,48]. Without Zn1, Zn2, and Zn3, a SSB cannot trigger PARP-1 DNA-dependent catalytic activation[48–51]; without Zn1 and Zn3, a DSB cannot trigger PARP-1 DNA-dependent catalytic activation[15,45–48]. Simultaneous deletion of Zn1 and Zn2 reduces PARP-1 binding affinity by over 250-fold, and a fragment containing only Zn1 and Zn2 binds to DNA with nearly the same affinity as wildtype PARP-1[47]. Isolated Zn1 fragments have much weaker binding affinities for DNA than do isolated Zn2[47]. But Zn2 deletion does not impact the binding affinity of PARP-1, and Zn1 deletion results in an approximately three-fold reduction in binding affinity[47]. Despite that fact that Zn2 has a much stronger affinity for DNA, Zn1 combined with the C-terminal region of the DBD can apparently compensate for the loss of Zn2[47].

Zn3, which has a different structure and function than Zn1 and Zn2[45,47,52], is thought to make crucial interdomain contacts that are required for PARP-1 activation[45,47] (Figure 2B). Similarly, the tryptophan-glycine-arginine-rich subdomain (WGR) also forms contacts with the DNA and contributes to important interdomain contacts (Figure 2B). I discuss these interdomain contacts and the roles they play in modulating PARP-1's catalytic activity in "Chapter 1.1.2.5:

PARP-1 Catalytic Activation is Modulated by Interdomain Interactions" and "Chapter 1.1.2.6: DNA-Unbinding is Regulated by Interdomain Interactions."

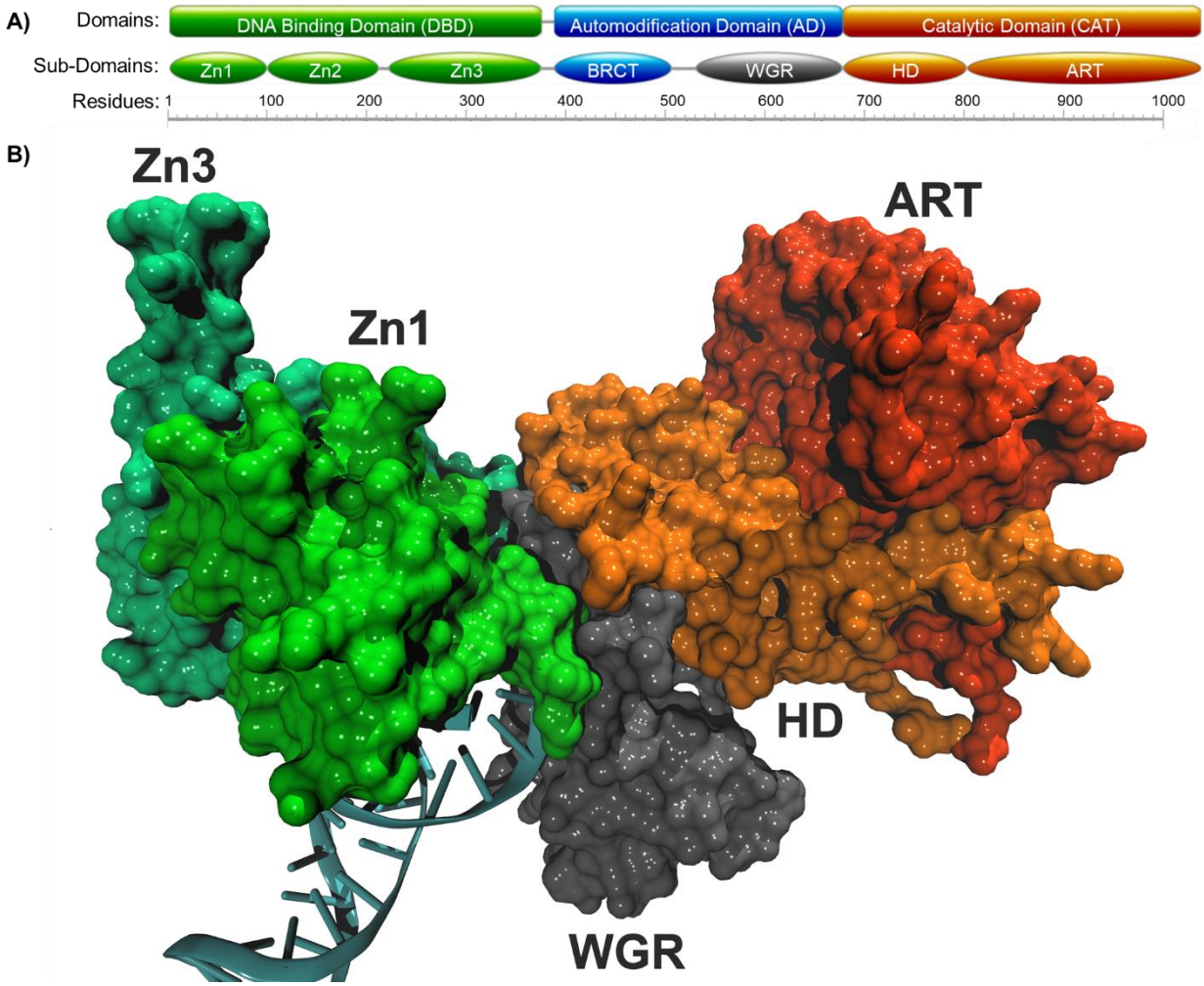


Figure 2. Representation of the three domains and seven subdomains of PARP-1.
A) The DNA binding domain (DBD) is shown in green, the automodification domain (AD) is shown in blue, and the catalytic domain (CAT) is shown in orange. There is some disagreement as to whether the WGR is its own domain or part of the AD or CAT, hence WGR is shaded gray[17,46,53]. Panel A was generated with the help of PROSITE[54]. B) PARP-1 bound to DNA (PDB ID: 4DQY)[45]. 4DQY is one of the most complete PARP-1 structures and contains the Zn1 (dark green), Zn3 (light green), WGR (gray), helical subdomain (HD) (light orange), and the ADP-ribosyl transferase subdomain (ART) (reddish orange). DNA is shown in cyan. The Zn2 and BRCA C-terminus (BRCT) subdomains are not shown, as they are not present in the crystal structure[45].

1.1.2.3 PARP-1 Catabolizes NAD+ to Perform ADP-Ribosylation

A primary function of PARP-1 is to act as an ADP-ribosyltransferase, which converts NAD+ to ADP-ribose and covalently attaches the ADP-ribose to an acceptor site. It has a highly conserved catalytic domain (CAT) that is responsible for binding NAD+ and catalyzing the reaction. Understanding the mechanisms of NAD+ binding and catalysis, as well as the structures that support that catalysis, is relevant to this dissertation because all FDA-approved PARP-1 inhibitors are NAD+ competitive inhibitors, and "inhibition of PARP-1" is most often measured in terms of PARP-1 catalytic activity[17].

PARP-1's CAT domain is located on the C-terminal region of the protein. As shown in Figure 2 (p. 11), the CAT is comprised of the helical subdomain (HD) and the ADP-ribosyl transferase subdomain (ART)[17,46]. The ART subdomain houses the catalytic pocket that is responsible for PAR synthesis. The same pocket also binds NAD+ and all FDA-approved PARP-1 inhibitors[17] (Figure 3A and B, respectively). When fully folded, the HD has an autoinhibitory function, blocking NAD+ from binding to the catalytic pocket[18] and limiting PARP-1's catalytic activity. PARP-1 binding to DNA triggers conformational changes in the automodification domain (AD)[18] that propagate to the HD via contacts with the WGR, resulting in the local unfolding of HD[18]. This unfolding enables successful NAD+ binding and PARP-1 catalytic activity (i.e., the catalytic activity is DNA dependent)[18]. Mutagenesis of key residues (A744, A870, G871) at an HD-WGR interface results in 3- to 10-fold increases in PARP-1 DNA-independent activity (i.e., PARP-1 catalytic activity when not bound to DNA) compared to the wildtype[18].

PARP-1 ADP-ribosylation begins with NAD+ binding to the catalytic site[16]. A non-hydrolyzable NAD+ analogue (benzamide adenine dinucleotide, or BAD) has been co-crystallized with the CAT domain (PDB: 6BHV[55]) (Figure 3A) and shows how NAD+ binds within the PARP-1 catalytic pocket[55]. The pocket houses a highly conserved catalytic triad consisting of H862, Y896, and E988 (Figure 3A), which are required for catalytic activity[16]. H862 positions the NAD+ molecule by binding to the 2' adenine-ribose hydroxyl group[16,56–58]. Y896 positions the aromatic ring of the nicotinamide moiety by forming a π-π stacking interaction, which is further assisted by similar interactions between Y907 and NAD+[56] (Figure 3A). E988 binds to the 2′-hydroxyl group of the nicotinamide ribose, which orients NAD+ as required for a nucleophilic attack[56] (Figure 3A). This nucleophilic attack both catabolizes NAD+ and covalently attaches the resulting ADP-ribose to the acceptor protein[56].

The catalytic pocket can be separated into four regions with distinct biochemical functions: the nicotinamide-binding pocket (NI site), the adenine-ribose binding site (AD site), the phosphate-binding site (PH site), and the acceptor site[56]. The NI site stabilizes the nicotinamide moiety of NAD+ and is defined by the catalytic triad residues E988 and Y896, as well as Y907[56]. The AD site stabilizes the adenine-ribose of the NAD+ and is defined by the catalytic triad residue H862, as well as G876, D770, S864, and R878[56]. The PH site, which consists of D766 and E763, stabilizes the pyrophosphate group of NAD+[56]. Collectively, the NI, AD, and PH sites make up a larger region referred to as the donor site, as these three subregions contribute to the binding and positioning of the NAD+ "donor" molecule[56]. The acceptor site binds and positions the pyrophosphate group of the "acceptor" residue or PAR chain that will be ADP-ribosyltated[56]. The acceptor site consists of H826, M890, K903, L985, and Y986[16,56] and is

thought to influence the branching properties of the PAR chains[16,59]. The majority of all PARP-1 inhibitors (PARPi) bind to PARP-1's donor site[16,59]; for instance, the FDA-approved PARPi niraparib is shown binding to the donor site in Figure 3B.

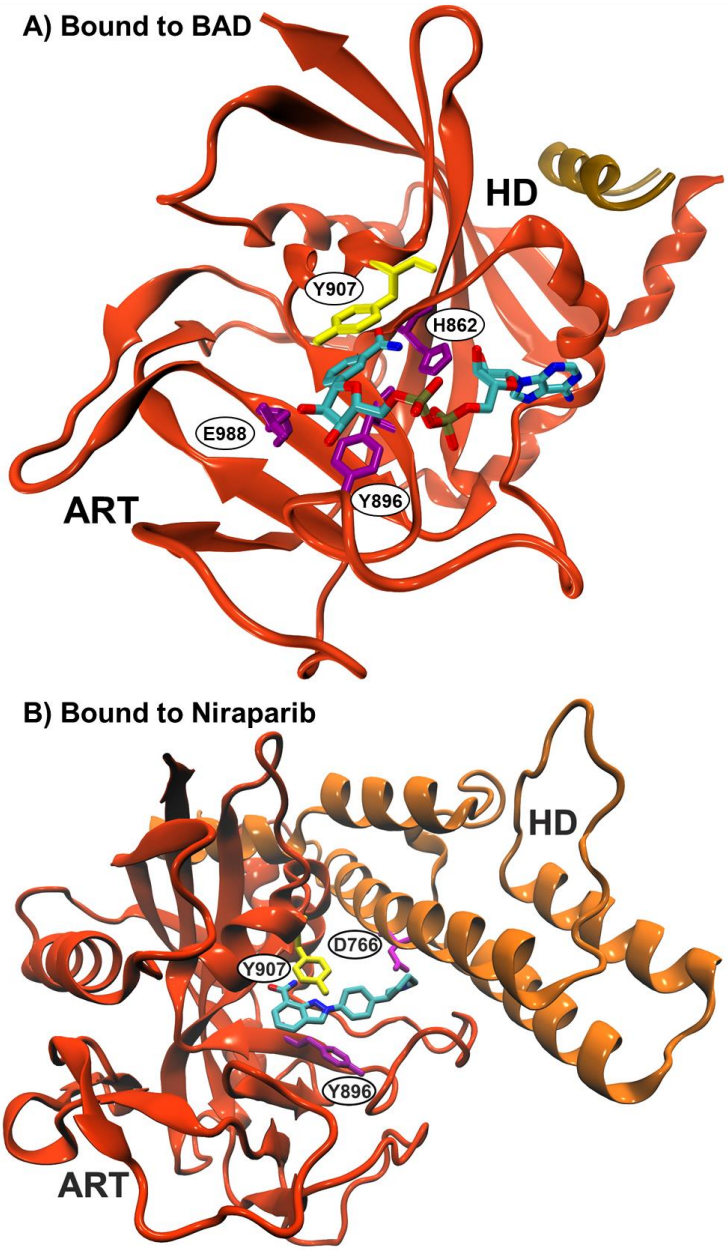


Figure 3. PARP-1 catalytic binding site.

A) An NAD+ analog, BAD (shown in multi-color stick representation), bound to the PARP-1 catalytic pocket (Protein Data Bank (PDB) ID: 6BHV Chain A[55]). The catalytic triad (purple) binds to and catabolizes NAD+. Y907 and the catalytic triad residue Y896 position the nicotinamide-like benzamide moiety by forming π-π stacking interactions. NAD+ likely binds within the catalytic pocket with a similar orientation. B) Niraparib (cyan stick representation), an FDA-approved PARPi for treating ovarian cancer[4,17], bound to the PARP-1 catalytic pocket (PDB ID: 4R6E Chain A[17]). This structure shows how the HD, shown in light orange, and the ART, shown in reddish orange, form contacts with the niraparib. Select residues that interact with niraparib are shown in colored-stick representation.

1.1.2.4 PARP-1 Acts as an ADP-Ribosyltransferase and ADP-Ribose Acceptor

PARylation modifications can attach to many nuclear proteins, including PARP proteins[60]. This dissertation focuses on PARP-1 drug-inhibition strategies, which generally aim to prevent PARP-1-mediated PARylation events. Understanding the protein targets of PARylation and the relationship between PARP-1 and those targets provides insight into potential inhibition strategies.

PARP-1 can ADP-ribosylate (1) itself (*in cis* modification) as a monomer[16]; (2) other PARP-1 molecules (*in trans* modification), both as symmetric and asymmetric homodimers[16]; and (3) other proteins[16], such as XRCC1, XRCC5, XRCC6, DNA topoisomerase 2-alpha (TOP2), replication protein A1 (RPA1), and RPA2[60].

The primary acceptor of PARP-1-mediated PARylation is PARP-1 itself[33,45]. PARylation of PARP-1 is called autoPARylation[9,15,16]. There are at least 102 potential PAR-acceptor sites on PARP-1[31,61–67] (Figure 4). Although there are PAR-acceptor sites throughout the entire PARP-1 protein, the automodification domain (AD) houses many of the most well-characterized autoPARylation sites[31,33,45,61–67] (Figure 4). The AD contains two subdomains, the BRCA C-terminus subdomain (BRCT) and the WGR[17,46] (Figure 2 on p.11). The WGR is essential for coordinating catalytic activity with the DBD in binding to DNA damage[45] (Figure 2 on p.11). The BRCT subdomain plays important roles in protein-protein interactions, such as binding the BER protein XRCC1 to PARP-1[68,69], but it is not essential for PARP-1 binding to DNA or PARP-1 catalytic activity[45]. The WGR subdomain is essential for the DNA-dependent catalytic activation of PARP-1 because of its roles in interdomain communication[45] (see the following section).

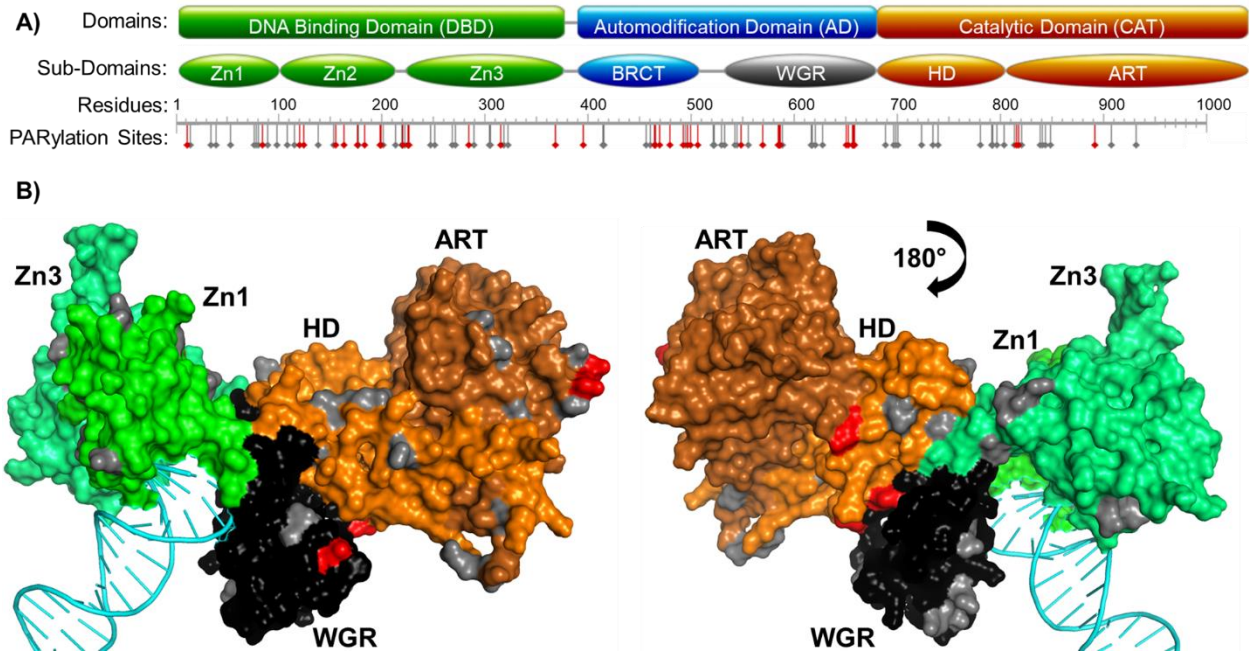


Figure 4. PARylation acceptor sites mapped onto PARP-1.

A) DBD is shown in green, AD is shown in blue, and CAT is shown in orange. WGR is shaded gray[17,46,53]. Verified PARylation sites compiled by Daniels *et al.* (2015)[31] from seven studies on PARP-1 autoPARylation[61–67] are shown in gray if they were identified by only one source paper[31,61–67], and red if they were identified by two or more independent studies[31,61–67]. Panel A was generated with the help of PROSITE[54]. B) PARP-1 bound to DNA (PDB ID: 4DQY[45]). 4DQY is one of the most complete PARP-1 structures and contains the Zn1 (green), Zn3 (light green), WGR (black), HD (light orange), and ART (brownish orange) subdomains. DNA is shown in cyan. The Zn2 and BRCT subdomains are not shown because they are not present in the crystal structure[45]. Views are ~180° rotations of each other. The verified PARylation sites are mapped onto the surface of the structure and shown in gray if they were identified by only one source paper, or red if they were identified by two or more independent studies[31,61–67].

1.1.2.5 PARP-1 Catalytic Activation is Modulated by Interdomain Interactions

In wildtype PARP-1, the catalytic activity of DNA-bound PARP-1 is significantly elevated compared to the low basal level of DNA-independent catalytic activity[33,45]. This modulation of PARP-1 catalytic activity is controlled through a network of interdomain interactions that enable allosteric communication[18,45]. Small molecules that disrupt interdomain communication thus have potential as PARP-1 drugs/inhibitors (see "Chapter 3.3.5: AutoGrow4 Applied to a Non-catalytic PARP-1 Pocket"). To prepare the reader for further discussion of allosteric PARP-1 inhibition, I will here discuss each of these interdomain interfaces in detail.

The minimum PARP-1 protein construct capable of near-wildtype levels of DNA-dependent PARP-1 activity contains the Zn1, Zn3, WGR, and CAT domains[45]. Zn1, Zn3, and the WGR are essential for DNA-binding and interdomain allosteric communication with the CAT, and the CAT domain is essential for catalytic activity[45]. Several key interdomain interfaces regulate DNA-dependent PARP-1 activity: (1) the Zn1-Zn3 interface (Figure 5A and B); (2) the Zn1-WGR-HD interface (Figure 5A and C); (3) the HD-WGR-Zn3 interface (Figure 5A and D); and (4) the two HD-ART interfaces (Figure 6).

Zn3 binds a surface on Zn1 that is exposed when Zn1 binds DNA, forming the Zn1-Zn3 interface[16,70]. The Zn1-Zn3 interface primarily consists of the C-terminal helix of Zn1 and the N-terminal helix of Zn3[45] (Figure 5A and B). It is close to but not in direct contact with the bound DNA, per the PDB:4DQY crystal structure[45]. Much of what we know of this interface comes from structural and mutagenesis studies. Several key residues have been mutated within this interface, including Zn1's L77, R78, W79, and K97, as well as Zn3's K238, W246, and K249[45,46,52,70,71]

(Figure 5B). Of particular note are the conservative mutation K97R and the non-conservative mutations L77P and K249E, which reduce PARP-1 catalytic activity to <0.5% as compared to the wildtype, despite retaining DNA-binding activity[71].

The Zn1-Zn3 interface is critical for the allosteric communication that activates DNA-dependent catalytic activity, but that interface does not influence DNA binding itself[45,46,52,70]. Mutations that disrupt major contacts between Zn1 and Zn3 (i.e., R78/Zn1 and W246/Zn3; and W79/Zn1 and K238/Zn3[52]; Figure 5B) decrease catalytic activity without disrupting DNA binding activity[45,46,52,70]. For instance, the W246A mutant had virtually no catalytic activity[45,70] and could not form the DNA-dependent Zn1-Zn3 interaction[70]. Additionally, R78A and W79A mutations both resulted in markedly decreased catalytic activity, but had DNA-binding activity nearly equal to that of the wildtype[46]. I will discuss the Zn1-Zn3 interface further in "Chapter 3.0AutoGrow4: Application to Poly (ADP-ribose) Polymerase-1 (PARP-1)", in the context of potential drug-discovery strategies for PARP-1 inhibition.

The Zn1-WGR-HD interface (Figure 5A and C) plays an important role in regulating DNA-dependent PARP-1 catalytic activity. The WGR acts as an intermediate between DNA binding and catalytic activity through contacts it forms with the DNA, Zn1, and the HD αE helix[45] (Figure 5C). The WGR W589 residue stacks with the ribose sugars of the DNA 5' strand[45]. WGR R591 forms a salt bridge with Zn1 D45 and makes contacts with the HD αE helix[45]. R591 thus serves as a bridge between the Zn1 and HD subdomains. As expected, the R591A mutation disrupts PARP-1 autoPARylation activity[45]. D45A and Q40A also reduce PARP-1 autoPARylation activity[45,47]. In summary, the Zn1-WGR-HD interface regulates DNA-dependent catalytic activity[45,47].

Like the Zn1-Zn3 and Zn1-WGR-HD interfaces, the HD-WGR-Zn3 interface also plays a significant role in DNA-dependent catalytic activation and interdomain communication. The HD αE helix interacts with the WGR and the extended loop of the Zn3 zinc ribbon fold, with Zn3 W318 at the center of the interface[45] (Figure 5A and D). W318 is on a loop that interacts with the HD R735 residue and the WGR K633 residue. W318R and W318A mutations reduce PARP-1 PAR synthesis activity, without impairing localization to sites of DNA damage[18,45,46]. W318R is thought to disrupt this interface, thereby preventing/limiting interdomain allosteric communication[18,45]. Similarly, mutations R735A and K633A reduced PARylation activity[45,46].

The two HD-ART interfaces, the αF-αJ interface and αD-active site loop (ASL) interface (Figure 6), influence the HD's structure and position relative to the catalytic pocket and thereby regulate PARP-1 catalytic activity[18]. Truncated PARP-1 constructs without the HD subdomain had levels of DNA-independent catalytic activity equal to that of the DNA-bound wildtype protein, indicating that the HD has an inhibitory effect on PARP-1 catalytic function[18]. PARP-1 catalytic activity is possible only when the HD subdomain undergoes a conformational change, which opens the catalytic pocket and allows NAD+ to bind[18]. This DNA-dependent conformational change involves the unfolding of select HD helical regions, including αB, αD, and the C-terminal of αF[18]. In particular, the unfolding of the HD αF has a profound effect on catalytic activity because it disrupts the HD's autoinhibitory effect on the ART αJ and allows NAD+ binding[18]. Consequently, the structure of the αF-αJ interface (Figure 6A and C) must be disrupted to allow catalytic activity. In fact, mutations that disrupt the αF-αJ interface (e.g., A774L, A774S, A870L, A870S, A871L, A871S) result in 3- to 10-fold increases in PARP-1 DNA-independent activity[18]. In contrast, the αD-ASL interface is thought to serve as a structural support anchoring the HD and ART together

in both the active and inactive states[18,55]. Mutations that disrupt the αD-ASL interface (e.g., S714A, Q717A, P882G, P885S, P885G, and the quadrupole substitution mutant E883A/A884S/P885S/V887A) largely reduce DNA-dependent activity and minimally affect DNA binding and DNA-independent activity[18]. The exception to this is the L713F mutation, which increases DNA-independent activity[18,45,72]. In summary, the HD domain acts as an intermediate between the PARP-1 DNA-interfacing domains and the ART subdomain.

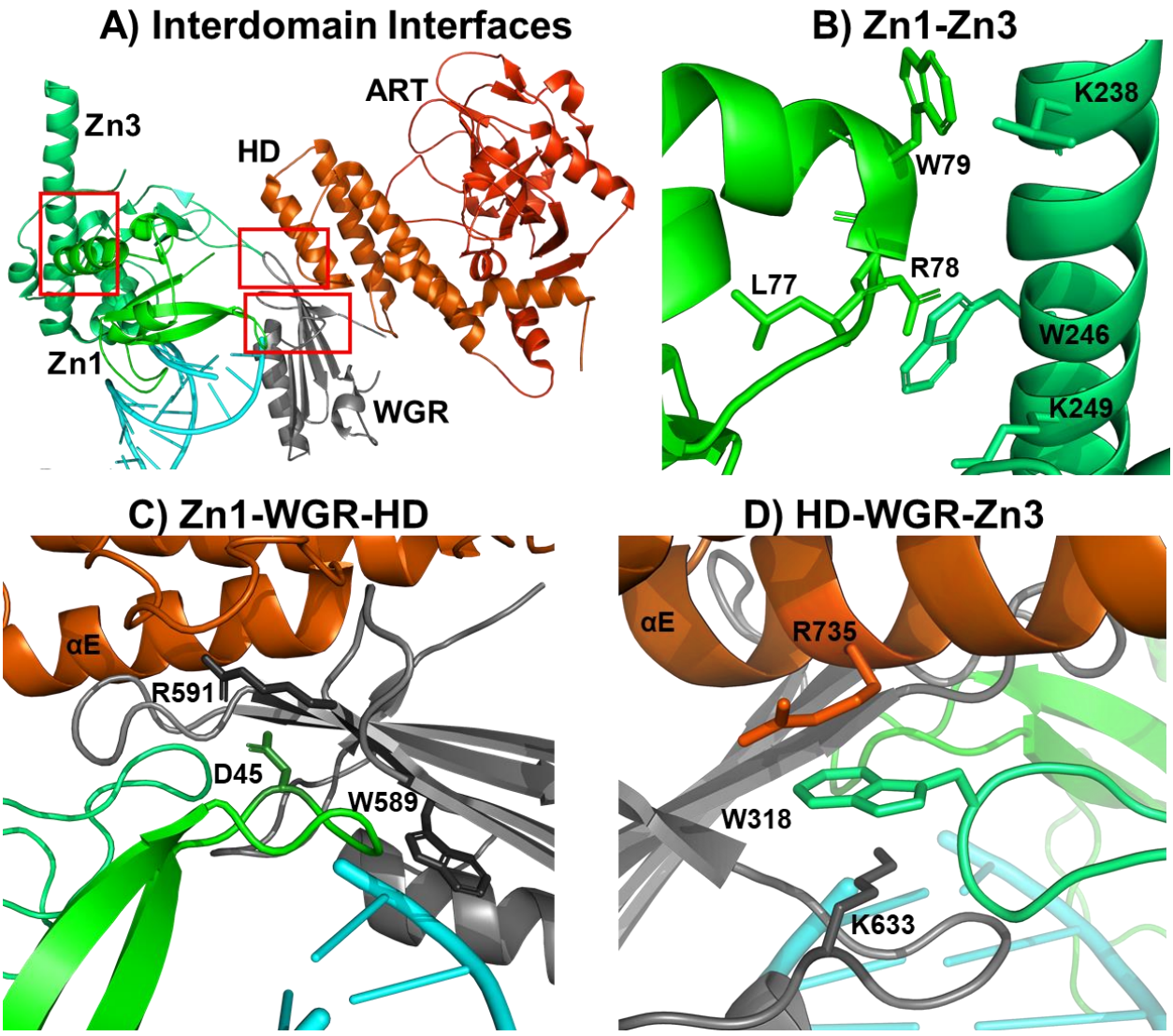


Figure 5. PARP-1 interdomain interfaces.

A) A view of near full-length PARP-1 (PDB ID: 4DQY[45]) with red boxes encompassing three critical interdomain interfaces. B) A zoomed-in view of the Zn1-Zn3 interface with several key residues that were mutated in previous studies[45,70,7145,46,52,70,71], shown in stick representation. K97 was omitted, as it was not part of the crystal structure[45]. C) A zoomed-in view of the Zn1-WGR-HD interface with several key residues that were mutated in previous studies[45], shown in stick representation. D) A zoomed-in view of the HD-WGR-Zn3 interface with several key residues that were mutated in previous studies[45,47], shown in stick representation. PARP-1 subdomain Zn1 (green), Zn3 (light green), WGR (black), HD (light orange), and ART (brownish orange) are shown in cartoon representation, and DNA is shown in cyan. This figure has been adapted with rights and permissions from figures presented in Langelier *et al.* (2012)[45].

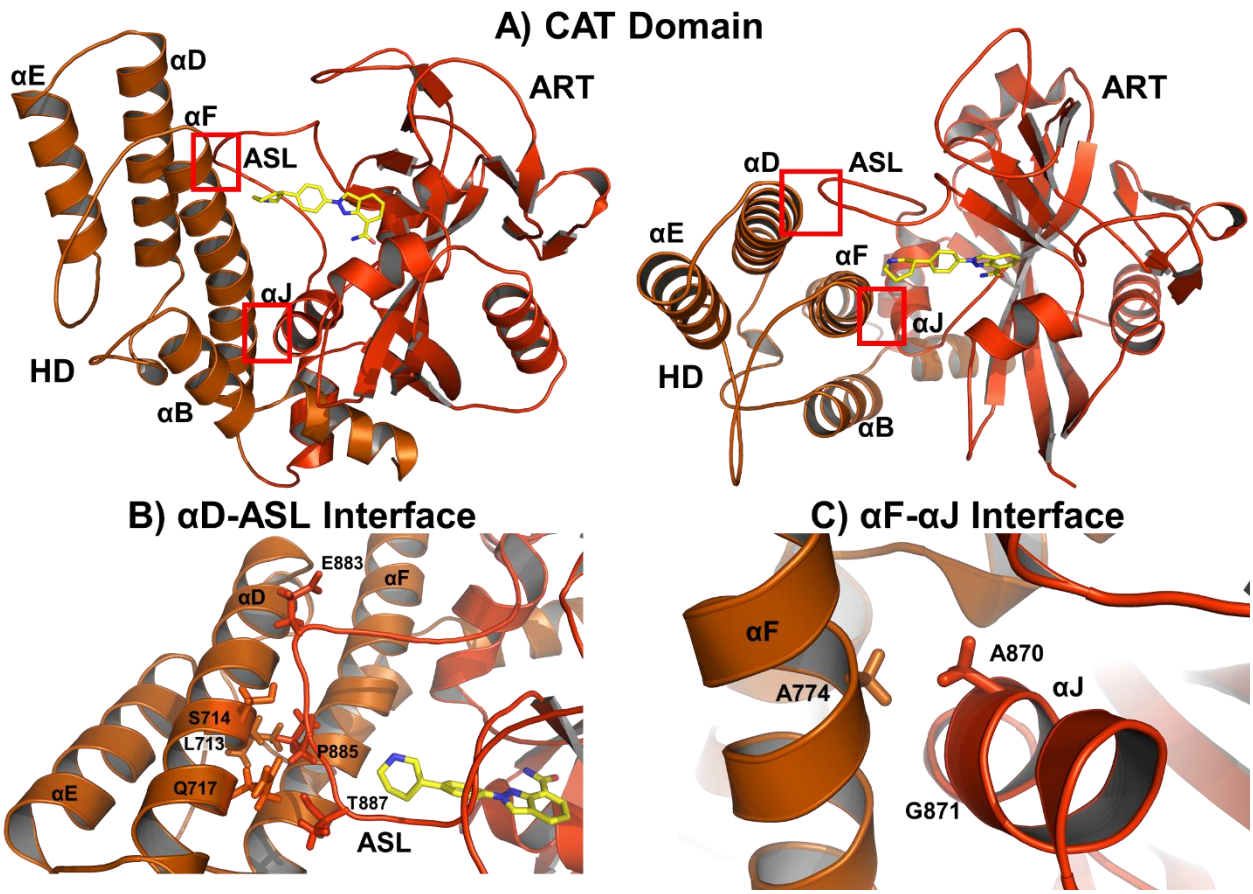


Figure 6. Critical interfaces within PARP-1's CAT domain.

A) The PARP-1 CAT domain (PDB ID: 4R6E Chain A[17]) with red boxes encompassing two critical interfaces between the HD (light orange) and ART (brownish orange) subdomains. Two views of the same structure are provided to better show the two interfaces. Niraparib (yellow stick representation), an FDA-approved PARPi for treating ovarian cancer[4,17], is bound in the crystal structure and illustrates the location of the NI site relative to these interfaces. Several key structural elements are labeled. B) A zoomed-in view of the interface between the αD helix of the HD and the active site loop (ASL) of the ART. Several key residues that were mutated in previous studies[18] are shown in stick representation. C) A zoomed-in view of the interface between the αF helix of the HD and the αF helix of the ART. Several key residues that were mutated in previous studies[18,45,72], shown in stick representation. This figure has been adapted with rights and permissions from figures presented in Dawicki-McKenna *et al.* (2015)[18].

1.1.2.6 DNA-Unbinding is Regulated by Interdomain Interactions

PARP-1 allostery has historically been discussed in terms of how binding to DNA influences catalytic activity[4,17,45], but in recent years the research has shifted to studying how the PARP-1 CAT influences DNA unbinding[55,73]. In this section, I will discuss PARP-1 release from DNA and how that mechanism informs the design of novel PARPi.

Many NAD+ analogs that bind to the catalytic pocket (e.g., the nonhydrolyzable NAD+ analog BAD[55], Figure 3, p.15) immobilize PARP-1 on DNA, a phenomenon referred to as "trapping"[5,13,55,74–77]. Trapping PARP-1 on DNA prevents DNA-repair proteins from accessing sites of DNA damage, stalls and/or collapses replication forks, and results in cell-cycle arrest or apoptosis[5]. The fact that catalytic-site occupancy influences DNA affinity and that different small-molecule ligands impact trapping to varying degrees[55,75,76] suggests that there is some allosteric regulation controlling DNA binding[55,73].

The degree to which a small-molecule catalytic-pocket ligand traps PARP-1 on DNA appears to be related to its ability to (de)stabilize the HD[73]. HD destabilization enables allosteric communication between the CAT and the DBD, resulting in stronger PARP-1 affinity for DNA and PARP-1 trapping[73]. In contrast, stabilizing the αB and αF helixes of the HD causes allosteric communication that weakens affinity and releases PARP-1 from DNA[73]. Small molecules that interact with the HD αF helix, such as the NAD+ analogs BAD and EB-47, tend to destabilize the HD and consequently have strong trapping capacity[55,73].

Several published experimental findings support this proposed trapping mechanism. First, a double mutant created by substituting alanine for D766 and D770 (two residues on the HD αF

helix that contacts EB-47 when EB-47 is bound to the catalytic pocket) prevented EB-47 from trapping PARP-1[73]. Second, veliparib, a PARPi that does not form contacts with the HD (Figure 7A, C and E) has a weaker trapping capacity and weaker toxicity to BRCA-deficient cells than does UKTT-15, a veliparib-derived inhibitor that contacts the N-terminal portion of the αF helix (Figure 7C-E), despite their similar abilities to inhibit PARylation[73]. Third, W318R and R591C, mutations that disrupt the HD-WGR-Zn3 and Zn1-WGR-HD interfaces, respectively, prevent EB-47 trapping activity. This suggests that PARP-1 trapping relies on allosteric interdomain communication. In fact, the R591C mutation was recently discovered in an ovarian cancer patient that had developed a resistance to the PARPi talazoparib[73,78]. Taken together, these recent findings suggest that the trapping capacity of a PARPi is linked to its ability to form stable contacts with HD's αF helix, and that trapping is controlled by allosteric communication via interdomain contacts[73].

Based on these results, a recently developed PARPi classification system now groups inhibitors into three types[73]: type I, which promote PARP-1 retention on DNA through allosteric regulation, resulting in long DNA-retention times and high toxicity[73]; type II, which are neither pro-retention nor pro-release (i.e., relatively allosterically neutral), resulting in moderate DNA-retention times and toxicity[73]; and type III, which promote PARP-1 release through allosteric mechanisms, resulting in short DNA-retention times and lower toxicity[73].

As examples, consider the PARPi veliparib and UKTT-15. Veliparib is a type III PARPi because it is unable to make contact with the HD (Figure 7C and E) and promotes PARP-1 dissociation from DNA[73]. In contrast, UKTT-15 is a type I PARPi because it not only contacts but disrupts the HD (Figure 7D and E) and so promotes trapping PARP-1 on DNA[73]. I will further

discuss the drug-discovery implications of trapping in “Chapter 1.1.3: PARP-1 Roles in Cancer and Cancer Treatments.”

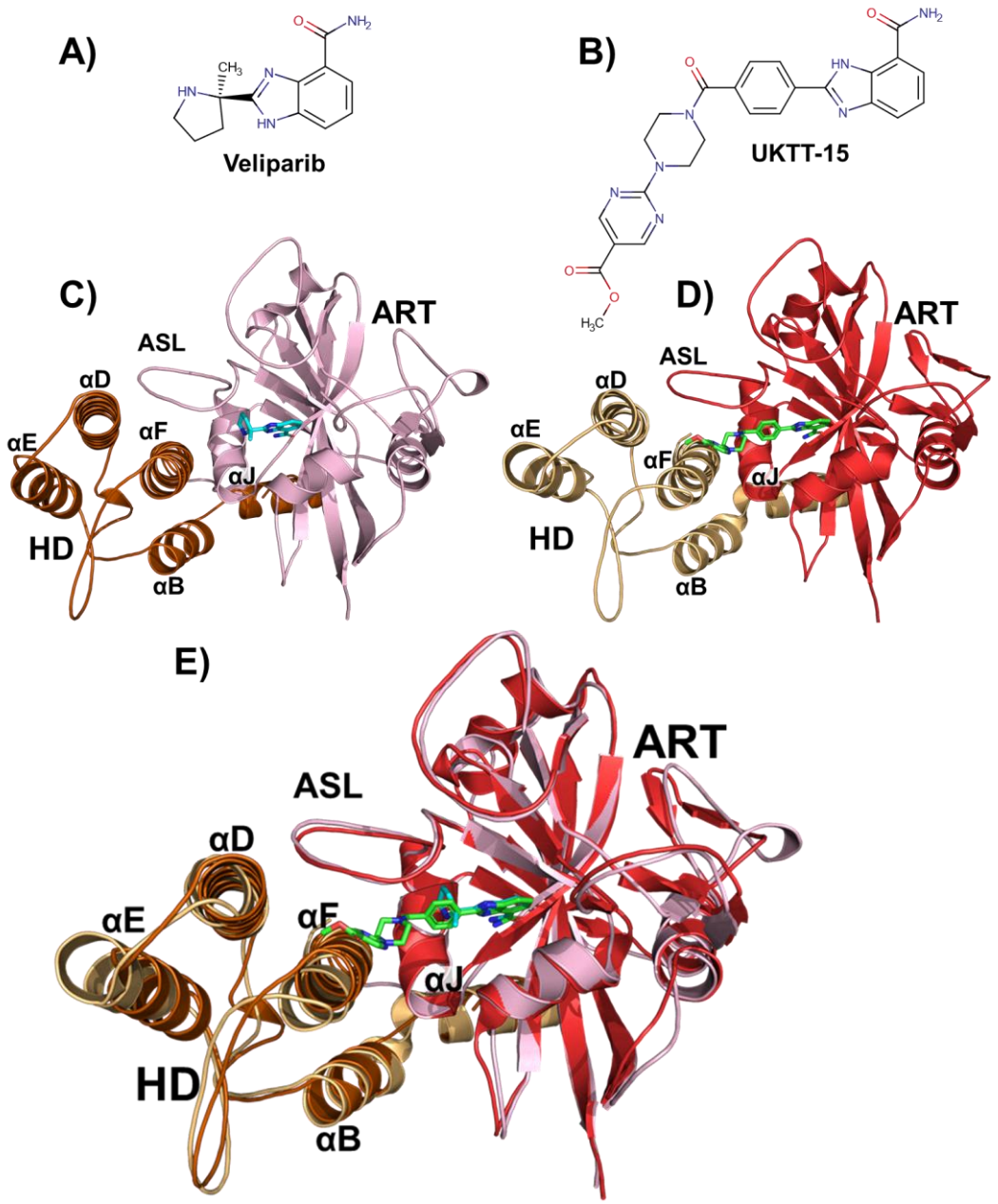


Figure 7. Differences in veliparib and UKTT-15 binding.

A) The structure of the PARP-1 inhibitor veliparib. B) The structure of the veliparib analog UKTT-15. C) The structure of the PARP-1 catalytic domain bound to veliparib (cyan) (PDB ID: 2RD6 Chain A[79]). The HD is shown in brownish orange, and the ART is shown in pink. D) The structure of the PARP-1 catalytic domain bound to UKTT-15 (green) (PDB ID: 6VKO Chain A[73]). The HD is shown in red, and the ART is shown in beige. E) The aligned 6VKO and 2RD6 structures illustrate that the shared 1H-1,3-benzodiazole-7-carboxamide groups of both ligands are bound in the same position. Veliparib does not contact the HD, but the extended moieties of UKTT-15 that contact the HD αF helix. The αF is displaced to accommodate UKTT-15 binding[73].

1.1.3 PARP-1 Roles in Cancer and Cancer Treatments

In this section, I will (1) discuss the statistics and standard treatment for patients with HR-deficient cancers; (2) describe the mechanisms that make drug inhibition of PARP-1 lethal to HR-deficient cancers; (3) overview current FDA-approved PARPi; (4) document common PARPi-resistance mechanisms; and (5) describe the current state of the field, including the shortcomings of current pharmaceutical options and how they can be improved. This section will prepare readers for "Chapter 3.0: AutoGrow4: Application to Poly (ADP-ribose) Polymerase-1 (PARP-1)," where I will describe my efforts to design novel PARPi candidates.

1.1.3.1 The Prevalence of and Standard-of-Care Management for HR-Deficient Cancers

Breast, ovarian, prostate, and pancreatic cancers are frequently HR deficient[80–83]. There were over two million new cases of breast cancer in 2018, which resulted in nearly 630,000 deaths[80,84]; approximately 30% of all hereditary breast cancers have HR defects due to mutations in the *BRCA1/2* genes[80]. Ovarian cancer is also common, affecting 11.8 per 100,000 women in the United States in 2014[85]. Approximately half of all epithelial ovarian cancers, which make up 90% of ovarian cancers[85], have HR defects, with BRCA1/2 defects being the most common[81]. Prostate cancers are also frequently HR-defective[82]. It is estimated that 5-10% of prostate adenocarcinomas have BRCA1/2 defects[82]. Lastly, one study found that individuals with a *BRCA2* mutation were nearly 22 times more likely than expected to develop pancreatic cancer, with men at a much higher

risk[86]. Given the high rates of HR-deficient cancers, finding effective treatments with improved patient survival rates is crucial.

There is also a need for improved cancer treatments with fewer side effects, to be used as either monotherapies (i.e., stand-alone treatments) or co-treatments alongside other chemotherapeutics. Since the 1970s, the primary treatment for patients with HR-deficient cancer cells has been a combination of surgical cytoreduction and chemotherapy agents (e.g., platinum-based cisplatin and the better-tolerated carboplatin, or taxanes such as paclitaxel and docetaxel)[81,87–92]. Platinum-based chemotherapies interfere with cell replication by crosslinking purine bases, resulting in cell-cycle arrest and apoptosis or necrosis[92]. Taxane chemotherapies primarily function by blocking microtubule depolymerization, resulting in growth-cycle arrest[92,93]. Although not fully understood, taxanes are thought to be antitumorigenic because they arrest actively dividing cells during metaphase[92,93]. Traditional chemotherapeutics can have severe side effects, and outcomes have not improved since the early 2000s[81,92,94]. Fortunately, drugs capable of inhibiting PARP-1 are effective treatments against many types of cancer[1–4,12,13,24,95]. These PARPi tend to be more toxic to HR-deficient cancer cells than to non-cancerous cells and have been shown to work well on their own or in combination with other chemotherapeuitics[1–4,12,13,24,95].

1.1.3.2 HR-Deficient Cells and PARPi Sensitivity

1.1.3.2.1 HR-Deficient Cells and PARPi Sensitivity: Synthetic Lethality

The HR pathway has a synthetically lethal relationship with PARP-1[24,43,95], so HR-deficient cells tend to be hypersensitive to PARP-1 inhibition[28,96]. Synthetic lethality is the dynamic between two genes or pathways wherein a cell can survive when one or either component is deficient, but not when both are deficient[30]. It arises when there are functional redundancies in vital cellular processes, such that one mechanism can compensate for the other's defects[97]. DNA-damage repair is one such cellular process that is handled by multiple pathways with overlapping roles. For instance, both NHEJ and HR are responsible for repairing DSB[24,25,29]. Repair of DSB is particularly important in actively replicating cells such as tumor cells because an unrepaired break can result in daughter cells that do not receive required genes[11]. Despite losing an important DSB repair pathway, HR-deficient cancer cells remain viable by exploiting the complementary functions of non-HR repair pathways[30]. They are far more reliant on non-HR DNA repair pathways such as BER and NHEJ than are normal cells[30].

A synthetic-lethal pharmaceutical approach relies on the premise that by inhibiting the "right" protein target from the non-HR DNA repair pathway, one can find an effective drug treatment that is non-lethal to HR-proficient cells but lethal to HR-deficient cells[30,43,95]. Since the late 1980s, PARP-1 has been one such target of interest[95]. PARP-1 inhibition is the first successful example of a synthetic-lethal approach in pharmaceutical treatment[43].

PARP-1, which can recruit BER[5] and alt-NHEJ[25], is often upregulated in PARPi-sensitive HR-deficient cancer cells[28,96]. In contrast, wildtype (HR-proficient) cells are PARPi insensitive due to their ability to repair DNA damage with high-fidelity HR[24,98]. A similar effect has been observed in BER-deficient cells treated with PARPi[5]. PARP-1 plays well-documented roles in multiple DNA repair pathways ("Chapter 1.1.1: DNA Damage Repair in Humans"). For these reasons, PARP-1 inhibition is an excellent strategy for treating multiple forms of cancer[1–4].

1.1.3.2.2 HR-Deficient Cells and PARPi Sensitivity: Catalytic Inhibition and Trapping

Although all current FDA-approved PARPi bind to the catalytic pocket and act as competitive inhibitors of NAD+[73], the synthetic lethality of PARPi in HR-deficient cells cannot be explained solely by competitive inhibition[77]. For example, cellular sensitivity to PARPi decreases when PARP-1 expression is reduced via siRNA knockdown[77]. PARPi toxicity in HR-deficient cells appears to result from both PARylation inhibition and PARP-1 trapping on the DNA[73,99,100]. As discussed in "Chapter 1.1.2.6: DNA-Unbinding is Regulated by Interdomain Interactions," trapping PARP-1 on DNA can cause stalling and/or collapse replication forks, resulting in cell-cycle arrest or apoptosis[5].

1.1.3.3 The Four FDA-Approved PARP-1 Inhibitors (PARPi)

Currently there are four FDA-approved PARPi (olaparib, rucaparib, niraparib, and talazoparib) for treating HR-deficient cancers[1–4,12,13,24,95] (Figure 8), and several more are in varying stages of clinical trials[1–4]. Olaparib was the first FDA-approved PARPi[2] and has been used

as both a monotherapy and co-treatment with additional chemotherapeutics[2,101,102]. Olaparib was first approved in 2014 for treating ovarian cancer with *BRCA1/2* mutations[2], but it is now approved to treat certain HR-defective types of breast[103], prostate[104], and pancreatic[105] cancers as well. Although some PARPi have been approved as standalone cancer treatments, they are commonly paired with other treatments such as DNA-damaging chemotherapy or radiation[12,13,95].

Many PARPi share structural similarities with nicotinamide[95] (Figure 8) and form similar protein-ligand interactions when bound to PARP-1[17–19,106–112]. For instance, many PARPi, including the NAD+ analog BAD and the PARPi niraparib (Figure 3 p.15), form π-π stacking interactions with PARP-1's Y907[17–19,106–112]. These shared features raise concerns about PARPi resistance and the future of orthosteric (i.e., non-allosteric) PARP-1 inhibition. These concerns are revisited in "Chapter 3.0: AutoGrow4: Application to Poly (ADP-ribose) Polymerase-1 (PARP-1)."

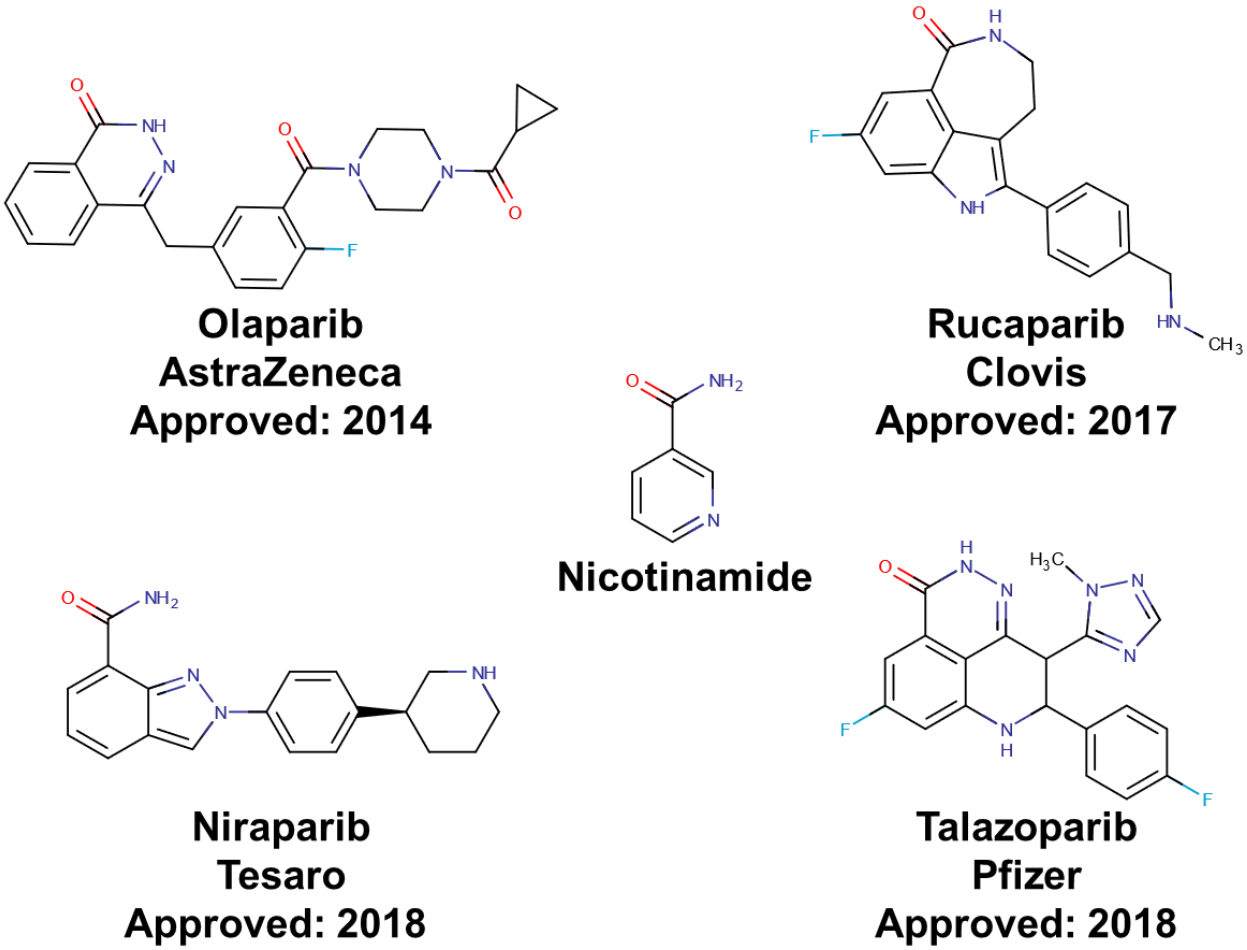


Figure 8. The four FDA approved PARPi.

The chemical structure of each FDA-approved PARPi is shown with the generic name, the company that developed it, and the year the FDA approved its use. Nicotinamide, a moiety of the endogenous ligand NAD+, is shown in the center to illustrate the chemical similarities between the PARPi and the endogenous ligand.

1.1.3.4 PARPi Resistance Mechanisms

PARP-1 is implicated in multiple complex biological pathways, which means there are also many possible resistance mechanisms to the four FDA-approved PARPi. In this discussion I will focus on resistance mechanisms known to affect HR-defective cells, separated into five categories: (1) HR reversion/increased HR capacity; (2) altered NHEJ capacity; (3) corrected replication forks; (4) decreased intracellular PARPi; and (5) modified PARP-1 expression, activity, allosteric regulation, PARylation, and/or PARPi binding.

1.1.3.4.1 PARPi Resistance Mechanisms: HR Reversion and Increased HR Capacity

Many proposed and clinically observed PARPi resistance mechanisms involve increased HR capacity[30,113]. HR-deficient cells tend to be more sensitive to PARPi than HR-proficient cells, largely because they rely more on non-HR DNA repair[30,113]. Consequently, partial or complete restoration of HR capacity results in PARPi resistance[30,113]. This section will discuss three mechanisms that increase HR capacity: (1) restoring HR-proficiency through secondary reversion mutation; (2) increasing HR protein availability; and (3) altering regulators of HR repair.

Because an HR defect can result in PARPi sensitivity, secondary mutations that restore HR proficiency reduce PARPi sensitivity[113]. HR-deficiencies are frequently caused by the disruption of the *BRCA1/2* genes[80,81], with the most common disruptions being single-nucleotide mutations, short insertions, and deletions that result in frameshifts[113]. Secondary reversion mutations that restore BRCA functionality have been clinically observed to desensitize previously BRCA-

deficient cells to PARPi[30,113,114]. The restoration of these genes speaks to the selective pressure that PARPi exhibit on tumors[113]. Unfortunately for patients and physicians, secondary *BRCA1/2* restoration mutations are not only associated with PARPi desensitization, but also with resistance to platinum-based chemotherapies[115]. Treatment with taxanes may remain an option in such circumstances because BRCAness (i.e., the phenotype of having deregulated HR, typically due to dysfunctional BRCA1/2 and/or BCRA-associating proteins[113,116]) tends to negatively impact treatment with taxanes[117].

Increasing the availability of HR proteins such as BRCA1/2 can also desensitize cells to PARPi[30,118–120]. For instance, a decrease in miR182, an microRNA that negatively regulates BRCA1 expression, results in higher levels of BRCA1 expression and consequently PARPi desensitization[30,119,120]. Additionally, PARP-1 negatively regulates BRCA2 expression by binding to the silencer-binding region of the *brca2*-promoter[30,118]. Inactivation and trapping of PARP-1 by PARPi could lead to increased expression of BRCA2, resulting in decreased PARPi sensitivity[30,118]. Overexpression of RAD51, a recombinase and important HR protein, is also a mechanism for PARPi resistance[30,121,122]. RAD51 is frequently overexpressed in BRCA1-deficient tumors and is associated with partially restored HR repair[30,121–124]. RAD51 overexpression has been observed in rucaparib-resistant high-grade ovarian carcinoma and colon carcinoma cells that were resistant to cotreatments of PARPi and temozolomide, an alkylating chemotherapeutic[30,121,122]. There are multiple ways RAD51 levels can be upregulated, including the downregulation miR-96[30,125] and Aurura-1[30,126], which lower RAD51 expression[30,125,126], and the upregulation of tumor suppressor phosphatase and tensin homolog (PTEN)[12,30,127,128].

Lastly, HR capacity can be increased by altering the proteins that regulate HR repair, such as ataxia telangiectasia mutated (ATM) and 53BP1[30,113,123,129,130]. The kinase ATM, upon sensing DNA damage, triggers a kinase cascade that results in HR repair[123,129]; 53BP1, along with BRCA1, modulates the use of HR and NHEJ repair. The loss of BRCA1 results in decreased HR repair and increased NHEJ repair, whereas loss of 53BP1 results in increased HR repair, particularly ATM-dependent HR repair, and decreased NHEJ repair[30,113,123,129]. Therefore, either an increase in ATM or a decrease in 53BP1 could provide PARPi resistance[30,123,129]. ATM inhibition paired with PARPi may therefore circumvent some PARPi resistance mechanisms[130].

1.1.3.4.2 PARPi Resistance Mechanisms: Altered NHEJ Capacity

Changes to an HR-defective cell's NHEJ capacity can also result in PARPi resistance[24,29,30,113]. As previously described, NHEJ repair is regulated by 53BP1 and BRCA1, and alterations to these proteins can impact NHEJ and HR capacities[30,113,123,129]. Additionally, modifications to the expression or function of NHEJ proteins such as DNA-PK and Ku proteins can alter NHEJ capacity and consequently the cell's sensitivity to PARPi[24,30,113].

Both the downregulation and upregulation of NHEJ drive chromosomal instability and may result in PARPi resistance[29,30]. Downregulation of c-NHEJ proteins diminish PARPi toxicity in HR-deficient cells[29,30]. However, NHEJ is error-prone and increases the opportunity for mutations[29,30]. The increased mutation rate in HR-deficient NHEJ-upregulated cells may further promote the acquisition of PARPi-resistance-causing mutations[119,131].

1.1.3.4.3 PARPi Resistance Mechanisms: Corrected Replication Forks

PARPi toxicity to HR-defective cells results in part from the tendency of DNA-bound PARP-1 to stall replication forks[5]. BRCA1/2 proteins are involved in DNA-strand protection at stalled forks[14], so BRCA-deficient cells are sensitive to fork collapse and nascent-strand shortening[113,132,133]. Consequently, proteins that stabilize the replication fork can contribute to PARPi resistance[133]. For example, nucleosome remodeling factor CHD4 was found to be expressed at lower levels in chemotherapy-resistant *BRCA2*-mutant cancer cells, even though those cells remain deficient in RAD51-dependent HR repair[133]. Reduction of CHD4 in BRCA2-deficient cells resulted in protected nascent replication tracts and PARPi resistance[133,134].

1.1.3.4.4 PARPi Resistance Mechanisms: Decreased Intracellular PARPi

Lowering the concentration of intracellular PARPi can also lessen the effects of PARPi treatment. Intracellular PARPi concentrations are regulated by the ATP-binding cassette (ABC) drug efflux transporters p-glycoproteins[30,113,135–138]. The impact of intracellular PARPi on PARPi sensitivity is suggested by the findings that reduced p-glycoprotein levels increased PARPi sensitivity in BRCA1-deficient mouse models[135,136]. Additionally, p-glycoprotein inhibition re-sensitized BRCA-deficient PARPi-resistant cells to PARPi[135,137]. Further evaluation of this resistance mechanism is necessary because it has mostly been explored only in animal and cell models[30,113,135–138].

1.1.3.4.5 PARPi Resistance Mechanisms: Altered PARP-1 Capacity

PARPi cytotoxicity in HR-deficient cells is caused by PARP-1 inhibition at the catalytic active site and trapping of PARP-1 on DNA[73,99,100]. Alterations to either of these processes, as well as to the function and levels of PARP-1 expression, change a cell's sensitivity to PARPi.

PARP-1 expression levels and catalytic activity correlate with PARPi sensitivity. For example, PARPi- and temozolomide-resistant clones of colorectal-carcinoma HCT116 cells have markedly low levels of PARP-1 expression[30,121]. Cancer cells with decreased PARylation activity but normal PARP-1 expression levels also tend to be more resistant to PARPi[30,139]. Additionally, HR-deficient cells with higher levels of PARylation tend to be more sensitive to PARPi[30,140].

Alterations to the PARPi binding site can also confer PARPi resistance. The proto-oncogene receptor tyrosine kinase c-Met, which is commonly overexpressed in many forms of cancer, alters the PARP-1 catalytic site by phosphorylating Y907 and thereby reducing PARPi binding[141,142]. As described in "Section 1.1.2.3: PARP-1 Catabolizes NAD+ to Perform ADP-Ribosylation," Y907 forms π-π stacking interactions with NAD+ and with PARPi[142]. Phosphorylated Y907 (pY907) has increased catalytic activity and weakened binding affinity to multiple PARPi[142]. Because c-Met can reduce PARPi binding, upregulation of c-Met can therefore confer PARPi resistance[142]. pY907 has been confirmed to cause resistance to veliparib, olaparib, and rucaparib[142]. Because all four of the FDA-approved PARPi bind to the same site[17], this resistance mechanism likely confers resistance to most competitive inhibitors of the PARP-1 catalytic site. This PARPi resistance mechanism is explored further in "Chapter 3.0: AutoGrow4: Application to Poly (ADP-ribose) Polymerase-1 (PARP-1)."

Mutations that disrupt the allosteric communications mediated by interdomain interfaces which control DNA trapping and PARP-1 catalytic activity (discussed in "Chapter 1.1.2.5: PARP-1 Catalytic Activation is Modulated by Interdomain Interactions" and "Chapter 1.1.2.6: DNA-Unbinding is Regulated by Interdomain Interactions") can provide additional routes for altered PARPi resistance. For instance, a R591C mutation at the Zn1-WGR-HD interface was found in an talazoparib-resistant ovarian tumor and significantly reduces PARP-1 trapping[73,78]. Many more human-engineered mutations also alter PARP-1 activity, PARPi inhibition, and/or PARPi trapping ("Chapter 1.1.2.5: PARP-1 Catalytic Activation is Modulated by Interdomain Interactions" and "Chapter 1.1.2.6: DNA-Unbinding is Regulated by Interdomain Interactions.") Although to date many of these have only been documented in lab-generated settings, these mutations still pose potential PARPi resistance mechanisms.

Lastly, the loss of PAR glycohydrase (PARG) activity, which degrades PAR chains, is associated with PARPi resistance[113,143]. This resistance mechanism, discovered in a screen of BRCA2-deficient mouse cells, results in increased PARylation, thereby partially rescuing PARP-1-dependent DNA-damage singling[113143]. Knockdown of PARG by shRNAs in human *BRCA1*-mutated cancer cells (SUM149PT) and BRCA2-deficient cancer cells (DLD-1) caused olaparib resistance[143]. The same study also found low-level PARG expression among a subset of human triple-negative breast tumors and serous ovarian carcinomas[143]. Co-treatment of a PARG inhibitor and a PARPi may circumvent this resistance mechanism.

1.1.4 The Current State of the Field: PARP-1 and Pharmaceutical Intervention

In this section, I will summarize the current state of PARPi pharmacology, including several limitations of current PARPi. These limitations motivated the research described in this dissertation. Over the past decade, four PARPi have been FDA approved[1–4]. Many studies have provided insight into PARP-1 dynamics and the underlying mechanisms of PARP-1 inhibition[16,46,48,55,70,73–75,144], and have identified common resistance mechanisms[12,30,113,115,135,142].

As discussed in "Chapter 1.1.3.4: PARPi Resistance Mechanism," all FDA-approved PARPi bind to PARP-1's same catalytic pocket in very similar manners. Consequently, many PARPi resistance mechanisms likely confer resistance to all current PARPi. There is thus a need for new PARPi that are not susceptible to the same resistance mechanisms.

Current PARPi function as both orthosteric inhibitors of catalytic function and allosteric inhibitors that trap PARP-1 on DNA. Ideal novel PARPi would also inhibit one or more PARP-1 functions. I propose that the next generation of PARPi should target both the catalytic site and the important interdomain interfaces that limit PARP-1 release from DNA and/or prevent DNA-dependent catalytic activation. Developing an array of PARPi that can target multiple PARP-1 sites and functions could circumvent many forms of PARPi resistance. Of course, expanding the PARPi options will not address certain PARPi resistance mechanisms such as secondary *BCRA1/2* mutations that resurrect HR function, or resistance mechanisms that limit PARP-1 expression. However, new PARPi that do not rely on the same binding interactions as current inhibitors may overcome resistance mechanisms caused by PARP-1 post-translation modifications or mutations.

This dissertation details my efforts to predict novel PARPi using computational techniques, particularly techniques that do not rely on preexisting small-molecule libraries. These techniques allowed me to survey many more compounds than would have been feasible *ex silico*. The following section, "Chapter 1.2: Computational Methodology," will detail the computational and cheminformatic background and rationale for the design of this dissertation project.

1.2 Computational Methodology

This dissertation describes the computational techniques I used to design and assess novel candidate PARP-1 inhibitors. One technique central to this dissertation is AutoGrow4, a genetic algorithm (GA) for *de novo* drug design. I used AutoGrow4 to identify candidate PARP-1 inhibitors, but it can be applied to virtually any protein target.

This section will review the computational concepts relevant to AutoGrow4. Since AutoGrow4 is a GA for *de novo* drug design, I will discuss optimization using GA and relevant strategies. I will also review the field of computer-aided drug design (CADD), compare *de novo* and virtual screening (VS) techniques, and discuss chemical drug-likeness filters. AutoGrow4 assesses compound fitness primarily by docking small molecules into a protein target, so I will also discuss protein-ligand docking. Lastly, I will compare AutoGrow4 to other *de novo* CADD programs and will discuss the limitations that motivated me to develop AutoGrow4.

1.2.1 Genetic Algorithms (GA)

I will begin with a thorough overview of the theory behind and rationale for using GA. This section will first discuss how GAs work conceptually, including the underlying concept of "search space." I will then explain how GAs problem solve (i.e., how they produce new solutions for a given problem), how GAs assess the quality of those solutions, and how GAs select solutions to be further tested. I will end by discussing the technique's limitations and the conditions most appropriate for implementing a GA.

1.2.1.1 Search Space and the Fundamentals of Genetic Algorithms (GA)

GA are stochastic problem-solving strategies that navigate "search space" by iteratively evolving populations of new solutions[145,146]. "Search space" in this context means the set of all possible solutions to a given problem[146]. In multimodal problems, there are often multiple "good" solutions with varying levels of "goodness" or "fitness." There may or may not be a true global optimum, but there are multiple local optima[147]. Indeed, the number of all potential solutions may be enormous[146,147]. For instance, the search space of all synthesizable drug-like molecules includes 10^{20} to 10^{23} possible solutions[148–150]. In such cases, testing all possible solutions is impractical or impossible.

GA excel in these scenarios because they balance computational costs with thoroughness of search and so can generate multiple "fit" answers to a problem[145,146]. They comprise a class of evolutionary algorithms that attempt to mimic Darwinian evolution by successively evolving

populations of solutions (referred to as generations)[145,146,151]. Traits of the most "fit" parent solutions are combined or altered to generate new child solutions[145–147,151]. These new solutions are then evaluated to determine their "fitness."[145–147,151] Selective pressure is applied by selecting fit solutions to parent the next generation[145–147,151]. Randomness is incorporated in several ways, such as through seed-population and trait selection[145–147,151]. This randomness means that the predicted solutions may differ each time the algorithm is run, which is why GA are considered stochastic.

1.2.1.2 Populating a Generation of Solutions

GAs create new generations of solutions by either advancing fit solutions from the previous generation or evolving new solutions from those of the previous generation[145–147,151]. AutoGrow4 "solutions" are small molecules that dock into a protein pocket. Operators are the functions that alter, merge, or advance solutions from one generation to the next[145–147,151]. The three most common operators are crossover, mutation, and elitism[145,146,151]. Crossovers merge traits of two parent solutions[145,146], mutations make small alterations to a single parent compound[145,146], and elitism advances a solution from a previous generation without alteration[145,146]. I describe these operators in the context of AutoGrow4 in "Section 2.2.2: Operators: Population Generation via Crossover, Mutation, and Elitism" (Figure 9, p.74).

1.2.1.3 Fitness

All GA require a metric with which to judge the quality of solutions[145–147,151]. This is referred to as a fitness metric and is determined by a fitness function[145–147,151]. In the case of AutoGrow4, the primary fitness metric is the predicted binding affinity that is calculated by a docking program. Offspring are created from well-scoring solutions to create a new generation of solutions[145–147,151].

The dynamics of how populations of solutions evolve can be viewed in terms of adaptive landscapes[152–154]. This concept was introduced by Sewall Wright in the 1930s as a way to describe the relationship between genotype and fitness as related to natural selection[152,153]. In his 1982 article, "The Shifting Balance Theory and Macroevolution,"[154] Wright describes population shifts from one adaptive peak (i.e., optima) to another as a three-step process: (1) "stochastic variability" (i.e., random genetic alteration or mutations) causes subpopulations to explore the adaptive landscape, which results in the genetic drift of a portion of the population[154]; (2) natural selection pressures subpopulations that exist between adaptive peaks to migrate through genotypic space towards a peak[154]; and (3) populations at the most advantageous peak(s) proliferate and mate with other subpopulations to confer advantageous traits to other portions of the population[154]. Wright's view of adaptive peaks separated by barriers transcends evolutionary biology and is a cornerstone of evolutionary algorithms that attempt to mimic Darwinian evolution[145,146,151,152].

In GA, solutions that have reached advantageous peaks (i.e., achieved high scores) proliferate at higher rates than other solutions and so can become overrepresented, resulting in a homogenized population[151]. Taken to an extreme, this loss of diversity within the solution pool,

referred to as convergence, is a sign that a local optima has been reached and that new solutions are not likely to be created from the current population[151]. Delaying convergence allows for more solution space to be explored, which may lead to the discovery of even better solutions[151]. To delay homogenization, AutoGrow4 provides a secondary fitness function that selects for chemically unique compounds. The details of both the primary and secondary fitness functions are discussed in "Section 2.2.7: Assessing Fitness." AutoGrow4 also provides multiple options for reassessing binding properties, which are detailed in "Section 2.2.7: Assessing Fitness."

1.2.1.4 Ranking and Selection Approaches

A core feature of GA is the application of a selective pressure to guide the fitness of successive generations, traditionally by selecting solutions that are able to spawn new solutions and/or solutions that advance via elitism[151]. Broadly speaking, three of the most commonly used strategies for selection in GA are Ranking selector, Roulette selector, and Tournament selector[151].

A Ranking selector grades solutions from best to worst and selects the best solutions. This is an effective way to find local optima, but it often yields inbred populations comprised of highly similar compounds[151]. In extreme cases, population homogenization can cause convergence wherein the algorithm perpetually recreates similar compounds without substantial fitness improvement.

A Roulette selector assigns each solution to an area on a metaphoric roulette wheel, with the size of each area weighted by fitness, and so incorporates randomness into each generation. This ideally minimizes the GA's chance of becoming trapped in local optima[151]. However, this

method also provides an opportunity for all potential solutions to advance, including the most unfit ones[151].

Lastly, a Tournament selector randomly chooses a subpopulation of candidates from a generation and selects the fittest from that subpopulation[151], thereby incorporating more randomness than a Ranking selector while mitigating the risk of selecting unfit solutions[151].

A user's choice of strategy depends on the goals and resources of the project. For instance, someone aiming to find a good local optimum with little computational overhead would benefit from using a deterministic approach with a Ranking selector. Alternatively, if the user's goal is to explore a wider range of search space and test the greatest possible diversity of solutions, a stochastic selection strategy that provides randomness is recommended. It is incumbent on users to choose the selection strategy most applicable to their studies.

1.2.1.5 Limitations of Genetic Algorithms (GA)

GA are most applicable when a search space is so large that it would be impractical to brute-force test all possible solutions, and finding the global optima is not essential[145]. The set of all possible drug-like small molecules is one such search space; it includes 10^{20} to 10^{23} compounds[148–150], an untestable number of compounds by any means. Because GA operate with limited prior knowledge of solution space, they are more efficient than brute-force testing in large search spaces[145]. Conversely, GA are often less efficient than more direct methods in small search spaces, or when prior knowledge can be used to intelligently limit the search space[145].

While some selection approaches can postpone convergence, GA still suffer from local optima "trapping,"[145,151] which leads to population homogenization as a few solutions begin to dominate[145] (discussed in "Section 1.2.1.3: Fitness"). This concern is further discussed in "Section 3.3.1.3: A Caution Regarding Homogeneity and Convergence"

The stochastic nature of GA means that repeated independent runs can produce different results[145]. This is conducive to a wide search of solution space, but it also raises concerns about reproducibility and parameter optimization. A user may have difficulty distinguishing whether random chance or a specific parameter is primarily responsible for a given result.

1.2.2 Computer-Aided Drug Design (CADD)

AutoGrow is a GA program for computer-aided drug design (CADD)[155,156]. CADD programs are now key components in many pharmaceutical pipelines. CADD leverages computing power and automation to perform *in silico* experiments at scales that would be too costly to perform *ex silico*[148–150,155,156]. Although an exhaustive search of chemistry space remains impractical, thoughtful use of CADD can narrow candidates to a testable subset. Though it cannot replace the intuition of trained chemists and biologists, CADD has been successfully applied to drug discovery, lead optimization, and compound synthesis[155–158].

This section will begin by defining several categories of CADD. I will then provide a comparison of *de novo* CADD programs. Lastly, I will describe useful properties and filters for intelligently designing better drug candidates.

1.2.2.1 Categories of CADD Techniques

The two most prominent CADD techniques are structure-based drug design (SBDD) and ligand-based drug design (LBDD)[157,159]. SBDD finds ligands that bind to the pocket of a known biological structure, and LBDD discovers new ligands based on the physiochemical properties of known ligands[157,159]. SBDD programs can be further separated into two main techniques: virtual screening (VS), which evaluates compounds from an *in silico* database, and *de novo* design, which generates novel compounds *in silico*[155–157,160].

1.2.2.2 *De novo* and VS CADD

De novo CADD techniques include lead generation and lead optimization. Lead generation identifies novel predicted ligands, and lead optimization refines existing candidate compounds to improve specific properties such as binding affinity or solubility[95,155,161].

Neither *de novo* CADD nor VS can search all possible small molecules; however, whereas VS is limited to searching pre-enumerated databases of compounds[162], *de novo* CADD often relies on stochastic algorithms such as GA and therefore can explore a wider range of possible solutions[155–157,160]. Additionally, VS databases may contain compounds that are subject to intellectual property restrictions[162]. Compounds identified through *de novo* techniques are more likely to avoid such issues because these techniques search a less restricted space.

Of course, *de novo* CADD also has its limitations. Because VS libraries are often comprised of readily available compounds, *ex silico* testing of candidates can begin with minimal

delay. In contrast, lead compounds discovered by *de novo* techniques require synthesis before *ex silico* testing can begin.

1.2.3 Chemical Properties for Selecting Drug-Like Compounds

In this section, I discuss important properties that are associated with drug-like compounds (i.e., compounds that are similar to clinically successful drugs[163]). Analyses of existing drugs have identified many properties that are typical of a "good drug." Cheminformatic programs can reproducibly predict physiochemical properties[164–166], which can be used to select compounds with desirable features[6,156,162,167]. Maximizing the drug-likeness among candidate drugs, whether by lead optimization to improve favorable traits or filtering to remove compounds with unfavorable traits, can prevent the needless testing of non-drug-like candidates and improve the chance of CADD success[6,156,168–174].

1.2.3.1 Absorption, Distribution, Metabolism, Excretion, and Pharmacokinetics (ADME-PK)

To predict future drugs, one must consider not only measurements such as binding affinity and IC_{50}, but also absorption, distribution, metabolism, excretion, and pharmacokinetics (ADME-PK)[175]. For instance, even the most potent compound cannot serve a drug if it cannot be delivered to its target location in the body[176]. An inhibitor intended to treat brain cancer will not be effective if it cannot enter the blood stream and cross the blood-brain barrier[169]. Additionally, the liver,

spleen, and kidney may modify, degrade, or filter drugs[177]. When designing a drug, one must carefully consider drug metabolism, drug excretion, and drug toxicity[175].

AutoGrow4 provides the option to filter its generated compounds by ADME-PK properties. The filters included in AutoGrow4 (version 4.0.0) are described in the following section.

1.2.3.2 Chemical Drug-Likeness Filters

Chemistry space, the set of all possible compounds[178], is so vast that even computational methods cannot explore it entirely[148–150,178]. Because much of chemistry space includes compounds that are not likely to become clinical (i.e., viable) drugs due to their physiochemical properties[148–150,178–180], CADD aims to prioritize viable candidate compounds for further experimental evaluation. One way that CADD does this is by filtering each candidate according to its drug-likeness, or similarity to known drugs[163].

Properties associated with drug-likeness include small-molecule solubility, biological reactivity, and similarity to common metabolites[163,168–174]. Filters can be used to select compounds for/against such predicted physical and chemical properties[163,168–174]. When applied before more costly steps in drug development, filters can save time and money by removing leads with problematic properties. I here discuss the nine chemical filters that are predefined in AutoGrow4 (Table 1): Lipinski, Lipinski*, Ghose, Ghose*, VandeWaterbeemd, Mozziconacci, BRENK, NIH, and PAINS.

AutoGrow's Lipinski filters (Lipinski, Lipinski*) are based on Lipinski's rule of fives for drug-likeness, which was published in 1997 and has become a standard in cheminformatics[163]. Lipinski's rule of fives states that for a molecule to be orally bioavailable and drug-like, its predicted octanol–water partition coefficient (logP), hydrogen-bond donor count, hydrogen-bond acceptor count, and molecular weight (MW) should fall within a specific range (Table 1)[163]. However, as Lipinski discussed, there are outliers that violate these chemical standards but are still successful clinical drugs[163]. Additionally, Lipinski recognized that his filter does not allow for one ideal property to salvage a bad property[163]. Both of these concerns are illustrated by the antibiotic azithromycin[163]. Azithromycin has excellent aqueous solubility and oral activity, but a high MW and poor permeability[163]. According to Lipinski's rule, the latter factors should disqualify it from drug-likeness, despite its actual efficacy[163]. Lipinski's original implementation partially addressed these concerns by allowing one violation before rejecting the compound[163]. AutoGrow3 and AutoGrow4 therefore provide two options for Lipinski filtering: "Lipinski," which allows a ligand to fail a single property and still pass, and "Lipinski*," which requires a molecule to pass all property restrictions (Table 1).

The Ghose filter, another established standard in cheminformatics, applies a similar set of physiochemical property rules[163,168]. Specifically, it requires that a compound's logP, MW, and molar refractivity conform to a restricted range (see Table 1)[168]. AutoGrow4 provides two Ghose filter options: "Ghose," which follows all of the rules of Ghose, and "Ghose*," a modified version that relaxes the upper limit of MW from 480 Da to 500 Da (Table 1). The modified Ghose* filter was designed to match the Ghose filter settings[156] used in AutoGrow3 (version 3.1.3).

AutoGrow4's five new predefined filters take advantage of recent studies that have found new properties that can predict drug-likeness, bioactivity, and permeability. The VandeWaterbeemd filter uses MW and polar surface area[169] to screen for drugs capable of crossing the blood-brain barrier (Table 1). The Mozziconacci filter is a drug-likeness filter that tests molecules for their number of rotatable bonds, rings, oxygen atoms, and halogen atoms[170] (Table 1).

Not all chemical filters test physiochemical properties. Structure/substructure-based approaches identify compounds with desirable or undesirable functional groups. RDKit[165], a Python library for cheminformatics, provides three such substructure filters, which have been incorporated into AutoGrow4: BRENK, NIH, and PAINS (Table 1). The BRENK filter is a lead-likeness filter that screens against undesirable functional groups based on 105 fragments identified as common false-positive drugs[171]. The NIH filter screens against compounds containing a set of undesirable functional groups[172,173]. Finally, the PAINS filter screens against compounds containing a set of substructures that are known to react with many biological targets and to yield false positives and off-target effects[174].

Table 1. Chemical filters.

Nine molecular filters that are predefined in AutoGrow4. Lipinski is the traditional interpretation of Lipinski's filter, which allows for one violation. Lipinski* is a stricter version of Lipinski's filter that does not allow for any violations. Ghose* is a more lenient version of Ghose that has a relaxed molecular-weight range (up to 500 Da). HD, hydrogen-bond donor; HA, hydrogen-bond acceptor; MW, molecular weight (Da); MR, molar refractivity (m^3mol^{-1}); Atoms, atom count; RotB, rotatable bonds; R, rings; N, O, and X, nitrogen, oxygen, and halogen atoms, respectively; PSA, polar surface area (A^2); and Sub, substructure searching. This table is taken from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. It is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License which "allows unrestricted use, distribution, and reproduction in any medium."[6]

Name	logP	HD; HA	MW	MR	Atoms	RotB	R	N; O; X	PSA	Sub
Lipinski [163]	≤5.0	≤5; ≤10	≤500							
Lipinski* [156]	≤5.0	≤5; ≤10	≤500							
Ghose [168]	-0.4-5.6		160-480	40-130	20-70					
Ghose* [156]	-0.4-5.6		160-500	40-130	20-70					
VandeWaterbeemd [169]			<450						<90	
Mozziconacci [170]						≤15	≤6	≥1; ≥1; ≤7		
BRENK [171]										+
NIH [172,173]										+
PAINS [174]										+

1.2.4 Protein-Ligand Interactions and Protein-Ligand Docking

Protein-ligand docking is a popular computational technique for predicting a small molecule's binding geometry ("pose", i.e., position/orientation) and binding strength ("score") relative to a protein target[155,156,181–185]. Docking is particularly popular for early-stage drug design because it provides information about protein-ligand interactions, and a single docking run requires minimal computational resources[155,156,181–185]. I list several prominent docking programs in Table 2.

Because AutoGrow4 relies on protein-ligand docking to assess the compound fitness, this section begins with a discussion of protein-ligand docking, including theoretical models of small-molecules/protein binding. I then detail two core processes common to many protein-ligand docking programs: conformational sampling and scoring the protein-ligand binding pose (i.e., position/orientation).

Table 2. Features of several prominent docking programs.

FOSS stands for "free and open-source software"; FAU stands for "free for academic use"; "Flexible-rigid" indicates that the docking software treats the ligand as flexible and the receptor as rigid; "Flexible-flexible" indicates that the docking software treats the ligand as flexible and treats the receptor as flexible to some degree. Conformational-sampling approaches are described in "Section 1.2.4.2: Conformational Sampling," and the scoring functions are described in "Section 1.2.4.3: Docking Scoring Functions."

Software	Availability	Flexibility (Ligand-Protein)	Conformational Sampling	Scoring Function
AutoDock Vina[185]	FOSS[185]	Flexible-rigid[185,186]	Stochastic[186]	Semi-empirical[186]
DOCK6[187]	FOSS[187]	Flexible-flexible[186]	Systematic[161,186]	Force-field[186]
Glide[188,189]	Commercial	Flexible-flexible[186]	Systematic[161,186,190]	Semi-empirical[186]/Empirical[161,190]
GOLD[191,192]	Commercial	Flexible-flexible[186]	Stochastic[161,186]	Semi-empirical[186]/Force-field[161]
PLANTS[193–195]	FAU[196]	Flexible-flexible[186,193]	Stochastic[161,190,196]	Empirical[161,195]
FlexX[197]	Commercial	Flexible-rigid[186]	Systematic[161,186,190]	Semi-empirical[186]

1.2.4.1 Models of Protein-Ligand Binding

Models of protein-ligand binding directly inform rational drug design and computational chemistry. Our understanding of protein dynamics and protein-ligand interactions has evolved since Emil Fischer proposed his "lock-and-key" model of protein-ligand binding more than 125 years ago. Fischer's model postulates that ligands fit into protein binding sites the way a key fits into a static lock[190,198]. His "lock-and-key" model was later replaced by the "induced fit" model, which states that a protein undergoes a conformational change as a result of binding to a ligand[190,199,200].

Recent theories challenge the induced fit model and treat proteins and ligands as dynamic entities that can assume an ensemble of conformations[190,199,201]. These conformations can be organized by their potential energies into an energy landscape, with wells representing energetically stable conformational states that are separated from inter-state conversion by energy barriers[190,199,202,203]. From this "energy landscape" view emerged the "conformational selection" paradigm, which suggests that among the many conformational states there are a select few that favor ligand binding and that, conversely, are stabilized by binding[190,199,204].

Now, evidence suggests that protein-ligand dynamics exist as a combination of induced fit and conformational selection[190,199]. Modern models such as Csermely's "extended conformational selection" model consider energy landscapes, induced fit, and conformational selection, embracing a dynamic back-and-forth between selection and adjustment[190,199].

1.2.4.2 Conformational Sampling

Docking programs sample ligand conformations within a protein pocket and score the quality of each pose[190], often by predicting protein-ligand binding affinity[185,205]. As discussed in "Chapter 1.2.3.2: Chemical Drug-Likeness Filters," there are many properties that make for an ideal drug candidate[155,156,181–185,163,168–174], but it is common practice in the early stages of lead discovery to select candidate compounds based on their predicted binding affinities[155,156,181–185]. Accurate predictions of protein-ligand binding save time by reducing the number of *ex silico* (validation) experiments needed. To estimate binding affinity, docking algorithms consider protein and ligand conformational states. The first docking algorithms were developed in the early 1980s and treated both protein and ligand as inflexible (rigid)[190,204]. These methods relied heavily on the "lock-and-key" theory of ligand binding, treating atoms as spheres and focusing primarily on steric interactions[190,204]. Given the computational resources of the 1980s, these simplifications were necessary. Treating any component of pose sampling as flexible increases the number of degrees of freedom, which increases computational cost[190,206].

Most modern docking programs treat the ligand as fully flexible[190], but it remains largely impractical to treat the entire protein as flexible (Table 2 p.55). Instead, many programs make approximations such as considering only partial receptor flexibility or limiting flexibility to sidechain motions (Table 2 p.55)[185,190,207,208]. Additionally, some modern techniques, such as ensemble docking, dock ligands into multiple protein conformations, which increases the computational cost[190,208,209].

Modern techniques for generating docked ligand poses can be grouped into three main strategies: systematic, simulation, and stochastic searches[190,206]. Systematic searches parameterize and test most or all combinations of each degree of freedom[190,206]. Simulation approaches utilize local optimization strategies that apply forces to advance the *in silico* protein/ligand system over time[190,206]. Molecular dynamics simulations and energy minimizations are the most common simulation approaches[190,206], though both often suffer from local minima trapping[190,206]. Lastly, stochastic methods for conformational sampling apply random small alterations to a system[161,186,190,196,206]. These tend to be fast, but they often lack convergence[190,206] and may miss a definitive best solution[190,206]. This limitation can be mitigated with repeated attempts or by adjusting the search's exhaustiveness (i.e., how comprehensive a pose-sampling search is)[190,206].

1.2.4.3 Docking Scoring Functions

As a docking program generates ligand poses, its scoring function must reliably assess each pose's binding strength, often by predicting its binding affinity[185,205]. The three main scoring approaches are force-field/physics-based, empirical, and knowledge-based[161,182,190,210] (Table 2 p.55). As a general rule, the more parameters a scoring function considers, the more accurate its calculated binding affinity but also the more costly the computation[161]. Docking scoring functions must balance speed and accuracy.

Force-field-based scoring functions predict binding energies by considering the bonded and nonbonded components of the system[161,190,210]. Bonded components include intramolecular terms such as torsions, bond stretching, and angle bending[161,190,210], whereas unbonded

components include intermolecular terms such as electrostatics and van der Waals forces[161,190,210]. An *ab initio* energy calculation is performed for every term (i.e., all bonded and non-bonded interactions)[161,206]. Force-field-based scoring functions tend to be more accurate than other types, but they also tend to be the most computationally expensive[161,182].

Like force-field-based scoring functions, empirical scoring functions sum various energetic terms like electrostatics and van der Waals forces[161,182,190,210], but they weigh each term by an optimized coefficient rather than calculating *ab initio* energy contributions[161,182,190,210]. These coefficients are determined by fitting an equation to a training data set containing 3D protein-ligand structures and experimentally measured binding-affinity data[161,190,210]. These weights are used to adjust each term in the equation[161,190,210]. These simplified calculations make empirical scoring faster but less accurate than force-field-based scoring[161,182,210].

Knowledge-based scoring functions assume that more commonly observed protein-ligand interactions correlate with favorable binding interactions[161,182,190,210]. These functions are trained on databases of protein-ligand interactions and assign energy components to each protein-ligand contact[161,182,190,210]. Once trained, they predict binding energetics by summing the energy components of the interactions associated with a given protein-ligand pose[161,182,190,210]. Knowledge-based scoring functions tend to have intermediate speed and accuracy compared to force-field-based and empirical scoring functions[161,182].

The distinction between these scoring functions is frequently blurred. Hybrid approaches such as semi-empirical scoring may use a force field equation with several weighted terms (Table 2 p.55)[190,210]. Machine-learning scoring functions, which are trained on data sets of 3D structures

and experimental binding data[182,190,210], as well as consensus scoring functions, which combine the scores of multiple scoring functions[182,190,206,210], are also becoming more popular.

1.2.5 Alternative Approaches for *de novo* CADD

There are many *de novo* drug design programs in addition to AutoGrow4. I here provide a summary of several recently published tools for *de novo* CADD (Table 3). This comparison will focus on their features and approaches. Experiments that compare AutoGrow4 with alternative *de novo* CADD techniques are provided in "Chapter 4.3.2: Comparison of AutoGrow4 Lead Optimization and other *De Novo* CADD Programs." This section will also detail the limitations of other techniques, which motivated me to develop AutoGrow4.

Table 3. Highlighting the features of several recently published *de novo* design programs. Each of these programs takes a different approach to *de novo* compound generation. FOSS stands for "free and open-source software," MPI stands for "message passing interface," and OS stands for "operating system." Table is modified from the AutoGrow4 manuscript, which was published in the Journal of Cheminformatics. It is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium."[6]

Program	FOSS	Docking Options	MPI Enabled	OS
AutoGrow4[6]	Yes	Vina/QVina2/Customizable	Yes	Linux/macOS/Windows (via docker)
AutoGrow3[156]	Yes	Vina	No	Linux/macOS/Windows (via docker)
MoleGear[211]	No	Autodock and Vina	Yes	Unspecified
GANDI[167]	Yes	DAIM/SEED/FFLD	Yes	Linux
***De novo* DOCK**[162]	Yes	DOCK	No	Linux/macOS
REINVENT[212]	Yes	N/A	Unspecified	Linux/macOS
LigDream[213]	Yes	N/A	Unspecified	Unspecified

1.2.5.1 Comparison of *de novo* CADD Software

AutoGrow4 is the fourth major release of the AutoGrow program series. Its predecessor and most recent previously published version, AutoGrow3, was published in 2013 and improved on the choice of docking software, the re-implementation of compound creation, and molecule scoring[156]. These changes improved AutoGrow's accuracy and efficiency, but docking software, cheminformatic libraries, and multithreading capabilities have since advanced[165,205,214]. AutoGrow4 takes advantage of these advancements, resulting in a faster, more stable, and more modularly designed program. Benchmark comparisons of AutoGrow4 and the most current release of AutoGrow3 (version 3.1.3) are provided in "Chapter 2.0: AutoGrow4: Implementation and Benchmarks". Here I detail other *de novo* CADD programs. I will perform comparisons with AutoGrow4 with several of these programs in "Chapter 4.0: Comparison of CADD Techniques".

MoleGear is another recently published program that performs *de novo* drug design using an evolutionary algorithm[211]. It provides many user options such as a graphical user interface and two docking-software options, AutoDock[215] and Vina[185]. Unfortunately, MoleGear does not yet appear to be publicly available, and the program is closed source, meaning that the underlying code that runs the algorithm is not publicly accessible[211].

On the other hand, *de novo* DOCK is an open-source algorithm that is free for academic use[162]. *De novo* DOCK is integrated into the docking program DOCK6[162,187] and uses an iterative fragment-growth method[162] based on DOCK6's anchor-and-grow search algorithm[162,187]. This approach identifies the core components of a given compound, referred to as "anchors," and expands each anchor layer-by-layer with molecular fragments to create new molecules[162,187]. *De*

novo DOCK is a powerful tool, but it appears to target expert users, particularly those who are already familiar with DOCK6. Many users have pipelines that use alternative docking software, so restricting the docking program options limits *de novo* DOCK's potential users. Additionally, the accuracy of some scoring programs, particularly those that have been trained on a specific subset of receptor types, varies based on the protein target[181]. Thus, users may wish to use the docking program that is most accurate for their biological target, which may or may not be DOCK6. In contrast, AutoGrow4 provides two predefined options for docking as well as the ability to easily incorporate additional docking and scoring programs.

GANDI, another free and open-source program, takes a different approach to *de novo* CADD while still using a GA[167]. Prior to running GANDI, the user must dock the library of source molecular fragments into the target protein pocket[167] using the programs DAIM[216], SEED[217], and FFLD[218]. GANDI evolves the docked fragments using a parallel GA model, often referred to as an island model[167], which evolves multiple populations of fragments independently with occasional events where fragments are exchanged between populations. To generate a final novel compound, GANDI links promising fragments together using linker compounds, which are not evolved but rather picked from a predefined table[167]. Both AutoGrow4 and GANDI are MPI enabled, but GANDI is limited to the Linux operating system[167] and targets experienced users who are familiar with the suite of DAIM[216], SEED[217], and FFLD[218].

Machine learning has also recently been applied to *de novo* design. REINVENT is a free and open-source algorithm that uses recurrent neural networks and reinforcement learning to generate *de novo* compounds[212]. LigDream, also free and open-source, focuses training on the 3D shape of an input compound[213]. To predict novel compounds, machine learning approaches such

as these are typically trained on preexisting drug-like compounds[212,213]. Additionally, they often require known ligands to seed the run. For instance, LigDream is applicable only to lead optimization projects because LigDream assesses "good" ligands for the target protein based on the provided lead compounds. LigDream does not factor in any other information about the protein target[213] (e.g., its structure), which limits its applicability to lead optimization[213]. AutoGrow4 can similarly optimize known ligands ("Chapter 3.3.2: PARP-1 Lead Optimization"), but it can also generate novel compounds in the absence of known inhibitors[6] ("Chapter 3.3.1: Large-Scale *de novo* PARPi Run.")

1.2.5.2 Motivation for Developing AutoGrow4

In this section, I will explain the limitations of free and open-source *de novo* CADD programs that motivated me to develop AutoGrow4. AutoGrow4 is a highly customizable, free and open-source program that can be easily incorporated into most drug-design pipelines[6]. It provides many predefined user options (e.g., multiple chemical-property filters and docking-software options), but it also allows users to incorporate new features to match their needs[6]. Additionally, AutoGrow4 has been optimized to work in various computational environments and can be scaled according to users' available resources[6].

First, while successful drug design requires careful consideration of ADME-PK properties[175], many open-source programs similar to AutoGrow4 lack extensive chemical-property filters (e.g., the option to filter compounds with poor ADME-PK properties), or require users to be familiar with drug-likeness parameters and set the program parameters

accordingly[6,156,162,167,212,213]. Programs that lack appropriate filter options use computational resources to explore poor candidates that could have been excluded from the beginning. Of the open-source programs reviewed in Table 3, AutoGrow4 provides the most comprehensive set of predefined chemical drug-likeness filters and is the only program that allows users to incorporate their own custom filters[6,156,162,167,212,213]. Both *de novo* DOCK and GANDI provide options for constraining some chemical properties, such as molecular weight and the number of rotatable bonds[162,167], but these programs are limited to a predefined set of properties and do not incorporate many properties that can predict drug-likeness (e.g., polar surface area[169]) or substructure filters (e.g., PAINS filter[174]). Additionally, AutoGrow3 and AutoGrow4 are the only open-source programs reviewed in Table 3 that provide predefined options for commonly used drug-likeness filters[6,156]. However, *de novo* DOCK and GANDI do provide the option to constrain several physiochemical properties, such as MW[162,167]. Given their limited documentation, these programs require users to have a more advanced understanding of drug-design and drug-likeness, thereby limiting their ease-of-use.

Second, many other *de novo* CADD programs have major usability limitations. CADD should reduce much of the unnecessary work in designing drugs, so the ease-of-use must be accessible/reasonable for more than just experts in computation and cheminformatics. Unfortunately, both *de novo* DOCK and GANDI rely on multiple third-party programs to prepare and manage files, and they do not provide extensive small molecule/fragment libraries[162,167]. For instance, *de novo* DOCK requires manual preparation of sphere and grid files (the files that describe DOCK6's conformational sampling function) prior to docking[162]. While *de novo* DOCK does provide several scripts to aid these processes, it also requires the additional installation and

use of programs[162] such as the visualization software Chimera[219]. GANDI requires users to manually prepare and dock all seed compounds/fragments prior to running, which also requires installation and operation of multiple additional programs (e.g., VMD[220], DAIM[216], SEED[217], FFLD[218], CgenFF[221–223])[167]. On the other hand, AutoGrow3 and AutoGrow4 automate all file conversion and docking processes[6,156] and so only require users to provide the coordinates of the desired pocket, a PDB file of the protein target, and the library of small-molecule seeds. AutoGrow4 provides multiple general-purpose small-molecule libraries, detailed tutorials, and supporting scripts to prepare custom libraries[6]. Additionally, AutoGrow3 and AutoGrow4 provide Docker containers (i.e., a package of software designed to run a program with minimal user effort) to simplify the installation of dependencies[6,156].

Lastly, many machine-learning-based *de novo* CADD programs, including LigDream[213] and REINVENT[212], are unable to perform lead generation in the absence of pre-existing inhibitors and were therefore insufficient for the PARPi design goals described in this dissertation. These techniques are useful when there are many known ligands with which to seed the runs, such as when optimizing inhibitors that bind the PARP-1 catalytic site, but they are not applicable in the absence of preexisting inhibitors, such as when designing novel inhibitors to bind the PARP-1 DBD. Additionally, LigDream[213] and REINVENT[212] do not consider the structure of the protein target, so they cannot predict novel lead compounds that avoid interactions with amino acids implicated in resistance mechanisms, such as when designing compounds that bind the PARP-1 catalytic pocket when Y907 is phosphorylated.

Previous versions of AutoGrow (e.g., AutoGrow3) did provide many features necessary for *de novo* CADD applied to PARP-1, such as the ability to consider the structure of the target,

perform both lead optimization and lead generation, and filter candidates according to ADME-PK physiochemical properties. However, AutoGrow3 is slow, restricted to an outdated docking program, and does not scale well to larger CADD runs (see "Chapter 2.4: Results and Discussion".) The drug-likeness filters provided with AutoGrow3 are also limited. In contrast, AutoGrow4 provides all the benefits of the previous AutoGrow versions with many added features and improved performance. Additionally, the ease-of-use and automation of many processes, such as protein/ligand file preparation and docking, make AutoGrow4 ideal for the experiments described in this dissertation.

1.3 Aims of the Dissertation

All current clinically successful PARPi act as competitive inhibitors that bind to PARP-1 using the same set protein-ligand interactions, despite many attempts to identify novel inhibitors[12,17,18,111,224]. As discussed in "Chapter 1.1: Biological Background," current treatment strategies are vulnerable to resistance mechanisms because of overreliance on inhibitors that mimic NAD+ binding. New PARP-1 inhibition strategies that do not rely on binding to the catalytic pocket may improve cancer treatment.

To help address this shortcoming, I used *de novo* computer-aided drug design (CADD) to search for unique novel candidates. Given the limitations of preexisting open-source *de novo* CADD programs (detailed in "Chapter 1.2.5.2: Motivation for Developing AutoGrow4"), I chose to design a new, highly optimized CADD program called AutoGrow4. I then used AutoGrow4 as

a search engine to find novel candidate PARP-1 inhibitors. Because of the wealth of preexisting information about PARP-1 and PARPi, PARP-1 both served as a test to verify AutoGrow4 utility and as a biological subject of great importance.

In addition to developing AutoGrow4 and using the PARP-1 catalytic domain as a test case, I also (1) designed novel candidate inhibitors predicted to bind with strong affinities to the catalytic binding pocket generally; (2) designed candidate inhibitors predicted to bind to the catalytic binding pocket independent of Y907 phosphorylation, a strategy that may circumvent a common PARPi resistance mechanism; (3) applied computational hot spot mapping to the PARP-1 interdomain interfaces to identify non-catalytic binding pockets; and (4) used AutoGrow4 to identify multiple candidate inhibitors that are predicted to bind PARP-1 with strong affinities outside the catalytic pocket.

2.0 AutoGrow4: Implementation and Benchmarks

This chapter describes the design improvements and features of AutoGrow4, compound libraries included in the AutoGrow4 download, and benchmark comparisons of AutoGrow4 verses its predecessor AutoGrow3 (version 3.1.3). The AutoGrow4 manuscript was published under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium"[6]. The work in this chapter expands on that manuscript. It is adapted and reprinted with rights and permission:

> **Jacob O Spiegel**†, & Jacob D Durrant. (2020) AutoGrow4: An open-source genetic algorithm for *de novo* drug design and lead optimization. *Journal of Cheminformatics*, 12, 25. https://doi.org/10.1186/s13321-020-00429-4.
>
> † Jacob O. Spiegel should be regarded as first author.

I am the first author of the published manuscript, having written the entire AutoGrow4 codebase, performed many of the experiments discussed in the paper, and analyzed the data. I designed the layout for all figures in the paper with Dr. Jacob Durrant. Dr. Durrant refined and generated many of the high-quality images for publication. Dr. Durrant also provided guidance and insight as described in the acknowledgement section. All writing in this chapter is original content written by Jacob O. Spiegel.

2.1 Overview and Rationale

AutoGrow4, a milestone in the AutoGrow program series, introduces many new features and improves the approach of the program. I was motivated to develop AutoGrow4 because AutoGrow3 was inefficient, limited in its scalability, restricted to a single docking program, unable to maintain chemical diversity across multiple generations, and provided only limited support for chemical filters and reactions. AutoGrow3 was capable, but not ideal for many large-scale *de novo* CADD applications, such as the design of novel candidate PARPi throughout this dissertation.

This chapter will detail the development of AutoGrow4 and the improvements that make it superior to AutoGrow3. I will begin with an in-depth discussion of the new features I developed. Then I will describe two benchmark experiments that compare AutoGrow4 and AutoGrow3 in terms of computational efficiency (i.e., the time required to run each program) and performance (i.e., the docking affinities and chemical diversity of the AutoGrow-produced populations). Additionally, I will examine the variables that contributed to the improved success of AutoGrow4 and will close this chapter with several proposed improvements for future AutoGrow4 releases.

These benchmarks demonstrate AutoGrow4's improved ability to produce novel candidate inhibitors and computational efficiency. However, I provide additional comparisons of AutoGrow4 with alternative CADD techniques, including VS and three other *de novo* CADD programs in "Chapter 4.0: Comparison of CADD Techniques."

2.2 AutoGrow4: Development and Improvements

AutoGrow is a free Python-based open-source program for *de novo* CADD that uses a genetic algorithm (GA) to create new molecules for a given protein target. AutoGrow starts with a population of seed compounds that are used to create a new population of potential solutions, often referred to as a generation. It then docks these compounds into the target protein pocket and scores their binding. These scores are used to rank the fitness of these potential solutions. New generations of solutions are seeded with the top-scoring molecules from the previous generation.

AutoGrow 1.0.0, released in 2009, was one of the first *de novo* CADD programs to perform fully flexible docking and was one of only a few free open-source programs for *de novo* CADD[155]. AutoGrow3 (version 3.0.0), published in 2013 and the most recent previously published version, improved on the choice of docking software, the re-implementation of compound creation, and molecule scoring[156]. At the time, these changes improved AutoGrow's accuracy and efficiency, but docking software, cheminformatic libraries, and multithreading capabilities have since advanced[165,205,214]. I have rewritten AutoGrow4 to take advantage of these advancements, resulting in a faster, more stable, and more modularly designed program. Additional implementation choices and improvements that are not pertinent to understanding the AutoGrow4 algorithm are provided in "Appendix A: Additional AutoGrow4 Implementation Details."

2.2.1 Ligand Handling

One of the most significant improvements to AutoGrow4 in how it handles ligands. Previous releases of AutoGrow worked with small molecules in 3D PDB format[155,156], meaning that tasks such as substructure searches, molecular alignments, and *in silico* reactions required computationally expensive 3D calculations[155,156], as well as extensive custom codebase to address each operation[155,156]. With such a complex system of operations, maintenance and expandability were problematic.

In contrast, AutoGrow4 was designed with accessibility and advancement in mind. AutoGrow4 uses Daylight's SMILES notation, a string notation that describes small molecules, and leverages the third-party Python library RDKit[165] to perform most ligand operations, including crossover, mutation, and filtration. This approach significantly reduces the computational cost of generating and manipulating compounds. RDKit also reduces the need for custom handling code, making AutoGrow4 more maintainable. Furthermore, many modules, including *in silico* reactions, filters, and docking software, accept plugin code. Plugin architecture allows users to incorporate their own customized functions and libraries. This efficient style of design is not only beneficial for users but also facilitates long-term maintenance and expansion.

2.2.2 Operators: Population Generation via Crossover, Mutation, and Elitism

An AutoGrow4 run begins with a user-defined initial seed population of compounds. This input population, referred to as generation-zero, can consist of either random fragments and small

molecules (for *de novo* design) or known ligands (for lead optimization). From this source population, AutoGrow4 creates the first generation of compounds through three operations: crossover, mutation, and elitism (Figure 9), each of which is explained in more detail below. For all subsequent generations, AutoGrow4 creates new compounds in a similar manner, but a subpopulation of seed compounds is derived from the immediately preceding generation (Figure 9).

AutoGrow4 provides a unique name for all generated compounds. The name includes the generation the compound was created in, whether it was created by crossover or mutation, and which compound(s) are its parents. Additionally, the names of mutation-generated compounds identify the *in silico* reaction that created the compound. AutoGrow4's naming scheme allows users to reconstruct the lineage of any given compound and, with an accessory script, analyze its evolution. This can help provide a potential path for synthesis.

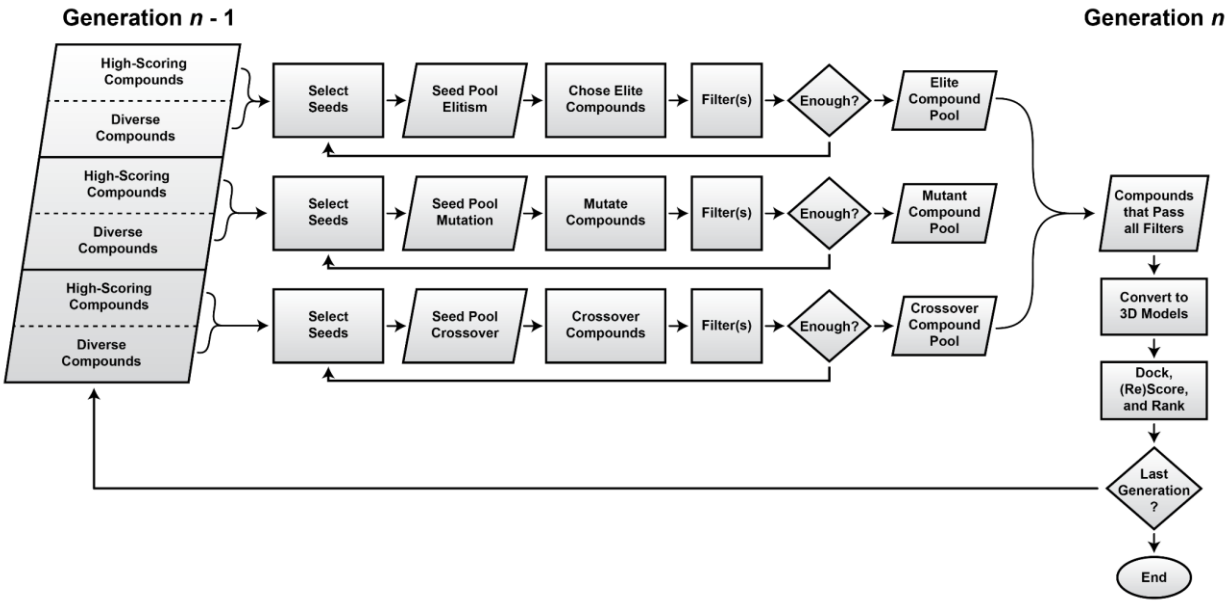


Figure 9. Process-flow diagram of the AutoGrow4 algorithm.

AutoGrow4 generates three independent seed pools consisting of well-scoring and diverse compounds from the previous generation (*n* - 1). These seed pools are used as parents to create the next generation of compounds (*n*) via crossover, mutation, and elitism. Gypsum-DL[214] is used to convert SMILES to 3D. Then, the compounds are converted to a dockable file format (e.g., PDBQT), docked, (re)scored, and ranked according to the fitness function. This process is repeated until the user-defined number of generations have been completed. Figure is from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. This figure is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License which "allows unrestricted use, distribution, and reproduction in any medium"[6].

2.2.2.1 Crossover Operator

Crossover operators create a child solution by merging two parent solutions (Figure 10A). In the case of AutoGrow4, the parent and child solutions are small molecules. To generate a child compound, AutoGrow4 first picks two random compounds from the selected list of seeds derived from the previous generation (Figure 9 p.74). Next, AutoGrow4 searches for the largest substructure shared between the two parent compounds. By default, AutoGrow4 requires that a minimum of four atoms be shared between parents; otherwise, it moves on to the next parent pair. This minimum number of shared atoms is a user-controllable parameter. Finally, the child is generated by randomly combining decorating moieties of the parents that branch off the shared substructure (Figure 10A).

AutoGrow3, the most recent previous version of AutoGrow, used the program Ligmerge to perform crossovers[156,225]. Ligmerge uses 3D representations of compounds, which requires geometric calculations to merge the parent compounds[225]. AutoGrow4 replaces Ligmerge with an internal module that relies on RDKit to generate child compounds from SMILES strings of the parents. This change reduces the computational cost of compound generation and simplifies the crossover portion of the AutoGrow codebase.

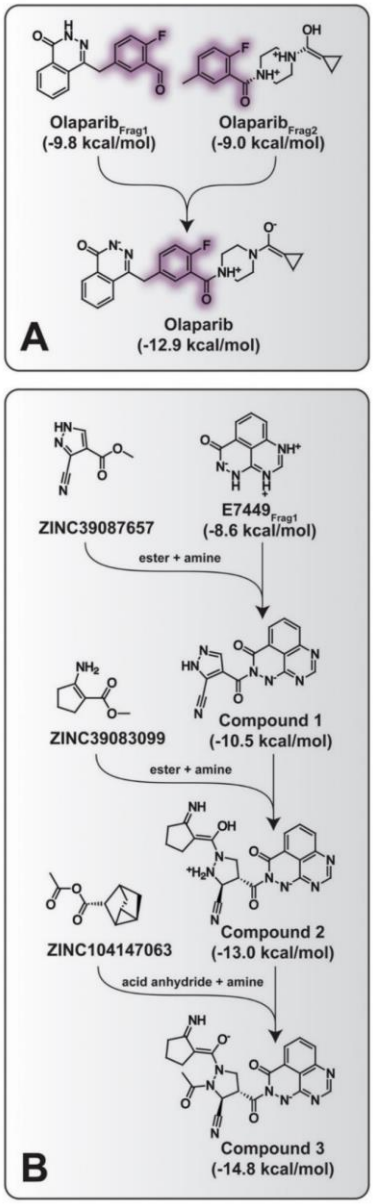


Figure 10. Crossover and mutation examples.

The lineage of two compounds generated from a lead-optimization run seeded with PARPi and PARPi fragments. All reported docking scores were calculated using QVina2. Occasionally the crossover operator will create a child identical to one of the parents. I have omitted these intermediates from the figure. A) AutoGrow4 recreated the PARPi olaparib from two source fragments. Their largest commonly shared substructure is highlighted in purple. B) A well-scoring compound derived from E7499 (a known PARPi) illustrates the modifications made by the mutation operator. Figure is from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. This figure is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License which "allows unrestricted use, distribution, and reproduction in any medium"[6].

2.2.2.2 Mutation Operator

AutoGrow's mutation operator creates child compounds by performing *in silico* reactions on a parent compound from a previous generation (Figure 10B p.76). These children compounds are slightly modified from their parent(s). AutoGrow4's mutation operator has been improved from previous AutoGrow releases. AutoGrow3 used a program called AutoClickChem[226] as its the mutation operator, which was limited to a set of 36 reactions based on Dr. Barry Sharpless's "click chemistry" reactions (a set of high-yielding reactions)[226,227]. AutoClickChem performs reactions using 3D models, which requires extensive custom code to properly perform substructure searches, molecule alignments, and reactions[156,226]. I streamlined AutoGrow4's mutation module and expedited the mutation operation by leveraging Daylight's SMARTS and RDKit code.

I implemented a plugin-style design for this operator, which allows the *in silico* reactions to be easily expanded. I converted the AutoClickChem[226] set of click chemistry reactions to Daylight's SMARTS-reaction notation, referred to as the AutoClickChemRxn set. I also added a second reaction library based on the 58-reaction set published in Hartenfeller *et al*. 2011[228], referred to as the RobustRxn set. AutoGrow4 provides a third reaction set consisting of the merged AutoClickChemRxn and RobustRxn sets, referred to collectively as the AllRxn set. The AllRxn set contains 94 reactions in total. All sets were manually inspected and extensively unit-tested. AutoGrow4 also permits custom reaction libraries to be easily incorporated, thereby becoming a long-term expandable codebase. The tutorial packaged in the AutoGrow4 download (Appendix D: AutoGrow4 Manual and Tutorial), which is available for free at http://durrantlab.com/autogrow4/, details how to create and incorporate custom reaction sets.

Another major improvement to AutoGrow4's mutation operator is to the complementary molecule libraries used to supplement reactions requiring more than one reactant. Of the 94 reactions in the AllRxn set, 79 reactions require two reactants. A given reaction that has multiple reactants may require specific functional groups not present within the population seeding it, so AutoGrow4's mutation module provides a library of complementary molecules/fragments in SMILES file (SMI) format to act as the secondary reactants. These libraries are included in the AutoGrow4 download. Further details regarding AutoGrow4's default libraries can be found in "Section 2.3.2: Preparation of Complementary Molecule Libraries." The tutorial included in the AutoGrow4 download (Appendix D: AutoGrow4 Manual and Tutorial) describes how users can incorporate custom complementary molecule libraries.

To showcase the impact that mutations can have on a predicted binding affinity, in Figure 11 I present an example of a single mutation that produced a well-scoring compound from an worse-scoring parent. The parent compound (Figure 11 top right) has a QVina2 score that is worse than 83.92% of the other compounds in the same generation. In contrast, the child compound (Figure 11 bottom) has a QVina2 score that is better than 82.35% of the other compounds in its generation (Figure 11). This compound was created *in silico* via a Fischer's indole synthesis reaction with phenylhydrazine (Figure 11)[228–230], which was taken from the complimentary molecule library (ZINC34276250, Figure 11 top left). Of course, not all mutation events create such dramatic improvements, and offspring can also score worse than their progenitors.

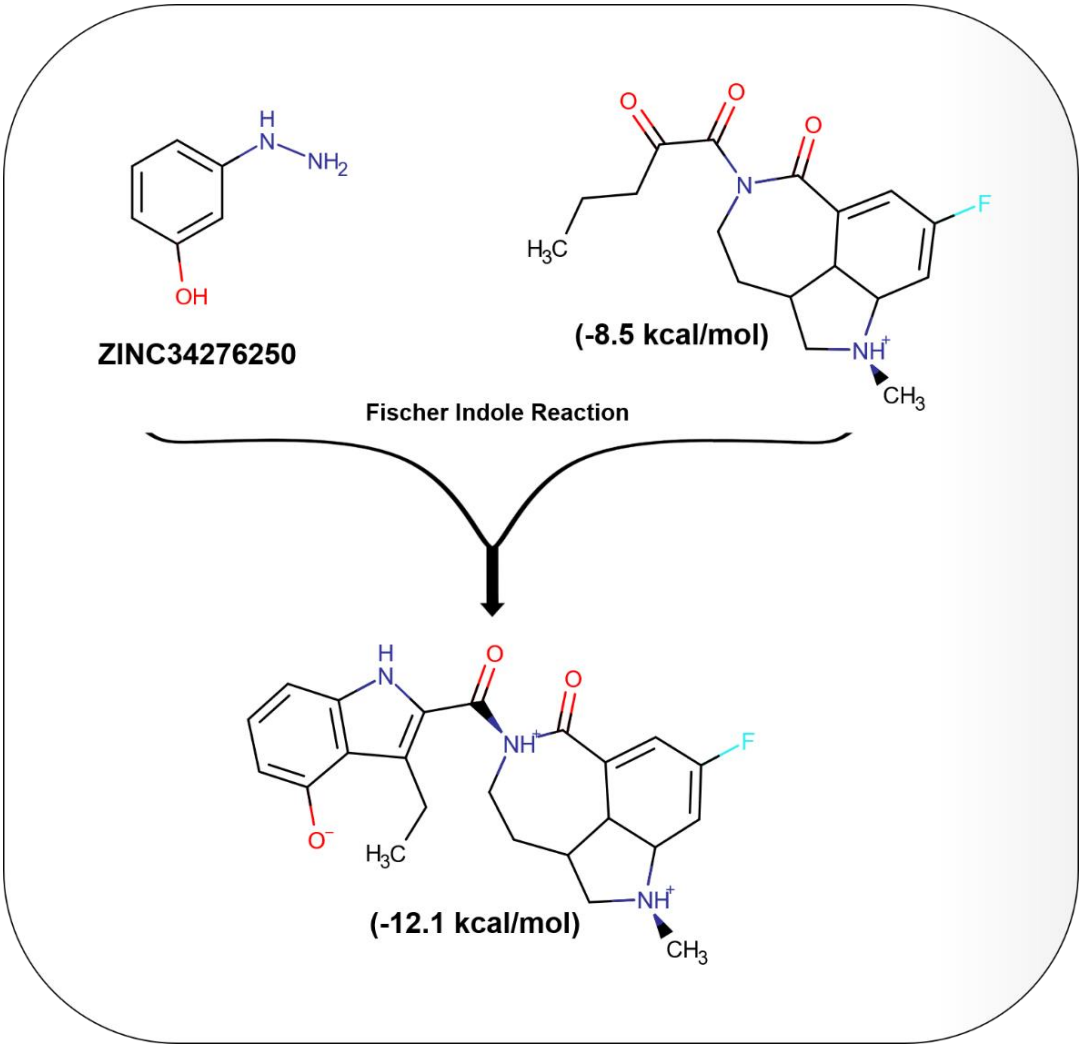


Figure 11. A fit compound generated from a worse-scoring parent via mutation.
From a poorly scoring parent (top right), this mutation event created a compound (bottom) that scored better than 82.35% of all compounds in its generation. ZINC34276250 (top left) served as the required phenylhydrazine for the reaction[228–230]. The parent compound was produced in the second generation and the child was created in the third generation of a PARPi lead-optimization run seeded with PARPi and PARPi fragments. All reported docking scores were calculated using QVina2 docking compounds into the catalytic pocket of PARP-1 (PDB: 4R6E Chain A[18]).

2.2.2.3 Elitism Operator

Elitism progresses a sub-population of the fittest compounds from the prior generation into the next generation without alterations. AutoGrow4's elitism operator has also been improved from previous versions to allow reassessment of compounds that have advanced. Because the pose sampling of many docking programs, including AutoDock Vina[185], is stochastic (Table 2 p.55), the programs might generate slightly different poses each time a compound is docked. AutoGrow4's reassessment includes regenerating variants with Gypsum-DL and redocking. Reassessing compounds adds additional computational costs; thus, this is an optional feature.

2.2.3 Chemical Filters

De novo molecule generation provides an opportunity to explore a wide range of chemistry space, but much of that space includes compounds that are not likely to become clinical drugs due to poor physiochemical properties (i.e., compounds that are not "viable")[148–150,178–180]. Common concerns include small-molecule solubility, biological reactivity, and similarity to common metabolites[163,168–174]. AutoGrow filters compounds according to their predicted physical and chemical properties before performing its computationally expensive docking step (Figure 9 p.74), thereby reducing resource investment in compounds that are unlikely to be viable drugs. These types of filters may remove some "good" candidates, but they improve the success rate of CADD techniques overall[163,168–174]. If an insufficient number of compounds pass the filter(s), AutoGrow generates more compounds to populate the generation.

AutoGrow4 has nine predefined filter options, described above in "Section 1.2.3.2: Chemical Drug-Likeness Filters" and shown in Table 1 (p.53). Any number of these filters may be run in series. AutoGrow4 also allows plugins to easily incorporate new chemical property filters. Descriptions for creating and incorporating custom filters are provided in the AutoGrow4 tutorial.

2.2.4 Conversion of SMILES to 3D PDB

AutoGrow4 uses SMILES to perform compound creation via mutation and crossover, but docking programs require 3D structures, so all compounds must be converted to a 3D format (Figure 9 p.74). AutoGrow4 uses Gypsum-DL, a free open-source Python software, to convert SMILES to a 3D format[214]. Gypsum-DL enumerates the ionization, tautomerization, chiral, cis/trans isomers, and ring-conformational forms of each compound, producing one or more 3D structures[214]. Because a single compound may have multiple 3D variants, Gypsum-DL—and in turn, AutoGrow4—allows users to define the maximum number of variations produced per SMILES. Gypsum-DL handles ionization with the free, open-source program Dimorphite-DL[214,231]. More information about Gypsum-DL is provided in "Appendix B: Gypsum-DL." Users can control the pH conditions that Dimorphite-DL uses for ligand protonation.

Protein-ligand docking is the most computationally expensive and time-consuming component of AutoGrow, and every compound variant generated by Gypsum-DL results in a docking event. Thus, the more variations generated per molecule, the more computationally

intense the simulation. On the other hand, sampling more variants provides more tests against the protein target, yielding a better chance of observing good binding.

2.2.5 Docking

Since AutoGrow3's release in 2013[156], the speed and accuracy of docking programs have improved[187,205,207]. AutoGrow4 was designed to take advantage of these advancements and to anticipate future developments. Docking in AutoGrow4 is now a modular component that can be easily updated, replaced, or substituted with custom code. A thorough description of docking is provided in "Section 1.2.4: Protein-Ligand Interactions and Protein-Ligand Docking."

AutoGrow3 was upgraded from using AutoDock (version 4.0.1) to AutoDock Vina (version 1.1.2) (Vina) but was limited to this single docking software option[156]. AutoGrow4 provides broader docking-program support. It maintains a Vina option but adds a new default docking software, QuickVina (version 2.1) (QVina2)[205], a Vina-based docking program that runs about two-fold faster than standard Vina with minimal change in accuracy[205]. AutoGrow4 comes pre-packaged with both Vina (version 1.1.2) and QVina2 (version 2.1), and also provides an option for specifying custom docking executables. I have also implemented a plugin structure that enables long-term expandability and user customization. With the appropriate customized code, virtually any software for assessing a ligand in a pocket could be substituted into AutoGrow4.

2.2.6 File Conversion for Docking

Protein-ligand docking programs require structure files of both the receptor (protein) and ligand (small molecule), and the required file format varies from program to program. For example, Vina and QVina2 require the receptor and ligand files to be in PDBQT format, a file format similar to PDB[185,205]. However, the docking program Glide by Schrödinger accepts ligands in multiple formats (e.g., Maestro, SD, Mol2, or PDB)[232]. AutoGrow4 provides a module that prepares these files for docking.

Scoring and docking programs, particularly machine learning-based programs, may be sensitive to differences in file preparations such as partial-charge assignment. For instance, the rescoring programs NNScore1 and NNScore2 were trained with compounds converted to the PDBQT format by MGLTools and docked by Vina (version 1.1.2)[215,233,234]; it is not recommended to use NNScore1 or NNScore2 on compounds prepared by any program other than MGLTools. AutoGrow4 provides code to convert both receptors and ligands using AutoDock MGLTools and Open Babel[164,215], as well as options plugins for alternative file conversion.

2.2.7 Assessing Fitness

GA require a fitness metric to assess the quality of each solution. A selection process later applies a selective pressure to choose the most fit compounds for elitism or to seed the next generation, progressively improving potential solutions (Figure 9 p.74). AutoGrow4 uses two such metrics: a primary fitness metric related to docking score, and a secondary fitness metric of

structural diversity. Previous releases of AutoGrow used only a single fitness metric based on a compound's docking score. This section will detail the implementation of the primary and secondary fitness metrics.

2.2.7.1 Primary Fitness Metric: Docking Score

AutoGrow4's primary fitness metric is an assessment of a ligand's binding to a protein pocket. By default, this metric is the binding affinity prediction of a compound's best pose calculated by the docking software, but AutoGrow4 also allows rescoring of a docked pose by standalone scoring functions. AutoGrow4 comes with built-in options to rescore docked poses with NNScore1 and NNScore2[233,234], but appropriate plugin code can incorporate virtually any (re)scoring assessment into AutoGrow4 to be used as the primarily fitness metric.

Additionally, a ligand's docking score or reassessment score can be divided by the number of non-hydrogen atoms in the ligand. This adjusted score estimates the docking fitness per structural element and is referred to as ligand efficiency[235].

AutoGrow4 is currently configured and intended to be used as a high-throughput technique for early stage lead discovery and optimization. Assessing compounds by their predicted binding affinity, as calculated by a docking program, is ideal for this application because docking has a low computational overhead and is highly parallelizable[185,205]. Compound leads predicted by AutoGrow4 should be further evaluated by more accurate techniques.

2.2.7.2 Secondary Fitness Metric: Structural Diversity

Previous versions of AutoGrow based compound selection solely on the docking fitness, but selection by a single metric can result in premature convergence at a local optima due to population homogenization[147]. In cases of extreme population homogeny, an evolutionary "cul-de-sac" condition develops where the algorithm becomes trapped in local optima and is unable to create unique compounds, instead repeatedly recreating the same compounds.

To offset this possibility, AutoGrow4 introduces a secondary fitness metric called the diversity score that is assigned to each compound following docking. This metric is optionally used to select a portion of the seed population based on structural uniqueness compared to compounds in the previous generation. By seeding a generation with a pool that comprises a combination of the best-scored and the most chemically diverse compounds, AutoGrow4 delays population convergence while still applying a selective pressure for desired binding affinity.

To calculate the diversity score, AutoGrow4 compares molecular fingerprint representations of each compound. Molecular fingerprints are simplified representations of small molecules as series of binary digits (bit-strings)[236] where each digit in the fingerprint represents a chemical feature. Comparing fingerprints provides a quantitative measurement of the similarity of two compounds. RDKit's Dice's coefficient similarity scoring is used to perform pairwise comparisons of the fingerprints of every compound in a generation[165,237,238]. The similarity, *s*, of two compounds, mol_A and mol_B, is determined by Equation 1.

$$s(F_A, F_B) = \frac{|F_A \cap F_B|}{|F_A| + |F_B|}$$

Equation 1: Similarity between two compounds, mol_A and mol_B, where F_A and F_B are the fingerprint bit-strings for mol_A and mol_B, respectively.

For a given pairwise comparison, the value of s is a number ranging between 0.0 and 1.0, with 1.0 representing two perfectly matched molecules and 0.0 representing two completely different molecules. The diversity score, d, of a given compound, mol_M, is the sum of all its similarity scores with all other compounds within a generation. This diversity score is calculated using Equation 2.

$$d(mol_M) = \sum_{N \neq M}^{n} s(F_M, F_N)$$

Equation 2: Diversity score of mol_M, where the summation is over all compounds, mol_N, in a generation that are not mol_M, and $s(F_M,F_N)$ is the similarity score of the fingerprint bit-strings for mol_M and mol_N, respectively (See Equation 1). AutoGrow4 uses fingerprints that are 2048 bits.

The diversity score provides a metric for compound uniqueness compared to other compounds in its generation. AutoGrow4 uses both a compound's affinity metric and its diversity score in two independent selections to choose the fittest compounds to seed the next generation (Figure 9 p.74), and allows users to control the number of compounds chosen for each metric. By seeding each generation with a combination of the best-scored and most unique compounds,

AutoGrow4 can search a wider range of chemistry space while still maintaining a selective pressure for well-scored compounds. Actively incorporating diversity also curtails population homogenization, thus enabling a run to escape local optima.

2.2.8 Compound Ranking and Seed Selection

The selection process by which compounds are either chosen to seed or chosen for elitism is a critical aspect of AutoGrow. This process occurs at the beginning of each generation (Figure 9 p.74). Broadly speaking, three of the most commonly used strategies for selection in GA are Ranking selector, Roulette selector, and Tournament selector[151], which are reviewed in "Section 1.2.1.4: Ranking and Selection Approaches."

Previous versions of AutoGrow exclusively used a Ranking selector. I have expanded AutoGrow4 to also provide Roulette and Tournament selectors. The Roulette selector provided in AutoGrow4 uses non-linear weighting to increase selective pressures in favor of well-performing compounds. These selectors can be applied to any fitness metric including docking score, revaluated affinity score, and diversity score.

Additionally, AutoGrow3 chose a single sub-population of compounds (the seed population) from the previous generation to serve as the parent compounds for all crossover, mutation, and elitism. In contrast, AutoGrow4 performs three independent pulls to form three seed populations, which are then used to create new compounds by means of a single process (e.g., crossover, mutation, or elitism) (Figure 9 p.74). In deterministic selections such as the Ranking selector, this change in seed selection has no effect and the three seed populations are identical.

However, for a stochastic selector such as the Roulette or Tournament selector, the three seed populations may be quite different with potential overlap. The three independent selections thus minimize the chance of excluding a fit ligand (Figure 9 p.74). As described in "Section 2.2.7: Assessing Fitness," AutoGrow4 selects its seed compounds based on their primary (docking) and secondary (diversity) fitness metrics while allowing users to control how many seed compounds are selected by each metric. The chosen selection method is used for selecting seeds for both fitness metrics.

Lastly, AutoGrow4 separates seed selection and elitism selection. In AutoGrow3, the compounds chosen for elitism were also used to seed crossover and mutation. AutoGrow4 separates these variables to allow users to control the selective pressures of seeding and elitism independently (Figure 9 p.74).

2.2.9 Availability and Requirements

Project name: AutoGrow4

Project home page: http://durrantlab.com/autogrow4/

Operating systems: Linux, macOS, and Windows (via Docker)

Programming language: Python 2/3

Other requirements: RDKit, NumPy, SciPy, funct_timeout, Mpi4py (optional), MGLTools (optional), Open Babel (optional)

License: Apache License, Version 2.0

2.3 Methods

2.3.1 Receptor File Preparation

For all AutoGrow runs in this chapter, I used the catalytic domain of PARP-1. I obtained a crystal structure of PARP-1's catalytic domain (PDB ID: 4R6E Chain A) from the Protein Data Bank (PDB)[17,239] and removed all atoms except chain A of the protein (4R6E:A) and the niraparib inhibitor bound in the pocket. Then I used the PDB2PQR server (version 2.1.1 using default settings) to add hydrogen atoms and adjust the protein to biological pH conditions (pH 7) [240,241]. This optimized the protein's hydrogen bond network for docking. I then converted the PQR file produced by PDB2PQR back to a PDB file using Open Babel (version 2.3.1)[164].

Vina and QVina2 require user-defined coordinates and dimensions of the target pocket. Choosing to target the catalytic site, I selected five protein residues that flank the bound niraparib ligand in the crystal pose: E763, I872, G888, T907, and E988. Using the Scoria Python library[242], I calculated a bounding box that contained all five of the selected residues. To ensure that the entirety of the pocket was included, I expanded the dimensions by several Å in all directions. The final dimensions of the box were 25.0 Å, 16.0 Å, and 25.0 Å in the x, y, and z directions, respectively. I also determined the geometric center of the pocket using the same bounding box. These coordinates and dimensions are relative to the specific orientation of the PDB file.

The final PDB file and pocket coordinates were used for all AutoGrow runs in this dissertation. The PDB file is provided in the AutoGrow4 download, and step-by-step instructions for processing and determining coordinates for docking are provided in the AutoGrow4 manual.

2.3.2 Preparation of Complementary Molecule Libraries

As described in "Section 2.2.2.2: Mutation Operator," reactions often require multiple reactants, which may or may not exist in the parent generation. For this reason, AutoGrow4 provides complementary molecule libraries organized by functional groups to supplement reactions with multiple reactants. All reactions contain one compound from the previous generation.

To generate these complementary molecule libraries, a set of 19,274,338 commercially available small molecules with molecular weight (MW) less than 250 Da and an octanol–water partition coefficient (logP) less than 5.0 were downloaded from the Zinc15 database on December 19, 2019[230]. The set was further filtered using the Lipinski* filter, which allows for no violations of Lipinski's rule of fives (Table 1 p.53). Compounds were sorted into functional-group categories, and any compound that did not contain at least one functional group from the default reaction set (AllRxn) was discarded. To manage the download size and speed of the large complementary molecule files, each functional-group category was limited to a maximum of 5,000 compounds, selecting those with the lowest MW. These compounds are provided in the AutoGrow4 download and serve as the default complementary molecule libraries for the three sets of *in silico* reactions. All AutoGrow4 runs in this dissertation used this set of complementary molecule libraries.

2.3.3 Efficiency Benchmarks

To compare the computational efficiency of AutoGrow4 and AutoGrow3, I compared the time required for each program to complete five generations. Both programs were set using as comparable of settings as possible. Additionally, I tested the computational efficiency of AutoGrow4 when using several new features.

2.3.3.1 Efficiency Benchmark: File Preparation

AutoGrow relies on an initial population of compounds to seed the first generation. Benchmarks comparing the efficiency of AutoGrow4 with its predecessor, AutoGrow3, require the source compound library to be supplied to both programs. AutoGrow3 provided a set of 117 naphthalene-containing small molecules in PDB format to be used as source compounds[156]. AutoGrow3 only accepts PDB files, whereas AutoGrow4 only accepts SMI files, so I converted this set of 117 PDB files to SMILES using Open Babel[164] and RDKit[165]. This list of naphthalene-containing small molecules is provided with the AutoGrow4 source-code as an SMI file.

2.3.3.2 Efficiency Benchmark: Run Conditions

To benchmark AutoGrow4's efficiency compared to its predecessor, I tested six conditions for AutoGrow4, varying the docking software and the maximum number of variants per molecule. AutoGrow3 was tested using only a single condition because it is limited to AutoDock Vina

docking and only creating a single 3D variant per molecule[156]. The AutoGrow4 benchmarks docked with either AutoDock Vina (version 1.1.2) or QuickVina2 (version 2.1), and varied the Gypsum-DL setting of maximum variants per molecule between one, three, or five. Runs of these seven test conditions were repeated independently 24 times. Benchmarks were tested in a Python 2.7 environment because AutoGrow3 is not Python 3 compatible.

In addition to computational variables such as memory, number of computer processors, and Python environment, I controlled for variables including population size, mutation reaction sets, seed molecules, the ratio of population size to seed size, and selection method. In all cases, Ghose* and Lipinski* filters (Table 1 p.53) were applied to the AutoGrow runs. Settings for these runs are provided in Appendix JSON 1 and Appendix JSON 2.

These experiments were performed on computer clusters at the University of Pittsburgh's Center for Research Computation (CRC). To control for variations in memory and processor speeds, I ran all benchmarks on the same set of SMP nodes. Experiments were performed using SMP multiprocessing because AutoGrow3 is not MPI enabled[156]. The SMP nodes were 12-core Xeon E5-2643v4 3.40 GHz Broadwell nodes with 512 GB RAM.

2.3.3.3 Efficiency Benchmarks: Post-Run Data Processing

For each run condition, I pooled the results of each completed generation into a corresponding pool. For instance, I pooled the compounds produced during the first generation of the AutoGrow4 runs that used Vina docking and a Gypsum-DL setting of one variant per molecule. Because the parallel runs could have produced duplicate compounds, I removed compounds with

identical canonical SMILES and retained the compound with the best docking score for each unique SMILES.

2.3.4 Performance Benchmarks

To showcase the improved performance of AutoGrow4 compared to its predecessor, I ran both programs using high-performance computational resources over a 24-hour period. Each program was set to produce large population sizes and to utilize many of their features. The goal of these comparisons is to demonstrate that the new features provided in AutoGrow4 improve not just run efficiency, but also the quality of the candidate ligands.

2.3.4.1 Performance Benchmarks: File Preparation

Benchmarks comparing the performance of AutoGrow4 (version 4.0.1) with its predecessor, AutoGrow3 (version 3.1.3), were seeded with a set of 6,103 ZINC15 small molecules with MW between 100 Da and 150 Da ("Chapter 2.3.2: Preparation of Complementary Molecule Libraries.") This same set is provided in the AutoGrow4 download. Because AutoGrow3 requires seed compounds in PDB format, I converted the source compounds from SMILES to PDB using Gypsum-DL[214].

2.3.4.2 Performance Benchmarks: Run Conditions

Because the goal was to compare the best possible compounds that each program produced using their respective optimized settings and resources, many of the settings differed between programs. I controlled for the duration of each run (24 hours), the target protein pocket (the catalytic pocket of 4R6E:A), the seed molecules ("Chapter 2.3.4.1: Performance Benchmarks: File Preparation"), and the selection method (the Ranking selector). Additionally, I controlled for population size so that each generation produced 2,500 compounds via mutation, 2,500 compounds via crossover, and advanced 1,000 compounds via elitism. Both programs applied the Lipinski* filter; however, the AutoGrow4 runs were additionally filtered with the Ghose and PAINS filters, and the AutoGrow3 runs were also filtered with Ghose*. Settings for these AutoGrow3 and AutoGrow4 runs are provided in Appendix JSON 3 and Appendix JSON 4, respectively.

Due to the limited features provided in AutoGrow3, there are several key differences in the settings used in these benchmarks. For instance, the AutoGrow4 runs used the AllRxn reaction library containing 94 reactions, whereas the AutoGrow3 runs were limited to the 58 reactions provided by AutoClickChem[226]. Also, because AutoGrow4 offers new 3D conformational sampling via Gypsum-DL and provides a faster docking option, I set the AutoGrow4 runs to produce a maximum of five variants per compound and used QVina2 docking. In contrast, AutoGrow3 only tested one variant per compound and used Vina docking. The AutoGrow4 runs evaluated all source compounds, producing a zeroth generation; AutoGrow3 does not provide this option and thus began with generation one.

The parallelization method, computational resources, and number of independent runs also differed between the AutoGrow4 and AutoGrow3. Because AutoGrow4 provides MPI-enabled parallelization, I ran the AutoGrow4 experiments on ten MPI-enabled CRC computer nodes with 28-core Broadwell processors and 64GB RAM/node, which were networked with Intel's Omni-Path communication architecture. The AutoGrow4 runs ran for 24 hours and were performed in triplicate. On the other hand, AutoGrow3 is limited to SMP parallelization, so these runs were performed using 28-core processors and 503GB RAM/node provided by the University of Pittsburgh's Center for Research Computing (CRC). In order to control for the number of processors applied to each set of runs, I ran the AutoGrow3 runs 30 times.

2.3.4.3 Performance Benchmarks: Post-Run Data Processing

I pooled the results of each completed generation into a corresponding pool. For instance, including the zeroth generation, two of the three AutoGrow4 runs completed 22 generations, and the third completed 23 generations, so the zeroth through 21st-generation pools consisted of compounds derived from all three runs, but the 22nd generation consisted exclusively of the single run that completed the 22nd generation. Because the parallel runs can produce duplicate compounds, I removed compounds with identical canonical SMILES and retained the compound with the best docking score for each unique SMILES. This process was repeated separately for the compounds produced by the 30 AutoGrow3 runs.

2.3.5 Calculating Normalized Diversity Scores

Throughout this dissertation, I will use chemical diversity (unitless, see Equation 2, p.86) as a metric for assessing (1) how well a *de novo* CADD program searches chemistry space and (2) to what extent a given population has homogenized. Because the diversity score is calculated by summing all pairwise comparisons of all compounds within a population, the score increases with the number (N) of compounds being compared (Equation 2; p.86). To compare populations of different sizes, I will normalize the diversity score by dividing it by N. This unitless "normalized diversity score" ranges between 0.0 and 1.0, where 0.0 indicates that the compound is completely unique relative to the other compounds in the population, and 1.0 indicates that it is identical to all other compounds (i.e., complete homogenization). The mean normalized diversity score for a given population is then calculated from the set of associated normalized diversity scores.

2.4 Results and Discussion

2.4.1 Efficiency Benchmarks

To run efficiency benchmark comparisons of AutoGrow4 and its predecessor AutoGrow3 (version 3.1.3), I ran both programs using the Ranking selection method and AutoDock Vina docking. I ran AutoGrow4 with Gypsum-DL set to produce a single variant per generation. These AutoGrow3/AutoGrow4 settings produced the same number of variants and thus allowed for a roughly equivalent comparison of the two algorithms. AutoGrow4 was on average 1.21 times

faster than AutoGrow3 in completing five generations (59.64 vs 49.34 minutes/run, see Figure 12A).

Since AutoGrow4 has many features not available in previous versions—most notably, offering new docking software and using Gypsum-DL[214] to generate alternate ionization, tautomeric, chiral, cis/trans isomeric, and ring-conformational variants for each input SMILES string—I also ran AutoGrow4 using optimized user-parameters. AutoGrow4 run times vary roughly linearly with the maximum number of variants produced by Gypsum-DL (Figure 12B). A maximum of three variations per molecule averaged 2.50 variants per compound; a maximum of five variations per molecule averaged 3.92 variants per compound.

I also assessed AutoGrow4's efficiency by varying the docking software used (Figure 12C). Docking is the most time-consuming process in both AutoGrow3 and AutoGrow4. QVina2, AutoGrow4's new default docking program[205], is faster than Vina and so further improves AutoGrow efficiency (Figure 12C). AutoGrow4, with a maximum variance per compound of five, is 1.63 times faster when docking with QVina2 (125.04 minutes/run) than with Vina (203.52 minutes/run) (Figure 12B-C).

Several important trends can be observed in the docking scores of these benchmarks. I chose to review the average docking score from each generation, both for the overall population and among the top 500 best-scored compounds. These were chosen to show how the selective pressures imposed by each program influence the entire population as well as the more elite compounds that are more likely to be chosen as lead candidates. The most predictable and noticeable trend is that average docking score improves from each generation, both for the overall population (Table 4) and among the top 500 best-scored compounds (Table 5). I looked at these

two groups to show how the selective pressures imposed by each program influences the entire population as well as the more elite compounds, which are more likely to be chosen as lead candidates. The improved docking score trend holds true independent of docking software, version of AutoGrow, or number of tested structural variants per compound (Table 4 and Table 5). Additionally, as expected, an increase in the maximum number of structural variants tested per compound resulted in an increase in mean docking scores, independent of docking software (Table 4 and Table 5). Lastly, all AutoGrow4 run conditions had higher average docking scores, both globally and among the top 500 best-scored compounds, than the corresponding generation produced by AutoGrow3. This last trend is examined further in the following section ("Chapter 2.4.2: Performance Benchmarks.")

Overall, AutoGrow4 is more computationally efficient, has more extensive user options, and creates compounds with better predicted binding affinities than AutoGrow3. This enables more extensive CADD searches that, when combined with expanded ADME-PK filters ("Chapter 2.2.3: Chemical Filters"), produced better predicted compounds.

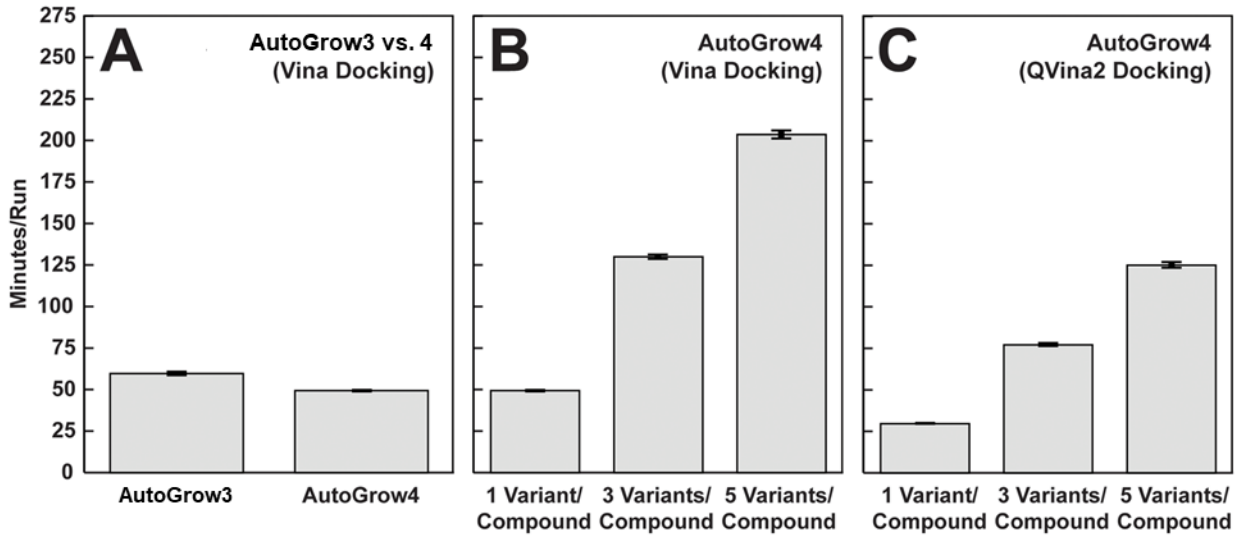


Figure 12. Benchmark results from 24 comparative runs of AutoGrow.
All results presented above had sample sizes of 24 runs. Bar height represents the mean time to complete a five-generation AutoGrow run, and error bars represent the standard error. A) Running AutoGrow 3.1.3 and AutoGrow4 with comparable user options (see Methods "Section 2.3.3: Efficiency Benchmarks"). B) Running AutoGrow4 using Vina docking with one, three, and five maximum variants per input compound. C) Running AutoGrow4 using QVina2 docking with one, three, and five maximum variants per input compound. Figure is modified from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. This figure is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License which "allows unrestricted use, distribution, and reproduction in any medium."[6]

Table 4. Global docking score averages of the efficiency benchmarks.

"Maximum Variants per Mol." indicates the maximum number of structural variants that were tested per compound. "Gen. #" indicates the AutoGrow-run generation number. All runs of each respective program were pooled together; "Mean" represents the mean of the pooled compounds. Means are in units of kcal/mol as measured by either Vina or QVina2. Gray triangles indicate that the mean binding affinity tends to increase as the number of considered variants increase, and as generations progress.

Maximum	AutoGrow3	AutoGrow4			AutoGrow4		
Variants per Mol.	1	1	3	5	1	3	5
Docking Program	Vina	Vina	Vina	Vina	QVina2	QVina2	QVina2
Gen. 1	-8.42	-8.82	-8.90	-8.93	-8.79	-8.91	-8.93
Gen. 2	-8.67	-9.01	-9.11	-9.15	-9.04	-9.17	-9.22
Gen. 3	-9.26	-9.66	-9.74	-9.82	-9.72	-9.86	-9.97
Gen. 4	-9.76	-10.17	-10.29	-10.40	-10.29	-10.45	-10.53
Gen. 5	-10.29	-10.62	-10.77	-10.88	-10.70	-10.95	-10.99

Table 5. Average docking score of the best 500 compounds from the efficiency benchmarks. "Maximum Variants per Mol." indicates the maximum number of structural variants that were tested per compound. "Gen. #" indicates the AutoGrow4-run generation number. All runs of each respective program were pooled together; "Mean" represents the mean of the 500 best-scored compounds. Means are in units of kcal/mol as measured by either Vina or QVina2. Gray triangles indicate that the mean binding affinity tends to increase as the number of considered variants increase, and as generations progress.

Maximum Variants per Mol.	AutoGrow3	AutoGrow4			AutoGrow4		
	1	1	3	5	1	3	5
Docking Program	Vina	Vina	Vina	Vina	QVina2	QVina2	QVina2
Gen. 1	-9.00	-9.95	-10.00	-10.05	-9.89	-10.07	-10.08
Gen. 2	-9.93	-10.61	-10.70	-10.82	-10.67	-10.82	-10.86
Gen. 3	-10.66	-11.09	-11.22	-11.32	-11.18	-11.37	-11.39
Gen. 4	-11.29	-11.47	-11.64	-11.78	-11.55	-11.80	-11.81
Gen. 5	-11.77	-11.82	-12.02	-12.14	-11.87	-12.15	-12.20

2.4.2 Performance Benchmarks

In this section I compare the performances of AutoGrow4 and AutoGrow3 when run for 24 hours. Whereas in the previous benchmarks I suppressed AutoGrow4 features that were not available in AutoGrow3 to allow for a fair comparison, here I enable several new AutoGrow4 features in order to provide a thorough comparison in terms of predicted binding affinity and chemical diversity.

2.4.2.1 Performance Benchmarks: Predicted Binding Affinity

Because both AutoGrow3 and AutoGrow4 use predicted binding affinity as their primary fitness metric[6,156], I begin by comparing the performance of both in terms of that metric. I will compare each program's best-scored populations (1) relative to each generation and (2) within the entire 24-hour run. The latter comparison will be heavily influenced by the number of generations each program completes.

In the allotted 24 hours, AutoGrow3 completed an average of 4.17 generations per run, ranging from one to five generations (Figure 13). In contrast, AutoGrow4 completed an average of 21.33 generations per run, beyond the zeroth generation (Figure 13).

I focus the first part of this analysis on the performance of the first five generations. This provides a generation by generation comparison of the two programs. By the end of the fifth generation, the two best AutoGrow3 compounds both had a docking score of -13.5 kcal/mol. By way of comparison, the AutoGrow4 runs had already surpassed that score by the second generation

(-13.6 kcal/mol, Figure 13). By the fifth generation, AutoGrow4 had produced 103 compounds with docking scores greater than or equal to -13.6 kcal/mol; the best-scored compound had a predicted binding affinity of -14.8 kcal/mol (Figure 13). For all generations, AutoGrow4 had better average docking scores than AutoGrow3. The same was true when the means of the top 1, 10, 50, and 1,000 compounds were considered (Figure 13). This superior performance is even more impressive given that AutoGrow3 was run 30 times and so sampled more compounds per generation, whereas AutoGrow4 was run only three times. For instance, the AutoGrow3 fourth generation had a total population of 134,747 unique compounds, whereas the AutoGrow4 fourth generation had a population roughly one tenth the size (14,409 unique compounds).

In terms of ability to produce well-scored compounds over the allotted 24 hours, AutoGrow4 again performed substantially better than AutoGrow3 (Figure 13). AutoGrow4's MPI-enabled parallelization allowed me to use ten times the number of CPUs per run, so AutoGrow4 completed many more generations than AutoGrow3 (Figure 13). This translated to a 3.0 kcal/mol difference between the best-scored AutoGrow4 compound (-16.5 kcal/mol, generation 21) and the best-scored AutoGrow3 compound (-13.5 kcal/mol, generation five) (Figure 13).

To explore the factors that contributed to AutoGrow4's superior performance, I first considered the hypothesis that AutoGrow4 benefits by docking and scoring the source compounds to form an already evaluated zeroth generation. This feature is not available in AutoGrow3, so AutoGrow3 runs do not evaluate the source compounds prior to the first generation. If this zeroth generation was solely responsible for AutoGrow4's improved performance, I would expect the average docking scores of AutoGrow4's n - 1 generation to be closer to those of AutoGrow3's n^{th} generation. However, with a few exceptions (Figure 13), this was generally not the case. For

instance, AutoGrow4's third and fourth generations had average docking scores for the entire population and for the top 1, 10, 50, and 1,000 compounds that were better than those of AutoGrow3's fourth and fifth generations, respectively (Figure 13). In fact, the average docking score of AutoGrow4's third generation (-9.85 kcal/mol) was nearly 0.56 kcal/mol better than AutoGrow3's fourth generation (-9.29 kcal/mol) (Figure 13). The improved performance of AutoGrow4 thus cannot be fully explained by its assessment of the zeroth generation.

I next tested the hypothesis that the docking software used might account for some of the differences in performance. QVina2 may be posing each compound more accurately and/or assigning better scores generally. To test this hypothesis, I reevaluated, in triplicate, the AutoGrow3 seed population using Gypsum-DL and the same docking parameters used in the AutoGrow4 runs (Table 6). The results of this reassessment clearly indicate that although QVina2 improves AutoGrow4's speed (Figure 12; p.99), there is not a noticeable difference in docking scores (Table 6). In fact, an unpaired *t*-test between the entire populations led me to accept the null hypothesis that there are not significant differences between using Vina and QVina2 (all tests had p-values ≥ 0.35). Furthermore, both Vina and QVina2 ranked the same compound in first place, with an identical score of -7.9 kcal/mol (Table 6).

I next considered the possibility that AutoGrow4's more thorough testing of multiple variants per compound (i.e., creating multiple variants via Gypsum-DL and docking each variant) improved its performance over than of AutoGrow3, which does not use Gypsum-DL. As the efficiency benchmark experiments show ("Chapter 2.4.1: Efficiency Benchmarks"), AutoGrow4 averages better docking scores when the maximum number of variants produced by Gypsum-DL is increased (Table 4 and Table 5; p.100-101). Because AutoGrow3 only samples a single

structural variant per compound, it is less likely to predict an ideal docking pose and thus will on average perform worse than AutoGrow4 in terms of docking scores. Testing multiple compound variants improves AutoGrow4's chances of identifying better scoring ligand poses for two reasons. First, the number of attempts at docking each compound is increased. Pose sampling algorithms, such as the ones used by Vina and QVina2, tend to use stochastic optimization-based algorithms[185,205]. Thus, the chance of finding a better scoring pose increases with repeated docking. Second, some protein-ligand interactions and poses are possible in one variant but not in others. For instance, depending on the protonation of the protein and ligand, an electrostatic interaction may or may not be identified. Sampling multiple compound variants increases the chance of identifying these interactions and so improves the average docking scores.

I next considered whether the different chemical filters applied to the runs influenced the docking scores. The AutoGrow4 runs applied a more restrictive Ghose filter than the Ghose* filter applied by AutoGrow3. AutoGrow4 also applied the PAINS filter, which is not available in AutoGrow3 ("Chapter 2.2.3: Chemical Filters")[6,156]. However, although the additional filter constraints imposed by AutoGrow4 should yield more drug-like compounds, it is unlikely that they improve docking scores. In fact, the more restrictive MW constraints of the Ghose filter used in the AutoGrow4 runs is likely to exclude higher MW compounds, which are more likely to have better docking scores[184]. However, despite more restrictive property filters, AutoGrow4 still performed better than AutoGrow3.

Lastly, I considered the possibility that docking scores were influenced by (1) AutoGrow4's expanded compound-production methods (e.g., the expanded mutation-reaction set) and (2) AutoGrow4's incorporation of chemical diversity into the selection process. Because

AutoGrow's compound-generation processes are intrinsically stochastic, it is not possible to perform a controlled comparison of compound generation. But, I would expect that the increased number of mutation reactions, as well as the larger complimentary molecule libraries, would improve the diversity of the generated compounds. I provide an additional analysis of chemical diversity in the following section ("Chapter 2.4.2.2: Chemical Diversity").

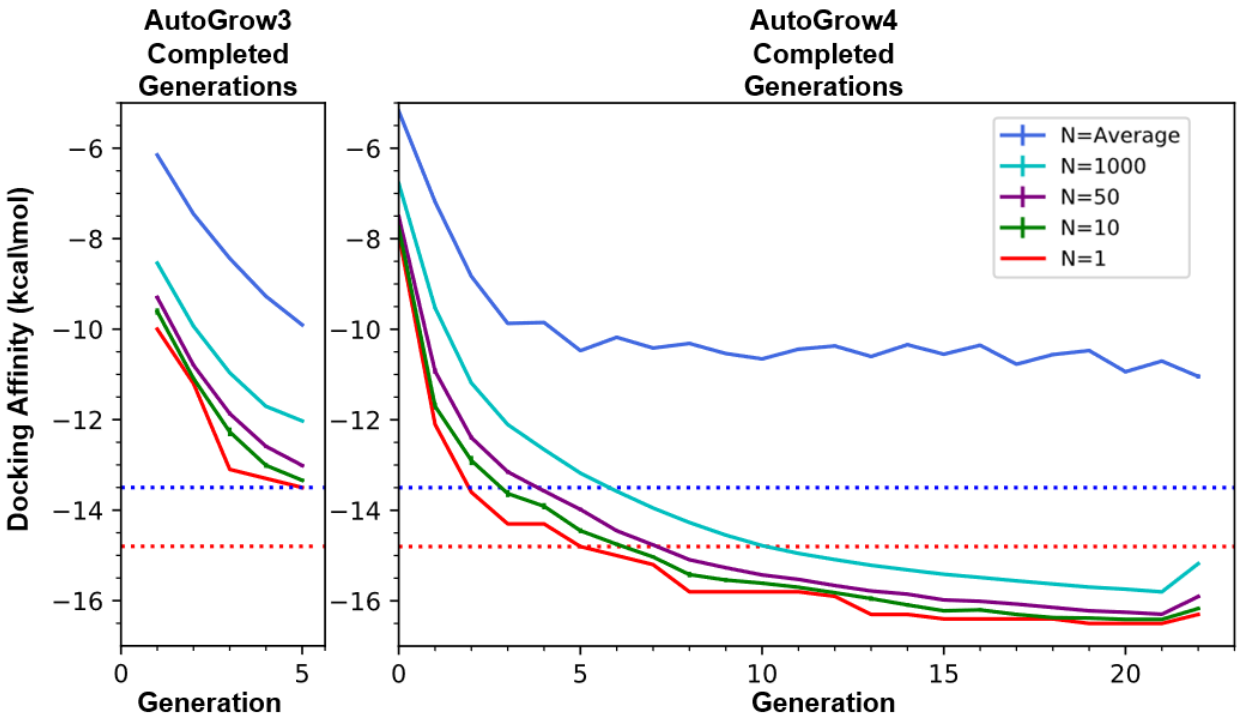


Figure 13. Plots of the performance benchmarks comparing docking scores.
In the allotted 24 hours, the AutoGrow3 runs completed between one and five generations per run (left), while the AutoGrow4 runs completed between 21 and 22 generations per run (right). Additionally, AutoGrow4 evaluated the zeroth generation (right). For each generation, the mean of all docking scores is shown in purple. The grand means of the top 1,000, 50, 10, and 1 compound are shown in blue, cyan, purple, green, and red, respectively. Error bars represent the standard error within each generation. For the reader's convenience, the docking scores of the best-scored compound from the fifth generation of AutoGrow3 (-13.5 kcal/mol; blue dotted line) and AutoGrow4 (-14.8 kcal/mol; red dotted line) are also shown.

Table 6. Zeroth generation reassessment with Vina and QVina2.

"Global" indicates that all unique compounds were included in the respective analyses; "N" indicates the number of the top-scoring compounds included in the respective analyses. "Mean" represents the mean of the subset; "Dev." represents the standard deviation of the included scores. Each assessment of the zeroth generation was performed in triplicate. All values are in units of kcal/mol, as predicted by the corresponding docking program.

Docking	Global			N=1	N=10		N=50		N=1000	
Program	Mean	Dev.	Size	Mean	Mean	Dev.	Mean	Dev.	Mean	Dev.
QVINA2	-5.23	0.72	9091	-7.9	-7.62	0.14	-7.35	0.17	-6.52	0.32
VINA	-5.24	0.72	8731	-7.9	-7.65	0.14	-7.36	0.18	-6.51	0.32

2.4.2.2 Chemical Diversity

AutoGrow4 introduced the option to select compounds based on their chemical diversity relative to other compounds in their respective generations. This feature helps maintain diversity within the population, thereby allowing AutoGrow4 to search a wider range of chemistry space. In this section I compare AutoGrow4 and AutoGrow3's ability to maintain chemical diversity over a 24-hour run.

Diversity scores are determined by performing pairwise comparisons of all compounds, so the required computational resources scale at a rate of N^2, where N is the number of compounds. Each AutoGrow3 generation (pooled across all 30 runs) included more than 100,000 compounds, requiring more than 10^{10} comparisons per generation ("Chapter 2.3.5: Calculating Normalized Diversity Scores"). Calculating global diversity scores was thus computationally intractable. Additionally, increasing the number of runs increases the chance of exploring unique regions of chemistry space and thereby increases a pooled population's diversity. This creates an artificial bias in favor of AutoGrow3, which pooled from ten times the number of runs as AutoGrow4. To compensate, I provide analysis of a pooled subset containing the three AutoGrow3 runs that produced the three best-scored compounds (two with docking scores of -13.5 kcal/mol and one with -13.4 kcal/mol) by the fifth generation (Table 7).

AutoGrow4 maintained the global normalized diversity of its generations within a range of 0.15 and 0.22 (unitless) (Table 7). In fact, the zeroth and first generations had the worst global diversity scores (0.20 and 0.22 respectively), and the diversity scores tended to improve in subsequent generations (Table 7). This general trend towards improved chemical diversity likely

results from AutoGrow4's new secondary fitness metric (i.e., the structural diversity score) ("Chapter 2.2.7.2: Secondary Fitness Metric: Structural Diversity"). AutoGrow3 generations, on the other hand, follow the opposite trend. They consistently become less diverse as the runs progress, for all tested population subsets (Table 7). Of course, the AutoGrow3 runs completed fewer generations in the allotted 24 hours than the AutoGrow4 runs; this limits my discussion to only shorter AutoGrow3 runs, though I suspect that the AutoGrow3 runs would continue to homogenize in later generations.

Despite these overall trends (i.e., AutoGrow3 tends to homogenize, and AutoGrow4 tends to maintain diversity), AutoGrow4 does lose diversity among its best-scoring sets of compounds (i.e., the best-scored 50 and 1,000, compounds) (Table 7). The average normalized diversity scores of AutoGrow4's best-scoring 50 and 1,000 compounds improves from the zeroth generation to the second generation, but worsens from the second generation onward (Table 7). However, by the tenth generation the diversity scores appear to stabilize (Table 7). The last AutoGrow4 generation, generation 22, was an outlier with dramatically worse average diversity scores. This result is likely artifactual because only a single run completed that generation, such that the diverse compounds of the other two runs were lost (Table 7).

This experiment demonstrates that AutoGrow4's new chemical-diversity fitness metric maintains structural diversity even in long runs. On the other hand, with no selective pressure for chemical diversity, AutoGrow3 tends to lose diversity in each successive generation. AutoGrow3 is unlikely to find a diversity equilibrium, as observed in the AutoGrow4 runs; instead, it tends to converge to a local minimum.

Table 7. Diversity scores of the three best AutoGrow3 runs.

"Gen. #" indicates the AutoGrow generation number. The AutoGrow3 generations are the pooled compounds of the AutoGrow3 runs that produced the three best-scored compounds, whereas the AutoGrow4 runs are the pooled compounds of all three AutoGrow4 runs. "Global" indicates that all compounds are included. "N" indicates the number of the top-scoring compounds selected. "Mean" represents the mean of the subset. For each subset of N compounds, the diversity score of each compound was assigned using Equation 2, and then normalized by dividing N ("Chapter 2.3.5: Calculating Normalized Diversity Scores"). The reported means and standard deviations are based on these normalized diversity scores (unitless). "Dev." represents the standard deviation of the included scores. "% Pop." represents the percent of scores that were included (100 * N / Size).

Program	Global			N=50			N=1000		
	Mean	Dev.	Size	Mean	Dev.	% Pop.	Mean	Dev.	% Pop.
AutoGrow3									
Gen. 1	0.21	0.03	13535	0.25	0.04	0.37	0.25	0.05	7.39
Gen. 2	0.24	0.04	14270	0.25	0.03	0.35	0.25	0.03	7.01
Gen. 3	0.24	0.03	14352	0.27	0.03	0.35	0.26	0.03	6.97
Gen. 4	0.25	0.03	14190	0.29	0.03	0.35	0.27	0.03	7.05
Gen. 5	0.26	0.03	13930	0.30	0.04	0.36	0.28	0.03	7.18
AutoGrow4									
Gen. 0	0.20	0.04	9091	0.36	0.07	0.55	0.27	0.07	11.00
Gen. 1	0.22	0.04	16234	0.28	0.03	0.31	0.25	0.03	6.16
Gen. 2	0.19	0.02	15558	0.25	0.02	0.32	0.23	0.02	6.43
Gen. 3	0.18	0.03	14585	0.26	0.02	0.34	0.24	0.02	6.86
Gen. 4	0.17	0.02	13810	0.28	0.04	0.36	0.25	0.02	7.24
Gen. 5	0.17	0.03	14514	0.29	0.04	0.34	0.25	0.02	6.89
Gen. 10	0.16	0.02	12149	0.35	0.04	0.41	0.30	0.04	8.23
Gen. 15	0.16	0.03	13607	0.33	0.03	0.37	0.31	0.03	7.35
Gen. 20	0.15	0.03	11983	0.36	0.05	0.42	0.30	0.02	8.35
Gen. 21	0.16	0.03	12811	0.37	0.05	0.39	0.31	0.02	7.81
Gen. 22	0.19	0.05	4322	0.55	0.05	1.16	0.44	0.08	23.14

2.5 Conclusion

2.5.1 AutoGrow3 and AutoGrow4 Benchmarks

AutoGrow4 is more efficient than its predecessor, as these extensive benchmark experiments show. The efficiency benchmarks show that AutoGrow4 is 1.21 times faster than AutoGrow3 when controlling for all program features, and even faster if using the upgraded docking features (Figure 12; p.99). The performance benchmarks further verified the improved speed of AutoGrow4. AutoGrow4 completed 22.33 generations per run on average, including the zeroth generation, over 24 hours of run time, whereas AutoGrow3 completed only 4.17 generations per run on average (Figure 13; p.107). AutoGrow4 completed more generations than AutoGrow3 because it is more computationally efficient and because of its MPI-enable parallelization, which allows users to run AutoGrow4 on many more CPU's than AutoGrow3.

AutoGrow4 also outperformed AutoGrow3 at predicting high-affinity compounds. In both benchmark experiments, AutoGrow4 generated compounds with better docking scores than AutoGrow3. I showed that the biggest contributing factors include: (1) AutoGrow4 allows testing multiple structural variants per compound, whereas AutoGrow3 is limited to a single structural variant per compound; and (2) AutoGrow4 assesses the zeroth generation and uses it to inform the creation of the first generation, thereby more intelligently designing first-generation compounds. The number of tested structural variants does in fact correlate with the population docking scores (Table 4; p.100). Additionally, AutoGrow3 generates its first generation without assessing the zeroth generation, which limits the first generation's scoring (Figure 13). I also show that the

difference in docking software does not contribute to the difference of each program's docking scores (Table 6 p.108). Lastly, I show that AutoGrow4's new diversity selection feature helps AutoGrow4 maintain chemical diversity across many generations, whereas AutoGrow3 lost diversity each successive generation (Table 7; p.111). Although I cannot directly tie these diversity and docking scores together, conventional wisdom suggests that diversity is beneficial to GA[151].

Ultimately, AutoGrow4 is better equipped to predict *de novo* candidate inhibitors. It was superior to AutoGrow3 in producing well-scoring compounds and maintaining chemical diversity, as well as also being more computationally efficient. As shown in the performance benchmarks, the improved computational performance and expanded features allowed for a much larger search of chemistry space, ultimately found candidates with stronger predicted binding affinities. Additionally, AutoGrow4 provides many user features and customizations that improve ease-of-use and make AutoGrow4 applicable to a wide range of CADD needs.

2.5.2 Summary of Improvements

AutoGrow4's stochastic search of chemistry space using a GA is a powerful technique for discovering potential compounds. The upgrades I made to AutoGrow4 make it a computationally efficient tool for *de novo* CADD. Intentional design choices enable long-term maintainability, and user customizability. As a free, open-source program with extensive user options, AutoGrow4 is an easy tool to incorporate into drug-discovery workflows.

A thorough tutorial is provided in the download, detailing installation and instructions. AutoGrow4 runs on Linux, macOS, and Windows (via a Docker container, Docker, Inc.). I

recommend using the AutoGrow4 Docker container included with the download, which automatically installs all dependencies.

2.5.3 Future Directions

The most important future direction of AutoGrow4 is its application to different protein targets. Since its public release in April 2020, Dr. Durrant and I have received numerous positive responses indicating that we were not alone in our desire for a more efficient, intuitive-to-use *de novo* CADD option. In the next chapter ("Chapter 3.0: AutoGrow4: Application to Poly (ADP-ribose) Polymerase-1 (PARP-1)"), I will apply AutoGrow4 to PARP-1 to target multiple pockets and design candidate PARPi that may be less susceptible to known resistance mechanisms.

Future development will also need to focus on expanding user features, further reducing computational costs, and providing more analysis tools. A graphical user interface or webserver implementation would improve ease-of-use, particularly for users who are not comfortable with command-line interfaces. Expanding the multiprocessing support to GPU multithreading and continuing to expand the reaction libraries and chemical filters are other possible avenues of improvement. Lastly, incorporating ensemble-style docking is another possible future avenue.

2.6 Acknowledgements

I would like to thank the following people for their contributions to the AutoGrow4 codebase and benchmark experiments. Dr. Jacob Durrant generated the original idea for revising

his line of AutoGrow software. He provided guidance in both code, experimental design, co-authoring the AutoGrow4 publication, editing this chapter of the dissertation, testing AutoGrow4 on macOS, and generating high-resolution versions of figures for the publication and dissertation.

I would like to thank Dr. Bennett Van Houten for his suggestion that I apply AutoGrow4 to PARP-1.

Additional contributions to AutoGrow4 include: Patrick J. Ropp for useful discussions and programming tips; Erich Hellemann for discussions and help with SMARTS reactions; Pauline Spiegel for manuscript editing and tutorial testing; Yuri Kochnev for compiling QVina2 for macOS; Kevin C. Cassidy for help with molecular rendering; and Harrison Green for help developing AutoGrow4 accessory scripts. I would also like to thank the University of Pittsburgh's Center for Research Computing for providing helpful computer resources. The default fragment libraries included with AutoGrow4 were derived from a subset of the ZINC database (https://zinc.docking.org/). I thank ZINC for allowing me to distribute these fragment libraries to AutoGrow4 users.

2.7 Author Contributions

Jacob O. Spiegel wrote and tested the codebase of AutoGrow4 and all accessory scripts provided in the AutoGrow4 download. AutoGrow4 leverages many libraries and other programs which have been cited throughout the code and all related publications. I designed and performed all experiments for AutoGrow4 and AutoGrow3 comparisons. Dr. Jacob Durrant and I authored

the publication of AutoGrow4. I designed the initial layout for all figures in the paper with Dr. Durrant. Dr. Durrant refined and generated many of the high-quality images used in the publication and throughout this dissertation. Only sections that I wrote were used verbatim in this dissertation. All writing in this chapter is original content written by Jacob O. Spiegel.

3.0 AutoGrow4: Application to Poly (ADP-ribose) Polymerase-1 (PARP-1)

This chapter details my application of AutoGrow4 to the protein target poly (ADP-ribose) polymerase-1 (PARP-1) and expands on work published in the AutoGrow4 manuscript. It will explore how AutoGrow4 can be used as a tool for lead generation, lead optimization, hypothesis generation, and hypothesis testing.

AutoGrow4 was published under the Creative Commons Attribution 4.0 International License which "allows unrestricted use, distribution, and reproduction in any medium"[6]. The work in this chapter is adapted and reprinted with rights and permission:

> **Jacob O Spiegel**†, & Jacob D Durrant. (2020) AutoGrow4: An open-source genetic algorithm for *de novo* drug design and lead optimization. *Journal of Cheminformatics,* 12, 25. https://doi.org/10.1186/s13321-020-00429-4.
>
> † Jacob O. Spiegel should be regarded as first author.

I am the first author of the published manuscript, having written the entire AutoGrow4 codebase, performed all AutoGrow4 runs discussed in the paper, and analyzed the data. I designed the initial layout for all figures in the paper with Dr. Jacob Durrant. Dr. Durrant refined and generated many of the high-quality images used in the publication and throughout this dissertation. Dr. Durrant also provided guidance and insight as described in the acknowledgement section. All writing in this chapter is original content written by Jacob O. Spiegel.

3.1 Overview and Rationale

Poly (ADP-ribose) polymerase-1 (PARP-1), a well-characterized DNA-damage-recognition protein ("Section 1.1: Biological Background"), was chosen to showcase AutoGrow4's *de novo* design and lead optimization capabilities. I chose to target PARP-1 because (1) its multiple biologically confirmed inhibitors (PARPi) serve as positive controls and as source compounds for lead optimization runs[1–5]; (2) PARPi are proven treatments for multiple cancers, including ovarian and breast cancers[1–5]; and (3) PARP-1's catalytic domain has a well-studied druggable pocket[17–19]. However, all current PARPi bind to the catalytic pocket and are susceptible to many of the same resistance mechanisms. Given this vulnerability, there is a need for next-generation PARPi that can evade these resistance mechanisms. Presented in this chapter are four sets of AutoGrow4 runs that target PARP-1: a single large-scale *de novo* design run applied to the PARP-1 catalytic site, PARPi lead-optimization runs applied to the PARP-1 catalytic site, PARPi lead-optimization runs applied to a post-translationally modified PARP-1 catalytic site, and large-scale *de novo* design runs applied to the PARP-1 DNA binding domain (DBD). These four AutoGrow4 applications aim to find novel candidates for next-generation PARP-1 inhibition, particularly ones less susceptible to known PARPi resistance mechanisms.

Broadly speaking, *de novo* design experiments are seeded with random small molecules/fragments and so are not biased by prior knowledge of the binding pocket or known inhibitors. In contrast, lead-optimization experiments refine preexisting ligands. By focusing the search of chemistry space (i.e., the coordinate set containing all possible compounds[178]) on likely

leads, lead-optimization approaches spend fewer computational resources testing compounds that are unlikely to be successful drugs.

I designed these AutoGrow4 runs to test different applications of the algorithm. Three sets of runs targeted the PARP-1 catalytic domain. The large-scale *de novo* run was designed to demonstrate how AutoGrow4 can create novel drug-like compounds optimized for predicted binding affinity. I also wanted to highlight AutoGrow4's scalability, which allows it to process hundreds of thousands of compounds in a single run. The initial PARPi lead-optimization runs were designed to demonstrate AutoGrow4's ability to create novel compounds that are similar to experimentally verified PARPi but with stronger predicted binding affinities. Additionally, another set of PARPi lead-optimization runs was designed to show how AutoGrow4 results can guide hypothesis generation. I applied this second set of lead-optimization runs to a post-translationally modified PARP-1 to find compounds that bind PARP-1 in both the modified and non-modified states, thereby circumventing resistance mechanisms.

Lastly, I applied AutoGrow4 to an additional PARP-1 surface pocket, the DBD. There are currently no clinically successful non-competitive inhibitors of PARP-1[12,17,18,111,224] that bind outside the catalytic pocket, but such PARPi could mitigate several forms of PARPi resistance, such as resistance mechanisms that alter the catalytic binding site[142]. Speculatively, such an inhibitor could be used as a cotreatment with competitive inhibitors of PARP-1. I used the computational hot-spot mapping technique FTMap[243], a computational technique that identifies regions or "hot spots" with favorable ligand binding properties[243], to identify druggable pockets (i.e., a protein pocket that has known ligands or is predicted to bind small molecules with high affinity[244]). I then compared the physiochemical properties of the pockets and previous mutagenic

studies of PARP-1. I ultimately selected the most druggable non-CAT hot spot, which is located near the interface of zinc finger I (Zn1) and zinc finger 3 (Zn3) in the DBD. Mutations of residues within the Zn1-Zn3 interface prevented PARP-1 DNA-dependent catalytic activity without altering the DNA binding activity[45,46,52,70]; so, ideally, small molecules that bind this site would have similar affects. Finally, I used AutoGrow4 to predict candidate inhibitors that bind at this interface. My results provide a preliminary exploration of candidate PARPi that do not rely on binding the catalytic site.

3.2 Methods

3.2.1 Large-Scale *de novo* Run

To design novel predicted PARPi that bind to the catalytic site, I ran a large AutoGrow4-guided lead-generation run. The goal of this run was to generate drug-like compounds with strong predicted binding affinities for the catalytic pocket.

3.2.1.1 File Preparation

The protein structure used for the AutoGrow4 large-scale *de novo* run is derived from PDB 4R6E:A. Details regarding the preparation of the receptor and the determination of docking coordinates can be found in "Section 2.3.1: Receptor File Preparation."

3.2.1.2 Preparation of Example Source Compounds

Large-scale runs require large, diverse source populations to effectively search a wide range of chemistry space. Seeding a run with too small or too undiversified a source population can result in homogenization and premature convergence. AutoGrow4 provides four sample seed lists, organized by molecular weight (MW), for general-purpose use.

Starting from the same large set of commercially available Lipinski*-filtered Zinc15[230] compounds that are described in "Section 2.3.2: Preparation of Complementary Molecule Libraries," I generated four sizable and chemically diverse general-purpose seed libraries. First, to ensure that all compounds could react in at least one of the 94 reactions in the AutoGrow4 default reaction set (AllRxn), I separated the compounds by functional group and discarded any that did not contain a functional group used by the AllRxn set. I then separated each functional group set into four subgroups based on MW (≤ 100 Da, 100 Da < MW≤ 150 Da, 150Da < MW ≤ 200 Da, and 200 Da < MW ≤ 250 Da). To maintain both chemical diversity and reasonable size, I randomly selected 100 compounds from overrepresented subgroups (>100 compounds) and discarded the remaining compounds. I pooled all compounds from all functional groups in each MW range to form the final source compound libraries. The final four source-library sets (one for each MW range) together include 24,595 unique molecules and are provided in the AutoGrow4 download.

3.2.1.3 Large-Scale *de novo* Run

To demonstrate that AutoGrow4 is scalable across hundreds of processors and to generate novel drug-like compounds with the strongest possible predicted binding affinities, I performed a large-scale *de novo* run seeded with small molecules from the 100 < MW ≤ 150 Da source-compound library, described in "Section 3.2.1.2: Preparation of Example Source Compounds." I performed this AutoGrow4 run in a Python 3.7 environment using MPI multiprocessing, QVina2 docking, a Ranking selector, and the Ghose, Lipinski*, and PAINS filters (Table 1 p.53). It ran for 30 generations. The exact settings are provided in Appendix JSON 5.

This experiment ran on ten MPI-enabled CRC computer nodes with 28-core Broadwell processors and 64GB RAM/node, which were networked with Intel's Omni-Path communication architecture. Blendmol[245] was used to generate the figures of the docked molecules.

3.2.2 PARPi Lead Optimization

To design novel PARPi with improved predicted binding affinities that are nevertheless similar to known PARPi, I ran three AutoGrow4-guided lead-optimization runs.

3.2.2.1 File Preparation

The protein structure used for the AutoGrow4 PARPi lead-optimization runs is the 4R6E:A. Details regarding the preparation of the receptor and the determination of docking coordinates can be found in "Section 2.3.1: Receptor File Preparation."

3.2.2.2 Preparation of PARP inhibitor (PARPi) Source Library

To demonstrate AutoGrow4 lead optimization, I seeded six runs with lead compounds (PARPi). I chose 11 PARPi at varying phases of clinical trials from www.clinicaltrials.gov. I fragmented these 11 PARPi using RDKit's Breaking of Retrosynthetically Interesting Chemical Substructures (BRICS) function, which decomposes molecules along retro-synthesizable bonds[158]. The accessory script for fragmenting these compounds is included in the AutoGrow4 download. This decomposition resulted in 11 original PARPi and 83 fragment molecules, which serve as seeds. This seed list is also included in the AutoGrow4 download.

3.2.2.3 PARPi Lead-Optimization Runs

To demonstrate AutoGrow4's ability to optimize known inhibitors, I seeded AutoGrow4 with a population of known PARPi to bias the search of chemistry space toward likely leads, thus sparing computational time that would otherwise be used exploring leads that are not PARPi-like. Because I also wanted to demonstrate AutoGrow4's ability to grow molecules from PARPi fragments, I chose to seed these runs with a combination of PARPi and PARPi fragments, as described in "Section 3.2.2.2: Preparation of PARP inhibitor (PARPi) Source Library."

To focus the search near these PARPi, I limited the number of generations to five but increased the docking-exhaustiveness, variants-per-molecule, and population-size settings. Due to the relatively small source compound library of 94 seed molecules, the first generation's size was set to be smaller than all subsequent generations. I limited the first generation to 500, 500, and 40

molecules derived using the mutation, crossover, and elitism operators, respectively. All subsequent generations had population sizes of 2,500, 2,500, and 250 compounds created by the mutation, crossover, and elitism operators, respectively. Exact settings are provided in Appendix JSON 6.

These experiments were repeated six independent times in a Python 3.7 environment on the same CRC MPI-enable nodes described in "Section 3.2.1.3: Large-Scale *de novo* Run." Blendmol[245] was used to generate the figures of the docked molecules.

3.2.3 Interaction Assessment of AutoGrow4-Generated Compounds

To determine the most common protein-ligand interactions critical for high-affinity predicted binding to the PARP-1 catalytic pocket, I analyzed the top-scoring compounds from both the large-scale *de novo* and PARPi lead-optimization runs. I used the BINANA algorithm (version 1.1.2), which detects interactions between a docked ligand and a receptor. These interactions include π-π, T-stacking, cation-π, hydrogen-bond, and electrostatic interactions[246]. Because Vina and QVina2 output multiple poses per docked compound, I limited my analysis to the single pose with the best predicted binding affinity per docked compound.

3.2.4 Phosphorylated Y907 PARP-1 Lead-Optimization Runs

The upregulation of receptor tyrosine kinase c-Met causes PARPi resistance by phosphorylating the PARP-1 residue Y907 (pY907)[142]. Because many PARPi form π-π stacking interactions with Y907[17–19,106–112], pY907 confers resistance to multiple PARPi[142]. To identify

potential PARPi that are effective independent of pY907, I modeled pY907 into the 4R6E:A PARP-1 catalytic domain and used AutoGrow4 to design novel candidate inhibitors that bind this modified PARP-1 pocket.

3.2.4.1 Building the pY907 PARP-1 Structure and Orienting the Phosphate Moiety

Because there are no high-resolution structures of PARP-1 with phosphorylated Y907 (pY907), I built a PARP-1 pY907 model using the PyMol plugin PyTMs (version 1.2)[247]. Beginning with the PDB 4R6E:A structure detailed in "Section 2.3.1: Receptor File Preparation," I created one pY907 model using PyTMs's default settings and another using PyTMs's "*Optimization*" option, which adjusts the dihedral angle of the phosphate moiety to avoid steric van der Waals clashes[247]. Because PyTMs also alters the protonation state of the protein, I manually substituted the atoms of the phosphate group into the 4R6E:A structure, which required replacing the hydrogen atom attached to the Y907 side-group oxygen atom with the phosphate group atoms and renumbering the protein's atom indices.

I then docked three known PARPi into the catalytic pockets of the two pY907 PARP-1 structures (pY907-PARP-1) and the original 4R6E:A. Docking was performed with QVina2 using the same docking coordinates as in the PARPi lead-optimization runs (Appendix JSON 6), with QVina2's *exhaustivity* parameter set to 25.

3.2.4.2 pY907-PARP-1 Lead-Optimization Runs

The pY907-PARP-1 lead-optimization runs were performed using the same parameters as the PARPi lead-optimization runs ("Section 3.2.2: PARPi Lead Optimization" and Appendix JSON 6), but docked into the pY907-PARP-1 structure ("Section 3.3.4.1: Building the pY907 PARP-1 Structure and Orienting the Phosphate Moiety").

These experiments were repeated six independent times in a Python 3.7 environment on the same CRC MPI-enable nodes described in "Section 3.2.1.3: Large-Scale *de novo* Run."

3.2.5 AutoGrow4 Applied to a Non-CAT PARP-1 Pocket

To design novel predicted PARPi that do not bind the catalytic site, I (1) identified candidate binding pockets using computational hot-spot mapping; (2) selected a pocket guided by druggability analysis and known mutations described in the literature that inhibit PARP-1 catalytic activity but do not affect PARP-1 DNA binding activity; (3) generated lead compounds that bind the selected pocket using three AutoGrow4 runs; and (4) analyzed the chemical-properties of the compounds with the best docking scores to assess properties that may influence future drug design targeting the new pocket.

3.2.5.1 Computational Hot-Spot Mapping, Druggability Analysis, and Pocket Selection

I performed computational hot-spot mapping to identify candidate non-CAT PARP-1 pockets and evaluated the identified hot spots in terms of druggability. Specifically, I used the

FTMap server[243] (default settings; accessed March 30th, 2020) to predict hot spots on the surface of the 4DQY[45] PARP-1 structure. I eliminated all hot spots located in the catalytic site, as well as hot spots on the co-crystalized DNA molecule.

To evaluate the remaining non-catalytic hot spots, I used two pocket analysis programs: POVME[248] (version 2.1) (default settings) and DoGSiteScorer[249] (version 2.0.0 run with default settings). POVME calculates the volume of a pocket by filling the empty space of the cavities with equidistant points[248]; points that are located near protein atoms are removed and the remaining points are used to calculate the pocket's volume[248]. Additionally, POVME outputs a PDB file of the remaining points, which is useful for visualizing the dimensions of a pocket[248]. DoGSiteScorer calculates multiple physiochemical properties (e.g., pocket volume, number of aromatic/positive/negative/polar/nonpolar amino acids) into a "Drug Score," with 0.0 being the least druggable and 1.0 being the most druggable[249]. I also analyzed the catalytic pocket with both programs to contextualize my results. The niraparib-bound 4R6E:A structure[17] serves as a positive control for a druggable pocket.

I added the criteria that the selected pocket must have residues that, if altered, retain WT affinity for DNA but disrupt PARP-1 catalytic activity. Current PARPi are lethal to HR- and BER-deficient cancer cells because they both inhibit PARP-1 catalytic activity and trap PARP-1 on DNA[5,13,55,74–77]. Thus, I reasoned that disrupting an ideal non-catalytic pocket would not influence DNA binding activity, but would have a profound influence on PARP-1 catalytic activation. A thorough literature search for mutations within each pocket supported my druggability analysis and pocket selection process. Ultimately, I selected a DBD pocket located at the interface of Zn1-Zn3 as the site for AutoGrow4-guided CADD.

3.2.5.2 AutoGrow4 Applied to the DBD

To generate novel compounds predicted to bind the Zn1-Zn3 interface, I applied AutoGrow4 using a large-scale lead-generation approach. I chose a lead-generation approach rather than a lead-optimization approach because no existing drugs bind this pocket. To enable a fair comparison between compounds that bind the DBD and the CAT, I used similar AutoGrow4 settings to those used in the large-scale *de novo* AutoGrow4 run described in "Chapter 3.3.1: Large-Scale *de novo* PARPi Run" (e.g., the same selector, seed population, population sizes, docking settings, and Gypsum-DL settings); however, in order to better optimize the run (see issues of the large-scale *de novo* run discussed in "Chapter 3.3.1: Large-Scale *de novo* PARPi Run"), I set the DBD runs to propagate for fewer generations (*num_generations* was set to 10) and set the amount of diversity to remain constant throughout the run (*diversity_seed_depreciation_per_gen* was set to zero). Both the DBD and CAT runs were performed in triplicate. The exact settings for the DBD runs are provided in Appendix JSON 7.

These AutoGrow4 run were performed using ten MPI-enabled CRC computer nodes with 28-core Broadwell processors and 64GB RAM/node, networked with Intel's Omni-Path communication architecture.

3.2.5.3 Post-Run Analysis

To analyze the compounds produced by the AutoGrow4 DBD runs, I pooled the compounds from the three runs by generation into generations. Because parallel simulations can

produce identical compounds, I eliminated all redundant canonical SMILES and preserved the best docking score. I repeated this process on the three CAT runs.

To compare the populations of compounds produced by the DBD and CAT runs, I analyzed their 'physiochemical properties such as compound flexibility (i.e., the ratio of rotatable to rigid bonds) and the size of the largest cyclic group. All properties were assessed using the Python library Scopy[166]. To assess the protein-ligand interactions of the DBD-targeted compounds, I used the BINANA[246] algorithm following the procedures described in "Chapter 3.2.3: Interaction Assessment of AutoGrow4-Generated Compounds." Additionally, I visually inspected the compounds with strongest predicted binding affinities to better understand high affinity binding within the chosen pocket.

3.3 Results and Discussion

In this section, I will present five applications of AutoGrow4 runs to PARP-1. First, I will present a large-scale lead-generation run targeting the catalytic site. Second, I will describe an AutoGrow4-guided PARPi lead-optimization run that also targets the catalytic site. Third, I will demonstrate hypothesis generation as a new application of the AutoGrow algorithm. This hypothesis-generation approach led me to my fourth application, applying AutoGrow4 to a post-translationally modified PARP-1 catalytic site that is resistant to known PARPi. Here I have the goal of finding novel inhibitors that are unaffected by the post-translational modification. Fifth, I

use AutoGrow4 to identify predicted inhibitors that do not bind to the catalytic site. My analysis of these compounds will help guide future rational PARPi design.

3.3.1 Large-Scale *de novo* PARPi Run

In this section I will discuss a large-scale AutoGrow4 lead-generation run applied to the PARP-1 catalytic site. The goal of this large-scale *de novo* run was to create drug-like compounds optimized for strong predicted binding affinities, as well as to highlight AutoGrow4's ability to process hundreds of thousands of compounds in a single run. I ran AutoGrow4 with large population sizes and for 30 generations. The run was seeded with a large, diverse library of random seed molecules and fragments. A more thorough description of my methods is provided in "Section 3.2.1: Large-Scale *de novo* Run," and exact settings are provided in Appendix JSON 5.

3.3.1.1 Predicted Ligands

This large-scale run produced many high scoring compounds (Figure 14). Compound 4, created in the 30th generation by crossover, is one of the best scored compounds in the run and has a QVina2-predicted binding affinity of -16.7 kcal/mol (Figure 14 and Figure 15). The crossover event that created Compound 4 randomly chose the same decorating moieties as one of Compound 4's parents, essentially resampling the parent with newly generated structural variants (via Gypsum-DL) and docking poses.

Some of the best-scoring compounds are very similar to each other. For instance, Compound 4 is identical to the best-scoring compound from the 20th generation, but with a

different ionization state and docking pose (i.e., binding position/orientation) (Figure 14A). Two other close analogs of Compound 4, including the best-scoring compound from the 25th generation (Figure 14B middle), also have the same overall best docking score of the run, -16.7 kcal/mol (Figure 14B). These best-scoring analogs share three structural regions: a 7-methylbenzo[d][1,2,3]triazol-1-ide on one end, a central 1H-oxazolo[4',5':3,4]benzo[1,2-d][1,2,3]triazole, and a 1-ethyltetrazolidine at the other end (Figure 14B). That highly similar chemical structures have come to dominate the set of best-scoring compounds suggests convergence on a local optimum. The convergence of this run, as well as the implications of convergence, will be discussed further in "Chapter 3.3.1.3: A Caution Regarding Homogeneity and Convergence."

Compound 4 has a 1H-benzo[d][1,2,3]triazole substructure that is predicted to form a π-π stacking interaction with the PARP-1 Y907, H862, and Y896 residues. Two of these residues (H862 and Y896) are part of the PARP-1 catalytic triad[16], which is conserved in PARP-1 through PARP-6[16]. Hydrogen bonds are also predicted to form between both compounds' cyclic nitrogen atoms and the backbone atoms of PARP-1 residues G863, R865, and R878.

Lastly, an electrostatic interaction between Compound 4's tetrazolidine substructure and D766 (located on the HD αF helix) suggests that Compound 4 is either a type I PARPi (an inhibitor that promotes PARP-1 retention on DNA through allosteric regulation, resulting in long DNA-retention times and high toxicity[73]) or a type II PARPi (an allosterically neutral inhibitor with moderate DNA-retention times and toxicity[73]). Type I PARPi drive HD instability, which causes a strong allosteric effect that traps PARP-1 on DNA[73]. Although the electrostatic interaction between Compound 4 and D766 is dependent on the protonation states of D766 and the

tetrazolidine substructure (Figure 15B), the close proximity of Compound 4 to D766 (Figure 15B) suggests that Compound 4 may be capable of disrupting the HD αF helix.

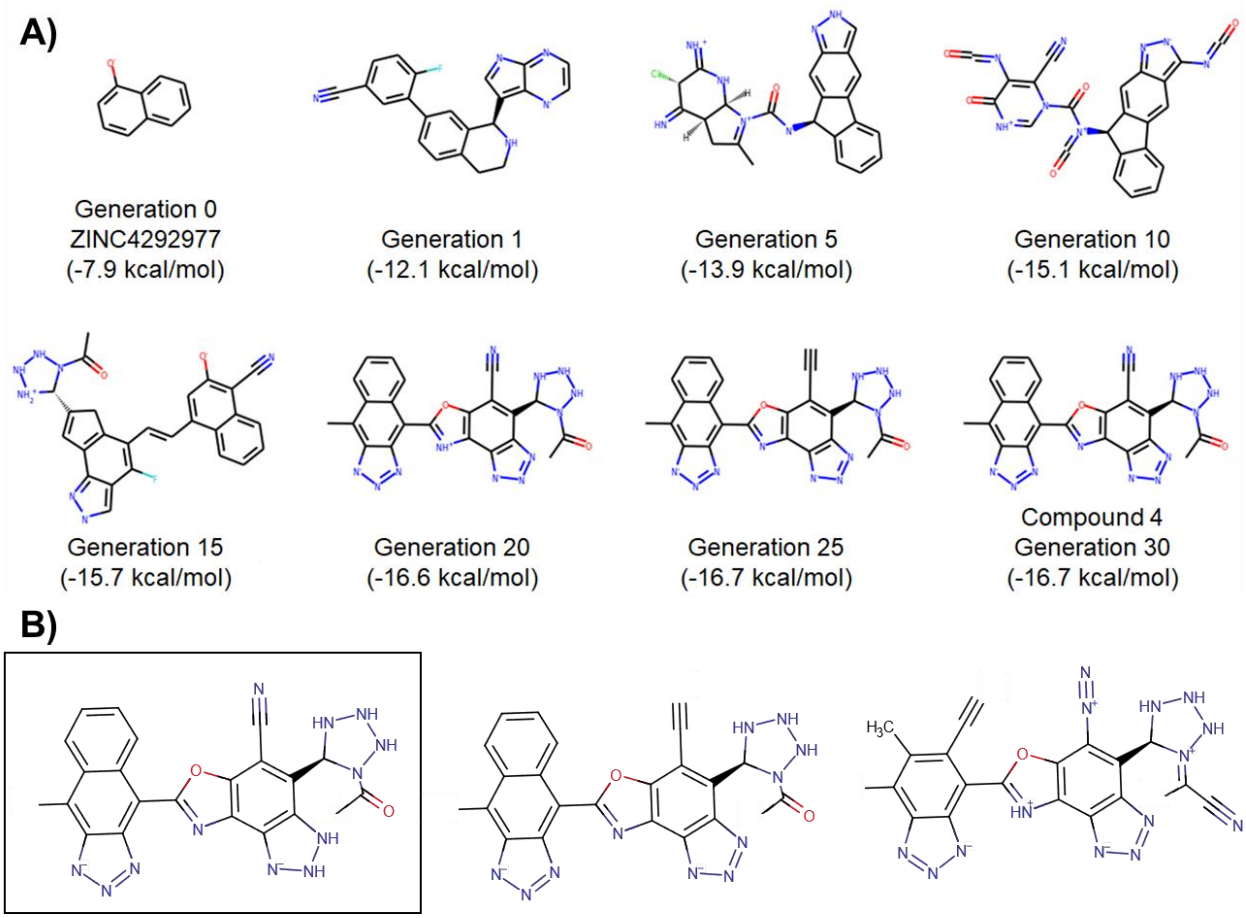


Figure 14. Best-scoring compounds from every fifth generation.

A) The best-scoring compounds from generations 0, 1, 5, 10, 15, 20, 25, and 30 of AutoGrow4's large-scale *de novo* run. ZINC4292977 (top left) is a source compound that was downloaded from the ZINC15 database[230]. Compound 4 (bottom right) is the best-scoring compound in the run and is also shown in Figure 15. All reported docking scores were calculated using QVina2. B) Compound 4 (boxed compound on left) and two close analogs (middle and right) had equally good docking scores (-16.7 kcal/mol).

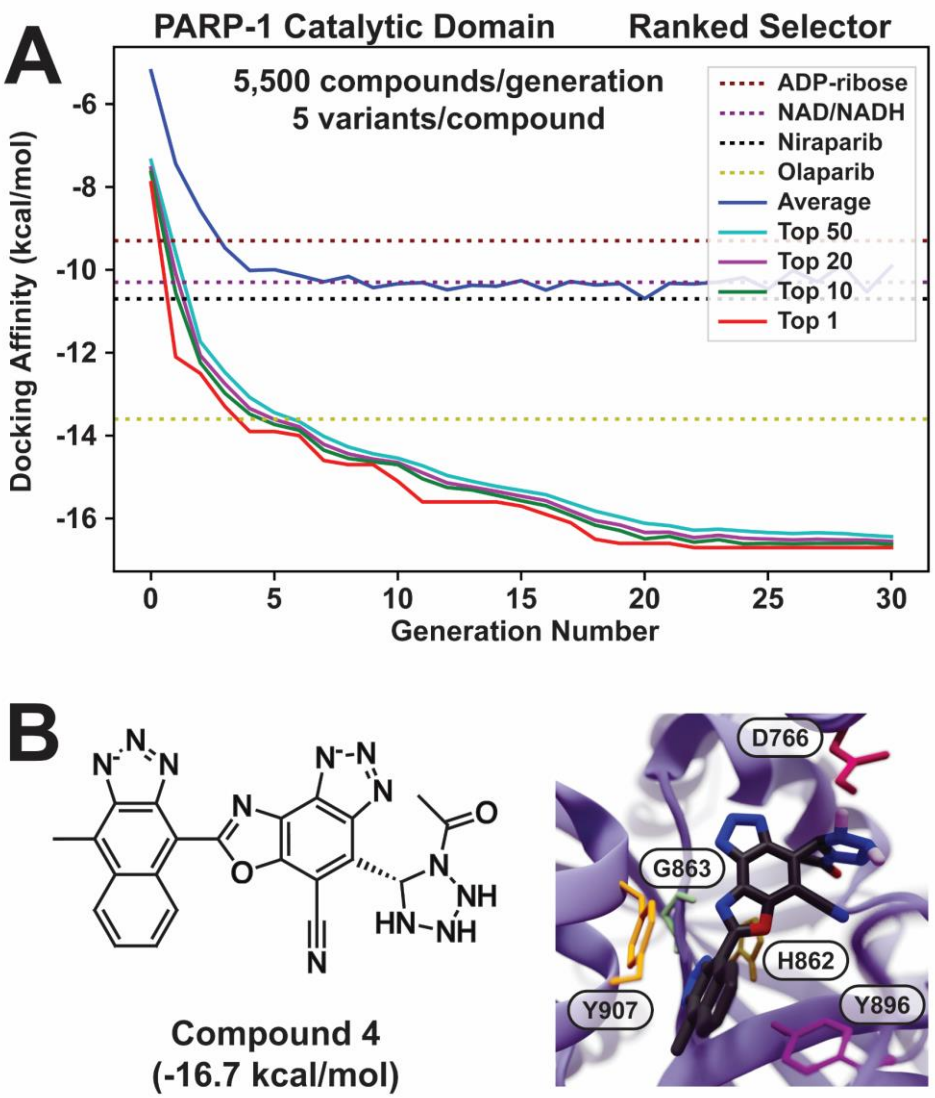


Figure 15. Results of the large-scale *de novo* run.

A) A plot of the QVina2 scores for each generation. Shown in blue line is the average score of all compounds per generation. The average scores of the top 50, 20, 10, and 1 compounds are shown in cyan, purple, green, and red line, respectively. The QVina2 docking scores of known ligands are shown as dashed lines. B) One of the best-scoring AutoGrow4-generated compounds, Compound 4, that was docked with QVina2 (30th generation). To the left is a 2D representation of Compound 4. To the right is Compound 4 docked into the PARP-1 catalytic domain (PDB ID: 4R6E:A, shown in blue ribbon). Select residues that interact with Compound 4 are shown in colored stick representation. Figure is from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. This figure is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium"[6].

3.3.1.2 A Caution Regarding Chemical Properties

Although longer AutoGrow4 runs have the potential to produce compounds with remarkable predicted binding affinities, I recommend running multiple independent runs for fewer generations. In this section, I will discuss several disadvantages of longer runs and approaches to counter these disadvantages.

First, as compounds evolve, biases within the fitness function and/or ligand-creation operators begin to apply undesired selective pressures. These selective pressures cause compounds to acquire chemical properties that are favored by these biases. For example, Compound 4 (Figure 15B p.134), one of the best-scoring compounds produced by the extensive PARP-1 *de novo* run, has a MW near the 480 Da maximum permitted by the applied Ghose filter (478.1 Da)[168]. The average MW of the top 100 compounds of this run is 476.98 Da. This apparent evolutionary pressure favoring increased MW is likely caused by (1) the Vina scoring function's known bias in favor of larger molecules[184], and (2) the AutoGrow4 mutation and crossover operators, which tend to expand compounds rather than truncate them. The latter is further evaluated in "Section 3.3.2.2: AutoGrow4 Operators and Molecular Weight." This size bias may be mitigated by using filters that place tighter restrictions on MW, as well as ligand-efficiency rescoring[235].

Second, longer AutoGrow4 runs can result in the accumulation of undesirable functional groups. The 24th-generation compound shown in the right position of Figure 14B (p.133), one of the best-scoring compounds in the large-scale *de novo* run, is an excellent example of a compound acquiring such undesirable moieties. It accumulated azo and ethyne moieties, two functional groups that belong to a broad category of substructures that are considered to be mutagenic,

pharmacokinetically unfavorable, reactive, and/or likely to interfere with typical high-throughput screening approaches[171]. These functional groups would have been filtered out had the BRENK filter been applied during the run[171].

The third disadvantage of longer AutoGrow4 runs relates to a chemical property of great importance in *de novo* design, compound synthesizability. The cumulative number of modifying events (e.g., crossover and mutations) that influence a population increases with every new generation, creating a population drift in which subsequent generations become less and less like the source compounds. Unfortunately, this also means that synthesizability tends to diminish with every generation. Again, filter(s) and scoring functions, particularly ones that factor in synthesizability, may help to limit this problem.

3.3.1.3 A Caution Regarding Homogeneity and Convergence

In this section, I will discuss the problem of population homogeneity for longer AutoGrow4 runs. Significant loss of population diversity often results in premature convergence[147]. This occurs when compounds become so fit that new compounds are unable to outcompete them. Taken to an extreme, these fit compounds can become overrepresented within the pool of generated small molecules, resulting in a homogenous population that is unable to escape a local optimum.

Early generations tend to rapidly improve in overall fitness[6] (Figure 15A p.134), but the rate of improvement slows in later generations despite requiring similar computational resources (Figure 15A p.134). This decrease was observed in the large-scale *de novo* run (Figure 15A p.134). The average docking score of the top 50 molecules improved -6.09 kcal/mol from generation zero

to five (-7.36 kcal/mol to -13.45 kcal/mol, respectively), but only improved another -2.99 kcal/mol by the 30th generation (-16.44 kcal/mol) (Figure 15A p.134). By the 20th generation the populations began to converge, with only minor additional improvements in fitness (Figure 15A p.134). In fact, the top-scoring compounds for the last ten generations included only Compound 4 and its analogs (Figure 15 p.134). Although it is possible for a GA to escape convergence, it is more computationally efficient to run multiple independent experiments for fewer generations.

To determine when the run had reached convergence, I calculated the normalized diversity scores for the total population, as well as for the best docking scored 50, 100, 200, and 500 compounds per generation ("Chapter 2.3.5: Calculating Normalized Diversity Scores"). The best-scoring compounds became increasingly homogenous as the run continued (Figure 14 p.133). For instance, the normalized diversity score of the top 500 docked compounds (9.09%) increased from a low of 0.23 in the second generation to 0.43 in the 30th generation (Figure 16). As described in "Section "1.2.1.3: Fitness," this loss of diversity and lack of score improvement indicates that a subset of compounds is so fit that new compounds are unable to outcompete them. These are signs of convergence. However, the normalized diversity of the entire population remained fairly constant from generation 5 to 30 (Figure 16), suggesting that the population did not fully converge.

AutoGrow4 offers several strategies to delay population convergence and homogeneity. Seeding an experiment with a sizable library of diverse seed molecules, particularly in earlier generations, can encourage the exploration of a larger subset of chemistry space. The incorporation of the diversity scoring function (see "Section 2.2.7: Assessing Fitness") also actively incorporates more diversity into the list of compounds that seed each generation (Figure 9 p.74). By ensuring that each generation is seeded with a combination of well-scoring (primary fitness metric) and

unique (secondary fitness metric) compounds, AutoGrow4 aims to search more of chemistry space while still maintaining selective pressure for well-scoring compounds. Despite these safeguards, I advise performing multiple runs of fewer generations rather than longer runs.

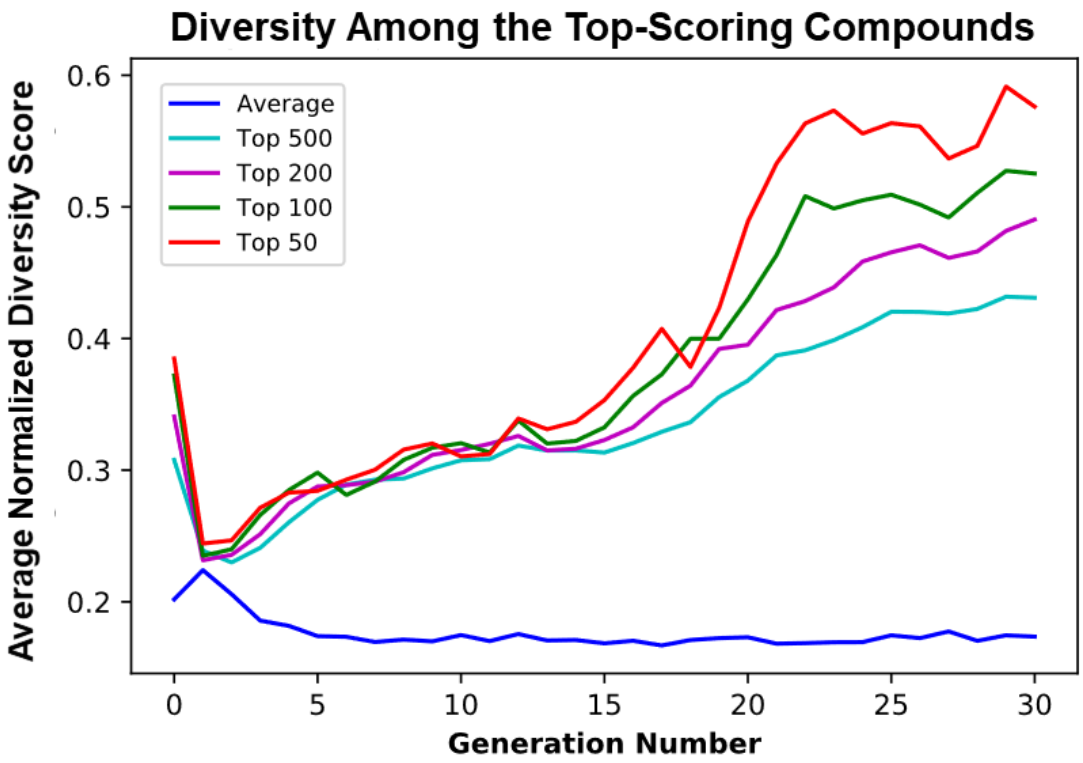


Figure 16. Diversity scores of the best-scoring compounds.
The normalized diversity score of each generation, as well as the top 50, 100, 200, and 500 compounds per generation. The larger the normalized diversity score, the more homogenous a subpopulation. Although there is little change in the overall diversity score, the best docking scored compounds become more homogenous in later generations.

3.3.2 PARP-1 Lead Optimization

In this section, I will discuss AutoGrow4-guided lead optimization of known PARPi. A *de novo* search seeded with random molecules is an excellent tool for finding well-scoring candidate inhibitors, but it invests a large amount of computational resources in testing unlikely leads. In contrast, seeding a *de novo* lead optimization search with a set of known inhibitors orients the search near these inhibitors. The search then loses less time to testing compounds unlikely to become successful drugs.

The goal of these lead-optimization runs was to evolve compounds with improved docking scores that remain chemically similar to the PARPi seeds and, in doing so, to illustrate AutoGrow4's utility as a tool for lead optimization. I seeded six independent AutoGrow4 runs with 11 PARPi, together with 83 fragments generated by BRICS decomposition of those 11 PARPi (see "Section 3.2.2.2: Preparation of PARP inhibitor (PARPi) Source Library"). To focus the search near these PARPi, I only ran AutoGrow4 for five generations. I increased the QVina2 docking-exhaustiveness parameter from the default 8 to 25. This increased docking parameter is computationally expensive but improves the chance of finding an optimal docked pose. I also used a large population size for each generation ("Section 3.2.2.3: PARPi Lead-Optimization" and Appendix JSON 6).

3.3.2.1 Predicted Ligands

AutoGrow4 produced many compounds with stronger predicted binding affinities than the seed PARPi (i.e., the zeroth generation). In fact, by the first generation the docking scores had already improved (Figure 17). The average QVina2 docking score for the top ten compounds across all six runs (the grand mean) was -13.0 kcal/mol (Figure 17). By the third generation, the grand mean of the top 20 compounds already equaled that of the best-scored pose of the best-scoring known PARPi, olaparib (AstraZeneca), with a score of -13.6 kcal/mol (Figure 17). By the fifth generation, the grand mean of the top 50 compounds (-14.0 kcal/mol) surpassed that of olaparib (Figure 17).

One of the best-scoring molecules from the six PARPi lead-optimization runs, Compound 5, has chemical groups derived from olaparib and CEP-9722 (Cephalon), with a QVina2 score of -14.7 kcal/mol (Figure 18). I focus the discussion here on Compound 5 rather than on the best-scoring compound, Compound 3 (Figure 10B p.76), because Compound 3 is comprised of a single PARPi that had been heavily modified by three mutation events (Figure 10B p.76). Compound 5, on the other hand, is derived from a single crossover event of two PARPi fragments ($Olaparib_{Frag3}$ and $CEP\text{-}9722_{Frag1}$) and is comprised exclusively of PARPi-derived moieties, providing a simple yet excellent example of AutoGrow4-directed lead optimization (Figure 18).

The shared substructure used to cross $Olaparib_{Frag3}$ and $CEP\text{-}9722_{Frag1}$ is a benzene ring (Figure 18 and Figure 19). Compound 5 is comprised of two main segments, a (4-(cyclopropylidene(hydroxy)methyl)piperazin-1-yl)(2-fluorophenyl)methanone derived from

olaparib, and a 4,5,6,7-tetrahydro-1H-cyclopenta[a]pyrrolo[3,4-c]carbazole-1,3(2H)-dione derived from CEP-9722 (Figure 18 and Figure 19).

Analysis of PARPi and Compound 5 using the BINANA algorithm[246] revealed several important interactions. Compound 5 participates in π-π stacking interactions with the PARP-1 Y907 and Y896 residues (Figure 19C). Additionally, a carbonyl oxygen atom of Compound 5 forms a hydrogen bond with G863 (Figure 19C). These same interactions are observed in many known PARPi[17,18,250,251,19,106–112], including the crystallographic structure of olaparib[18] (Figure 19A) and the best-scored pose of CEP-9722 (Figure 19B).

The region of Compound 5 that is predicted to participate in these π-π stacking interactions is a five-ringed aromatic group derived from CEP-9722 (Figure 19). Interestingly, this interaction is also present in CEP-9722 even though half of the compound is completely different from CEP-9722, and this five-ringed aromatic group aligns in a very similar manner (Figure 19B-C). In addition to π-π stacking, a CEP-9722-derived oxygen atom forms a hydrogen bond with G863 (Figure 19). This hydrogen bond is located near the Y907 residue (Figure 19).

The π-π stacking interactions with Y907 and Y896 appear to be very important for PARPi binding. This interaction can also be observed in the crystal structure of olaparib bound in the PARP-1 catalytic pocket (PDB: 5DS3[18]) (Figure 19A), as well as in crystal structures of niraparib (PDD: 4R6E[17]), rucaparib (PDB: 4RV6[17]), and talazoparib (PDB: 4UND[17]). Many of these same interactions are found in the top AutoGrow4-generated compounds from both the large-scale *de novo* and PARPi lead-optimization runs.

Compound 5 contains a piperazine moiety that is derived from olaparib (Figure 18 and Figure 19). CEP-9722 also contains a piperazine, but it is attached to the molecule in a different

location, and the CEP-9722 and olaparib piperazine moieties are oriented in different directions within the pocket (Figure 19B-C). Interestingly, the olaparib-derived piperazine of Compound 5 docks in the same orientation as the CEP-9722 piperazine (Figure 19). Several additional crystal structures of PARPi with piperazine moieties have alternate positionings[18,110,252,253]. One prominent example is the structure of the type I PARPi EB-47[254,25573] bound to PARP16 (PDB ID: 6HXR). In this crystal structure the piperazine group is placed in a similar manner to Compound 5's docked pose. However, it should be noted that EB-47 was not included in the PARPi source-library that seeded the six lead-optimization runs.

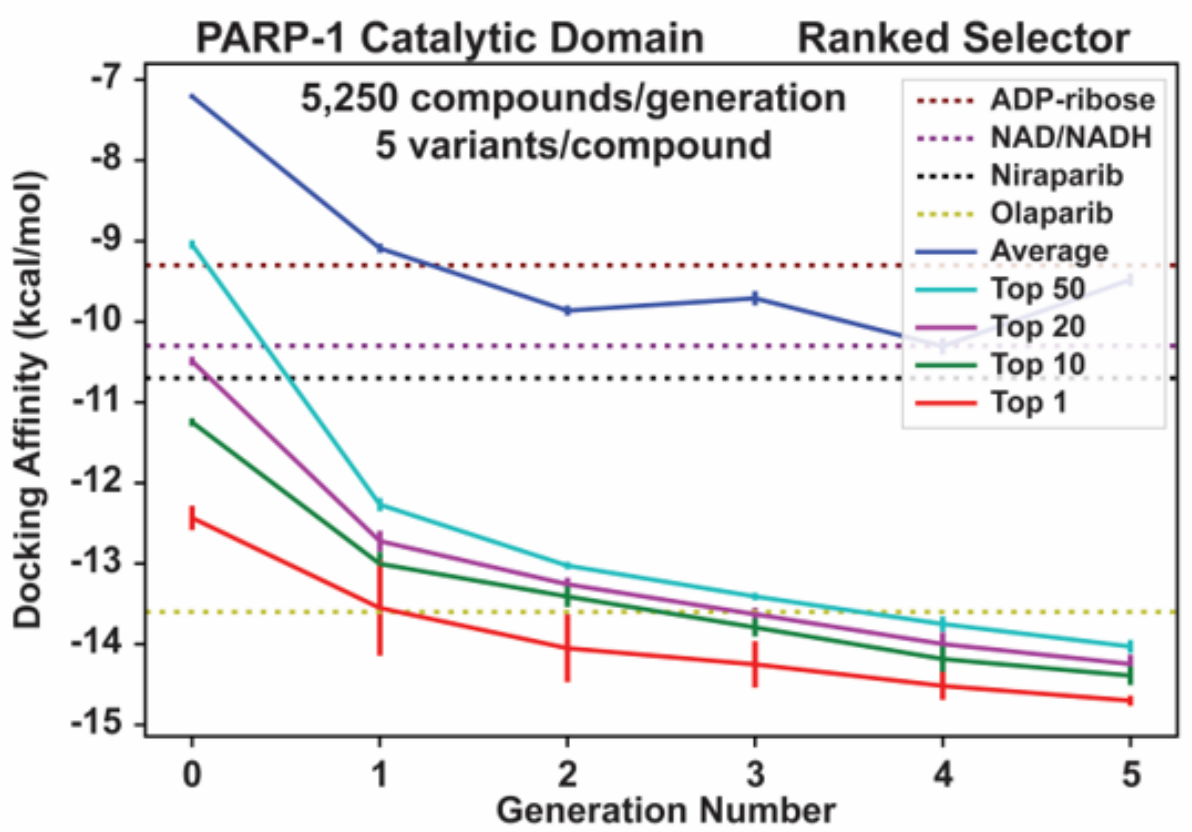


Figure 17. A plot of the six PARPi lead-optimization runs.
Generation zero is the initial seed population consisting of 11 PARPi and 83 PARPi fragments. Generations one to five are the AutoGrow4-generated populations. For each generation, the grand mean of all QVina2 scores across all six runs is shown in blue. The grand means of the top 50, 20, 10, and 1 compounds are shown in cyan, purple, green, and red, respectively. Error bars represent the standard deviations for the six independent runs. Figure is from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. This figure is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium."[6]

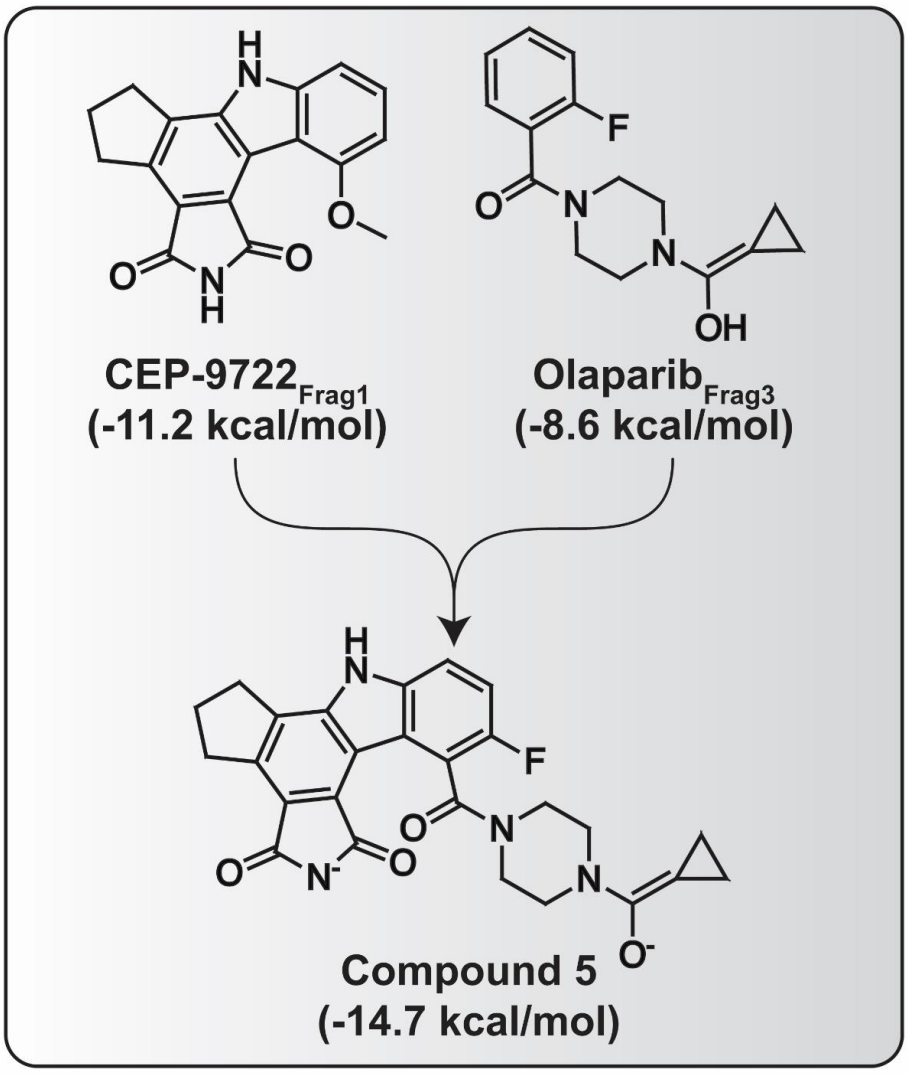


Figure 18. The lineage of Compound 5 generated by a PARPi lead-optimization run. Compound 5, one of the best-scoring compounds as assessed by QVina2, was created in the first generation via a single crossover between CEP-9722$_{Frag1}$ and Olaparib$_{Frag3}$. Figure is from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. This figure is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium."[6]

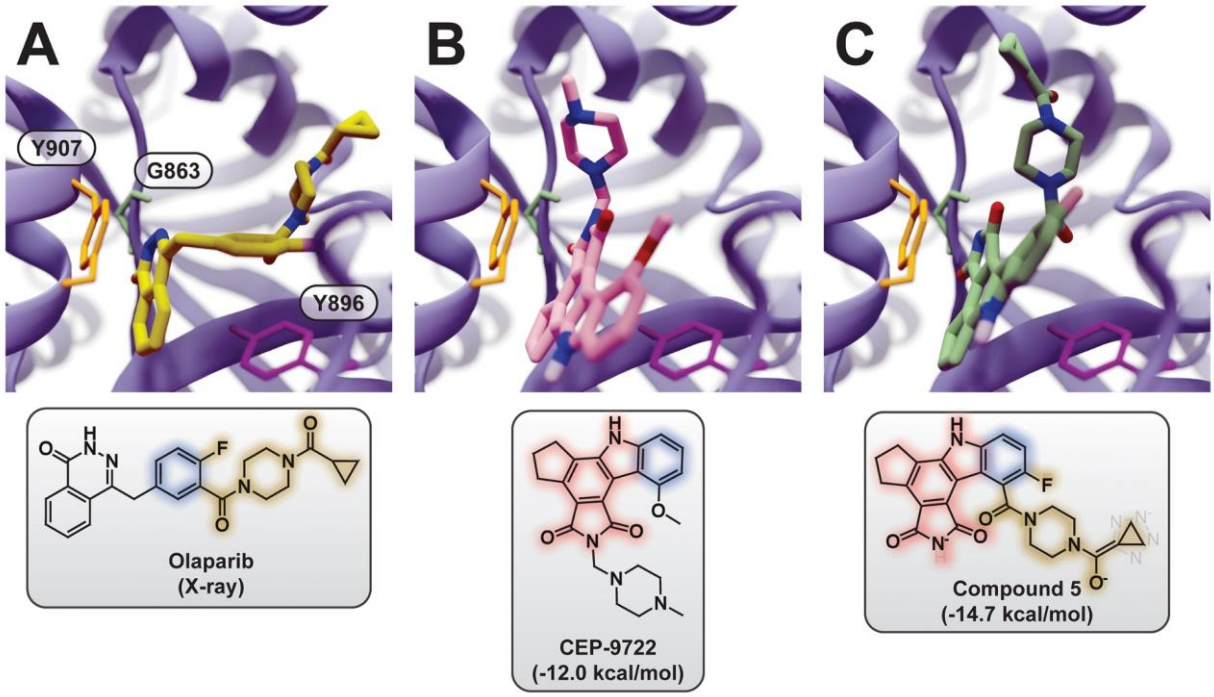


Figure 19. Example ligand structures and poses from a lead-optimization run. Select residues of important interactions are shown in colored stick representation (top). The shared substructures are highlighted in blue, yellow, and pink (bottom). A) For easy comparison, the crystallographic olaparib pose (PDB: 5DS3) was aligned and superimposed on the 4R6E:A structure (blue ribbon). B) The best-scoring pose of CEP-9722 docked into the 4R6E:A structure using QVina2. C) The best-scoring pose of Compound 5 docked into the 4R6E:A structure using QVina2. Figure is from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. This figure is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium."[6]

3.3.2.2 AutoGrow4 Operators and Molecular Weight

Molecular weight (MW) is an important property for drug-likeness[163,168], and the majority of small-molecule drugs have MWs between 160 Da and 480 Da[168]. In this section I will examine the MW trends in the AutoGrow4 runs, as well as describe some strategies for mitigating these trends.

Compounds in later generations tended to be larger than those in earlier generations. The average MW of the source PARPi and PARPi fragments is 196.37 Da, but by the fifth generation the average MW of generated compounds was 386.18 Da. This increase in MW could be caused by an AutoGrow4 tendency to grow compounds rather than truncate them. Alternatively, because the Vina scoring function has been shown to favor larger compounds[184], scoring function bias may also contribute to the observed increase in compound size. To determine the role AutoGrow4 plays in growing compounds, we examined the lineages of compounds generated in the PARPi lead-optimization runs.

First, we analyzed compounds created by mutation. The MW of the 55,683 compounds created by the mutation operator in the six PARPi lead-optimization runs was on average 28% greater (66.0 Da) than their respective progenitor compounds. Despite this, MW decreased in 11% of all mutation events. For example, the mutation event shown in Figure 20A produced a compound with a lower MW than the parent compound.

Second, we analyzed compounds created by crossover events. Because crossovers rely on two parent compounds, we compared the MW of the child to the average MW of the parents. The six PARPi lead-optimization runs produced 50,169 compounds via the crossover operator; 43% of

these had a MW less than the average of its two parent compounds. The average change in MW was an increase of 5% (11.5 Da). In the example shown in Figure 20B, the MW of the child compound is 13.1% (45.6 Da) less than the average MW of the parent compounds.

Both the mutation and crossover operators tend to grow compounds. AutoGrow4 offers several strategies should users want to mitigate this tendency, such as providing chemical-property filters that restrict the MW of compounds. Additionally, the number of generations AutoGrow4 produces before the compound population approaches filter-imposed MW cutoffs can be expanded by seeding generation zero with low-MW small molecules. Increasing the number of crossover events relative to the number of mutations may also help because crossovers increase compound size to a lesser extent than mutations and so are more likely to reduce the MW of offspring. Lastly, AutoGrow4 provides an option to rescore compounds by ligand efficiency, which normalizes the docking score by the number of non-hydrogen atoms[235], effectively penalizing high-MW compounds.

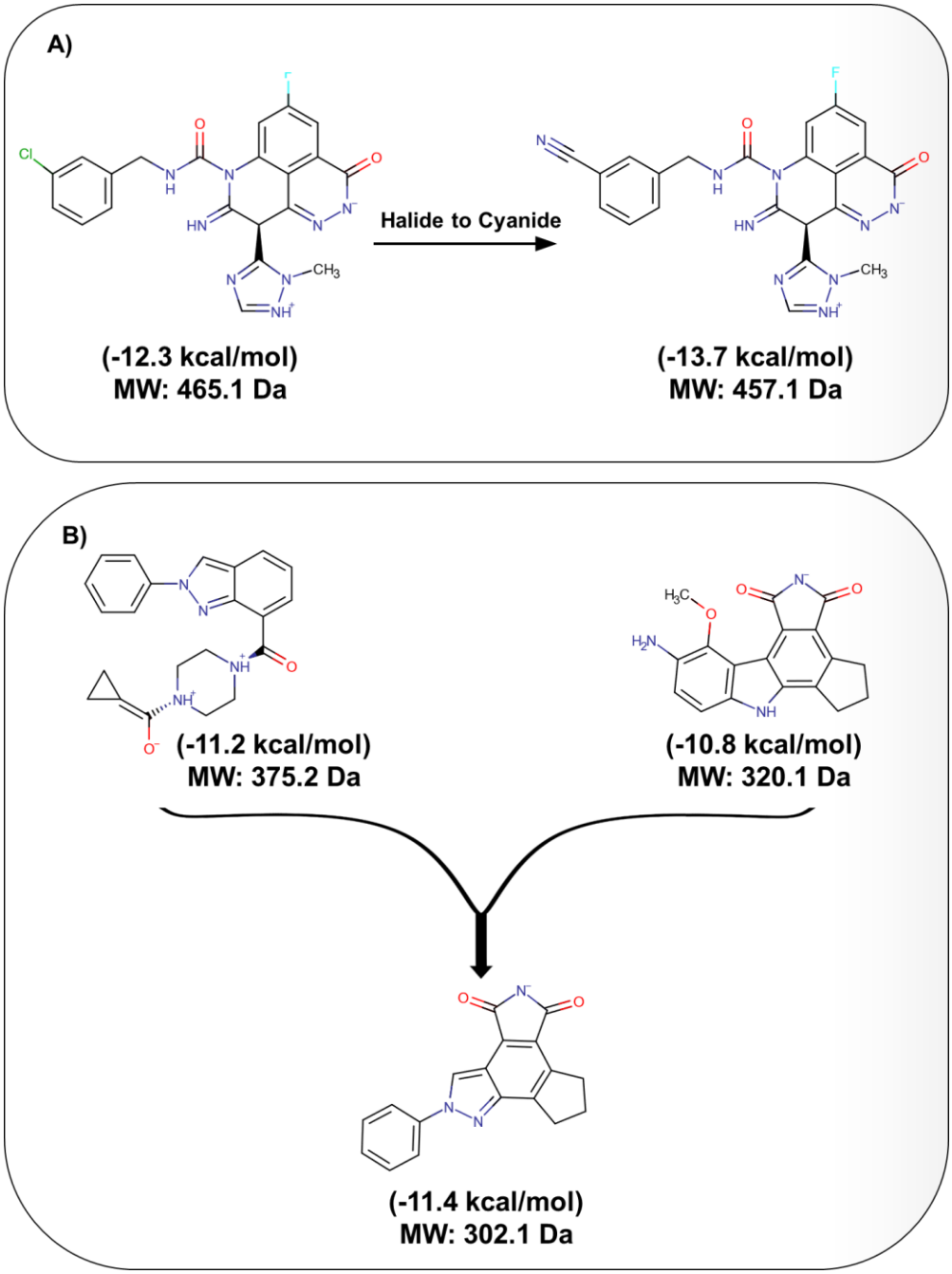


Figure 20. Example mutation and crossover events that reduce MW.
A) An example of a halide-to-cyanide mutation that produced a compound with a lower MW than that of the parents. The MW of the child compound is 8 Da less than that of the parent compound, a 1.7% reduction. B) An example of a crossover that produced a novel compound with properties of both parents, but a lower MW. All compounds in this figure were generated in the AutoGrow4 PARPi lead-optimization runs. Docking scores were calculated by QVina2, and MWs were calculated using RDKit.

3.3.3 AutoGrow4 as a Tool for Hypothesis Generation

In this chapter, I will use the data generated by the AutoGrow4 large-scale *de novo* run and the six parallel PARPi lead-optimization runs to demonstrate AutoGrow4's usefulness for generating hypotheses regarding pharmacologically important protein-ligand interactions. The top-scoring compounds from the large-scale *de novo* run provide insight into protein-ligand interactions that have been optimized by 30 generations of selective pressure, whereas the PARPi lead-optimization runs provide insight into interactions that are closer to those of experimentally verified PARPi. Cataloguing the most prevalent interactions among the best-scoring AutoGrow4-generated compounds can inform future experiments ranging from quantitative structure-activity relationship (QSAR)-based drug design to site-directed mutagenesis. Analysis of the protein-ligand interactions within both these data sets can teach us about the important interactions of the PARP-1 catalytic pocket.

3.3.3.1 Large-Scale *de novo* Run Validates AutoGrow4 as a Tool for Lead Generation

To show that AutoGrow4 is a capable tool for hypothesis generation, I identified the 100 compounds with the best docking scores from the hundreds of thousands of docking events that occurred throughout the large-scale *de novo* run. I then analyzed the single best predicted pose of each compound using the BINANA (version 1.1.2) algorithm[246]. Four interactions were present in all 100 docked poses: two π-π stacking interactions with Y907 and H862, respectively; an electrostatic interaction with D766; and a hydrogen-bond interaction with G863 (Table 8). Several

other prominent interactions were found that were not as universal: an electrostatic interaction with D770 (41%), a hydrogen-bond interaction with R865 (56%), and electrostatic and hydrogen-bond interactions with R878 (9% and 47%, respectively) (Table 8).

The large-scale *de novo* run was seeded with random small molecules, thereby blinding AutoGrow4 from prior knowledge of existing PARPi and their interactions, and thus serves as a test of AutoGrow4's ability to predict pharmacologically important interactions against the well-characterized catalytic pocket of PARP-1. As discussed above, AutoGrow4 recapitulated many of the interactions observed in crystallographic structures of known PARPi. For example, the crystallographic pose of the known PARPi olaparib participates in many of the same interactions seen among the top AutoGrow4 compounds[18]. The fact that AutoGrow4 so frequently generated compounds that mimic the binding interactions of known PARPi demonstrates AutoGrow4's proficiency at *de novo* generation and lead-optimization approaches.

This application of AutoGrow4 also generated many compounds with better docking scores than known PARPi (Figure 15A p.134). These *in silico* compounds provide insight into future optimization strategies. For example, all 100 top-scoring AutoGrow4-generated compounds formed electrostatic interactions with D766 but olaparib does not, suggesting that adding a positively charged moiety extending the olaparib piperazine may enable an additional interaction with D766.

Table 8. Protein-ligand interactions of the 100 best-scored compounds from the large-scale *de novo* run.

This table is limited to the single best-scored pose per compound and only includes interactions detected by BINANA (version 1.1.2), using the default settings. Infrequent interactions (< 10%) are excluded from the table. All values are given as percentages. Table is from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. This table is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium."[6]

	D766	**D770**	**H862**	**G863**	**R865**	**R878**	**Y907**
Cation- π	0	0	98	0	0	0	0
Hydrogen bond	3	0	1	100	56	47	1
Electrostatic	100	41	3	0	0	9	0
T-stacking	0	0	71	0	0	0	0
π-π	0	0	100	0	0	0	100

3.3.3.2 PARPi Lead-optimization Runs Detect Additional Interactions

Determining the most prevalent protein-ligand interactions among well-scoring compounds can inform future lead-optimization efforts. A single interaction was unanimous among the top 100 compounds generated by the PARPi lead-optimization runs : π-π stacking with Y907 (Table 9). Similar to the large-scale *de novo* run, the most prevalent interactions included interactions with H862 (42% π-π stacking, 8% cation-π, 6% T-stacking, and 1% hydrogen bonding), hydrogen bonding with G863 (74%), interactions with R865 (19% hydrogen bonding and 1% electrostatic interaction), and hydrogen bonding with R878 (50%) (Table 9). Unlike the large-scale *de novo* run, where electrostatic interactions were commonly formed with the HD αF residues D766 and D770 (100% and 41% respectively), only 6% of the top lead optimization pool formed electrostatic interactions with D766, and none formed electrostatic interactions with D770. Because disruption of the HD αF helix plays an important role in determining the strength of DNA-trapping[73], it is likely that the PARPi lead-optimization compounds would have weaker trapping effects than the compounds from the large-scale *de novo* run.

Two interactions were common in the lead-optimization pool that were not common in the large-scale *de novo* pool: hydrogen bonding with N868 (10% lead-optimization pool; not present in *de novo* pool), and π-π stacking and hydrogen bonding with Y896 (22% and 1% respectively in the lead-optimization pool; 4% and 2% respectively in the *de novo* pool) (Table 9). N868 and R865 are located toward the middle of the catalytic pocket. Y907 and D766 are on opposite ends of the pocket[17]. Like R865 interactions, N868 interactions appear to stabilize linker regions of compounds that connect moieties docked near Y907 and moieties docked near D766 and N878.

The tyrosine residues Y896 and Y907 are located near each other in the 4R6E structure[17] in the nicotinamide-binding site[56], which is where prominent interactions located near Y907 occur. The decreased prevalence of π-π stacking with Y896 in the *de novo* pool is likely due to a closer positioning of aromatic groups near Y907; 37 compounds in the *de novo* pool were < 2.5 Å from Y907, but none of the compounds in the lead-optimization pool were proximal to Y907.

Table 9. Protein-ligand interactions of the 100 best-scored compounds from the PARPi lead-optimization runs.

This table is limited to the single best-scored pose per compound and only includes interactions detected by BINANA (version 1.1.2), using the default settings. Infrequent interactions (< 10%) are excluded from the table. All values are given as percentages.

	H862	**G863**	**R865**	**N868**	**R878**	**Y896**	**Y907**
Cation-π	8	0	0	0	0	0	1
Hydrogen bond	1	74	19	10	50	1	2
Electrostatic	0	0	1	0	0	0	0
T-stacking	6	0	0	0	0	0	0
π-π	42	0	0	0	0	22	100

3.3.3.3 Y907 π-π Stacking Interaction and the Future of Orthosteric PARPi

All of the best-scoring AutoGrow4-generated compounds from both the large-scale *de novo* and lead-optimization runs were predicted to form π-π stacking interactions with Y907 (Table 8 p.152, Table 9 p.155). Additionally, most prominent interactions are located near Y907, either in the nicotinamide-binding pocket (NI site) (e.g., Y896, and Y907) or in the adenine-ribose binding site (AD site) (e.g., D770, H862, and R865) (Table 8 p.152, Table 9 p.155, Figure 15 p.134, and Figure 19 p.146). That many co-crystallized PARPi also π-π stack with Y907[17–19,106–112] reinforces the critical role Y907 plays in high-affinity binding and suggests that this interaction may be broadly critical regardless of chemical scaffold.

The high frequency of Y907 interactions raises concerns for the future of orthosteric (i.e., non-allosteric) PARPi. Y907 is phosphorylated by the receptor tyrosine kinase c-Met[142], which results in increased catalytic activity and weakened binding affinity to PARPi[142]. Because c-Met can reduce PARPi binding, upregulation of c-Met provides a potential mechanism for PARPi resistance[142]. This highlights the need for further development of next-generation PARPi that do not rely on Y907 interaction, either by targeting a different pocket or by exploiting a different set of protein-ligand interactions.

3.3.4 AutoGrow4 Applied to PARP-1 with Phosphorylated Y907 (pY907)

Multiple AutoGrow4 runs revealed that Y907 plays an important role in mediating small-molecule binding to the PARP-1 catalytic pocket. Because phosphorylation of Y907 (pY907) by

the receptor tyrosine kinase c-Met decreases PARP-1 inhibition and weakens the binding affinity of veliparib, rucaparib, and olaparib[142], finding novel PARP-1 inhibitors that bind independent of Y907 phosphorylation is important for treating HR-deficient cancer cells that have gained resistance to PARPi via pY907. In this section, I will use AutoGrow4 to design candidate inhibitors that are predicted to bind PARP-1 when Y907 is phosphorylated.

3.3.4.1 Building the pY907 PARP-1 Structure and Orienting the Phosphate Moiety

To identify novel compounds that bind PARP-1 with phosphorylated Y907, I generated models of PARP-1 with pY907 for use in AutoGrow4 lead-optimization runs. Because there currently is no high-resolution structure of PARP-1 with pY907, I used the PyMOL plugin PyTMs (version 1.2)[247] to add and orient the phosphate group. I produced two structures of pY907, one using PyTMs' default settings and the other using PyTMs' "*Optimization*" option (Figure 21), which adjusts the dihedral angle of the phosphate moiety to reduce steric van der Waals clashes[247]. Using the default settings, PyTMs oriented the phosphate group away from the NAD+ binding site (Figure 21B). On the other hand, the "optimized" version oriented the phosphate group toward the NAD+ binding site (Figure 21C).

To determine which version of the pY907 structure is more likely to be biologically relevant, I used QVina2 to exhaustively dock the known PARPi veliparib (PDB: 5LX6[17]), rucaparib (PDB: 4RV6[17]), and olaparib (PDB: 5DS3[18]) into each version of pY907 PARP-1, as well as into the nonphosphorylated structure 4R6E:A. Because these three PARPi have weakened binding affinities when Y907 is phosphorylated[142], the model associated with the worst docking

scores is more likely to be biologically relevant. I predicted that the optimized pY907 PARP-1 model would yield worse docking scores because the phosphate group blocks π-π stacking with pY907. As expected, the docking scores associated with the optimized structure were worse for all three PARPi vs. nonphosphorylated PARP-1 (Table 10). I therefore selected this model to use in subsequent AutoGrow4 runs.

The average docking score for these three PARPi was -8.9 kcal/mol when using the optimized pY907 PARP-1, which was 13.6% weaker compared to the average score when using the nonphosphorylated PARP-1 structure (-10.3 kcal/mol) (Table 10). For the non-optimized pY907 PARP-1, QVina2 actually predicted stronger binding affinities for all three PARPi, with the average docking score of -10.5 kcal/mol (Table 10). Based on these results, I rejected the pY907 PARP-1 structure created with default settings and selected the optimized pY907 PARP-1 structure for further experiments. I refer to the optimized pY907 PARP-1 structure as pY907-PARP-1.

This approach for modeling PARP-1 with pY907 has its limitations, as it ignores the possibility that post-translational phosphorylation causes conformational changes within the catalytic pocket. However, there are currently no high-resolution PARP-1 structures with pY907 to suggest what rearrangements may take place in the pocket as a result of phosphorylation. Future research using techniques such as molecular dynamics, NMR, and X-ray crystallography to determine the structure of pY907 PARP-1 would greatly improve our understanding of the mechanism of pY907-triggered PARPi resistance.

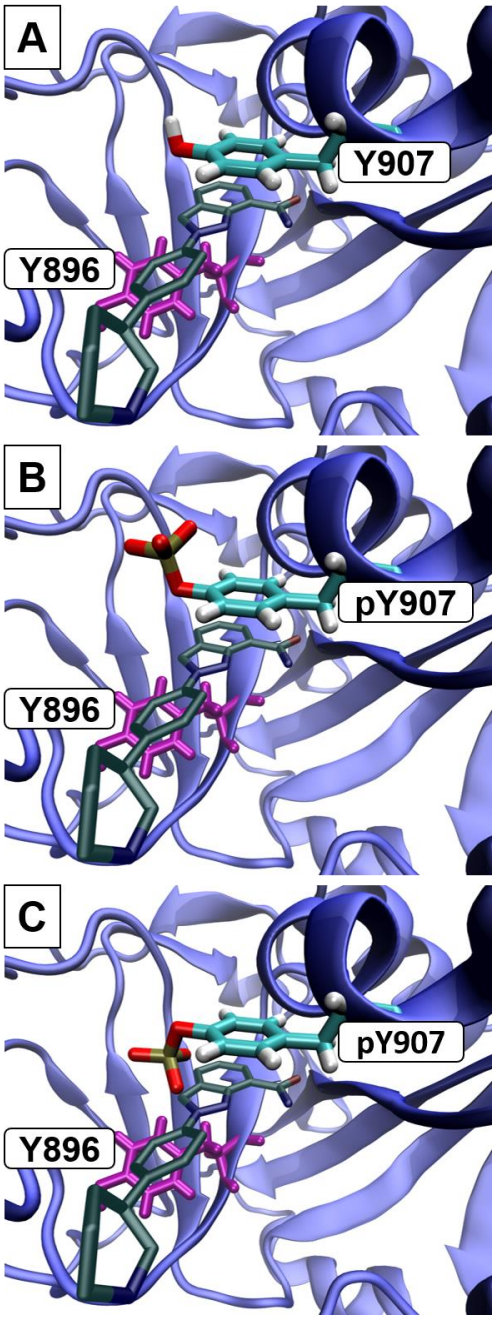


Figure 21. Comparisons of potential phosphate orientations for pY907.
A) The catalytic pocket of the nonphosphorylated PARP-1 catalytic domain (PDB: 4R6E:A). In this structure the hydrogen atom attached to the side-chain oxygen atom is pointing away from the binding pocket. B) The PyTMs-generated position of pY907 using default settings. The phosphate moiety is pointing away from the binding pocket. C) The PyTMs-generated position of the pY907 using the PyTMs "*Optimization*" option. The phosphate moiety is pointing into the binding pocket. PARP-1 is shown in blue cartoon representation. The crystal structure of niraparib (PDB: 4R6E) (gray) and Y896 (purple) are shown to provide context.

Table 10. Docking scores of three PARPi positioned within the nonphosphorylated and pY907-PARP-1 structures.

The three PARPi that were docked into the PARP-1 structures. "PARP-1" is the nonphosphorylated 4R6E:A structure. "pY907 Default" is the pY907-PARP-1 structure generated using PyTMs' default settings. "pY907 Optimized" is the pY907-PARP-1 structure generated using PyTMs' "*Optimization*" option. All docking scores were calculated using QVina2.

PARPi (PDB)	PARP-1	pY907 Default	pY907 Optimized
Veliparib (5LX6)	-8.4 kcal/mol	-8.7 kcal/mol	-7.5 kcal/mol
Rucaparib (4RV6)	-10.2 kcal/mol	-10.4 kcal/mol	-8.8 kcal/mol
Olaparib (5DS3)	-12.2 kcal/mol	-12.4 kcal/mol	-10.3 kcal/mol

3.3.4.2 AutoGrow4 Predicts Leads that Bind to pY907-PARP-1

To identify novel compounds that are predicted to bind PARP-1 independent of the Y907 phosphorylation state, I applied AutoGrow4 to the pY907-PARP-1 catalytic pocket. I ran six AutoGrow4 runs, using the same settings as the previous PARPi lead-optimization runs.

Over five generations, AutoGrow4 identified many candidate compounds with stronger predicted pY907-PARP-1 binding affinities than the initial seed population (Figure 22). The grand mean of the top 50 compounds improved 56.8% from -7.99 kcal/mol in generation zero to -12.53 kcal/mol in the fifth generation (Figure 22). This improvement is comparable to that seen in the first five generations of the initial nonphosphorylated PARP-1 lead-optimization runs (55.1%, Figure 17 p.144). The best-scored compound of the six runs had a docking score of -13.5 kcal/mol (Figure 23 and Figure 24). Analysis of this best-scored compound using the BINANA algorithm[246] identified an electrostatic interaction with D770 and a hydrogen bond with D770 and N868 (Figure 24B). Although this compound's bicyclo[3.2.0]heptane moiety was positioned only 3.2 Å from the phosphate group, BINANA did not detect any specific protein-ligand interactions with pY907 (Figure 24B).

To further assess the leads generated by these runs, I re-docked the top ten compounds into the nonphosphorylated PARP-1 structure (Figure 23). Of the ten reassessed compounds, half had improved docking scores when docked into the nonphosphorylated PARP-1, four had weaker scores, and one had the same predicted score (Figure 23). These results suggest that the phosphorylation of Y907 does not exclude binding to the catalytic domain. Re-docking the best-scored compound into the nonphosphorylated PARP-1 structure resulted in a near identical

positioning of the compound within the pocket, although with a weakened predicted binding affinity (-13.1 kcal/mol) (Figure 23 and Figure 24). Despite this, this compound had a better predicted binding affinity when docked into both PARP-1 structures than olaparib (PDB: 5DS3[18]) when docked into the nonphosphorylated PARP-1 structure (Figure 23, Figure 24, and Table 10 p.160).

Analysis of the 100 best-scored compounds using the BINANA algorithm[246] revealed several frequent interactions. Despite the presence of the phosphate moiety, 24% of the best-scoring compounds were still predicted to form π-π stacking interactions with pY907 (Table 11). In contrast, in the lead-optimization runs applied to the nonphosphorylated catalytic site, 100% of top compounds were predicted to form π-π stacking interactions with Y907 (Table 9 p.155). These results suggest that it is possible to design novel PARPi that bind independent of PARP-1's Y907/pY907 state. Additionally, BINANA found that 20% of the top compounds from the pY907-PARP-1 runs are predicted to form π-π stacking interactions with H862 (Table 11). Again, this interaction is more prevalent in the nonphosphorylated PARP-1 runs, where 42% of top compounds were predicted to participate in a π-π stacking interaction with H862 (Table 9 p.155). Lastly, BINANA detected hydrogen bonding in the pY907-PARP-1 runs: 13% with R878, 4% with H862, and 3% pY907 (Table 11). In the lead-optimization runs applied to the nonphosphorylated catalytic site, 50% of the top compounds were predicted to form hydrogen bonds with R878 (Table 9 p.155). One potential reason for fewer identified prominent (≥ 10%) protein-ligand interactions is that the pY907 disrupted the interactions responsible for strong binding.

This application of AutoGrow4 to pY907-PARP-1 identified candidate PARPi compounds that score well when docked into both the nonphosphorylated and pY907 PARP-1 catalytic pockets. My results suggest that pY907 does not preclude small-molecule binding to the catalytic domain, though it may obstruct several protein-ligand interactions that contribute to high-affinity binding to the nonphosphorylated pocket. Although these candidate compounds have yet to be experimentally verified as PARPi, this work suggests a method for designing inhibitors that bind regardless of the Y907 phosphorylation state. Current inhibitors are susceptible to PARPi resistance[142] because they all target the same catalytic pocket[17–19,106–112], so finding alternative inhibitors that are not susceptible to the same resistance mechanisms is important for treating HR-deficient cancer cells.

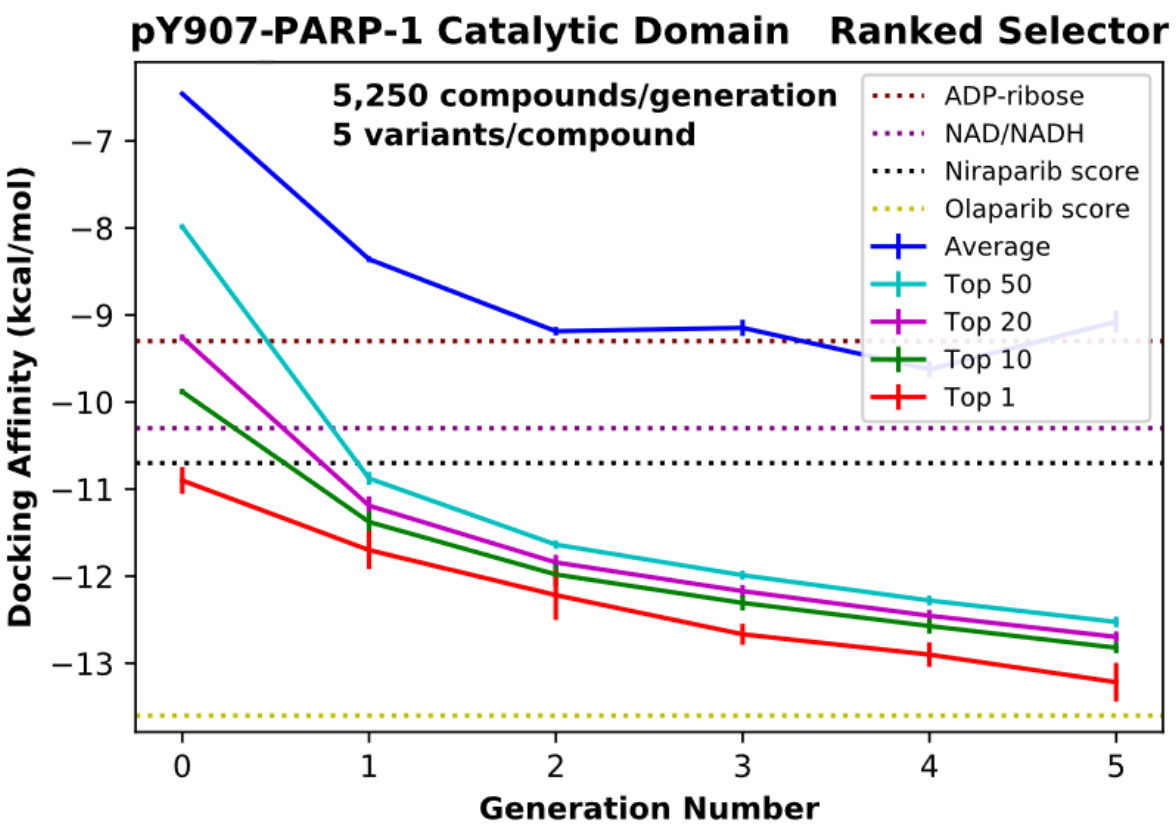


Figure 22. A plot of the QVina2 scores of six PARPi lead-optimization runs applied to pY907-PARP-1.

Generation zero is the initial seed population consisting of 11 PARPi and 83 PARPi fragments. Generations one to five are the AutoGrow4-generated populations. For each generation, the grand mean of all QVina2 scores across all six runs is shown in blue. The grand means of the top 50, 20, 10, and 1 compounds are shown in cyan, purple, green, and red, respectively. Error bars represent the standard deviations for the six independent runs. For ease of comparison, the dotted lines represent the docking scores of select compounds docked into the nonphosphorylated PARP-1 structure (4R6E:A).

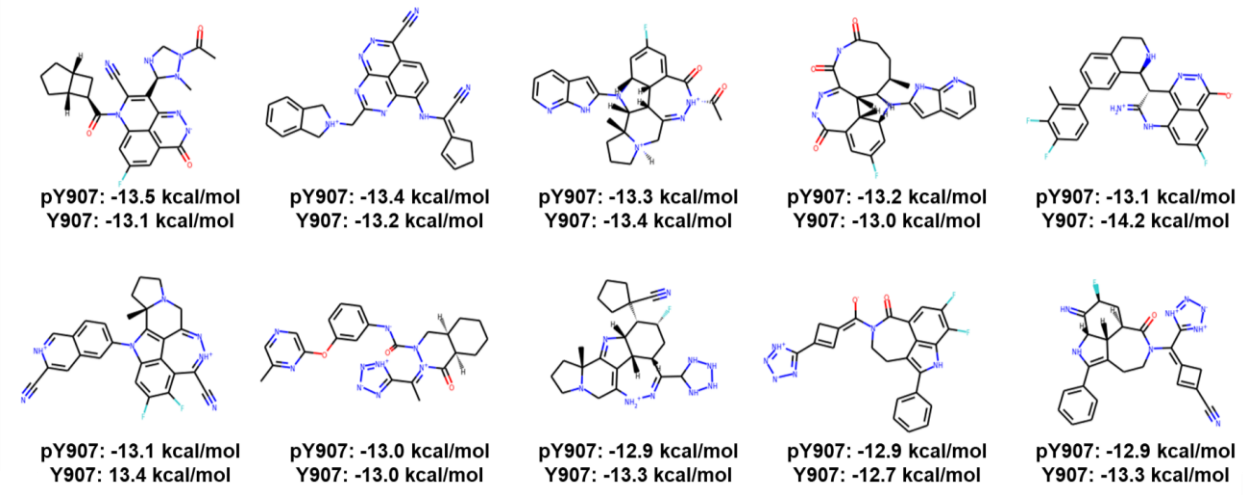


Figure 23. The ten best scoring compounds from the pY907-PARP-1 lead-optimization runs.

The docking scores were determined by exhaustively docking the compound into the pY907-PARP-1 (top score) and the nonphosphorylated PARP-1 (bottom score) structures. Docking scores were determined using QVina2.

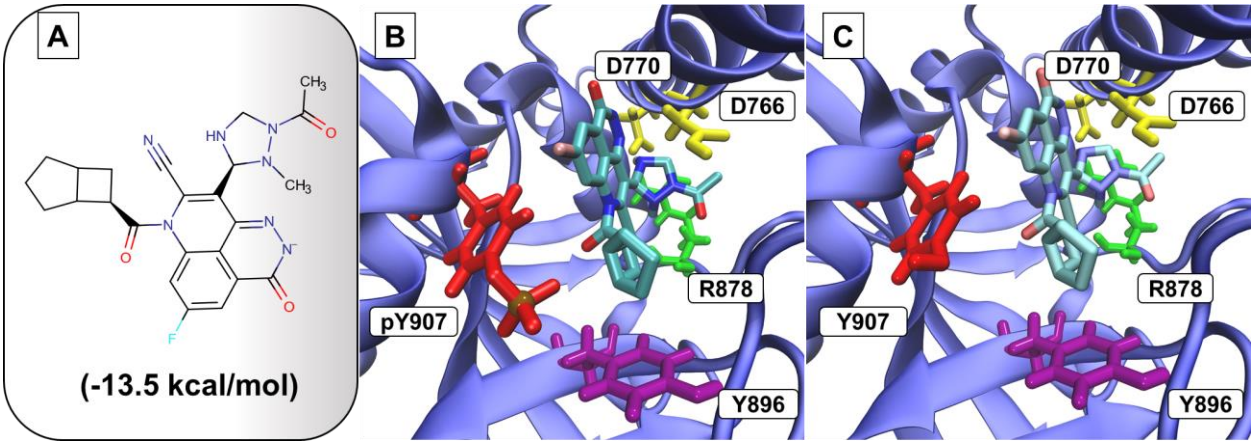


Figure 24. The best-scored compound of the pY907-PARP-1 PARPi lead-optimization runs. A) A 2D representation of the best-scoring compound. The docking score was determined by QVina2. B) The docked pose of the best-scoring AutoGrow4-generated compound docked into pY907-PARP-1. The phosphorous atom of pY907 is shown in brown. C) The same compound re-docked into the nonphosphorylated PARP-1 catalytic domain using QVina2. Select residues are shown in colored-stick representation.

Table 11. Protein-ligand interactions of the 100 best-scored compounds from the pY907-PARP-1 lead-optimization runs.

This table is limited to the single best-scored pose per compound and only includes interactions detected using BINANA (version 1.1.2), default settings. Infrequent interactions (< 10%) are excluded from the table. All values are given as percentages.

	H862	R878	pY907
Cation-π	1	0	2
Hydrogen bond	4	13	3
Electrostatic	0	0	1
T-stacking	0	0	0
π-π	20	0	24

3.3.5 AutoGrow4 Applied to a Non-catalytic PARP-1 Pocket

As discussed in "Chapter 3.3.4: AutoGrow4 Applied to PARP-1 with Phosphorylated Y907 (pY907)," all current PARPi bind the PARP-1 catalytic pocket and thus are vulnerable to the same resistance mechanisms. PARPi that bind PARP-1 at pockets other than the catalytic site might serve to address this vulnerability. In this section, I describe a preliminary search for non-competitive PARP-1 inhibitors. I will begin by using computational hot-spot mapping and prior research into PARP-1 structure and function (e.g., about the known effects of mutations at specific sites and the mechanism of allosteric regulation through interdomain interactions) to identify druggable PARP-1 pockets. Then, I will use AutoGrow4 to predict inhibitors that bind to one of these non-catalytic pockets, at a DBD pocket located near the Zn1-Zn3 interface. Lastly, I will compare the compounds predicted to bind the DBD with those predicted to bind the catalytic domain.

3.3.5.1 Selecting a Druggable Non-Catalytic Pocket

To identify druggable PARP-1 pockets, I applied the computational hot-spot mapping technique FTMap[243] to the most complete, high-resolution PARP-1 structure (PDB: 4DQY[45]), which includes a co-crystallized DNA molecule. FTMap identified five hot spots, excluding those positioned on the DNA helix (Figure 25). Of the five, two are located at the Zn1-Zn3 interface (Figure 25, top left), one is located at the Zn1-WGR-HD interface (Figure 25A, bottom center), and two are located in the CAT (one in the center of the ART subdomain, the other near the αF-αJ

interface; Figure 25A, top right). Given the goals of my project, I focused on the Zn1-Zn3 and Zn1-WGR-HD hot spots because the two CAT hot spots both occupy the catalytic binding site, where existing inhibitors bind. The two hot spots located near the Zn1-Zn3 interface are close to each other (Figure 25), so I will treat them as a single site for AutoGrow4-guided CADD.

Both the Zn1-Zn3 and Zn1-WGR-HD interfaces are promising druggable targets because disrupting these interfaces could prevent catalytic activity without influencing DNA binding. As discussed in "Chapter 1.1.2.5: PARP-1 Catalytic Activation is Modulated by Interdomain Interactions," both interfaces are involved in interdomain communication, and disruption to either interface reduces PARP-1 catalytic activity without altering DNA-binding activity[45,46,70]. For instance, both the W246A (Zn1-Zn3 interface) and W318R (Zn1-WGR-HD interface) mutants maintained near-wildtype levels of DNA-binding activity despite having dramatically reduced catalytic activity[45,46,70]. I reason that a small molecule capable of disrupting either the Zn1-Zn3 or Zn1-WGR-HD interface could similarly limit catalytic activity without preventing PARP-1 trapping on DNA, which is important because the effectiveness of current PARPi at treating HR- and BER-deficient cancer cells is related to the combined effects of inhibiting PARP-1 activity and trapping the inhibited PARP-1 on DNA[75,77]. Pockets near both interfaces are thus good candidate sites for drug inhibition of PARP-1.

To maximize the chance of identifying useful candidate PARPi, I evaluated each pocket in terms of small-molecule druggability. One concern was that both hot spots exist at interdomain interfaces. Disrupting protein-protein interfaces with small molecules has been historically challenging because protein-protein binding typically occurs on flat, large surfaces that interact via many weak interactions[256]. In contrast, small-molecule/protein binding typically occurs inside

deep concavities on the protein surface, allowing the small molecule to make several strong interactions[256]. I will select the most druggable hot spot for AutoGrow4-guided CADD to maximize the chance of finding good candidate inhibitors. I will also analyze the properties of the niraparib-bound PARP-1 structure (PDB: 4R6E:A), which serves as a reference.

The Zn1-Zn3 and Zn1-WGR-HD pockets have many similar physicochemical properties, but I predict that the Zn1-Zn3 pocket is more druggable (Table 12). Several factors motivate this prediction. First, larger and deeper pockets are typically considered more druggable[257], and the volume of the Zn1-Zn3 pocket is larger than that of the Zn1-WGR-HD pocket, although both are less than half the size of the catalytic pocket (Table 12). Second, DoGSiteScorer[249], a program that assesses pocket druggability (i.e., the likelihood that small molecules bind), suggested that the Zn1-Zn3 pocket is more druggable than the Zn1-WGR-HD pocket (Drug Scores, unitless, of 0.14 and 0.04, respectively) (Table 12); the catalytic pocket (positive control) scored considerably better than both (Drug Score of 0.57) (Table 12). Because DoGSiteScorer is closed source, the details regarding its scoring algorithm are unavailable[249]; however, DoGSiteScorer does output physicochemical properties of the analyzed pockets, which can indicate why a pocket scored as it did[249]. Unlike the Zn1-Zn3 and Zn1-WGR-HD hot spots, the catalytic pocket is less densely packed with protein residues (Table 12), and the number of aromatic atoms is much higher in both the Zn1-Zn3 and catalytic pocket (10.84% and 11.95% of amino acids, respectively) compared to the Zn1-WGR-HD pocket (4.71% of amino acids) (Table 12). Pockets with more aromatic residues provide more opportunities for protein/ligand stacking interactions. The number of hydrogen-bond donor and acceptor atoms, as well as the hydrophobicity, were all comparable among the three pockets (Table 12).

I selected the hot spot located near the Zn1-Zn3 interface for further exploration. This hot spot has a larger pocket volume, higher density of aromatic groups, and is ranked more druggable by DoGSiteScorer than the hot spot near the Zn1-WGR-HD interface.

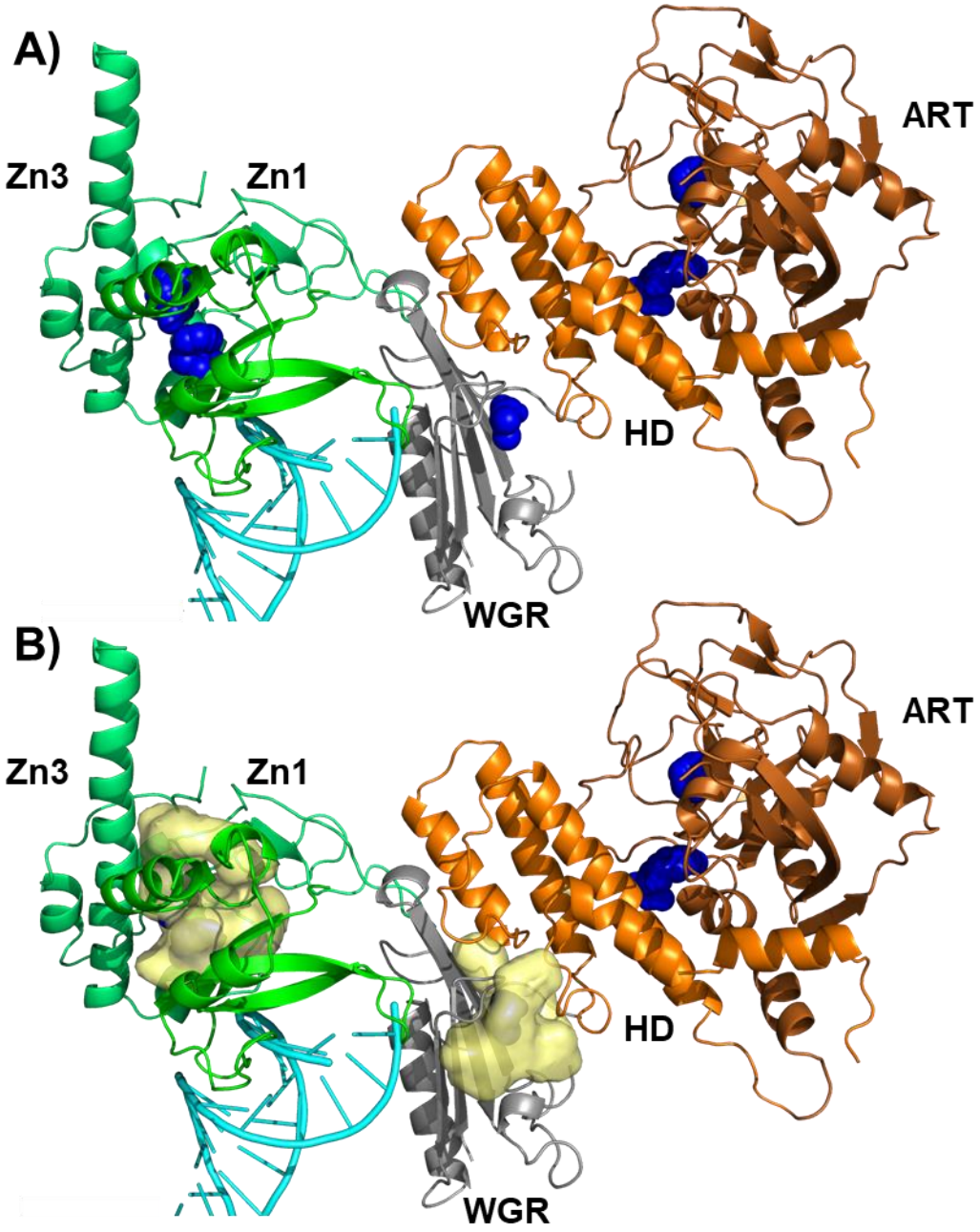


Figure 25. FTMap-detected PARP-1 hot spots (PDB: 4DQY).
A) The molecular fragments used to identify hot spots are shown as blue spheres. Hot spots were detected between the interface of Zn1 (green) and Zn3 (light green), the interface of Zn1 WGR (gray) and HD (light orange), and the interface of HD and ART (brownish orange). DNA is shown in cyan. There are two hot spots at the Zn1-Zn3 interface and two in the catalytic pocket. B) The pocket volumes of the Zn1-Zn3 interface hot spot (top left) and the Zn1-WGR-HD interface hot spot (bottom center) are shown as transparent yellow surfaces.

Table 12. Physicochemical properties of hot-spot pockets.

The physicochemical properties of two FTMap hot spots near the Zn1-Zn3 and Zn1-WGR-HD interfaces (PDB: 4DQY). "CAT" indicates the catalytic pocket of PARP-1 (PDB: 4R6E:A), which serves as a positive control. "Vol." indicates the volume of each pocket as predicted by either POVME[248] (version 2.1) or DoGSiteScorer[249] (version 2.0.0). "# Atoms" indicates the number of heavy atoms (i.e., non-hydrogen atoms) within the pocket volume. "HD; HA; Aro." indicates the number of hydrogen-bond donor, hydrogen-bond acceptor, and aromatic atoms within the pocket volumes, respectively. "Total (# AA)" indicates the number of amino acids; "Pos; Neg; Polar; Non" indicates the number of positively charged, negatively charged, polar, and nonpolar amino acids within the pocket volume. "Drug Score" is DoGSiteScorer's final druggability score for each pocket, with 0.0 being the least druggable and 1.0 being the most druggable. Both the Drug Score and the Hydrophobicity Ratio are unitless.

	POVME	DoGSiteScorer						
Pocket	Vol. (Å^3)	Vol. (Å^3)	Total (# Atoms)	HD; HA; Aro. (# Atoms)	Total (# AA)	Pos; Neg; Polar; Non (# AA)	Hydrophobicity Ratio	Drug Score
Zn1-Zn3	537	634	369	67; 56; 40	72	13; 11; 20; 28	0.68	0.14
Zn1-WGR-HD	428	595	361	67; 52; 17	79	6; 8; 25; 39	0.693	0.04
CAT (4R6E:A)	1464	1262	251	43; 46; 30	47	6; 7; 18; 16	0.673	0.57

3.3.5.2 AutoGrow4 Applied to the PARP-1 DBD

In this section, I describe the results of AutoGrow4-guided CADD targeting the DBD. As described in "Chapter 3.3.5.1: Selecting a Druggable Non-Catalytic Pocket," I selected a hot spot for drug targeting located near the Zn1-Zn3 DBD interface. I ran three independent AutoGrow4 runs for ten generations each using the same settings and source compounds as the large-scale *de novo* PARPi runs described in "Chapter 3.3.1: Large-Scale *de novo* PARPi Run." I ran the large-scale catalytic-domain-targeted *de novo* lead-generation run twice more, using the same conditions, to provide a fair comparison to the three AutoGrow4 runs targeting the DBD.

My goal in applying AutoGrow4 to the DBD is to predict novel drug-like compounds that impact the PARP-1 Zn1-Zn3 interface, avoiding PARPi resistance mechanisms such as that caused by Y907 phosphorylation (see "Chapter 1.1.3.4: PARPi Resistance Mechanisms").

3.3.5.2.1 Predicted Zn1-Zn3-Interface Ligands

After ten generations, the three AutoGrow4 runs produced many compounds with strong predicted binding affinities (Figure 26). The best-scored compound, Compound 6, had a QVina2 score of -14.0 kcal/mol, which surpasses the predicted affinities of known PARPi that bind the catalytic pocket (e.g., the crystal structure of olaparib had a QVina2 score of -12.2 kcal/mol to the catalytic pocket; see Table 10 p.160). However, the produced compounds had weaker predicted DBD binding affinities than the novel compounds produced during the large-scale, catalytic-pocket *de novo* AutoGrow4 runs. For instance, by the tenth generation, the top 1,000 compounds

of the three catalytic-site runs had an average docking score of -14.34 kcal/mol and the top 50 compounds had an average score of -14.98 kcal/mol. In contrast, the top 1,000 and top 50 compounds of the three DBD runs had average scores of -12.67 kcal/mol and -13.40 kcal/mol, respectively.

The best-scoring DBD compounds consist primarily of elongated small molecules with multiple small cyclic groups, which are frequently connected by one or more rotatable bonds (Figure 26B). For instance, the best-scoring compound (Compound 6, top-left of Figure 26B) contains four ring systems containing a total of six rings, four of which are aromatic. These are connected by four rotatable bonds, which create two stereocenters (Figure 26B). Compared to the candidate ligands that target the catalytic pocket, the compounds predicted to bind the Zn1-Zn3 interface tend to be more flexible (i.e., the unitless ratio of rotatable to rigid bonds is higher). The top 1,000 compounds from the tenth generation of the DBD and CAT runs having flexibility scores of 0.16 and 0.09 respectively (Table 13).

Additionally, despite having a comparable ring count and MW, the CAT-targeted compounds tend to have much larger ring systems than the DBD-targeted compounds; the CAT-targeted tenth generation's top 1,000 compounds on average had 14.37 heavy atoms in each compound's largest ring system. In contrast, the DBD-targeted tenth generation's top 1,000 compounds averaged 9.22 heavy atoms in the largest ring system per compound (Table 13). This indicates that compounds that are flexible and elongated are amenable to high-affinity DBD binding, and that compounds that score well in the CAT can be more compact and contain large multicyclic groups. The compactness of the CAT-targeted compounds suggests that high-affinity binding in the catalytic domain can be achieved by a few strong interactions or by a localized,

dense cluster of residues that allow many interactions in a small region. I favor the former explanation because in both the lead-generation and PARPi lead-optimization experiments, I observed a limited number of interactions with other residues ("Chapters 3.3.1-3.3.4"). Many of these protein-ligand interactions were strong stacking interactions or electrostatic interactions, such as the prominent π-π stacking with Y907 (Table 8 and Table 9 on p.152 and p.155, respectively). On the other hand, the elongation, flexibility, and the smaller cyclic groups observed in the DBD-targeted compounds suggest that high-affinity binding to the DBD typically involves multiple weak interactions and/or that compounds be flexible enough to fit pocket contours. I will explore both options in the following section.

Lastly, as a proof of principal, these experiments show that AutoGrow4 evolves unique compounds that are customized to a respective protein pocket. The DBD-targeted and CAT-targeted AutoGrow4 runs produced distinctly different compounds, despite being derived from the same source compounds/fragments. Of course, not all protein pockets are amenable to drug-inhibition. However, even though DoGSiteScorer deemed the DBD less druggable than the catalytic pocket, AutoGrow4 managed to find novel compounds that docked well for both.

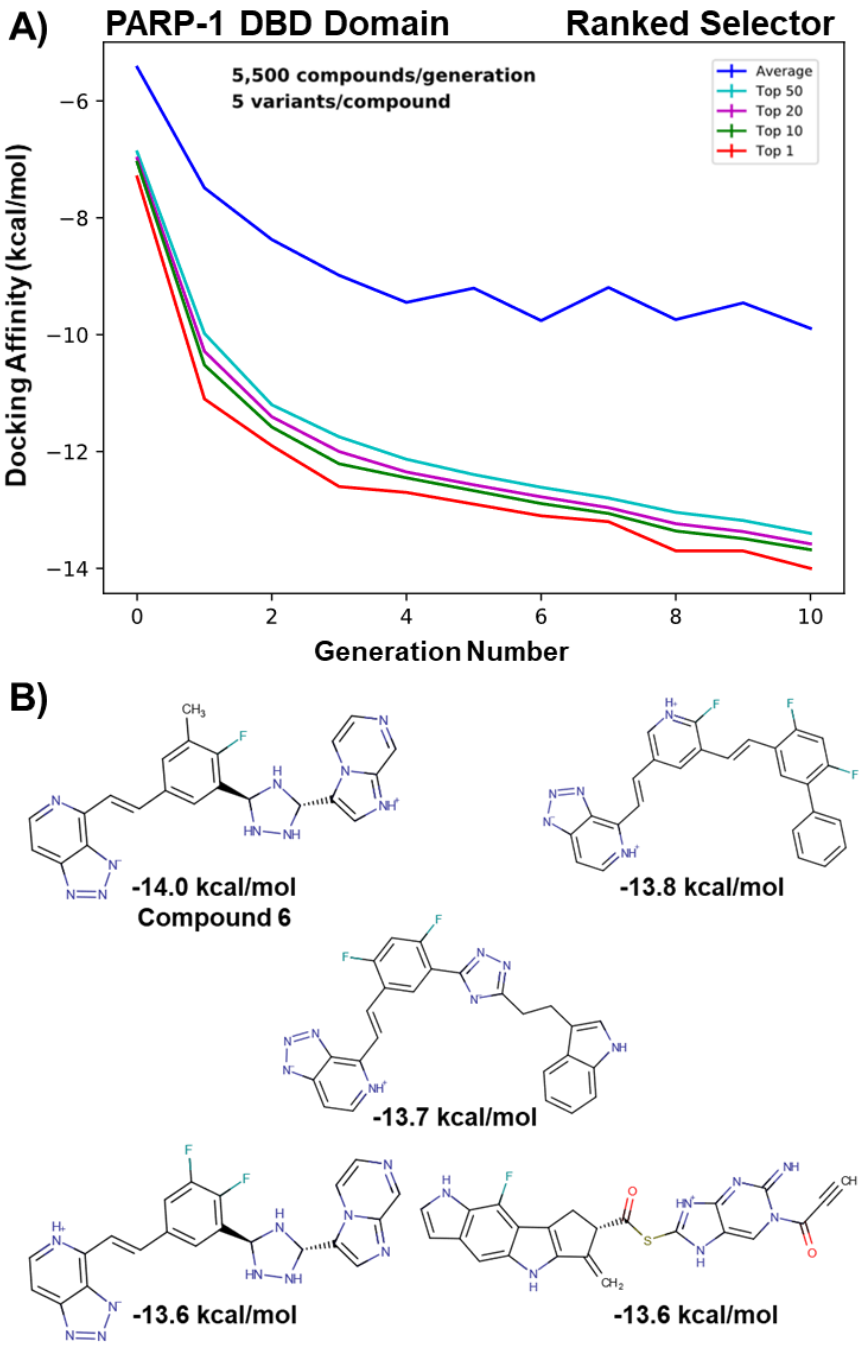


Figure 26. Results of AutoGrow4 runs applied to the DBD.

A) A plot of the three AutoGrow4 runs applied to the Zn1-Zn3 interface. For each generation, the mean of all QVina2 scores across all three runs is shown in blue. The average of the top 50, 20, 10, and 1 compounds are shown in cyan, purple, green, and red, respectively. B) The five best-scoring compounds generated during the three AutoGrow4 runs.

Table 13. Chemical properties of the best 1,000 compounds from the three AutoGrow4 runs targeting the DBD and CAT.

"DBD" indicates the AutoGrow4 runs targeting the Zn1-Zn3 interface, and "CAT" indicates the AutoGrow4 runs targeting the catalytic site. "Gen. #" indicates the AutoGrow4 generation number. Generations are the three independent runs pooled, with redundant compounds removed. "% Pop." represents the percent of scores that were included (100 * N / Size), where N is 1,000. "Flexibility" is the ratio of the number of rotatable bonds ("# Rot. Bonds") to rigid bonds. "Max. Ring System Size" is the number of heavy atoms that the largest ring system of a given compound contains. "# Rings" indicates the total number of rings contained within a given compound and considers each ring in multicyclic groups to be an independent ring; "# Aromatic Rings" indicates the number of aromatic rings; "# Stereocenter" indicates the number of stereocenters a given compound contains, and "MW" indicates molecular weight in Daltons. All measurements were calculated using the Python program Scopy[166]. "Dev." represents the standard deviation of the included scores.

	Size	Flexibility		# Rot. Bonds		Max. Ring System Size		# Rings		# Aromatic Rings		# Stereocenter		MW	
	% Pop.	Mean	Dev.	Mean	Dev.	Mean	Dev.	Mean	Dev.	Mean	Dev.	Mean	Dev.	Mean	Dev.
DBD															
Gen. 0	11.3	0.14	0.14	0.99	0.79	6.52	1.64	1.38	0.54	0.92	0.67	0.41	0.84	144.2	4.90
Gen. 1	35.3	0.27	0.19	3.62	1.90	7.35	1.67	2.44	0.77	1.98	0.88	0.67	1.06	275.9	47.1
Gen. 5	7.9	0.20	0.07	5.37	1.46	8.78	1.85	4.79	0.90	3.85	1.15	1.48	1.69	442.8	30.1
Gen. 10	7.4	0.16	0.05	4.55	1.15	9.22	2.31	5.17	0.84	4.03	1.19	1.99	1.75	456.7	19.1
CAT															
Gen. 0	10.8	0.13	0.12	0.90	0.73	6.68	1.52	1.31	0.49	1.08	0.54	0.16	0.52	142.4	6.3
Gen. 1	34.1	0.19	0.16	2.81	2.15	8.10	1.86	2.51	0.73	2.02	0.83	0.39	0.81	259.5	62.3
Gen. 5	8.1	0.13	0.05	3.88	1.35	10.93	2.55	5.48	0.88	4.25	1.13	1.42	1.53	448.2	23.7
Gen. 10	8.0	0.09	0.04	2.87	1.15	14.37	2.98	5.93	0.81	4.20	0.99	3.39	2.31	460.9	14.7
Gen. 15	8.0	0.07	0.03	2.31	1.03	14.87	2.78	6.02	0.65	4.09	0.82	3.92	2.03	465.4	11.2
Gen. 20	8.0	0.07	0.03	2.27	0.89	13.90	2.79	5.97	0.63	4.16	0.88	3.2	1.90	468.8	8.90
Gen. 25	8.3	0.07	0.03	2.33	0.93	13.36	2.23	6.12	0.66	4.48	0.96	2.59	1.74	471.2	7.30
Gen. 30	7.9	0.07	0.03	2.37	0.96	12.85	1.64	6.27	0.73	4.66	1.02	2.23	1.52	472.2	6.40

3.3.5.2.2 Predicted Protein-Ligand Interactions Within the DBD Pocket

In this section, I will further examine the binding of candidate Zn1-Zn3-targeted ligands by analyzing the predicted protein-ligand interactions of the 100 best-scoring AutoGrow4-generated compounds. I will also consider the docked pose of the best-scoring compound at the pocket near the Zn1-Zn3 interface. These analyses will provide insight into the interactions that contribute to predicted high-affinity binding to this pocket.

To assess the interactions frequently involved in predicted DBD protein-ligand binding, I applied the BINANA algorithm (version 1.1.2) to the top 100 best-scoring compounds from the three DBD AutoGrow4 runs (Table 14). I also ran a similar analysis on the best-scoring 100 compounds that target the CAT (Table 14). The CAT-targeted compounds formed more stacking interactions (i.e., π-π, cation-π, and T-stacking interactions) with PARP-1 than did the DBD-targeted compounds (Table 14). In contrast, the DBD-targeted compounds formed more hydrophobic interactions (10.54 hydrophobic interactions per compound) than did the CAT-targeted compounds (7.58 hydrophobic interactions per compound) (Table 14). Both the DBD and CAT compound sets formed a comparable number of hydrogen bonds: 2.45 and 2.13 per compound, respectively (Table 14).

Although stacking interactions were less common, the best-scoring DBD-targeted compounds did frequently form π-π interactions with the Zn3 residue W246 (Table 15). As discussed in "Chapter 1.1.2.5: PARP-1 Catalytic Activation is Modulated by Interdomain Interactions," W246 forms a contact with the Zn1 residue R78 that is important for DNA-dependent catalytic activity (Figure 5 p.22). W246 mutations disrupt the DNA-dependent

formation of the Zn1-Zn3 interface, thereby preventing virtually all PARP-1 DNA-dependent catalytic activity without altering DNA-binding activity[45,46,52,70]. Small molecules capable of disrupting the Zn1-Zn3 interaction, perhaps by stacking with W246, are thus promising because they may have similar effects.

Next, I analyzed the best-scoring compound produced by the AutoGrow4 DBD runs (Compound 6) to determine how the topology of the pocket influenced compound generation and binding. First, there is a protruding helix at the interface that separates two surface-accessible protein cavities (Figure 27A and B). For a small molecule to bind to both cavities it must either be flexible enough to bend around the protrusion and/or be angled. In the case of Compound 6, the two moieties that branch from the central 1-fluoro-2-methylbenzene are positioned meta to one another. This provides the necessary 120° angle that allows Compound 6 to straddle the protrusion and extend into both cavities (Figure 27C). This protrusion likely accounts for the shape of the best-scoring AutoGrow4 DBD compounds. Additionally, Compound 6 forms more hydrogen-bond and stacking interactions with the Zn1 than with the Zn3 (Figure 27C), whereas the top 100 best-scoring compounds form a comparable number of interactions with both domains (Table 15). Given the conformational flexibility of the pocket itself as well and these two zinc fingers generally[70], it would be useful for future DBD-targeted CADD studies to reevaluate this pocket using multiple conformations.

Table 14. The average number of protein-ligand interactions per compound.
"DBD" indicates the 100 best-scoring compounds produced by the three AutoGrow4 runs applied to the pocket near the Zn1-Zn3 interface. "CAT" indicates the 100 best-scoring compounds produced by the three AutoGrow4 runs targeting the catalytic site. Reported values are the average number of each interaction type per compound, as detected by the BINANA algorithm.

	Average # of Interactions of the Top 100 Compounds	
Interaction	**DBD**	**CAT**
Hydrogen Bonds	2.45	2.13
Hydrophobic	10.54	7.58
Cation-π Stacking	0.11	0.98
π-π Stacking	0.13	2.04
T-Stacking	0.33	0.71
Electrostatic	0.06	1.53
Total:	**13.62**	**14.97**

Table 15. Protein-ligand interactions of the 100 best-scoring compounds from the AutoGrow4 runs applied to the DBD.

This table is limited to the single best-scoring pose per compound and only includes interactions detected by BINANA (version 1.1.2) using default settings. Infrequent interactions (< 10%) are excluded for readability, as are hydrophobic interactions. All values are given as percentages.

Subdomain	**Zn1**				**Zn3**				
Residue	**F74**	**S75**	**Q82**	**K86**	**Q241**	**N242**	**W246**	**R282**	**R355**
Cation- π	0	0	0	0	0	0	0	0	11
Hydrogen bond	0	49	61	11	11	37	0	17	33
Electrostatic	0	0	0	0	0	0	0	0	0
T-stacking	20	0	0	0	0	0	0	0	0
π-π	0	0	0	0	0	0	12	0	0

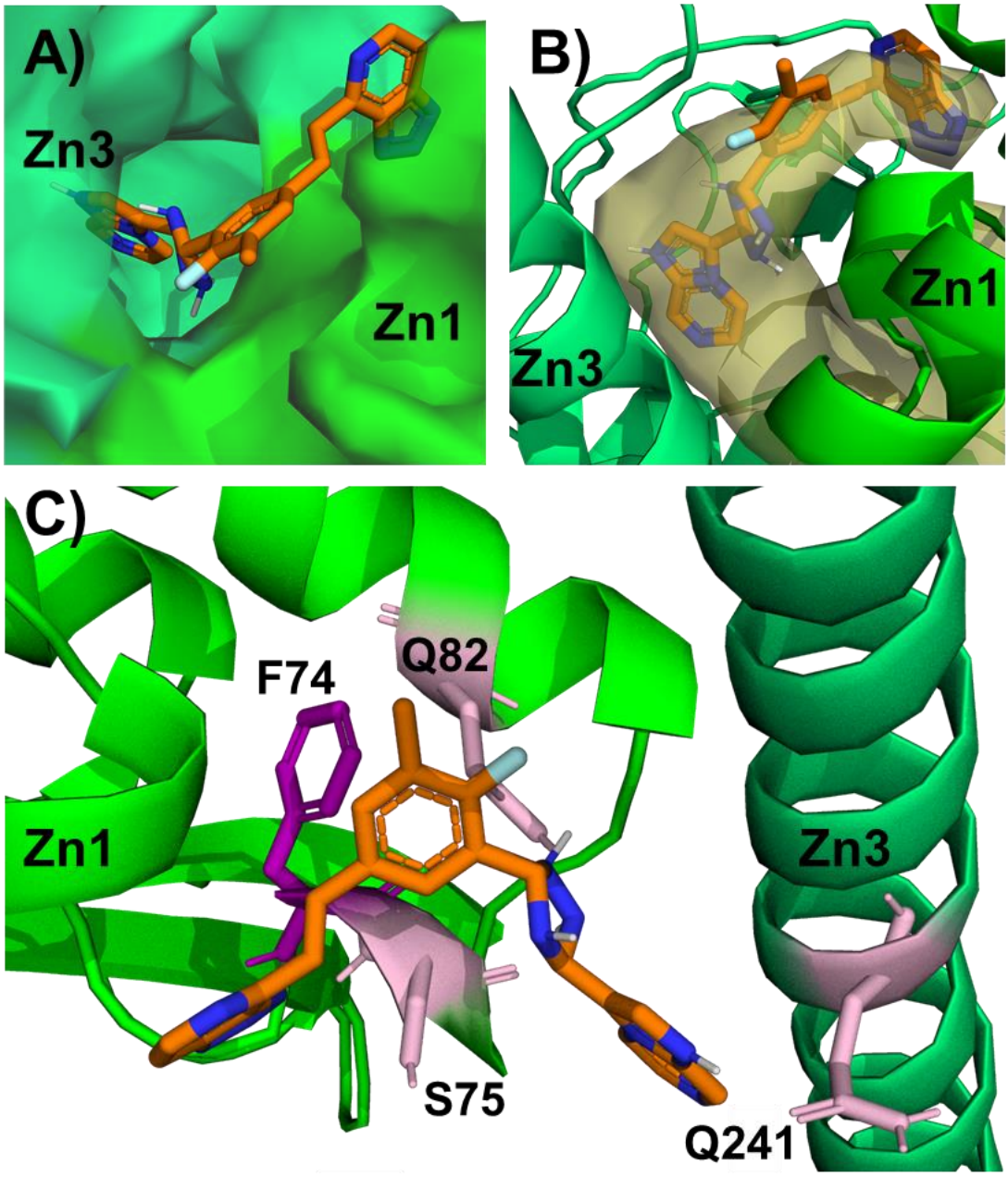


Figure 27. The best-scoring compound at the Zn1-Zn3 interface.
A) A surface representation of the protein structure shows a protrusion at the interface of the Zn1 (lime green) and Zn3 (light green). This protrusion requires the best-scoring AutoGrow4-generated DBD compound, Compound 6 (orange stick representation), to bend to form contacts with both the Zn1 and the Zn3. B) The pocket volume as predicted by POVME[248] (version 2.1) (transparent yellow surface) shows how Compound 6 bends to fit the contours of pocket. C) Compound 6 forms a T-stacking interaction with F74 (purple) and hydrogen bonds with S75, Q82, and Q241 (pink).

3.4 Conclusions

In addition to discussing the significant improvements that I've added to AutoGrow4's performance and features, I have shown how AutoGrow4 can be used as a hypothesis-generation tool by demonstrating how it can recapitulate many protein-ligand interactions observed in crystal structures of known PARPi. In fact, the highest frequency interactions among well-scoring AutoGrow4-generated compounds were with residues known to be critical for PARP-1 catalytic activity, PARPi binding, and potential PARPi resistance mechanisms[17,18]. Further runs testing pY907-PARP-1 found several potential leads that are predicted to bind to the catalytic pocket of PARP-1 with better affinities than known PARPi, both when Y907 is phosphorylated and when it is not. This analysis of AutoGrow4-generated compounds targeting the PARP-1 catalytic domain both validates AutoGrow4's utility and raises concern about the future of PARP-1 orthosteric inhibition.

Additionally, my application of AutoGrow4 to the Zn1-Zn3 interface resulted in a set of flexible, highly customize candidate ligands that are predicted to bind with high affinity. *In vitro* and *in vivo* testing is required to further evaluate these compounds, and the conclusions of this study could be different if I had applied AutoGrow4 to a different conformation of the DBD. However, this study offers preliminary insights into the properties that affect protein-ligand binding in this pocket.

AutoGrow4 also generated many compounds with better predicted binding affinities than known PARPi when applied for both *de novo* design and lead optimization. Taken together, the results of these experiments indicate that AutoGrow4 is effective at predicting well-scoring drug-

like compounds that complement the targeted pocket, exploit important protein-ligand interactions, and can inform future experiments.

3.5 Acknowledgments

I would like to thank the following people for their contributions to the AutoGrow4 codebase and the analysis. Dr. Jacob Durrant generated the original idea for revising his line of AutoGrow software. He provided guidance in coding, experimental design, co-authoring the AutoGrow4 publication, editing this chapter of the dissertation, testing AutoGrow4 on macOS, and generating high-resolution versions of figures for the publication and this dissertation.

I would like to thank Dr. Bennett Van Houten for his suggestion that I apply AutoGrow4 to PARP-1.

Additional contributions to AutoGrow4 include: Patrick J. Ropp for useful discussions and programming tips; Erich Hellemann for discussions and help with SMARTS reactions; Pauline Spiegel for manuscript editing and tutorial testing; Yuri Kochnev for compiling QVina2 for macOS; Kevin C. Cassidy for help with molecular rendering; and Harrison Green for help developing AutoGrow4 accessory scripts. I would also like to thank the University of Pittsburgh's Center for Research Computing for providing helpful computer resources. The default fragment libraries included with AutoGrow4 were derived from a subset of the ZINC database (https://zinc.docking.org/). I thank ZINC for allowing me to distribute these fragment libraries to AutoGrow4 users.

3.6 Author Contributions

Jacob O. Spiegel, wrote and tested the entire AutoGrow4 codebase and all accessory scripts provided in the AutoGrow4 download. Many libraries and other programs are leveraged and have been cited throughout the code and all related publications. I designed and performed many of the experiments discussed in the paper. Dr. Jacob Durrant and I authored the publication of AutoGrow4. Dr. Durrant also contributed to data analysis. Only sections that I wrote were used verbatim in this dissertation. All writing in this chapter is original content written by Jacob O. Spiegel.

4.0 Comparison of CADD Techniques

In this chapter I compare the AutoGrow4-guided PARP-1 lead-optimization runs from "Chapter 3.3.2: PARP-1 Lead Optimization" to a lead-optimization-style virtual screen (VS), as well as lead-optimization runs performed with three other *de novo* CADD programs.

AutoGrow4 was published under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium."[6] This chapter contains work that is adapted and reprinted with rights and permission:

> **Jacob O Spiegel**†, & Jacob D Durrant. (2020) AutoGrow4: An open-source genetic algorithm for *de novo* drug design and lead optimization. *Journal of Cheminformatics*, 12, 25. https://doi.org/10.1186/s13321-020-00429-4.
>
> † Jacob O. Spiegel should be regarded as first author.

I am the first author of the published manuscript, having written the entire AutoGrow4 codebase, performed all AutoGrow4 runs discussed in the paper, and analyzed the data. I designed the initial layout for all figures in the paper with Dr. Jacob Durrant, who then generated and refined the images. I am solely responsible for the additional figures that appear exclusively in this dissertation. I performed all tests and analyses required to compare AutoGrow4 with the three alternative *de novo* programs. Dr. Durrant performed and analyzed the VS experiment from the publication. Dr. Durrant also provided guidance and insight as described in the acknowledgement section. All writing in this chapter is original content written by Jacob O. Spiegel.

4.1 Overview and Rationale

I have previously shown that AutoGrow4 is useful for early-stage lead discovery and optimization, so in this chapter I will compare AutoGrow4 to alternative lead-refinement/optimization approaches. I compare AutoGrow4's ability to optimize a set of lead compounds (known PARPi) with that of (1) a more conventional virtual screening (VS) lead-optimization approach[258,259] and (2) three other *de novo* CADD programs. An additional review of VS and *de novo* CADD programs is provided in "Chapter 1.2.2: Computer-Aided Drug Design (CADD)."

As a high-throughput *in silico* technique, AutoGrow4 is most useful during the early stages of drug design when the goal is to find a pool of drug-like candidate compounds that can then be further examined using more costly techniques. Although many factors determine the quality of a drug candidate[175] (e.g., bioavailability, solubility, toxicity, and specificity), the predicted binding affinity is commonly used to evaluate compounds during early-stage drug development[155,156,181–185,163,168–174]. Accordingly, AutoGrow4 relies on docking affinity for selecting initial candidate compounds.

VS CADD is one of the most common alternatives to *de novo* CADD for early stage-drug design[183] ("Chapter 1.2.2: Computer-Aided Drug Design (CADD)"). Unlike AutoGrow4, which creates novel *in silico* ligands, VS tests a dataset of ligands that are curated before the start of the screen[258,259]. Like AutoGrow4, VS frequently uses molecular docking to assess compounds for potential binding[183–185]. Additionally, AutoGrow4 and VS can both be used for lead generation and lead optimization[183,258,259]. VS lead optimization typically includes generating a library of

compounds that are similar to the lead compounds before screening for the compounds with the best docking scores compounds[258,259]. For the VS lead-optimization run described in this thesis, we developed an *in silico* small-molecule library comprised of compounds from PubChem[260] that are structurally similar to known PARP inhibitors. This library was prepared with Gypsum-DL[214] and then docked with QVina2[205]. The QVina2 docking score was used to assess the predicted accuracy of each compound's binding to the structure of the PARP-1 catalytic pocket, described in "Chapter 2.3.1: Receptor File Preparation".

I also selected three *de novo* CADD programs (*de novo* DOCK[162], GANDI[167], and LigDream[213]) to compare to the AutoGrow4 lead-optimization runs. I limited my comparison to free, open-source, and free-for-academic-use programs, which eliminated the program MoleGear[211]. Additionally, I limited the comparison to programs that are fully functional without modifying their source code[162,167,213], which excluded the program REINVENT[212]. I tested *de novo* DOCK[162], GANDI[167], and LigDream[213] using lead optimization rather than lead generation because LigDream is strictly a lead-optimization technique[213]. For these runs, each program started from the same set of PARPi and PARPi fragments used in the AutoGrow4 PARPi lead-optimization runs. All runs targeted the 4R6E:A catalytic pocket ("Chapter 2.3.1: Receptor File Preparation"), with the exception of LigDream which does not use a protein structure[213]. I evaluated the resulting compounds in terms of predicted binding affinity, diversity, synthesizability, drug-likeness, and ADME-PK properties.

4.2 Methods

4.2.1 Comparison of AutoGrow4 PARPi Lead Optimization and VS Lead Optimization

In this section I compare the AutoGrow4-guided PARPi lead-optimization runs with a traditional lead-optimization VS. In this context, the goal is not to find novel drugs; rather, it is to find a set of candidate compounds that are structurally similar to known PARP inhibitors but with improved docking scores. As described in "Chapter 3.0: AutoGrow4: Application to Poly (ADP-ribose) Polymerase-1 (PARP-1)," by focusing our search on compounds that are similar to known inhibitors, we spend fewer computational resources testing compounds that are unlikely to be viable drugs.

4.2.1.1 VS PARPi Lead Optimization

To perform the lead-optimization VS, we developed an *in silico* library of PARPi-like compounds. We first downloaded 2,184 unique PARPi SMILES from the BindingDB dataset[261,262]. There were many close analogs in this dataset, so we removed redundancies by clustering the compounds into 40 groups using a Tanimoto-based clustering algorithm[263], with a Tanimoto coefficient cutoff of 0.65. The Tanimoto coefficient is a comparison of two bit-strings (i.e., digital representations of small molecules as fingerprints) that is calculated by dividing the number of shared bits (digits in the bit-strings) by the total number of bits in the two bit-strings[263]. The most central compound of each group (i.e., the compound with the most neighbors) was chosen to represent each cluster. We then downloaded at most 250 similar compounds for each of

these 40 PARPi[260], selecting compounds that had Tanimoto coefficients greater than 0.80 with respect to a given PARPi. This process yielded 8,444 unique molecules. We again applied the clustering algorithm (with a Tanimoto cutoff of 0.2) to reduce the list to a set of 4,657 PARPi-like compounds that were not in the initial PARPi BindingDB dataset.

These resulting compounds were then prepared for docking following the same procedures and settings as the AutoGrow4 PARPi lead-optimization runs ("Chapter 3.2.2: PARPi Lead Optimization" and Appendix JSON 6). First, the compounds were converted to PDB files using Gypsum-DL (version 1.1.2)[214], then the PDB files were converted to PDBQT using MGLTools (version 1.5.6)[215]. Lastly, the compounds were docked into the structure of PARP-1 described in "Chapter 2.3.1: Receptor File Preparation" and scored by QVina2[205] using the same docking parameters used in the AutoGrow4 PARPi lead-optimization runs ("Chapter 3.2.2: PARPi Lead Optimization" and Appendix JSON 6). Finally, the compounds were imported back into SMILES using Open Babel[164,215] and re-imported into RDKit[165] for analysis. Of these compounds, 4,614 unique SMILES strings were successfully converted to PDBQTs, docked, and re-imported into RDKit[165].

4.2.1.2 AutoGrow4 PARPi Lead Optimization

For these experiments, I use the six AutoGrow4 PARPi lead-optimization runs described in "Chapter 3.3.2: PARP-1 Lead Optimization." Please see that chapter for a description of the file preparation and run conditions.

4.2.2 Comparison of AutoGrow4 PARPi Lead Optimization and *De Novo* CADD Programs

In this section, I compare the AutoGrow4 lead-optimization run with lead-optimization runs performed using three alternative *de novo* CADD programs: (1) *de novo* DOCK[162], (2) GANDI[167], and (3) LigDream[213]. I chose these programs because they are all open-source, free-for-academic-use, and functional without having to modify the source code[162,167,213].

4.2.2.1 De Novo DOCK

De novo DOCK is a free-for-academic-use *de novo* CADD program that is built into the DOCK6 docking program[162]. *De novo* DOCK grows fragments into candidate compounds[162]. I optimized a set of PARPi and PARPi fragments with both *de novo* DOCK and AutoGrow4 to compare program performance.

4.2.2.1.1 File Preparation and Run Conditions

To compare AutoGrow4 to *de novo* DOCK (version DOCK6.9), I used *de novo* DOCK to optimize the same 94 PARPi and PARPi-fragment leads used in the AutoGrow4 PARPi lead-optimization runs. *De novo* DOCK requires both the protein and ligand files to be in MOL2 format[162]. I converted the 4R6E:A structure from PDB to MOL2 and prepared the docking files using Chimera[219] and the DOCK6 sphgen and grid tools[187,219,264,265]. I also converted the 94 compounds from SMILES to PDB using Gypsum-DL (version 1.1.2)[214], and then to the MOL2 format using Open Babel (version 2.3.2)[164,215]. The MOL2 ligands were further processed using

GANDI's `mol2_cleaner` script[167] and the CgenFF server (interface version 1.0.0 and force field version 3.0.1, using the default settings)[221–223]. Ninety of the PARPi and PARPi fragments produced one or more 3D variants in MOL2 format. These MOL2 files were then processed using the DOCK6 fragment-library generator, which separates structures and fragments into MOL2 files containing either "linker," "scaffold," or "sidechain" fragments[187]. These same fragment sets were used to seed *de novo* DOCK.

De novo DOCK provides a limited set of filter options (e.g., MW, number of heavy atoms, number of rotatable bonds, and formal charge)[162]. To best match the AutoGrow4 Ghose filter used in the PARPi lead-optimization runs, I set *de novo* DOCK to restrict the MW of its output molecules to 160-480 Da and the number of heavy atoms to 20-70.

Although the docking program used by *de novo* DOCK, DOCK6 (version 6.9), is MPI enabled, MPI multiprocessing is currently disabled for the *de novo* DOCK function. I submitted 30 independent *de novo* DOCK runs on SMP-enabled CRC nodes with 28-core AMD EPYC 7302 processors. The exact settings for these runs are provided in Appendix JSON 8. Unique random seeds were provided for each run.

4.2.2.1.2 Post-Run Processing

For each generated molecule, *de novo* DOCK output 3D poses positioned within the target receptor site, which were docked and scored using DOCK6. I scored each ligand pose using QVina2's (version 2.1)[205] scoring function without resampling (i.e., without redocking). The

MOL2 file(s) for each ligand were converted to PDBQT using OpenBabel[164,215] so they could be rescored with QVina2.

Many of the output compounds were redundant. To identify unique compounds, I converted every *de novo* DOCK-generated MOL2 file into a canonical SMILES representation using OpenBabel[164,215] and RDKit[165], and used the best docking score for each unique canonical SMILES. This reduced the total number of output molecules from 709,417 to 37,883 unique compounds.

4.2.2.1.3 Post-Processing Compound Analysis

Following all post-run processing, I evaluated the population of generated compounds using several metrics. First, I scored each compound in terms of normalized chemical diversity relative to other generated compounds ("Chapter 2.3.5: Calculating Normalized Diversity Scores"). I also scored each compound in terms of predicted synthetic accessibility score (SA_Score) using the Scopy (version 1.2.1)[166] implementation of Ertl and Schuffenhauer's SA_Score[266]. Next, to assess the physiochemical properties of the compounds, I calculated MW and the molecular quantitative estimate of drug-likeness (QED) of all generated compounds. QED is calculated by evaluating multiple physiochemical properties such as the number of rotatable bonds, polar surface area, and logP. All compounds were evaluated using Scopy's QED_{Mean} measurement[166,267]. I then filtered the population of compounds using the same chemical filters that were used during the AutoGrow4 runs: the Lipinski*, Ghose, and PAINS filters. Lastly, I filtered the population of compounds using two Scopy-implemented toxicity filters: the

idiosyncratic[166,268] and SureChEMBL[166,269,270] filters, which will be described and discussed in "Chapter 4.3.2.4: Comparison of *De Novo* CADD Programs: ADME-PK" (p.217).

4.2.2.2 GANDI

GANDI is a free and open-source program that uses a GA to generate novel *de novo* compounds[167]. GANDI uses an island model GA, which evolves populations of compounds independently, with occasional events exchanging compounds between populations[167]. I optimized the same set of PARPi and PARPi fragments using GANDI and AutoGrow4 to compare program performance.

4.2.2.2.1 File Preparation and Run Conditions

To compare AutoGrow4's performance to that of GANDI (version 2.0)[167], I applied GANDI to the 94 PARPi and PARPi fragments used in my AutoGrow4 PARPi lead-optimization runs. Like *de novo* DOCK, GANDI also requires that both the protein and ligand files be in MOL2 format. To prepare the protein for GANDI, I assigned partial charges to the protein using VMD's `Automatic PSF Builder` function[220] and converted the 4R6E:A structure from PDB to MOL2 using Chimera[219]. The receptor file was further processed using the `mol2tripos_to_seed_protein` script provided by the program SEED[217].

Because GANDI requires source compounds that are already positioned in the target pocket, I converted the 94 PARPi and PARPi fragments from SMILES to PDB files using Gypsum-DL (version 1.1.2), and then to the MOL2 format using Open Babel (version 2.3.2)[164,215]. The

MOL2 ligands were further processed using GANDI's `mol2_cleaner` script[167] and the CgenFF server (interface version 1.0.0 and force field version 3.0.1, using default settings)[221–223]. Ninety of the PARPi and PARPi fragments produced one or more 3D variants in MOL2 format. Based on instructions in the GANDI manual, I then docked the 3D small-molecule variants using the program SEED[217]. The output docked poses were used as the *docked fragments* required for GANDI runs.

GANDI provides limited predefined filter options to guide small-molecule production: MW, number of hydrogen bond donors, and number of hydrogen bond acceptors. To best match the Ghose and Lipinski* filters used in the AutoGrow4 lead-optimization runs, I set the MW range to 160-480 Da, the number of hydrogen bond donors to 0-5, and the number of hydrogen bond acceptors to 0-10.

I ran GANDI six times using 112 islands, with 1,000 individuals and 1,000 iterations per run. Each run was provided a unique random seed. I used the linker fragments provided in the GANDI download. This experiment ran on ten MPI-enabled CRC nodes with four 28-core (112 CPUs) Broadwell processors and 64GB RAM/node, which were networked with Intel's Omni-Path communication architecture. The exact settings for these runs are provided in Appendix JSON 9.

4.2.2.2.2 Post-Run Processing and Analysis

For each generated molecule, GANDI output 3D poses positioned within the PARP-1 catalytic site. To compare the output ligands to AutoGrow4-generated compounds, I scored the

binding affinity of each ligand pose using QVina2's (version 2.1)[205] scoring function without resampling the pose (i.e., without redocking). The MOL2 file(s) for each ligand were converted to PDBQT using OpenBabel[164,215] so they could be rescored with QVina2.

Because many of the output compounds were redundant, I converted every GANDI-generated MOL2 into canonical SMILES representation using OpenBabel[164,215] and RDKit[165], and used the best docking score for each unique SMILES. This resulted in a total of 589 unique compounds.

Lastly, using the same methods described in "Chapter 4.2.2.1.3: Post-Processing Compound Analysis," I evaluated the compounds in terms of normalized chemical diversity, SA_Score, MW, and QED. I also filtered the population using the Lipinski*, Ghose, PAINS, idiosyncratic, and SureChEMBL filters.

4.2.2.3 LigDream

LigDream is a free and open-source program for lead optimization that uses a neural network trained on the 3D shape of drug-like compounds[213]. I optimized the same set of PARPi and PARPi fragments using LigDream and AutoGrow4 to compare program performance.

4.2.2.3.1 File Preparation and Run Conditions

To compare LigDream (version 1.0)[213] and AutoGrow4 lead optimization, I ran LigDream multiple times with different settings to optimize the same set of PARPi and PARPi fragments. For all LigDream runs, the maximum number of output molecules was set to 100. Based on the settings

used in the LigDream publication[213], I ran LigDream with the λ-factor set to 0, 1, 5, 10, and 15. The λ-factor controls noise within the algorithm, where a higher λ results in the generated molecules deviating more from the source compounds. For each λ value, I ran LigDream sampling both with and without the probabilistic recurrent neural network. Each condition was run in triplicate. Duplicate SMILES were eliminated, resulting in 21,855 unique LigDream-generated compounds. The exact settings for these runs are provided in Appendix JSON 10.

4.2.2.3.2 Post-Run Processing and Analysis

Because LigDream outputs molecules in SMILES format without assessing compound binding to the target pocket, I docked each LigDream-generated compound into the PARP-1 catalytic domain (4R6E:A) using QVina2 (version 2.1)[205], using the same settings and approach as in the AutoGrow4 PARPi lead-optimization runs. I first converted the SMILES to PDBQT format using Gypsum-DL (version 1.1.2)[214] (maximum variance set to five) and MGLTools (version 1.5.6)[215], and then docked each variant exhaustively (*exhaustivity* set to 25) using QVina2.

Lastly, using the same methods described in "Chapter 4.2.2.1.3: Post-Processing Compound Analysis," I evaluated the compounds in terms of normalized chemical diversity, SA_Score, MW, and QED. I also filtered the population using the Lipinski*, Ghose, PAINS, idiosyncratic, and SureChEMBL filters.

4.2.2.4 AutoGrow4 PARPi Lead Optimization

AutoGrow4 is a free and open-source *de novo* CADD program for lead optimization and lead generation using a GA[6]. I optimized a set of PARPi and PARPi fragments using AutoGrow4 to compare its performance with three other *de novo* CADD programs.

4.2.2.4.1 File Preparation and Run Conditions

For these experiments, I used the six AutoGrow4 PARPi lead-optimization runs described in "Chapter 3.3.2: PARP-1 Lead Optimization." Please see that chapter for a description of the file preparation and run conditions.

4.2.2.4.2 Post-Run Processing and Analysis

I pooled the results of the six AutoGrow4 PARPi lead-optimization runs by generation. Compounds within each generation were ranked by docking score. Duplicate canonical SMILES were removed, preserving the best docking score for each unique SMILES.

Using the same methods described in "Chapter 4.2.2.1.3: Post-Processing Compound Analysis," I then evaluated the compounds in terms of normalized chemical diversity, SA_Score, MW, and QED, and I filtered the population using the idiosyncratic and SureChEMBL filters.

4.3 Results and Discussion

4.3.1 Comparison of AutoGrow4 PARPi Lead Optimization and VS Lead Optimization

In this section I compare AutoGrow4 PARPi lead-optimization and VS lead-optimization runs in terms of predicted binding affinity as measured by QVina2 docking and normalized compound diversity. The normalized diversity scores were calculated by dividing the diversity scores (Equation 2) by the number (N) of compounds being compared ("Chapter 2.3.5: Calculating Normalized Diversity Scores"). A lower normalized diversity score indicates that a compound is more unique within the set of compounds used for comparison, and a higher score means it is more similar ("Chapter 2.3.5: Calculating Normalized Diversity Scores"). The zeroth generation of AutoGrow4 runs consists of known PARPi and PARPi fragments, and so the zeroth generation scores provide a context for my results.

Overall, AutoGrow4 yielded compounds with higher predicted docking scores than the VS (Table 16). Both AutoGrow4 and the VS successfully predicted PARPi-like compounds with stronger predicted binding affinities than the positive control PARPi and PARPi fragments (i.e., the AutoGrow zeroth generation) (Table 16). However, by the first generation AutoGrow4 had already predicted a compound with a docking score 1.2 kcal/mol better than the best compound discovered by the VS (Table 16 and Figure 28). Additionally, the AutoGrow4 first generation also had stronger average docking scores among its top 50 and 1,000 compounds than the compounds found by the VS (Table 16). The average AutoGrow4 docking scores improved each successive

generation, so by the fifth generation the mean score of the top 1,000 compounds (-13.62 kcal/mol) was stronger than that of the best compound in the VS (Table 16 and Figure 28).

The global population of each AutoGrow4 generation was on average more chemically diverse than the VS compounds (Table 17). Of course, the chemical diversity of the VS compounds is determined by how the library was designed.

Overall, AutoGrow4 produced compounds with stronger predicted binding affinities that were also more chemically diverse than those predicted by the VS. These results suggest that AutoGrow4 is in fact searching a wider range of chemistry space and finding better optima than the VS.

Table 16. Comparison of VS and AutoGrow4 docking scores.

"Gen. #" indicates the generation number of the AutoGrow4 PARPi lead-optimization runs. The "VS" row displays the virtual-screen data. "Global" indicates data for the total of a given generation or VS, and "N" indicates the number of top-scoring compounds included in the analyses. The data from the six AutoGrow4 runs has been pooled; "Mean" represents the mean of the pooled compounds. Means are in units of kcal/mol as measured by QVina2. "Dev." represents the standard deviation of the included scores. "% Pop." represents the percent of scores that were included (100 * N / Size).

Gen. #	Global			N=1	N=50			N=1000		
	Mean	**Dev.**	**Size**	**Mean**	**Mean**	**Dev.**	**% Pop.**	**Mean**	**Dev.**	**% Pop.**
VS	-9.48	1.38	4614	-13.5	-12.70	0.26	1.08	-11.30	0.55	21.67
0	-7.89	2.32	244	-12.4	-11.05	0.65	20.49	---	---	---
1	-9.07	1.61	5402	-14.7	-13.03	0.31	0.93	-11.40	0.63	18.51
2	-9.84	1.42	28964	-14.7	-13.40	0.31	0.17	-12.52	0.34	3.45
3	-9.64	2.09	28999	-14.7	-13.78	0.26	0.17	-13.00	0.28	3.45
4	-10.20	2.06	28272	-14.7	-14.18	0.22	0.18	-13.35	0.28	3.54
5	-9.30	2.47	27993	-14.8	-14.35	0.18	0.18	-13.62	0.26	3.57

Table 17. Comparison of VS and AutoGrow4 diversity.

"Gen. #" indicates the generation number of the AutoGrow4 PARPi lead-optimization runs. The row labeled "VS" displays the virtual screen data. "Global" indicates data for the total of a given generation or VS, and "N" indicates the number of the top-scoring compounds included in the analysis. The data from the six AutoGrow4 runs has been pooled. For each subset of N compounds, the diversity score of each compound was assigned using Equation 2, and then normalized by dividing N ("Chapter 2.3.5: Calculating Normalized Diversity Scores"). The reported means and standard deviations are based on these normalized diversity scores (unitless). "Dev." represents the standard deviation of the included scores. "% Pop." represents the percent of scores that were included (100 * N / Size). The reported means and standard deviations are based on these normalized diversity scores (unitless).

Gen. #	Global			N=50			N=1000		
	Mean	**Dev.**	**Size**	**Mean**	**Dev.**	**% Pop.**	**Mean**	**Dev.**	**% Pop.**
VS	0.31	0.03	4614	0.34	0.03	1.08	0.33	0.03	21.67
0	0.23	0.05	244	0.29	0.03	20.49	---	---	---
1	0.24	0.03	5402	0.36	0.06	0.93	0.27	0.03	18.51
2	0.23	0.03	28964	0.29	0.03	0.17	0.29	0.04	3.45
3	0.20	0.02	28999	0.26	0.02	0.17	0.28	0.03	3.45
4	0.20	0.03	28272	0.28	0.03	0.18	0.27	0.03	3.54
5	0.20	0.03	27993	0.27	0.03	0.18	0.26	0.02	3.57

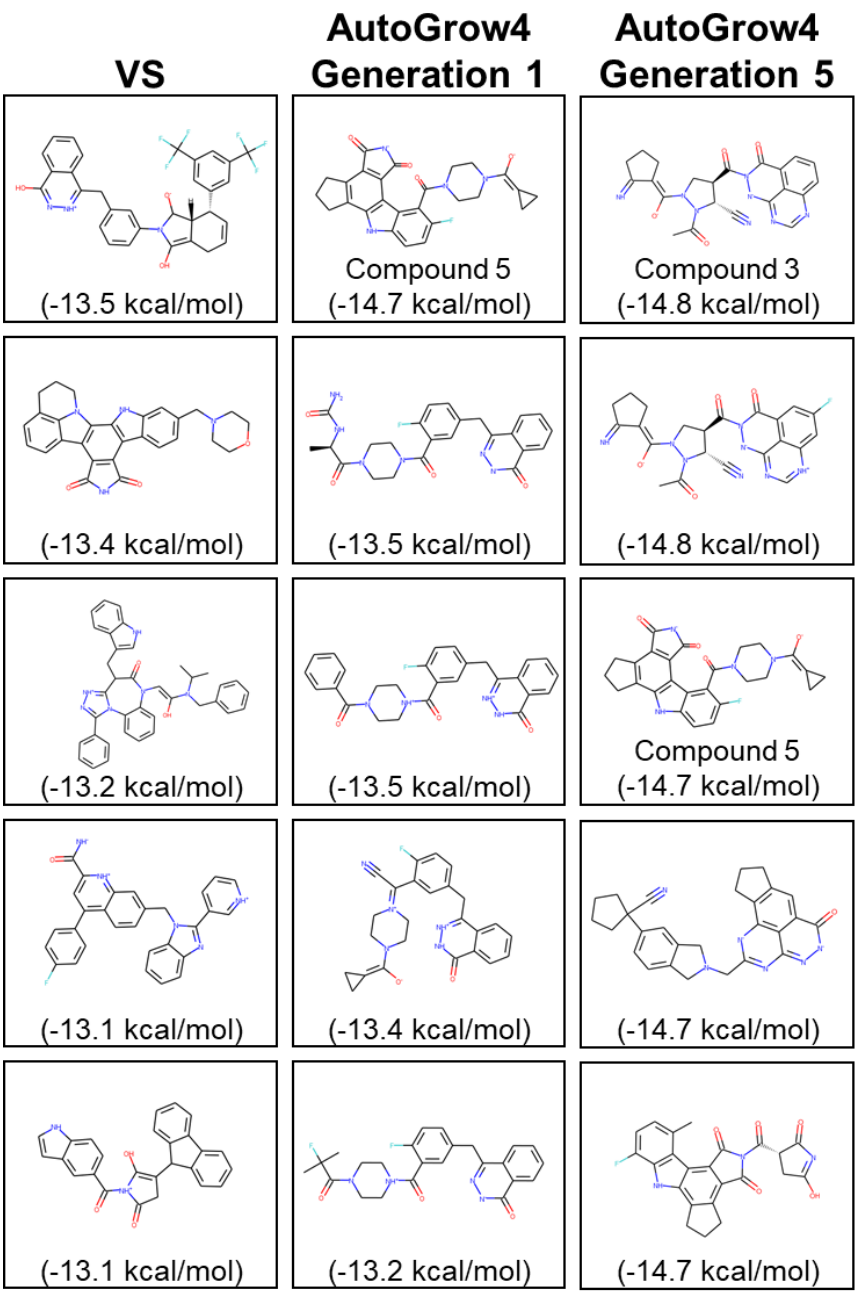


Figure 28. **The five best-scoring compounds from the VS, as well as the first and fifth AutoGrow4 generations.**

The top five compounds generated by VS lead optimization (left) and AutoGrow4 PARPi lead optimization in the first generation (middle) and fifth generation (right), after removing compounds that are preexisting PARPi. The reported docking scores were determined by QVina2. The AutoGrow4-generated compounds, Compound 3 and Compound 5, are discussed in "Chapter 2.2.2: Operators: Population Generation via Crossover, Mutation, and Elitism" and "Chapter 3.3.2: PARP-1 Lead Optimization", respectively.

4.3.2 Comparison of AutoGrow4 Lead Optimization and other *De Novo* CADD Programs

In this section I compare AutoGrow4's lead-optimization performance to that of three other *de novo* CADD programs: *de novo* DOCK, GANDI, and LigDream. An in-depth comparison of each program can be found in "Chapter 1.2.5: Alternative Approaches for *de novo* CADD." An additional comparison of AutoGrow4 to (1) VS-based lead optimization and (2) AutoGrow3, the previous version, can be found in "Chapter 4.3.1: Comparison of AutoGrow4 PARPi Lead Optimization and VS Lead Optimization" and "Chapter 2.4: Results and Discussion," respectively.

I compare the compounds produced by each program in terms of predicted binding affinity as assessed by QVina2 as well as chemical diversity and synthetic accessibility as assessed using Ertl and Schuffenhauer's SA_Score[266]. Predicted binding affinity is a metric for assessing the candidate inhibitors. Chemical diversity is a metric for assessing the breadth of each program's search of chemistry space. Synthetic accessibility is a metric for predicting how feasible the compounds are to synthesize. Additionally, I compare the compounds in terms of drug-likeness and predicted ADME-PK properties.

4.3.2.1 Comparison of *De Novo* CADD Programs: Predicted Binding Affinity

AutoGrow4 selects compounds based primarily on predicted binding affinity as calculated by a docking program. Inherent in this process are two assumptions: (1) predicted binding affinity correlates with experimental binding affinity, and (2) compounds with stronger binding affinities are better inhibitors. Both assumptions are flawed. Firstly, the accuracy of predicted binding

affinities varies depending on the protein being targeted, the docking software being used, and the exhaustivity of the pose sampling. Additionally, stronger predicted binding affinity does not always result in stronger inhibition; for example, talazoparib is a more potent PARP-1 inhibitor than olaparib[75], but QVina2 predicts that olaparib has a stronger binding affinity than talazoparib when docking into 4R6E:A (-13.6 vs. -11.1 kcal/mol). Also, binding affinity does not account for many critical factors that determine the success of a drug (e.g., synthesizability, drug-likeness, and cytotoxicity), several of which I will explore in later sections ("Chapters 4.3.2.3 and 4.3.2.4."). However, while recognizing these limitations, relying on predicted binding affinity enables AutoGrow4 to be used as a high-throughput technique for early-stages of drug design.

Among the three tested programs, AutoGrow4 produced the compounds with the best predicted binding affinities (Table 18). The AutoGrow4 first generations performed better than all tested programs in terms of the best-scored compound and the mean docking score of the top 50 compounds (Table 18). While *de novo* DOCK had better average docking scores among the top 1,000 compounds (Table 18), by the second generation AutoGrow4 outperformed all other programs in terms of the best average docking score for the entire population and the best-scored 1, 50, and 1,000 compounds (Table 16 and Table 18 on p.202 and p.209, respectively). By the fifth generation, AutoGrow4's best-scored 1,000 compounds had an average docking score better than that of the next best program, *de novo* DOCK, with average docking scores of -13.62 kcal/mol and -11.67 kcal/mol respectively.

De novo DOCK likely had better average docking scores than GANDI and LigDream among the respective top 50 and 1,000 compound sets (Table 18) because it docks the generated compounds into the target pocket. In contrast, GANDI and LigDream do not dock[162,167,213].

GANDI relies on the poses of the source compounds, which users must dock separately prior to running the program[167]. Because GANDI does not reassess the poses of its generated compounds, these output poses are arguably more likely to be suboptimal[167]. Even so, GANDI did produce the single best-scoring compound of any non-AutoGrow4 program (-14.52 kcal/mol) and had the second-best global mean docking scores (-9.45 kcal/mol).

On the other hand, LigDream does not use docking to generate and assess novel compounds[213]. It is unique among the tested *de novo* CADD programs because it does not consider the structure of the target protein pocket[213]. Instead, LigDream relies on a trained recurrent neural network to predict compounds that are similar to both known drugs and to the input seed compounds[213]. LigDream assumes that the input SMILES are valid ligands that bind the target pocket[213]. To compare the predicted binding affinities of the compounds across programs, I converted the LigDream-output SMILES strings into 3D PDB files and docked them into the PARP-1 catalytic pocket using QVina2 ("Chapter 4.2.2.3: LigDream.") The top 50, and 1,000 best-scoring compounds generated by LigDream had worse docking scores than those generated by (1) AutoGrow4's first through fifth generations and (2) *de novo* DOCK (Table 18).

In general, these comparisons understandably favor the AutoGrow4-generated populations because AutoGrow4 is the only tested program that natively uses QVina2 for both ligand-pose sampling and scoring. But I chose QVina2 to assess these compounds because of its computational efficiency and accuracy. Furthermore, a recent independent study comparing modern docking programs found that QVina2 and Vina are more accurate than DOCK6 at ligand-pose placement when comparing docked poses against crystal poses[271].

AutoGrow4 performed markedly better than the other programs in terms of QVina2-predicted binding affinities and was able to produce the single compound with the strongest predicted affinity. By the fifth generation, the average of AutoGrow4's top 1,000 compounds were predicted to bind better than the fifty best compounds produced by any other program (Table 18).

Table 18. Comparison of docking scores of *de novo* CADD-generated compounds.
"Gen. #" indicates the AutoGrow4 generation number (PARPi lead-optimization runs). "Global" indicates all compounds of a given generation, or produced by a given program. "N" indicates the number of top-scoring compounds selected. All runs of each respective program were pooled together. "Mean" represents the mean of the subset of pooled compounds. Means are in units of kcal/mol, as calculated by QVina2. "Dev." represents the standard deviation of the included scores. "% Pop" represents the percent of top-scoring compounds included (100 * N / Size).

Program	Global			N=1	N=50			N=1000		
	Mean	Dev.	Size	Mean	Mean	Dev.	% Pop.	Mean	Dev.	% Pop.
***De novo* DOCK**	-8.28	1.97	37883	-14.3	-12.81	0.44	0.13	-11.67	0.39	2.64
GANDI	-9.45	1.54	523	-14.5	-11.95	0.80	9.56	---	---	---
LigDream	-9.12	1.03	21586	-13.0	-12.13	0.26	0.23	-11.19	0.35	4.63
AutoGrow4										
Gen. 0	-7.89	2.32	244	-12.4	-11.05	0.65	20.49	---	---	---
Gen. 1	-9.07	1.61	5402	-14.7	-13.03	0.31	0.93	-11.40	0.63	18.51
Gen. 5	-9.30	2.47	27993	-14.8	-14.35	0.18	0.18	-13.62	0.26	3.57

4.3.2.2 Comparison of *De Novo* CADD Programs: Chemical Diversity

Within the context of *de novo* drug design, chemical diversity represents the sampling of different regions of chemical space. Beyond improving the desired metric (e.g., docking affinity score, and drug-likeness), an ideal *de novo* CADD approach must also sample a diverse range of chemical structures. Diversity is particularly important in an iterative *de novo* CADD approach, such as a genetic algorithm, because later iterations require a sufficiently diverse population to create novel solutions; otherwise, premature convergence is inevitable. Because chemical diversity is crucial to drug design, I compared the chemical diversity of each AutoGrow4 generation, as well as the compounds with the top 50 and 1,000 best docking scores produced by each *de novo* design program.

Globally, AutoGrow4 produced more diverse compounds than any other program tested. Each AutoGrow4 generation was more diverse than the populations produced by the other three programs, with a fifth-generation average normalized diversity score of 0.20 (Table 17 and Table 19 on p.203 and p.212, respectively, where populations with normalized diversity scores near 0.0 contain more unique compounds). GANDI performed the worst, with an average normalized diversity score of 0.30. *De novo* DOCK and LigDream performed equally well (0.26 each, Table 19). LigDream and AutoGrow4's fifth generation tied for the most diverse populations among their top 50 best-scoring compounds, but the average diversity of the AutoGrow4's top 50 and 1,000 best-scoring compounds fluctuates from generation to generation (Table 17 and Table 19 on p.203 and p.212, respectively).

LigDream produced equally or more diverse compounds than both *de novo* DOCK and GANDI in all subpopulation sizes (Table 19). LigDream's excellence in producing diverse compounds may be in part due to its user-controllable λ-factor, which determines how similar the generated compounds must be to the source compounds[213]. Based on the LigDream publication[213], I ran LigDream multiple times with different λ-factor values. Because of this, the LigDream-generated compound pool consists of a mix of compounds with a wide range of similarities to the source compounds. Had these runs been performed with more restrictive λ-factor values or performed with the same λ-factor for all runs, the LigDream-generated compounds would likely be less diverse.

However, the top 1, 50, and 1,000 LigDream compounds also had subpar average docking scores compared to the other programs (Table 19 on p.212), perhaps because of the increased deviance from the source compounds. Rather than assessing the generated compounds with regards to the protein structure, LigDream relies purely on the input seed(s) to determine what might be a "good" ligand for the target protein; therefore, increasing its deviation from the seed PARPi also diverts the algorithm away from the proven drug-like seed compounds. In contrast, AutoGrow4, which also provides user controllable diversity parameters, corrects for population drift by evaluating each generated compound relative to the target protein pocket. AutoGrow4 thereby balances its incorporation of diversity with a focus on compounds that are optimized for a specific protein target.

Table 19. Comparison of diversity scores of *de novo* CADD-generated compounds. "Gen. #" indicates the AutoGrow4 generation number (PARPi lead-optimization runs). "Global" indicates all compounds of a given generation, or produced by a given program. "N" indicates the number of top-scoring compounds selected. All runs of each respective program were pooled together. "Mean" represents the mean of the subset. For each subset of N compounds, the diversity score of each compound was assigned using Equation 2, and then normalized by dividing N ("Chapter 2.3.5: Calculating Normalized Diversity Scores"). The reported means and standard deviations are based on these normalized diversity scores (unitless). "Dev." represents the standard deviation of the included scores. "% Pop." represents the percent of scores that were included (100 * N / Size).

Program	Global			N=50			N=1000		
	Mean	**Dev.**	**Size**	**Mean**	**Dev.**	**% Pop.**	**Mean**	**Dev.**	**% Pop.**
***De novo* DOCK**	0.26	0.06	37883	0.30	0.05	0.13	0.29	0.06	2.64
GANDI	0.30	0.03	523	0.30	0.03	9.56	---	---	---
LigDream	0.26	0.03	21586	0.27	0.02	0.23	0.27	0.02	4.63
AutoGrow4									
Gen. 0	0.23	0.05	244	0.29	0.03	20.49	---	---	---
Gen. 1	0.24	0.03	5402	0.36	0.06	0.93	0.27	0.03	18.51
Gen. 5	0.20	0.03	27993	0.27	0.03	0.18	0.26	0.02	3.57

4.3.2.3 Comparison of *De Novo* CADD Programs: Synthetic Accessibility

Synthesizability is critical in *de novo* drug design because it determines the cost, time, and feasibility of advancing a molecule from *in silico* to *ex silico* testing. To compare the synthesizability of compounds generated by each *de novo* CADD programs, I calculated an SA_Score[266] for each generated compound. The SA_Score predicts how synthesizable a given compound is and ranges from 1.0 (easiest to synthesize) to 10.0 (most difficult to synthesize)[266].

As predicted, AutoGrow4's mean SA_Score tended to increase as the generations progressed, suggesting that later AutoGrow4 generations are less synthesizable than earlier generations. The mean SA_Score of the 1,000 best docking scored compounds increased from 3.99 to 4.49 between generations one and five (Table 20). The accumulation of mutation and crossover events used to create each subsequent population may have caused the populations to drift from the PARPi source compounds, thus becoming less synthesizable.

Of all the tested programs, LigDream created the most synthesizable compounds in terms of entire populations and the compounds with the 50 and 1,000 best docking scores. Additionally, LigDream's best docking scored compound was more synthesizable than the best compound produced by any other tested program excluding AutoGrow4's generation zero which consists of know (and therefore synthesizable) PARPi and PARPi fragments.

A synthesizable compound is only a good drug candidate if it also binds to its target protein; however, a small molecule that binds strongly to its target but cannot reasonably be synthesized will not likely progress. Therefore, I calculated the ratio of the predicted binding affinity (kcal/mol) to the SA_Score (unitless) for each individual compound produced by all of the considered *de*

novo CADD programs. I then calculated the average ratio for each population and for the compounds with the best 50 and 1,000 docking scores produced by each program. This efficiency metric rewards improved binding affinity and penalizes poorer SA_Scores; more negative results indicate better candidate compounds. For instance, the AutoGrow4-generated compound with the best docking score between the first and fourth generation, Compound 5 (-14.7 kcal/mol), had only a 0.1 kcal/mol difference in docking score from the compound with the best docking score in the fifth generation, Compound 3 (-14.8 kcal/mol) (Table 16 and Figure 28; p.202 and p.204 respectively), but it also had a more favorable SA_Score (3.90 and 5.15 respectively) (Table 20 and Table 21). This docking-to-SA_Score ratio favored Compound 5 over the better-scored Compound 3 (-3.77 and -2.87 respectively) (Table 21) because the gain in predicted binding affinity was not enough to overcome the poorer synthesizability.

The reevaluation of docking-to-SA_Score ratios also shows that among the compounds with the best 50 and 1,000 docking score compounds, AutoGrow4 and LigDream produced compounds with better docking-to-SA_Score ratio averages than both GANDI and *de novo* DOCK (Table 21).

Table 20. Comparison of SA_Scores of *de novo* CADD-generated compounds.
"Gen. #" indicates the AutoGrow4 generation number (PARPi lead-optimization runs). "Global" indicates all compounds of a given generation, or produced by a given program. "N" indicates the number of top-scoring compounds selected. All runs of each respective program were pooled together. "Mean" represents the mean of the subset of pooled compounds. "Dev." represents the standard deviation of the included scores. "% Pop" represents the percent of top-scoring compounds included (100 * N / Size). The reported means and standard deviations are based on SA_Scores (unitless).

Program	Global			N=1	N=50			N=1000		
	Mean	**Dev.**	**Size**	**Mean**	**Mean**	**Dev.**	**% Pop.**	**Mean**	**Dev.**	**% Pop.**
***De novo* DOCK**	4.22	1.28	37883	5.83	5.10	1.41	0.13	4.46	1.30	2.64
GANDI	5.24	0.71	523	4.40	5.17	0.67	9.56	---	---	---
LigDream	3.31	0.73	21586	3.79	3.68	0.74	0.23	3.54	0.71	4.63
AutoGrow4										
Gen. 0	3.58	1.14	244	2.81	3.78	0.72	20.49	---	---	---
Gen. 1	3.80	0.87	5402	3.90	4.11	0.58	0.93	3.99	0.78	18.51
Gen. 5	4.81	0.83	27993	5.15	4.53	0.71	0.18	4.49	0.69	3.57

Table 21. Comparison of docking-to-SA_Score ratios of *de novo* CADD-generated compounds.

"Gen. #" indicates the generation number in the AutoGrow4 PARPi lead-optimization runs. "Global" indicates data for the total of a given generation or the total produced by a given program, while "N" indicates the number of the top scoring compounds selected. All runs of each respective program were pooled together; "Mean" represents the mean of the subset of pooled compounds. The docking-to-SA_Score ratios for each molecule was determined by dividing the docking score (kcal/mol) by the SA_Scores (unitless). "Dev." represents the standard deviation of the included scores. "% Pop." represents the percent of scores that were included (100 * N / Size). The reported means and standard deviations are determined based on those docking-to-SA_Score ratios.

Program	Global			N=1	N=50			N=1000		
	Mean	Dev.	Size	Mean	Mean	Dev.	% Pop.	Mean	Dev.	% Pop.
***De novo* Dock**	-2.13	0.79	37883	-2.46	-2.73	0.81	0.13	-2.83	0.78	2.64
GANDI	-1.85	0.46	523	-3.30	-2.36	0.39	9.56	---	---	---
LigDream	-2.87	0.64	21586	-3.43	-3.43	0.68	0.23	-3.28	0.65	4.63
AutoGrow4										
Gen. 0	-2.44	0.99	244	-4.41	-3.05	0.68	20.49	---	---	---
Gen. 1	-2.51	0.69	5402	-3.77	-3.23	0.49	0.93	-2.97	0.60	18.51
Gen. 5	-2.00	0.70	27993	-2.87	-3.24	0.49	0.18	-3.11	0.49	3.57

4.3.2.4 Comparison of *De Novo* CADD Programs: ADME-PK Properties

In this section I compare the ADME-PK properties of the compounds produced by the four *de novo* CADD programs. As discussed in "Chapter 1.2.3: Chemical Properties for Selecting Drug-Like Compounds," ideal chemotherapeutic candidates not only bind and inhibit their intended target, but also have acceptable physiochemical properties that facilitate drug efficacy. Additionally, an ideal cancer chemotherapeutic should have few off-target interactions and a lower toxicity to non-cancer cells than to cancer cells. I evaluate the *de novo* CADD-generated compounds using two common cheminformatic metrics: MW (Table 22) and molecular quantitative estimate of drug-likeness (QED) (Table 23). I also filter the *de novo* DOCK, GANDI, and LigDream-generated compounds with the three filters applied during the AutoGrow4 runs (Lipinski*, Ghose, and PAINS, Table 24) to show that, because of their lack of chemical filter options, the non-AutoGrow4 programs waste resources evaluating less drug-like compounds. Lastly, to show that AutoGrow4 produced fewer toxic compounds than the other programs did, I filtered the compounds produced by all programs with two toxicity filters: the idiosyncratic[166,268] and SureChEMBL[166,269,270] filters (Table 25).

4.3.2.4.1 Physiochemical Properties and Drug-Likeness: MW and QED

AutoGrow4's use of ADME-PK filters is one its major advantages over other *de novo* CADD programs. These filters select for more favorable compound properties and thereby guide the run towards more favorable regions of chemistry space. For instance, after five generations the

AutoGrow4-generated population still had a lower global average MW than those of *de novo* DOCK, GANDI, and LigDream (Table 22), in part due to AutoGrow4's use of the Lipinski* and Ghose filters. Those filters refine candidates by properties that exclude large compounds (e.g., MW and atom count). Additionally, through its use of these filters AutoGrow4 maintains a global average MW that is well within the drug-like range (i.e., 160-480 Da[168]), which allows for continued sampling (i.e., more generations of testing) without running into issues associated with high MW, such as poor compound solubility. However, for all subpopulation sizes and programs, the top-scoring compounds tended to have higher MWs than their respective global average MW (Table 22). This is likely because high-MW small molecules can make more interactions than smaller fragment-like compounds and so tend to have better docking scores.

Across all tested programs, the average MWs of the compounds with the best docking scores were frequently more similar to each other than to the global-average MWs (Table 22). For example, the average MWs of the 50 compounds with the best docking scores from AutoGrow4 generations one to five ranged from 450.7 Da to 465.7 Da. Similarly, the GANDI and *de novo* DOCK compounds, which respectively had the lowest and highest non-AutoGrow4 average MW for their compounds with the 50 best docking scores, had a range of 456.8 Da to 469.1 Da (Table 22). Even though LigDream was the only tested program that does not provide options to set MW constraints, the LigDream-generated compounds frequently had lower MW than the GANDI and *de novo* DOCK-generated compounds (Table 22). This is likely because LigDream's models were trained on drug-like compounds with MWs ranging from 250 Da and 500 Da[213].

AutoGrow4 also performed well in terms of QED (Table 23). QED is an integrative metric that evaluates multiple physicochemical properties (e.g., the number of rotatable bonds, polar

surface area, and logP) to produce a single score ranging from 0.0 to 1.0, where 0.0 indicates that a compound has all unfavorable properties and 1.0 indicates that a compound contains all favorable properties[166,267]. QED is useful because compounds with very favorable properties that have only a few mildly unfavorable properties can still be evaluated as drug-like[166,267]. Although the average QED scores tend to decrease for each successive AutoGrow4 generation (e.g., global average QED went from 0.62 in the zero generation to 0.46 in the fifth generation), the global average QED of AutoGrow4's fifth generation (0.46 +/-0.18) was better than that of *de novo* DOCK (0.40 +/-0.15) and comparable to that of LigDream (0.48 +/-0.14) and GANDI (0.45 +/-0.14) (Table 23). Despite multiple generations of evolution and many more novel compounds predicted, AutoGrow4 maintains drug-likeness comparably or better than the other programs.

Table 22. Molecular weight (MW) of *de novo* CADD-generated compounds.
"Gen. #" indicates the AutoGrow4 generation number (PARPi lead-optimization runs). "Global" indicates all compounds of a given generation or produced by a given program. "N" indicates the number of top-scoring compounds selected. All runs of each respective program were pooled together. "Mean" represents the mean of the subset of pooled compounds. "Dev." represents the standard deviation of the included scores. "% Pop" represents the percent of top-scoring compounds included (100 * N / Size). All values are in units of Daltons.

Program	Global			N=1	N=50			N=1000		
	Mean	Dev.	Size	Mean	Mean	Dev.	% Pop.	Mean	Dev.	% Pop.
***De novo* DOCK**	432.3	53.2	37883	485.3	469.1	17.9	0.13	456.9	23.8	2.64
GANDI	442.9	34.8	523	471.3	456.8	30.1	9.56	---	---	---
LigDream	390.1	73.8	21586	445.2	462.3	35.1	0.23	441.5	48.9	4.63
AutoGrow4										
Gen. 0	227.7	95.7	244	434.2	359.3	45.7	20.49	---	---	---
Gen. 1	319.2	80.9	5402	472.2	450.7	17.0	0.93	408.2	46.2	18.51
Gen. 5	384.3	84.8	27993	458.2	465.7	10.5	0.18	461.1	15.3	3.57

Table 23. Quantitative estimate of drug-likeness (QED) of *de novo* CADD-generated compounds.

"Gen. #" indicates the AutoGrow4 generation number (PARPi lead-optimization runs). "Global" indicates all compounds of a given generation or produced by a given program. "N" indicates the number of top-scoring compounds selected. All runs of each respective program were pooled together. "Mean" represents the mean of the subset of pooled compounds. "Dev." represents the standard deviation of the included scores. "% Pop" represents the percent of top-scoring compounds included (100 * N / Size). The reported means and standard deviations are based on QED scores (unitless), calculated using Scopy's implemintation of QED with average property weights (i.e., QED_{Mean})[166,267]. QED scores range from 0.0 to 1.0, where 0.0 indicates that a compound has all unfavorable properties and 1.0 indicates that a compound contains all favorable properties[166,267].

Program	Global			N=1	N=50			N=1000		
	Mean	Dev.	Size	Mean	Mean	Dev.	% Pop.	Mean	Dev.	% Pop.
De novo										
DOCK	0.40	0.15	37883	0.54	0.38	0.13	0.13	0.37	0.11	2.64
GANDI	0.45	0.14	523	0.53	0.47	0.12	9.56	---	---	---
LigDream	0.48	0.20	21586	0.45	0.40	0.15	0.23	0.42	0.16	4.63
AutoGrow4										
Gen. 0	0.62	0.16	244	0.61	0.66	0.13	20.49	---	---	---
Gen. 1	0.62	0.17	5402	0.46	0.51	0.10	0.93	0.54	0.14	18.51
Gen. 5	0.46	0.18	27993	0.51	0.46	0.10	0.18	0.46	0.11	3.57

4.3.2.4.2 Lipinski*, Ghose, and PAINS Applied to *de novo* CADD-Generated Compounds

AutoGrow4's predefined chemical filters help refine lead candidates and reduce the resources wasted on evaluating non-drug-like compounds. I applied the Lipinski*, Ghose, and PAINS filters to all AutoGrow4 runs, thereby removing unfavorable compounds prior to the computationally expensive processes of 3D conversion and docking. Unlike AutoGrow4, the other three *de novo* CADD programs had only limited property filters. To facilitate comparison, I subsequently filtered all compounds produced by *de novo* DOCK, GANDI, and LigDream with the Lipinski*, Ghose, and PAINS filter. As a positive control, I also filtered the eleven PARPi that in part seeded these runs. Iniparib was the only seed PARPi that failed any of the filters (Ghose) because it had fewer non-hydrogen atoms than the 20 atom minima (Table 24).

The LigDream-generated compounds passed all three filters at the highest rate, only failing all three filters 46.26% of the time (Table 24). In contrast, *de novo* DOCK compounds failed all three filters 61.65% of the time, the most of any program considered (Table 24). These percentages indicate that these programs dedicate around half of the available computational resources to generating compounds that are undesirable according to cheminformatics standards, which are based on what is known about existing drugs. Additionally, by outputting so many leads with so little chance of being developed into drugs, these programs require the users to test more ineffectual compounds to find good candidates.

Results published in Skalic *et al.* (2019)[213] showed that LigDream-generated compounds tend to fail the PAINS filter less frequently when the probabilistic recurrent neural network is enabled, than when it is disabled[213]. My results are consistent with this study. In applying

LigDream to PARPi lead optimization, I found that compounds produced using recurrent-neural-network sampling failed the PAINS filter 6.64% of the time, compared to 8.25% when recurrent-neural-network sampling was disabled (Table 24). Similarly, the LigDream publication reported slightly higher PAINS-filter fail rates when disabling the probabilistic recurrent neural network (14.4% vs. 13.5% without)[213]. Again, these rates are consistent with my results, given differences in source compounds and run conditions.

Table 24. Drug-likeness filters applied to *de novo* CADD-generated compounds.
"Global" indicates all compounds of a given generation or produced by a given program. "N" indicates the number of top-scoring compounds selected. "All Three Filters" indicates filtering a population with the Lipinski*, Ghose, and PAINS filters. Filters were applied to the compounds after running each program and scoring/ranking the compounds. AutoGrow4 was excluded from this table because the final set of AutoGrow4-generated compounds already passed these three filters prior to 3D conversion with Gypsum-DL[6]. All values indicate the percent of the subpopulation that passed the filter(s).

Program	Lipinski* Filter			Ghose Filter			PAINS Filter			All Three Filters		
	Global	N=1000	N=50	Global	N=1000	N=50	Global	N=1000	N=50	Global	N=1000	N=50
***De novo* DOCK**	73.76	65.60	66.0	42.22	26.10	20.0	97.77	99.30	100.0	38.35	23.50	20.0
GANDI	98.09	---	94.0	39.01	---	32.0	100.00	---	100.0	38.81	---	30.0
LigDream	65.33	57.50	54.0	65.16	49.90	46.0	92.75	93.80	90.0	53.74	43.60	40.0

4.3.2.4.3 Toxicity Filters Applied to *de novo* CADD-Generated Compounds

To assess the toxicity of compounds produced by the different *de novo* CADD programs, I filtered all generated compounds with two ADME-PK filters: the idiosyncratic filter[166,268] and the SureChEMBL filter[166,269,270]. The idiosyncratic filter, provided by the Python library Scopy[166], identifies compounds that are prone to cause infrequent (between 1-in-1,000 to 1-in-10,000) events of xenobiotic-induced toxicity (i.e., idiosyncratic adverse drug reactions)[268]. Idiosyncratic reactions often go unnoticed in the early and middle stages of drug development because they influence only small portions of the population and because animal models are unreliable for predicting these reactions[268]. Therefore, early detection and elimination of idiosyncratic-prone functional groups can reduce risk and save both time and resources during further drug development[268]. The SureChEMBL filter, also provided by the Python library Scopy[166], is a substructure-based filter that eliminates compounds containing any of 164 toxicophores (i.e., moieties associated with environmental toxicity or human cytotoxicity)[166,269,270]. Early elimination of toxicophore-containing compounds reduces the risk of off-target effects and similarly saves time and resources[269,270]. Both filters were also applied to the eleven seed PARPi, which served as positive controls. Iniparib was the only seed PARPi that failed the idiosyncratic filter, and none failed the SureChEMBL filter.

Most subpopulations of AutoGrow4 compounds passed the toxicity filters at a higher percent than the subpopulations of non-AutoGrow4 compounds (Table 25). AutoGrow4 and *de novo* DOCK had higher idiosyncratic-filter success rates than GANDI or LigDream in terms of both their top 50 and 1,000 best docking scored compounds as well as the compounds of their

entire populations (Table 25). Similarly, most AutoGrow4 generations passed the SureChEMBL filter at higher rates than the other programs (Table 25). GANDI had the lowest pass rates for all filters and all subpopulation sizes.

Table 25. Comparison of *de novo* CADD-generated compounds with ADME-PK filters. "Gen. #" indicates the AutoGrow4 generation number (PARPi lead-optimization runs). "Global" indicates all compounds of a given generation or produced by a given program. "N" indicates the number of top-scoring compounds selected. All values indicate the percent of the subpopulation that passed a given filter.

Program	Idiosyncratic Filter			SureChEMBL Filter		
	Global	**N=1000**	**N=50**	**Global**	**N=1000**	**N=50**
***De novo* DOCK**	92.42	85.40	88.0	57.88	65.10	70.0
GANDI	50.86	---	50.0	33.65	---	40.0
LigDream	67.28	80.60	82.0	61.55	60.60	48.0
AutoGrow4						
Gen. 0	97.95	---	100.0	76.23	---	80.0
Gen. 1	86.43	92.10	86.0	68.35	72.90	60.0
Gen. 5	75.95	89.00	94.0	54.00	77.80	70.0

4.4 Conclusions

In this section, I used PARPi lead-optimization techniques to compare AutoGrow4 to a traditional VS as well as to three *de novo* CADD programs. I showed that AutoGrow4 performed better than the other current programs by several metrics.

To compare AutoGrow4 and VS lead optimization, I focused primarily on how well they generated novel PARPi with improved predicted binding affinities. Both approaches found new compounds that are similar to known PARPi but with improved docking scores (Table 16 p.202 and Figure 28 p.204). When testing a comparable number of compounds, AutoGrow4's GA approach yielded better-scoring compounds than the VS approach after evolving only a single generation (Table 16 p.202 and Figure 28 p.204). Extended to the fifth generation, the mean docking score of AutoGrow's top 1,000 compounds was better than that of the best-scoring compound in the VS (Table 16 p.202). Of course, leads found via any high-throughput computational method (VS or AutoGrow4) can generally benefit from further optimization via more advanced computational methods (e.g., molecular dynamic simulations) or biochemical testing.

My comparison of AutoGrow4 and VS has its limitations because small changes to variables could have significant impacts on the results. For instance, if the Tanimoto cutoff used to select the PARPi-like compound dataset was stricter, the VS dataset would have consisted of compounds more closely related to known PARPi, which may have yielded even better scoring compounds; however, this would have also limited the chemical diversity of the VS library.

Additionally, restricting the VS search to the compounds most similar to known PARPi increases the chance of encountering intellectual property restriction issues[162].

AutoGrow4 performed as well as or better than the three alternative *de novo* CADD programs in virtually all considered metrics. As predicted, AutoGrow4 excelled at producing compounds with high predicted binding affinities while maintaining synthesizability and favorable ADME-PK properties. Because of its filters and GA approach, AutoGrow4 produced more compounds that passed toxicity and drug-likeness filters than other programs. Overall, the CADD techniques explored in this thesis are powerful tools for finding initial candidate compounds for further evaluation.

4.5 Acknowledgments

I would like to thank the following people for their contributions to the AutoGrow4 codebase and the analysis. Dr. Jacob Durrant generated the original idea for revising his line of AutoGrow software. He provided guidance in coding, experimental design, co-authoring the AutoGrow4 publication, editing this chapter of the dissertation, testing AutoGrow4 on macOS, performing the VS, and generating high-resolution versions of figures for the publication and this dissertation.

I would like to thank Dr. Bennett Van Houten for his suggestion that I apply AutoGrow4 to PARP-1.

Additional contributions to AutoGrow4 include: Patrick J. Ropp for useful discussions and programming tips; Erich Hellemann for discussions and help with SMARTS reactions; Pauline Spiegel for manuscript editing and tutorial testing; Yuri Kochnev for compiling QVina2 for macOS; Kevin C. Cassidy for help with molecular rendering; and Harrison Green for help developing AutoGrow4 accessory scripts. I would also like to thank the University of Pittsburgh's Center for Research Computing for providing helpful computer resources. The default fragment libraries included with AutoGrow4 were derived from a subset of the ZINC database (https://zinc.docking.org/). I thank ZINC for allowing me to distribute these fragment libraries to AutoGrow4 users.

4.6 Author Contributions

Jacob O. Spiegel wrote and tested the entire AutoGrow4 codebase and all accessory scripts provided in the AutoGrow4 download. Many libraries and other programs are used by AutoGrow4 and have been cited throughout the code and all related publications. I designed and performed the AutoGrow4 lead-optimization runs, and Dr. Jacob Durrant designed and performed the VS run. Dr. Jacob Durrant and I both analyzed the VS data and authored the publication of AutoGrow4. Only sections that I wrote were used verbatim in this dissertation. All writing, figures, and tables in this chapter is original content written by Jacob O. Spiegel

5.0 Dissertation Summary

In this chapter, I will (1) overview the work presented in this dissertation; (2) propose future directions for the field of PARP-1 research, with a focus on PARP-1 drug inhibition; and (3) propose future directions for AutoGrow and the field of *de novo* CADD.

5.1 Overview of Dissertation

This dissertation catalogs the development and validation of AutoGrow4 and its application to poly (ADP-ribose) polymerase-1 (PARP-1).

5.1.1 Summary of Chapter 2

In Chapter 2, I detailed the implementation choices that make AutoGrow4 a milestone in the AutoGrow program series. AutoGrow4 remains true to its origins by employing a genetic algorithm to create novel compounds and by using protein-ligand docking as the scoring metric, but I have significantly altered the underlying algorithm. My important improvements include redesigning the seed selection process and adding a secondary fitness metric (i.e., diversity score) that assesses ligand uniqueness. These innovations both delay convergence and help avoid local optima trapping. Through carefully designed benchmarks, I demonstrated that AutoGrow4 is faster and more effective at predicting compounds with improved docking scores than its most recent

predecessor, AutoGrow3, in part because it samples multiple 3D variants per compound and evaluates the zeroth generation. Additionally, I designed the new codebase to be expandable so AutoGrow4 can be updated as cheminformatic and docking programs advance. Lastly, AutoGrow4 incorporates new ADME-PK filters so users can better focus their search of chemistry space on drug-like compounds. These improvements better equip AutoGrow4 for modern CADD studies.

5.1.2 Summary of Chapter 3

Chapter 3 describes several AutoGrow4 runs in which I designed novel candidate PARP-1 inhibitors. These runs are useful for five main reasons: (1) they generated novel potential PARP-1 inhibitors (PARPi) with stronger predicted binding affinities than those of known PARPi; (2) they validated AutoGrow4 as a tool for *de novo* drug design, lead optimization, and hypothesis creation; (3) they identified novel potential PARPi that are predicted to bind to both a phosphorylated (PARPi-resistant) and nonphosphorylated PARP-1 catalytic pocket; (4) they predicted novel inhibitors to target the Zn1-Zn3 interface; and (5) they revealed new insights into the physiochemical environment of the Zn1-Zn3 interface.

In the lead-optimization runs, AutoGrow4 created a novel compound (Compound 5) with a docking score of -14.7 kcal/mol after only a single crossover event. This score surpasses those of known PARPi (Figure 17 p.144). Further analysis revealed that Compound 5's predicted binding to PARP-1 mimics that of both its parent PARPis, olaparib and CEP-9722 (Cephalon). The key protein-ligand interactions and the novel structures presented in this dissertation will aid those who wish to design novel PARPi that target the catalytic pocket.

I also evaluated the biological relevance of catalytic-site-targeted compounds generated during the AutoGrow4 large-scale *de novo* run. The run was seeded with random small molecules, blinding it from prior knowledge about known PARPi, so this run doubled as a test of AutoGrow4's ability to predict compounds designed for a targeted pocket. The resulting compounds' binding poses mimicked those observed in crystallographic structures of experimentally verified PARPi bound to PARP-1, in terms of predicted binding affinity, chemical structure, and predicted protein-ligand interactions.

I also used the *de novo* run to demonstrate how results from AutoGrow4 can help researchers generate new hypotheses. Because AutoGrow4 can test hundreds of thousands of compounds in a single run, the amount of information generated can be used to direct numerous subsequent experiments. For example, analysis of the *de novo* run detected repeated interactions between the best-scoring compounds and a number of pocket residues. These results led me to hypothesize that developing catalytic-pocket inhibitors that do not interact with Y907 (a catalytic-site residue that confers PARPi resistance when phosphorylated[142]) will be difficult, so I performed subsequent lead-optimization runs that targeted the pY907-PARP-1 structure. These prevalent interactions could also inform future studies such as quantitative structure-activity relationship (QSAR) and site-directed mutagenesis.

The results of these runs contribute to the growing realization that there is a need for PARP-1-based chemotherapies that do not rely on the same set of protein-ligand interactions typical of current PARPi. All FDA-approved PARPi bind to the PARP-1 catalytic pocket and rely on many of the same interactions for binding[17,18]. Additionally, despite the chemical diversity of the compounds generated in my AutoGrow4-guided searches, a similar set of consistent protein-ligand

interactions was prevalent among the best-scoring compounds generated by both the catalytic-site-targeted large-scale *de novo* and initial lead-optimization runs (Table 8 and Table 9 on p.152 and p.155, respectively).

For example, in my pY907-PARP-1 lead-optimization runs, 24% of the 100 best-scoring compounds formed π-π stacking interactions with pY907. Because phosphorylation of Y907 by c-Met has been verified as a PARPi resistance mechanism[142], treatment strategies that rely on this common interaction are vulnerable to resistance mechanisms. These results, combined with the growing number of identified PARPi resistance mechanisms[30,43,95,129,142,272], illustrate the need to diversify PARP-1 cancer treatments.

Finally, I used computational hot-spot mapping, paired with a thorough literature search, to identify two non-catalytic pockets located near the Zn1-Zn3 and Zn1-WGR-HD interfaces. Analysis of the physiochemical properties of both pockets led me to conclude that the Zn1-Zn3 pocket was more druggable ("Chapter 3.3.5.1: Selecting a Druggable Non-Catalytic Pocket"). AutoGrow4-guided CADD targeting the Zn1-Zn3 pocket produced many novel compounds that were predicted to bind with high affinity (Figure 26 p.177). The AutoGrow4-generated compounds that target the Zn1-Zn3 pocket are more flexible (i.e., had a higher ratio of rotatable to rigid bonds) and had smaller ring structures than those that target the catalytic site (Table 13 p.178). Additionally, the Zn1-Zn3 compounds formed fewer stacking and electrostatic interactions than the catalytic-site compounds, instead more frequently binding via hydrophobic and hydrogen-bond interactions (Table 14 p.181).

Further analysis of the Zn1-Zn3 pocket revealed a protruding helix that separates two surface-accessible cavities within the pocket (Figure 27 p.183). AutoGrow4 evolved and selected

for compounds that contour around the protruding protein density to form interactions in both cavities. The best-scoring compound of the AutoGrow4 Zn1-Zn3-targeted runs, Compound 6, has two moieties angled at 120° that can straddle the protrusion and extend into both cavities (Figure 27 p.183). This last study revealed important insights into the structure of the Zn1-Zn3 pocket, provided several novel candidate PARPi that do not bind to the PARP-1 catalytic site, and showcased AutoGrow4's ability to customize novel ligands to unique protein pockets.

If computational resources were limitless, I would use molecular dynamics simulations of PARP-1 to evaluate the binding of AutoGrow4-predicted PARPi and to test their influence on PARP-1 dynamics. Microsecond-timescale molecular dynamics simulations are commonly used to assess protein conformational changes and protein-ligand binding[273]. However, these simulations often rely on molecular-mechanical force fields that are less computationally costly but also less accurate than quantum mechanical (QM) approaches[274]. With unlimited computational resources, QM molecular dynamics simulations could assess small-molecule binding to PARP-1. These simulations could determine protein-ligand binding kinetics and could even be used to assess the compound's PARP-1 DNA trapping activity. Additionally, because it is not currently possible to perform whole-cell simulations at atomic resolution, molecular dynamics simulations can miss potentially important interactions between PARP-1 and other compounds present *in vivo*. With limitless computational resources, I would perform partial or whole-cell simulations that would further improve accuracy and biological relevance. Lastly, if time and computational resources were truly unlimited, there would be sufficient computational resources to test all of chemistry space. CADD optimization techniques (e.g., the genetic algorithm implemented in AutoGrow4) would no longer be necessary. In such a scenario, I would develop a

virtual library consisting of all synthesizable, drug-like compounds, which could be used as a universal small-molecule drug library. I would screen this entire library to find the compounds that are the strongest PARP-1 inhibitors.

5.1.3 Summary of Chapter 4

In Chapter 4, I compared PARPi lead optimization using AutoGrow4 to (1) a virtual screen (VS) and (2) three alternative *de novo* CADD programs (*de novo* DOCK[162], GANDI[167], and LigDream[213]). Considered alongside the benchmark experiments from Chapter 2, my tests demonstrate that AutoGrow4 is superior at designing novel drug-like small molecules than similar free and open-source *de novo* CADD techniques. Although all approaches identified compounds that are similar to known PARPi, AutoGrow4 produced the compounds with the best docking scores. AutoGrow4 globally produced the most diverse compounds, and each AutoGrow4 generation was more diverse than the populations predicted by VS or the other three *de novo* CADD programs. Additionally, AutoGrow4 provides the most comprehensive set of ADME-PK filters[6,162,167,213]. These filters helped AutoGrow4 produce drug-like compounds that passed toxicity filters more frequently than compounds produced by the other three programs. AutoGrow4 also predicted more synthesizable compounds than *de novo* DOCK and GANDI.

If time and resources were not limiting factors, I would experimentally test the ligands predicted by each program in a wet-lab setting. Rather than comparing the predicted ligands in terms of docking score and predicted toxicity, I would compare them in terms of experimentally measured PARP-1 binding affinity, PARP-1 catalytic inhibition, PARP-1 trapping, and

cytotoxicity. Because computational approaches such as docking are imperfect in predicting protein-ligand binding affinity and pose stability[271], these wet-lab experiments would provide a more accurate assessment of the ligands predicted by each program. I describe an experimental protocol for validating candidate PARPi in "Chapter 5.2.1.3: Verifying AutoGrow4-Generated Leads," but current resources cannot feasibly conduct a thorough *ex silico* comparison of the 168,161 unique canonical SMILES (Table 18 p.209) predicted by the four tested *de novo* CADD programs (AutoGrow4, *de novo* DOCK, GANDI, and LigDream).

5.2 Future Directions of PARP-1 Inhibition

In this dissertation, I generated novel compounds that are predicted to bind PARP-1's catalytic pocket and DNA-binding-domain interface with higher affinity than FDA-approved PARPi. I also discovered several novel leads that are predicted to bind PARP-1's catalytic pocket even when Y907 is phosphorylated, thereby potentially circumventing the pY907 resistance mechanism[142]. Of course, these early-stage leads have only been assessed by molecular docking. Further *in silico* evaluation, synthesis, and experimental validation are the next steps in this drug design and development pipeline.

In this section, I will propose several future directions for PARPi design, including biological experiments. First, I will describe proposed experiments that focus on PARPi inhibition of the catalytic domain (i.e., orthosteric inhibition). These experiments will elucidate the mechanism of pY907-triggered PARPi resistance and verify novel AutoGrow4-generated

candidate PARPi. Second, I will propose directions for expanding treatment options of HR-deficient cancers that do not rely purely on PARP-1 orthosteric inhibition. These will include the design and validation of AutoGrow4-generated candidate PARPi that do not bind the catalytic site.

5.2.1 Future Directions of Orthosteric PARPi

The aims of the future experiments proposed in this section are to (1) understand the mechanism of pY907-mediated PARPi resistance; and (2) verify candidate PARPi leads that are less susceptible to the pY907 PARPi-resistance mechanism. The first step in addressing these aims is to generate a high-resolution structure of the catalytic site when Y907 is phosphorylated. Then I will suggest steps for further evaluation of AutoGrow4-generated leads and the eventual transition from *in silico* to wet-lab testing.

5.2.1.1 Understanding the Mechanism of pY907-Triggered PARPi Resistance

pY907 has been shown to confer PARPi resistance by weakening PARPi binding and increasing PARP-1 catalytic activity[142], but the specific molecular mechanism responsible for this effect remains unknown. A crystal structure of pY907 PARP-1 would help answer unresolved structural-biology questions regarding the mechanism of pY907-mediated PARPi resistance, particularly if the crystallography captured PARP-1 bound to a PARPi or a non-hydrolyzable NAD+ analog such as benzamide adenine dinucleotide (BAD). Such crystals would reveal how pY907 influences small-molecule binding and whether there are conformational changes within the pocket that account for the increased catalytic activity.

Whether pY907 alters the binding affinity of the endogenous ligand NAD+ is also uncertain. Given the structural similarities between PARPi and NAD+[30,75] (Figure 8 p.33), I predict that pY907 does weaken NAD+ binding. But if so, why the increase in NAD+ catalytic activity? On the other hand, if pY907 instead increases the binding affinity of NAD+, thereby increasing enzymatic activity, why the observed decrease in PARPi affinity?

As a possible explanation, I hypothesize that Y907 phosphorylation causes additional conformational changes—not captured in my computational models—that account for both the increased catalytic activity and the decreased PARPi binding affinity. Alternatively, phosphorylation may simply affect the binding of PARPi and NAD+ differently, so that it negatively impacts PARPi binding but strengthens NAD+ binding. A third option is that PARP-1 catalytic activity increases because phosphorylation affects other factors responsible for PARP-1 activity that are unrelated to the catalytic pocket itself, such as interactions with the acceptor site or the stability of HD, with consequential impacts on the ligand turnover rate.

To distinguish between these possible explanations for pY907-mediated PARPi resistance, I propose (1) calculating the binding kinetics of NAD+ and several other NAD+ analogs to pY907 in order to determine if pY907 affects PARPi and NAD+ binding differently, and (2) resolving a high-resolution structure of pY907-PARP-1 to reveal whether pY907 induces substantial catalytic-pocket conformational changes. Calculating the binding kinetics of NAD+ and other NAD+ analogs can be accomplished following the protocols in Du *et. al.* (2016)[142]. However, because determining a high-resolution structure of pY907-PARP-1 requires a custom protocol, I detail that proposed experiment in the following section.

5.2.1.2 Determining the Structure of the pY907-PARP-1 Catalytic Pocket

In this section I propose an approach to determining the structure of the pY907-PARP-1 catalytic pocket that can also be used in future CADD efforts targeting the pY907-PARP-1 catalytic site. Determining a high-resolution structure of pY907 PARP-1 would improve the accuracy of future CADD efforts targeting the pY907-PARP-1 catalytic site. The modelling method I used to produce the pY907-PARP-1 structure in "Chapter 3.2.4: Phosphorylated Y907 PARP-1 Lead-Optimization" cannot account for any conformational changes caused by phosphorylation.

I propose resolving a high-resolution pY907 PARP-1 structure using X-ray crystallography. The first step to obtain phosphorylated PARP-1 is to purify truncated PARP-1 using PARP-1 constructs that are known to crystallize, such as those used to produce the 4R6E[17] or 4DQY[45] structures. Next, I would purify the constituently active, cytosol-localized c-Met mutant $MET^{\Delta 7\text{-}8}$ (which lacks the regions encoded by exons 7 and 8[275]) using His-tag and Ni-affinity chromatography, as others have done[275]. I would then incubate the purified PARP-1 with an excess of purified $MET^{\Delta 7\text{-}8}$, thereby encouraging PARP-1 phosphorylation. With a similar incubation, Du *et al*. (2016)[142] were able to obtain pY907 PARP-1 protein, although they did not attempt to crystallize pY907 PARP-1[142]. Following the incubation, I would further purify PARP-1, remove $MET^{\Delta 7\text{-}8}$ using chromatography, and then proceed with crystallography.

If crystals do not form or if $MET^{\Delta 7\text{-}8}$ does not phosphorylate Y907, molecular dynamics simulations starting with my pY907-PARP-1 model could also reveal how the catalytic pocket

responds to Y907 phosphorylation. This approach is a promising future direction because it would provide better sampling of the catalytic pocket than does my current static pY907-PARP-1 model.

5.2.1.3 Verifying AutoGrow4-Generated Leads

The work described in this dissertation has produced a large dataset of AutoGrow4-generated leads. In this section, I propose a process of verifying that these predicted ligands are in fact PARPi. I focus these future directions on candidates predicted to bind independent of the phosphorylation state of Y907, although similar evaluation and testing can be applied to candidates predicted to preferentially bind nonphosphorylated PARP-1.

5.2.1.3.1 Evaluating and Verifying AutoGrow4-Generated Leads as PARPi

The first step in evaluating the AutoGrow4-generated leads is to perform additional *in silico* testing. Because I have already generated many well scoring leads, I propose pooling the best compounds from the AutoGrow4 runs applied to both 4R6E:A and pY907-PARP-1. Next, I would filter for compounds that are predicted to be synthetically tractable, drug-like, and non-toxic by considering SA_Score[266], QED[166,267], and the idiosyncratic[166,268]/SureChEMBL[166,269,270] filters described in "Chapter 4.2.2.1.3: Post-Processing Compound Analysis". Selecting synthesizable compounds will expedite the transition to wet-lab testing, and selecting compounds that are drug-like and non-toxic will minimize issues such as poor solubility and idiosyncratic adverse drug reactions that may otherwise arise in later stages of testing[163,268].

Having selected the most promising compounds, I would next perform molecular dynamics simulations of PARP-1 and pY907 PARP-1 to sample multiple protein conformational states. I would run molecular dynamics simulations of the pY907 PARP-1 structure determined in "Chapter 5.2.1.2: Determining the Structure of the pY907-PARP-1 Catalytic Pocket"; however, the computationally modeled pY907-PARP-1 structure ("Chapter 3.3.4.1: Building the pY907 PARP-1 Structure and Orienting the Phosphate Moiety") would serve as a backup if crystallography efforts fail. I would reassess each compound's predicted binding affinity using an ensemble-docking approach, wherein compounds are docked into multiple conformations to account for protein flexibility. This high-throughput reassessment would further prune the number of candidate ligands.

Once a tractable number of compounds are selected, I would begin synthesizing the compounds. A collaboration with a trained synthetic chemist would be invaluable during this stage. Alternatively, screening commercially available compounds that are similar to my identified leads and removing those that have poor docking scores may identify promising candidates without requiring custom compound synthesis.

Once I have synthesized or purchased compounds, I would assess the ability of each compound to inhibit PARP-1's catalytic activity using an *in vitro* PARP-1 inhibition assay such as the fluorescence-detection-based assay designed by Putt and Hergenrother[276]. After incubating PARP-1, nicked DNA, NAD+, and each candidate inhibitor[276], this high-throughput assay converts any remaining NAD+ into a fluorophore. A fluorescent plate reader can then be used to determine the concentration of NAD+. This initial screen would allow me to eliminate compounds that do not inhibit PARP-1 catalytic activity.

I would next perform a cell-viability assay[75,277] to test whether the inhibitors are (1) lethal to BRCA-deficient cells, using HEK293T cells transfected with shRNA vector to knockdown BRCA1/2[142], (2) non-lethal to cells with functional BRCA proteins, using HEK293T cells transfected with control vector[142], and (3) lethal to BRCA-deficient cells that have upregulated c-Met, using HEK293T cells transfected with shRNA vector to knockdown BRCA1/2 and ectopically express constituently active c-Met mutant ($MET^{\Delta 7\text{-}8}$)[142,275]. This combined shRNA knockdown and ectopic expression enables a controlled comparison of the inhibitors. Ideally, several compounds would be lethal to BRCA-deficient cells independent of c-Met levels, and non-lethal/less lethal to BRCA-proficient cells. This entire process would require multiple rounds of inhibitor refinement and testing, but would ideally identify PARPi that could then progress to animal testing and eventually human clinical trials.

5.2.2 Future Directions of PARP-1 Inhibition and PARPi Drug Design

In this section, I will propose several future directions for designing PARPi that do not bind to the PARP-1 catalytic site, as well as several other strategies for treating HR-deficient cancer cells. To effectively recruit DNA repair proteins to sites of DNA-damage, PARP-1 must be able to bind DNA-damage sites[17,45,46], activate catalytic activity through interdomain communication[18], and catalyze NAD+ to poly (ADP-ribose)[61]. Both reduced PARP-1 catalytic activity and increased PARP-1 trapping on DNA make PARPi effective against HR-deficient cells[75,77].

My research illustrates the critical role Y907 plays in high-affinity, catalytic-pocket binding. Considering that all FDA-approved PARPi target this pocket and rely on interactions with Y907[1–4,17,18], and that phosphorylation of Y907 by c-Met confers resistance to multiple PARPi[142], future efforts for PARPi design should focus on binding sites outside of the catalytic binding pocket.

I showed that one potential binding site is located near the Zn1-Zn3 interface. My preliminary AutoGrow4-guided CADD efforts targeting this secondary pocket identified novel compounds predicted to bind with high affinity and provided insight into the pocket environment. As proposed in "Chapter 5.2.1.3.1: Evaluating and Verifying AutoGrow4-Generated Leads as PARPi," I recommend reevaluating candidate leads by docking them into an ensemble of protein conformations. Furthermore, I propose wet-lab validation of these compounds to verify that they bind to the pocket and inhibit PARP-1 ("Chapter 5.2.1.3.1: Evaluating and Verifying AutoGrow4-Generated Leads as PARPi"). An additional potential binding site is located at the Zn1-WGR-HD interface. Future CADD exploration and wet-lab verification of this interface may also yield additional PARP-1 inhibitors.

Future studies could also focus on treatments that thwart additional PARPi resistance mechanisms by combining both PARP-1 inhibitor(s) and c-Met inhibitor(s)[142]. These approaches will not work against all resistance mechanisms, such as reversed *BRCA* gene(s) mutations that restore HR function[30,129,272], but they may yield improved treatment outcomes for HR-deficient cancer patients.

5.3 Future Directions of AutoGrow

AutoGrow4 is a major milestone in the AutoGrow program series that includes many new features such as improved ligand handling and the introduction of a plugin architecture. To stay current with future computer-hardware and algorithmic advances, future AutoGrow releases will no doubt require additional refinements. In this section I detail features that I hope will one day be integrated into AutoGrow as technology and computational resources allow.

5.3.1 Protein-Ligand Assessment: The Balance between High-Throughput Testing and Accuracy

Future versions of AutoGrow4 could incorporate more advanced methods for sampling protein-ligand poses and assessing protein-ligand interactions. AutoGrow4 is currently configured for early-stage drug design and so relies on high-throughput protein-ligand docking to assess compound fitness. Although AutoGrow4 provides a plugin option that allows for the easy integration of alternative docking software and protein-ligand assessments methods, it currently does not provide predefined options for more advanced, computationally intensive techniques, which may be useful in the later stages of CADD. In this section, I will explore several of these advanced techniques.

Effective CADD requires one to balance the costs of computation, thoroughness of search, and accuracy of assessment[274]. Techniques such as protein-ligand docking tend to favor computational efficiency over accuracy. In contrast, quantum mechanical (QM) methods for

calculating protein-ligand binding affinities tend to favor accuracy over the overall computational cost[274]. Additionally, docking into a single rigid protein conformation is cost-effective but ignores the dynamics of proteins and protein-ligand binding[278]. But treating a protein as flexible during docking is computationally costly due to the high dimensionality of a protein's conformational space[279]. For instance, calculating the motions of binding-pocket sidechains alone requires the additional evaluation of perhaps ten to twenty residues[279] with an average of 1.85 rotatable bonds per residue[280].

Incorporating the option to treat proteins as flexible, as well as offering multi-conformation ensemble docking to account for larger motions[279], could improve the docking accuracy of AutoGrow4 and fulfill more user needs. On the other hand, ensemble docking does not improve search outcomes when applied to similar conformations of a rigid protein pocket[279]. Short molecular dynamics simulations can optimize pose placement and test pose stability[281], but they too are computationally expensive. Because most pipelines start with high-throughput techniques that are followed by increasingly accurate but costly reassessments, selecting the best approach for a given step is important. Future versions of AutoGrow should provide options that suit a variety of user needs.

5.3.2 Chemical Filters

A CADD program that considers docking scores alone without examining absorption, distribution, metabolism, excretion, and pharmacokinetics (ADME-PK) is unlikely to predict successful drugs[175]. AutoGrow4's expanded set of chemical-property filters gives it a major

advantage over other free and open-source *de novo* CADD programs ("Chapter 4.3.2: Comparison of AutoGrow4 Lead Optimization and other *De Novo* CADD Programs"). Future AutoGrow releases should continue to incorporate new filters and scoring functions to account for drug-likeness properties other than predicted binding affinity.

Expanding AutoGrow4's filter options to allow researchers to screen out compounds with undesirable ADME-PK properties should improve the success rate of predicted lead compounds. AutoGrow4 provides nine predefined chemical-property filters. The Lipinski[163], Lipinski*, Ghose[168], Ghose*, VandeWaterbeemd[169], and Mozziconacci[170] filters focus on physiochemical properties such as molecular weight, logP, and polar surface area. These are useful for predicting solubility and bioavailability but do not address drug metabolism or drug toxicity[163,168–170]. The BRENK[171], NIH[172,173], and PAINS[174] filters screen for false positives and off-target effects but do not address compound stability or drug toxicity[171–174]. The idiosyncratic[166,268] and SureChEMBL[166,269,270] filters, which I used in "Chapter 4.3.2.4.3: Toxicity Filters Applied to *de novo* CADD-Generated Compounds" to compare the compounds generated by different *de novo* CADD programs, could be incorporated into AutoGrow4 to provide useful toxicity filters.

The next release of AutoGrow4 could also incorporate ADME-PK properties into its fitness functions, thereby actively selecting for compounds with favorable ADME-PK properties rather than merely filtering out poor compounds. Because the AutoGrow4 chemical filters are designed to screen large numbers of compounds, they also potentially eliminate candidate compounds that have other favorable properties. For example, an average of 5,565.3 compounds were removed by the Lipinski*, Ghose, and PAINS filters per run in the fifth generation of the pY907-PARP-1 lead-optimization runs, which is approximately one compound removed for every compound that

passed. A scoring system that penalizes compounds that violate filter constraints without eliminating them entirely may effectively refine compound populations for ADME-PK properties.

Lastly, future releases of AutoGrow4 could incorporate measures of compound synthesizability into the filter and scoring-function options. AutoGrow4 generates compounds that often require custom synthesis before experimental testing, so filters and scoring functions that consider compound synthesizability will be useful.

5.3.3 Compound Generation

In this section I will discuss two alternative compound-generation methods that could be incorporated into AutoGrow4. AutoGrow4 currently creates novel compounds by crossover and mutation. Incorporating additional methods could improve the quality of generated compounds.

First, a machine-learning-based approach could help produce unique and chemically synthesizable compounds. In my comparison of AutoGrow4 and alternative *de novo* CADD programs, the machine-learning-based program LigDream[213] performed exceptionally well creating a diverse set of synthesizable and drug-like compounds; however, it did not compare favorably against AutoGrow4 in terms of average predicted binding affinity, likely because LigDream does not assess the predicted compounds relative to the target protein pocket ("Chapter 4.3.2: Comparison of AutoGrow4 Lead Optimization and other *De Novo* CADD Programs"). Using a program such as LigDream to generate novel inhibitors within the AutoGrow framework could be a successful merge the strengths of these two programs.

Second, I propose incorporating an additional compound-creation operator that fragments parent compound(s) ("Chapter 3.3.2.2: AutoGrow4 Operators and Molecular Weight"). Although some crossover and mutation events result in child compounds that are smaller than the parent(s), the MW of AutoGrow4-generated compounds tends to increase with each successive generation ("Chapter 3.3.2.2: AutoGrow4 Operators and Molecular Weight") until the MW of the population reaches the limit imposed by the chemical-property filters. A user option for fragmenting compounds would mitigate this issue by introducing low-MW fragments back into the evolving populations.

5.3.4 Additional Improvements

In this section, I will describe several technical improvements that can further enhance AutoGrow4's ease-of-use, computational efficiency, and multiprocessing support.

First, providing a graphical user interface or webserver implementation would improve AutoGrow's ease of use, particularly for users who are not comfortable with command-line interfaces. AutoGrow4 is currently a command-line program that requires users to separately install several third-party programs and libraries. Despite AutoGrow4's extensive documentation describing installation and use (Appendix D: AutoGrow4 Manual and Tutorial), these requirements limit the userbase to researchers with at least some computational experience.

Second, future releases could also include improved parallelization options. In recent years, computational advances have come more from improved parallelization across many processors than from faster individual processors[282]. AutoGrow4 was the first AutoGrow program to allow

MPI parallelization, which permits job distribution across multiple computer nodes. This enables AutoGrow4 to use more central processing units (CPU) than are normally available on a single node. However, docking methods that use graphics processing units (GPU) can perform up to 90 times faster than CPU-based docking methods[283]. AutoGrow4 is currently a CPU-intensive program that is not configured for GPU processing. Future releases could focus on providing a GPU-based molecular-docking option.

Appendix A Additional AutoGrow4 Implementation Details

This appendix details additional AutoGrow4 implementation choices that were not pertinent to understanding the program or its application.

Appendix A.1 Parallelization

AutoGrow4 borrows multiprocessing code from the program Gypsum-DL[214]. Gypsum-DL's multiprocessing code provides the option to distribute jobs across multiple CPUs using two parallelization architectures: symmetric multiprocessing (SMP), which is used by most personal computers and smaller single-node clusters, and message passing interface (MPI), used on MPI-enabled clusters[214]. AutoGrow4 supports both of these options and allows users to scale jobs to their own needs and resources. AutoGrow4's introduction of MPI-enabled parallelization allows significantly larger-scale runs to be performed in a reduced amount of time.

Appendix A.2 Dependencies

AutoGrow4 runs on Linux and macOS. Users are strongly encouraged to employ the AutoGrow4 Docker (Docker, Inc.) container included with the download, which automatically

installs all dependencies and further enables use on Windows. AutoGrow4 is Python 2.7 and 3.7 compatible.

AutoGrow4 requires a Python environment with the following third-party Python libraries installed: RDKit[165], numpy[284,285], scipy[286], matplotlib[287], and func_timeout (available via *pip* installation or https://github.com/kata198/func_timeout download). The third-party Python library mpi4py is also required for MPI multiprocessings[288]. macOS requires the additional installation of the coreutils package (available via the homebrew package manager).

To simplify installation, AutoGrow4 pre-packages many of the dependencies. AutoGrow4 comes installed with the most current versions of AutoDock Vina (version 1.1.2)[185], QuickVina2 (version 2.1)[205], NNScore1 (version 1.1)[233], NNScore2 (version 2.02)[234], MolVS (version 1.1.0) (https://molvs.readthedocs.io), Gypsum-DL (version 1.1.2)[214], and Dimorphite-DL (version 1.2.2)[231]. Additionally, a Docker container (Docker, Inc.) that will create a Linux environment with all the necessary dependencies to run AutoGrow4 is provided in the download.

Docking using Vina and QVina2 requires conversion of PDB to PDBQT format. AutoGrow4 provides two predefined options, AutoDock MGLTools and Open Babel[164,215], but independent installation of either AutoDock MGLTools or Open Babel is required. Instructions for AutoGrow4 installation and its dependencies are provided in the AutoGrow4 tutorial (Appendix D: AutoGrow4 Manual and Tutorial).

Appendix A.3 Accessory Scripts

AutoGrow4 provides several tools for processing and analyzing AutoGrow4 data. With the increased population size enabled by MPI multiprocessing, manually analyzing the initial results from an AutoGrow4 run may be impractical. I have written several scripts to produce initial analyses of results, including one that plots the average docking score for each generation and another that tracks a molecule's ancestry and creates images of all parent molecules. I also provide scripts for file handling and file conversion, as well as for compression/decompression of population data to allow for efficient file transfer. For lead optimization experiments, I offer a script that fragments user-provided lists of compounds as well as scripts that help prepare custom complementary molecule libraries and source compound libraries. These accessory programs are designed to prepare experiments and analyze large population data, and require less custom code. An in-depth explanation of how to use AutoGrow4's accessory tools is provided in the tutorial.

Appendix A.4 Miscellaneous

AutoGrow4 has been carefully designed to maximize maintainability, expandability, readability, and modularity. Its detailed and well-documented modules allow developers and advanced users to test and edit individual sections of the code. Many of the modules such as the *in silico* reactions, docking software scripts, scoring functions, and filtration modules provide a plugin option to easily incorporate custom code as well as a framework to expand AutoGrow4 as cheminformatics and CADD grow as fields.

AutoGrow4 also provides several new options that improve usability. AutoGrow4 runs create several files per compound (e.g., Gypsum-DL SDF, PDB/PDBQT, and docking output files). One large file transfers faster and requires less storage memory than many separate files of equal cumulative size. For this reason, AutoGrow4 provides options to purge unnecessary files (Gypsum-DL logs and SDF files) at the end of each generation and to concatenate all 3D files (PDB, PDBQT, docked files and logs, rescoring files) into a single file to be compressed. This reduces file size and improves file transfer speeds. I include an accessory script that decompresses and disjoins the files.

AutoGrow4 also provides a plotting option that generates a line graph at the end of each simulation, as well as an accessory script that creates the same line plot after the simulation in case the user needs to replot or add control lines, such as the PARPi lines in Figure 15A.

Appendix B Gypsum-DL

This appendix details my work and contributions to the Gypsum-DL codebase and publication. Gypsum-DL is a free and open-source program that assigns 3D coordinates to small molecules by enumerating ionization, tautomeric, chiral, cis/trans isomeric, and ring-conformational states.

The Gypsum-DL publication is published under the Creative Commons Attribution 4.0 International License which "allows unrestricted use, distribution, and reproduction in any medium"[214]. This appendix has been adapted and reprinted with permission from:

> Ropp, Patrick J†, **Jacob O Spiegel**†, Jennifer L Walker, Harrison Green, Guillermo A Morales, Katherine A Milliken, John J Ringe, & Jacob D Durrant. (2019) Gypsum-DL: an open-source program for preparing small-molecule libraries for structure-based virtual screening. *Journal of Cheminformatics*, 11, 34. http://doi.org/gf48dh.
>
> † Jacob O. Spiegel and Patrick J. Ropp should be regarded as joint first authors.

My contributions to the project include work on the Gypsum-DL codebase (development, testing, optimization, and documentation), experimental design, background research, and manuscript writing. As the contents of the Gypsum-DL publication are not entirely my original words, this appendix is a rewrite of important parts of the paper, focusing on my contributions and the information necessary to understand them.

The figures and tables are taken directly from the Gypsum-DL publication with rewritten legends. Dr. Jacob Durrant provided editorial guidance on this appendix. All words in this appendix are my own original words.

Appendix B.1 Introductions

As evidenced by AutoGrow4 ("Chapter 2.0: AutoGrow4: Implementation and Benchmarks"), a single computer-aided drug design (CADD) study may require both 1/2D representations (such as Daylight's SMILES and flat SDF files) and 3D representations (such as 3D SDF and PDB files). Additionally, virtual screening (VS) techniques rely on a finite database of compounds[289,290] that often only contain a single variant per molecule, resulting in isomers being missed by the screen. This can limit the success of the search. Therefore, accurate conversion between representations is vital to many CADD pipelines.

Gypsum-DL is a free and open-source Python-based program for preparing 3D small-molecule representations. It accepts Daylight's SMILES and flat SDF files and converts them to 3D SDF and optionally PDB format. To assign 3D coordinates, Gypsum-DL enumerates ionization, tautomeric, chiral, cis/trans isomeric, and ring-conformational forms for each molecule.

Gypsum-DL is available for free download at http://durrantlab.com/gypsum-dl/ under the terms of the Apache License, Version 2.0.

Appendix B.2 Implementation

Appendix B.2.1 Gypsum-DL Workflow

Since SMILES notation allows multiple compounds in the same SMILES string, Gypsum-DL begins generating 3D representations by desalting those compounds (Appendix Figure 1). If an input compound has multiple small molecules or fragments, Gypsum-DL assumes the largest fragment is the intended compound.

Gypsum-DL uses the Dimorphite-DL algorithm[231] to create separate models for each potential ionization state within the user-defined pH range. Dimorphite-DL[231] uses the cheminformatic Python library RDKit (http://www.rdkit.org). By default, Gypsum-DL uses a biologically relevant pH range (6.4-8.4).

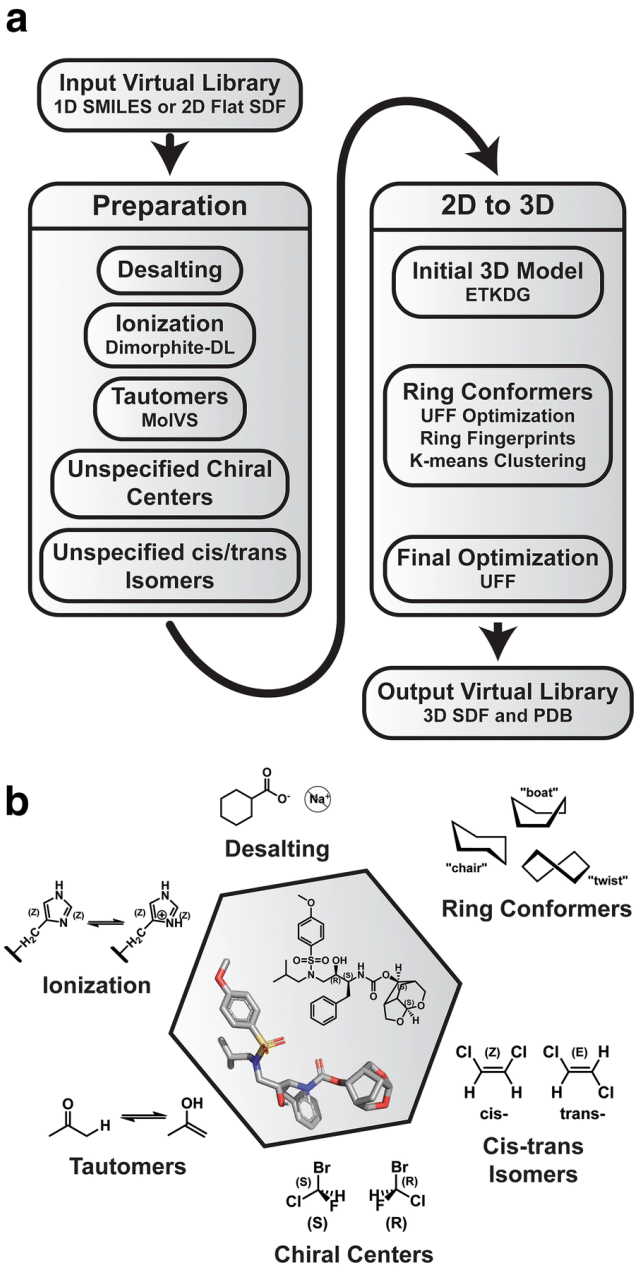
Next, Gypsum-DL uses MolVS (https://molvs.readthedocs.io), a Python-based tool that standardizes and validates chemical structures, to generate models for each tautomeric state (Appendix Figure 1). MolVS has its own limitations and occasionally generates problematic models. Gypsum-DL addresses this by rejecting models with an altered number of aromatic rings or chiral centers and filtering out models that contain chemically improbable tautomeric forms.

Gypsum-DL then enumerates unspecified chiral centers and unspecified cis/trans double bonds (Appendix Figure 1). Gypsum-DL does this exhaustively, generating each isomer for each unspecified site.

The next step is to assign initial 3D coordinates (Appendix Figure 1). 3D conformations are generated using RDKit's Experimental Torsion with Knowledge Distance Geometry (ETKDG) method[291].

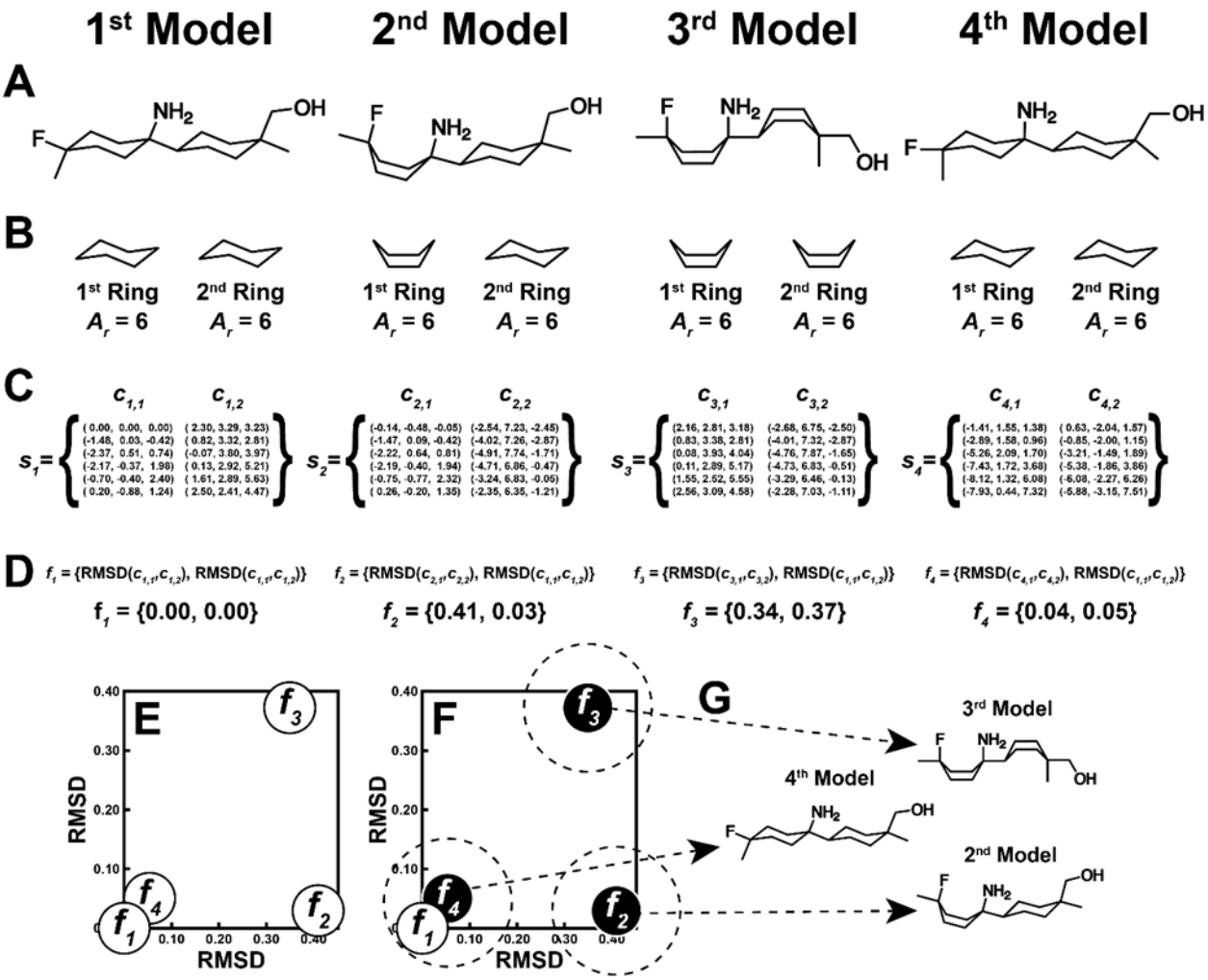
Gypsum-DL then addresses alternative non-aromatic ring conformations (Appendix Figure 1). Many docking programs, such as AutoDock Vina[185], treat rotatable bonds as flexible but consider rings rigid and essentially limit them to a single conformation. To generate alternative non-aromatic ring conformations, 3D models are optimized using RDKit's Universal Force Field (UFF) method[292]. The 3D coordinates of each non-aromatic ring are calculated for each model. The minimum root-mean-square deviation (RMSD) is calculated by comparing each ring to its corresponding ring in the first model (Appendix Figure 2). The list of ring RMSDs for a given model comprises its "ring-conformation fingerprint," which is used to describe a ring's conformational geometry and to cluster the 3D models using *k*-means clustering[293] (Appendix Figure 2). Because models may be similar to one another, Gypsum-DL reduces redundancy by retaining only the most central 3D model of each cluster (Appendix Figure 2).

Lastly, a final geometric optimization is performed (Appendix Figure 1) using RDKit's UFF method[292]. The resulting 3D model(s) is output to SDF and optionally to PDB and HTML file(s).



Appendix Figure 1. The function and workflow of Gypsum-DL.

A) Gypsum-DL begins with a set of input molecules. Once the molecules have been desalted and alternative ionization, tautomeric, and isomeric forms have been considered, the models are assigned 3D coordinates using the ETKDG method. Next, models are optimized using an UFF. Lastly, the 3D compounds are output to 3D SDF files and optionally to PDB files. B) Examples of the output from each Gypsum-DL step.



Appendix Figure 2. **Ring conformation sampling schematic.**

A) Multiple 3D variants are generated. B) All non-aromatic rings are extracted. C) Coordinates are calculated for each extracted ring. D) Extracted rings are converted to a fingerprint using the RMSD between each ring and the corresponding ring from the first model. E, F) The fingerprints are clustered using the k-means clustering algorithm. Fingerprints are represented by small circles on the plots, while clusters are represented by large dashed circles. To reduce redundant structures, only the most central fingerprint from each cluster is selected to progress. Selected fingerprints are represented by black circles. G) The selected models are geometrically different.

This figure is modified and reprinted with rights and permission under the Creative Commons Attribution 4.0 International License[214].

Appendix B.2.2 Managing Combinatorial Explosion

Gypsum-DL enumerates possible variations of multiple properties (ionization, tautomeric, chiral, isomeric, and ring-conformational states), meaning it can generate an intractable number of output variants. Testing each model as a result of this combinatorial explosion would be costly for both Gypsum-DL and for subsequent calculations such as docking the large number of output structures. To mitigate this, Gypsum-DL prunes the set of variants following each step in the algorithm according to two user-controlled parameters, *max_variants_per_compound* (*m*) and *thoroughness* (*t*). Following a given step, Gypsum-DL randomly selects *m* x *t* variants and converts each into a 3D conformer using RDKit's ETKDG method[291]. The conformers' energies are evaluated using RDKit's UFF[292]. Of the *m* x *t* variants tested, *m* will be kept by Gypsum-DL. Increases in *m* or *t* increase the computation cost.

Appendix B.3 Methods

Appendix B.3.1 Gypsum-DL Benchmarking

Gypsum-DL has been designed to run tasks simultaneously on multiple processors. Distributing the workload reduces the amount of time required to complete a given task. Gypsum-DL's *job_manager* parameter allows users to select one of three task-management options: "serial," "multiprocessing," or "mpi" mode. The serial mode restricts Gypsum-DL to a single processor. This allows Gypsum-DL to be used in low-resource environments and in programs that

have an independent job manager. Windows operating system is currently limited to the serial mode. To test Gypsum-DL's performance and scalability across multiple processors, we ran benchmarks in both multiprocessing and mpi mode.

The multiprocessing mode distributes jobs across processors on the same computer using symmetric multiprocessing (SMP) architecture. Multiprocessing mode benchmarks tested a late-stage beta version of Gypsum-DL on 1,000 compounds. Benchmarks were performed in triplicate on a 24-core Skylake processor provided by the University of Pittsburgh's Center for Research Computing (CRC).

The mpi mode distributes jobs across multiple computer systems using message passing interface (MPI) architecture. The mpi mode benchmarks tested 20,000 compounds and were performed on MPI-enabled computing clusters with 28-core Broadwell Processors provided by the CRC. These computing clusters were networked using Intel's Omni-Path communication architecture. All experiments were performed in triplicate.

Appendix B.3.2 Impact on the Accuracy of Docking Pose Prediction

We compared the impact of file preparation using Gypsum-DL or Open Babel on docking pose prediction. We derived a library of 3,177 protein-ligand complexes from the PDBBind refined set, with 2,438 different small molecules[294,295]. The PDBBind refined set contains 4,463 high-quality complexes[294,295]; a complex was removed if the ligand had a molecular weight over 500 Dalton, the ligand was an amino acid or a peptide, the ligand contained atoms with improper

names, and/or the ligand file did not match the corresponding entry in the Protein Data Bank (https://www.rcsb.org/)[239].

SMILES strings for each ligand in the dataset were obtained from the Protein Data Bank[239]. Since we were testing Gypsum-DL and Open Babel's abilities to prepare 3D representations of small molecules, which includes predicting charge and isomer information, charge was neutralized and isomer information was removed from each SMILES. 3D models of these SMILES were created using both Open Babel (version 2.3.2) and a late-stage beta version of Gypsum-DL that had only minor differences from the published version. We used the following Open Babel flags: *-d* (delete hydrogens), *-h* (add hydrogens), and *-gen3D* (generate 3D coordinates). Gypsum-DL used default values consistent with the 1.0.0 defaults. Protein and ligands were converted from PDB to PDBQT using MGLTools (version 1.5.6)[215].

AutoDock Vina (version 1.1.2) (Vina) was used to dock each ligand into the corresponding protein for each protein-ligand complex. This was repeated for each 3D model generated by Open Babel and Gypsum-DL. To ensure docking poses were rigorously tested, Vina's *exhaustivity* parameter was set to 100. All other Vina parameters used the default settings.

Following docking, hydrogens were removed from all docked ligands using the Python library, Scoria[242]. Next, the Open Babel accessory program, *obrms*[164], was used to calculate the RMSD between a docked pose and its respective crystallographic pose. Twenty-six additional protein–ligand complexes were removed from the data set due to discrepancies between the atom connectivity of the crystallographic pose versus that of the docked models. It is worth noting that Open Babel produces only a single structural variant, whereas Gypsum-DL may produce multiple structural variants. We used the variant that had the lowest RMSD from the crystallographic pose.

Appendix B.4 Results and Discussion

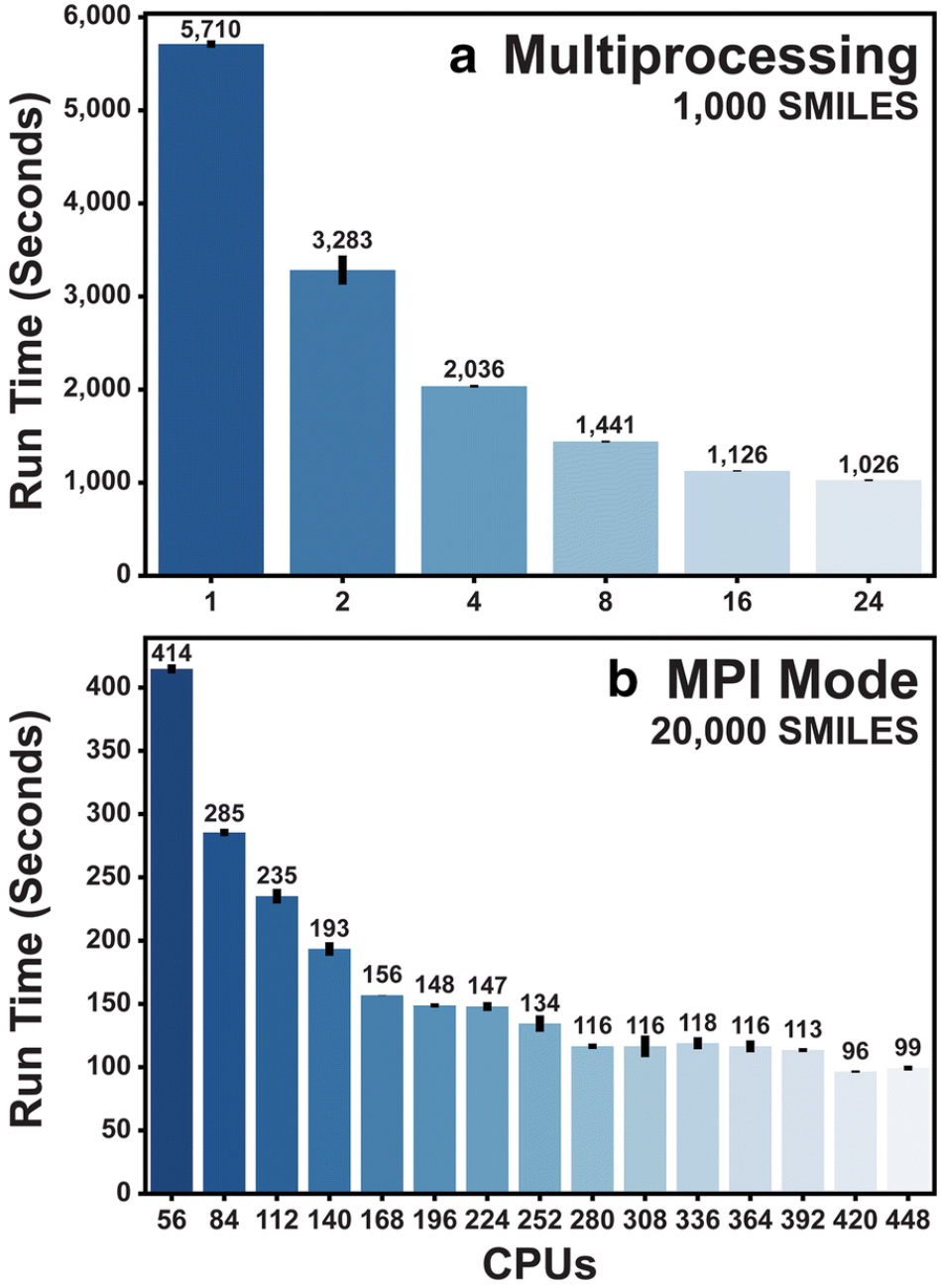
Appendix B.4.1 Benchmarks

Gypsum-DL's multiprocessing mode, which distributes jobs across processors on the same computer, uses a dynamic load-balancing approach that distributes tasks to processors as soon as a processor becomes available. The time required to process 1,000 compounds decreased as the number of processors increased (Appendix Figure 3A). In theory, this should have scaled linearly but it did not. We attribute this largely to the fixed time required to perform tasks that could not be parallelized, such as the processes of distributing/collecting data to/from processors for multiprocessing.

The mpi mode parallelizes tasks across multiple computer systems (e.g., nodes). This is particularly applicable in high-performance computing clusters. Mpi mode uses a static load-balancing approach, which distributes a set of jobs to each processor at the start of the parallelization. The various processors perform their respective calculations simultaneously and return the results when completed. Mpi mode allows calculations on a much larger scale but requires an MPI-enabled network of computers and an installation of the mpi4py Python package[288,296,297].

The static load-balancing of mpi mode reduces required communication between nodes but can result in idle processors that have completed all of their tasks while other processors are still working. This can occur under two circumstances: first, if one processor receives less time-consuming tasks than another processor; second, if the number of tasks is not evenly divisible by

the number of processors. For example, if the job manager has 402 jobs to distribute across 100 processors, two processors would receive five tasks while the other 98 processors receive four. If every task requires the same time, 98 processors would remain idle while the two finish processing their last jobs.



Appendix Figure 3. Benchmarks from multiprocessing and mpi mode parallelization.
A) The time required to process 1,000 SMILES using the multiprocessing mode on 24-core Skylake processor computers. B) The time required to process 20,000 SMILES using the mpi mode on 28-core Broadwell processors. All experiments were performed in triplicate. The bars represent standard deviation. This figure is reprinted with rights and permission under the Creative Commons Attribution 4.0 International License[214].

Appendix B.4.2 Impact on the Accuracy of Docking Pose Prediction

To assess Gypsum-DL's impact on docking pose predictions, we prepared ligands from 3,151 protein-ligand complexes using both Gypsum-DL and Open Babel (version 2.3.2)[164]. Then, after docking, we measured the RMSD of each pose compared to its corresponding crystallographic pose.

Of the 3,151 protein-ligand complexes, 71.4% of Gypsum-DL-processed ligands docked with an RMSD less than 3.0 Å from the corresponding crystallographic pose, versus only 53.0% of the Open Babel-processed ligands. The mean RMSD for Gypsum-DL-processed ligands was 2.37 Å (standard deviation: 2.03 Å), whereas Open Babel-processed poses had a mean RMSD of 3.40 Å (standard deviation: 2.51 Å). We performed a *t*-test to confirm the statistical significance of Gypsum-DL's superior performance in pose accuracy. To determine which *t*-test to use, we assessed the variance of the two sample sets. An *F*-test led us to conclude unequal variance in Gypsum-DL and Open Babel RMSD ($p = 0.00$), so we selected a two-tailed *t*-test that assumes unequal variances. Based on the *t*-test results, we rejected the hypothesis that Gypsum-DL- and Open Babel-prepared poses have an equal mean RMSD ($p = 0.00$). We concluded that by accounting for multiple forms of a given compound, Gypsum-DL improves pose-prediction accuracy.

Appendix B.4.3 Comparison with Similar Programs

A number of capable programs, each having advantages and disadvantages, can be used to assign 3D coordinates to small molecules and convert them between formats.

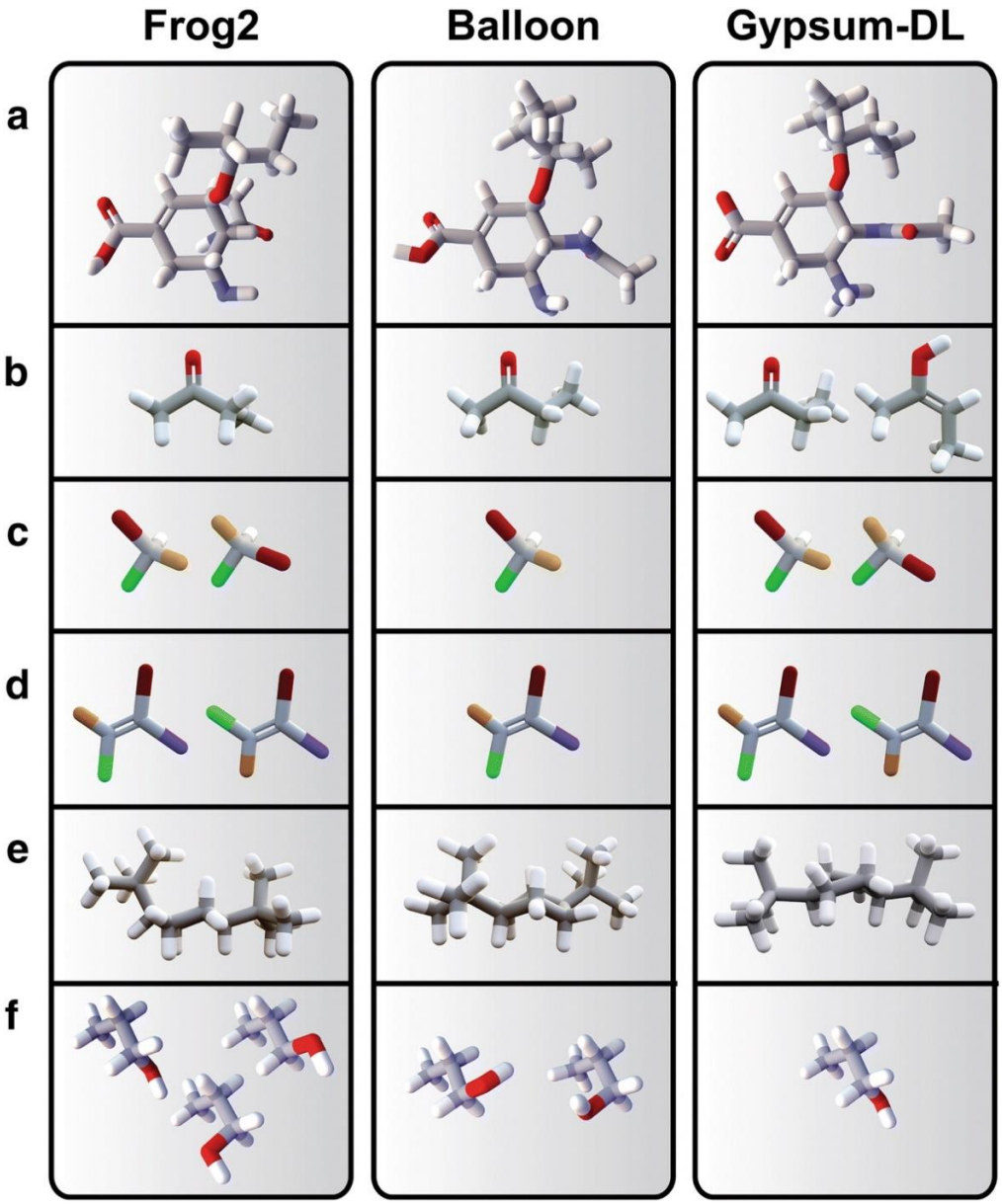
Excellent commercial programs, including OpenEye's OMEGA/QUACPAC[298,299] and Schrödinger's LigPrep[300] (Schrödinger, LLC) (Appendix Table 1), can be costly and often have licensing restrictions.Free programs, such as Frog2[301] and Balloon[302,303], are excellent alternatives (Appendix Table 1). Frog2 is an open-source program with an intuitive web interface. However, it handles ionization using Open Babel, which does not consider multiple ionization states, whereas Gypsum-DL uses the Dimorphite-DL[231] algorithm. Additionally, Frog2 is unable to fully sample non-aromatic ring conformations and tautomers (Appendix Figure 4 and Appendix Table 1). These limitations are illustrated by several test cases in Appendix Figure 4.

Balloon[302,303] is a free closed-source program. Like Frog2, it does not enumerate alternative ionization or tautomeric states (Appendix Figure 4 and Appendix Table 1).

Additionally, some cheminformatic libraries, such as RDKit[165] and Open Babel[164], can perform many of the same functions as Gypsum-DL but require that users design their own workflows. These target more advanced users by providing the core functions necessary to process and prepare small-molecules.

Appendix Table 1. Features of several prominent programs for assigning small-molecule 3D coordinates.

Program	Ionize	Tautomers	Chiral	Cis/Trans	Rings	Optimize	Free	Open Source	Web
Gypsum-DL[214]	✓	✓	✓	✓	✓	✓	✓	✓	
OpenEye[298,299]	✓	✓	✓	✓	✓	✓			
LigPrep[300]	✓	✓	✓	✓	✓	✓			
Frog2[301]			✓	✓		✓	✓	✓	✓
Balloon[302,303]			✓	✓	✓	✓	✓		



Appendix Figure 4. Comparison of output for Frog2, Balloon, and Gypsum-DL.
A) Ionization of oseltamivir carboxylate. B) Only Gypsum-DL generates both ketone and enol variants for butan-2-one. C) Frog2 and Gypsum-DL generate both (R) and (S) enantiomer variants of bromochlorofluroiodomethane, while Balloon does not enumerate both chiral enantiomers. D) Frog2 and Gypsum-DL generate both E and Z isomers of 1-bromo-2-chloro-2-fluoro-1-iodoethene, while Balloon does not. E) Non-aromatic ring conformations of cis-1,4-di-tert-butylcyclohexane. Gypsum-DL is able to generate the twist-boat conformation of cis-1,4-di-tert-butylcyclohexane, while Frog2 and Balloon only consider the chair conformation. F) Frog2 and Balloon create redundant models of propan-1-ol, while Gypsum-DL creates only a single model. This figure is reprinted with rights and permission under the Creative Commons Attribution 4.0 International License[214].

Appendix B.5 Conclusion

Accurate preparation of 3D small-molecule structures is crucial in CADD. Gypsum-DL provides a free and open-source platform for preparing 3D small-molecules that is computationally efficient, scalable, and accurate. Its intuitive command-line interface and parallelized codebase make Gypsum-DL easy to incorporate into any drug-discovery pipeline.

As with all programs, Gypsum-DL is not without its limitations. For example, large macrocycles can have many possible conformations. The ETKDG algorithm uses experimentally derived acyclic-bond torsion patterns to generate the initial 3D coordinates for macrocycles[291]. Gypsum-DL then geometrically optimizes these structures with the UFF. Despite being valid structures, Gypsum-DL-generated large-macrocycle conformations often are not the most energetically favorable. Future improvements to the ETKDG algorithm will hopefully improve generation of the initial macrocycle structures. Despite this limitation, Gypsum-DL is an accurate program for small-molecule preparation and 3D conformation generation.

Gypsum-DL is available for free download at http://durrantlab.com/gypsum-dl/ under the terms of the Apache License, Version 2.0.

Appendix B.6 Acknowledgments

I would like to thank my coauthors on the Gypsum-DL project. This project was supported by Dr. Jacob Durrant's computer-allocation grant from the University of Pittsburgh's Center for Research Computing (CRC).

Appendix B.7 Authors Contribution

All writing in this appendix is my original work and is based on the Gypsum-DL publication. Figures and tables are taken directly from the publication, but I have rewritten the legends. Dr. Jacob D. Durrant contributed editing and guidance to this appendix.

As stated in the Gypsum-DL publication: "PJR, JOS, GAM, and JDD designed the study. PJR, JOS, HG, and JDD contributed to the Gypsum-DL codebase. JOS, JLW, HG, GAM, KAM, JJR, and JDD contributed to the text, tables, and/or figures."[214]

Appendix C Supporting JSON-Formatted Parameters

This appendix provides the exact JSON-formatted parameters used for the *de novo* CADD runs performed in Chapters 2, 3, and 4. Several of these JSON-formatted parameters are taken directly from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. This is reprinted under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium"[6]:

> **Jacob O Spiegel**†, & Jacob D Durrant. (2020) AutoGrow4: An open-source genetic algorithm for *de novo* drug design and lead optimization. *Journal of Cheminformatics,* 12, 25. https://doi.org/10.1186/s13321-020-00429-4.
>
> † Jacob O. Spiegel should be regarded as first author.

I am the first author of the published manuscript, having written the entire AutoGrow4 codebase, performed many of the experiments discussed in the paper, and analyzed the data. I designed the experiments and the JSON-parameters presented in this appendix. All writing in this appendix is original content written by Jacob O. Spiegel.

```
{
 "filename_of_receptor": "/autogrow4/tutorial/PARP/4r6eA_PARP1_prepared.pdb",
 "center_x": -70.76,
 "center_y": 21.82,
 "center_z": 28.33,
 "size_x": 25.0,
 "size_y": 16.0,
 "size_z": 25.0,
 "additional_autoclickchem_parameters": "+all_reactions”
 "allow_modification_without_frag_addition": true,
 "directory_of_source_compounds": "/autogrow/autogrow/tutorial/starting_compounds/",
 "directory_of_fragments": "/autogrow/autogrow/fragments/MW_150/",
 "number_of_mutants_first_generation": 50,
 "number_of_crossovers_first_generation": 50,
 "number_of_mutants": 85,
 "number_of_crossovers": 85,
 "top_ones_to_advance_to_next_generation": 70,
 "num_generations": 6,
 "max_seconds_per_generation": 18000,
 "use_lipinski_filter": true,
 "use_strict_lipinski_filter": true,
 "use_ghose_filter": true,
 "scoring_function": VINA,
 "score_by_ligand_efficiency": false,
 "maintain_core": false,
 "minimum_core_atoms_required": 4,
 "vina_executable":
"/autogrow4/autogrow/docking/docking_executables/vina/autodock_vina_1_1_2_linux_x86/bin/v
ina",
 "num_processors": 12
}
```


Appendix JSON 1. AutoGrow3 settings used in the efficency benchmarks.

For ease of comparison, settings are presented in the JSON format even though AutoGrow3 (version 3.1.3) does not accept JSON-format parameters. All unspecified parameters were set to the AutoGrow3 defaults.

I compensated for several differences between AutoGrow3 and AutoGrow4 to make these benchmarks a fair comparison. In all generations except the first, AutoGrow3 treats compounds

that advance via elitism as new compounds. In contrast, AutoGrow4 does not count elite compounds against the total number of compounds created per generation. Therefore, I increased both the *number_of_mutants* and *number_of_crossovers* to ensure that AutoGrow3 would create exactly 100 new compounds in each generation, in addition to the 70 compounds that advanced via elitism. AutoGrow3 also produces an extra generation that summarizes the best compounds from all previous generations. This extra generation neither creates nor tests any new compounds, but is included simply to perform input/output operations. I thus set the *num_generations* parameter to six to yield five production generations.

Paths beginning with */autogrow/*, such as the directory of source compounds, refer to files contained in the AutoGrow3 download. Paths beginning with */autogrow4/*, such as the filename of the receptor, refer to files contained in the AutoGrow4 download. I do not include the parameters that specify the paths to Open Babel (*openbabel_bin_directory*), MGLTools (*mgltools_directory*), and the output directories (*output_dir*) because these paths depend on the specific computer used for testing.

This appendix JSON is taken from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. It is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium."[6]

```
{
 "filename_of_receptor": "/autogrow4/tutorial/PARP/4r6eA_PARP1_prepared.pdb",
 "center_x": -70.76,
 "center_y": 21.82,
 "center_z": 28.33,
 "size_x": 25.0,
 "size_y": 16.0,
 "size_z": 25.0,
 "source_compound_file": "/autogrow4/source_compounds/naphthalene_smiles.smi",
 "number_of_mutants_first_generation": 50,
 "number_of_crossovers_first_generation": 50,
 "number_of_mutants": 50,
 "number_of_crossovers": 50,
 "top_mols_to_seed_next_generation": 70,
 "number_elitism_advance_from_previous_gen": 70,
 "number_elitism_advance_from_previous_gen_first_generation": 0,
 "diversity_mols_to_seed_first_generation": 0,
 "diversity_seed_depreciation_per_gen": 0,
 "num_generations": 5,
 "number_of_processors": 12,
 "scoring_choice": "VINA",
 "LipinskiStrictFilter": true,
 "GhoseModifiedFilter": true,
 "filter_source_compounds": false,
 "start_a_new_run": true,
 "selector_choice": "Rank_Selector",
 "dock_choice": "VinaDocking", *
 "max_variants_per_compound": 1, *
 "generate_plot": false,
 "debug_mode": true,
 "reduce_files_sizes": false
}
```


Appendix JSON 2. AutoGrow4 settings used in the efficency benchmarks.

I set several AutoGrow4 parameters to match the run conditions of the AutoGrow3 efficiency benchmark. I converted the source compounds present in the AutoGrow3 *directory_of_source_compounds* directory (Appendix JSON 1) from PDB to SMILES (*naphthalene_smiles.smi*). AutoGrow3 does not separate elitism selection and seeding selection, so in AutoGrow4's runs both *top_mols_to_seed_next_generation* and

number_elitism_advance_from_previous_gen were set to 70. I also used the *GhoseModifiedFilter*, which is noted as the Ghose* filter in Table 1, to match the AutoGrow3 Ghose filter and applied *LipinskiStrictFilter*, which is noted in Table 1 as Lipinski*. I silenced several additional AutoGrow4 features (e.g., plotting the results, deleting temporary files, reducing file size by compression, and selecting by diversity) because these features are not available in AutoGrow3.

To test different AutoGrow4 configurations, I varied two parameters: *dock_choice* and *max_variants_per_compound* (noted with an asterisk). I set the *dock_choice* parameter, which specifies which docking software to use, to either *VinaDocking* or the default choice of *QuickVina2Docking*. I set the *max_variants_per_compound* parameter, which determines the maximum number of variants that the Gypsum-DL module should produce per input compound, to either 1, 3, or 5. I tested all permutations of *dock_choice* and *max_variants_per_compound*, resulting in six unique parameter configurations.

In this description of the settings I again excluded the path to the MGLTools directory (*mgltools_directory*) and the root output folder (*root_output_folder*). All unspecified parameters were set to the AutoGrow4 defaults.

This appendix JSON is taken from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. It is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium,"[6]

```
{
 "filename_of_receptor": "/autogrow4/tutorial/PARP/4r6eA_PARP1_prepared.pdb",
 "center_x": -70.76,
 "center_y": 21.82,
 "center_z": 28.33,
 "size_x": 25.0,
 "size_y": 16.0,
 "size_z": 25.0,
 "additional_autoclickchem_parameters": "+all_reactions”
 "allow_modification_without_frag_addition": true,
 "directory_of_source_compounds": "/Fragment_MW_100_to_150_reformat/",
 "directory_of_fragments": "/autogrow/autogrow/fragments/MW_150/",
 "number_of_mutants_first_generation": 2500,
 "number_of_crossovers_first_generation": 2500,
 "number_of_mutants": 2500,
 "number_of_crossovers": 2500,
 "top_ones_to_advance_to_next_generation": 1000,
 "num_generations": 50,
 "use_lipinski_filter": true,
 "use_strict_lipinski_filter": true,
 "use_ghose_filter": true,
 "scoring_function": VINA,
 "score_by_ligand_efficiency": false,
 "maintain_core": false,
 "minimum_core_atoms_required": 4,
 "vina_executable":
"/autogrow4/autogrow/docking/docking_executables/vina/autodock_vina_1_1_2_linux_x86/bin/v
ina",
 "num_processors": 28
}
```


Appendix JSON 3. AutoGrow3 settings used in the performance benchmarks.

For ease of comparison, settings are presented in JSON format even though AutoGrow3 does not accept JSON-formatted parameters. All unspecified parameters were set to the AutoGrow3 defaults.

Paths beginning with */autogrow/*, such as the directory of source compounds, refer to files contained in the AutoGrow3 download. Paths beginning with */autogrow4/*, such as the filename of the receptor, refer to files contained in the AutoGrow4 download. I do not include the parameters

that specify the paths to Open Babel (*openbabel_bin_directory*), MGLTools (*mgltools_directory*), and the output directories (*output_dir*) because these paths depend on the specific computer used for testing. The source compounds (*directory_of_source_compounds*) are the compounds provided in /*autogrow4/source_compounds/Fragment_MW_100_to_150.smi*, though I converted them to PDB using Gypsum-DL for use with AutoGrow3.

```
{
 "filename_of_receptor": "/autogrow4/tutorial/PARP/4r6eA_PARP1_prepared.pdb",
 "center_x": -70.76,
 "center_y": 21.82,
 "center_z": 28.33,
 "size_x": 25.0,
 "size_y": 16.0,
 "size_z": 25.0,
 "source_compound_file": "/autogrow4/source_compounds/Fragment_MW_100_to_150.smi ",
 "number_of_mutants_first_generation": 2500,
 "number_of_crossovers_first_generation": 2500,
 "number_of_mutants": 2500,
 "number_of_crossovers": 2500,
 "top_mols_to_seed_next_generation_first_generation": 500,
 "top_mols_to_seed_next_generation": 500,
 "number_elitism_advance_from_previous_gen_first_generation": 1000,
 "number_elitism_advance_from_previous_gen": 1000,
 "diversity_seed_depreciation_per_gen": 500,
 "diversity_mols_to_seed_first_generation": 0,
 "num_generations": 50,
 "number_of_processors": 280,
 "scoring_choice": "VINA",
 "LipinskiStrictFilter": true,
 "GhoseModifiedFilter": true,
 "PAINSFilter": true,
 "filter_source_compounds": false,
 "selector_choice": "Rank_Selector",
 "dock_choice": "QVina2Docking",
 "max_variants_per_compound": 5,
 "generate_plot": false,
 "reduce_files_sizes": false
 "use_docked_source_compounds": true,
 "rxn_library": "all_rxns",
 "multithread_mode": "mpi"
}
```


Appendix JSON 4. AutoGrow4 settings used in the performance benchmarks.

I set several AutoGrow4 parameters to match the run conditions of the AutoGrow3 performance benchmark, and silenced several additional AutoGrow4 features (e.g., plotting the

results and reducing file size by compression) because these features are not available in AutoGrow3.

In this description of the settings I excluded the path to the MGLTools directory (*mgltools_directory*) and the root output folder (*root_output_folder*). All unspecified parameters were set to the AutoGrow4 defaults.

```
{
 "filename_of_receptor": "/autogrow4/tutorial/PARP/4r6eA_PARP1_prepared.pdb",
 "center_x": -70.76,
 "center_y": 21.82,
 "center_z": 28.33,
 "size_x": 25.0,
 "size_y": 16.0,
 "size_z": 25.0,
 "source_compound_file": "/autogrow4/source_compounds/Fragment_MW_100_to_150.smi",
 "number_of_mutants_first_generation": 500,
 "number_of_crossovers_first_generation": 500,
 "number_elitism_advance_from_previous_gen_first_generation": 40,
 "number_of_mutants": 2500,
 "number_of_crossovers": 2500,
 "number_elitism_advance_from_previous_gen": 500,
 "top_mols_to_seed_next_generation_first_generation": 50,
 "top_mols_to_seed_next_generation": 500,
 "diversity_mols_to_seed_first_generation": 500,
 "diversity_seed_depreciation_per_gen": 5,
 "num_generations": 30,
 "number_of_processors": 280,
 "dock_choice": "QuickVina2Docking",
 "scoring_choice": "VINA",
 "selector_choice": "Rank_Selector",
 "LipinskiStrictFilter": true,
 "GhoseFilter": true,
 "PAINSFilter": true,
 "reduce_files_sizes": true,
 "max_variants_per_compound": 5,
 "filter_source_compounds": false,
 "use_docked_source_compounds": true,
 "rxn_library": "all_rxns",
 "multithread_mode": "mpi"
}
```


Appendix JSON 5. AutoGrow4 settings used in the large-scale *de novo* run.

The source compounds were taken from AutoGrow4's *Fragment_MW_100_to_150.smi* library, which consists of ZINC15 molecules that have molecular weights ranging from 100 Da to 150 Da. I applied the Ghose, Lipinski*, and PAINS filters. In this description of the settings I again excluded the path to the MGLTools directory (*mgltools_directory*) and the root output folder

(*root_output_folder*). All unspecified parameters were set to the AutoGrow4 defaults. This appendix JSON is from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. It is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium."[6]

```
{
 "filename_of_receptor": "/autogrow4/tutorial/PARP/4r6eA_PARP1_prepared.pdb",
 "center_x": -70.76,
 "center_y": 21.82,
 "center_z": 28.33,
 "size_x": 25.0,
 "size_y": 16.0,
 "size_z": 25.0,
 "source_compound_file": "/autogrow4/source_compounds/PARPI_BRICS_frags.smi",
 "number_of_mutants_first_generation": 500,
 "number_of_crossovers_first_generation": 500,
 "number_elitism_advance_from_previous_gen_first_generation": 40,
 "number_of_mutants": 2500,
 "number_of_crossovers": 2500,
 "number_elitism_advance_from_previous_gen": 250,
 "top_mols_to_seed_next_generation_first_generation": 50,
 "top_mols_to_seed_next_generation": 500,
 "diversity_mols_to_seed_first_generation": 500,
 "diversity_seed_depreciation_per_gen": 25,
 "num_generations": 5,
 "number_of_processors": 280,
 "dock_choice": "QuickVina2Docking",
 "scoring_choice": "VINA",
 "selector_choice": "Rank_Selector",
 "LipinskiStrictFilter": true,
 "GhoseFilter": true,
 "PAINSFilter": true,
 "reduce_files_sizes": true,
 "docking_exhaustiveness": 25,
 "max_variants_per_compound": 5,
 "gypsum_timeout_limit": 60,
 "docking_timeout_limit": 600,
 "filter_source_compounds": false,
 "use_docked_source_compounds": true,
 "rxn_library": "all_rxns",
 "multithread_mode": "mpi",
 "start_a_new_run": true
}
```


Appendix JSON 6. AutoGrow4 settings used in the PARPi lead-optimization runs.

The source compounds, taken from *PARPI_BRICS_frags.smi*, consist of 11 PARPi molecules, as well as 83 PARPi-derived fragments that I generated using BRICS decomposition.

In this description of the settings I excluded the path to the MGLTools directory (*mgltools_directory*) and the root output folder (*root_output_folder*). All unspecified parameters were set to the AutoGrow4 defaults.

This appendix JSON is from the AutoGrow4 manuscript, which has been published in the Journal of Cheminformatics. It is reprinted with rights and permissions under the Creative Commons Attribution 4.0 International License, which "allows unrestricted use, distribution, and reproduction in any medium."[6]

```
{
 "filename_of_receptor": "4DQY_ChainA.pdb",
 "center_x": -25.32,
 "center_y": 58.93,
 "center_z": 16.89,
 "size_x": 30.0,
 "size_y": 20.0,
 "size_z": 25.0,
 "source_compound_file": "/autogrow4/source_compounds/Fragment_MW_100_to_150.smi",
 "number_of_mutants_first_generation": 500,
 "number_of_crossovers_first_generation": 500,
 "number_elitism_advance_from_previous_gen_first_generation": 40,
 "number_of_mutants": 2500,
 "number_of_crossovers": 2500,
 "number_elitism_advance_from_previous_gen": 500,
 "top_mols_to_seed_next_generation_first_generation": 50,
 "top_mols_to_seed_next_generation": 500,
 "diversity_mols_to_seed_first_generation": 500,
 "diversity_seed_depreciation_per_gen": 0,
 "num_generations": 10,
 "number_of_processors": 280,
 "dock_choice": "QuickVina2Docking",
 "scoring_choice": "VINA",
 "selector_choice": "Rank_Selector",
 "LipinskiStrictFilter": true,
 "GhoseFilter": true,
 "PAINSFilter": true,
 "reduce_files_sizes": true,
 "max_variants_per_compound": 5,
 "filter_source_compounds": false,
 "use_docked_source_compounds": true,
 "rxn_library": "all_rxns",
 "multithread_mode": "mpi"
}
```


Appendix JSON 7. AutoGrow4 settings used in the DBD-targeted runs.

The source compounds were taken from AutoGrow4's *Fragment_MW_100_to_150.smi* library, which consists of ZINC15 molecules that have molecular weights ranging from 100 Da to 150 Da. I applied the Ghose, Lipinski*, and PAINS filters. In this description of the settings I

excluded the path to the MGLTools directory (*mgltools_directory*) and the root output folder (*root_output_folder*). All unspecified parameters were set to the AutoGrow4 defaults.

These runs were designed to closely match the AutoGrow4 large-scale run (Appendix JSON 5); however, the *diversity_seed_depreciation_per_gen* and *num_generations* variables differ. The *diversity_seed_depreciation_per_gen* was set to zero to maintain diversity selection throughout the run, and the *num_generations* was lowered to ten generations. Additionally, the pocket parameters and receptor file were adjusted to target the appropriate DBD pocket.

```
{
 "conformer_search_type": "denovo",
 "vdw_defn_file": "/dock6/parameters/vdw_AMBER_parm99.defn",
 "flex_defn_file": "/dock6/parameters/flex.defn",
 "flex_drive_file": "/dock6/parameters/flex_drive.tbl",
 "dn_user_specified_anchor": "no",
 "dn_unique_anchors": 50,
 "use_database_filter": "yes",
 "dn_constraint_mol_wt": 480.0,
 "dbfilter_max_molwt": 480,
 "dbfilter_min_molwt": 160,
 "dbfilter_max_rot_bonds": 999,
 "dbfilter_min_rot_bonds": 0,
 "dbfilter_max_heavy_atoms": 70,
 "dbfilter_min_heavy_atoms": 20,
 "dbfilter_max_formal_charge": 10.0,
 "dbfilter_min_formal_charge": -10.0,
 "simplex_random_seed": *
}
```


Appendix JSON 8. *De novo* DOCK settings for the *de novo* CADD comparison.

For ease of comparison, settings are presented in JSON format, even though *de novo* DOCK (version DOCK6.9) requires parameters to be provided as a .in file. All unspecified parameters were set to the *de novo* DOCK defaults. *De novo* DOCK parameters are described in the DOCK6 manual[187].

To ensure that runs were not repeated, a unique random seed (*simplex_random_seed*) was provided to each *de novo* DOCK run (noted with an asterisk). The physiochemical property constraints (e.g., molecular weight, number of rotatable bonds, and number of heavy atoms) were set to best match AutoGrow4's Lipinski* and Ghose filters. Because neither filter tests for formal charge, I set the formal charge constraints to be as unrestrictive as allowed.

Paths beginning with */dock6/*, such as the file that defines the Van der Waals radius (*vdw_defn_file*), refer to files contained in the DOCK6 download. The following parameters were

excluded from this description of the settings because they specify files/directories that depend on the specific computer used for testing: *dn_fraglib_scaffold_file*, *dn_fraglib_linker_file*, *dn_fraglib_sidechain_file*, *dn_torenv_table*, *grid_score_grid_prefix*, *receptor_site_file,* and *dn_output_prefix*. The *dn_fraglib_scaffold_file*, *dn_fraglib_linker_file*, *dn_fraglib_sidechain_file*, and *dn_torenv_table* parameters specify custom libraries and tables that are based on the PARPi and PARPi-fragment seed compounds, in this case created using the DOCK6 fragment-library generator. The *grid_score_grid_prefix* and *receptor_site_file* are the grid and sphere files, respectively, that were used for docking into the 4R6E:A structure. Lastly, the *dn_output_prefix* specifies the path where *de novo* DOCK writes its output files.

```
{
 “NUMTHREADS”: 280,
 “NUMISLANDS”: 280,
 “NUMINDIVIDUALS”: 1000,
 “LIGANDSIZE”: 3,
 “EXCHANGESTEP”: 50,
 “MAXCONV”: 0.25,
 “CONSTRAINT_MW”: “160 480”,
 “CONSTRAINT_HBDONOR”: “0 5”,
 “CONSTRAINT_HBACCEPTOR”: “0 10”,
 “FSIMCUT”: 0.9,
 “LINKER”: “/GANDI/test_case/linkers/*”,
 “PARAMETERFILE”: “/GANDI/test_case/gandi.prm”,
 “RECEPTOR”: [689, 763, 766, 767, 769, 770, 861, 862, 863, 864, 865, 868, 872, 877, 878, 879,
880, 881, 888, 889, 890, 894, 895, 896, 897, 898, 903, 904, 907, 909, 988],
 “RANDSEED”: *,
}
```


Appendix JSON 9. GANDI settings for the *de novo* CADD comparison.

For ease of comparison, settings are presented in the JSON format, even though GANDI (version 2.0) requires parameters to be provided as a .ini file. All unspecified parameters were set to GANDI defaults. GANDI parameters are described in the GANDI user manual[167].

GANDI requires a list of all linker files (*LINKER*); for brevity, I provide just the directory containing these files (noted with an asterisk). To ensure that runs were not repeated, a unique random seed (*RANDSEED*) was provided to each GANDI run (noted with an asterisk). The physiochemical-property constraints (e.g., the molecular weight, the number of hydrogen bond donors, and the number of hydrogen bond acceptors) were set to best match AutoGrow4’s Lipinski* and Ghose filters.

To define the coordinates of the binding site, GANDI requires a list of residue numbers be provided with the path to the receptor file in the *RECEPTOR* variable. The receptor file path (a MOL2 file of the 4R6E:A structure) was omitted from my description of the settings, as it is

specific to the computer used for testing. The residue numbers defined by *RECEPTOR* were the list of all residues identified using the BINANA algorithm[246] of the top 500 compounds produced during the large-scale *de novo* run and of the top 500 compounds produced during the PARPi lead-optimization AutoGrow4 runs. All residues within 2.5Å and/or that formed any interaction(s) with a top-scoring AutoGrow4-generated compound were included.

Paths beginning with */GANDI/* refer to files contained in the GANDI download. The *RECEPTOR*, *DOCKEDFRAGMENTS,* and *OUTPUTDIR* parameters were excluded from my description because they specify files/directories that depend on the specific computer used for testing. *DOCKEDFRAGMENTS* specifies the file paths of the PARPi and PARPi-fragment seed compounds in MOL2 format, which I previously docked into the catalytic site using SEED (version 4.0.0)[217]. Lastly, *OUTPUTDIR* specifies the path where GANDI writes its output files.

```
{
 "seed_mol": "/autogrow4/source_compounds/PARPI_BRICS_frags.smi",
 "n_attemps": 100,
 "lam_fact": *,
 "probab": true/false,
 "use_filter_unique_valid": true
}
```

Appendix JSON 10. LigDream settings for the *de novo* CADD comparison.

For ease of comparison, settings are presented in JSON format, even though LigDream[213] (version 1.0) does not accept JSON-formatted files. All LigDream runs used the trained models provided in the LigDream download and were seeded with the PARPi and PARPi fragments provided in the AutoGrow4 download. Additionally, all runs were set to 100generations (*n_attempts*) and to remove any invalid and/or duplicate SMILES (*use_filter_unique_valid*). All unspecified parameters were set to LigDream defaults.

The two variables that were changed to test different LigDream configurations, *lam_fact* and *probab*, are noted with an asterisk (*). All combinations of these settings were tested in triplicate. The λ-factor (*lam_fact*) controls how similar the generated compounds are to the source compounds, where a higher λ results in greater deviance in the generated molecules from the source compounds[213]. Based on settings used in the LigDream publication[213], I set λ to either 0, 1, 5, 10, or 15. The *probab* variable determines whether probabilistic recurrent neural network decoding is used. I set it to either *true* or *false*[213].

Paths beginning with */autogrow4/* refer to files contained in the AutoGrow4 download.

Appendix D AutoGrow4 Manual and Tutorial

This appendix is an adaptation of the user manual and tutorial distributed with the AutoGrow4 download. AutoGrow4 is available free of charge under the terms of the open-source Apache License, version 2.0. This manual is adapted and reprinted with rights and permissions from the authors of AutoGrow4.

AutoGrow4 was published under the Creative Commons Attribution 4.0 International License which "allows unrestricted use, distribution, and reproduction in any medium"[6]. The work in this appendix is adapted and reprinted with rights and permission:

> **Jacob O Spiegel**†, & Jacob D Durrant. (2020) AutoGrow4: An open-source genetic algorithm for *de novo* drug design and lead optimization. *Journal of Cheminformatics,* 12, 25. https://doi.org/10.1186/s13321-020-00429-4.
>
> † Jacob O. Spiegel should be regarded as first author.

I am the first author of the published manuscript, having written the entire AutoGrow4 codebase, performed many of the experiments discussed in the paper, and analyzed the data. I designed the layout for all figures in the paper with Dr. Jacob Durrant. Dr. Durrant refined and generated many of the high-quality images for publication. Dr. Durrant also provided guidance and insight while designing this manual/tutorial. All writing in this appendix is original content written by Jacob O. Spiegel.

Appendix D.1 Welcome to AutoGrow4

This document will break down how to run AutoGrow4. It will also cover what dependencies are required.

Please note that file paths are relative to a given system. There are several paths used in this tutorial that need to be replaced with the proper paths of one's own system. These paths will be indicated by a string of ALL_CAPS. For example (from "Appendix D.3: Installing AutoGrow4"):

$ cd /PATH_TO/DESIRED_DIR/

You must replace /PATH_TO/DESIRED_DIR/ with the path to the Autogrow4 directory on your own system. On an Ubuntu distribution, this may look like:

$ cd /home/jacob/Documents/

For brevity, the main Autogrow4 directory is simplified as /autogrow4/ throughout this tutorial. You may need to supply the /PATH_TO/DESIRED_DIR/ described above before /autogrow4/.

Appendix D.2 Computer Requirements

AutoGrow4 has been tested on Ubuntu 16.04 and higher, as well as on MacOS 10.13 High Sierra. It has been verified to work on an HPC cluster using SMP multithreading (RedHat Enterprise Server release 7.3 Maipo).

AutoGrow4 has not been configured for Windows OS, but a script capable of running AutoGrow4 within a Docker container on Windows can be found in:

/autogrow4/docker/autogrow_in_docker.py

This script should run on any Docker-enabled machine, and should be capable of multithreading. Details on running AutoGrow4 within a Docker container can be found below, in the "Appendix D.7: Docker Submission."

Appendix D.3 Installing AutoGrow4

A copy of AutoGrow4 can be downloaded from the Durrant lab website at: http://git.durrantlab.com/jdurrant/autogrow4.

You can also install AutoGrow4 using the git clone command:

```
$ cd /PATH_TO/DESIRED_DIR/
$ git clone https://git.durrantlab.pitt.edu/jdurrant/autogrow4
```

Appendix D.4 Dependencies

AutoGrow4 has several dependencies that may need to be installed separately.

Appendix D.4.1 Bash (Required))

A modern installation of Bash is required to run AutoGrow4. AutoGrow4 has been tested using GNU Bash, version 4.4.19. MacOS and Linux come with Bash preinstalled.

Appendix D.4.2 Coreutils (Required for macOS)

Most Linux OS come preinstalled with modern Bash and timeout tool (part of the coreutils package) that AutoGrow4 requires. Use on macOS requires the additional installation of the coreutils package, available through homebrew, which provides the equivalent gtimeout binary.

This can be done using the package manager homebrew by running:

$ sudo brew install coreutils

Appendix D.4.3 Python Installation (Required)

AutoGrow4 is primarily written in Python. A modern version of Python can be installed using conda:

•https://docs.conda.io/projects/conda/en/latest/user-guide/install/ or

•http://www.python.org/getit/.

AutoGrow4 has been tested with Python 2.7, 3.6, and 3.7. Future support and updates will focus on 3.7. We recommend using the most current version of Python available, 3.7 or newer.

Appendix D.4.4 MGLTools

MGLTools[215] is written by the creators of Autodock Vina. It is used by AutoGrow4 to convert .pdb files to the .pdbqt format. The .pdbqt format is required by Vina-type docking programs, including Autodock Vina[185] and QuickVina2[205].

Morris, G. M., Huey, R., Lindstrom, W., Sanner, M. F., Belew, R. K., Goodsell, D. S. and Olson, A. J. (2009) Autodock4 and AutoDockTools4: automated docking with selective receptor flexibility. *J. Computational Chemistry* 2009, 16: 2785-91

If you prefer to not use MGLTools for file conversion, you may also use Open Babel (obabel) or custom file-converting/docking software.

Installation:

WARNING: We recommend that you DO NOT pip or conda install MGLTools, as those package managers provide an outdated Python package and create issues with environments.

The best way to install MGLTools is to download the latest release of the command-line version (NOT THE GUI VERSION) from http://mgltools.scripps.edu/downloads.

Once the command-line version of the MGLTools package has been downloaded, follow this example installation (Linux, MGLTools 1.5.6):

1. To extract files, unzip/untar the package:

   $ tar -xvf /PATH_TO/mgltools_x86_64Linux2_1.5.6.tar.gz

2. Go to the extract folder:

   $ cd /PATH_TO/mgltools_x86_64Linux2_1.5.6

3. Run the installation script and make sure MGLToolsPckgs is unpacked:

– If /PATH_TO/mgltools_x86_64Linux2_1.5.6/MGLToolsPckgs/ is a folder:

$ bash install.sh

– If /PATH_TO/mgltools_x86_64Linux2_1.5.6/MGLToolsPckgs/ is not a folder, you must manually unzip/untar MGLToolsPckgs.tar.gz:

$ tar -xvf /PATH_TO/mgltools_x86_64Linux2_1.5.6/MGLToolsPckgs.tar.gz

4. Click 'OK' to the licensing agreement, which should open automatically. Please note MGLTools is free for academic use but may require a license for commercial usage.
5. Find the path for the AutoGrow4 variable (see next section, Additional Pathing Instructions).

Additional Pathing Instructions:

To use MGLTools to convert files, AutoGrow4 is required to know the path to the MGLTools directory. The path can be found by:

1. Going to the extract folder:

$ cd /PATH_TO/mgltools_x86_64Linux2_1.5.6

2. Using the pwd command in Bash to get the absolute path to the MGLTools directory:

$ pwd

To run AutoGrow4, provide the path of this MGLTools directory using the variable `--mgltools_directory`:

$ python RunAutogrow.py ... \

--mgltools_directory /PATH_TO/mgltools_x86_64Linux2_1.5.6 ...

On Linux and macOS machines, AutoGrow4 will automatically locate three important file paths based on mgltools_directory:

1. prepare_ligand4.py: mgltools_directory + /MGLToolsPckgs/AutoDockTools/Utilities24/prepare_ligand4.py
2. prepare_receptor4.py: mgltools_directory + /MGLToolsPckgs/AutoDockTools/Utilities24/prepare_receptor4.py
3. mgl_python: mgltools_directory + /bin/pythonsh

If you're running on Windows OS (limited support), please provide those paths to AutoGrow4 manually:

```
$ python RunAutogrow.py ... \
    --mgltools_directory /PATH_TO/mgltools_win32_1.5.6\ \
    --prepare_ligand4.py /PATH_TO/mgltools_win32_1.5.6\ \
    --prepare_receptor4.py /PATH_TO/mgltools_win32_1.5.6\ \
    --mgl_python /PATH_TO/mgltools_win32_1.5.6\ + /PATH_TO/pythonsh ...
```

Providing custom paths for the `--prepare_ligand4.py`, `--prepare_receptor4.py`, and `--mgl_python` parameters will override the paths that AutoGrow4 determines using the `--mgltools_directory` parameter alone.

Appendix D.4.5 Open Babel

Open Babel[164] is optional, but a conversion method is required. You must use either MGLTools, obabel, or a custom file converter and docking software.

Open Babel citations:

N M O'Boyle, M Banck, C A James, C Morley, T Vandermeersch, and G R Hutchison. "Open Babel: An open chemical toolbox." *J. Cheminf*. (2011), 3, 33. DOI:10.1186/1758-2946-3-33

The Open Babel Package, version 2.3.1 http://openbabel.org (accessed Oct 2011)

obabel is a command-line tool for cheminformatic file conversion. It is used by AutoGrow4 to convert .pdb files to .pdbqt format, as required by Vina-type docking programs including Autodock Vina[185] and QuickVina2[205]:. An alternative conversion option is MGLTools.

Installation:

An easy installation on Linux/macOS machines is:

$ sudo apt-get install openbabel

$ sudo apt-get update

Full instructions for obabel installation can be found on their site: https://openbabel.org/docs/dev/Installation/install.html

AutoGrow4 has been tested with obabel version 2.4.1.

Additional Pathing Instructions:

To use obabel to convert files, AutoGrow4 requires the path to the obabel executable. Once Open Babel is installed, the path to the obabel executable can be found by running:

$ which obabel

This path should be provided to AutoGrow4 using the `--obabel_path` variable:

```
$ python RunAutogrow.py ...\
  --obabel_path /PATH_TO/obabel ...
```

Appendix D.4.6 Python APIs (Required)

AutoGrow4 also uses several Python API libraries beyond those found in the standard library. These must be installed separately. Most can be installed via conda or pip.

Mandatory Installations:

RDKit[165]: Cheminformatic library. RDKit can be downloaded via conda/pip. To install using conda use the command:

$ conda install -c rdkit rdkit

We use the following RDKit sub-libraries in AutoGrow4:

```
>>> import rdkit
>>> from rdkit import RDLogger, Chem, DataStructs
>>> from rdkit.Chem import MolSurf, Crippen, rdFMCS, Descriptors
>>> from rdkit.Chem import AllChem, FilterCatalog, Lipinski, rdDepictor
>>> from rdkit.Chem.Draw import PrepareMolForDrawing, rdMolDraw2D
>>> from rdkit.Chem.rdMolDescriptors import GetMorganFingerprint
>>> from rdkit.Chem.FilterCatalog import FilterCatalogParams
>>> from rdkit.Chem.rdchem import BondStereo
```

NumPy[285] (mathematical functions) can be downloaded via conda/pip. AutoGrow4 has been tested using NumPy version 1.15.0. It can be conda installed using the command

$ conda install -c anaconda numpy

SciPy[286] (mathematical functions) can be downloaded via conda/pip. AutoGrow4 has been tested using SciPy version 1.1.0. It can be conda installed using the command

$ conda install -c anaconda scipy

Matplotlib[287] (Python graphing tool) can be downloaded via conda/pip. AutoGrow4 has been tested using Matplotlib version 3.0.2. It can be conda installed using the command

$ conda install matplotlib

func_timeout (Pythonic timeout tool) can be downloaded via pip. AutoGrow4 has been tested using func_timeout version 4.3.5. It can be pip installed using the command

$ pip install func-timeout

Optional Installations:

mpi4py[288] (MPI multithreading Python library) is required for multithreading using MPI. It can be downloaded via conda/pip. AutoGrow4 has been tested using mpi4py version 3.0.1. It can be conda installed using the command

$ conda install -c anaconda mpi4py

This may require a preinstallation of mpich:

$ sudo apt install mpich

AutoGrow4 requires mpi4py version 2.1.0 and higher. To check the version:

1. open a python window.
2. enter into the window:

```
>>> import mpi4py
>>> mpi4py.__version__
  3.0.1
```

MPI use also requires an MPI-enabled computer environment. OpenMPI was used by the authors.

OpenMPI installation instructions can be found: http://lsi.ugr.es/jmantas/pdp/ayuda/datos/instalaciones/Install_OpenMPI_en.pdf

Quick OpenMPI installation can be done by running in a Bash terminal:

$ sudo apt-get install openmpi-bin openmpi-common openssh-client openssh-server libopenmpi1.3 libopenmpi-dbg libopenmpi-dev

Establishing a fully MPI-enabled computer network is complicated and should only be attempted by qualified technicians. The authors used an Intel's Omni-Path communication architecture that was established by experts at the University of Pittsburgh's Center for Research Computing. The authors DO NOT RECOMMEND ATTEMPTING THIS ON YOUR OWN.

Appendix D.4.7 Pre-Installed Python and Binary Dependencies

AutoGrow4 comes with several dependencies preinstalled, requiring no additional effort by the user. These have licenses that allow them to be freely redistributed. If a dependency updates, please feel free to contact us, and we will do our best to make our code future-compatible.

Docking Programs:

AutoGrow4 comes preinstalled with two docking programs, Autodock Vina[185] and QuickVina2.1[205]:

- Autodock Vina 1.1.2 (packaged with executables for Linux, macOS, and Windows)
  - Version: 1.1.2
  - Location: /autogrow4/autogrow/docking/docking_executables/vina/
  - Citation:

    Trott, O., & Olson, A. J. (2010). AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of computational chemistry*, 31(2), 455–461. doi:10.1002/jcc.21334

  - License: Apache version 2

- QuickVina2.1 (compatible with Linux OS and macOS)
  - Version: 2.1
  - Location: /autogrow4/autogrow/docking/docking_executables/q_vina_2/
  - Citation:

    Alhossary, A., Handoko, S. D., Mu, Y. & Kwoh, C. K. Fast, accurate, and reliable molecular docking with QuickVina 2. Bioinformatics 31, 2214–2216 (2015). DOI:10.1093/bioinformatics/btv082
  - License: Apache version 2

Both of these software can be found within the directory: /autogrow4/autogrow/docking/docking_executables/

AutoGrow4 allows users to provide custom docking software. This could be as simple as using a different version of Autodock Vina:

```
$ python RunAutogrow.py ... \
  --docking_executable /PATH_TO/Autodock_Vina_version_X_executable
```

More advanced use allows users to provide a custom docking program. Details regarding how to use custom docking software are provided below in the "Appendix D.8: Providing Custom Plugins."

Scoring/Rescoring Programs:

NNScore 1[233] and NNScore 2[234] are free and open-source programs that are distributed with AutoGrow4. Both NNScore1 and NNScore2 reassess ligand docking. They were trained

using Autodock Vina 1.1.2 so to use these programs we require the docking be performed using Autodock Vina 1.1.2.

AutoGrow4 allows users to provide custom scoring/rescoring software. Details for custom scoring/rescoring suites are provided below in the "Appendix D.8: Providing Custom Plugins."

- NNScore 1:
  - Version: 1.1
  - Location: /autogrow4/autogrow/docking/scoring/nn_score_exe/nnscore1/
  - Citation:

    NScore: A Neural-Network-Based Scoring Function for the Characterization of Protein-Ligand Complexes. Jacob D. Durrant, J. Andrew McCammon. *Journal of Chemical Information and Modeling*, 2010, 50 (10), pp865-1871.
  - License: GNU General Public version 3
- NNScore 2:
  - Version: 2.02
  - Location: /autogrow4/autogrow/docking/scoring/nn_score_exe/nnscore2/
  - Citation:

    NNScore 2.0: A Neural-Network Receptor–Ligand Scoring Function. Jacob D. Durrant, Andrew McCammon. *Journal of Chemical Information and Modeling*, 2011, 51 (11), pp 2897-2903.
  - License: GNU General Public version 3

SMILES Conversion to 3D and Protonation Adjustments:

AutoGrow4 performs most of its ligand handling using 2D SMILES. AutoGrow4 uses the free and open-source program Gypsum-DL[214] to convert small molecules from SMILES to 3D SDF format. Gypsum-DL is prepackaged in AutoGrow4. Gypsum-DL itself also includes the MolVS and Dimorphite-DL[231] packages.

- Gypsum-DL:
  - Version: 1.1.2
  - Location: /autogrow4/autogrow/operators/convert_files/gypsum_dl/
  - Citation:

    Ropp PJ, Spiegel JO, Walker JL, Green H, Morales GA, Milliken KA, Ringe JJ, Durrant JD. Gypsum-DL: An Open-Source Program for Preparing Small-Molecule Libraries for Structure-Based Virtual Screening. *J Cheminform*. 11(1):34, 2019. [PMID: 31127411] [doi: 10.1186/s13321-019-0358-3]
  - License: Apache version 2.0
- Dimorphite-DL:
  - Version: 1.2.2
  - Location:

    /autogrow4/autogrow/operators/convert_files/gypsum_dl/gypsum_dl/Steps/SMILES/dimorphite_dl
  - Citation:

Ropp PJ, Kaminsky JC, Yablonski S, Durrant JD (2019) Dimorphite-DL: An open-source program for enumerating the ionization states of drug-like small molecules. *J Cheminform* 11:14. doi:10.1186/s13321-019-0336-9.

  - License: Apache version 2.0

- MolVS:
  - Version: v0.1.1 2019 release
  - Location:

    /autogrow4/autogrow/operators/convert_files/gypsum_dl/gypsum_dl/molvs
  - Citation:

    https://molvs.readthedocs.io; Downloaded from https://github.com/mcs07/MolVS
  - License: MIT License

Appendix D.5 Running AutoGrow4

To run AutoGrow4, use the Python script RunAutogrow.py, located in the top AutoGrow4 directory, from the command line. AutoGrow4 accepts user input via two methods:

1. Command-line submission: executing directly from the command line.

```
$ cd /PATH_TO/autogrow4/
$ python RunAutogrow.py \
  --filename_of_receptor
/autogrow4/autogrow/tutorial/PARP/4r6eA_PARP1_prepared.pdb \
```

```
    --center_x -70.76 --center_y  21.82 --center_z 28.33 \
    --size_x 25.0 --size_y 16.0 --size_z 25.0 \
    --source_compound_file
/autogrow4/autogrow/source_compounds/naphthalene_smiles.smi \
    --root_output_folder /PATH_TO/output_directory/ \
    --number_of_mutants_first_generation 50 \
    --number_of_crossovers_first_generation 50 \
    --number_of_mutants 50 \
    --number_of_crossovers 50 \
    --top_mols_to_seed_next_generation 50 \
    --number_elitism_advance_from_previous_gen 50 \
    --number_elitism_advance_from_previous_gen_first_generation 10 \
    --diversity_mols_to_seed_first_generation 10 \
    --diversity_seed_depreciation_per_gen 10 \
    --num_generations 5 \
    --mgltools_directory /PATH_TO/mgltools_x86_64Linux2_1.5.6/ \
    --number_of_processors -1 \
    --scoring_choice VINA \
    --LipinskiLenientFilter \
    --start_a_new_run \
    --rxn_library ClickChem \
    --selector_choice Rank_Selector \
```

```
--dock_choice VinaDocking \
--max_variants_per_compound 5 \
--redock_elite_from_previous_gen False \
--generate_plot True \
--reduce_files_sizes True \
--use_docked_source_compounds True \
>  /PATH_TO/OUTPUT/text_file.txt \
2>  /PATH_TO/OUTPUT/text_errormessage_file.txt
```

2. JSON file submission: store AutoGrow4 parameters in a .json file

```
$ cd /PATH_TO/autogrow4/
$ python RunAutogrow.py -j /PATH_TO/json_file_with_variable.json
```

Examples of the .json files can be found in the folder /autogrow4/sample_sub_scripts/.

Appendix D.6 Understanding AutoGrow4 Parameters

An explanation of every parameter can be retrieved by running

```
$ python /autogrow4/RunAutogrow.py --help
```

Custom options such as custom filters, docking software, reaction libraries, etc., are described in other parts of the tutorial. Additionally, a description of how to prepare the receptor file is also provided in the section "Appendix D.9: Preparing the Receptor." Details for preparing source compound files are provided directly below.

Appendix D.6.1 Source Compound Files

Source compound files are provided to AutoGrow4 as a path to tab-delineated SMILES files (.SMI). Specify the path using the parameter `--source_compound_file`.

Examples of source compound files can be found at: /autogrow4/source_compounds/

A detail log of how the examples files were prepared is located at: /autogrow4/source_compounds/Example_source_compound_notes.txt

An accessory script that converts a folder of PDB files to a tab-delineated .SMI file is provided at /autogrow4/accessory_scripts/convert_directory_ligands_pdb_to_smi.py

Details for using this accessory script are provided towards the bottom of this document, in the section "Appendix D.12.1: Preparation Scripts Pre-Run."

Seeding an AutoGrow4 Run with Already-Docked Compounds:

AutoGrow4 can be set to assess the fitness of source compounds in addition to the fitness of compounds created by AutoGrow4. This will dock the source compounds prior to seeding the first generation. It creates a generation 0, consisting only of the compounds in the source compound file.

To use this option, set `--use_docked_source_compounds` as True.

If one is running multiple independent runs using the same source compounds, it may be worth it to first test all source compounds and then seed all runs with the same scores for generation 0. This is accomplished by providing a source compound file with a float value in the second-to-last tab-delineated slot. Such a SMI file would be formatted like this:

SMILE Name ...any info... primary_fitness diversity_fitness

Please remember that docking scores are relative to a specific protein and pocket. Changing the coordinates, protein, docking software, or (re)scoring method will invalidate any information.

More information is provided in the `--use_docked_source_compounds` section of the RunAutoGrow.py help menu, which can be obtained by running

$ python RunAutoGrow.py –help

Appendix D.7 Docker Submission

The /autogrow4/docker/ directory contains the scripts to run AutoGrow4 within a Docker container. These scripts are useful when using an OS that is not compatible with AutoGrow4 or its dependencies, such as Windows.

Prior to running these scripts, please install the Docker software. Also, be sure to ALWAYS RUN THESE SCRIPTS WITH SUDO (LINUX/MACOS) OR ADMINISTRATOR PRIVILEGES (WINDOWS).

Running AutoGrow4 via Docker will take a few minutes longer the first time because Docker must install the dependencies. This is also true if the Docker images have been purged.

Depending on the AutoGrow4 settings, processor speed/count, etc., AutoGrow4 may complete within minutes or may take as long as multiple days. Please make sure to use settings that are appropriate for your system. Using nohup may be a useful wrapper for longer runs or when running jobs remotely (i.e., running a job over ssh).

More details are provided directly below and in the /autogrow4/docker/README.md section.

Appendix D.7.1 How to Setup AutoGrow4 in Docker

Dockerized AutoGrow4 requires the user to specify parameters via a JSON file (not the command line).

To run the autogrow_in_docker.py script:

Linux/MacOS:

1. Go into the /autogrow4/docker/ directory in a Bash terminal:

   $ cd /autogrow4/docker/

2. Run autogrow_in_docker.py with sudo and supply a .json file using the normal pathing of your system. Please note that the Docker downloads its own copy of obabel and MGLTools, so you do not need to provide those paths. Execute autogrow_in_docker.py with sudo privileges, providing it with a .json file (MUST EXECUTE FROM /autogrow4/docker/):

   $ sudo python autogrow_in_docker.py -j ./examples/sample_autogrow_docker_json.json

3. Results will appear in the output directory specified by the `--root_output_folder` parameter.

Windows OS:

1. Open a Docker-enabled and Bash-enabled terminal with administrator privileges.
2. Go to the /autogrow4/docker/ directory in a Bash enabled terminal:

   $ cd /autogrow4/docker/

3. Execute autogrow_in_docker.py with sudo privileges, providing it with a .json file (MUST EXECUTE FROM /autogrow4/docker/):

   $ python autogrow_in_docker.py -j ./examples/sample_autogrow_docker_json.json

4. Results will appear in the output directory specified by the `--root_output_folder` parameter.

Appendix D.8 Providing Custom Plugins

AutoGrow4 was designed to be modular. This allows for the easy swapping of code. AutoGrow4 is intended to be a living codebase. If you have added custom code and would like to make it open source, please contact the authors so that we can grow the user options.

Many of the AutoGrow4 functions can be supplemented with custom options. These functions include:

1. Custom Ligand Filters ***
2. Custom Docking Code ***
3. Custom Ligand Conversion Code from PDB to Dockable Format (e.g., PDBQT) ***
4. Custom Scoring/Rescoring Code ***
5. Custom Reaction Libraries
6. Custom Complementary Molecule Libraries

*** Indicates that when using this feature, the code is automatically copied into the appropriate AutoGrow4 directory. This is only done once, so please unit-test the code prior to incorporating it into AutoGrow4. A print message will indicate where the file has been copied to.

That file can be manually deleted or overwritten by the user. Restart AutoGrow4 after the custom files have been automatically copied into the proper locations. After that, the new script should be integrated into AutoGrow4.

AutoGrow4 ASSUMES ALL CUSTOM CODE HAS BEEN TESTED AND FUNCTIONS WITH SPECIFIED I/O. For example, it assumes that scoring favors the most negative docking score.

- AutoGrow4 will continue to assume all custom scoring scripts set the most fit score to the most negative for all metrics besides diversity.
- It also assumes in ranked .smi files that the last column is the diversity fitness and assumes the second-to-last column is the metric for "docking/rescored" fitness.
- If a custom script scores ligands such that the most fit ligand has the highest score, AutoGrow4 may inadvertently be favoring ligands that are least fit. If your favored scoring software sets the most fit ligand as most positive, please correct this by multiplying the custom docking score by -1.0.

Appendix D.8.1 Custom Ligand Filters ***

This feature allows the user to incorporate custom Python scripts for filtering ligands. These filters are applied to ligands after they are created by mutation/crossover but before Gypsum-DL conversion to 3D.

This custom code will be copied to the directory:
/autogrow4/autogrow/operators/filter/filter_classes/filter_children_classes/

Script Formatting:

These filters use a class-based inheritance architecture with filter classes that must:

1. Inherit ParentFilterClass located at: /autogrow4/autogrow/operators/filter/filter_classes/parent_filter_class.py
2. Have a unique name: class unique_name(ParentFilter) (unique_name cannot match one of the predefined filters).
3. Have at least one function called run_filter (run_filter takes a single variable which must be an RDKit molecule object).

Running Custom Filters:

Because parameters can be supplied to AutoGrow4 via command-line or JSON file, we provide an example of each when submitting custom filters.

1) Submission through JSON format:
   - Where the custom file is located at /PATH_TO/custom_filter_1.py
   - Unique class name is custom_filter_1 (this will be what it is called in future submissions)
   - To run multiple files, replace [["custom_filter_1","/PATH_TO/custom_filter_1.py"]] with: [["custom_filter_1","/PATH_TO/custom_filter_1.py"],["custom_filter_2","/PATH_TO/custom_filter_2.py"]]

```
{
   ...,
   "alternative_filter":          [["custom_filter_1","/PATH_TO/custom_filter_1.py"]]
}
```

Submit in terminal:

```
$ python RunAutogrow.py -j /PATH_TO/json_file_with_variable.json
```

2) Submission through command-line format:
   - Custom file is located at /PATH_TO/custom_filter_1.py
   - Unique class name is custom_filter_1 (this will be what it is called in future submissions)
   - To run multiple custom filters, replace [["custom_filter_1","/PATH_TO/custom_filter_1.py"]] with: [["custom_filter_1","/PATH_TO/custom_filter_1.py"],["custom_filter_2","/PATH_TO/custom_filter_2.py"]]

```
$ python RunAutogrow.py ... \
  --alternative_filter [["custom_filter_1","/PATH_TO/custom_filter_1.py"]]
```

Appendix D.8.2 Custom Docking Code ***

This feature allows the user to incorporate custom Python scripts for docking ligands. Currently AutoGrow4 is configured to dock using Autodock Vina[185] and QuickVina2[205], but AutoGrow4 is not limited to these docking programs. A custom script can be added to run docking using virtually any software.

This custom code will be copied to the directory: /autogrow4/autogrow/docking/docking_class/docking_class_children/

Script Formatting:

These docking scripts use a class-based inheritance architecture which require:

1. Docking class must inherit ParentDocking:

   /autogrow4/autogrow/docking/docking_class/parent_dock_class.py

2. Must have a unique name: class unique_name(ParentDocking) (unique_name cannot be one of the predefined docking scripts, which are currently just VinaDocking and QuickVina2Docking)

3. Must have at least have three functions following the below formatting:

```
def __init__(self, vars=None, receptor_file=None, test_boot=True):
    """
    get the specifications for ligand assessment/docking from vars
    load them into the self variables we will need
    and convert the receptor to the proper file format (ie pdb-> pdbqt)

    Inputs:
    :param dict vars: Dictionary of User variables
    :param str receptor_file: the path for the receptor pdb
    :param bool test_boot: used to initialize class without objects for testing purpose
    """

def run_dock(self, pdbqt_filename):
    """
    this function runs the docking. Returns None if it worked and the name if it failed to
dock.

    Inputs:
    :param str pdbqt_filename: the pdbqt file of a ligand to dock and score
            if using a docking software that use a file format other than pdbqt please
substitute that file here
    Returns:
    :returns: str smile_name: name of smiles if it failed to dock
                    returns None if it docked properly
    """

def rank_and_save_output_smi(self, vars, current_generation_dir, current_gen_int,
smile_file, deleted_smiles_names_list):
    """
    Given a folder with PDBQT's, rank all the SMILES based on docking score (High to
low).
    Then format it into a .smi file.
```

Then save the file.

Inputs:
:param dict vars: vars needs to be threaded here because it has the parallelizer object which is needed within Scoring.run_scoring_common
:param str current_generation_dir: path of directory of current generation
:param int current_gen_int: the interger of the current generation indexed to zero
:param str smile_file: File path for the file with the ligands for the generation which will be a .smi file
:param list deleted_smiles_names_list: list of SMILES which may have failed the conversion process

Returns:
:returns: str output_ranked_smile_file: the path of the output ranked .smi file
"""

Running Custom Docking Scripts:

Please note integrating a new docking software into AutoGrow4 will likely require corresponding custom conversion and scoring scripts. Documentation for these is provided in the next two subsections. The example below ignores these extras.

AutoGrow4 will need to be restarted after this has been incorporated into the code base.

1) Submission through JSON format:

— Where the JSON file is located at: /PATH_TO/To/json_file_with_variable.json

— Where the docking software executable is located at:

/PATH_TO/EXECUTABLE_FOR_CUSTOM_DOCKING/custom_docking

— Where the Python script for running docking is located at:

/PATH_TO/CLASS_OBJECT_FOR_CUSTOM_DOCKING/custom_docking.py

— Where the name of custom docking class is: custom_docking

{
...
"docking_executable":
"/PATH_TO/EXECUTABLE_FOR_CUSTOM_DOCKING/custom_docking",
"dock_choice": "Custom",

"custom_docking_script": ["custom_docking", "/PATH_TO/CLASS_OBJECT_FOR_CUSTOM_DOCKING/custom_docking.py"]
}

Submit via terminal

$ python RunAutogrow.py -j /PATH_TO/To/json_file_with_variable.json

2) Command-line submission format:

— Where docking software executable is located at:

/PATH_TO/EXECUTABLE_FOR_CUSTOM_DOCKING/custom_docking

— Where Python script for running docking is located at:

/PATH_TO/CLASS_OBJECT_FOR_CUSTOM_DOCKING/custom_docking.py

— Where name of custom docking class is: custom_docking

$ python RunAutogrow.py ... \

--docking_executable

"/PATH_TO/EXECUTABLE_FOR_CUSTOM_DOCKING/custom_docking" \

--dock_choice Custom \

--alternative_filter ["custom_docking",

"/PATH_TO/CLASS_OBJECT_FOR_CUSTOM_DOCKING/custom_docking.py"]

Appendix D.8.3 Custom Ligand Conversion Code from PDB to Dockable Format (e.g., PDBQT)

If using a docking software other than Vina/QuickVina, you may need to convert the PDB-formatted ligands into a different format. In this case, you must provide a custom script to convert the ligands.

This custom code will be copied to the directory: /autogrow4/autogrow/docking/docking_class/docking_class_children/

Script Formatting:

These conversion scripts use a class-based inheritance architecture:

1. Conversion class object must inherit ParentPDBQTConverter:

   /autogrow4/autogrow/docking/docking_class/parent_pdbqt_converter.py

2. Have a unique name: class unique_name(ParentPDBQTConverter) (unique_name cannot be one of the predefined docking scripts)
   - Currently files named: convert_with_mgltools.py and convert_with_obabel.py
   - Class names already in use are: MGLToolsConversion and ObabelConversion

3. Must have at least two functions following the below formatting:

```
def convert_receptor_pdb_files_to_pdbqt(self, receptor_file, mgl_python,
receptor_template, number_of_processors):
    """
    Make sure a PDB file is properly formatted for conversion to pdbqt

    Inputs:
    :param str receptor_file:  the file path of the receptor
    :param str mgl_python: file path of the pythonsh file of mgl tools
    :param str receptor_template: the receptor4.py file path from mgl tools.
    :param int number_of_processors: number of processors to multithread
```

```
        """
        raise NotImplementedError("convert_receptor_pdb_files_to_pdbqt() not
    implemented")

    def convert_ligand_pdb_file_to_pdbqt(self, pdb_file):
        """
        Convert the ligands of a given directory from pdb to pdbqt format

        Inputs:
        :param str pdb_file: the file name, a string.
        Returns:
        :returns: bool bool: True if it worked;
                        False if it's the gypsum param file or if it failed to make PDBQT
        :returns: str smile_name: name of the SMILES string from a pdb file
                            None if it's the param file
        """
        raise NotImplementedError("rank_and_save_output_smi() not implemented")
```

Running Custom Conversion Scripts:

AutoGrow4 will need to be restarted once this has been incorporated into the code base.

1) Submission through JSONformat:

— Where the .json is located at: /PATH_TO/To/json_file_with_variable.json

— Where the custom Python conversion script is located at:

/PATH_TO/CLASS_OBJECT_FOR/custom_conversion.py

— Where the name of custom conversion class is: custom_conversion

```
{
  ...
  "conversion_choice": "Custom",
  "custom_conversion_script": ["custom_conversion",
"/PATH_TO/CLASS_OBJECT_FOR/custom_conversion.py"]
}
```

Submit via terminal:

```
$ python RunAutogrow.py -j /PATH_TO/JSON_FILE/json_file_with_variable.json
```

2) Submission through command-line format:

— Where the custom Python conversion script is located at:

/PATH_TO/CLASS_OBJECT_FOR/custom_conversion.py

— Where the name of custom conversion class is: custom_conversion

```
$ python RunAutogrow.py ... \
  --conversion_choice Custom \
  --custom_conversion_script ["custom_conversion",
"/PATH_TO/CLASS_OBJECT_FOR/for/custom_conversion.py"]
```

Appendix D.8.4 Custom Scoring or Rescoring Code

This feature allows the user to incorporate custom Python scripts for scoring and rescoring ligands.

Currently AutoGrow4 is configured to dock using AutoDock Vina and QuickVina2. There are also two options to rescore a ligand using either NNScore 1 or NNScore 2. Additionally, ligand efficiency (dividing the score/rescore value by the number of non-hydrogen atoms) can be applied with any float-based scoring value.

Users can incorporate custom scoring and rescoring options into AutoGrow4. This custom code will be copied to the directory: /autogrow4/autogrow/docking/scoring/scoring_classes/

Script Formatting:

These (re)scoring scripts use a class-based inheritance architecture:

1. Scoring class object must inherit parent_scoring_class:

   /autogrow4/autogrow/docking/scoring/scoring_classes/parent_scoring_class.py

2. Have a unique name: class unique_name(parent_scoring_class)
   - unique_name cannot be one of the predefined docking scripts.
   - Currently files named: vina.py, nn1.py, nn2.py, and lig_efficiency.py
   - Class names already in use are: VINA, NN1, NN2, and LigEfficiency
3. Must have at least have two functions following the below formatting:

```
def get_name(self):
    """
    Returns the current class name.
    Returns:
    :returns: str self.__class__.__name__: the current class name.
    """
    return self.__class__.__name__

def run_scoring(self, input_string):
    """
    run_scoring is needs to be implemented in each class.
    Inputs:
    :param str input_string:  A string to raise an exception
    """
    raise NotImplementedError("run_scoring() not implemented")
```

Running Custom Scoring/Rescoring Scripts:

AutoGrow4 will need to be restarted once this has been incorporated into the code base.

1) Submission through JSON format:
   - Where the .json is located at: /PATH_TO/To/json_file_with_variable.json
   - Where the Python scoring script is located at: /PATH_TO/CLASS_OBJECT_FOR/custom_scoring.py
   - Where the name of custom scoring class is: custom_scoring_name

```
{
  ...
  "scoring_choice": "Custom",
  "custom_scoring_script": ["custom_scoring_name",
```

"/PATH_TO/CLASS_OBJECT_FOR/custom_scoring.py"]
}

Submit via terminal:

$ python RunAutogrow.py -j /PATH_TO/JSON_FILE/json_file_with_variable.json

2) Submission through command-line format:

— Where the Python scoring script is located at:

/PATH_TO/CLASS_OBJECT_FOR/custom_scoring.py

— Where the name of custom scoring class is: custom_scoring_name

$ python RunAutogrow.py ... \

--conversion_choice Custom \

--custom_conversion_script ["custom_scoring_name", "/PATH_TO/CLASS_OBJECT_FOR/custom_scoring.py"]

Appendix D.8.5 Custom Reaction Libraries

AutoGrow4 assumes all custom scripts have been unit-tested. Please ensure all reactions and libraries are accurate before using this option.

Unlike the other custom options, reaction libraries are stored as human-readable JSON dictionaries. In contrast, all other custom options use inherited class scripts. These .json files do not need to be incorporated into AutoGrow4 and thus require no restarting or copying of files.

Reaction libraries are stored as .json files and are dictionaries of dictionaries. The outer dictionary uses the reaction's name as the key and the sub-dictionary containing all information about the reaction as the item.

We provide a script to check complementary molecule and reaction libraries at: /autogrow4/accessory_scripts/test_complementary_mol_library.py

A tutorial is provided in the Accessory Scripts section of this document.

Three Requirements for Custom Reaction Libraries:

Custom reaction libraries require three pieces of information, each explained below.

Reaction Library .json File Contains Reactions and All Reaction Information:

Each sub-dictionary must contain the following information:

- "reaction_name": "Name of the reaction",
- "example_rxn_product": "SMILES of Product using example example_rxn_reactants",
- "example_rxn_reactants": ["SMILES of example reactant_1"],
  - If two or more reactants in reaction ["SMILES of example reactant_1","SMILES of example reactant_2",…]
- "functional_groups": ["functional group name reactant_1"],
  - If two or more reactants in reaction ["functional group name reactant_1","functional group name reactant_2",…]
- "group_smarts": ["functional_group SMARTS reactant_1"],
  - If two or more reactants in reaction ["functional_group SMARTS reactant_1","functional_group SMARTS reactant_2",…]

- “num_reactants”: 1,
  - (int) if 2 or more reactants change accordingly
- “reaction_string”: “reaction string i.e.,

  reactant_1_smart.reactant_2_smart>>product_SMART”,
  - This uses Daylights SMARTS reaction notation

- “RXN_NUM”: 3
  - (int) a unique reaction number. This is used in naming products of mutation. For example, a ligand named Gen_1_Mutant_72_867328 is a ligand from generation 1 created by the 72 reaction in a reaction library

Simplified Example of a Reaction library (from click_chem_rxns_library.json):


```
{
   "1_Epoxide_and_Alcohol":    {
      "reaction_name": "1_Epoxide_and_Alcohol",
      "example_rxn_product": "CC(C)(C)C(O)(OCCCF)C(C)(C)O",
      "example_rxn_reactants": ["CC(C)(C)C1(O)OC1(C)C", "FCCC(O)"],
      "functional_groups": ["Epoxide_clickchem", "Alcohol_clickchem"],
      "group_smarts": ["[CR1;H2,H1X4,H0X4]1O[CR1;H2,H1X4,H0X4]1",
"[#6&$([CR0,R1X3,R1X4])&!$([#6](=,-[OR0,SR0])[OR0])]-[OR0;H1,-]"],
      "num_reactants": 2,
      "reaction_string":
"[CR1;H2,H1X4,H0X4:1]1O[CR1;H2,H1X4,H0X4:2]1.[#6&$([CR0,R1X3,R1X4])&!$(
[#6](=,-[OR0,SR0])[OR0]):3]-[OR0;H1,-]>>O[C:1][C:2]O-[#6:3]",
      "RXN_NUM": 1
      },
   "2_Epoxide_and_Thiol":    {
      "reaction_name": "2_Epoxide_and_Thiol",
      "example_rxn_product": "CC(C)(C)C(O)(SCCC(=O)OC(=O)[O-])C(C)(C)O",
      "example_rxn_reactants": ["CC(C)(C)C1(O)OC1(C)C", "O=C([O-])OC(=O)CCS"],
      "functional_groups": ["Epoxide_clickchem", "Thiol_1R_clickchem"],
      "group_smarts": ["[CR1;H2,H1X4,H0X4]1O[CR1;H2,H1X4,H0X4]1",
"[#6&$([CR0,R1X3,R1X4])&!$([#6](=,-[OR0,SR0])[SR0])]-[SR0;H1,-]"],
      "num_reactants": 2,
```

```
      "reaction_string":
"[CR1;H2,H1X4,H0X4:1]1O[CR1;H2,H1X4,H0X4:2]1.[#6&$([CR0,R1X3,R1X4])&!$(
[#6](=,-[OR0,SR0])[SR0]):3]-[SR0;H1,-]>>O[C:1][C:2]S-[#6:3]",
      "RXN_NUM": 2
      },
   "3_Alkene_Oxidized_To_Epoxide":    {
      "reaction_name": "3_Alkene_Oxidized_To_Epoxide",
      "example_rxn_product": "CNC1(C)OC1(O)Br",
      "example_rxn_reactants": ["BrC(O)=C(C)NC"],
      "functional_groups": ["Alkene_clickchem"],
      "group_smarts": ["[CR0;X3,X2H1,X1H2]=[CR0;X3,X2H1,X1H2]"],
      "num_reactants": 1,
      "reaction_string":
"[CR0;X3,X2H1,X1H2:1]=[CR0;X3,X2H1,X1H2:2]>>[C:1]1O[C:2]1",
      "RXN_NUM": 3
      },
   ...,
}
```


PLEASE SEE THE EXAMPLE REACTION LIBRARIES FOUND AT:

- /autogrow4/autogrow/operators/mutation/smiles_click_chem/reaction_libraries/all_rxns/All_Rxns_rxn_library.json
- /autogrow4/autogrow/operators/mutation/smiles_click_chem/reaction_libraries/click_chem_rxns/click_chem_rxns_library.json
- /autogrow4/autogrow/operators/mutation/smiles_click_chem/reaction_libraries/robust_rxns/Robust_Rxns_rxn_library.json

Reaction libraries identify ligands capable of participating in a given reaction using the information found in the sub-dictionary's items "functional_groups" and "group_smarts."

Functional Group Library .json File Simple JSON Dictionary Containing Each Functional Group and Its Smarts Definition:

Functional group libraries are simple dictionaries of the functional groups used by a

reaction library. Every moiety used by the reaction library must have an entry in the functional group library.

Functional group libraries are formatted as such: (From click_chem_functional_groups.json):


```
{
   "Acid_Anhydride_Noncyclic_clickchem": "[*]C(=O)-[O;R0]-C(=O)[*]",
   "Alcohol_clickchem": "[#6&$([CR0,R1X3,R1X4])&!$([#6](=,-[OR0,SR0])[OR0])]-
[OR0;H1,-]",
   "Alkene_clickchem": "[CR0;X3,X2H1,X1H2]=[CR0;X3,X2H1,X1H2]",
   "Alkyne_clickchem": "[CR0;X2,X1H1]#[CR0;X2,X1H1]",
   "Amine_2R_clickchem":  "[#7;$([#7;H3+,H2R0X1]-[#6]),$([#7&!H3;H1R1X3](:,-
[#6R1]):,-[#6R1,#7R1]),$([#7&!H3;H2]-[#6]),$([#7&!H3;H0R1X2](:,-[#6R1;X3H1]):,-
[#6R1X3H1]),$([#7&!H3;H0R1X2](:,-[#6R1;X3]):,-[#7R1X3]),$([#7&!H3;H1R0X3](-
[#6])-[#6R0])]",
   "Azide_1R_clickchem": "[*;#6]-[$(N=[N+]=[N-]),$([N-][N+]#N)]",
   "Carbonochloridate_clickchem": "Cl[C;X3](=O)-O[*]",
   "Carboxylate_clickchem": "[*;!O]-
[$([CR0;X3](=[OR0&D1])[OR0&H1]),$([CR0;X3](=[OR0&D1])[OR0-])]",
   "Epoxide_clickchem": "[CR1;H2,H1X4,H0X4]1O[CR1;H2,H1X4,H0X4]1",
   "Ester_clickchem": "[*;#6]C(=O)-O[*]",
   "Halide_clickchem": "[Cl,Br,I][$([CX4,c]),$([#6X3]=[O,S])]",
   "Isocyanate_clickchem": "[#6]N=C=O",
   "Isothiocyanate_clickchem": "[#6]N=C=S",
   "Primary_Amine_1R_clickchem": "[#7;$([H3+]),$([H2R0;!+])]-[#6]",
   "Sulfonyl_Azide_clickchem": "[*]S(=O)(=O)-[$(N=[N+]=[N-]),$([N-][N+]#N)]",
   "Thio_Acid_clickchem": "[C]-
[$([CX3R0]([S;H1,X1])=[OX1]),$([CX3R0]([O;H1,X1])=[SX1])]",
   "Thiol_1R_clickchem": "[#6&$([CR0,R1X3,R1X4])&!$([#6](=,-[OR0,SR0])[SR0])]-
[SR0;H1,-]"
}
```


Examples can be found here:

- /autogrow4/autogrow/operators/mutation/smiles_click_chem/reaction_libraries/all_rxns/All_Rxns_functional_groups.json

- /autogrow4/autogrow/operators/mutation/smiles_click_chem/reaction_libraries/click_chem_rxns/click_chem_functional_groups.json
- /autogrow4/autogrow/operators/mutation/smiles_click_chem/reaction_libraries/robust_rxns/Robust_Rxns_functional_groups.json

The SMARTS strings provided in this file should also be present in each sub-dictionary of the reaction library .json file that references that functional group, placing the name of the group in the list of functional group names of reactants found under sub-dictionary key "functional_groups" and placing the SMARTS string of the group in the list of functional group SMARTS of reactants found under sub-dictionary k ey " group_smarts."

Directory of Complementary Molecule Libraries, Directory of .smi Files:

Any reaction containing more than one reactant will require a complementary molecule to supplement the reaction.

For this reason, we require a directory populated with .smi files containing small molecules that match each functional group.

The name of each .smi file should be the name of the functional group (the keys of the functional-group-library .json file) + .smi.

Example: The .smi file for t he functional group " Acid_Anhydride_Noncyclic_clickchem" should be:

/PATH_TO/complementary_mol_directory/Acid_Anhydride_Noncyclic_clickchem.smi

There must be one entry per functional group. Because names are cap sensitive in some OS and not in others, please check that your name is unique independent of caps.

Important Formatting Notes about the .smi File for complementary_mol_directory:

1. No headers are allowed in the file.
2. .smi files can be either tab or four-space delineated.
3. The only columns are the first two columns.
   - Column 1: SMILES string
   - Column 2: ligand name/identifier (ONE WORD, NO SPACES)

We strongly recommend thoroughly checking that each molecule in each library matches the intended functional group. If a ligand does not match the intended functional group, the reaction will fail and it will slow the process of mutant creation.

Running Custom Reactions:

Running a custom reaction library requires four parameters to be set:

1. rxn_library
2. rxn_library_file
3. function_group_library
4. complementary_mol_directory

1) Submission through JSON format:

— Where the .json is located at /PATH_TO/To/json_file_with_variable.json

— Where the reaction library .json file is located at /PATH_TO/rxn_library_file.json

— Where the function group .json file is located at /PATH_TO/function_group_library.json

— Where the directory of .smi for complementary libraries is located at /PATH_TO/complementary_mol_directory/

{

...

```
    "rxn_library": "Custom",
    "rxn_library_file": "/PATH_TO/rxn_library_file.json",
    "function_group_library": "/PATH_TO/function_group_library.json",
    "complementary_mol_directory": "/PATH_TO/complementary_mol_directory/",
}
```

Submit via terminal

```
$ python RunAutogrow.py -j /PATH_TO/json_file_with_variable.json
```

2) Submission through command-line format:

— Where the reaction library .json file is located at: /PATH_TO/rxn_library_file.json

— Where the function group .json file is located at:

/PATH_TO/function_group_library.json

— Where the directory of .smi for complementary libraries is located at:

/PATH_TO/complementary_mol_directory/

```
$ python RunAutogrow.py ... \
    --rxn_library Custom \
    --rxn_library_file /PATH_TO/rxn_library_file.json \
    --function_group_library /PATH_TO/function_group_library.json \
    --complementary_mol_directory /PATH_TO/complementary_mol_directory/
```

Appendix D.8.6 Custom Complementary Molecule Libraries

One can provide custom libraries of molecules to supplement reactions using the `--complementary_mol_directory` option.

This can be used in conjunction with any of the predefined reactions sets (i.e., click_chem_rxns, robust_rxns, all_rxns), but this requires that all functional groups used by those reaction libraries have a corresponding .smi file in the custom complementary_mol_directory.

We strongly recommend thoroughly checking that each molecule in each library matches the intended functional group. If a ligand does not match the intended functional group, the reaction will fail and it will slow the process of mutant creation.

We provide a script to check complementary molecule libraries at: /autogrow4/accessory_scripts/test_complementary_mol_library.py

A tutorial is provided in the "Appendix D.12.2: Preparing Custom Reaction Libraries Pre-Run."

THERE MUST BE ONE ENTRY PER FUNCTIONAL GROUP. BECAUSE NAMES ARE CAP SENSITIVE IN SOME OS'S AND NOT IN OTHERS, PLEASE CHECK THAT YOUR NAME IS UNIQUE INDEPENDENT OF CAPS.

Important Formatting Notes about the .smi File for complementary_mol_directory:

1. No headers are allowed in the file.
2. .smi files can be either tab or four-space delineated.
3. The only columns are the first two columns.
   - Column 1: SMILES string
   - Column 2: ligand name/identifier (ONE WORD, NO SPACES)

Running Custom Reactions:

1) Submission through JSON format:

— Where the .json is located at: /PATH_TO/To/json_file_with_variable.json

— Where the directory of .smi for complementary libraries is located at: /PATH_TO/complementary_mol_directory/

```
{
  ...
  "complementary_mol_directory": "/PATH_TO/complementary_mol_directory/",
}
```

Submit via terminal:

```
$ python RunAutogrow.py -j /PATH_TO/json_file_with_variable.json
```

2) Submission through command-line format:

— Where the directory of .smi for complementary libraries is located at: /PATH_TO/complementary_mol_directory/

```
$ python RunAutogrow.py ... \
  --complementary_mol_directory /PATH_TO/complementary_mol_directory/
```

Appendix D.9 Preparing the Receptor

AutoGrow4 takes a single .pdb file for the receptor. Although not required, we recommend carefully preparing the receptor file prior to submitting to AutoGrow4.

1. Remove all ligands, water, or non-protein atoms. This can be done in a PDB viewer such as PyMOL or VMD.
   – If a ligand is already bound to the target pocket, you may want to use that ligand to identify the pocket location prior to removing it.
2. Remove chains not being tested.

- i.e., many protein structures contain multiple protein chains and even multiple proteins. We recommend removing all chains you are not explicitly testing. This can be done in a PDB viewer such as PyMOL or VMD.

3. Adjust the protonation of the receptor to the appropriate pH. Crystal structures frequently do not include hydrogen atoms.
   - More accurate scoring requires proper protonation. This can be done using the program PDB2PQR[240,241], available via the webserver http://nbcr-222.ucsd.edu/pdb2pqr_2.0.0/
     - If you use the PDB2PQR to protonate the receptor, you will need to convert it back to .pdb.
       - To convert back, we recommend obabel. Installation instructions for obabel are provided in the Dependencies section.

```
$ obabel -ipqr /PATH_TO/PQR_FILE.pqr -opdb \
-O /PATH_TO/PDB_OUTPUT_FILE.pdb
```

4. Determine and define the binding pocket:
   - Docking software such as Vina and QuickVina require six float parameters to define a binding pocket:
     - Coordinates: The center of the pocket location in the x, y, z axis: center_x, center_y, center_z
     - Dimensions: The distance from the center of the pocket which will be considered part of the pocket in the x, y, z axis: size_x, size_y, size_z
   - AutoGrow4 requires all six parameters to run the docking portion of the code.

– To determine these, we recommend using the Python API library Scoria[242].

  • Citation:

  Ropp, P., Friedman, A., & Durrant, J. D. (2017). Scoria: A Python module for manipulating 3D molecular data. Journal of cheminformatics, 9(1), 52. doi:10.1186/s13321-017-0237-8

  • Installation of Scoria:

    – Scoria can be installed either by pip installation or manual download.

    – We recommend pip installation:

      $ pip install scoria

    – Download scoria from https://durrantlab.pitt.edu/scoria/

Determining the binding pocket using Scoria:

a. Manually inspect the pocket of your protein in a protein visualizer such as PyMOL, Chimera, or VMD.

  • Pick out three to six residues which will be used to define the protein pocket.

  • For the AutoGrow4 publication we used Chain A of the PARP-1 catalytic domain X-ray structure PDB:4R6E[17]. The selected residues used to define the pocket were: 763, 872, 888, 907, 988.

b. Determine the geometric center of the pocket with Scoria's get_geometric_center function in Python.

  • In a Python terminal or in a Jupyter environment:

```
# Import the scoria API
>> import scoria
>>
```

 # define your protein pdb file
 # The protein pdb file used for the publication can be found at:
/autogrow4/autogrow/tutorial/PARP/4r6eA_PARP1_prepared.pdb
>> pdb_file = "/PATH_TO/OF/PDB_FILE.pdb"

 # create a scoria mol object from the protein pdb file
>> mol = scoria.Molecule(pdb_file)

 # select which residues are going to be used to define pocket with resseq (the residue number)
>> sel = mol.select_atoms({"resseq":[763, 872, 888, 907, 988]})

 # get geometric center of the protein pocket
>> geometric_center = mol.get_geometric_center(sel)
>> print(geometric_center)
array([-70.75619481, 21.815, 28.32835065])
From this you can set: "center_x" = -70.756,"center_y" =21.815,"center_z"= 28.328

5. Determine the dimensions of the pocket.

Determining the binding pocket using Scoria's bounding_box function in Python:

 # Import the scoria API
>> import scoria
>>
 # define the protein molecule from the PDB file
>> mol = scoria.Molecule("/PATH_TO/OF/PDB_FILE.pdb")

 # select which residues are going to be used to define pocket with resseq (the residue number)
>> sel = mol.select_atoms({"resseq":[763, 872, 888, 907, 988]})

 # get the dimensions of the box that encompasses the protein pocket
>> bounding_box = mol.get_bounding_box(sel)
>> mol.get_bounding_box(sel)
array([[-83.764, 15.015, 15.305],
[-60.814, 29.578, 36.727]])
From this we need to take the difference from the first and second coordinate for

x,y,z:

a. 1st box coordinate: x_1st = -83.764, y_1st = 15.015, z_1st = 15.305

b. 2nd box coordinate: x_2nd = -60.814, y_2nd = 29.578, z_2nd = 36.727

c. Absolute value of diff from 1st and 2nd: "size_x" = 22.950,"size_y" = 14.563,"size_z"= 21.422

We suggest rounding these up to ensure the entire pocket is included: `“size_x” = 25.00,“size_y” = 16.00,“size_z”= 25.00

Appendix D.10 Other Factors for Consideration Prior to Running AutoGrow4

Appendix D.10.1 Processors and Multiprocessing Style

AutoGrow4 is recommended to be run on a larger computer or a cluster but it can be run on a local computer such as a laptop or PC.

If Running on a Laptop or PC:

We recommend lowering some AutoGrow4 parameters to reduce the computational overhead for smaller machines.

- Lower the population size and number of generations. This will mean a less intense search of chemistry space but will make run times more reasonable.
- Lower the max_variation to 1. This means for every ligand created by AutoGrow4, we will only create 1 conformer and thus only dock once per ligand. This of course means a trade-off of getting more useful information for each ligand for computational efficiency.

We also recommend considering how long you can allow the computer to run. If you need to continually use the computer while running AutoGrow4 then you want to fix the number_of_processors to leave several available to perform other activities.

If you can leave the computer to run undisturbed for an extended period, we recommend setting number_of_processors = -1, which will use all available processors.

If Running on a Larger Super Computer:

We recommend fixing the number_of_processors to however many processors you intend to run AutoGrow4. If number_of_processors = -1, then all available processors will be occupied running AutoGrow4.

If Running on a Cluster:

We recommend setting the number_of_processors = -1 and defining the number of processors in an SBATCH-type submission script.

Appendix D.11 Multiprocessing/MPI/Parallelization/Parallelizer

AutoGrow4 uses the Parallelizer.py script from Gypsum-DL[214]:
/autogrow4/autogrow/operators/convert_files/gypsum_dl/gypsum_dl/Parallelizer.py

This script creates a Parallelizer class object which can divide jobs in three manners:

1. Serial: run all jobs one at a time
2. Multiprocessing: dynamically allocate distribution of jobs across multiple CPUs on the same device
3. MPI: static allocation of jobs across many CPUs across multiple machines.

Appendix D.11.1 Important Notes when Running on Clusters Using SLURM

1. Multiprocessing: When running AutoGrow4 in Multiprocessing mode using SLURM:
    - First, run the cache_prerun option on a single processor.

        $ srun -n 1 python RunAutogrow.py -c

        – Use srun or mpirun for the cache_prerun. This limits the prerun to a single processor, thus preventing errors caused by race conditions when creating pycache files.
    - Then, run AutoGrow4 as intended.

        $ python RunAutogrow.py -j custom_parameters.json
2. MPI: When running AutoGrow4 in MPI mode using SLURM:
    - First, run the cache_prerun option on a single processor.

        $ srun -n 1 python RunAutogrow.py -c

        – USE srun or mpirun for the cache_prerun. This limits the prerun to a single processor thus preventing errors caused by race conditions when creating pycache files.
    - Then, run the simulation as intended.

$ mpirun -n num_processors python -m mpi4py RunAutogrow.py -j custom_parameters.json

- Make sure to provide the `-m mpi4py` before RunAutoGrow.py. This tells Python how to handle exceptions.

Appendix D.12 Accessory Scripts

AutoGrow4 provides several accessory scripts for preparing files, processing data, and analyzing data.

These files can be found within the /autogrow4/accessory_scripts/ folder.

Appendix D.12.1 Preparation Scripts Pre-Run

/autogrow4/accessory_scripts/remove_duplicates_from_smi.sh

This script accepts a file path to a tab-delineated .smi file. It then filters the file for redundancies in the first and second columns of the file.

The output file is the input file +`_no_dup.smi`.

This script uses Bash rather than Python because it is less memory intensive when dealing with large .smi files in the millions of compounds range. This is important when filtering through large databases such as ZINC15[230].

This script takes one input variable (filename str: Required). This is the path to the tab-delineated .smi file to remove any redundancies.

Example submit:

```
$ bash /autogrow4/accessory_scripts/remove_duplicates_from_smi.sh \
  /PATH_TO/TO/SMILES.smi
```

/autogrow4/accessory_scripts/convert_directory_ligands_pdb_to_smi.py

This script converts a directory of .pdb files (small molecules only, not proteins) to SMILES and creates a single .smi file with all SMILES.

This script takes three input arguments:

1. `--source_folder` str (-s) Required. Path to folder containing .pdb files to convert. File must contain a single small molecule without proteins. Files must end with either .pdb or .PDB'
2. `--output_folder` str (-o) Required. Path to folder where we will output a .smi file of converted .pdb files.
3. `--number_of_processors` int (-p). Number of processors to use for parallel calculations. This script is not MPI enable but is able to multithread using SMP architecture. Set to -1 for all available CPUs.

Example run:

```
$ python /autogrow4/accessory_scripts/convert_directory_ligands_pdb_to_smi.py \
  --source_folder /PATH_TO/OF/PDBS/ \
  --output_folder /PATH_TO/TO/OUTPUT/ \
  --number_of_processors -1
```

/autogrow4/accessory_scripts/fragmenter_of_smi_mol.py

This script will fragment compounds from a .smi file. It is useful for lead optimization. This script was used for the PARPi lead-optimization runs in the AutoGrow4 paper.

This can fragment compounds in two manners:

1. BRICS decomposition: This fragments along synthesizable bonds.
2. Fragment rotatable bonds: This breaks compounds along rotatable bonds. There is an option to skip carbon-carbon single bonds.

For each molecule, all permutations of fragments are calculated. For example, fragment rotatable bonds C-O-C1CCCC1 could produce any of the following fragments:

- C-O-C1CCCC1, not breaking any bonds.
- C and O-C1CCCC1, breaking the first bond.
- C-O and C1CCCC1, breaking the second bond.
- C and O and C1CCCC1, breaking the first bond and second bond.

A limit on maximum number of fragments per compound and a minimum number of atoms per fragment can be set.

This script takes seven input arguments:

1. `--smi_file` str Required. Path to tab-delineated .smi file to fragment.
2. `--output_smi_file` str (-o). Path to output tab-delineated .smi file of fragments. If not provided, it will play a file in the same directory as smi_file titled smi_file + _Fragmented.smi.
3. `--frags_per_seed_lig` int. Number of fragments to create per input SMILES. Default is -1, which mean all possible fragments.
4. `--run_brics` bool. Whether to fragment ligands using BRICS fragmentation. This fragments along synthesizable bonds. Default is True.
5. `--run_frag` bool. Whether to fragment ligands over all rotatable bonds. Default is True.

6. `--c_c_bonds_off` bool. Whether to exclude fragmenting carbon-carbon single bonds. Default is True. If True, it will ignore fragments on C-C bonds; if False, it will fragment.
7. `--number_of_processors` int (-p). Number of processors to use for parallel calculations. This script is not MPI enabled but is able to multithread using SMP architecture. Set to -1 for all available CPUs.

Example run:

```
$ python /autogrow4/accessory_scripts/fragmenter_of_smi_mol.py \
  --smi_file /PATH_TO/OF/SMILES.smi
```

Appendix D.12.2 Preparing Custom Reaction Libraries Pre-Run

/autogrow4/accessory_scripts/test_complementary_mol_library.py

This script will test a complementary molecule library to ensure all compounds react in all reactions they may be used in.

We recommend running this test if creating custom complementary libraries or reaction libraries. This script takes five input arguments:

1. `--rxn_library_file` str: Required. This path to a custom .json file of SMARTS reactions to use for mutation.
2. `--function_group_library` str: Required. This path for a dictionary of functional groups to be used for mutation.

3. `--complementary_mol_directory` str: Required. This path to the directory containing all the molecules being used to react with. The directory should contain .smi files contain SMILES of molecules containing the functional group represented by that file. Each file should be named with the same title as the functional groups described in rxn_library_file and function_group_library + .smi. All functional groups specified in function_group_library must have their own .smi file. We recommend filtering these dictionaries prior to AutoGrow4 for the drug-likeness and size filters you will run AutoGrow4 with.
4. `--output_folder` str: Required. This path to where filtered .smi files and log files will be placed. Will save a file in this directory for mols which failed sanitization, mols which failed to react in specific reactions, and .smi files that contain all mols that reacted properly.
5. `--number_of_processors` int (-p). Number of processors to use for parallel calculations. This script is not MPI-enabled but is able to multithread using SMP architecture. Set to -1 for all available CPUs.

Example submit:

$ python /autogrow4/accessory_scripts/test_complementary_mol_library.py \

--rxn_library_file /autogrow4/autogrow/operators/mutation/smiles_click_chem/reaction_libraries/click_chem_rxns/ClickChem_rxn_library.json \

--function_group_library /autogrow4/autogrow/operators/mutation/smiles_click_chem/reaction_libraries/click_chem_rxns/ClickChem_functional_groups.json \

--complementary_mol_directory
/autogrow4/autogrow/operators/mutation/smiles_click_chem/reaction_libraries/click_chem_rxns
/complementary_mol_dir \
--output_folder /autogrow4/accessory_scripts/output/

Appendix D.12.3 File Handling Post-Run

/autogrow4/accessory_scripts/convert_single_ligand_pdbqt_to_pdb.py:

This script will convert a .pdbqt file into a .pdb file. This is done by removing a column of the PDB file. This script takes two input arguments:

1. `--pdbqt_file` str (-f) Required. Path to .pdbqt file to convert to a .pdb file. This must be a single ligand and must end with .pdbqt.
2. `--output_folder` str (-o) Required. Path to file where we will output .pdb file. If not provided, the output .pdb will be the same as the input pdbqt_file but ending with .pdb instead of .pdbqt.

Example run:

$ python /autogrow4/accessory_scripts/convert_single_ligand_pdbqt_to_pdb.py \
--pdbqt_file /PATH_TO/OF/PDBQT_file.pdbqt \
--output_folder /PATH_TO/TO/OUTPUT/FOLDER/

/autogrow4/accessory_scripts/convert_vina_docked_pdbqt_to_pdbs.py:

This script will convert a docked .pdbqt.vina file into separate .pdb file. This is done by splitting up a single .pdbqt.vina into separate .pdbqt files for each docked pose. Then, it removes a column of the .pdbqt and saves it as a .pdb file.

- If parameter `--max_num_of_poses` is not set, it will convert all poses to .pdb.
- If `--max_num_of_poses` == 1, it will only convert the top docked pose to .pdb.
- If `--max_num_of_poses` == 2, it will only convert the top two docked poses to .pdb.
- If `--max_num_of_poses` == 10 but there only eight poses, it will convert the eight poses and stop.
- If `--max_docking_score` is not set, it will convert all poses to .pdb.
- If `--max_docking_score` == -10.0, it will only convert poses with docking scores less than or equal to -10.0.
- If both `--max_docking_score` and `--max_num_of_poses` are set, they work as AND type operators.
- If `--max_docking_score` == -11.4 and `--max_num_of_poses` == 5, it will take the top five poses as long as they also have docking scores <=-11.4

Remember, docking scores are better when more negative.

This script takes six input arguments:

1. `--vina_docked_pdbqt_file` str (-f): Required. Path to .pdbqt.vina file to split into one .pdb file per pose that matches all criteria. If this is a directory it will convert all of the files with the extension .pdbqt.vina.
2. `--output_folder` str (-o). Path to folder where the .pdb files will be placed. Files will be the basename of the docked file with *pose*{pose_number}.pdb replacing the extension .pdbqt.vina.
3. `--max_num_of_poses` int. Each docked file will have one or more poses of the ligand. This setting controls how many are converted. Default is -1 which means all poses possible. `--

max_num_of_poses'=1 means only the best-scored pose will be converted. If additional criteria like `--max_docking_score` is applied, a pose must meet both criteria to be converted. (i.e., if `--max_num_of_poses`= 5 and `--max_docking_score`=-13.0, for a pose to be converted it must be between the first and fifth pose in the file and must have docked with a score less than or equal to -13.0.)

4. `--max_docking_score` float. The most positive docking score to be converted. (More negative scores are predicted to bind better). If additional criteria such as `--max_num_of_poses` is applied, a pose must meet both criteria to be converted. (i.e., if `--max_num_of_poses`= 5 and `--max_docking_score`=-13.0, for a pose to be converted it must be between the first and fifth pose in the file and must have docked with a score less than or equal to -13.0.)
5. `--min_docking_score` float. The most negative docking score to be converted. (More negative scores are predicted to bind better). If additional criteria such as `--max_num_of_poses` is applied, a pose must meet both criteria to be converted. (i.e., if `--min_docking_score`= -15.0 and `--max_docking_score`=-13.0, for a pose to be converted it must: -13.0. <= docking score <= -15.0)
6. `--number_of_processors` int (-p). Number of processors to use for parallel calculations. This script is not MPI-enabled but is able to multithread using SMP architecture. Set to -1 for all available CPUs.

Example submit:

$ python /autogrow4/accessory_scripts/convert_vina_docked_pdbqt_to_pdbs.py \

--vina_docked_pdbqt_file

/PATH_TO/Run_1/Run_0/generation_30/PDBs/Gen_30_Cross_313228__1.pdbqt.vina \

--output_folder /PATH_TO/outfolder/ \

--max_num_of_poses 1 --number_of_processors -1

/autogrow4/accessory_scripts/convert_single_ligand_pdbqt_to_pdb.py:

This script is used to decompress or recompress AutoGrow4 data.

If you use the `--reduce_files_sizes` option during an AutoGrow4 run, AutoGrow4 will convert, concatenate, and compress all files in the PDB directory of each generation. This is useful when doing larger runs because data transfer will be faster, and data storage is reduced when files are merged and compressed.

The concatenation script that is run in AutoGrow4 can be found at /autogrow4/autogrow/docking/concatenate_files.py.

This script will either:

1. Return the files back to their original uncompressed and deconcatenated formatting.
2. Concatenate and then compress the files into a single file.

The formatting of the concatenation is:

"\n##############################File_name:
{}\n".format(os.path.basename(file_name_1))
... Content of the 1st file...
"\n##############################$$*END*_FILE$$
{}".format(os.path.basename(file_name_1))
"\n##############################File_name:
{}\n".format(os.path.basename(file_name_2))
... Content of the 2nd file...
"\n##############################$$*END*_FILE$$
{}".format(os.path.basename(file_name_2))

This concatenated file is then tar.gz compressed.

This script takes two input arguments:

1. `--compress_or_decompress` str (-s) Required. choices=["compress", "decompress"]. Choose whether to compress or decompress a directory
2. `--input_folder_or_file` str (-i) Required. Path to directory/file to compress or decompress.

Example decompression:

```
$ python /autogrow4/accessory_scripts/file_concatenation_and_compression.py \
    --compress_or_decompress decompress \
    --input_folder_or_file
PATH_TO_RUN/Run_0/generation_1/PDBs/compressed_PDBS.txt.gz
```

Example compression:

```
$ python /autogrow4/accessory_scripts/file_concatenation_and_compression.py \
    --compress_or_decompress compress \
    --input_folder_or_file PATH_TO_RUN/Run_0/generation_1/PDBs/
```

Appendix D.12.4 Graph Generation for Post-Run Analysis

/autogrow4/accessory_scripts/plot_autogrow_run.py:

This script will create a line plot of the average score for each AutoGrow4 generation. This is the same type of figure as the `--generate_plot` option that AutoGrow4 already provides, but it also allows plotting of reference lines.

This script takes four input arguments:

1. `--infolder` str (-i): Required. Path to input folder containing the AutoGrow4 run. This should be the top folder which contains the vars.json file.
2. `--outfile` str (-o). Path to folder to output files. It will be created if does not exist. If not provided, it will be placed in the infolder/data_line_plot.svg.
3. `--outfile_format` str. choices = [svg png jpg pdf]. The type of file for the figure to be exported as default is .svg file.
4. `--plot_reference_lines` list. This will be a list of lists, with each sublist being a different dotted-line reference to plot. For each sublist, the order of information should be: [name, value, matplotlib_color]
    - For example, [['Olaparib score',-12.8,'y'],['Niraparib score',-10.7,'k']] will add horizontal dotted lines at -12.8 (yellow) and -10.7 (black) with Olaparib and Niraparib added to the legend. Spaces must be within quotes and not be between variables. matplotlib colors can be found using mcolors.get_named_colors_mapping().keys()

Example submit:

$ python /autogrow4/accessory_scripts/plot_autogrow_run.py \

-i /PATH_TO/Run_1/Run_0/ \

--plot_reference_lines [['Olaparib Score',-12.8,'y'],['Niraparib',-10.7,'k'],['NAD/NADH',-10.3,'purple'],['ADP-ribose',-9.3,'maroon']]

/autogrow4/accessory_scripts/make_lineage_figures.py:

This script creates figures that list all ligands that parented a given ligand.

All compounds for the entire AutoGrow4 run will be compiled into a dictionary that is used to trace lineages. We pickle these dictionaries so these dictionaries do not need to be recreated

every time the script is run. For this reason, the first time running this script will take longer than future runs. A pre-run option will compile these data sets without generating figures.

1. `--output_dir` str (-o): Required. Path to output folder. Will be created if does not exist.
2. `--input_dir` str (-i): Required. Path to input folder containing the AutoGrow4 run. This should be the top folder which contains the vars.json file.
3. `--mol_name` str: Required unless prerun. This is the name of the molecule whose lineage will be traced back. If not provided or None, the script will simply compile the necessary dictionaries/pickle files and then terminate. These pickle files are stored in the input folder containing the vars.json file from the AutoGrow4 run. Example mol_name: Gen_5_Cross_203131 or Gen_4_Mutant_7_802531. Can also be provided as full-name, i.e.: (Gen_2_Mutant_7_97143)Gen_4_Mutant_7_802531
4. `--complementary_mol_directory` str. If using a custom complementary molecule library for mutations, this path is required. If not, the script will try to autodetect the location of the predefined complementary_mol_directory. Many molecules generated by mutation will required the complementary molecule that helped spawn them.
5. `--source_compound_file` str: Required. This is the source .smi file used to seed generation zero of the AutoGrow4 run. This is an essential file.
6. `--pre_run` bool. If True, this will compile the necessary dictionaries/pickle files and then terminate. These pickle files are stored in the input folder containing the vars.json file from the AutoGrow4 run.

    Example submit:

    $ python /autogrow4/accessory_scripts/make_lineage_figures.py \

```
-o /PATH_TO/output_folder/ \
-i /PATH_TO/INPUT_RUN/Run_3/ \
--source_compound_file /autogrow4/source_compounds/PARPI_BRICS_frags.smi \
--mol_name Gen_17_Cross_727024
```

Appendix D.13 Acknowledgements

I would like to thank the following people for their contributions to the AutoGrow4 codebase and benchmark experiments. Dr. Jacob Durrant generated the original idea for revising his line of AutoGrow software. He provided guidance in both code, experimental design, co-authoring the AutoGrow4 publication, editing this chapter of the dissertation, testing AutoGrow4 on macOS, and generating high-resolution versions of figures for the publication and dissertation.

I would like to thank Dr. Bennett Van Houten for his suggestion that I apply AutoGrow4 to PARP-1.

Additional contributions to AutoGrow4 include: Patrick J. Ropp for useful discussions and programming tips; Erich Hellemann for discussions and help with SMARTS reactions; Pauline Spiegel for manuscript editing and tutorial testing; Yuri Kochnev for compiling QVina2 for macOS; Kevin C. Cassidy for help with molecular rendering; and Harrison Green for help developing AutoGrow4 accessory scripts. I would also like to thank the University of Pittsburgh's Center for Research Computing for providing helpful computer resources. The default fragment libraries included with AutoGrow4 were derived from a subset of the ZINC database

(https://zinc.docking.org/). I thank ZINC for allowing me to distribute these fragment libraries to AutoGrow4 users.

Appendix D.14 Author Contributions

Jacob O. Spiegel wrote and tested the codebase of AutoGrow4 and all accessory scripts provided in the AutoGrow4 download. AutoGrow4 leverages many libraries and other programs which have been cited throughout the code and all related publications. Dr. Jacob Durrant and Jacob O. Spiegel authored the tutorial provided in the AutoGrow4 download. Only sections that I wrote were used verbatim in this dissertation. All writing in this appendix is original content written by Jacob O. Spiegel.

Bibliography

1. Balasubramaniam, S. *et al.* FDA approval summary: Rucaparib for the treatment of patients with deleterious BRCA mutation–associated advanced ovarian cancer. *Clin. Cancer Res.* **23**, 7165–7170 (2017).

2. Kim, G. *et al.* FDA approval summary: Olaparib monotherapy in patients with deleterious germline BRCA-mutated advanced ovarian cancer treated with three or more lines of chemotherapy. *Clin. Cancer Res.* **21**, 4257–4261 (2015).

3. Hoy, S. M. Talazoparib: First Global Approval. *Drugs* **78**, 1939–1946 (2018).

4. Ison, G. *et al.* FDA approval summary: Niraparib for the maintenance treatment of patients with recurrent ovarian cancer in response to platinum-based chemotherapy. *Clin. Cancer Res.* **24**, 4066–4071 (2018).

5. Horton, J. K. *et al.* Base excision repair defects invoke hypersensitivity to PARP inhibition. *Mol. Cancer Res.* **12**, 1128–1139 (2014).

6. Spiegel, J. O. & Durrant, J. D. AutoGrow4: an open-source genetic algorithm for de novo drug design and lead optimization. *J. Cheminform.* **12**, 25 (2020).

7. Stern, H. R., Sefcikova, J., Chaparro, V. E. & Beuning, P. J. Mammalian DNA Polymerase Kappa Activity and Specificity. *Molecules (Basel, Switzerland)* (2019). doi:10.3390/molecules24152805

8. Van Houten, B., Santa-Gonzalez, G. A. & Camargo, M. DNA repair after oxidative stress: Current challenges. *Curr. Opin. Toxicol.* **7**, 9–16 (2018).

9. Bitler, B. G., Watson, Z. L., Wheeler, L. J. & Behbakht, K. PARP inhibitors: Clinical utility and possibilities of overcoming resistance. *Gynecol. Oncol.* **147**, 695–704 (2017).

10. Dietlein, F., Thelen, L. & Reinhardt, H. C. Cancer-specific defects in DNA repair pathways

as targets for personalized therapeutic approaches. *Trends in Genetics* (2014). doi:10.1016/j.tig.2014.06.003

11. Deng, C. X. BRCA1: Cell cycle checkpoint, genetic instability, DNA damage response and cancer evolution. *Nucleic Acids Res.* **34**, 1416–1426 (2006).

12. Rouleau, M., Patel, A., Hendzel, M. J., Kaufmann, S. H. & Poirier, G. G. PARP inhibition: PARP1 and beyond. *Nat. Rev. Cancer* **10**, 293–301 (2010).

13. Turk, A. A. & Wisinski, K. B. PARP Inhibition in BRCA-Mutant Breast Cancer. *Cancer* **124**, 2498–2506 (2018).

14. Chen, C.-C., Feng, W., Lim, P. X., Kass, E. M. & Jasin, M. Homology-Directed Repair and the Role of BRCA1, BRCA2, and Related Proteins in Genome Integrity and Cancer. *Annu. Rev. Cancer Biol.* **2**, 313–336 (2018).

15. Lin, K. Molecular Mechanism of Poly(ADP-ribosyl)ation Catalyzed by Human Poly(ADP-ribose) Polymerase-1. (The University of Texas at Austin, 2015).

16. Alemasova, E. E. & Lavrik, O. I. Poly(ADP-ribosyl)ation by PARP1: reaction mechanism and regulatory proteins. *Nucleic Acids Res.* **47**, 3811–3827 (2019).

17. Thorsell, A.-G. G. *et al.* Structural Basis for Potency and Promiscuity in Poly(ADP-ribose) Polymerase (PARP) and Tankyrase Inhibitors. *J. Med. Chem.* **60**, 1262–1271 (2017).

18. Dawicki-McKenna, J. M. *et al.* PARP-1 Activation Requires Local Unfolding of an Autoinhibitory Domain. *Mol. Cell* **60**, 755–768 (2015).

19. Kinoshita, T. *et al.* Inhibitor-induced structural change of the active site of human poly(ADP-ribose) polymerase. *FEBS Lett.* **556**, 43–46 (2004).

20. Hegde, M. L., Hazra, T. K. & Mitra, S. Early steps in the DNA base excision/single-strand interruption repair pathway in mammalian cells. *Cell Res.* **18**, 27–47 (2008).

21. Kim, Y.-J. & M. Wilson III, D. Overview of Base Excision Repair Biochemistry. *Curr.*

Mol. Pharmacol. **5**, 3–13 (2011).

22. Caldecott, K. W. Single-strand break repair and genetic disease. *Nat. Rev. Genet.* **9**, 619–631 (2008).

23. Chatterjee, N. & Walker, G. C. Mechanisms of DNA damage, repair, and mutagenesis. *Environ. Mol. Mutagen.* **58**, 235–263 (2017).

24. Patel, A. G., Sarkaria, J. N. & Kaufmann, S. H. Nonhomologous end joining drives poly(ADP-ribose) polymerase (PARP) inhibitor lethality in homologous recombination-deficient cells. *Proc. Natl. Acad. Sci. U. S. A.* **108**, 3406–3411 (2011).

25. Chiruvella, K. K., Liang, Z. & Wilson, T. E. Repair of double-strand breaks by end joining. *Cold Spring Harb. Perspect. Biol.* **5**, 1–21 (2013).

26. Gassman, N. R., Stefanick, D. F., Kedar, P. S., Horton, J. K. & Wilson, S. H. Hyperactivation of PARP Triggers Nonhomologous End-Joining in Repair-Deficient Mouse Fibroblasts. *PLoS One* **7**, (2012).

27. Wang, M. *et al.* PARP-1 and Ku compete for repair of DNA double strand breaks by distinct NHEJ pathways. *Nucleic Acids Res.* **34**, 6170–6182 (2006).

28. Ko, H. L. & Ren, E. C. Functional Aspects of PARP1 in DNA Repair and Transcription. *Biomolecules* **2**, 524–548 (2012).

29. Sishc, B. J. & Davis, A. J. The role of the core non-homologous end joining factors in carcinogenesis and cancer. *Cancers (Basel).* **9**, (2017).

30. Montoni, A., Robu, M., Pouliot, É. & Shah, G. M. Resistance to PARP-Inhibitors in Cancer Therapy. *Front. Pharmacol.* **4**, 1–7 (2013).

31. Daniels, C. M., Ong, S. E. & Leung, A. K. L. The Promise of Proteomics for the Study of ADP-Ribosylation. *Mol. Cell* **58**, 911–924 (2015).

32. Wei, H. & Yu, X. Functions of PARylation in DNA Damage Repair Pathways. *Genomics,*

Proteomics and Bioinformatics (2016). doi:10.1016/j.gpb.2016.05.001

33. D'Amours, D., Desnoyers, S., D'Silva, I. & Poirier, G. G. Poly(ADP-ribosyl)ation reactions in the regulation of nuclear functions. *Biochemical Journal* **342**, 249–268 (1999).

34. Palazzo, L. *et al.* Serine is the major residue for ADP-ribosylation upon DNA damage. *Elife* (2018). doi:10.7554/eLife.34334

35. Amé, J. C., Spenlehauer, C. & De Murcia, G. The PARP superfamily. *BioEssays* (2004). doi:10.1002/bies.20085

36. Héberlé, E., Amé, J.-C., Illuzzi, G., Dantzer, F. & Schreiber, V. Discovery of the PARP Superfamily and Focus on the Lesser Exhibited But Not Lesser Talented Members. in (2015). doi:10.1007/978-3-319-14151-0_2

37. Javle, M. & Curtin, N. J. The role of PARP in DNA repair and its therapeutic exploitation. *British Journal of Cancer* (2011). doi:10.1038/bjc.2011.382

38. Pleschke, J. M., Kleczkowska, H. E., Strohm, M. & Althaus, F. R. Poly(ADP-ribose) Binds to Specific Domains in DNA Damage Checkpoint Proteins. *J. Biol. Chem.* **275**, 40974–40980 (2000).

39. Ahel, I. *et al.* Poly(ADP-ribose)-binding zinc finger motifs in DNA repair/checkpoint proteins. *Nature* (2008). doi:10.1038/nature06420

40. Wang, Z. *et al.* Recognition of the iso-ADP-ribose moiety in poly(ADP-ribose) by WWE domains suggests a general mechanism for poly (ADP-ribosyl)ation-dependent ubiquitination. *Genes Dev.* (2012). doi:10.1101/gad.182618.111

41. Kim, I.-K. *et al.* Structure of mammalian poly(ADP-ribose) glycohydrolase reveals a flexible tyrosine clasp as a substrate-binding element. *Nat. Struct. Mol. Biol.* **19**, 653–656 (2012).

42. Slade, D. *et al.* The structure and catalytic mechanism of a poly(ADP-ribose) glycohydrolase. *Nature* **477**, 616–620 (2011).

43. Helleday, T. The underlying mechanism for the PARP and BRCA synthetic lethality: Clearing up the misunderstandings. *Mol. Oncol.* **5**, 387–393 (2011).

44. David, K. K., Andrabi, S. A., Dawson, T. M. & Dawson, V. L. Parthanatos, A messenger of death. *Front. Biosci.* **14**, 1116–1128 (2009).

45. Langelier, M.-F. M. F., Planck, J. L., Roy, S. & Pascal, J. M. Structural Basis for DNA Damage-Dependent Poly(ADP-ribosyl)ation by Human PARP-1. *Science (80-.).* **336**, 728–732 (2012).

46. Steffen, J. D. *et al.* Targeting PARP-1 allosteric regulation offers therapeutic potential against cancer. *Cancer Res.* **74**, 31–37 (2014).

47. Langelier, M. F., Planck, J. L., Roy, S. & Pascal, J. M. Crystal structures of poly(ADP-ribose) polymerase-1 (PARP-1) zinc fingers bound to DNA: Structural and functional insights into DNA-dependent PARP-1 activity. *J. Biol. Chem.* (2011). doi:10.1074/jbc.M110.202507

48. Ikejima, M. *et al.* The zinc fingers of human poly(ADP-ribose) polymerase are differentially required for the recognition of DNA breaks and nicks and the consequent enzyme activation. Other structures recognize intact DNA. *J. Biol. Chem.* **265**, 21907–13 (1990).

49. Gradwohl, G. *et al.* The second zinc-finger domain of poly(ADP-ribose) polymerase determines specificity for single-stranded breaks in DNA. *Proc. Natl. Acad. Sci. U. S. A.* **87**, 2990–2994 (1990).

50. Langelier, M. F., Servent, K. M., Rogers, E. E. & Pascal, J. M. A third zinc-binding domain of human poly(ADP-ribose) polymerase-1 coordinates DNA-dependent enzyme activation. *J. Biol. Chem.* (2008). doi:10.1074/jbc.M708558200

51. Langelier, M. F., Ruhl, D. D., Planck, J. L., Kraus, W. L. & Pascal, J. M. The Zn3 domain of human poly(ADP-ribose) polymerase-1 (PARP-1) functions in both DNA-dependent poly(ADP-ribose) synthesis activity and chromatin compaction. *J. Biol. Chem.* **285**, 18877–18887 (2010).

52. Langelier, M.-F. & Pascal, J. M. PARP-1 mechanism for coupling DNA damage detection to poly(ADP-ribose) synthesis. *Curr. Opin. Struct. Biol.* **23**, 134–143 (2013).

53. Li, X. *et al.* Binding to WGR Domain by Salidroside Activates PARP1 and Protects Hematopoietic Stem Cells from Oxidative Stress. *Antioxid. Redox Signal.* **20**, 1853–1865 (2014).

54. Sigrist, C. J. A. *et al.* ProRule: a new database containing functional and structural information on PROSITE profiles. *Bioinformatics* **21**, 4060–4066 (2005).

55. Langelier, M. F., Zandarashvili, L., Aguiar, P. M., Black, B. E. & Pascal, J. M. NAD+ analog reveals PARP-1 substrate-blocking mechanism and allosteric communication from catalytic center to DNA-binding domains. *Nat. Commun.* **9**, 844 (2018).

56. Steffen, J. D., Brody, J. R., Armen, R. S. & Pascal, J. M. Structural implications for selective targeting of PARPs. *Frontiers in Oncology* (2013). doi:10.3389/fonc.2013.00301

57. Barkauskaite, E., Jankevicius, G. & Ahel, I. Structures and Mechanisms of Enzymes Employed in the Synthesis and Degradation of PARP-Dependent Protein ADP-Ribosylation. *Molecular Cell* (2015). doi:10.1016/j.molcel.2015.05.007

58. Vyas, S. *et al.* Family-wide analysis of poly(ADP-ribose) polymerase activity. *Nat. Commun.* (2014). doi:10.1038/ncomms5426

59. Ruf, A., Rolli, V., De Murcia, G. & Schulz, G. E. The mechanism of the elongation and branching reaction of Poly(ADP-ribose) polymerase as derived from crystal structures and mutagenesis. *J. Mol. Biol.* (1998). doi:10.1006/jmbi.1998.1673

60. Jungmichel, S. *et al.* Proteome-wide Identification of Poly(ADP-Ribosyl)ation Targets in Different Genotoxic Stress Responses. *Mol. Cell* **52**, 272–285 (2013).

61. Altmeyer, M., Messner, S., Hassa, P. O., Fey, M. & Hottiger, M. O. Molecular mechanism of poly(ADP-ribosyl)ation by PARP1 and identification of lysine residues as ADP-ribose acceptor sites. *Nucleic Acids Res.* (2009). doi:10.1093/nar/gkp229

62. Tao, Z., Gao, P. & Liu, H. W. Identification of the ADP-ribosylation sites in the PARP-1 automodification domain: Analysis and implications. *J. Am. Chem. Soc.* (2009). doi:10.1021/ja906135d

63. Sharifi, R. *et al.* Deficiency of terminal ADP-ribose protein glycohydrolase TARG1/C6orf130 in neurodegenerative disease. *EMBO J.* (2013). doi:10.1038/emboj.2013.51

64. Chapman, J. D., Gagné, J. P., Poirier, G. G. & Goodlett, D. R. Mapping PARP-1 auto-ADP-ribosylation sites by liquid chromatography-tandem mass spectrometry. *J. Proteome Res.* (2013). doi:10.1021/pr301219h

65. Zhang, Y., Wang, J., Ding, M. & Yu, Y. Site-specific characterization of the Asp-and Glu-ADP-ribosylated proteome. *Nat. Methods* (2013). doi:10.1038/nmeth.2603

66. Daniels, C. M., Ong, S. E. & Leung, A. K. L. Phosphoproteomic approach to characterize protein mono- and poly(ADP-ribosyl)ation sites from cells. *J. Proteome Res.* (2014). doi:10.1021/pr401032q

67. Gagné, J. P. *et al.* Quantitative site-specific ADP-ribosylation profiling of DNA-dependent PARPs. *DNA Repair (Amst).* (2015). doi:10.1016/j.dnarep.2015.02.004

68. Masson, M. *et al.* XRCC1 Is Specifically Associated with Poly(ADP-Ribose) Polymerase and Negatively Regulates Its Activity following DNA Damage. *Mol. Cell. Biol.* (1998). doi:10.1128/mcb.18.6.3563

69. Beernink, P. T. *et al.* Specificity of protein interactions mediated by BRCT domains of the XRCC1 DNA repair protein. *J. Biol. Chem.* (2005). doi:10.1074/jbc.M502155200

70. Eustermann, S. *et al.* Structural Basis of Detection and Signaling of DNA Single-Strand Breaks by Human PARP-1. *Mol. Cell* **60**, 742–754 (2015).

71. Trucco, C., Flatter, E., Fribourg, S., de Murcia, G. & Ménissier-de Murcia, J. Mutations in the amino-terminal domain of the human poly(ADP-ribose) polymerase that affect its catalytic activity but not its DNA binding capacity. *FEBS Lett.* **399**, 313–316 (1996).

72. Miranda, E. A., Dantzer, F., O'Farrell, M., de Demurcia, G. & De murcia, J. M. Characterization of a gain-of-function mutant of poly(ADP-Ribose) polymerase. *Biochem. Biophys. Res. Commun.* (1995). doi:10.1006/bbrc.1995.1972

73. Zandarashvili, L. *et al.* Structural basis for allosteric PARP-1 retention on DNA breaks. *Science (80-.).* **368**, eaax6367 (2020).

74. Liu, L. *et al.* PARP1 changes from three-dimensional DNA damage searching to one-dimensional diffusion after auto-PARylation or in the presence of APE1. *Nucleic Acids Res.* **45**, 12834–12847 (2017).

75. Hopkins, T. A. *et al.* Mechanistic dissection of PARP1 trapping and the impact on in vivo tolerability and efficacy of PARP inhibitors. *Mol. Cancer Res.* **13**, 1465–1477 (2015).

76. Hopkins, T. A. *et al.* PARP1 trapping by PARP inhibitors drives cytotoxicity in both cancer cells and healthy bone marrow. *Mol. Cancer Res.* (2019). doi:10.1158/1541-7786.MCR-18-0138

77. Murai, J. *et al.* Trapping of PARP1 and PARP2 by Clinical PARP Inhibitors. *Cancer Res.* **72**, 5588–5599 (2012).

78. Pettitt, S. J. *et al.* Genome-wide and high-density CRISPR-Cas9 screens identify point mutations in PARP1 causing PARP inhibitor resistance. *Nat. Commun.* **9**, 1849 (2018).

79. Park, C. H. PARP complexed with A861695. (2007). doi:10.2210/pdb2RD6/pdb

80. Nicolas, E., Bertucci, F., Sabatier, R. & Gonçalves, A. Targeting BRCA deficiency in breast cancer: What are the clinical evidences and the next perspectives? *Cancers (Basel).* **10**, 1–22 (2018).

81. Konstantinopoulos, P. A., Ceccaldi, R., Shapiro, G. I. & D'Andrea, A. D. Homologous Recombination Deficiency: Exploiting the Fundamental Vulnerability of Ovarian Cancer. *Cancer Discov.* **5**, 1137–1154 (2015).

82. Sztupinszki, Z. *et al.* Detection of Molecular Signatures of Homologous Recombination Deficiency in Prostate Cancer with or without BRCA1/2 Mutations. *Clin. Cancer Res.* 2673–2681 (2020). doi:10.1158/1078-0432.ccr-19-2135

83. Zhu, H. *et al.* PARP inhibitors in pancreatic cancer: Molecular mechanisms and clinical applications. *Mol. Cancer* **19**, 1–15 (2020).

84. Bray, F. *et al.* Global cancer statistics 2018: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA. Cancer J. Clin.* (2018). doi:10.3322/caac.21492

85. Torre, L. A. *et al.* Ovarian cancer statistics, 2018. *CA. Cancer J. Clin.* **68**, 284–296 (2018).

86. Mersch, J. *et al.* Cancers associated with BRCA 1 and BRCA 2 mutations other than breast and ovarian. *Cancer* **121**, 269–275 (2015).

87. Konstantinopoulos, P. A. & Awtrey, C. S. Management of ovarian cancer: A 75-year-old woman who has completed treatment. *JAMA - Journal of the American Medical Association* (2012). doi:10.1001/jama.2012.269

88. Cannistra, S. A. Cancer of the ovary. *New England Journal of Medicine* (2004). doi:10.1056/NEJMra041842

89. Rossof, A. H. *et al.* Phase II evaluation of cis-dichlorodiammineplatinum(II) in advanced malignancies of the genitourinary and gynecologic organs: A southwest oncology group study. *Cancer Treat. Rep.* (1979).

90. Thigpen, T., Shingleton, H., Homesley, H., LaGasse, L. & Blessing, J. cis-Dichlorodiammineplatinum(II) in the treatment of gynecologic malignancies: Phase II trials by the gynecologic oncology group. *Cancer Treat. Rep.* (1979).

91. Ozols, R. F. *et al.* Phase III trial of carboplatin and paclitaxel compared with cisplatin and paclitaxel in patients with optimally resected stage III ovarian cancer: A Gynecologic Oncology Group study. *J. Clin. Oncol.* (2003). doi:10.1200/JCO.2003.02.153

92. Mikuła-Pietrasik, J. *et al.* Comprehensive review on how platinum- and taxane-based chemotherapy of ovarian cancer affects biology of normal cells. *Cell. Mol. Life Sci.* **76**, 681–697 (2019).

93. Weaver, B. A. How Taxol/paclitaxel kills cancer cells. *Mol. Biol. Cell* **25**, 2677–2681 (2014).

94. Oun, R., Moussa, Y. E. & Wheate, N. J. The side effects of platinum-based chemotherapy

drugs: A review for chemists. *Dalton Transactions* (2018). doi:10.1039/c8dt00838h

95. Ferraris, D. V. Evolution of Poly(ADP-ribose) Polymerase-1 (PARP-1) Inhibitors. From Concept to Clinic. *J. Med. Chem.* **53**, 4561–4584 (2010).

96. Ossovskaya, V., Koo, I. C., Kaldjian, E. P., Alvares, C. & Sherman, B. M. Upregulation of poly (ADP-Ribose) polymerase-1 (PARP1) in triple-negative breast cancer and other primary human tumor types. *Genes and Cancer* (2010). doi:10.1177/1947601910383418

97. Nijman, S. M. B. Synthetic lethality: General principles, utility and detection using genetic screens in human cells. *FEBS Letters* (2011). doi:10.1016/j.febslet.2010.11.024

98. Edwards, S. L. *et al.* Resistance to therapy caused by intragenic deletion in BRCA2. *Nature* (2008). doi:10.1038/nature06548

99. Murai, J. *et al.* Stereospecific PARP trapping by BMN 673 and comparison with olaparib and rucaparib. *Mol. Cancer Ther.* (2014). doi:10.1158/1535-7163.MCT-13-0803

100. Shen, Y., Aoyagi-Scharber, M. & Wang, B. Trapping poly(ADP-Ribose) polymerase. *Journal of Pharmacology and Experimental Therapeutics* (2015). doi:10.1124/jpet.114.222448

101. Oza, A. M. *et al.* Olaparib combined with chemotherapy for recurrent platinum-sensitive ovarian cancer: A randomised phase 2 trial. *Lancet Oncol.* (2015). doi:10.1016/S1470-2045(14)71135-0

102. Veneris, J. T., Matulonis, U. A., Liu, J. F. & Konstantinopoulos, P. A. Choosing wisely: Selecting PARP inhibitor combinations to promote anti-tumor immune responses beyond BRCA mutations. *Gynecologic Oncology* (2020). doi:10.1016/j.ygyno.2019.09.021

103. FDA approves olaparib for germline BRCA-mutated metastatic breast cancer. (2018). Available at: https://www.fda.gov/drugs/resources-information-approved-drugs/fda-approves-olaparib-germline-brca-mutated-metastatic-breast-cancer. (Accessed: 4th June 2020)

104. FDA approves olaparib for HRR gene-mutated metastatic castration-resistant prostate

cancer. (2020). Available at: https://www.fda.gov/drugs/drug-approvals-and-databases/fda-approves-olaparib-hrr-gene-mutated-metastatic-castration-resistant-prostate-cancer. (Accessed: 4th June 2020)

105. FDA approves olaparib for gBRCAm metastatic pancreatic adenocarcinoma. (2019). Available at: https://www.fda.gov/drugs/resources-information-approved-drugs/fda-approves-olaparib-gbrcam-metastatic-pancreatic-adenocarcinoma. (Accessed: 4th June 2020)

106. Ye, N. *et al.* Design, synthesis, and biological evaluation of a series of benzo[de][1,7]naphthyridin-7(8 H)-ones bearing a functionalized longer chain appendage as novel PARP1 inhibitors. *J. Med. Chem.* **56**, 2885–2903 (2013).

107. Patel, M. R. *et al.* Discovery and structure-activity relationship of novel 2,3-dihydrobenzofuran-7-carboxamide and 2,3-dihydrobenzofuran-3(2 h)-one-7-carboxamide derivatives as poly(ADP-ribose)polymerase-1 Inhibitors. *J. Med. Chem.* **57**, 5579–5601 (2014).

108. Aoyagi-Scharber, M. *et al.* Structural basis for the inhibition of poly(ADP-ribose) polymerases 1 and 2 by BMN 673, a potent inhibitor derived from dihydropyridophthalazinone. *Acta Crystallogr. Sect. Struct. Biol. Commun.* **70**, 1143–1149 (2014).

109. Papeo, G. *et al.* Discovery of 2-[1-(4,4-Difluorocyclohexyl)piperidin-4-yl]-6-fluoro-3-oxo-2,3-dihydro-1H-isoindole-4-carboxamide (NMS-P118): A Potent, Orally Available, and Highly Selective PARP-1 Inhibitor for Cancer Therapy. *J. Med. Chem.* **58**, 6875–6898 (2015).

110. Velagapudi, U. K. *et al.* Design and Synthesis of Poly(ADP-ribose) Polymerase Inhibitors: Impact of Adenosine Pocket-Binding Motif Appendage to the 3-Oxo-2,3-dihydrobenzofuran-7-carboxamide on Potency and Selectivity. *J. Med. Chem.* **62**, 5330–5357 (2019).

111. Chen, X. *et al.* Design and synthesis of 2-(4,5,6,7-tetrahydrothienopyridin-2-yl)-benzoimidazole carboxamides as novel orally efficacious Poly(ADP-ribose)polymerase (PARP) inhibitors. *Eur. J. Med. Chem.* **145**, 389–403 (2018).

112. Hattori, K. *et al.* Rational approaches to discovery of orally active and brain-penetrable

quinazolinone inhibitors of poly(ADP-ribose)polymerase. *J. Med. Chem.* **47**, 4151–4154 (2004).

113. Noordermeer, S. M. & van Attikum, H. PARP Inhibitor Resistance: A Tug-of-War in BRCA-Mutated Cells. *Trends Cell Biol.* **29**, 820–834 (2019).

114. Goodall, J. *et al.* Circulating cell-free DNA to guide prostate cancer treatment with PARP inhibition. *Cancer Discov.* (2017). doi:10.1158/2159-8290.CD-17-0261

115. Norquist, B. *et al.* Secondary Somatic Mutations Restoring BRCA1/2 Predict Chemotherapy Resistance in Hereditary Ovarian Carcinomas. *J. Clin. Oncol.* **29**, 3008–3015 (2011).

116. Tanino, H. *et al.* BRCAness and prognosis in triple-negative breast cancer patients treated with neoadjuvant chemotherapy. *PLoS One* (2016). doi:10.1371/journal.pone.0165721

117. Akashi-Tanaka, S. *et al.* BRCAness predicts resistance to taxane-containing regimens in triple negative breast cancer during neoadjuvant chemotherapy. *Clin. Breast Cancer* (2015). doi:10.1016/j.clbc.2014.08.003

118. Wang, J. *et al.* Poly(ADP-ribose) polymerase-1 down-regulates BRCA2 expression through the BRCA2 promoter. *J. Biol. Chem.* (2008). doi:10.1074/jbc.M803693200

119. Moskwa, P. *et al.* MiR-182-Mediated Downregulation of BRCA1 Impacts DNA Repair and Sensitivity to PARP Inhibitors. *Mol. Cell* (2011). doi:10.1016/j.molcel.2010.12.005

120. De Summa, S. *et al.* BRCAness: A deeper insight into basal-like breast tumors. *Ann. Oncol.* **24**, viii13–viii21 (2013).

121. Liu, X. *et al.* Acquired resistance to combination treatment with temozolomide and ABT-888 is mediated by both base excision repair and homologous recombination DNA repair pathways. *Mol. Cancer Res.* (2009). doi:10.1158/1541-7786.MCR-09-0299

122. Kondrashova, O. *et al.* Secondary somatic mutations restoring RAD51C and RAD51D associated with acquired resistance to the PARP inhibitor rucaparib in high-grade ovarian carcinoma. *Cancer Discov.* (2017). doi:10.1158/2159-8290.CD-17-0419

123. Bunting, S. F. *et al.* 53BP1 Inhibits Homologous Recombination in Brca1-Deficient Cells by Blocking Resection of DNA Breaks. *Cell* **141**, 243–254 (2010).

124. Martin, R. W. *et al.* RAD51 up-regulation bypasses BRCA1 function and is a common feature of BRCA1-deficient breast tumors. *Cancer Res.* (2007). doi:10.1158/0008-5472.CAN-07-0290

125. Wang, Y., Huang, J. W., Calses, P., Kemp, C. J. & Taniguchi, T. MiR-96 downregulates REV1 and RAD51 to promote cellular sensitivity to cisplatin and PARP inhibition. *Cancer Res.* (2012). doi:10.1158/0008-5472.CAN-12-0103

126. Sourisseau, T. *et al.* Aurora-A expressing tumour cells are deficient for homology-directed DNA double strand-break repair and sensitive to PARP inhibition. *EMBO Mol. Med.* (2010). doi:10.1002/emmm.201000068

127. Mendes-Pereira, A. M. *et al.* Synthetic lethal targeting of PTEN mutant cells with PARP inhibitors. *EMBO Mol. Med.* (2009). doi:10.1002/emmm.200900041

128. Dedes, K. J. *et al.* PTEN Deficiency in Endometrioid Endometrial Adenocarcinomas Predicts Sensitivity to PARP Inhibitors. *Sci. Transl. Med.* **2**, 53ra75-53ra75 (2010).

129. Aly, A. & Ganesan, S. BRCA1, PARP, and 53BP1: conditional synthetic lethality and synthetic viability. *J. Mol. Cell Biol.* **3**, 66–74 (2011).

130. Yazinski, S. A. *et al.* ATR inhibition disrupts rewired homologous recombination and fork protection pathways in PARP inhibitor-resistant BRCA-deficient cancer cells. *Genes Dev.* **31**, 318–332 (2017).

131. Chiarugi, A. A snapshot of chemoresistance to PARP inhibitors. *Trends Pharmacol. Sci.* **33**, 42–48 (2012).

132. Schlacher, K., Wu, H. & Jasin, M. A Distinct Replication Fork Protection Pathway Connects Fanconi Anemia Tumor Suppressors to RAD51-BRCA1/2. *Cancer Cell* (2012). doi:10.1016/j.ccr.2012.05.015

133. Chaudhuri, A. R. *et al.* Replication fork stability confers chemoresistance in BRCA-

deficient cells. *Nature* (2016). doi:10.1038/nature18325

134. Cong, K. *et al.* PARPi synthetic lethality derives from replication-associated single-stranded DNA gaps. *bioRxiv* (2019). doi:10.1101/781989

135. Mweempwa, A. & Wilson, M. K. Mechanisms of resistance to PARP inhibitors - an evolving challenge in oncology. *Cancer Drug Resist.* 608–617 (2019). doi:10.20517/cdr.2019.50

136. Jaspers, J. E. *et al.* Loss of 53BP1 causes PARP inhibitor resistance in BRCA1-mutated mouse mammary tumors. *Cancer Discov.* (2013). doi:10.1158/2159-8290.CD-12-0049

137. Rottenberg, S. *et al.* High sensitivity of BRCA1-deficient mammary tumors to the PARP inhibitor AZD2281 alone and in combination with platinum drugs. *Proc. Natl. Acad. Sci. U. S. A.* (2008). doi:10.1073/pnas.0806092105

138. Durmus, S. *et al.* Breast cancer resistance protein (BCRP/ABCG2) and P-glycoprotein (P-GP/ABCB1) restrict oral availability and brain accumulation of the PARP inhibitor rucaparib (AG-014699). *Pharm. Res.* **32**, 37–46 (2015).

139. Oplustilova, L. *et al.* Evaluation of candidate biomarkers to predict cancer cell sensitivity or resistance to PARP-1 inhibitor treatment. *Cell Cycle* **11**, 3837–3850 (2012).

140. Gottipati, P. *et al.* Poly(ADP-Ribose) Polymerase Is Hyperactivated in Homologous Recombination–Defective Cells. *Cancer Res.* **70**, 5389–5398 (2010).

141. Boccaccio, C. & Comoglio, P. M. Invasive growth: A MET-driven genetic programme for cancer and stem cells. *Nature Reviews Cancer* (2006). doi:10.1038/nrc1912

142. Du, Y. *et al.* Blocking c-Met–mediated PARP1 phosphorylation enhances anti-tumor effects of PARP inhibitors. *Nat. Med.* **22**, 194–201 (2016).

143. Gogola, E. *et al.* Selective Loss of PARG Restores PARylation and Counteracts PARP Inhibitor-Mediated Synthetic Lethality. *Cancer Cell* **33**, 1078-1093.e12 (2018).

144. Marchand, J. R. *et al.* Investigating the allosteric reverse signalling of PARP inhibitors with microsecond molecular dynamic simulations and fluorescence anisotropy. *Biochim. Biophys. Acta - Proteins Proteomics* **1844**, 1765–1772 (2014).

145. Manning, T., Sleator, R. D. & Walsh, P. Naturally selecting solutions: the use of genetic algorithms in bioinformatics. *Bioengineered* **4**, 266–278 (2013).

146. Eiben, A. E. & Smith, J. E. *Introduction to evolutionary computing genetic algorithms*. *Natural Computing Series* (2003). doi:10.1007/978-3-662-44874-8

147. Casas, N. Genetic Algorithms for multimodal optimization: a review. (2015).

148. Ertl, P. Cheminformatics analysis of organic substituents: Identification of the most common substituents, calculation of substituent properties, and automatic identification of drug-like bioisosteric groups. in *Journal of Chemical Information and Computer Sciences* **43**, 374–380 (2003).

149. Bohacek, R. S., McMartin, C. & Guida, W. C. The art and practice of structure-based drug design: A molecular modeling perspective. *Medicinal Research Reviews* (1996). doi:10.1002/(SICI)1098-1128(199601)16:1<3::AID-MED1>3.0.CO;2-6

150. Polishchuk, P. G., Madzhidov, T. I. & Varnek, A. Estimation of the size of drug-like chemical space based on GDB-17 data. *J. Comput. Aided. Mol. Des.* (2013). doi:10.1007/s10822-013-9672-4

151. Shukla, A., Pandey, H. M. & Mehrotra, D. Comparative review of selection techniques in genetic algorithm. *2015 1st Int. Conf. Futur. Trends Comput. Anal. Knowl. Manag. ABLAZE 2015* 515–519 (2015). doi:10.1109/ABLAZE.2015.7154916

152. Obolski, U., Ram, Y. & Hadany, L. Key Issues Review: Evolution on rugged adaptive landscapes. *bioRxiv* **81**, 112177 (2017).

153. Wright, S. Evolution in Mendelian Populations. *Genetics* **16**, 97–159 (1931).

154. Wright, S. The Shifting Balance Theory and Macroevolution. *Annu. Rev. Genet.* **16**, 1–20 (1982).

155. Durrant, J. D., Amaro, R. E. & McCammon, J. A. AutoGrow: A Novel Algorithm for Protein Inhibitor Design. *Chem. Biol. Drug Des.* **73**, 168–178 (2009).

156. Durrant, J. D., Lindert, S., Andrew McCammon, J. & McCammon, J. A. AutoGrow 3.0: An improved algorithm for chemically tractable, semi-automated protein inhibitor design. *J. Mol. Graph. Model.* **44**, 104–112 (2013).

157. Sliwoski, G., Kothiwale, S., Meiler, J. & Lowe, E. W. Computational methods in drug discovery. *Pharmacological Reviews* (2014). doi:10.1124/pr.112.007336

158. Degen, J., Wegscheid-Gerlach, C., Zaliani, A. & Rarey, M. On the art of compiling and using 'drug-like' chemical fragment spaces. *ChemMedChem* (2008). doi:10.1002/cmdc.200800178

159. Yu, W. & Mackerell, A. D. Computer-aided drug design methods. in *Methods in Molecular Biology* **1520**, 85–106 (2017).

160. Kawai, K., Nagata, N. & Takahashi, Y. De novo design of drug-like molecules by a fragment-based molecular evolutionary approach. *J. Chem. Inf. Model.* **54**, 49–56 (2014).

161. Ferreira, L. G., Dos Santos, R. N., Oliva, G. & Andricopulo, A. D. *Molecular docking and structure-based drug design strategies*. *Molecules* **20**, (2015).

162. Allen, W. J., Fochtman, B. C., Balius, T. E. & Rizzo, R. C. Customizable de novo design strategies for DOCK: Application to HIVgp41 and other therapeutic targets. *J. Comput. Chem.* **38**, 2641–2663 (2017).

163. Lipinski, C. A., Lombardo, F., Dominy, B. W. & Feeney, P. J. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Adv. Drug Deliv. Rev.* **23**, 3–25 (1997).

164. O'Boyle, N. M. *et al.* Open Babel: An Open chemical toolbox. *J. Cheminform.* (2011). doi:10.1186/1758-2946-3-33

165. Landrum, G. RDKit: Open-source cheminformatics; http://www.rdkit.org. (2017).

166. Yang, Z. & Yang, S. Scopy: An integrated negative design python library for desirable HTS/VS database design. (2020). doi:10.5281/zenodo.3841226

167. Dey, F. & Caflisch, A. Fragment-based de novo ligand design by multi-objective evolutionary optimization. *J. Chem. Inf. Model.* **48**, 679–690 (2008).

168. Ghose, A. K., Viswanadhan, V. N. & Wendoloski, J. J. A knowledge-based approach in designing combinatorial or medicinal chemistry libraries for drug discovery. *J. Comb. Chem.* **1**, 55–68 (1999).

169. Van De Waterbeemd, H., Camenisch, G., Folkers, G., Chretien, J. R. & Raevsky, O. A. Estimation of blood-brain barrier crossing of drugs using molecular size and shape, and H-bonding descriptors. *J. Drug Target.* (1998). doi:10.3109/10611869808997889

170. Mozziconacci, J. C. Preparation of a Molecular Database from a Set of 2 Million Compounds for Virtual Screening Applications: Gathering, Structural Analysis and Filtering. in *9th Electronic Computational Chemistry Conference* (2003).

171. Brenk, R. *et al.* Lessons learnt from assembling screening libraries for drug discovery for neglected diseases. *ChemMedChem* **3**, 435–444 (2008).

172. Jadhav, A. *et al.* Quantitative analyses of aggregation, autofluorescence, and reactivity artifacts in a screen for inhibitors of a thiol protease. *J. Med. Chem.* **53**, 37–51 (2010).

173. Doveston, R. G. *et al.* A unified lead-oriented synthesis of over fifty molecular scaffolds. *Org. Biomol. Chem.* **13**, 859–865 (2015).

174. Baell, J. B. & Holloway, G. A. New Substructure Filters for Removal of Pan Assay Interference Compounds (PAINS) from Screening Libraries and for Their Exclusion in Bioassays. *J. Med. Chem.* **53**, 2719–2740 (2010).

175. Lombardo, F. *et al.* In Silico Absorption, Distribution, Metabolism, Excretion, and Pharmacokinetics (ADME-PK): Utility and Best Practices. An Industry Perspective from the International Consortium for Innovation through Quality in Pharmaceutical Development. *J. Med. Chem.* **60**, 9097–9113 (2017).

176. Tibbitts, J., Canter, D., Graff, R., Smith, A. & Khawli, L. A. Key factors influencing ADME properties of therapeutic proteins: A need for ADME characterization in drug discovery and development. *mAbs* (2016). doi:10.1080/19420862.2015.1115937

177. Krüger, A., Gonçalves Maltarollo, V., Wrenger, C. & Kronenberger, T. ADME Profiling in Drug Discovery and a New Path Paved on Silica. in *Drug Discovery and Development - New Advances* (2020). doi:10.5772/intechopen.86174

178. Shanmugasundaram, V. & Maggiora, G. M. Application of Shannon-like diversity measures to cell-based chemistry spaces. *J. Math. Chem.* (2011). doi:10.1007/s10910-010-9745-8

179. Rafiq, Z. Characterization of a High Efflux MDR Klebsiella pneumoniae Isolate and Bioactivity Guided Purification of a Novel Pyttolidine With Efflux Inhibition Activity. (Dr. M.G.R. Educational and Research Institute, 2016).

180. Boccitto, M. Genetic and Pharmacological Approaches to Preventing Neurodegeneration. (University of Pennsylvania, 2012).

181. Xu, W., Lucke, A. J. & Fairlie, D. P. Comparing sixteen scoring functions for predicting biological activities of ligands for protein targets. *J. Mol. Graph. Model.* **57**, 76–88 (2015).

182. Li, J., Fu, A. & Zhang, L. An Overview of Scoring Functions Used for Protein–Ligand Interactions in Molecular Docking. *Interdiscip. Sci. Comput. Life Sci.* **11**, 320–328 (2019).

183. Ripphausen, P., Nisius, B., Peltason, L. & Bajorath, J. Quo Vadis, Virtual Screening? A Comprehensive Survey of Prospective Applications. *J. Med. Chem.* **53**, 8461–8467 (2010).

184. Chang, M. W., Ayeni, C., Breuer, S. & Torbett, B. E. Virtual screening for HIV protease inhibitors: A comparison of AutoDock 4 and Vina. *PLoS One* (2010). doi:10.1371/journal.pone.0011955

185. Trott, O. & Olson, A. J. AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *J. Comput. Chem.* **31**, NA-NA (2009).

186. Fan, J., Fu, A. & Zhang, L. Progress in molecular docking. *Quant. Biol.* **7**, 83–89 (2019).

187. Allen, W. J. *et al.* DOCK 6: Impact of new features and current docking performance. *J. Comput. Chem.* **36**, 1132–1156 (2015).

188. Halgren, T. A. *et al.* Glide: A New Approach for Rapid, Accurate Docking and Scoring. 2. Enrichment Factors in Database Screening. *J. Med. Chem.* (2004). doi:10.1021/jm030644s

189. Friesner, R. A. *et al.* Glide: A New Approach for Rapid, Accurate Docking and Scoring. 1. Method and Assessment of Docking Accuracy. *J. Med. Chem.* (2004). doi:10.1021/jm0306430

190. Salmaso, V. & Moro, S. Bridging Molecular Docking to Molecular Dynamics in Exploring Ligand-Protein Recognition Process: An Overview. *Front. Pharmacol.* **9**, 1–16 (2018).

191. Jones, G., Willett, P., Glen, R. C., Leach, A. R. & Taylor, R. Development and validation of a genetic algorithm for flexible docking. *J. Mol. Biol.* (1997). doi:10.1006/jmbi.1996.0897

192. Verdonk, M. L., Cole, J. C., Hartshorn, M. J., Murray, C. W. & Taylor, R. D. Improved protein-ligand docking using GOLD. *Proteins Struct. Funct. Genet.* (2003). doi:10.1002/prot.10465

193. Korb, O., Stützle, T. & Exner, T. E. An ant colony optimization approach to flexible protein–ligand docking. *Swarm Intell.* (2007). doi:10.1007/s11721-007-0006-9

194. Korb, O., Stützle, T. & Exner, T. E. PLANTS: Application of ant colony optimization to structure-based drug design. in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2006). doi:10.1007/11839088_22

195. Korb, O., Stützle, T. & Exner, T. E. Empirical scoring functions for advanced Protein-Ligand docking with PLANTS. *J. Chem. Inf. Model.* (2009). doi:10.1021/ci800298z

196. Rawal, K. *et al.* An extensive survey of molecular docking tools and their applications using text mining and deep curation strategies. *PeerJ Prepr.* (2019).

doi:10.7287/peerj.preprints.27538v1

197. Kramer, B., Rarey, M. & Lengauer, T. Evaluation of the FLEXX incremental construction algorithm for protein-ligand docking. *Proteins Struct. Funct. Genet.* **37**, 228–241 (1999).

198. Fischer, E. Einfluss der Configuration auf die Wirkung der Enzyme. *Berichte der Dtsch. Chem. Gesellschaft* **27**, 3479–3483 (1894).

199. Csermely, P., Palotai, R. & Nussinov, R. Induced fit, conformational selection and independent dynamic segments: an extended view of binding events. *Trends Biochem. Sci.* **35**, 539–546 (2010).

200. Koshland, D. E. Application of a Theory of Enzyme Specificity to Protein Synthesis. *Proc. Natl. Acad. Sci.* **44**, 98–104 (1958).

201. Monod, J., Wyman, J. & Changeux, J. P. On the nature of allosteric transitions: A plausible model. *J. Mol. Biol.* **12**, 88–118 (1965).

202. Frauenfelder, H., Sligar, S. & Wolynes, P. The energy landscapes and motions of proteins. *Science (80-.).* **254**, 1598–1603 (1991).

203. Qiao, W. *et al.* From mutations to mechanisms and dysfunction via computation and mining of protein energy landscapes. *BMC Genomics* **19**, 671 (2018).

204. Kumar, S., Ma, B., Tsai, C.-J., Sinha, N. & Nussinov, R. Folding and binding cascades: Dynamic landscapes and population shifts. *Protein Sci.* **9**, 10–19 (2000).

205. Alhossary, A., Handoko, S. D., Mu, Y. & Kwoh, C. K. Fast, accurate, and reliable molecular docking with QuickVina 2. *Bioinformatics* **31**, 2214–2216 (2015).

206. Brooijmans, N. & Kuntz, I. D. Molecular Recognition and Docking Algorithms. *Annu. Rev. Biophys. Biomol. Struct.* **32**, 335–373 (2003).

207. Hassan, N. M., Alhossary, A. A., Mu, Y. & Kwoh, C. K. Protein-Ligand Blind Docking Using QuickVina-W with Inter-Process Spatio-Temporal Integration. *Sci. Rep.* **7**, 1–13

(2017).

208. Amaro, R. E. *et al.* Ensemble Docking in Drug Discovery. *Biophys. J.* **114**, 2271–2278 (2018).

209. De Paris, R., Vahl Quevedo, C., Ruiz, D. D., Gargano, F. & de Souza, O. N. A selective method for optimizing ensemble docking-based experiments on an InhA Fully-Flexible receptor model. *BMC Bioinformatics* **19**, 235 (2018).

210. Huang, S.-Y., Grinter, S. Z. & Zou, X. Scoring functions and their evaluation methods for protein–ligand docking: recent advances and future directions. *Phys. Chem. Chem. Phys.* **12**, 12899 (2010).

211. Chu, Y. & He, X. Molegear: A Java-based platform for evolutionary de novo molecular design. *Molecules* **24**, 1444 (2019).

212. Olivecrona, M., Blaschke, T., Engkvist, O. & Chen, H. Molecular de-novo design through deep reinforcement learning. *J. Cheminform.* **9**, 48 (2017).

213. Skalic, M., Jiménez, J., Sabbadin, D. & De Fabritiis, G. Shape-Based Generative Modeling for de Novo Drug Design. *J. Chem. Inf. Model.* **59**, 1205–1214 (2019).

214. Ropp, P. J. *et al.* Gypsum-DL: an open-source program for preparing small-molecule libraries for structure-based virtual screening. *J. Cheminform.* **11**, 34 (2019).

215. Morris, G. M. *et al.* AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility. *J. Comput. Chem.* (2009). doi:10.1002/jcc.21256

216. Kolb, P. & Caflisch, A. Automatic and efficient decomposition of two-dimensional structures of small molecules for fragment-based high-throughput docking. *J. Med. Chem.* (2006). doi:10.1021/jm060838i

217. Majeux, N., Scarsi, M., Apostolakis, J., Ehrhardt, C. & Caflisch, A. Exhaustive docking of molecular fragments with electrostatic solvation. *Proteins Struct. Funct. Genet.* (1999). doi:10.1002/(SICI)1097-0134(19991001)37:1<88::AID-PROT9>3.0.CO;2-O

218. Budin, N., Majeux, N. & Caflisch, A. Fragment-based flexible ligand docking by evolutionary optimization. *Biol. Chem.* (2001). doi:10.1515/BC.2001.168

219. Pettersen, E. F. *et al.* UCSF Chimera - A visualization system for exploratory research and analysis. *J. Comput. Chem.* **25**, 1605–1612 (2004).

220. Humphrey, W., Dalke, A. & Schulten, K. VMD: Visual molecular dynamics. *J. Mol. Graph.* (1996). doi:10.1016/0263-7855(96)00018-5

221. Vanommeslaeghe, K. & MacKerell, A. D. Automation of the CHARMM general force field (CGenFF) I: Bond perception and atom typing. *J. Chem. Inf. Model.* (2012). doi:10.1021/ci300363c

222. Yu, W., He, X., Vanommeslaeghe, K. & MacKerell, A. D. Extension of the CHARMM general force field to sulfonyl-containing compounds and its utility in biomolecular simulations. *J. Comput. Chem.* (2012). doi:10.1002/jcc.23067

223. Vanommeslaeghe, K., Raman, E. P. & MacKerell, A. D. Automation of the CHARMM General Force Field (CGenFF) II: Assignment of Bonded Parameters and Partial Atomic Charges. *J. Chem. Inf. Model.* (2012). doi:10.1021/ci3003649

224. Baptista, S. J. *et al.* Novel PARP-1 Inhibitor scaffolds disclosed by a dynamic structure-based pharmacophore approach. *PLoS One* **12**, 1–20 (2017).

225. Lindert, S., Durrant, J. D. & Mccammon, J. A. LigMerge: A Fast Algorithm to Generate Models of Novel Potential Ligands from Sets of Known Binders. *Chem. Biol. Drug Des.* **80**, 358–365 (2012).

226. Durrant, J. D. & McCammon, J. A. Autoclickchem: Click chemistry in silico. *PLoS Comput. Biol.* **8**, (2012).

227. Kolb, H. C., Finn, M. G. & Sharpless, K. B. Click Chemistry: Diverse Chemical Function from a Few Good Reactions. *Angewandte Chemie - International Edition* (2001). doi:10.1002/1521-3773(20010601)40:11<2004::AID-ANIE2004>3.0.CO;2-5

228. Hartenfeller, M. *et al.* A collection of robust organic synthesis reactions for in silico

molecule design. *J. Chem. Inf. Model.* **51**, 3093–3098 (2011).

229. Robinson, B. The Fischer Indole Synthesis. *Chem. Rev.* **63**, 373–401 (1963).

230. Sterling, T. & Irwin, J. J. ZINC 15 - Ligand Discovery for Everyone. *J. Chem. Inf. Model.* (2015). doi:10.1021/acs.jcim.5b00559

231. Ropp, P. J., Kaminsky, J. C., Yablonski, S. & Durrant, J. D. Dimorphite-DL: An open-source program for enumerating the ionization states of drug-like small molecules. *J. Cheminform.* **11**, 1–8 (2019).

232. Schrödinger, L. Glide. (2009).

233. Durrant, J. D. & McCammon, J. A. NNScore: A neural-network-based scoring function for the characterization of protein-ligand complexes. *J. Chem. Inf. Model.* **50**, 1865–1871 (2010).

234. Durrant, J. D. & McCammon, J. A. NNScore 2.0: A neural-network receptor-ligand scoring function. *J. Chem. Inf. Model.* **51**, 2897–2903 (2011).

235. Reynolds, C. H., Tounge, B. A. & Bembenek, S. D. Ligand binding efficiency: Trends, physical basis, and implications. *J. Med. Chem.* **51**, 2432–2438 (2008).

236. Rogers, D. & Hahn, M. Extended-connectivity fingerprints. *J. Chem. Inf. Model.* (2010). doi:10.1021/ci100050t

237. Landrum, G. Getting Started with the RDKit in Python. *Manual* (2011). Available at: https://www.rdkit.org/docs/GettingStartedInPython.html.

238. Bajusz, D., Rácz, A. & Héberger, K. Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations? *J. Cheminform.* **7**, (2015).

239. Berman, H. M. *et al.* The protein data bank. *Acta Crystallogr. Sect. D Biol. Crystallogr.* (2002). doi:10.4135/9781412994231.n75

240. Dolinsky, T. J., Nielsen, J. E., McCammon, J. A. & Baker, N. A. PDB2PQR: An automated pipeline for the setup of Poisson-Boltzmann electrostatics calculations. *Nucleic Acids Res.* (2004). doi:10.1093/nar/gkh381

241. Dolinsky, T. J. *et al.* PDB2PQR: Expanding and upgrading automated preparation of biomolecular structures for molecular simulations. *Nucleic Acids Res.* **35**, (2007).

242. Ropp, P., Friedman, A. & Durrant, J. D. Scoria: A Python module for manipulating 3D molecular data. *J. Cheminform.* **9**, (2017).

243. Ngan, C. H. *et al.* FTMAP: Extended protein mapping with user-selected probe molecules. *Nucleic Acids Res.* **40**, 271–275 (2012).

244. Makley, L. N. & Gestwicki, J. E. Expanding the Number of 'Druggable' Targets: Non-Enzymes and Protein-Protein Interactions. *Chem. Biol. Drug Des.* (2013). doi:10.1111/cbdd.12066

245. Durrant, J. D. Blendmol: Advanced macromolecular visualization in Blender. *Bioinformatics* **35**, 2323–2325 (2019).

246. Durrant, J. D. & McCammon, J. A. BINANA: A novel algorithm for ligand-binding characterization. *J. Mol. Graph. Model.* **29**, 888–893 (2011).

247. Warnecke, A., Sandalova, T., Achour, A. & Harris, R. A. PyTMs: A useful PyMOL plugin for modeling common post-translational modifications. *BMC Bioinformatics* (2014). doi:10.1186/s12859-014-0370-6

248. Durrant, J. D., Votapka, L., Sørensen, J. & Amaro, R. E. POVME 2.0: An enhanced tool for determining pocket shape and volume characteristics. *J. Chem. Theory Comput.* (2014). doi:10.1021/ct500381c

249. Volkamer, A., Kuhn, D., Rippmann, F. & Rarey, M. DoGSiteScorer: a web server for automatic binding site prediction, analysis and druggability assessment. *Bioinformatics* **28**, 2074–2075 (2012).

250. Lindgren, A. E. G. *et al.* PARP inhibitor with selectivity toward ADP-ribosyltransferase

ARTD3/PARP3. *ACS Chem. Biol.* **8**, 1698–1703 (2013).

251. Fu, L. *et al.* Crystal structure-based discovery of a novel synthesized PARP1 inhibitor (OL-1) with apoptosis-inducing mechanisms in triple-negative breast cancer. *Sci. Rep.* **6**, (2016).

252. Wahlberg, E. *et al.* Family-wide chemical profiling and structural analysis of PARP and tankyrase inhibitors. *Nat. Biotechnol.* **30**, 283–288 (2012).

253. Upton, K. *et al.* Design and synthesis of potent inhibitors of the mono(ADP-ribosyl)transferase, PARP14. *Bioorganic Med. Chem. Lett.* **27**, 2907–2911 (2017).

254. Jagtap, P. G. *et al.* The discovery and synthesis of novel adenosine substituted 2,3-dihydro-1H-isoindol-1-ones: Potent inhibitors of poly(ADP-ribose) polymerase-1 (PARP-1). *Bioorganic Med. Chem. Lett.* **14**, 81–85 (2004).

255. Jagtap, P. & Szabo, C. Poly(ADP-ribose) polymerase and the therapeutic effects of its inhibitors. *Nature Reviews Drug Discovery* **4**, 421–440 (2005).

256. Fry, D. C. Targeting Protein-Protein Interactions for Drug Discovery. in *Protein-Protein Interactions: Methods and Applications* (eds. Meyerkord, C. L. & Fu, H.) 93–106 (Springer New York, 2015). doi:10.1007/978-1-4939-2425-7_6

257. Loving, K. A., Lin, A. & Cheng, A. C. Structure-Based Druggability Assessment of the Mammalian Structural Proteome with Inclusion of Light Protein Flexibility. *PLoS Comput. Biol.* (2014). doi:10.1371/journal.pcbi.1003741

258. Alvesalo, J. K. O., Siiskonen, A., Vainio, M. J., Tammela, P. S. M. & Vuorela, P. M. Similarity Based Virtual Screening: A Tool for Targeted Library Design. *J. Med. Chem.* **49**, 2353–2356 (2006).

259. de Souza Neto, L. R. *et al.* In silico Strategies to Support Fragment-to-Lead Optimization in Drug Discovery. *Front. Chem.* **8**, 1–18 (2020).

260. Wang, Y. *et al.* PubChem: a public information system for analyzing bioactivities of small molecules. *Nucleic Acids Res.* **37**, W623–W633 (2009).

261. Chen, X., Lin, Y., Liu, M. & Gilson, M. K. The Binding Database: data management and interface design. *Bioinformatics* **18**, 130–139 (2002).

262. Liu, T., Lin, Y., Wen, X., Jorissen, R. N. & Gilson, M. K. BindingDB: A web-accessible database of experimentally determined protein-ligand binding affinities. *Nucleic Acids Res.* **35**, D198–D201 (2007).

263. Butina, D. Unsupervised data base clustering based on daylight's fingerprint and Tanimoto similarity: A fast and automated way to cluster small and large data sets. *J. Chem. Inf. Comput. Sci.* (1999). doi:10.1021/ci9803381

264. DesJarlais, R. L. *et al.* Using Shape Complementarity as an Initial Screen in Designing Ligands for a Receptor Binding Site of Known Three-Dimensional Structure. *J. Med. Chem.* (1988). doi:10.1021/jm00399a006

265. Meng, E. C., Shoichet, B. K. & Kuntz, I. D. Automated docking with grid-based energy evaluation. *J. Comput. Chem.* (1992). doi:10.1002/jcc.540130412

266. Ertl, P. & Schuffenhauer, A. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *J. Cheminform.* (2009). doi:10.1186/1758-2946-1-8

267. Bickerton, G. R., Paolini, G. V., Besnard, J., Muresan, S. & Hopkins, A. L. Quantifying the chemical beauty of drugs. *Nat. Chem.* (2012). doi:10.1038/nchem.1243

268. Kalgutkar, A. S. & Soglia, J. R. Minimising the potential for metabolic activation in drug discovery. *Expert Opin. Drug Metab. Toxicol.* **1**, 91–142 (2005).

269. Non MedChem-Friendly SMARTS. Available at: https://www.surechembl.org/knowledgebase/169485. (Accessed: 20th June 2020)

270. Sushko, I., Salmina, E., Potemkin, V. A., Poda, G. & Tetko, I. V. ToxAlerts: A Web Server of Structural Alerts for Toxic Chemicals and Compounds with Potential Adverse Reactions. *J. Chem. Inf. Model.* **52**, 2310–2316 (2012).

271. Çınaroğlu, S. S. & Timuçin, E. Comparative Assessment of Seven Docking Programs on a

Nonredundant Metalloprotein Subset of the PDBbind Refined. *J. Chem. Inf. Model.* **59**, 3846–3859 (2019).

272. Barber, L. J. *et al.* Secondary mutations in BRCA2 associated with clinical resistance to a PARP inhibitor. *J. Pathol.* (2013). doi:10.1002/path.4140

273. Zuckerman, D. M. & Chong, L. T. Weighted Ensemble Simulation: Review of Methodology, Applications, and Software. *Annu. Rev. Biophys.* **46**, 43–57 (2017).

274. Ryde, U. & Söderhjelm, P. Ligand-Binding Affinity Estimates Supported by Quantum-Mechanical Methods. *Chem. Rev.* (2016). doi:10.1021/acs.chemrev.5b00630

275. Navis, A. C. *et al.* Identification of a novel MET mutation in high-grade glioma resulting in an auto-active intracellular protein. *Acta Neuropathol.* (2015). doi:10.1007/s00401-015-1420-5

276. Putt, K. S. & Hergenrother, P. J. An enzymatic assay for poly(ADP-ribose) polymerase-1 (PARP-1) via the chemical quantitation of NAD+: Application to the high-throughput screening of small molecules as potential inhibitors. *Anal. Biochem.* (2004). doi:10.1016/j.ab.2003.11.015

277. Riss, T. L. *et al. Cell Viability Assays*. *Assay Guidance Manual* (2004).

278. Knegtel, R. M. A., Kuntz, I. D. & Oshiro, C. M. Molecular docking to ensembles of protein structures. *J. Mol. Biol.* (1997). doi:10.1006/jmbi.1996.0776

279. Totrov, M. & Abagyan, R. Flexible ligand docking to multiple receptor conformations: a practical alternative. *Current Opinion in Structural Biology* (2008). doi:10.1016/j.sbi.2008.01.004

280. Doig, A. J. Frozen, but no accident – why the 20 standard amino acids were selected. *FEBS Journal* (2017). doi:10.1111/febs.13982

281. Liu, K. & Kokubo, H. Exploring the Stability of Ligand Binding Modes to Proteins by Molecular Dynamics Simulations: A Cross-docking Study. *J. Chem. Inf. Model.* (2017). doi:10.1021/acs.jcim.7b00412

282. Perez, A., Morrone, J. A., Simmerling, C. & Dill, K. A. Advances in free-energy-based simulations of protein folding and ligand binding. *Current Opinion in Structural Biology* (2016). doi:10.1016/j.sbi.2015.12.002

283. Iakovou, G., Hayward, S. & Laycock, S. D. Adaptive GPU-accelerated force calculation for interactive rigid molecular docking using haptics. *J. Mol. Graph. Model.* (2015). doi:10.1016/j.jmgm.2015.06.003

284. Ascher, D.; Dubois, P. F.; Hinsen, K.; James, J. H.; Oliphant, T. Numerical Python, UCRL-MA-128569; *Lawrence Livermore Natl. Lab. Livermore, CA,* (1991).

285. Oliphant, T. & Millma, J. k. A guide to NumPy. *Trelgol Publishing* (2006). doi:DOI:10.1109/MCSE.2007.58

286. Jones, E., Oliphant, T., Peterson, P. & Others. SciPy: Open Source Scientific Tools for Python, 2001 (http://www.scipy.org/). *Http://Www.Scipy.Org/* (2015).

287. Hunter, J. D. Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* (2007). doi:10.1109/MCSE.2007.55

288. Dalcín, L., Paz, R., Storti, M. & D'Elía, J. MPI for Python: Performance improvements and MPI-2 extensions. *J. Parallel Distrib. Comput.* (2008). doi:10.1016/j.jpdc.2007.09.005

289. Tanrikulu, Y., Krüger, B. & Proschak, E. The holistic integration of virtual screening in drug discovery. *Drug Discovery Today* (2013). doi:10.1016/j.drudis.2013.01.007

290. Lionta, E., Spyrou, G., Vassilatis, D. & Cournia, Z. Structure-Based Virtual Screening for Drug Discovery: Principles, Applications and Recent Advances. *Curr. Top. Med. Chem.* (2014). doi:10.2174/1568026614666140929124445

291. Riniker, S. & Landrum, G. A. Better Informed Distance Geometry: Using What We Know to Improve Conformation Generation. *J. Chem. Inf. Model.* (2015). doi:10.1021/acs.jcim.5b00654

292. Rappé, A. K., Casewit, C. J., Colwell, K. S., Goddard, W. A. & Skiff, W. M. UFF, a Full Periodic Table Force Field for Molecular Mechanics and Molecular Dynamics Simulations.

J. Am. Chem. Soc. (1992). doi:10.1021/ja00051a040

293. Hartigan, J. A. & Wong, M. A. Algorithm AS 136: A K-Means Clustering Algorithm. *Appl. Stat.* (1979). doi:10.2307/2346830

294. Wang, R., Fang, X., Lu, Y. & Wang, S. The PDBbind database: Collection of binding affinities for protein-ligand complexes with known three-dimensional structures. *J. Med. Chem.* (2004). doi:10.1021/jm030580l

295. Wang, R., Fang, X., Lu, Y., Yang, C.-Y. & Wang, S. The PDBbind Database: Methodologies and Updates. *J. Med. Chem.* **48**, 4111–4119 (2005).

296. Dalcin, L. D., Paz, R. R., Kler, P. A. & Cosimo, A. Parallel distributed computing using Python. *Adv. Water Resour.* (2011). doi:10.1016/j.advwatres.2011.04.013

297. Dalcín, L., Paz, R. & Storti, M. MPI for Python. *J. Parallel Distrib. Comput.* (2005). doi:10.1016/j.jpdc.2005.03.010

298. Hawkins, P. C. D. & Nicholls, A. Conformer generation with OMEGA: Learning from the data set and the analysis of failures. *J. Chem. Inf. Model.* (2012). doi:10.1021/ci300314k

299. Hawkins, P. C. D., Skillman, A. G., Warren, G. L., Ellingson, B. A. & Stahl, M. T. Conformer generation with OMEGA: Algorithm and validation using high quality structures from the protein databank and cambridge structural database. *J. Chem. Inf. Model.* (2010). doi:10.1021/ci100031x

300. Schrödinger. LigPrep | Schrödinger. *Schrödinger Release 2018-1* (2018).

301. Miteva, M. A., Guyon, F. & Tufféry, P. Frog2: Efficient 3D conformation ensemble generator for small compounds. *Nucleic Acids Res.* (2010). doi:10.1093/nar/gkq325

302. Puranen, J. S., Vainio, M. J. & Johnson, M. S. Accurate conformation-dependent molecular electrostatic potentials for high-throughput in silico drug discovery. *J. Comput. Chem.* (2010). doi:10.1002/jcc.21460

303. Vainio, M. J. & Johnson, M. S. Generating conformer ensembles using a multiobjective genetic algorithm. *J. Chem. Inf. Model.* (2007). doi:10.1021/Ci6005646