UNIVERSITY OF SOUTHAMPTON

Faculty of Earth and Physical Sciences

School of Chemistry

# Assessing synthetic difficulty in computational organic materials discovery

*by*

**Joshua Thomas Dickman**

MChem

ORCiD: 0000-0001-6125-773X

*A thesis for the degree of*

*Doctor of Philosophy*

November 2025

University of Southampton

Abstract

Faculty of Earth and Physical Sciences

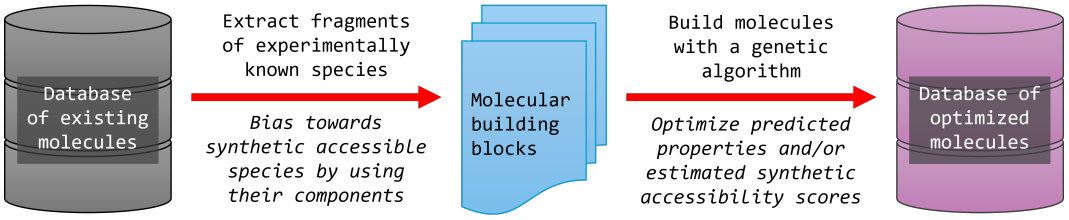
School of Chemistry

Doctor of Philosophy

**Assessing synthetic difficulty in computational organic materials discovery**

by Joshua Thomas Dickman

This thesis presents a study of computational organic materials discovery, focused on the generation and assessment of small molecule aromatic fused ring systems. This was accomplished through the use of MolBuilder, an evolutionary algorithm built to efficiently navigate chemical space by guiding molecular generation according to calculated properties; various methods to incorporate the assessment of synthetic feasibility in generated molecules are demonstrated, in particular the use of computational tools which estimate synthetic difficulty from molecular structure.




Several molecular generation campaigns were conducted with MolBuilder, aiming to optimize physical properties and incorporate a bias towards synthetically feasible candidates for applications in organic semiconducting materials, by constructing fitness functions which optimize one or more objectives as part of the molecular generation process. Top performing species sampled in this manner exhibit promising calculated reorganisation energy values while maintaining low predicted synthetic complexity, and could be suggested as targets for experimental work.

# Contents

**List of Figures** **xi**

**List of Tables** **xxi**

**List of Listings** **xxi**

**Declaration of Authorship** **xxi**

**Acknowledgements** **xxv**

**1 Introduction** **1**
1.1 Background and Motivations . . . . . 1
1.1.1 Discovering new materials . . . . . 1
1.1.2 Materials discovery aided by computation . . . . . 2
1.1.3 Predicting synthetic difficulty . . . . . 3
1.2 Thesis overview . . . . . 4
1.3 Thesis structure . . . . . 5

**2 Literature Review** **7**
2.1 Computational molecular generation . . . . . 7
2.1.1 Producing molecules with a genetic algorithm . . . . . 7
2.1.2 Reaction-based approaches to molecular generation . . . . . 9
2.2 Synthetic difficulty . . . . . 10
2.2.1 Estimating synthetic difficulty with computational tools . . . . . 11
2.2.2 Importance in materials discovery frameworks . . . . . 13
2.2.3 When to consider synthetic difficulty . . . . . 14
2.2.4 Benefits to the experimental chemists . . . . . 15
2.3 Materials discovery . . . . . 16
2.3.1 Experimental approach to materials discovery . . . . . 16
2.3.2 Benefits of the computational approach . . . . . 16
2.3.3 Considering synthesis and stability in materials discovery . . . . 17
2.3.4 Molecular targets for materials applications . . . . . 18
2.3.4.1 Small-molecule organic semiconductors . . . . . 18
2.3.4.2 Hydrogen-bonded organic frameworks (HOFs) . . . . . 20

**3 Theory Framework** **21**
3.1 Computational handling of organic chemistry . . . . . 21
3.1.1 Storing and querying organic molecules . . . . . 21

3.1.2 Molecular fingerprinting, similarity and diversity . . . . . 23
3.1.3 Representing and applying reactions . . . . . 25
3.2 Using ideas from genetics to manipulate molecules . . . . . 27
3.2.1 Genetic operations from a chemical standpoint . . . . . 28
3.2.2 Elitism and survival of the fittest . . . . . 29
3.2.3 Exerting more control over genetic algorithms . . . . . 30
3.2.4 Use in the context of materials discovery . . . . . 31
3.3 Combining known reactions and precursors . . . . . 31
3.4 Molecular modelling and simulation of the solid state . . . . . 32
3.4.1 Geometry optimization . . . . . 32
3.4.2 Crystal structure prediction . . . . . 33
3.4.3 Charge carrier mobility . . . . . 35
3.5 Synthetic difficulty estimation . . . . . 37
3.5.1 Capturing the rules of organic chemistry in computational settings 37
3.5.2 Synthetic difficulty scoring . . . . . 37
3.5.2.1 Molecular complexity . . . . . 38
3.5.2.2 Computer-aided synthesis planning . . . . . 39
3.5.2.3 Reaction network-based scoring . . . . . 41

**4 Code Development 43**
4.1 Working with molecules and reactions in Python . . . . . 43
4.1.1 Conversion between representations . . . . . 43
4.1.2 Defining building blocks . . . . . 44
4.1.3 Storing molecules . . . . . 45
4.1.4 Ordering calculation files . . . . . 45
4.1.5 Substructure searching and pattern matching . . . . . 46
4.1.6 ReactionSMARTS and RDKit reactions . . . . . 46
4.2 MolBuilder - an evolutionary algorithm for chemical space . . . . . 48
4.2.1 Setting up a genetic algorithm and storing results . . . . . 48
4.2.2 Defining building blocks . . . . . 49
4.2.3 ReactionSMARTS for genetic operations . . . . . 51
4.2.3.1 Using RDKit reactions to modify molecules . . . . . 51
4.2.3.2 Fragmenting molecules and adding attachment points . 54
4.2.3.3 Adding a ring to a molecule . . . . . 56
4.2.3.4 Mutating molecules to add or remove side groups . . . 58
4.2.3.5 Creating molecules from building blocks . . . . . 59
4.2.3.6 Performing recombinations . . . . . 60
4.2.3.7 Crossing over two parent molecules . . . . . 61
4.2.3.8 Creating the next generation of molecules . . . . . 62
4.2.4 Benefits to this approach . . . . . 63
4.2.5 Improved control over design rules . . . . . 64
4.2.5.1 Counting building blocks . . . . . 64
4.2.5.2 Smarter mutations . . . . . 65
4.2.5.3 Single parent crossover . . . . . 67
4.2.6 ”Reverse-MolBuilder” . . . . . 69
4.3 Full chemical space exploration . . . . . 70
4.3.1 Splitting molecular generation into steps . . . . . 71

4.3.2 Parallelisation and distribution of CSE . . . 72
4.3.3 Reaction product enumeration . . . 73
4.4 Synthetic difficulty prediction . . . 75
4.4.1 SYBA . . . 75
4.4.2 SCScore . . . 76
4.4.3 AIZynthFinder . . . 77
4.5 Property prediction . . . 80
4.5.1 Geometry optimization . . . 80
4.5.2 Crystal structure prediction . . . 81
4.5.3 Charge mobility calculations . . . 83
4.5.3.1 Reorganisation energy . . . 83
4.5.3.2 Electron mobility . . . 85

**5 Projects basis and setup 87**
5.1 Azapentacenes - testing space . . . 87
5.1.1 Reason for working in this space . . . 87
5.1.1.1 Past work . . . 88
5.1.1.2 Past results of exhaustive and directed exploration . . . 89
5.1.2 Updated chemical space setup . . . 90
5.2 OCELOT - organic semiconductors . . . 91
5.2.1 Original Dataset . . . 91
5.2.1.1 Filtering the dataset . . . 92
5.2.1.2 Extracting and choosing building blocks . . . 92
5.2.1.3 Final building block choices . . . 93
5.2.2 Genetic algorithm setup . . . 93
5.2.2.1 Rules on molecular generation . . . 93
5.2.2.2 Properties to optimize . . . 95
5.2.3 Selecting initial populations for genetic algorithms . . . 97
5.2.3.1 Populations with all building blocks . . . 97
5.2.3.2 Populations with reduced building block sets . . . 98
5.3 Combi-HOFs - porous frameworks . . . 102
5.3.1 Combining cores and end groups . . . 102
5.3.2 Core molecules . . . 102
5.3.3 Functionalisation reactions . . . 104
5.3.4 Generating a library of structures . . . 104
5.3.5 Next steps for the Combi-HOFs . . . 104

**6 Exhaustive chemical space exploration 105**
6.1 Azapentacenes . . . 105
6.1.1 Generating a class of similar molecules . . . 105
6.1.2 Describing and separating a class of similar molecules . . . 108
6.1.2.1 Elemental composition . . . 108
6.1.2.2 Molecular shapes and scaffolds . . . 109
6.2 OCELOT - Organic Semiconductors . . . 110
6.2.1 Exploration using reduced building block sets . . . 111
6.3 Conclusions on full chemical space exploration . . . 113

**7 Molecular reorganisation energy GAs** **115**
7.1 Behaviour of genetic algorithms . . . 115
7.1.1 General behaviour . . . 116
7.1.2 Minimizing electron reorganisation energy . . . 118
7.1.3 Convergence of GAs from different starting points . . . 121
7.2 Repeating GAs from the same starting point . . . 122
7.2.1 Divergence of runs from the same initial populations . . . 126
7.3 Behaviour when the building block set is reduced . . . 127
7.3.1 Aza-substituted fused ring systems . . . 127
7.3.2 Aza- and thio- substituted ring systems . . . 129
7.3.3 Acenes with amide/ester-type rings . . . 130
7.3.4 Varied ring types . . . 131
7.4 Looking at all the GAs together . . . 133
7.4.1 Frequently found molecules . . . 133
7.4.2 Distribution of reorganisation energies . . . 133
7.4.2.1 Comparing initial and final populations . . . 134
7.4.2.2 Comparison to known experimental molecules . . . 134
7.4.3 Common molecular shapes and substructures . . . 135
7.5 Electron mobility as a fitness function . . . 136
7.5.1 Sampling molecules with mobility genetic algorithms . . . 137
7.5.2 Convergence of electron mobility values . . . 138
7.5.3 Molecular composition and diversity of populations . . . 139
7.5.4 Overlap between separated populations . . . 139
7.5.5 Comparing mobility to reorganisation energy . . . 140
7.6 Conclusions after property-led genetic algorithms . . . 141

**8 Post-hoc synthetic difficulty prediction** **143**
8.1 Azapentacenes - similar molecules . . . 143
8.1.1 Distribution of synthetic difficulty scores . . . 144
8.1.1.1 SYBA . . . 145
8.1.1.2 SCScore . . . 146
8.1.1.3 Analysing clusters . . . 147
8.1.1.4 Comparing SYBA and SCScore . . . 148
8.1.2 Retrosynthetic analysis with AiZynthFinder . . . 149
8.1.3 Experimentally known molecules . . . 150
8.2 OCELOT - reorganisation energy GA results . . . 151
8.2.1 SYBA score distributions . . . 151
8.2.2 SCScore distributions . . . 152
8.2.3 Restriction on building blocks to lower synthetic difficulty . . . 154
8.2.4 How do SYBA and SCScore match up? . . . 155
8.2.5 Comparison to the experimental set . . . 156
8.2.6 Retrosynthetic analysis with AiZynthFinder . . . 158
8.2.7 Accessible molecules with low reorganisation energies . . . 159
8.3 Conclusions from standalone synthetic difficulty analysis . . . 161

**9 Synthetic difficulty GAs** **163**
9.1 Behaviour of SCScore-led genetic algorithms . . . 163

9.1.1 Changes to diversity and molecular composition . . . 165
9.1.2 Reducing the number of building blocks . . . 165
9.2 Behaviour of SYBA-led genetic algorithms . . . 166
9.2.1 Maximization of SYBA scores . . . 167
9.2.2 Molecular composition and population diversity . . . 167
9.2.3 Reducing the number of building blocks . . . 168
9.3 Analysis of all molecules sampled with synthetic difficulty GAs . . . 169
9.3.1 Common motifs in 'synthetically accessible' molecules . . . 170
9.3.2 Synthetically accessible molecules with low reorganisation energies 170
9.4 Conclusions from synthetic difficulty-led GAs . . . 171

**10 Multi-objective GAs 173**
10.1 Behaviour of multi-objective genetic algorithms . . . 173
10.1.1 Optimizing reorganisation energy and SCScore . . . 174
10.1.2 Varying property weights . . . 176
10.2 Top performing molecules . . . 178
10.2.1 Overlap with single-objective GAs . . . 178
10.3 Conclusions from multi-objective genetic algorithms . . . 179

**11 Mixed crystals project 181**
11.1 Binary mixed crystal systems . . . 181
11.2 Validation of single-species predicted crystal structures . . . 182
11.2.1 Comparing structures between species . . . 182
11.3 Creating binary mixed crystal supercells . . . 183
11.3.1 Dealing with configurational entropy . . . 184
11.3.2 Optimizing a mixed crystal supercell . . . 184
11.4 Analysing mixed crystals with varying imposter content . . . 186
11.5 Comparison to experimental findings . . . 187

**12 Conclusions 189**
12.1 Key findings . . . 189
12.2 Future work . . . 192
12.3 Final remarks . . . 193

**Appendix A OCELOT genetic algorithms 195**
Appendix A.1 Initial populations . . . 195
Appendix A.1.1 Full building block sets . . . 195
Appendix A.1.2 Reduced building block sets . . . 216
Appendix A.2 Top performers . . . 233
Appendix A.3 Additional plots . . . 234
Appendix A.3.1 Reorganisation energy - reduced populations . . . 234
Appendix A.4 Synthetic difficulty analysis . . . 234
Appendix A.5 Synthetic difficulty genetic algorithms . . . 239

**References 241**

# List of Figures

2.1 Flowchart describing the evolutionary algorithm process for generating molecules with optimized properties. . . . . 8
2.2 Two molecules, T2 and P2, discovered through computational efforts; one synthesisable, and one not. Figure adapted from the work of Pulido et al. (2017). . . . . 13
2.3 Three approaches two implementing synthetic difficulty consideration within the molecular generation process; based on Figure 1 from the work of Gao and Coley (2020). . . . . 14
2.4 4Cl-TAP, a chlorine-substituted azapentacene molecule discovered by Chu et al. (2018) which exhibits remarkably high electron mobility values. 19
2.5 Diagram adapted from the HOF mini-review by Hisaki et al. (2019); examples of urea as a hydrogen bonding 'supramolecular synthon', which appears in 'T2' (from the work of Pulido et al. (2017)), to form porous frameworks (experimentally observed by Mastalerz and Oppel (2012)). . 20

3.1 Showcasing the human-readability of SMILES strings - alkyne bonds are easily noticed by presence of '#', while aromatic rings and connectivity of atoms can be noted by lowercase atom symbols, and numbers which specify which atoms connect to which. . . . . 22
3.2 Simple example of a reaction mapped and represented using reactionSMARTS language . . . . . 26
3.3 Flowchart depicting the general ideas behind genetic algorithms, and how they can be used with a fitness function to produce optimized populations of molecules . . . . . 27
3.4 The molecular modification operations, or genetic operators, used within MolBuilder to generate molecules . . . . . 30
3.5 Example CSP results for dibenzothiophene, from a collaborative study with Villeneuve et al. (2022); plotting the energy and density of each generated crystal structure give a good summary of the species' crystal landscape. . . . . 34
3.6 Example synthetic pathway to an aza-substituted pentacene molecule, predicted with AiZynthFinder . . . . . 40
3.7 Diagram from the work of Coley et al. (2018) to elaborate on the idea which drives SCScores; as molecules become more similar to the 'products', deeper into a reaction network, scores become larger. . . . . 42

4.1 Showcasing the ability to convert between molecule representations, with RDKit `Mol` objects acting as a bridge between formats. . . . . 44
4.2 Utility of canonical SMILES generation to avoid duplicate molecules. . . 45

4.3 Example usage of reactionSMARTS; two unconnected carbon atoms with 'attachment points' can be labelled (carbon 1 and carbon 2), then attached, using careful definition of the reactants and products in a reactionSMARTS string. . . . 47
4.4 Running reactions with RDKit - after running the reaction, all outcomes including 'duplicate' products related by symmetry are returned. These outcomes can be filtered by conversion to InChI, and checking for duplicated InChI strings. . . . 48
4.5 The function used to generate fragments from a ring building block . . . 50
4.6 Example reactionSMARTS string, encoding the 'start point', 'agents' needed to facilitate the reaction, and the 'end point' . . . 51
4.7 The importance of atom mapping in an exemplar single-point recombination . . . 52
4.8 Atom mapping in reactionSMARTS for an exemplar two-point addition 53
4.9 Diagram representing the function `fragmentation_retention()` on an example molecule . . . 55
4.10 Diagram representing the function `fragmentation()` on an example molecule 55
4.11 Diagram representing the function `addition()` with an example molecule and fragment pair . . . 56
4.12 Diagram representing the `mutation()` function with an example molecule. Random mutations are chosen from options defined in the configuration file. . . . 59
4.13 Flowchart describing the process of generating a population of molecules, given a set of ring types and side groups. . . . 60
4.14 Diagram representing the function `recombination()` on an example molecule 60
4.15 Diagram representing the function `crossover()` with an example pair of 'parent' molecules, producing two 'child' molecules. . . . 61
4.16 Diagram representing the specialised reactionSMARTS pattern used to combine two fragments with preserved attachment points, including an example depicting the possible outcomes. . . . 62
4.17 Diagram showing the process by which two 'child' molecules are created from two 'parent' molecules. . . . 63
4.18 Cases where two ring building block types can be detected by when searching for another ring type, which appears as a substructure. . . . 65
4.19 Flowchart representing the process undertaken by 'smarter' mutation operations . . . 66
4.20 Flowchart representing the process undertaken during 'single parent crossover' 68
4.21 Overall process followed by the new Reverse MolBuilder tool; molecules are iteratively fragmented, all ring/side group types needed to re-produce the species with MolBuilder are identified if possible.. . . . 69
4.22 Example molecule fragmented with the reverse MolBuilder tool, along with the outputted fragment and mutation types. . . . 69
4.23 Two workflows for exhaustive chemical space exploration, chosen dependant on the number of potential unmutated molecular 'backbones' . 71
4.24 The workflow used to generate a full library of reaction outcomes, given a list of reagents and reactions, including unreacted and partially reacted species. . . . 75

4.25 The Jupyter Notebook GUI available to run AiZynthFinder analysis given a SMILES string. . . . . 78
4.26 The general approach used to generate landscapes of predicted crystal structures, from an article by Day (2011). . . . . 83

5.1 Definitions of building blocks for the azapentacene chemical space in older versions of MolBuilder. . . . . 88
5.2 Illustration of fragment addition to coves, fjords and bays within a aromatic system. These types of additions were forbidden by the chemical space exploration algorithm. . . . . 89
5.3 Promising azapentacene motifs identified during evolutionary chemical space exploration Cheng et al. (2020), labelled with their fitness values of reorganisation energy (eV) . . . . . 89
5.4 Illustration of using `mutation()` to convert random sets of aromatic CH positions into nitrogen heteroatoms . . . . . 90
5.5 A selection of OCELOT experimental molecule pi-conjugated systems, with their corresponding CSD REFCODES. . . . . 91
5.6 The selection of building blocks and mutation types generated from the filtered OCELOT experimental dataset. . . . . 93
5.7 MolBuilder code is limited to 2-point addition, avoiding the formation of certain fused ring structures (highlighted red), and ensuring only two-atom fused ring bonds can be created (highlighted green) . . . . . 94
5.8 Analysis of the OCELOT molecules from which building blocks were extracted, used to inform design choices in MolBuilder genetic algorithms 95
5.9 Heatmaps comparing the Tanimoto fingerprint similarity between 20 initial 'full' populations constructed with the OCELOT building blocks . . 98
5.10 Three of the five chosen side group types were used in each type of reduced OCELOT building block set. . . . . 98
5.11 Ring building blocks used to create and modify the first set of 'reduced' population molecules: aza-substituted ring systems. . . . . 99
5.12 Ring building blocks used to create and modify the second set of 'reduced' population molecules: aza- and thio-substituted ring systems. . . 99
5.13 Ring building blocks used to create and modify the third set of 'reduced' population molecules: acenes with amide/ester-type rings. . . . . 99
5.14 Ring building blocks used to create and modify the most complex set of 'reduced' population molecules, attempting to capture the variety of ring OCELOT fragments while working in a smaller space. . . . . 99
5.15 Heatmaps comparing the Tanimoto fingerprint similarity between 16 initial 'reduced' populations constructed with subsets of the OCELOT building blocks, also separated by the type of subset used. . . . . 100
5.16 100 molecules generated for initial 'full' population 1 . . . . . 101
5.17 Given a core with reactive groups (in this case, ortho-dianiline), a set of functionalisation reactions can be applied to generate reaction products. 102
5.18 Example molecular cores which fit the criteria applied, and were subsequently passed through computational 'reactions' to generate potential combi-HOFs . . . . . 103
5.19 The four selected functionalisation reactions, which convert ortho-dianiline groups into ring structures. . . . . 104

6.1 A selection of azapentacene-type molecules generated using MolBuilder, using the 'frags-in-situ' approach to build an exhaustive list of unique species. . . . 106
6.2 The 12 molecular backbones which all azapentacene species in the space explored are built from. . . . 107
6.3 Discovery rate plot for the exploration campaigns of mutation patterns for each carbon backbone in the azapentacene chemical space. . . . 107
6.4 The selection of building blocks and mutation types used to exhaustively generate molecules 'surrounding' the OCELOT chemical space . . . 110
6.5 Discovery rate plot for the exploration campaign using OCELOT building blocks. . . . 110
6.6 A selection of molecules generated from the OCELOT building blocks using the MolBuilder 'frags-in-situ' approach to build an exhaustive list of unique species. . . . 111
6.7 Discovery rate plot for exploration campaigns using the four 'reduced' sets of OCELOT building blocks. . . . 112

7.1 Plots to show how many unique molecules were sampled: by all 20 GAs (left); by each of the 20 GAs separately (right) . . . 116
7.2 Plots to show how many molecules get 'carried over' from between generations: counts of how many molecules from the previous generation reappear (left); how many molecules in the current population were also in the initial population (right) . . . 117
7.3 Plots to show diversity and composition for each population: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right) . . . 118
7.4 Plots to show how the fitness values (molecular electron reorganisation energy) change as each of the 20 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right) . . . 119
7.5 Plots to show how the fitness values (molecular electron reorganisation energy) change as each of the 20 GAs progress: minimum values per generation (left); minimum values above a 'reasonable' threshold (0.1 eV) of reorganisation energy values (right) . . . 120
7.6 Top 5 molecules sampled by each of 4 jobs, started from populations 1, 2, 11 and 18. . . . 120
7.7 Plots to show overlap, or molecules shared, between populations (left) and between molecules sampled up to a given generation (right) . . . 121
7.8 Plots to show how many unique molecules were sampled: by all 30 GAs using the chosen repeat populations (left); grouped by each 5 initial population types (right). . . . 122
7.9 Plots to show how many unique molecules were sampled: by all 6 GAs using population 18 (left); by each of the 6 GAs separately (right). . . . 123
7.10 Grid showing the top 5 molecules discovered in each repeat run from population 18 . . . 124
7.11 Plots to show how many unique molecules were sampled: by all 6 GAs using population 12 (left); by each of the 6 GAs separately (right). . . . 125
7.12 Grid showing the top 5 molecules discovered in each repeat run from population 12 . . . 125

7.13 Overlap plots examining each set of 6 repeats for populations 12 and 18: per generation (left) and cumulative (right) . . . 126
7.14 Three chosen side group types used in each reduced OCELOT building block set. . . . 127
7.15 Ring building blocks used to create and modify the first set of 'reduced' population molecules: aza-substituted ring systems. . . . 127
7.16 Plots to show how many unique molecules were sampled: by all 4 GAs using the aza-substituted ring system building blocks (left); by each of the 4 GAs separately (right) . . . 128
7.17 Plots to show how the fitness values (molecular electron reorganisation energy) change as each of these 4 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right) . . . 128
7.18 Plots to show diversity and composition for each population: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right) . . . 128
7.19 Overlap plots examining each set of 4 aza-system reduced population genetic algorithms: per generation (left) and cumulative (right) . . . 129
7.20 Ring building blocks used to create and modify the second set of 'reduced' population molecules: aza- and thio-substituted ring systems. . . 129
7.21 Plots to show how many unique molecules were sampled: by all 4 GAs using the aza- and thio-substituted ring system building blocks (left); by each of the 4 GAs separately (right) . . . 130
7.22 Overlap plots examining each set of 4 aza-/thio-system reduced population genetic algorithms: per generation (left) and cumulative (right) . . 130
7.23 Ring building blocks used to create and modify the third set of 'reduced' population molecules: acenes with amide/ester-type rings. . . . 130
7.24 Plots to show diversity and composition for each population: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right) . . . 131
7.25 Ring building blocks used to create and modify the most complex set of 'reduced' population molecules, attempting to capture the variety of ring OCELOT fragments while working in a smaller space. . . . 131
7.26 Plots to show how many unique molecules were sampled: by all 4 GAs using half of the original building block set (left); by each of the 4 GAs separately (right) . . . 132
7.27 Plots to show how the fitness values (molecular electron reorganisation energy) change as each of these 4 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right) . . . 132
7.28 Plots to show diversity and composition for each population: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right) . . . 132
7.29 Grid showing 24 molecules commonly found in the 61 genetic algorithms run so far, where reorganisation energies are below 0.125 eV. . . . 133
7.30 Plots to show the distribution of reorganisation energy values for all molecules sampled up to this point: majority of the distribution (main) and outliers with values higher than 0.7 eV (subplot). . . . 133

7.31 Plots to show the difference in distributions of reorganisation energies between initial populations (chosen at random, blue) and final populations (optimized via GAs, red). . . . . 134
7.32 Plots to show the difference in distributions of reorganisation energies between GA sampled molecules (blue) and the original experimental set (green). . . . . 135
7.33 Drawings of the top 18 'Murcko scaffolds' appearing amongst sampled molecules with reorganisation energies between 0.1 and 0.2 eV. Below each image is a label with the number of appearances the scaffold makes, and the average reorganisation energy value found in molecules sampled with this scaffold. . . . . 135
7.34 Drawings of 15 pentacene derivative molecules, ordered and labelled by reorganisation energy values. . . . . 136
7.35 Plots to show how many unique molecules were sampled: by all 25 mobility-led GAs (left); by separated into groups of five by initial population number (right) . . . . . 137
7.36 Plots to show how the fitness values change as each of the 25 mobility-led genetic algorithms progress, grouped by the initial population chosen: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right) . . . . . 138
7.37 Plots to show diversity and composition for set of five populations: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right) . . . . . 139
7.38 Overlap plots examining each set of 5 repeats for each initial population run in mobility-led genetic algorithms: per generation (left) and cumulative (right) . . . . . 139
7.39 Scatter plot comparing calculated electron mobilities and electron reorganisation energies of molecules sampled through the 25 electron mobility-led genetic algorithms. . . . . 140
7.40 The top 5 sampled molecules according to electron mobility . . . . . 140

8.1 The best (top) and worst (bottom) scoring azapentacene molecules according to SYBA . . . . . 145
8.2 Distribution of calculated SYBA scores for all molecules sampled in an exhaustive exploration of the azapentacenes; anything scoring below 0 is suggested to be synthetically difficult. . . . . 145
8.3 The best (top) and worst (bottom) scoring azapentacene molecules according to SCScore . . . . . 146
8.4 Distribution of calculated SCScore scores for all molecules sampled in an exhaustive exploration of the azapentacenes; higher scoring molecules are suggested to be synthetically difficult. . . . . 146
8.5 Violin plots describing synthetic difficulty scores for the azapentacenes, grouped by nitrogen count. . . . . 147
8.6 Violin plots describing synthetic difficulty scores for the azapentacenes, grouped by generic backbone type. . . . . 148
8.7 Scatter plots comparing SYBA and SCScore synthetic difficulty scores. . 149
8.8 Predicted synthetic route to an example azapentacene molecule, produced by AiZynthFinder; green borders mean that the molecule is available as a reagent in the ZINC15 database . . . . . 149

8.9 5 ChEMBL matches (top) and 15 PubChem matches (bottom) . . . . . 150
8.10 Distribution of calculated SYBA scores for all molecules sampled with reorganisation energy genetic algorithms; anything scoring below 0 is suggested to be synthetically difficult. . . . . 151
8.11 The best (top) and worst (bottom) scoring reorganisation energy GA sampled molecules according to SYBA . . . . . 152
8.12 Distribution of calculated SCScores for all molecules sampled with reorganisation energy genetic algorithms; higher scoring molecules are suggested to be synthetically difficult. . . . . 153
8.13 The best (top) and worst (bottom) scoring reorganisation energy GA sampled molecules according to SCScore . . . . . 154
8.14 Distribution of calculated SYBA/SCScores for molecules sampled by reorganisation energy genetic algorithms, partitioned into those with all building blocks (blue) and those without (red). . . . . 155
8.15 Scatter plot of SYBA and SCScore results for all molecules sampled with reorganisation energy genetic algorithms. . . . . 156
8.16 Grid of species discovered in reorganisation energy genetic algorithms, where post-hoc analysis with SYBA and SCScore results in a disagreement regarding their percieved synthetic difficulties. . . . . 156
8.17 Scatter plot of SYBA and SCScore results for the original OCELOT molecules (red) used to select molecular building blocks, compared to those sampled through the genetic algorithms (blue). . . . . 157
8.18 12 molecules marked as highly synthetically difficult according to AiZynthFinder. . . . . 158
8.19 Examples predicted synthetic routes to targets from the OCELOT space with very high AiZynthFinder scores . . . . . 159
8.20 12 molecules which exhibit both optimal reorganisation energies and synthetic difficulty scores according to SYBA and SCScore. . . . . 160

9.1 Plots to show how many unique molecules were sampled: by all 20 SCScore-led GAs (left); by each of the 20 GAs separately (right) . . . . . 164
9.2 Plots to show how the fitness values (SCScore) change as each of the 20 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right) . . . . . 164
9.3 Plots to show diversity and composition for each population in 20 SCScore-led GAs: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right) . . . . . 165
9.4 Plots to show how the fitness values (SCScore) change as each of the 16 reduced-population GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right) 166
9.5 Plots to show how many unique molecules were sampled: by all 20 SYBA-led GAs (left); by each of the 20 GAs separately (right) . . . . . 166
9.6 Plots to show how the fitness values (SYBA) change as each of the 20 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right) . . . . . 167
9.7 Plots to show diversity and composition in 20 SYBA-led GAs . . . . . 167
9.8 Plots to show how many unique molecules were sampled: by all 16 reduced population SYBA-led GAs (left); by each of the 20 GAs separately (right) . . . . . 168

9.9 18 molecules sampled by synthetic difficulty GAs multiple times, while achieving low SCScores and high SYBA scores. . . . . 169
9.10 10 generic molecular shapes commonly sampled by synthetic difficulty genetic algorithms, with average calculated properties. . . . . 170
9.11 15 molecules sampled by synthetic difficulty genetic algorithms, achieving the lowest average reorganisation energies from these runs. . . . . 171

10.1 Plots to show how many unique molecules were sampled: by all 14 multi-objective GAs (left); by each of the 14 GAs separately (right) . . . . 173
10.2 Plots to show how the individual properties change as each of the 14 GAs progress: reorganisation energy (left) and SCScore (right) . . . . . 174
10.3 Plots to show how the individual properties change for the top 100 molecules sampled up to each generation each of the 14 GAs progress: reorganisation energy (left) and SCScore (right) . . . . . 175
10.4 Plots to show how the minimum values of each property changes as each of the 14 GAs progress: reorganisation energy (left) and SCScore (right) . 175
10.5 Plots to show how the individual properties change as each of the 14 GAs progress, grouped by the weights placed on each property in the fitness function: reorganisation energy (left) and SCScore (right) . . . . . 176
10.6 Plots to show how the individual properties change for the top 100 molecules sampled up to each generation each of the 14 GAs progress, grouped by the weights placed on each property: reorganisation energy (left) and SCScore (right) . . . . . 177
10.7 Scatter plots comparing reorganisation energies and SCScores for molecules sampled with multi-objective algorithms, categorised by the weights placed on each property. . . . . 177
10.8 10 top performing molecules, with both low reorganisation energies and low SCScores, discovered during multi-objective genetic algorithms. . . 178
10.9 40 top performing molecules, with both low reorganisation energies and low SCScores, rediscovered during multi-objective genetic algorithms. . 180

11.1 Species studied as mixed crystals: Dibenzothiophene (DBT), dibenzofuran (DBF), fluorene (FLU), and carbazole (CBZ). . . . . 181
11.2 Predicted crystal-structure landscape for DBT. . . . . 182
11.3 Predicted crystal-structure landscape for DBF. . . . . 183
11.4 As the host:imposter ratio reaches 50:50, the number of possible imposter position sets increases rapidly . . . . . 184
11.5 Overview of computational simulation of mixed crystal systems. . . . . 185
11.6 Plots of the dibenzofuran / dibenzothiophene mixed crystal system, summarising 40 sets of randomly configured supercells with varying compositions, when the initial 'host' supercell was built from dibenzofuran *Pnma* unit cells. . . . . 186
11.7 Plots of the dibenzofuran / dibenzothiophene mixed crystal system, summarising 40 sets of randomly configured supercells with varying compositions, when the initial 'host' supercell was built from dibenzofuran *P21/n* unit cells. . . . . 187
11.8 A 'dual energy' plot, used to compare data from the dibenzofuran / dibenzothiophene system. Red lines indicate compositions produced experimentally; DBF:DBT ratios 77:23, 54:46, 41:59, 27:73 and 20:80 . . . . . 188

11.9 From the work of Villeneuve et al. (2022); sublimed crystals of pure DBT imaged by optical microscopy under polarized light. The thin plates correspond to the metastable *Pnma* polymorph and the needles to the previously reported *P21/n* form. . . . . 188

12.1 10 top performing molecules sampled through genetic algorithms, which exhibit the most promising calculated reorganisation energies alongside low molecular and synthetic complexity scores. . . . . 190
12.2 Sampled species where disagreement arises between scoring of molecular complexity (SYBA) and synthetic complexity (SCScore). . . . . 191

Appendix A.1 100 molecules generated for initial 'full' population 1 . . . . . 196
Appendix A.2 100 molecules generated for initial 'full' population 2 . . . . . 197
Appendix A.3 100 molecules generated for initial 'full' population 3 . . . . . 198
Appendix A.4 100 molecules generated for initial 'full' population 4 . . . . . 199
Appendix A.5 100 molecules generated for initial 'full' population 5 . . . . . 200
Appendix A.6 100 molecules generated for initial 'full' population 6 . . . . . 201
Appendix A.7 100 molecules generated for initial 'full' population 7 . . . . . 202
Appendix A.8 100 molecules generated for initial 'full' population 8 . . . . . 203
Appendix A.9 100 molecules generated for initial 'full' population 9 . . . . . 204
Appendix A.10 100 molecules generated for initial 'full' population 10 . . . . 205
Appendix A.11 100 molecules generated for initial 'full' population 11 . . . . 206
Appendix A.12 100 molecules generated for initial 'full' population 12 . . . . 207
Appendix A.13 100 molecules generated for initial 'full' population 13 . . . . 208
Appendix A.14 100 molecules generated for initial 'full' population 14 . . . . 209
Appendix A.15 100 molecules generated for initial 'full' population 15 . . . . 210
Appendix A.16 100 molecules generated for initial 'full' population 16 . . . . 211
Appendix A.17 100 molecules generated for initial 'full' population 17 . . . . 212
Appendix A.18 100 molecules generated for initial 'full' population 18 . . . . 213
Appendix A.19 100 molecules generated for initial 'full' population 19 . . . . 214
Appendix A.20 100 molecules generated for initial 'full' population 20 . . . . 215
Appendix A.21 100 molecules generated for initial 'reduced' population 1 . . 217
Appendix A.22 100 molecules generated for initial 'reduced' population 2 . . 218
Appendix A.23 100 molecules generated for initial 'reduced' population 3 . . 219
Appendix A.24 100 molecules generated for initial 'reduced' population 4 . . 220
Appendix A.25 100 molecules generated for initial 'reduced' population 5 . . 221
Appendix A.26 100 molecules generated for initial 'reduced' population 6 . . 222
Appendix A.27 100 molecules generated for initial 'reduced' population 7 . . 223
Appendix A.28 100 molecules generated for initial 'reduced' population 8 . . 224
Appendix A.29 100 molecules generated for initial 'reduced' population 9 . . 225
Appendix A.30 100 molecules generated for initial 'reduced' population 10 . . 226
Appendix A.31 100 molecules generated for initial 'reduced' population 11 . . 227
Appendix A.32 100 molecules generated for initial 'reduced' population 12 . . 228
Appendix A.33 100 molecules generated for initial 'reduced' population 13 . . 229
Appendix A.34 100 molecules generated for initial 'reduced' population 14 . . 230
Appendix A.35 100 molecules generated for initial 'reduced' population 15 . . 231
Appendix A.36 100 molecules generated for initial 'reduced' population 16 . . 232

Appendix A.37 Top 5 molecules sampled in the first round of reorganisation energy GAs, for initial populations 1-6 . . . 233
Appendix A.38 Top 5 molecules sampled in the first round of reorganisation energy GAs, for initial populations 7-12 . . . 234
Appendix A.39 Top 5 molecules sampled in the first round of reorganisation energy GAs, for initial populations 13-20 . . . 235
Appendix A.40 Plots to show how the fitness values (molecular electron reorganisation energy) change as each of these 4 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right) . . . 236
Appendix A.41 Plots to show diversity and composition for each population: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right) . . . 236
Appendix A.42 Plots to show how many unique molecules were sampled: by all 4 GAs using the amide/ester ring system building blocks (left); by each of the 4 GAs separately (right) . . . 236
Appendix A.43 Plots to show how the fitness values (molecular electron reorganisation energy) change as each of these 4 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right) . . . 237
Appendix A.44 Overlap plots examining each set of 4 amide/ester-type system reduced population genetic algorithms: per generation (left) and cumulative (right) . . . 237
Appendix A.45 Overlap plots examining each set of 4 half building block reduced population genetic algorithms: per generation (left) and cumulative (right) . . . 237
Appendix A.46 Violin plots describing the AIZynthFinder best scores for predicted synthetic routes to molecules generated with the OCELOT building blocks in reorganisation energy genetic algorithms. . . . 238
Appendix A.47 Distribution of calculated SYBA/SCScores for molecules sampled by reorganisation energy genetic algorithms, partitioned according to the type of reduced block building set used. . . . 238
Appendix A.48 Plots to show how many molecules get 'carried over' from between generations in SCScore-led GAs: counts of how many molecules from the previous generation reappear (left); how many molecules in the current population were also in the initial population (right) . . . 239
Appendix A.49 Plots to show how many unique molecules were sampled: by all 16 reduced population SCScore-led GAs (left); by each of the 20 GAs separately (right) . . . 239
Appendix A.50 Plots to show diversity and composition for each population in 16 reduced population SCScore-led GAs: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right) . . . 239
Appendix A.51 Plots to show how the fitness values (SYBA) change as each of the 16 reduced-population GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right) . . . 240
Appendix A.52 Plots to show diversity and composition in 16 reduced population SYBA-led GAs . . . 240

# List of Tables

4.1 Table describing some important parameters to be set by a configuration file before running a MolBuilder genetic algorithm. An example configuration file can be found in the appendix. . . . . 49
4.2 Table describing modified configuration parameters to be set before executing reaction product enumeration with MolBuilder's 'reactor' molecular generation functionality. . . . . 73

6.1 Azapentacene structure counts after grouping by empirical formula; equivalent to clustering by nitrogen substitution level. . . . . 108

7.1 Summaries of the repeat runs conducted for 'full population' reorganisation energy genetic algorithms. . . . . 123

# Listings

4.1 Function to fragment a molecule along an aromatic bond, and return two fragments. One fragment retains 'attachment points', marking where it was originally connected to the other fragment. . . . . 54
4.2 Function to add a fragment to a molecule . . . . . 57
4.3 Function to perform 'mutations' on a molecule, adding side groups to a molecule at aromatic CH positions, or removing side groups. . . . . 58
4.4 The base function used in MolBuilder's 'reactor' molecule generator, which applies a defined reactionSMARTS to a molecular SMILES, returning a single outcome chosen at random . . . . . 74
4.5 Utilising the SYBA classifier to predict a SYBA score from SMILES. . . . 75
4.6 Utilising SCScore to predict a SCScore from SMILES. . . . . 77
4.7 Generating 3D molecular geometries from SMILES strings with RDKit UFF optimization. . . . . 80
5.1 Building blocks and design rules required to generate azapentacene-type molecules. . . . . 90

# Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as:
   Norbert M Villeneuve, Joshua Dickman, Thierry Maris, Graeme M Day, and James D Wuest. Seeking rules governing mixed molecular crystallization. *Crystal Growth & Design*, 23(1):273–288, 2022
   Rebecca J Clements, Joshua Dickman, Jay Johal, Jennie Martin, Joseph Glover, and Graeme M Day. Roles and opportunities for machine learning in organic molecular crystal structure prediction and its applications. *MRS Bulletin*, 47(10): 1054–1062, 2022

Signed:..... Date:.....

# Acknowledgements

We acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton. Via our membership of the UK's HEC Materials Chemistry Consortium, which is funded by EPSRC (EP/X035859), this work also used the ARCHER2 UK National Supercomputing Service http://www.archer2.ac.uk for computational resources.

I'd like to thank to my supervisor Professor Graeme Day for his support, assistance and patience over the course of both this project and my undergraduate degree; he has been there for me during every step of my journey through higher education, and has gone above and beyond to help me throughout this time. The Day Research Group also deserves thanks both for their contributions to the CSPy code-base, and for nurturing a brilliant and kind team of researchers; without their support, performing this work would've been far more difficult, and the developments made during my time with them never ceased to impress.

Finally, I'd like to thank my family for listening to my thoughts throughout this project, even when the science gets a bit too complex for them; with their cheers of support, I've been able to push through difficult times, and this work been able to flourish because of them. I'm also thankful for the Familingus, particularly Kia, Devito, and of course, Twigbot_. This incredibly welcoming community helped me to push through the final stages of writing this project with laughs, cheers and gifts; many late nights of work would've felt so much longer without them.

# Chapter 1

# Introduction

## 1.1 Background and Motivations

The work described in this thesis can broadly be separated into three categories; the use of a computational approach to materials discovery, how molecules can be generated with programming and optimized in terms of desired properties, and how synthetic difficulty can be estimated using computational tools given only information on a molecular structure.

### 1.1.1 Discovering new materials

Materials discovery encompasses the broad range of work, searching for new materials with promising properties for real-world applications, for example the class of organic molecules which can be used as semiconducting materials in devices such as organic light-emitting diodes (OLEDs) or organic field-effect transistors (OFETs), and molecules which are able to form porous framework structures for applications in gas-absorption or catalysis.

Traditionally, the materials discovery process has been largely reliant on a trial-and-error approach, where past knowledge of materials which exhibit promising properties is exploited to determine the direction of future experimentation; for the most part, this means that research often becomes restricted to small iterations of known material classes, particularly due to the risk associated with exploring completely new ideas.

### 1.1.2 Materials discovery aided by computation

Recent advancements in data-driven techniques, increases in computational power, and an ever-growing base of knowledge found in the literature have increased the feasibility of implementing computational tools into materials discovery processes.

Computational tools which were previously used to analyse and rationalise experimental findings can also be used in a predictive manner, before any physical experimentation takes place. The results of such analysis can guide the exploration of potential material spaces towards candidates with promising calculated properties, or be used to rank and prioritise species suggested to experimental chemists.

Through the use of a computational 'lab bench', molecules can be constructed with little to no cost, through the combination of molecular substructures, or the prediction of outcomes given reaction and reagent combinations.

These processes may not always follow the rules of organic chemistry, or mirror the *actual outcomes* of reactions, meaning that unrealistic or difficult-to-synthesise targets can be generated; the occurrence of such cases can be mitigated through the careful definition of rules when generating molecules, and by adding consideration of the experimental synthesis process into the computational 'toolbox'.

### 1.1.3 Predicting synthetic difficulty

Computational exploration of a chemical space is an excellent step on the path to material discovery, but experimental validation is vital; generated molecules need to be synthesisable in a laboratory, both to be used as materials in the first place, and to ensure that predicted properties are realistic. This means that any molecules generated through computational engines must be checked to ensure that they are chemically sound, and are accessible through feasible synthetic pathways.

Synthetic difficulty is an abstract and relatively subjective concept, describing a molecule in terms of how easy or hard it may be to generate in a laboratory; this is not a property that can be calculated through the simulation of molecular structure, or assigned a definitive value.

Even scores from synthetic chemists can vastly differ when asked to assign synthetic difficulties to a given species, as their often diverge due to differing experience and skill sets; predictive methods, such as the work described by Bonnet (2012), that are based on responses from a group of chemists have required the use of average scores, due to a disparity in their opinions.

Capturing and considering all of the relevant factors of organic synthesis is a monumental task; instead of trying to estimate scores which consider every aspect contributing to easier or more difficult synthetic targets, many computational tools look at different sets of factors to determine how synthetically feasible a given molecule may be.

Synthetic difficulty scoring can be based on the molecular structure alone, through the use of computational models trained with datasets of known molecules, or on information pulled from computer-aided synthesis planning and databases of known reactions such as Reaxys (`https://www.reaxys.com/`). By assessing the results of such scoring functions during the materials discovery workflow, molecules generated through computation can be suggested for further experimental work, with improved confidence that they can actually be synthesised and studied in laboratory settings.

## 1.2 Thesis overview

As stated at the start, this project can be split into three overall aims; the research and work covered by this thesis aims to showcase the following:

- Improvements made to the Day Group CSPy code used as part of a larger materials discovery workflow; in particular, the approaches to molecular generation, and optimization of target properties through the use of molecular modification operations within a genetic algorithm.
- Execution of organic materials discovery campaigns with MolBuilder, an evolutionary algorithm designed to efficiently explore chemical space while optimizing molecular and solid state properties.
- Implementations to add consideration of synthetic feasibility during chemical explorations with MolBuilder, through the addition of synthetic difficulty estimation methods which assess molecules discovered during materials discovery campaigns, both during and after-the-fact.

With these changes to the code, and subsequent generated results, this thesis will aim to answer the following research questions:

- How useful is the assessment of synthetic difficulty during material discovery workflows, and what kind of information is gained from this analysis?
- Synthetic difficulty can be considered at several points within the materials discovery process; at what point is it best to estimate and utilise these results?
- Different synthetic difficulty estimation tools look at seperate aspects of the molecule, its predicted synthetic path, or its place in a wider reaction network of reagents and products; which approach is the most effective for producing synthetically feasible species?
- Can the limitations of narrow focus on synthetic difficulty reduction be mitigated by considering other properties in a multi-objective optimization approach?

## 1.3 Thesis structure

Starting with a review of the literature, current computational approaches to synthetic difficulty prediction will be described, both to provide context for this work, and to begin an explanation of the theory behind these tools. Literature concerning the use of computational tools within materials discovery process will be discussed, which would place the results of this work in their wider context, along with some discussion concerning the molecular classes targeted and their applications.

The approach to molecular generation and chemical space exploration will be summarised in the context of past work, and literature produced by the Day Group, providing an overview of the MolBuilder evolutionary algorithm and the theory it is based on. Other approaches to molecular generation will be discussed, such as the combination of known reagents and compatible reagents to produce libraries of potential products; these topics will be brought together to describe their respective places and use-cases within materials discovery workflows.

The theory supporting relevant computational tools, such as crystal structure prediction and various property calculations, will be described to ensure that their use within this work is justified.

This will lead into a discussion of the code that has been developed to implement processes described by the theory; many changes have been made to MolBuilder, allowing finer control over this evolutionary algorithm for molecular exploration. These changes will be explained and justified, with a focus on how some new functionality was designed to introduce biasing towards more synthetically feasible molecules.

The research described in this thesis can be broken down into separate projects, each of which takes a different approach to the consideration of synthetic difficulty, and examines a different 'type' of chemical space. In these sections, the decisions made regarding chemical space design and rationalisations behind them will be explained, and the overall setup of these projects will be outlined.

Results found and analysis conducted during this project will be separated into chapters according to the methodology used, ordered by the time at which each set of experiments was conducted. The first section concerns the idea of 'full chemical space exploration', where the goal is to generate all possible molecules within a defined chemical space; this highlights the sheer number of species that are accessible even through small sets of molecular building blocks, and the need for more efficient ways to explore these spaces.

This will be followed by analysis of the results generated with a set of genetic algorithms, which explored chemical spaces defined by sets of ring structures and side groups, producing generations of molecules with iteratively better electron reorganisation energies. More costly computational approaches will also be examined where molecules with higher electron mobilities are targeted, showcasing the use of large-scale CSP within evolutionary algorithms. The results from these experiments will be further analysed in terms of synthetic difficulty, with multiple tools used to estimate how easy or difficult generated molecules may be to synthesise in laboratory settings.

A chapter will then discuss the optimization of estimated synthetic difficulty scores as fitness functions to be optimized by genetic algorithms, and how this impacts the results of chemical space exploration.

The idea of 'multi-objective' genetic algorithms, which seek to optimize more than one property, is then explored; this chapter focuses on bringing together promising material properties and lower synthetic difficulty while exploring chemical space, and the balance that needs to be found between objectives in multiple objective fitness functions for optimization in genetic algorithms.

A final conclusions chapter will cover the main results and outline key findings, such as how synthetic difficulty estimation with computational tools has impacted the outcomes of genetic algorithms, and a comparison to results when these tools are used as a post-hoc filter; potential avenues for future work will be discussed, particularly tasks which will become feasible in the near future due to the current rapid increases in computational power and resources.

# Chapter 2

# Literature Review

## 2.1 Computational molecular generation

There are several methods available for the creation of molecules through computational settings in the literature; this work utilises both genetic algorithms and reaction product enumeration to generate molecules.

### 2.1.1 Producing molecules with a genetic algorithm

Evolutionary algorithms (or genetic algorithms) use ideas from natural selection to solve tasks, and are widely applied to problems in the cheminformatics field as noted by Brown (2011); this type of approach was utilised as early as 1992, where genetic algorithms were introduced as a method to calculate the 'minimum chemical distance' between groups of molecules by Fontain (1992).

Genetic algorithms work well when applied to molecular generation tasks, for example the work of Jensen (2019) in 'travelling' chemical space to optimize logP values in molecules whilst maintaining synthetic accessibility through computational scoring; Tripp and Hernández-Lobato (2023) highlight that these algorithms are frequently able to outperform more complex, machine learning based approaches to unconditional and single-objective optimization of molecular populations.

MolBuilder is an evolutionary algorithm developed by Cheng et al. (2020) to search defined chemical spaces by generating populations of candidate molecules with ideal properties for a given material application. Development of this evolutionary algorithm was handed over at the start of this project; the changes undertaken during this time will be discussed in later chapters, such as modifications to the way in which molecules are modified, and the implementation of synthetic difficulty consideration.

Populations of optimized molecules are generated with MolBuilder by executing the four general steps in an evolutionary process: initialization, selection, genetic operations and termination. As the algorithm progresses, 'fitter' members of the population will survive and proliferate, while 'unfit' members will 'die off' and not contribute to further generations.

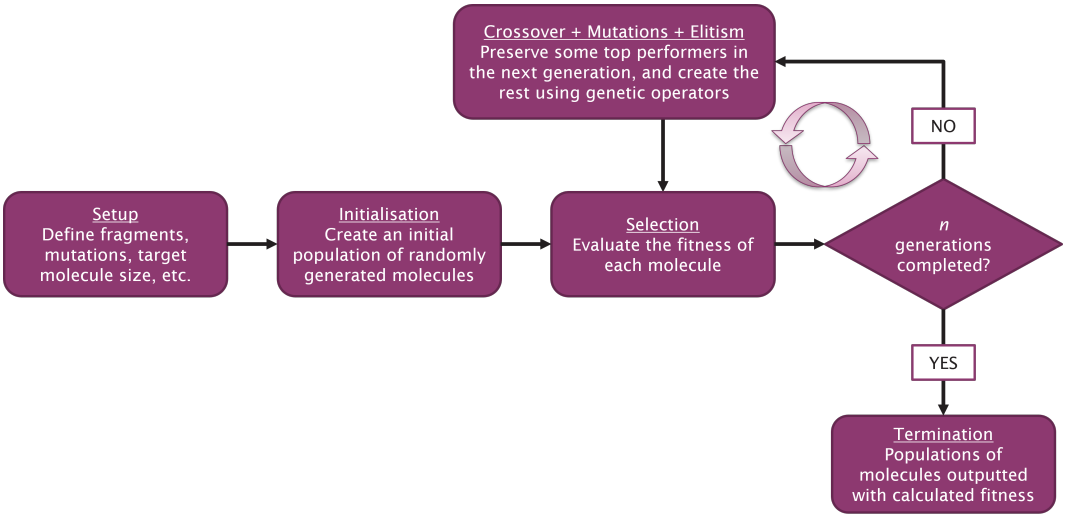



FIGURE 2.1: Flowchart describing the evolutionary algorithm process for generating molecules with optimized properties.

In the case of MolBuilder evolutionary algorithms, these four steps are applied to generate, modify and optimize populations of molecules:

- Initiation: the user defines a starting set of molecular building blocks, which determines what constructed molecules will be composed of; these are then used to generate a random initial population of molecules.
- Selection: whenever a population is generated, each molecule is subjected to a fitness function calculation, ranking the population by how promising predicted

properties are; fitness functions can be any property accessible from the molecular structure.

- Genetic operations: new populations are mostly generated through the use of genetic operations, which take 'parent' molecules from the previous generation, and combine fragments from each 'parent' to produce 'children'.
- Termination: Once the genetic algorithm has been run for a chosen number of populations, the entire set of generations is outputted, along with any calculated properties used in the fitness calculation.

'Genetic operations' refers to the ways in which new 'child' molecules can be generated based on molecules in previous populations, through crossovers of fragments extracted from pairs of 'parent' molecules, followed by a chance of recombinations and mutations on each 'child'; these operations and their implementations within MolBuilder are described fully in the theory framework and code development chapters.

In an initial study of the method by Cheng et al. (2020), aza-substituted pentacene species were targeted for use as organic semiconductors; reorganisation energies were calculated with Gaussian09 (Frisch et al.), and fitness functions based on this property were used to bias the algorithm.

MolBuilder was able to generate promising candidate species with optimal calculated reorganisation energies, which acted as a filter to extract top performing molecules from the larger chemical space accessible; further, more expensive calculations were performed only on the most promising subset of molecules.

### 2.1.2 Reaction-based approaches to molecular generation

Rather than generating molecules by combining fragments together in an arbitrary way, there are several approaches which utilise reactions, encoded for use in computational settings; these can then be applied to chosen molecules, producing possible reaction products; Bradshaw et al. (2020) perfectly describe this approach to

molecular generation, stating ”it is not only important what to make, but crucially how to make it”.

If molecules are constructed by iteratively applying known reactions, starting from easy-to-obtain precursor species, then the molecular generation process effectively captures and considers synthetic difficulty *in-situ*; Bradshaw et al. (2020) accomplish this by automating the generation of directed acyclic graphs (DAGs) which represent multi-step syntheses, through the use of a specialised deep-learning model.

In a similar publication, Schürer et al. (2005) optimize so-called ’vProtocols’, which dynamically combine sequences of chemical transforms through the use of genetic algorithms. These reaction sequences are evaluated by enumerating all potential reaction products that each could produce with a given set of precursor molecules, and calculating predicted properties for these reaction products.

Approaching molecular generation through the combination of reactions and reagents was used by Patel et al. (2020) to produce a library of over one billion ’readily synthesisable’ molecules, for use in drug discovery; 150,000 commercially available molecular ’building blocks’ were combined with a variety of chemical transformation steps, meaning that the generated molecules should have reliable synthetic pathways, given access to the right reagents and equipment.

## 2.2 Synthetic difficulty

Computational methods are only one part of the materials discovery workflow; once novel species have been identified, experimental chemists will want to synthesise them. This can be a monumental task, especially when large collections of candidates have been generated.

Estimation of synthetic difficulty can help bridge the gap between computational and experimental chemists. Rather than suggesting a large set of candidates, many of which could be unfeasible targets, species can be ranked by how difficult they are to synthesise. The smaller, more accessible set of candidates can then be suggested to

experimentalists for synthesis and analysis in laboratory settings, reducing material costs, saving time, and accelerating the discovery process.

### 2.2.1 Estimating synthetic difficulty with computational tools

Many tools are available to predict 'scores' regarding the synthetic difficulty of target species, which focus on different aspects of molecules, or information extracted from further analysis, which can influence how easy or difficult they may be to produce in labs. Boda et al. (2007) suggest three general approaches to the prediction of synthetic accessibility; these methods aren't mutually exclusive, in fact a combination of approaches can help to cover the shortcomings of each:

- Complexity-based methods: structures are examined using 'complexity factors' based on pre-existing data, such as the frequency of substructure appearances in experimental databases. Scores are produced based on how complex the substructures appearing in the target molecule are. Similarity to possible starting materials and potential reaction pathways are not considered.
- Starting material-based methods: Building upon pure complexity-based methods, similarity to starting materials can be considered by analysis of frequently appearing substructures in a database of purchasable precursors. While similarity to precursors can make some syntheses easier, information on the synthetic route itself is not considered; connecting purchasable fragments or modifying them to produce target molecules may not always be feasible.
- Retrosynthetic-based methods: Consideration of reaction pathways can be achieved through computer-aided synthesis planning, where structures are broken down into 'retrons' which can be recombined by a known reaction. This process is repeated until all retrons are available as precursors, or cannot be broken down further. While retrosynthetic analysis can be costly and time consuming, it is an obvious choice when investigating synthetic difficulty.

Boda et al. (2007) constructed a scoring function with aspects of each 'category', incorporating molecular complexity (size, symmetry, branching, rings, unsaturation, heteroatoms, stereochemistry), similarity to available precursors (identifying complex substructures which can be purchased to reduce synthetic difficulty) and retrosythetic reaction fitness (decomposing the target into smaller components, targeting common 'retrons').

Voršilák et al. (2020) created a fragment-complexity based synthetic difficulty estimation tool, SYBA, for the rapid classification of organic compounds as easy- or hard-to-synthesize. Predictions of synthetic difficulty can be made by using molecular complexity as a proxy; the appearance of fragments in the target molecule are compared to frequencies of those substructures in both a database synthetically accessible species, and a 'synthetic' dataset of hard-to-synthesise species.

By making use of the Reaxys database, Coley et al. (2018) generate a reaction network, and describe a method to predict synthetic difficulty based on the position of target molecules within this network. These SCScores are likely to be lower in molecules which are more similar to reagents than reaction products, implying less difficulty in synthesis; if a molecule is similar to the product(s) of reactions, it likely requires a reaction step to produce.

The ability to produce a feasible synthesis plan for a target molecule is a good indication that it may be synthetically accessible, however this isn't the only useful information which can be obtained from retrosynthetic analysis: the number of steps, yield of each step, and the availability of precursors can all be linked to synthetic difficulty. Furthermore, if a given reaction results in a mixture of by-products, separation and purification steps can increase difficulty in synthesis; the impact of using reactions with multiple products should be considered. Many computational retrosynthesis prediction tools already exist, based on a variety of algorithms/models.

One example is AiZynthFinder, a neural network-based retrosynthetic planning tool produced by Genheden et al. (2020), which utilises a Monte Carlo tree search to iteratively break down molecules into precursors. These precursors are compared against databases of known reagents (e.g. the ZINC database), and synthetic

pathways are constructed from a set of reaction templates (e.g. data from the US Patent and Trademark Office, USPTO).

As suggested above, information on the number of reaction steps and in-stock reagents can be used to build up some idea of a target molecules' synthetic difficulty. AiZynthFinder includes a scoring function based on these values, where species with short synthetic routes for which all reagents are available score highly.

### 2.2.2 Importance in materials discovery frameworks

When synthetic difficulty is not considered during molecular generation, candidates may be too challenging to synthesise in the lab Gao and Coley (2020). During their study of molecules which form porous materials in the solid state, Pulido et al. (2017) discovered two promising candidates through the use of crystal structure prediction and subsequent property predictions, labelled as T2 and P2; while T2 was synthesisable, and exhibited good solid-state properties, time and resources were spent in an effort to find a synthetic route to P2.

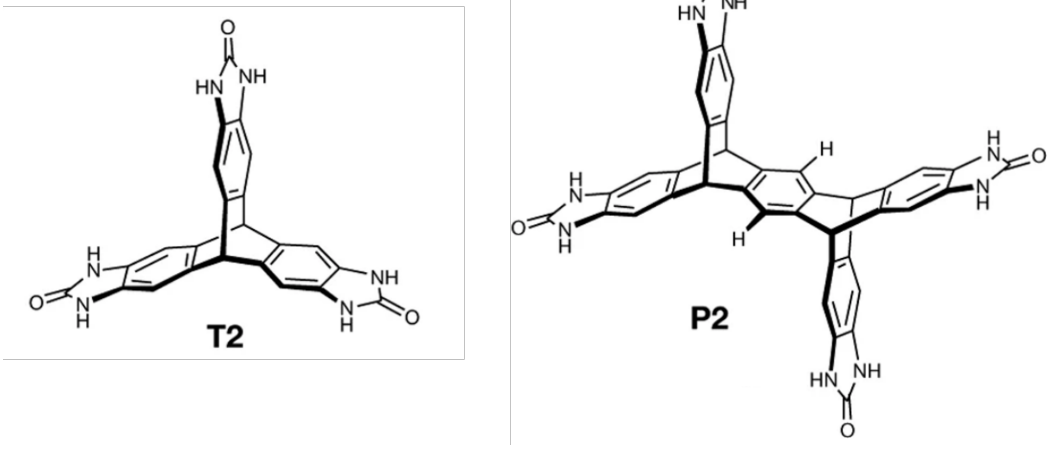



FIGURE 2.2: Two molecules, T2 and P2, discovered through computational efforts; one synthesisable, and one not. Figure adapted from the work of Pulido et al. (2017).

The opposite case can also be problematic, as too much focus on synthetic difficulty when generating molecules could lead to discovery campaigns getting 'stuck' in regions of chemical space with molecules that only have known, reliable synthetic

pathways, or species that are too simple, likely already existing in databases of known molecules.

While this may seem like a benefit at first, as all proposed molecules should be feasible for experimental validation, there is a significant drawback; by limiting discovery to species marked as synthetically accessible, the molecular generation algorithm is more likely to propose known species. This result will appear for different reasons dependant on the type of metric used.

### 2.2.3 When to consider synthetic difficulty

Synthetic difficulty can be assessed at multiple points within materials discovery workflows, even when only considering the computational portion.

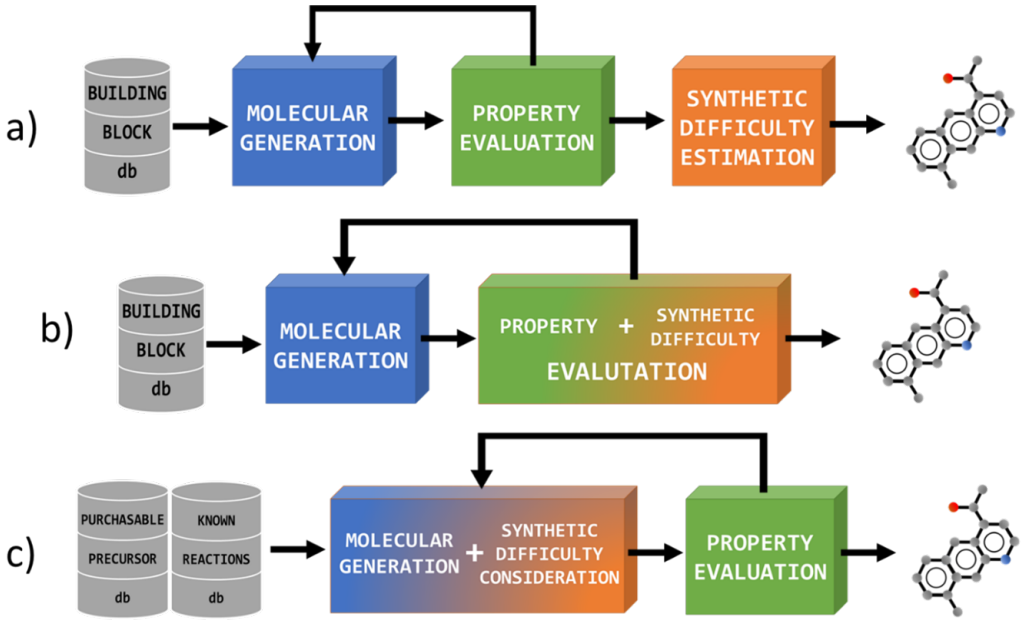



FIGURE 2.3: Three approaches two implementing synthetic difficulty consideration within the molecular generation process; based on Figure 1 from the work of Gao and Coley (2020).

Evaluation of synthetic difficulty as a post-hoc filter (Figure 2.3 part a) is the simplest approach, where molecules are generated without any consideration of syntheses; instead, the entire set of generated molecules is passed through a final filtering step after exploration is complete.

Heuristic biases can be implemented to guide molecular generation towards more synthetically accessible species (Figure 2.3 part b), however typical rapid scoring functions for synthetic accessibility such as SYBA focus on molecular complexity; structurally complex candidates are not always synthetically complex, as steps to add more complicated substructures can be feasible with a reasonable set of starting materials and reactions.

Explicit constraints on the building blocks and molecular transformations of a chemical space exploration campaign limits the proportion of the space that can be assessed, but should produce more synthetically accessible candidates (Figure 2.3 part c); this is analogous to the approach used by Patel et al. (2020), to produce the SAVI library of 'readily synthesisable' molecules.

### 2.2.4 Benefits to the experimental chemists

Computational material discovery campaigns can result in a large set of potential targets being suggested to experimental labs for synthesis; ranking the set of molecules by predicted synthetic difficulty gives a guide to which molecules may be 'easiest' to generate in a laboratory. Anything that can be done to reduce the amount of work in the experimental step of the material discovery process may result in saved time, materials, and laboratory costs.

Alongside these savings, some approaches to synthetic difficulty prediction will provide extra information and alternative approaches to the in-lab preparation of candidate materials. In the case of tools such as AiZynthFinder Genheden et al. (2020), predicted synthetic routes and required starting materials can be suggested alongside the target molecular structure; even if the predicted pathway isn't perfect, this information may serve as a good 'starting point' for the experimental chemists' approach to synthesis.

## 2.3 Materials discovery

Materials discovery encompasses a range of fields, but can be boiled down to the combination of materials science and 'information sciences'; the power of computational methods, particularly with ever-increasing access to large amounts of data, can accelerate experimental processes and direct research towards materials with given properties, filtering out candidates which aren't fit for their desired purpose.

### 2.3.1 Experimental approach to materials discovery

Traditional materials discovery work generally relies on the exploitation of past knowledge, where species known to perform well for given tasks are iteratively modified in an attempt to improve their properties; without prior knowledge about candidate materials, experimental attempts are more like 'shots in the dark', which can be an unwanted risk due to the associated time and material costs required for in-lab synthesis and analysis of candidate species.

Mroz et al. (2022) note that this issue compounds when the hypothetical size of the design space is considered; for organic species alone, the number of potential molecules is estimated to be between $10^{23}$ and $10^{60}$, which is far too many to examine. As the amount of research completed in materials science increases, new knowledge allows for further exploitation of 'safe', known material classes; however the inclusion of computational methods can help accelerate the discovery process.

### 2.3.2 Benefits of the computational approach

Computation has traditionally been used to explain material properties and structure-property relationships *after* experimentation. Mroz et al. (2022) describe the use of computation to conduct rapid screening of vast numbers of materials, predicting their properties and identifying promising candidates much faster than traditional experimental methods; especially in comparison to the more traditional 'trial and error' approach, as noted by Gu et al. (2019).

Computational methods can significantly reduce the time and cost associated with synthesising and characterizing materials in the laboratory; Keith et al. (2021) state that thousands of molecules can be screened with computation in the same time it would take for a single material to be synthesised and tested in-lab.

Despite the potential acceleration that computational approaches can provide, Mroz et al. (2022) note that experimental researchers need to trust and rely on computational predictions, particularly when exploring into novel material space; as noted above, much of the 'traditional approach' to materials discovery is knowledge based, and suggesting something completely novel can be met with hesitancy.

### 2.3.3 Considering synthesis and stability in materials discovery

As written by Oganov et al. (2019) in their review of the field, increasing availability of computational resources and the development of reliable structure prediction methods have been driving progress in materials discovery; many of the current approaches, however, make no consideration of a system's stability in the solid state, or the presence of a viable synthetic route, according to Greenaway and Jelfs (2021).

In a letter to Nature Materials, Jansen and Schön (2004) describe this approach as 'putting the cart before the horse', stating that the landscape of a chemical system should be explored for energetically stable regions before considering target physical properties. A similar argument could be made for the synthetic route to candidate materials; the set of molecules returned by a materials discovery campaign should all be synthetically feasible, even if this eliminates a large proportion of candidates.

Montoya et al. (2020) frame materials discovery as an optimization problem, with desired materials placed at minima/maxima of an arbitrary function:

$$material* = argmin(Experiment(material))$$

Finding the 'experiment' function is no simple task, as in many cases the actual behaviour a class of materials exhibits is highly complex; computational autonomous materials discovery (CAMD) passes this problem to computational 'research agents',

which apply their own logic and consider previously collected results to choose which candidate materials to investigate. Feeding past results back into these decision making 'agents' allows them to make use of heuristics, preventing excessive re-sampling and promoting future hypotheses which move the experimental 'campaign' towards its target.

CAMD links computational discovery with the outside world via experiment-specific application programming interfaces (APIs); the authors suggest density functional theory (DFT) and molecular dynamics (MD) workflows as experiments, however due to the program's modular design, extension to real-life experiments (particularly automated/robotic labs) should be feasible. This would involve computational construction of experimental methods, or at the very least some consideration of synthetic accessibility, which Montoya *et al.* do not discuss.

### 2.3.4 Molecular targets for materials applications

In this project, the majority of the work focuses on organic molecules which could be used for semiconductor applications; side projects also consider a class of molecules which could form porous organic frameworks, connected through hydrogen bonding between rigid molecular units.

#### 2.3.4.1 Small-molecule organic semiconductors

Organic molecules which exhibit promising properties as semiconducting materials are candidates for use in various electronic applications, as noted by Chen et al. (2023), such as organic light-emitting diodes (OLEDs), organic field-effect transistors (OFETs), and as organic photovoltaic materials (OPVs) for solar cells.

In contrast to their inorganic counterparts, Quinn et al. (2017) highlight the tendency of organic semiconductor materials to carry charges through a 'hopping' mode, due to larger intermolecular distances, more complex geometry of molecular units, and higher levels of disorder in the solid state. These factors place higher importance on the packing arrangements adopted by organic semiconductors; molecular units

should fall into stable arrangements that allow for effective transport of charge carriers, which can be boosted further through low energetic barriers to molecular geometry reorganisation.

Moving from inorganic to organics also brings a higher level of design to materials discovery, where small changes to the structure of molecular units can influence the mobility of charge carriers in the solid state. Chu et al. (2018) describe their strategy of using halogenation to enhance electron mobility; by varying the number and position of chlorine and fluorine substituents on tetra-azapentacenes, reorganisation energies can be lowered, and the crystalline packing arrangement can be influenced to promote charge transport.

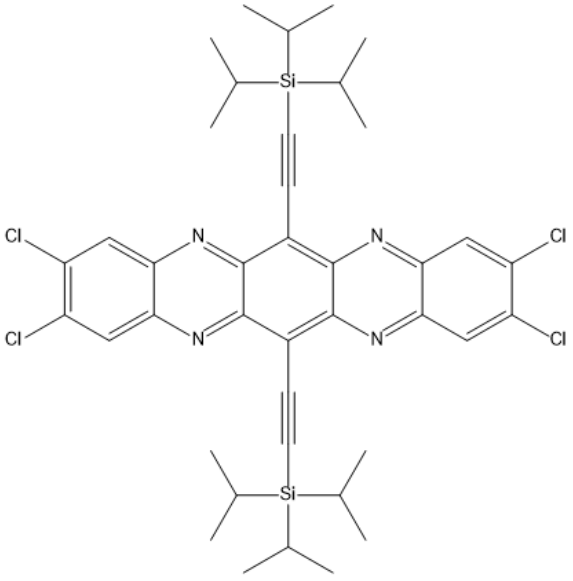



FIGURE 2.4: 4Cl-TAP, a chlorine-substituted azapentacene molecule discovered by Chu et al. (2018) which exhibits remarkably high electron mobility values.

Through this approach, Chu et al. (2018) highlight the discovery of 4Cl-TAP (Figure 2.4), which exhibited electron mobility values as high as 27.8 $cm^2V^{-1}s^{-1}$; this value was attributed to a reduced molecular reorganization energy and more optimal $\pi$-stacking of the 4Cl-TAP molecular units, each resulting from chlorine substitution.

#### 2.3.4.2 Hydrogen-bonded organic frameworks (HOFs)

HOFs are potentially porous materials, comparable to other framework material classes such as MOFs (metal-organic frameworks) and COFs (covalent-organic frameworks).

Rigid organic molecular units with directional hydrogen-bonding capable motifs are able to self-assemble through reversible hydrogen bonding interactions, forming framework structures with open pores; the size and dimensionality of these pores can be controlled through modification of the molecular units themselves.

Lin and Chen (2022) note high diversity in the potential applications of these frameworks: gas storage in HOFs with large pore spaces, enantioselective separation of mixtures through incorporation of chiral centres in HOF units, and use as heterogeneous catalysts through post-synthetic modification of HOF frameworks to include catalytic centers.

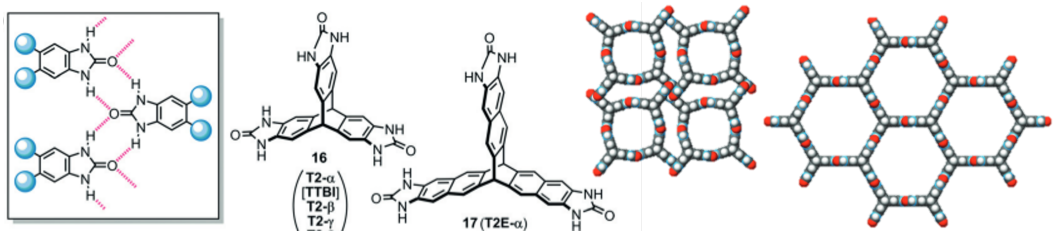



FIGURE 2.5: Diagram adapted from the HOF mini-review by Hisaki et al. (2019); examples of urea as a hydrogen bonding 'supramolecular synthon', which appears in 'T2' (from the work of Pulido et al. (2017)), to form porous frameworks (experimentally observed by Mastalerz and Oppel (2012)).

Research on this class of materials has gained momentum in the past decade, as noted by Hisaki et al. (2019); heterocycles and urea motifs are highlighted as common 'supramolecular synthons', which provide directional hydrogen bonding and aid the formation of rigid frameworks.

# Chapter 3

# Theory Framework

This section will delve into the ideas and theory behind methods used in the project; while the implementations of these concepts will not be explained here, points will be closely linked with the following section, which will elaborate on the code developed to execute this theory and generate results.

## 3.1 Computational handling of organic chemistry

Organic chemistry can be approached from a computational standpoint by encoding molecular structures and manipulating them with code. There are many ways in which molecules can be represented and stored, and unique analysis can be performed in a computational context; furthermore, the potential outcomes of reactions can be generated by converting reactive substructures into product substructures.

### 3.1.1 Storing and querying organic molecules

Molecules are frequently converted to and from different representations for storage and use in computational workflows, where the structural information is translated into one of several 'languages'; these representations each have particular benefits and downfalls, so choices must be made on the best type to use for different tasks.

RDKit `Molecule` (or `Mol`) objects are at the heart of conversions between structure representations and manipulation of molecules within this project; these Python objects can be generated from, and converted to, the various representations utilised in this work, with differing levels of information about the molecule in question dependant on which representation is used.

SMILES (Simplified Molecular Input Line Entry System) is a commonly used representation for storing molecules as text; symbols are used to describe atoms, and their connectivity. A given molecule can be described with multiple SMILES strings, where the same set of atoms and bonds can be described in different ways, so canonicalisation of SMILES strings is vital to ensure no duplication of molecules in sets; one 'canonSMILES' is selected, and any SMILES representation a given molecule can exhibit will be replaced by the chosen 'canonSMILES' instead.

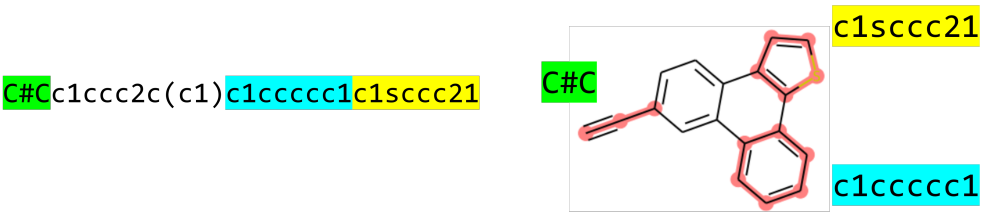



FIGURE 3.1: Showcasing the human-readability of SMILES strings - alkyne bonds are easily noticed by presence of '#', while aromatic rings and connectivity of atoms can be noted by lowercase atom symbols, and numbers which specify which atoms connect to which.

In this project molecules are mostly stored as SMILES, due to the balance it exhibits between human-readable and computer-readable. 2D structure and connectivity is preserved well on conversion to RDKit `Mols`, and SMILES strings can be examined 'at a glance' to rapidly understand information about the stored molecules without fully loading them and visualising a sketch; for example, it's easy to see the presence of aromatic heteroatoms, side groups, and certain bond types within molecules, as shown in Figure 3.1.

SMARTS (SMiles ARbitrary Target Specification) is an extension to the SMILES molecular entry 'language', built for sophisticated pattern matching of substructures within molecules. Atom and bond types can be defined at different levels of specificity, allowing flexibility to perform both vague and precise substructure

matching; both cases have their uses, where less specific SMARTS patterns can be used filter a dataset of molecules, selecting either two amine groups connected to two adjacent carbon atoms, or a more specific pattern could be written to select only *ortho-dianiline* substructures, where the carbon atoms must be aromatic.

Since these SMARTS strings are so flexible, they find many uses across the work described in this thesis, including the identification and counting of ring structures and side groups within generated molecules, filtering of datasets to remove species with undesirable substructues, and their use in the reactionSMARTS strings, which represent reactions as text by defining reacting centres with a SMARTS pattern, and the outcome as a second SMARTS describing the result of the transformation.

InChI are unique text representations of molecules; as opposed to SMILES, which can exhibit multiple representations for a given molecule, one InChI string always corresponds to a single species. InChiKeys can be generated from InChI through 'hashing' in a one-way conversion; InChIKeys cannot be converted back to other representations, but find use in this project for the unique naming of output files and folders, where representations such as SMILES and regular InChI contain 'illegal characters' that cannot be used in the naming of files.

XYZ files store 3D molecular structures as a list of atomic types, and positions of atoms in a Cartesian coordinate system; these are used for running calculations in programs such as Gaussian09 (Frisch et al.) and Psi4 (Parrish et al. (2017)), where the initial coordinates are a starting point for processes like molecular geometry optimizations, leading to property calculations such as molecular reorganisation energies and crystal structure prediction.

### 3.1.2 Molecular fingerprinting, similarity and diversity

Working with molecules in a computational setting allows for the similarity of species to be evaluated and assigned a score, through the use of molecular 'fingerprints'; these fingerprints encode the structure of molecules, allowing for comparison between species and a calculation of similarity scores, acting as a more defined metric rather

than the 'human approach' of simply looking at two molecules, and subjectively deciding how similar they are.

This work utilised 'Morgan' extended connectivity circular fingerprints, which can be generated from molecular structures.

1. Circular 'neighbourhoods' around each atom are selected; e.g. the neighbouring atoms, or those atoms plus their neighbours, and so on.
2. For each of these atomic neighbourhoods, a unique 'sub-fingerprint' is generated, which encodes the presence / lack of specific substructures. This can be assigned a unique substructure ID.
3. A hash function and bit-vector mapping can be applied to each substructure ID; the resulting bit vectors from these are used to determine which bits in the 'overall' Morgan fingerprint are set to 1, and which remain as zero.
4. Morgan fingerprints are generated at a fixed length; this can be changed, where a larger fingerprint size allows for more unique substructures to be represented (at the cost of more memory usage).

Once generated, these molecular fingerprints can then be compared with similarity metrics such as Tanimoto similarity. Given the bit vector fingerprints of two molecules, a Tanimoto measure of similarity can be calculated, where the intersection (number of positions where both A and B have a 1) of bit vectors $A$ and $B$ is divided by their union (total number of positions where either A or B has a 1); molecules which share more 'bits' are more similar, so score higher in this metric.

$$TanimotoSim(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

Molecular similarity can be 'flipped' to be used as a diversity measure (where $diversity = 1 - similarity$), which becomes useful when looking at populations of molecules generated with MolBuilder; a metric to determine 'population diversity' can be constructed using the average fingerprint diversity from all pairwise comparisons of molecules in the set. This value is particularly useful when populations of 'random' molecules are required; firstly to ensure that a set of many populations are 'at similar levels of diversity', and secondly to confirm that the molecules within each population are not too similar to one another.

### 3.1.3 Representing and applying reactions

Once molecules are loaded into RDKit, it is often desirable to perform modifications to them, whether the idea is to predict the outcomes of reactions, or to attempt transformations without 'real' reaction equivalents. As mentioned in the discussion of SMARTS notation, so-called reactionSMARTS strings allow for the encoding of transformation patterns, following the syntax of molecular SMARTS to define reactant and product substructures.

In a given reaction, only a subset of the atoms will actually be transferred, modified, moved or reconnected; in many cases, the majority of the surrounding atoms can be ignored, and changes are only made at the reaction centre. An obvious exception to this in synthesis is the presence of additional substructures within a molecule which can react in unintended ways; through the careful definition of highly specific reactant patterns, this exception can often be mitigated.

Using reactionSMARTS, the transformations relating to 'real reactions' can be encoded as text, including changes to atom connectivity, ring formation/destruction, inclusion of new atoms and removal of 'extra' atoms.

Figure 3.2 outlines the structure of an exemplar reactionSMARTS pattern, and a depiction of the reaction it corresponds to. The string representation can be split into three sections, separated by '>':

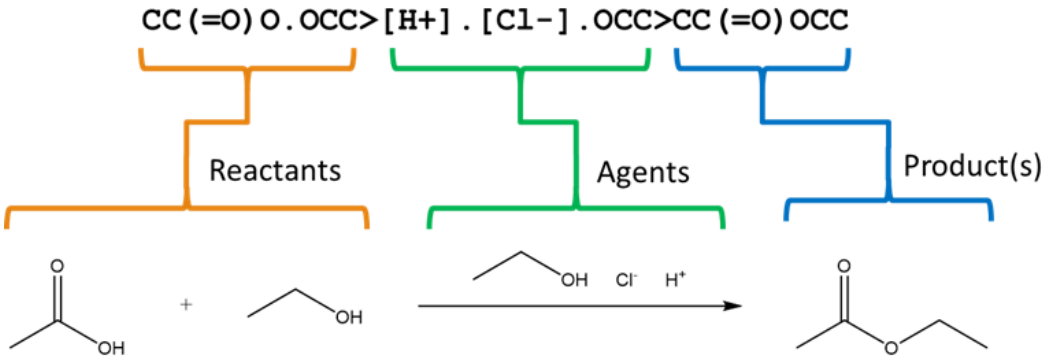



FIGURE 3.2: Simple example of a reaction mapped and represented using reactionSMARTS language

- The reactants, or molecular substructure(s) which get targeted/transformed by the reaction. These can be numbered atoms, which allows for the definition of their location in the products section.
- The agents, additional molecules/materials which need to be present, but aren't necessarily present in the reactants or products. ReactionSMARTS used in this project don't specify agents.
- The products, or resultant molecular substructures to be outputted by the reaction; in cases where reagent atoms have been numbered, their connectivity can be explicitly defined, through the matching of numbered atoms with the reactant pattern, to allow for more complex operations such as ring formations.

Since reactionSMARTS encode only the reaction site to transformed, optionally including neighbouring atoms to increase specificity of the target substructure, they can be applied to any molecule with the corresponding reaction site, converting said pattern to the product of the reaction. This flexibility must be used with caution, since if the reagent pattern is too vague, it may apply to unexpected reaction sites; as an example, a vague pattern intended to target ketones could also match the carbonyl of an amide structure.

The reactionSMARTS language isn't limited to 'realistic' chemistry; as long as both the reactant and product patterns follow SMARTS syntax, 'unrealistic' modifications can be executed on molecules using frameworks such as RDKit. As discussed later on,

molecular operations in the MolBuilder genetic algorithm utilise reactionSMARTS to perform addition of molecular fragments, mutation of atoms into side groups, and the fragmentation/recombination of one/multiple molecules. These operations don't correspond to 'real organic chemistry', but allow us to manipulate molecules in a way that allows for interesting exploration and sampling of chemical space.

## 3.2 Using ideas from genetics to manipulate molecules

MolBuilder, the Day Group evolutionary algorithm built to efficiently explore chemical spaces, pulls ideas from natural selection in order to optimize the 'populations' of molecules it generates; at each iteration, top-performing members according to the fitness function are either preserved, or modified with genetic operations to produce the next generation of molecules via crossover operations. The general idea stems from the theory of evolution, where the characteristics which make top-performing populations members are preserved in new populations by transferring them from parents to children; the general process undertaken by genetic algorithms can be followed in the flowchart Figure 3.3.

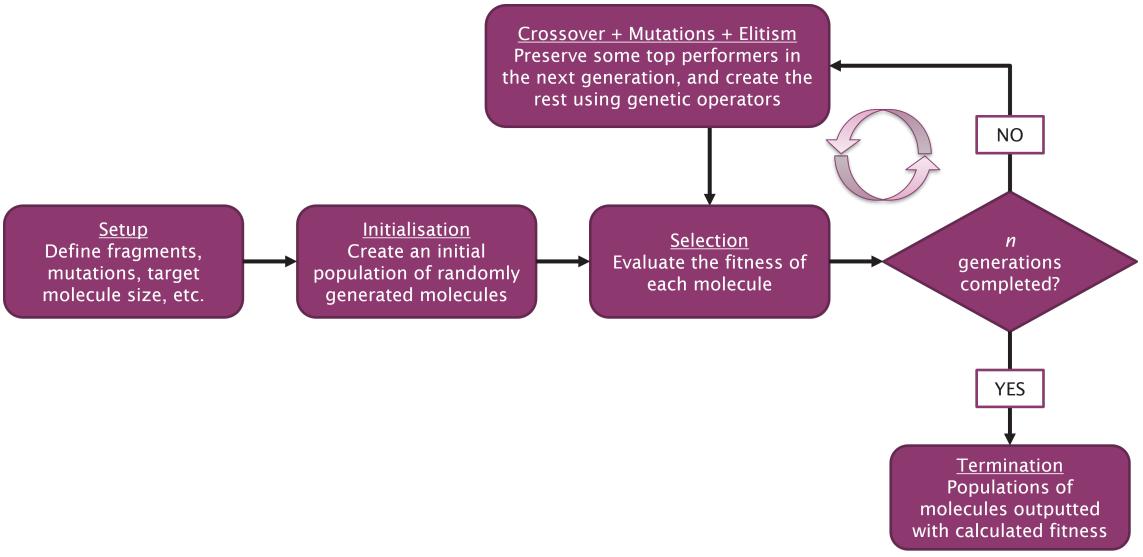



FIGURE 3.3: Flowchart depicting the general ideas behind genetic algorithms, and how they can be used with a fitness function to produce optimized populations of molecules

The molecular modifications used in MolBuilder are referred to as 'genetic operators' due to the evolutionary nature of the algorithm; it's useful to draw comparisons between the workings of MolBuilder and the metaphor of generational changes to populations, such as how the genetics of animals would change on an island over time.

If molecules are thought of as 'population members', we can treat them as if they're made up of 'genes', or molecular fragments; molecules which perform well according to the fitness function are more likely to have good 'genes'. It's beneficial to keep said 'genes', or molecular fragments, in the populations as the algorithm progresses, as on average, the population of molecules should exhibit better properties when more of these good genes are present.

### 3.2.1 Genetic operations from a chemical standpoint

Genetic operations can be thought of in terms of the modification of molecules, as outlined in Figure 3.4; in the case of MolBuilder, species are limited to fused ring systems, so 'genes' correspond to the molecular fragments resulting from breaking such fused ring systems along bonds shared between aromatic rings. Despite the small number of operations, these can be combined to produce tens of millions of molecules given a deceptively small set of molecular building blocks.

The 'genetic' idea is clearest when looking at the crossover operation, where two 'parent' molecules are fragmented into a set of four fragments; these fragments can be treated as genes, which are then swapped around and re-combined to produce 'child' species. If the parent molecules are top-performing species selected from the previous generation, the resulting 'children' carry aspects of their 'parents' structure, potentially alongside more optimal properties.

The addition operation is the simplest of the genetic operations; this effectively adds a new gene to a molecule, where the gene is a fragment either extracted from another molecule, or chosen from a user-defined list of building blocks. This is useful as part of the crossover operation, where two gene fragments need to be combined, and in

cases where good genes are not yet present in the population, or where better genes could be accessed via the addition of another fragment.

The mutation operation takes a molecule, and modifies it slightly through transformation between atom types, or the addition of side groups; this is analogous to the occurrence of random mutations in DNA. Mutations randomly occur when new molecules get generated, and are useful to 'push' the genetic algorithm in directions that aren't necessarily optimal, but could lead to 'better' populations further down the line. Random mutations also help the algorithm avoid stagnation in the populations of molecules, by performing 'nudges' away from the current species, often resulting in newly discovered population members.

Recombination is a similar operation to crossover, but utilises only one parent molecule which is split into two genes and 'recombined' in a different configuration. This is useful in cases where a molecule is already composed of good 'genes', but rearrangement to an isomer could give different fitness values; this operation is considered as an additional 'mutation' operation, and has a random chance to be applied to 'child' molecules before they get placed into a new population.

### 3.2.2 Elitism and survival of the fittest

Elitism is a concept which allows genetic algorithms to preserve the best members of previous generations when creating new populations, drawing on the idea of 'survival of the fittest'; at each iteration, before creating 'child' molecules to fill up a new population, the best percentage of molecules from the current population according to the calculated fitness function are carried over, without any modification.

This concept also applies when selecting the 'parent' molecules from which new 'child' molecules are created. For each parent in a pair, a tournament selection process decides between two members from the previous population; their relative fitness values are compared and the better parent is selected *in most cases*, with a usually small chance of the worse parent being selected instead. This is designed to match cases in evolution where the fittest population members do not always get the chance

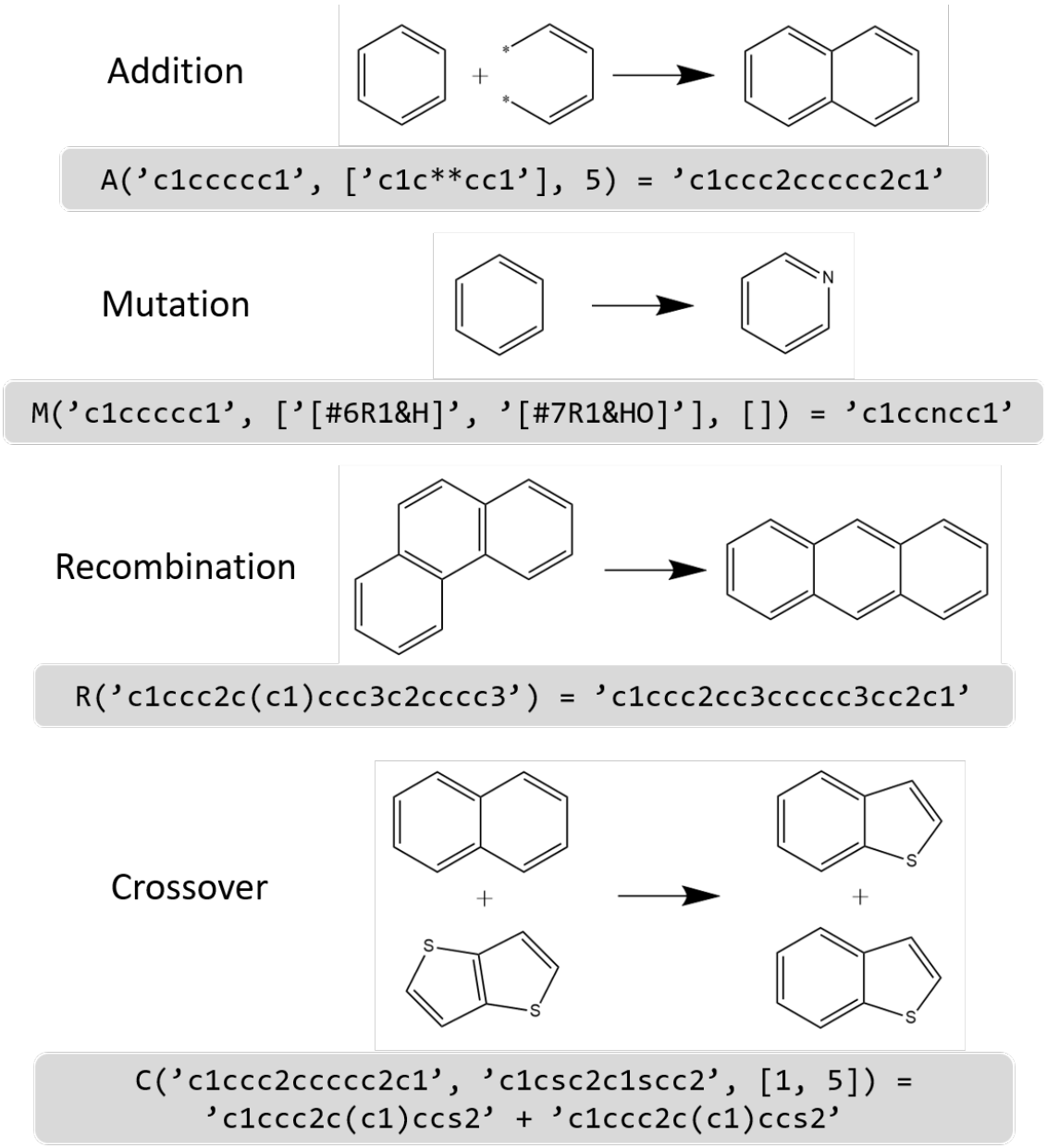



FIGURE 3.4: The molecular modification operations, or genetic operators, used within MolBuilder to generate molecules

to reproduce, and parents with 'less optimal genetics' are still able to pass their characteristics on to the next population.

### 3.2.3 Exerting more control over genetic algorithms

Changes to the setup of a genetic algorithm play a significant role in what kind of populations get produced, and how the evolution of these populations progresses; in this work, certain parameters are chosen to implement design rules on the molecules

to be constructed, and ensure a balance between chemical space exploration, where new molecules are sampled, versus the exploitation of top-performing species.

This allows for the genetic algorithms to be tailored to the specific goals of the project, including implicit consideration of synthetic difficulty through the choices of molecular building blocks and limits on molecular size, and the selection of defined initial molecule populations, meaning that several parallel genetic algorithms can be initiated from the same starting point, while focusing on fitness functions considering different properties.

### 3.2.4 Use in the context of materials discovery

The results of molecular generation through genetic algorithms can be considered as the first step in a larger materials discovery process; by performing calculations of desired physical properties as part of a fitness function to be optimized, populations of promising candidates can be generated and passed to experimental chemists for synthesis and analysis in a laboratory.

Providing results to aid experimental steps does not need to be the final contribution from computational work; given the right framework, experimental results and analysis can be fed back into the computational portion, and used to inform future explorations. This would be particularly beneficial when considering difficult-to-calculate physical properties; this joint experimental-computational process would entail experimental chemists providing feedback to inform a computational engine on which molecules exhibit promising properties, effectively acting as a fitness evaluation in the 'selection' step of genetic algorithms.

## 3.3 Combining known reactions and precursors

Given a set of reactions and compatible precursors, a library of potential reaction products can be generated by applying these reactions to supplied reagents, as

described by Patel et al. (2020) in their production of a 'synthetically accessible virtual inventory'.

This approach would be useful in cases where a genetic algorithm is not necessary, for example due to a relatively small set of molecules to explore; rather than optimizing properties to direct exploration, all possible reaction outcomes could be generated, and passed through post-hoc analysis of properties to identify the most promising candidates before in-lab experimentation.

Reactions and precursors can be modelled within a computational context through RDKit, and combined to predict potential products; for precursors which could be converted to several products, perhaps due to multiple possible reactive targets, a computational process can be executed to take note of all potential byproducts, including the outcomes of a 'partially complete' reaction, where some reactive sites are left untouched.

In terms of a collaboration with experimental teams, custom reaction product libraries can be constructed based on the reagents available, and the reaction types that could be performed. Rather than selecting a promising molecule, and requiring synthesis before analysis can be conducted to determine if it exhibits the desired behaviour, computational libraries of molecules ranked by their predicted properties can be used to prioritise the most promising candidates for synthesis.

## 3.4 Molecular modelling and simulation of the solid state

### 3.4.1 Geometry optimization

3D molecular geometries are often required before physical properties such as reorganisation energy, and the prediction of potential crystal structures, can take place. When a molecule stored in text forms such as SMILES or InChI is loaded with RDKit, it is initially generated with a set of 2D coordinates, which are used to draw flat sketches of the molecule. Initial estimates of the 3D geometry can be made through the use of a universal force-field optimization with RDKit.

These estimates can then be fully optimized with density functional theory, where the energy of a molecule in a defined geometry can be calculated based on the relative positions of each atom; atomic positions are iteratively changed based on this calculation, working to minimize the total energy and hence minimize strain between atoms, eventually settling into an optimal geometry where each atom is 'settled' as much as possible.

### 3.4.2 Crystal structure prediction

Some physical properties cannot be calculated from molecular geometries alone, and require information regarding the arrangement of molecular units in the solid state; attempts to determine these arrangements can be made through crystal structure prediction, or CSP.

The Day group maintains a Python-based crystal structure prediction software library named `CSPy`, which handles the sampling of crystal energy landscapes, leading to generation and minimization of possible crystal packing arrangements for a species given its molecular structure.

Before generation of crystal structures can occur, the molecular geometry must be optimized. In most cases, the conformation is assumed to be rigid after this point; a flexible-molecule CSP approach has recently been under development, where multiple molecular conformations are sampled alongside crystal packing parameters, at a much higher computational cost.

Given the optimized molecular geometry, trial crystal structures are quasi-randomly generated Case et al. (2016), adhering to restrictions on parameters set by the desired crystalline space groups; these space groups determine the level of symmetry required within crystal structures, according to symmetry operations such as translations, rotations and reflections. Typically, the most common set of 10 - 25 space groups is sampled.

There are several degrees of freedom to consider when sampling crystal structure parameters, from the length and angle dimensions of the unit cell, to molecular

positions and orientations; sampling through a quasi-random Sobol sequence allows for even sampling of these parameters.

Sampled structures are then checked for 'clashes' between molecules, and any unit cells where this occurs are processed through use of the seperating axis theorem, expanding the unit cell to remove any overlap between each molecular unit's convex hull; if this can be done without too much increase in the unit cell volume, the trial structure is accepted and moves on to the next phase.

Acceptable trial structures are then lattice energy minimized, allowing the molecules to 'settle' into their optimal positions relative to one another; minimization is performed through use of atomic multipoles within each molecule, generated from the optimal gas-phase molecular geometry through distributed multipole analysis, up to rank 6 (single point charges up to hexadecapoles). This process moves trial structures around on the multi-dimensional landscape of lattice parameters, locating local energy minima which indicate potential energetically stable packing arrangements. The final set of energy-minimized predicted crystal structures can then be visualised as an energy-density landscape, such as the plot of CSP results for dibenzothiophene shown in 3.5.

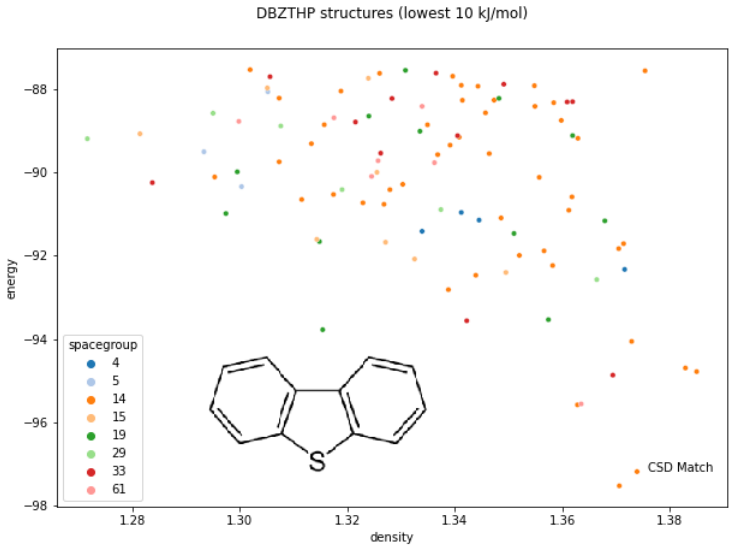



FIGURE 3.5: Example CSP results for dibenzothiophene, from a collaborative study with Villeneuve et al. (2022); plotting the energy and density of each generated crystal structure give a good summary of the species' crystal landscape.

In cases where the species already exists, the generated crystal structures can be compared to experimental data from sources like the Cambridge Structural Database (CSD). Often the lowest energy structure on a generated crystal landscape will correspond to the experimental structure, but as shown in Figure 3.5, this isn't guaranteed.

Such cases may be due to some inaccuracies in the force-field model used to perform minimizations, or where experimentally observed packing arrangements are not the product of thermodynamic processes; the lowest energy predicted crystal structure should be thermodynamically stable, but experimental results may reveal a preference for kinetic products in some systems, where a higher energy crystal forms perhaps due to a large energy barrier 'blocking' the lowest energy arrangement from forming.

### 3.4.3 Charge carrier mobility

Molecules which exhibit high charge carrier mobilities are promising candidates for use in organic semiconducting materials, where charges are able to flow between molecules easily. Charge mobility has been considered in two ways during this project; assessing election reorganisation energy through the use of molecular geometry optimizations, and the determination of electron mobility through the simulation of electron 'hopping' between molecules in predicted crystal structures.

Reorganisation energy describes the energy barrier to overcome during the process of a charge carrier moving from one molecule to another. The size of this barrier is determined by energy changes incurred through two processes; the geometry change of a charged molecule releasing its charge and returning to a ground state, and the geometry change of a neutral molecule receiving this charge.

Molecules with low reorganisation energies are targeted for use in semiconducting materials, since the energy barrier between neutral and charged states is lower; in particular, intramolecular reorganisation energies are a viable, relatively low-cost property to predict when searching for candidate species to use in this application.

In order to calculate the intramolecular electron reorganisation energy ($\lambda$), the energies of two molecular geometries at two different charge states (neutral and ionic) are combined using the following equation to calculate a reorganisation energy value:

$$\lambda = [E^-(R_0) - (E^0(R_0)] + [E^0(R_-) - (E^-(R_-)]$$

1. $E^-(R_0)$: Anion of the molecule, in the optimal geometry of the neutral form
2. $E^0(R_0)$: Neutral molecule in its optimal geometry
3. $E^0(R_-)$: Neutral molecule, in the optimal geometry of the anionic form
4. $E^-(R_-)$: Anion of the molecule in its optimal geometry

Electron mobility is predicted through the determination of charge hopping rates with Marcus theory; assuming that an electron 'hops' from a charged donor molecule to a neutral acceptor molecule sequentially, the rate at which such 'hopping' occurs ($k$) can be determined through a combination of the calculated reorganisation energy ($\lambda$), an electronic coupling term ($V$), and a chosen temperature ($T$):

$$k = \frac{|V|^2}{\hbar}\sqrt{\frac{\pi}{\lambda k_B T}} exp(\frac{-\lambda}{4K_B T})$$

The electronic coupling term $V$ describes how the electronic wave functions between molecules overlap and interact with one another, and can be calculated at reduced cost through the use of an atomic overlap model, fitted with $\pi$-conjugated organic dimers in mind as described by Gajdos et al. (2014); this models systems such as acenes and arenes with sulphur, nitrogen and oxygen heteroatoms, which covers the classes of species examined in this project.

## 3.5 Synthetic difficulty estimation

The synthetic difficulty (or the inverse, referred to as synthetic accessibility or SA) of a molecule is a measure of how easy or difficult-to-synthesise it is. It can be defined as a metric, and estimated computationally, in multiple ways based on molecular structure alone or a more in-depth retrosynthetic analysis.

### 3.5.1 Capturing the rules of organic chemistry in computational settings

Being able to store molecules/reactions and manipulate them is a brilliant step, but when working with molecules in a computational context the rules dictating how organic chemistry works aren't enforced automatically; for instance, while generated SMILES should generally be valid, and *look like molecules*, that doesn't guarantee that they are chemically feasible, let alone synthetically accessible. Prediction of synthetic difficulty can be used as a way to check the validity of molecules generated by algorithms, and act as a 'sanity check' by attempting to answer the question "could this molecule be made in a lab".

There are several ways that the idea of 'synthetic difficulty' can be interpreted, and two chemists' opinions may differ even when given the same criteria; as mentioned in the literature review, Bonnet (2012) note that when producing scores with a model based on the opinions of a group of chemists, the average of responses had to be used due to a variation in the scores assigned.

### 3.5.2 Synthetic difficulty scoring

Different views on synthetic difficulty can be considered when estimating scores, each with benefits and drawbacks; as stated in previous sections, a combination of these views could be created to extract the most information, and cover the shortcomings that each approach has.

#### 3.5.2.1 Molecular complexity

One approach to the prediction of synthetic difficulty is the analysis of molecular complexity, which can be constructed in many ways; while these methods aren't strictly related to synthetic accessibility, they can provide an estimate based on the appearance of substructures common to sets of known molecules.

One such example produced by Ertl and Schuffenhauer (2009) is named SAScore, producing scores based on fragment contributions from the PubChem database; by assuming that molecules with high similarity to fragments in a representative set of known molecules are more likely to have synthetic paths, and implementing additional penalties for more complex structural features, the synthetic difficulty can be estimated rapidly from molecular structure alone. These scores generally sit between values of 1 and 10, with anything scoring above 6 considered to be synthetically difficult.

Voršilák et al. (2020) build upon the Ertl SAScore with their scoring method, SYBA (SYnthetic Bayesian Accessibility), by basing fragment contributions on two representative sets of molecules; an 'easy-to-synthesise' set from the ZINC15 database, and a 'hard-to-synthesise' set generated by the authors acting as negative samples. These scores are based on the fragments appearing in the target molecule; the more fragments from the target that appear in the 'easy-to-synthesise' database, the higher the score, and appearance of 'difficult-to-synthesise' fragments lowers the score. In this project, scores have ranged from -70 to 70; any molecule receiving a positive score is deemed more likely to be 'easier-to-synthesise'.

If molecular generation is biased too much by structural complexity measures such as SYBA, the species generated may be composed solely with fragments which appear often in the reference database used; species made up of these fragments will outperform those with any fragments outside of those used as reference. The likelihood of discovering novel species would likely drop as a result, where, for example, genetic algorithms become 'stuck' in regions where the fragments are overly

favoured; SYBA is one of the synthetic difficulty scoring approaches implemented as a fitness function for use in genetic algorithms in this work.

#### 3.5.2.2 Computer-aided synthesis planning

Rather than using molecular complexity as a proxy to synthetic difficulty, these methods are based on the results of synthetic pathway predictions; this generally takes longer than complexity-based approaches, but is closer to the retrosynthesis process used by synthetic chemists to determine a reaction pathway to a target molecule. Since scores are based on predicted pathways, the results should be examined closely and taken more as suggestions, as predictive models are not perfect; as with any data-driven model, the results can only be as good as the data used in training.

Computational prediction of synthetic pathways often struggles to see the bigger picture, searching for the 'best' synthetic step at each iteration; a reaction which these tools may see as a 'step in the wrong direction' may lead to a more viable intermediate, resulting in an easier synthetic route which is completely missed.

AiZynthFinder is an open-source tool produced by Genheden et al. (2020) for retrosynthetic prediction, which can calculate synthetic difficulty scores based on the synthetic paths it generates. Predictions are based on a Monte Carlo tree search guided by a pre-trained neural network policy, iteratively breaking down the target molecule based on a library of known reactions. The tree search continues until a set number of reaction steps have been taken, a time-limit is hit, or a set of precursors marked as purchasable are found.

The set of reactions available to the algorithm, and a database of 'purchasable' precursors, are defined before retrosynthetic analysis can begin; the defaults provided by AiZynthFinder's developers are the USPTO reactions (Lowe (2012)) and the ZINC database of precursors (Sterling and Irwin (2015)), which can be swapped for alternative sources if desired.

After retrosynthetic analysis has completed, the generated pathways can be visualised. A given molecule may have had several pathways generated, so the

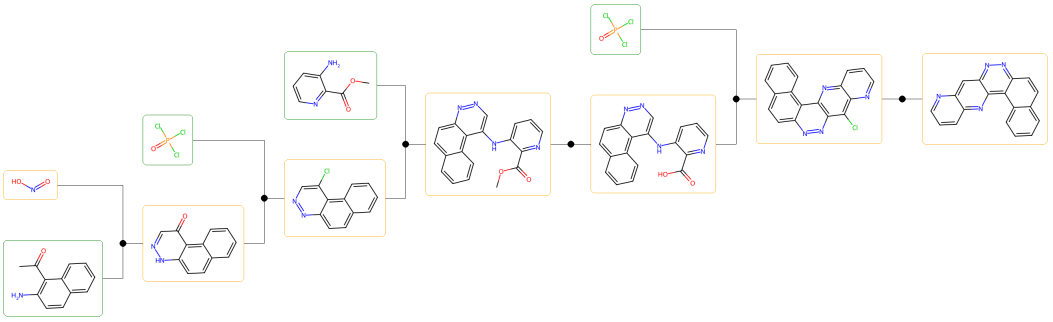


FIGURE 3.6: Example synthetic pathway to an aza-substituted pentacene molecule, predicted with AiZynthFinder

top-scoring pathway is selected by default. AiZynthFinder defines a default score to predicted routes based on the number of reaction steps taken, and the availability of precursors. Pathways with many steps, or with any precursors not present in the database, achieve lower scores than those with fewer steps/all precursors available; scores are mostly reliant on the fraction of discovered precursors which are marked as 'in stock', with a small contribution from the number of reaction steps.

$$AiZynthScore = 0.95 * \frac{n\_in\_stock}{n\_precursors} + 0.05 * max\_transforms$$

Over-biasing molecular generation with retrosynthetic-based synthetic difficulty tools may lead to the algorithm getting trapped in regions of chemical space accessible with reliable syntheses. Novel structures may require synthetic steps which are either not present in the literature, or simply not considered by the synthetic difficulty metric. Rather than implementing AiZynthFinder into the molecular generation process for this project, it has been used as a post-hoc analysis tool to predict and score synthetic pathways to molecules sampled with genetic algorithms.

#### 3.5.2.3 Reaction network-based scoring

Coley et al. (2018) use reaction data extracted from the Reaxys database to train a neural network model in synthetic difficulty prediction, generating SCScores. The model estimates synthetic complexity on the assumption that:

> *"On average, the products of published chemical reactions should be more synthetically complex than their corresponding reactants"*

For each reaction, an inequality constraint is applied to the chemicals present, stating that reagent complexity should be lower than product complexity; this concept is shown in Figure 3.7. Processing over 10 million syntheses from Reaxys this way allows the model to estimate synthetic difficulty of unseen species informed by historical trends in the database, in the context of known molecules used within known reactions.

Coley et al. (2018) write that because synthetic complexity should be nonlinear with respect to structure, simpler model architectures that may have enabled more straightforward interpretations were not explored; this reinforces earlier points made regarding the use of fragment-based molecular complexity scoring as a proxy for synthetic difficulty.

Information on starting material availability is also learnt by the model; rather than providing a stocklist, structures which tend to be present as reactants rather than products are perceived as readily available. The authors state that this causes discrepancies with previous synthetic difficulty scores to arise, since molecules typically not considered precursors are labelled as such.

SCScore values range between 1 and 5, where molecules scoring highly are considered to be synthetically complex, or more similar to the products of reactions than reactants. This is one of the synthetic difficulty scoring approaches used in this project, to be optimized during the generation of molecules with genetic algorithms.

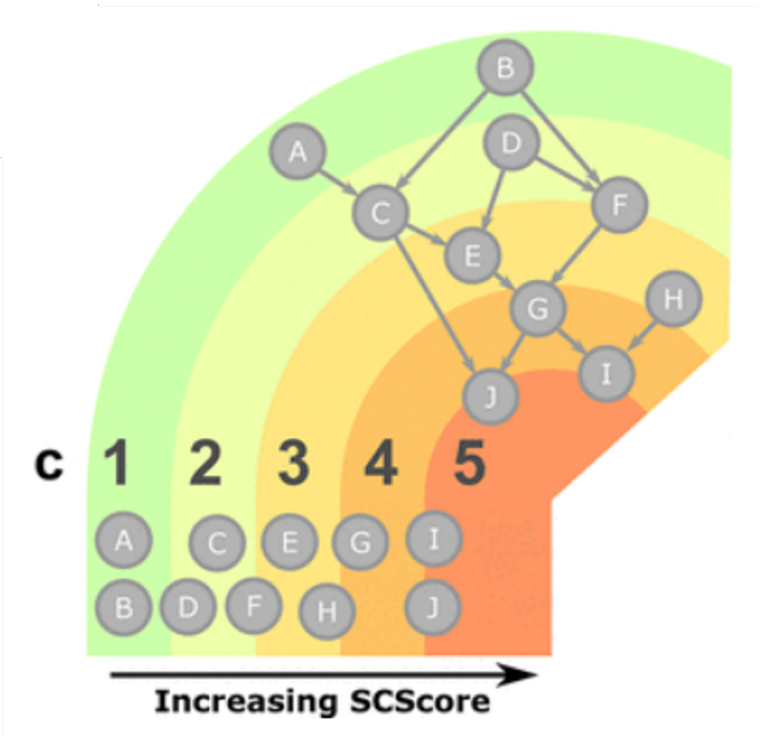



FIGURE 3.7: Diagram from the work of Coley et al. (2018) to elaborate on the idea which drives SCScores; as molecules become more similar to the 'products', deeper into a reaction network, scores become larger.

# Chapter 4

# Code Development

In order to execute the theory described in the last chapter, several existing computational tools were implemented into, or used alongside, the Day Group's code base, `CSPy`; the code used to run `MolBuilder` also underwent significant development over the course of this project. The implementation and use of computational methods will be described in this chapter, which will elaborate on the work done to produce results with these tools.

## 4.1 Working with molecules and reactions in Python

The Python library RDKit provides the base functionality for most of the manipulation of molecules, conversion between and storage of various molecular representations this project deals with. This section will discuss how this library and others have been utilised to execute the theory discussed in the previous section.

### 4.1.1 Conversion between representations

The flexibilty of RDKit molecules shines through here; once a valid Mol object has been created from one of many different representations, it can be manipulated using RDKit functionality and/or converted to one of several other representations if needed.

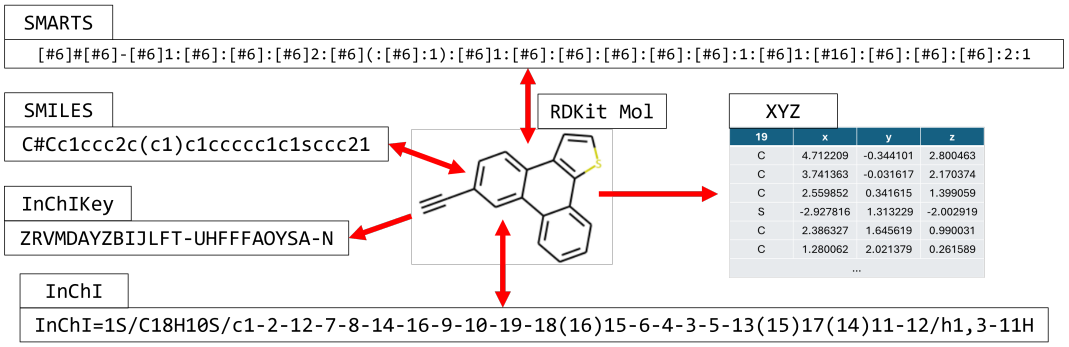



FIGURE 4.1: Showcasing the ability to convert between molecule representations, with RDKit `Mol` objects acting as a bridge between formats.

As shown in Figure 4.1, most conversions between formats are reversible, with the only exceptions being InChIKeys and XYZ files; InChIKeys are generated by performing irreversible hashing on the corresponding InChI, while XYZ files store only the coordinates and types of each atom in a molecule, without any information on bonding types and connectivity between them. The code in this project makes use of several representations for different tasks, and the chosen representations changed in some cases as the work progressed.

### 4.1.2 Defining building blocks

Molecular building blocks to be used in genetic algorithms were defined using SMILES for the ring types, and SMARTS for the side group types. These were chosen to make configurations more user friendly, as these representations are more human-readable than others. SMILES worked well for the definition of ring types, as single rings are relatively easy to define.

For the side group types, SMARTS made a better choice; as explained in a later section, by using reactionSMARTS for genetic operators, defined mutation SMARTS patterns could be appended to the end of a partial reactionSMARTS string, producing the relevant 'full' mutation reaction on-the-fly.

### 4.1.3 Storing molecules

At first, molecules were stored as InChI, making use of the behaviour that these exhibit where one InChI relates to only one molecule. This choice was changed to SMILES strings when the approach to editing molecules was modified (described later), and to make the inputs and outputs for MolBuilder consistent; building blocks are supplied as SMILES, so it makes sense for the output molecule populations to be stored as SMILES too.

Additionally, RDKit automatically 'sanitizes' molecules when converting between the Mol object and SMILES, ensuring that the bonding patterns make sense in the generated SMILES output.

In order to ensure that SMILES can be checked against each other for uniqueness, the RDKit function `Chem.CanonSmiles()` was utilised; a given molecule can have more than one valid SMILES string, and this function will return the same SMILES for a molecule given any valid options.

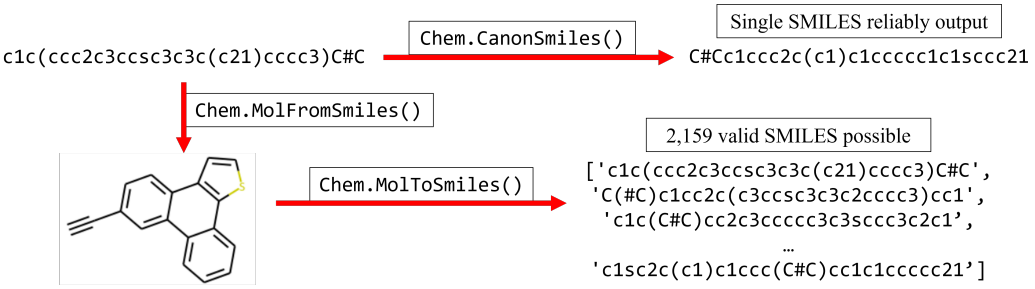



FIGURE 4.2: Utility of canonical SMILES generation to avoid duplicate molecules.

### 4.1.4 Ordering calculation files

For tasks where calculations were performed on larger sets of molecules, and required structured directories to keep files organised, the InChIKey worked perfectly. In order to access the relevant folder for a given molecule, the InChIKey can be generated on-the-fly from other representations. There are very few cases, if any, where a regular user of the software would require the reverse operation (i.e. taking a folder of results

and finding the corresponding molecule), so the inability to convert back from InChIKey to other formats would not be an issue.

### 4.1.5 Substructure searching and pattern matching

Given a fragment, or substructure, of a molecule, a SMARTS pattern can be written and utilised with RDKit's `Mol.GetSubstructMatches()` function to detect any occurrences of that pattern within an RDKit molecule (Mol). There are many cases where this functionality is useful within the project, mainly:

- Detecting substructures which should be excluded, such as rotatable bonds, large side groups, etc.
- Counting the number of times a substructure, such as a ring or pair of rings, appears within a molecule
- Screening a large set of molecules to find the subset which contain a reactive center (such as ortho-dianiline).

### 4.1.6 ReactionSMARTS and RDKit reactions

As described in the theory chapter, reactions can also be encoded as text, using reactionSMARTS. A typical reactionSMARTS string is shown in Figure 4.3, where reactant patterns are on the left and product patterns are on the right, separated by two 'less than' symbols (>>). Only the reacting atoms need to be encoded, along with connected atoms if relevant to the operation taking place.

As in Figure 4.3, reacting atoms can be labelled with numbers, which are tracked between reactants and products; this allows for the defined movement of specific atoms, and for operations such as connecting atoms or creating ring structures.

ReactionSMARTS can be used in conjunction with RDKit's `AllChem.ReactionFromSmarts()` to generate a `Reaction` object, which can then be provided with compatible RDKit `Mol` objects to apply the reaction to a molecule with

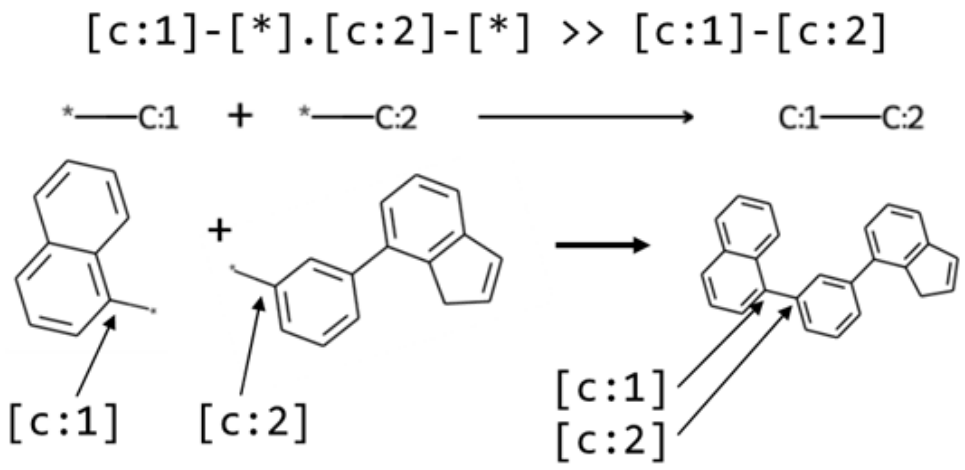



FIGURE 4.3: Example usage of reactionSMARTS; two unconnected carbon atoms with 'attachment points' can be labelled (carbon 1 and carbon 2), then attached, using careful definition of the reactants and products in a reactionSMARTS string.

`Reaction.RunReactants()`. `RunReactants()` applies the reaction to the molecule(s) in all possible ways, and returns a list of possible reaction outcomes.

As an example (see Figure 4.4), a fluorination 'reaction' can be considered as converting an aromatic carbon with hydrogen attached into an aromatic carbon with fluorine attached; this can be encoded as reactionSMARTS and applied to pyridine using RDKit. There are five possible carbon positions which match the 'reactants' portion of the reactionSMARTS string (`[c&H1:1]` = aromatic carbon with one hydrogen), meaning that there are five products produced. Two of these products are duplicates, as they can be related to other products via symmetry. This is dealt with by conversion to a unique text representation such as InChI, and removal of duplicate entries; leaving the three unique reaction outcomes, i.e. substitution at ortho, meta and para positions.

ReactionSMARTS and RDKit's reaction functionality form the basis of the new molecular operations used in MolBuilder, which will be described in the following section.

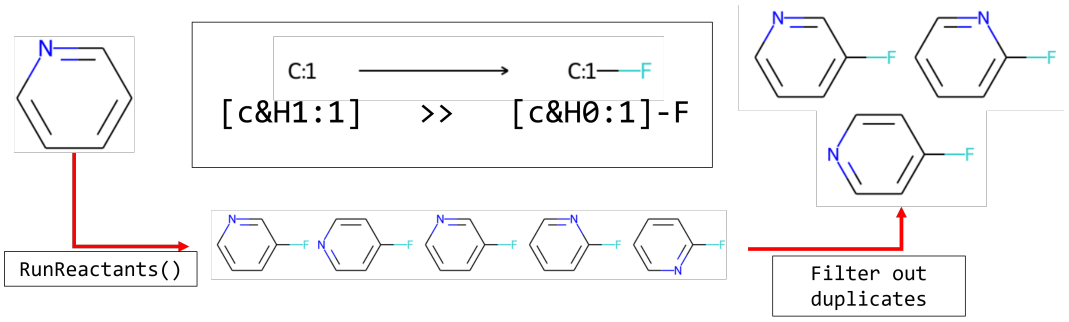



FIGURE 4.4: Running reactions with RDKit - after running the reaction, all outcomes including 'duplicate' products related by symmetry are returned. These outcomes can be filtered by conversion to InChI, and checking for duplicated InChI strings.

## 4.2 MolBuilder - an evolutionary algorithm for chemical space

As mentioned in earlier sections, MolBuilder is a genetic algorithm designed to explore defined chemical spaces with a bias optimizing target properties via fitness functions Cheng et al. (2020). Development of this software was taken over from a previous group member (Chi Cheng) when the project started, aiming to make changes and additions to increase the code's readability and functionality.

MolBuilder is currently limited to working with systems of fused aromatic rings, optionally including substituent side groups, which allows the exploration of a large class of molecules covering many potential organic semiconductor species; by restricting MolBuilder to these types of molecule, the variation in each genetic operator's targets is dramatically reduced.

### 4.2.1 Setting up a genetic algorithm and storing results

Before a genetic algorithm is started a configuration file is required, which instructs MolBuilder on how the algorithm runs; this covers many settings, parameters, and the types of 'building blocks' which can be used to generate molecules. A selection of important parameters are outlined in Table 4.1.

By setting up genetic algorithms in this manner, several 'linked' MolBuilder runs can be set up and run in parallel with ease; this is useful in cases where many different starting populations are to be used with the same settings, as the initial set of

| Parameter | Meaning | Example Input |
|---|---|---|
| molecules | Ring types | ['c1ccccc1', 'c1ccncc1', 'O=c1ccccо1'] |
| mutations_1 | Side groups | ['[c:1]-F', '[c:1]-Cl', '[c:1]-C#C'] |
| generations | Initial population | [{0: ['Fc1cncc2ncncc12', ...]}] |

| Parameter | Meaning | Ex. Input |
|---|---|---|
| molsize | Min/max ring count allowed | [2, 5] |
| unique_ring_max | Max unique ring types | 3 |
| unique_mutation_max | Max unique side group types | 2 |
| total_generations | Number of generations to run | 50 |
| population_size | Population size | 100 |
| calculation_type | Fitness function to use | input |
| maximize | Maximise fitness function or not | False |
| elitism_population_size | Number of elites to carry over | 10 |
| mutation_rate | Chance of mutations occuring | 0.05 |
| tournament_win_rate | Chance of picking 'better parents' | 0.75 |
| carrier | Pick charge carrier (if needed) | electron |
| method | Pick DFT method (if needed) | B3LYP |
| basis_set | Pick basis set (if needed) | 6-311G** |

TABLE 4.1: Table describing some important parameters to be set by a configuration file before running a MolBuilder genetic algorithm. An example configuration file can be found in the appendix.

molecules can be changed by editing `generations` in a copy of the original configuration file.

This format is also utilised to store the outputs of a genetic algorithm, namely the generated populations and their calculated fitness scores, and is kept updated as the algorithm progresses; if the total number of generations is not met when the algorithm stops, for example due to a runtime limit, then it can be restarted using this file to 'pick up where it left off'. Storing results in this manner also preserves the original settings, meaning the configuration and the resulting populations of molecules are linked if either set of information is required for future use.

### 4.2.2 Defining building blocks

In previous iterations of the MolBuilder code, the building blocks used to construct molecules had to be defined with attachment points, which dictated how a ring could be attached to the current working molecule. This approach could be detrimental in

cases where users did not define all possible ways that a given ring could be attached, and leads to lengthy lists of building blocks if multiple ring types are to be used.

During this project an alternative approach was developed, named 'frags-in-situ', which avoids such issues by automating the selection of attachment points. Molecular ring building blocks are defined as SMILES strings without any attachment points, and get passed through the `get_frag()` function before use; Figure 4.5 shows how this function works.

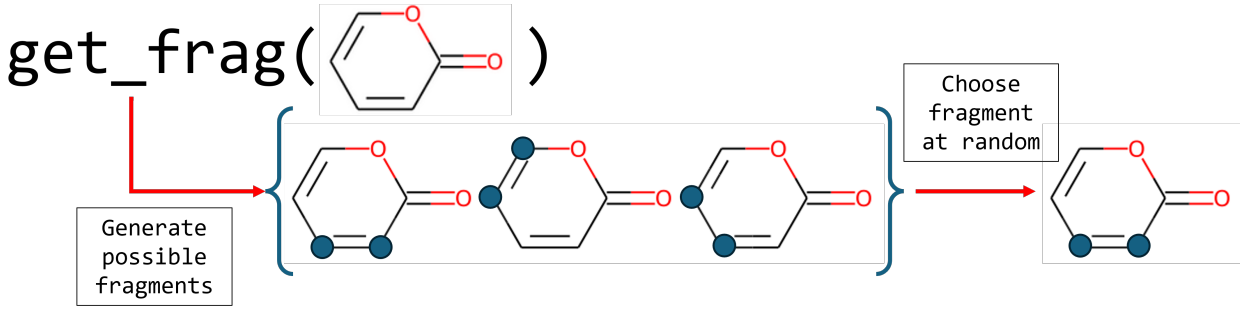



FIGURE 4.5: The function used to generate fragments from a ring building block

Given an aromatic ring, the `get_frag()` function uses a reactionSMARTS string to identify all aromatic carbon-carbon bonds, and convert one randomly selected option into an attachment point. The function was written this way to avoid removal of heteroatoms from aromatic heterocycles, which would modify the building block from what was originally defined.

Generating the attachment points this way removes some steps in the setup of genetic algorithms, making the task easier particularly for users unfamiliar with the code; but it should be noted that there is a loss of control associated with this approach, since attachment points are chosen at random. In order to provide options to users, the 'original' approach where fragments are supplied with pre-defined attachment points is also available.

### 4.2.3 ReactionSMARTS for genetic operations

Previously, MolBuilder explicitly defined atom indices within a molecule to run genetic operations using various RDKit functions:

- Identified target attachment points by substructure matching with SMARTS, which were stored as atom indices, and marked within fragments using dummy atoms;
- The `RDKit.Chem.CombineMols()` and `Chem.RWMol()` functions were used to manually attach a fragment to the working molecule;
- Manual addition of bonds and removal of dummy atoms by exploration around these atoms and their neighbours;
- This was followed by a check that the molecule was 'chemically sound' with `Chem.SanitizeMol()`, then double checked this clean-up by conversion to and from SMILES;
- The RDKit Mol object was passed between operations.

#### 4.2.3.1 Using RDKit reactions to modify molecules

As stated earlier, RDKit contains functionality to parse reactionSMARTS strings and run reactions; by encoding the genetic operators as reactionSMARTS, the above steps can be 'skipped' and coded in a much neater, more human-readable manner.

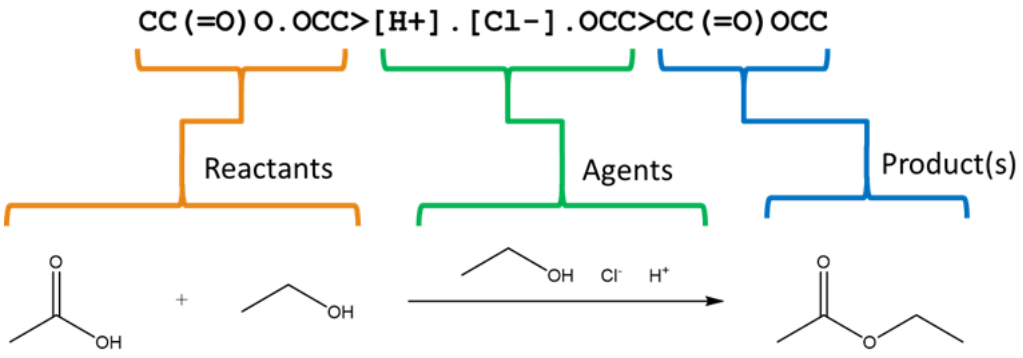



FIGURE 4.6: Example reactionSMARTS string, encoding the 'start point', 'agents' needed to facilitate the reaction, and the 'end point'

In the context of molecular operations such as the genetic operators required for MolBuilder, reactionSMARTS strings can take the following form:

```
pattern_to_modify >> modified_pattern
```

Where `pattern_to_modify` refers to a SMARTS string identifying which part of the molecule to target for changes, and `modified_pattern` is the intended result. These patterns can contain atom-atom mapping information; if a given atom present in the original molecule is also present in the desired output, it can be labelled to ensure it stays in place.

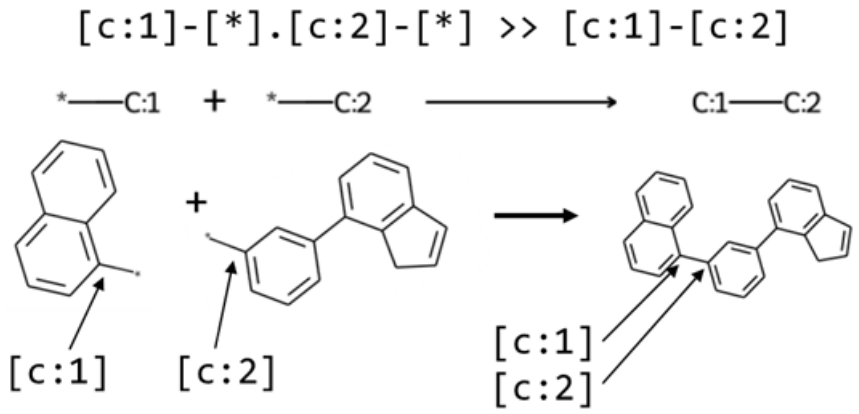



FIGURE 4.7: The importance of atom mapping in an exemplar single-point recombination

In the example shown by Figure 4.7, two fragments are combined at their attachment points (labelled with $-[*]$), and the atoms neighbouring attachment points are conserved in the final result. Atom mapping becomes more important when adding fragments at more than one point in the target molecule; for two point addition (addition along a bond as opposed to single atoms), atom mapping ensures that the connectivity of the original molecule is maintained once the new fragment is added.

As a more relevant example for this project, the reactionSMARTS string in Figure 4.8 effectively identifies two cyclic carbons bonded to one another using the pattern `[cH,CH1;r5, r6:1][cH,CH1;r5, r6:5]`, then replaces these atoms with aromatic carbons (**[c:1][c:2]**) while adding the rest of the benzene fragment. The 'reactant' pattern can be broken down to explain its meaning:

- Must be in a 5-membered or 6-membered ring (`r5, r6`)
- May be aromatic or not, but must only be bonded to one hydrogen atom (`cH,CH1`)

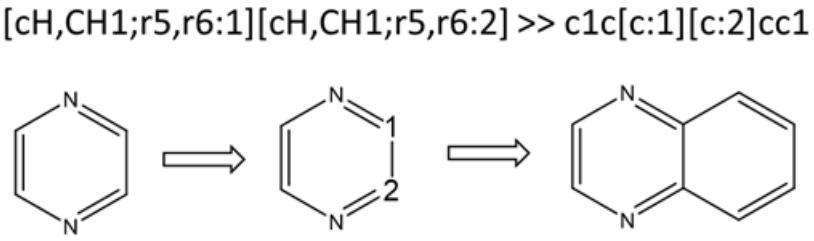



FIGURE 4.8: Atom mapping in reactionSMARTS for an exemplar two-point addition

By replacing the original genetic operators with this reactionSMARTS based approach, most of the 'heavy lifting' in the molecular modification process can be handed to RDKit functions such as `AllChem.ReactionFromSmarts()` and `rxn.RunReactants()`.

In the above example (Figure 4.8), there are actually two possible results as the fragment could be added at two positions: those marked 1,2 and those on the opposite side of the ring. While the end result is the same for this example this is not always the case, and as stated before RDKit outputs all possible outcomes, including duplicates, so an additional duplicate filtering step should be used to ensure only unique outcomes are considered.

#### 4.2.3.2 Fragmenting molecules and adding attachment points

Most genetic operations require a fragmentation step, and the selection of attachment points on such fragments which dictate how they should be combined to form a new molecule. This is handled with a pair of reactionSMARTS patterns, which target any aromatic carbon-carbon bond shared between two rings. Two patterns were required in order to accomplish this task; one to fragment along a bond shared between two 6-membered rings, and one to fragment the bond between a 6-membered ring and a 5-membered ring.

```
def fragmentation(smi):
    break_2pt_6m = # Two point fragmentation SMARTS pattern 6M rings
    break_2pt_5m = # Two point fragmentation SMARTS pattern 5M rings
    # Generate RDKit Mol object
    mol = Chem.MolFromSmiles(smi)
    # Initialise the 6M ring fragmentation reaction, and run it
    rxn = AllChem.ReactionFromSmarts(break_2pt_6m)
    result_6 = rxn.RunReactants((mol,))
    # Initialise the 5M ring fragmentation reaction, and run it
    rxn = AllChem.ReactionFromSmarts(break_2pt_5m)
    result_5 = rxn.RunReactants((mol,))
    # Combine the outcomes from 6+6/5+6 fragmentations
    result = result_6 + result_5
    # Select one outcome pair at random, or flag no outcomes found.
    output1, output2 = choice(result) if len(result) > 0 else (False, False)
    # Output fragments as SMILES, or flag no outcomes found
    if output1 and output2:
          return Chem.MolToSmiles(output1), Chem.MolToSmiles(output2)
    else:
        return False, False
```

LISTING 4.1: Function to fragment a molecule along an aromatic bond, and return two fragments. One fragment retains 'attachment points', marking where it was originally connected to the other fragment.

Given a molecule to fragment, the `fragmentation()` function (Listing 4.1) uses these reactionSMARTS patterns to perform all possible fragmentations, and selects one resultant pair of molecular fragments to output.

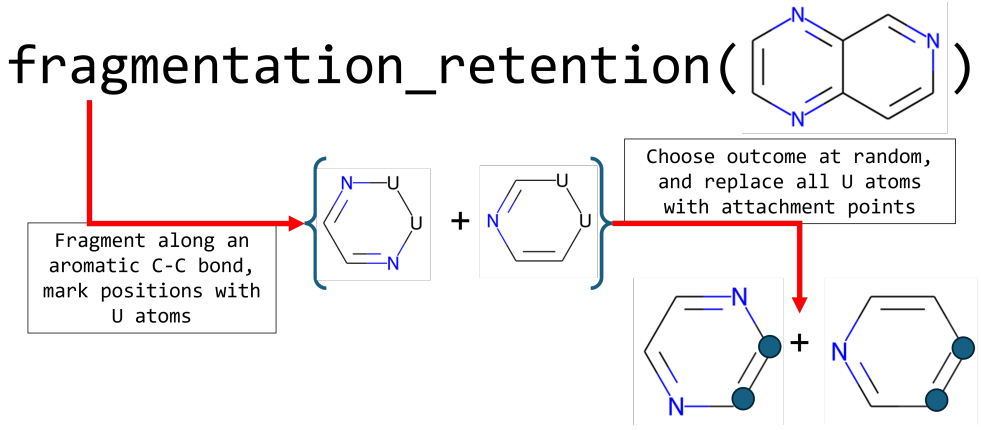



FIGURE 4.9: Diagram representing the function `fragmentation_retention()` on an example molecule

There are two versions of this function; one returns both fragments with their original fragmented bond position marked as attachment points (Figure 4.9), while the other returns one fragment with attachment points, and one without (Figure 4.10). Both variations of this function are utilised by different genetic operators, dependant on if both sets of attachment points need to be conserved during the operation or not.

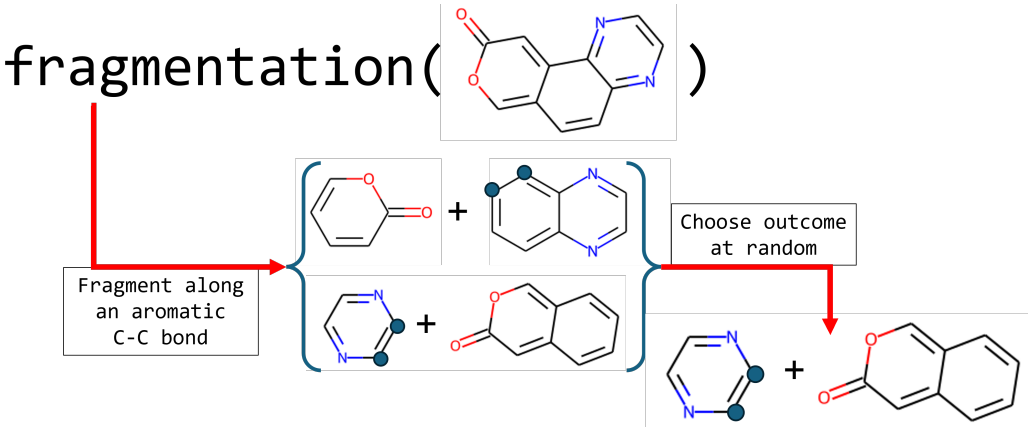



FIGURE 4.10: Diagram representing the function `fragmentation()` on an example molecule

#### 4.2.3.3 Adding a ring to a molecule

The simplest genetic operator, addition of a ring/fragment to a molecule, takes advantage of the reactionSMARTS syntax by starting with a partial reaction string (Listing 4.2, `rxn_string` variable), which contains only the 'reactants', or target molecular substructure; any pair of aromatically bonded carbon atoms, which are in a single ring only, and have hydrogen atoms attached.

The fragment to be added (`frag_smi`) must have attachment points marked with asterisks, which are converted into labelled carbon atoms. This modified fragment can then be appended to the partial reactionSMARTS string, generating a full reaction 'in-situ'; this 'reaction' can be described as targeting an aromatic CH-CH bond, and converting that bond into a fused ring bond, shared between the working molecule (`smi_`) and the fragment.

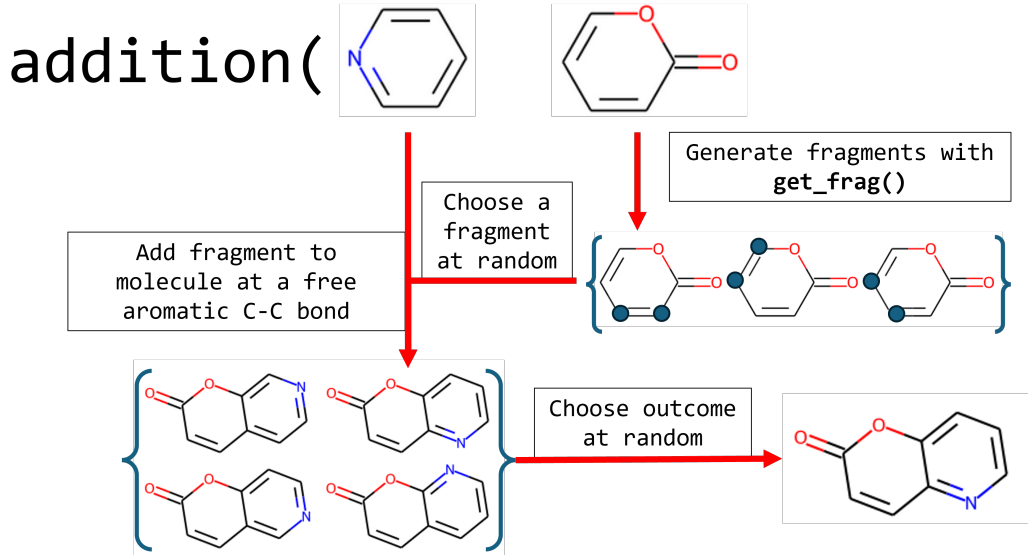



FIGURE 4.11: Diagram representing the function `addition()` with an example molecule and fragment pair

This reaction is then applied to the working molecule with RDKit's reaction functionality, returning all possible outcomes. After filtering out duplicate outcomes, a single molecule is chosen at random to be outputted.

```
def addition(smi_, frag_smi_):
    # Partial reactionSMARTS string
    rxn_string = '[cH,CH1,CH2;r5,r6;R1:1][cH,CH1,CH2;r5,r6;R1:2]>>'
    # Convert frag attachment points to labelled atoms
    frag = smiles_fragment_to_rxn_smiles(frag_smi)
    # Generate RDKit Mol object
    mol = Chem.MolFromSmiles(smi)
    # Generate the RDKit Reaction object
    rxn = AllChem.ReactionFromSmarts(rxn_string + frag)
    # Run the reaction on the working molecule
    result = rxn.RunReactants((mol,))
    # Filter duplicate results out with conversion to CanonSMILES
    smis = list(set([Chem.MolToSmiles(x[0], isomericSmiles=True)
                     for x in list(result)]))
    # Select one of the unique outcomes at random and output it
    output = choice(smis) if len(smis) > 0 else smi
    return output
```

LISTING 4.2: Function to add a fragment to a molecule

The `addition()` function also appears in other genetic operators where any molecule/fragment pair need to be combined; `fragmentation()`, `fragmentation_retention()` and `addition()` are re-used as nested functions in `recombination()`, `crossover()` and `single_parent_crossover()`, as described in the following sections.

#### 4.2.3.4 Mutating molecules to add or remove side groups

Mutation operations also utilise the syntax of reactionSMARTS to allow flexibility in the function, where a target substructure and intended mutation type are combined as strings to form a full 'reaction'.

```
def mutation(smi, mutation, targets):
    mol = Chem.MolFromSmiles(smi)
    # Pick a random group to target (aromatic CH for side group addition)
    pattern = choice(targets)
    # Create reactionSMARTS string from chosen pattern and mutation
    mutation_string = pattern + '>>' + mutation
    rxn = AllChem.ReactionFromSmarts(mutation_string)
    # Run the reaction, and select a unique outcome at random to output
    outcomes = [i[0] for i in rxn.RunReactants((mol, ))]
    filtered_outcomes = list(set([Chem.MolToSmiles(x) for x in outcomes]))
    return choice(filtered_outcomes)
```

LISTING 4.3: Function to perform 'mutations' on a molecule, adding side groups to a molecule at aromatic CH positions, or removing side groups.

In order to add a side group, `mutation()` operates on any aromatic CH position on the molecule with the SMARTS target `[cH:1]`, and performs a 'reaction' to convert it into a mutation chosen at random from the list of options defined in the configuration `mutations_1`.

To perform the reverse operation and remove a side group, `mutation()` will instead target any side groups which matches the pattern chosen at random from `targets`, and convert the chosen side group back into a C-H bond. This functionality is used in 'smarter' mutation operations, where side groups currently present in the molecule are detected.

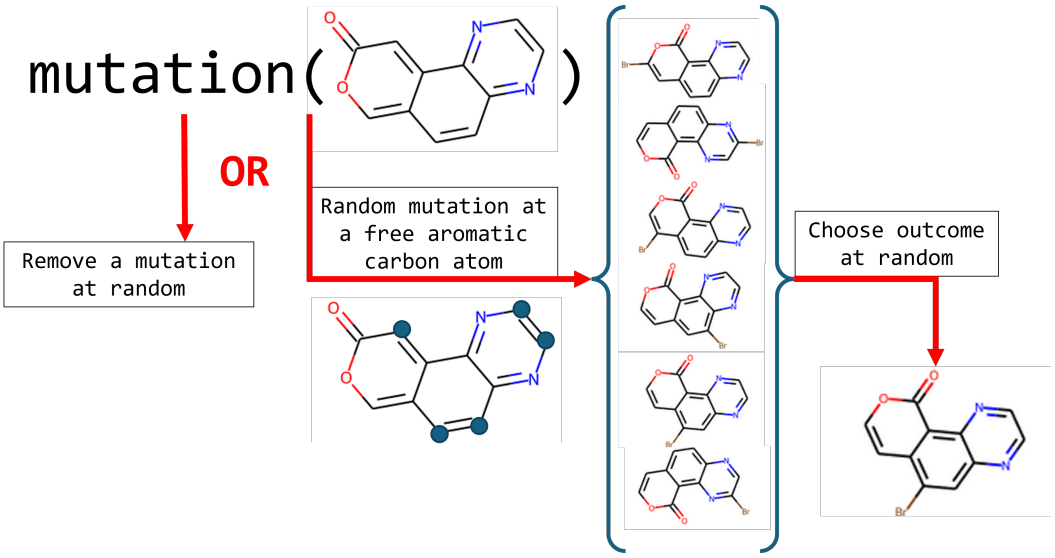



FIGURE 4.12: Diagram representing the `mutation()` function with an example molecule. Random mutations are chosen from options defined in the configuration file.

#### 4.2.3.5 Creating molecules from building blocks

In order to generate a set of molecules to be used as the initial population for a genetic algorithm run, the `addition()` and `mutation()` operations are required, along with a few parameters set in the configuration file.

For each molecule to be generated, counts of rings and side groups are chosen at random between limits set in the configuration; rings are combined one-by-one using `addition()` until the chosen ring count is achieved, followed by side group addition with the `mutation()` function.

In order to ensure avoidance of duplicate molecules within initial populations, generated molecules are compared to the current population before being added, with repeated species being discarded; the flowchart in Figure 4.13 outlines the process undertaken to generate initial populations for use in MolBuilder genetic algorithms.

For the purposes of this project, initial populations were created outside of the genetic algorithm run, to allow for repeat runs with different settings from the same starting point. If needed, MolBuilder has the functionality to generate a random initial population from the building blocks defined in the configuration file.

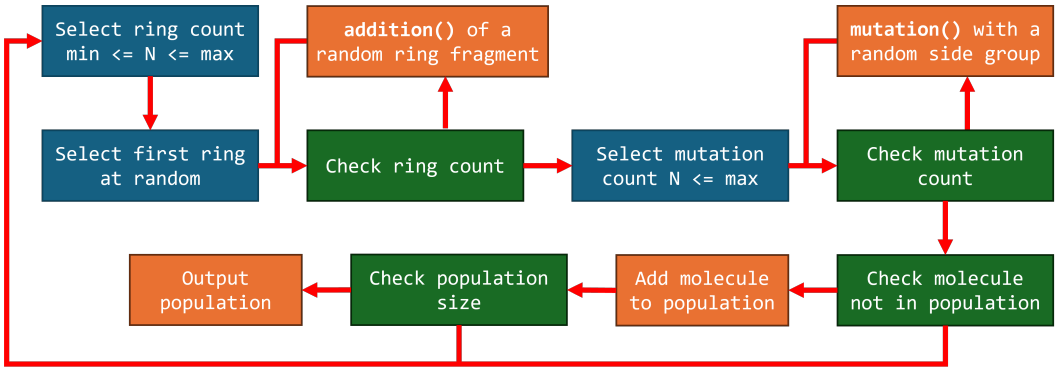



FIGURE 4.13: Flowchart describing the process of generating a population of molecules, given a set of ring types and side groups.

Only initial populations are generated in this manner; further populations are created through the use of elitism, recombination, crossover and new mutations; additional operators will be outlined in the next sections, followed by a description of the approach used to create a new generation from the previous population.

#### 4.2.3.6 Performing recombinations

The next genetic operator function, `recombination()`, takes a single molecule, fragments it into two genes, and recombines them in a different configuration. This is achieved with a combination of the functions `fragmentation()`, `get_frag()` and `addition()`.

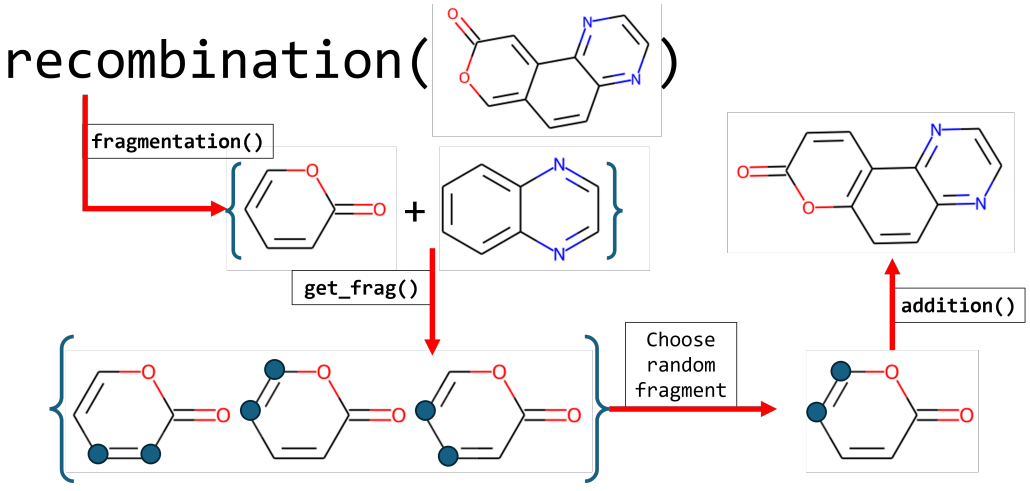



FIGURE 4.14: Diagram representing the function `recombination()` on an example molecule

When performing recombination, the attachment points on one fragment are preserved, while the other fragment has a new attachment point selected at random. New attachment points are selected for one of the fragments to allow more variation in the possible outcomes of a given recombination operation, while preserving some information from the other fragment.

#### 4.2.3.7 Crossing over two parent molecules

In order to achieve crossover operations, two 'parent' molecules must be fragmented, the resultant genes get swapped, and the two 'child' molecules are generated by recombining the fragments. In this case, `addition()` is not used, as attachment points are retained for both fragments of each parent molecule through the use of the function `fragmentation_retention()`; this choice was made to preserve more information from each parent's pair of genes.

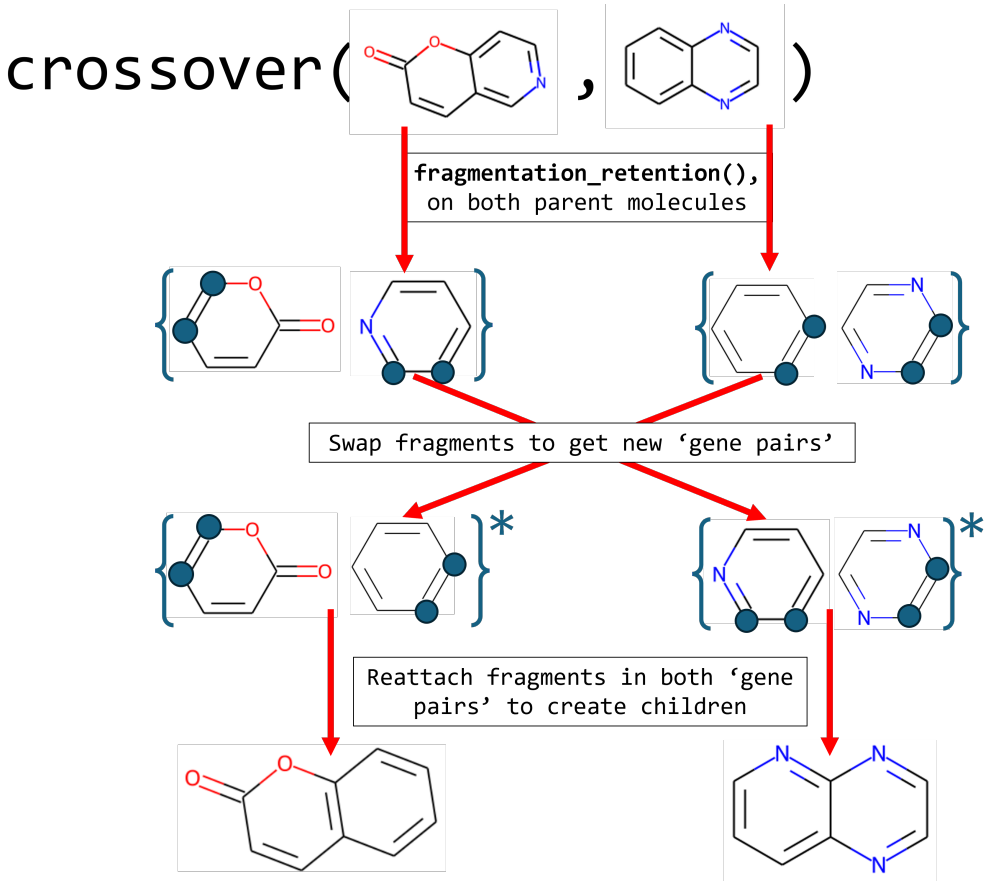



FIGURE 4.15: Diagram representing the function `crossover()` with an example pair of 'parent' molecules, producing two 'child' molecules.

Instead, a specialised reactionSMARTS pattern is used which targets two sets of attachment points, one on each fragment, and combines those positions while converting them to aromatic carbon atoms shared between the two ring fragments. Figure 4.16 shows a representation of this operation, with an example pair of fragments.

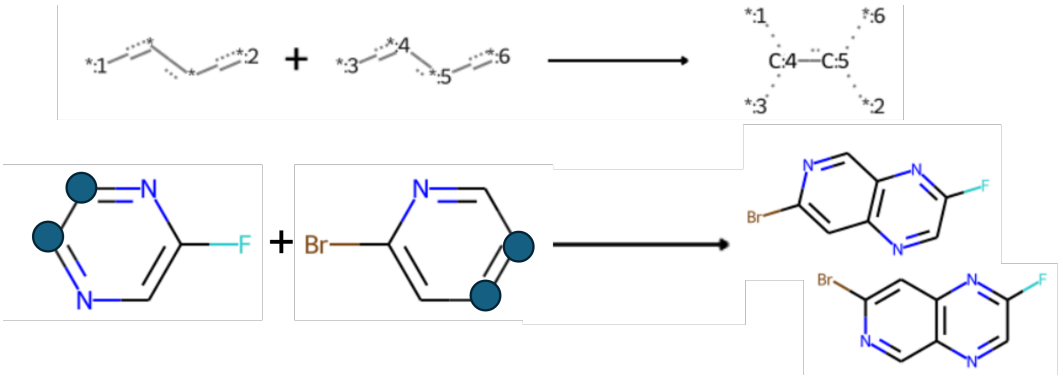



FIGURE 4.16: Diagram representing the specialised reactionSMARTS pattern used to combine two fragments with preserved attachment points, including an example depicting the possible outcomes.

#### 4.2.3.8 Creating the next generation of molecules

At each iteration of a MolBuilder run, a new population of molecules is generated based on members of the previous population; the new population can be thought of in two portions: a selection of elite, top performing molecules directly carried over from the previous population without any modification, and the remainder created from randomly chosen 'parent' molecules in the previous population.

The elite population is selected according to calculated fitness values for each molecule; the previous population is ordered by their fitness, then the top N molecules are chosen and added to the new population, where N is dictated by `elitism_population_size` in the configuration file.

New molecules are then generated from pairs of molecules picked from the previous generation using `recombination()`, `crossover()` and `mutation()`; these 'parent' molecules are chosen by a tournament selection process.

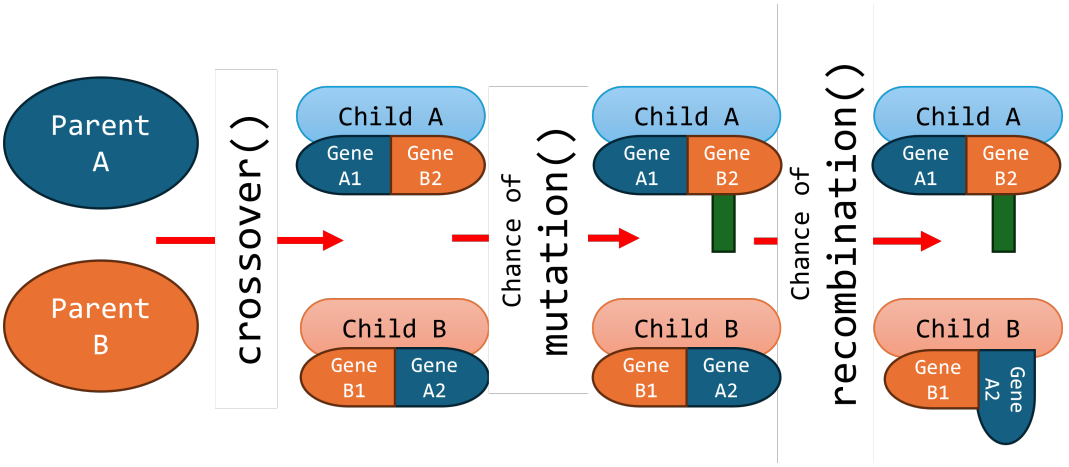



FIGURE 4.17: Diagram showing the process by which two 'child' molecules are created from two 'parent' molecules.

To select each parent, two molecules are chosen at random from the previous population, and their fitness values are compared. The configuration parameter `tournament_win_rate` determines if the better parent is chosen; a random number is uniformly sampled between 0 and 1, and if this is smaller than the defined tournament win rate, the better parent is selected.

Once a pair of parent molecules are found, `crossover()` is applied to generate two 'child' molecules, after which `mutation()` and `recombination()` each have a chance to be applied in turn, according to parameters chosen in the configuration (such as `mutation_rate`); again, these rates are determined based on comparison between the chosen value and a random number uniformly sampled between 0 and 1. This process is shown in Figure 4.17.

### 4.2.4 Benefits to this approach

A significant benefit arising from these changes is the reduced quantity of code needed to achieve the same goal; in fact, these operators are able to access a wider variety of species. Previously, MolBuilder was only generating molecules with an even number of mutations (i.e. only addition of even numbers of nitrogen heteroatoms was possible), however with reactionSMARTS, odd-numbered mutations are accessible.

Another benefit, particularly for users unfamiliar with the code base or programming in general, is increased readability of the code. As each genetic operation is based either on a reactionSMARTS pattern, or constructed with other genetic operator functions, the steps which make up molecular generation in MolBuilder are much easier to digest.

### 4.2.5 Improved control over design rules

Further functions have been added to MolBuilder in order to exert some more control over the molecules that are generated, and ensure that the defined rules are being followed, such as limits on how many unique ring and side group types are present.

#### 4.2.5.1 Counting building blocks

Given a molecule generated during a MolBuilder run, the functions `count_rings()` and `count_groups()` provide analysis of the building blocks used, and how many of each type are included.

These functions make use of the building block sets (`molecules` and `mutations_1`); for each building block, substructure matching and counting is performed to detect how many occurrences (if any) are present in the molecule. `count_rings()` and `count_groups()` return a dictionary object, where `keys` correspond to the present building block types, and `values` store the number of each type.

Due to the nature of SMARTS pattern matching, certain ring types may 'overlap' with one another, resulting in double counting or false detection; an example of this can be found amongst the building blocks extracted for the OCELOT project, as shown in Figure 4.18.

In order to solve this issue and allow for reliable counting of ring types within generated molecules, the `determine_overlap()` function examines if any ring building blocks appear as substructures within more specific building blocks (in the case of Figure 4.18, ketone-containing rings are more specific), and stores them in a

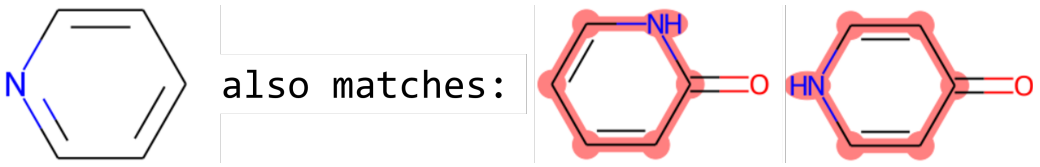



FIGURE 4.18: Cases where two ring building block types can be detected by when searching for another ring type, which appears as a substructure.

dictionary for use in `count_rings()`; when these are detected during ring counting, a further substructure match is completed for the other possible ring types, and these matches are taken into account before returning results.

`count_rings()` also avoids some pitfalls which RDKit's ring counting functionality suffers with, where cyclopentadiene rings aren't always identified and counted. Detection of these rings required a specific SMARTS pattern, shown below, since they aren't consistently marked as aromatic rings; in order to ensure detection, the pattern matches a 5-carbon ring structure (ring bonds marked with `@`), where one of the carbon atoms has two hydrogens attached (`[#6&H2]`).

```
cyclopentadiene_pattern = '[#6]@[#6]@[#6&H2]@[#6]@[#6]'
```

Counting building blocks with `count_rings()` and `count_groups()` provides information to MolBuilder when generating new populations of molecules, particularly with the modified genetic operators described below. These functions are also used as final checks for generated child molecules; there are cases where a crossover operation can return molecules with more than the desired number of rings or side groups, which must be discarded to ensure that the design rules set in the configuration are followed.

#### 4.2.5.2 Smarter mutations

More control can be exerted over the mutation operation in order to influence the number, and type, of side groups present on a working molecule; this has been implemented with the `smart_mutations()` function, which analyses the current molecule with `count_groups()` and makes decisions based on the results.

Smarter mutations have the ability to add or remove side groups from a molecule, decided at random with a 50% chance of either option. If adding a side group is selected, four outcomes are available.

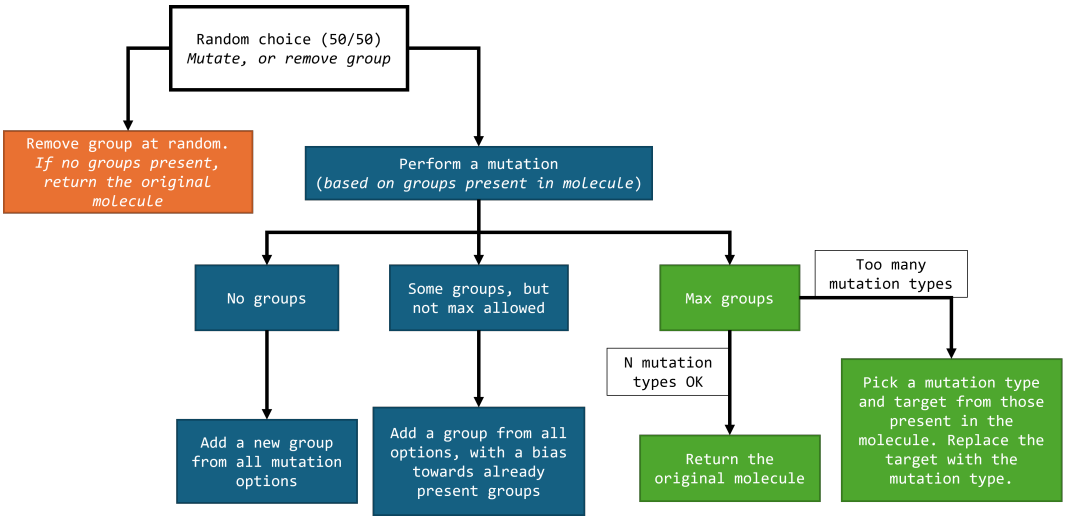



FIGURE 4.19: Flowchart representing the process undertaken by 'smarter' mutation operations

The possible outcomes, also shown in Figure 4.19, are as follows:

1. Molecule currently has no side groups, in which case a new group is added at random from those defined in the configuration;

2. Molecule has some side groups attached, but not the maximum number allowed; in this case a new group is added, with a bias towards those already present on the molecule, aiming to reduce synthetic complexity by lowering the variation in building block types;

3. Molecule has the maximum number of side groups, and the number of side group types does not exceed `unique_mutations_max` as defined in the configuration; this molecule remains unchanged;

4. Molecule has the maximum number of side groups, and more types than the `unique_mutations_max` limit; in this case, two present side group types are selected, and one is converted to the other. This lowers the number of unique side group types.

Smarter mutations benefit from the flexibility of the base `mutation()` function, which is able to target any valid SMARTS pattern and convert it into other compatible SMARTS structures; this means that side group addition, conversion between side group types *and* side group removal can all be handled with a single function.

#### 4.2.5.3 Single parent crossover

Standard crossover operations require two parent molecules, which get fragmented into four genes before swapping and recombination to produce children. An alternative operation was developed to allow the production of two children using only a single parent, using the function `single_parent_crossover()`. During any crossover operation, there is a 25% chance of the single parent variation occurring, where one of the parents is selected at random from the pair chosen through tournament selection processes; this molecule is then fragmented into a pair of genes, which each get 'completed' by ring additions to create new child molecules.

`single_parent_crossover()` operations will first analyse the latest population of molecules, and implement a bias towards adding ring types which are already present in the population often. In order to ensure that the limit on the number of unique ring types is followed, there are two possible outcomes for each gene as outlined in Figure 4.20.

Fragmentation is performed for this operation without any attachment point retention, allowing new rings to be added at any valid position through the use of `get_frag()` on the new ring, followed by `addition()` of this ring fragment to the gene.

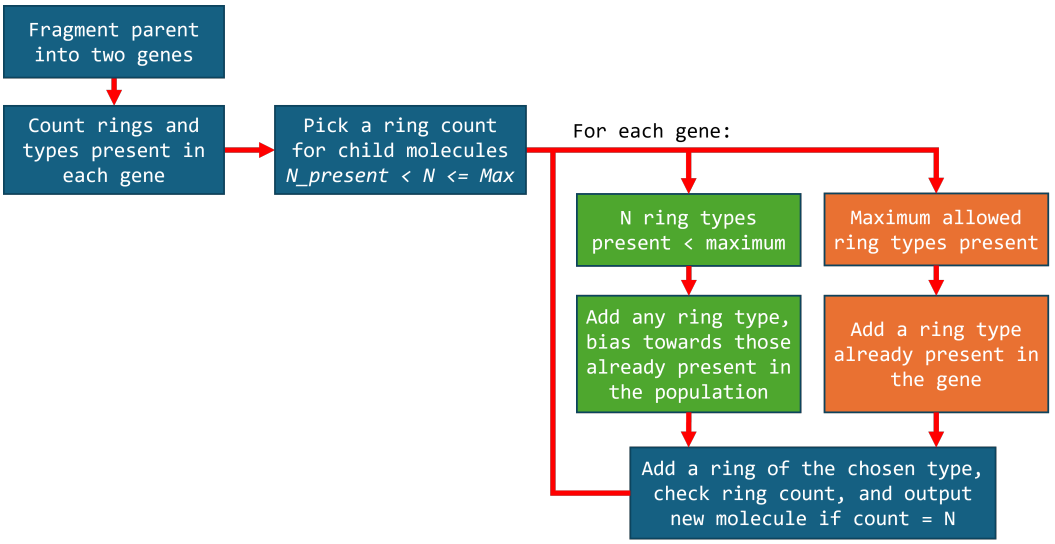



FIGURE 4.20: Flowchart representing the process undertaken during 'single parent crossover'

For each gene, and after each ring addition during this process, there are two options:

1. The number of unique ring types is less than the limit set in the configuration by `unique_ring_max`; any ring type can be added, biasing towards those more common in the last population;

2. The maximum number of unique ring types is present in the molecule; in this case, a ring type already present in the molecule is added, ensuring `unique_ring_max` is not exceeded.

`single_parent_crossover()` allows for wider exploration of chemical space by effectively combining each gene extracted from the parent with a new 'gene', which has been constructed on the fly, using building blocks, while maintaining limits on the variation in ring types set by the configuration; this means that crossover operations are not limited to only the genes available via fragmentation of molecules in the previous population.

### 4.2.6 "Reverse-MolBuilder"

As MolBuilder requires a set of fragments and mutations defining the chemical space it will explore, the ability to determine which building blocks are needed to generate a given molecule would be beneficial. To this end, a 'reverse MolBuilder' tool was developed, which iteratively fragments the target molecule into a set of single rings and mutation types.

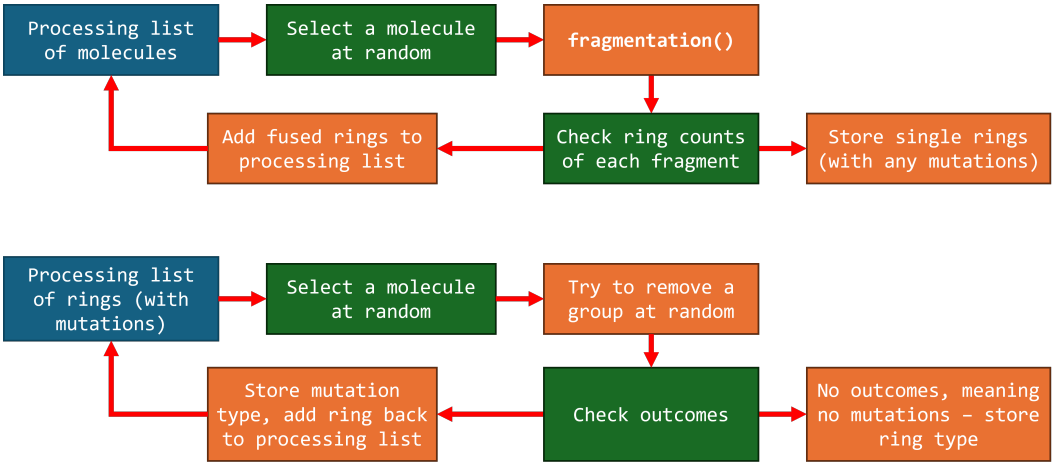



FIGURE 4.21: Overall process followed by the new Reverse MolBuilder tool; molecules are iteratively fragmented, all ring/side group types needed to re-produce the species with MolBuilder are identified if possible..

The script starts by iteratively performing `fragmentation()` to break apart fused ring systems, returning single ring types with any present heteroatoms and acyclic mutations. From here, acyclic mutations (such as methyl, halide and alkyl groups) are identified, removed from the rings and stored.

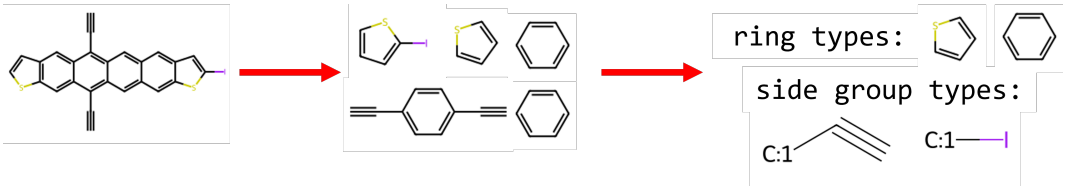



FIGURE 4.22: Example molecule fragmented with the reverse MolBuilder tool, along with the outputted fragment and mutation types.

Reverse MolBuilder then returns a pair of lists containing the ring types and mutations that would be required to build the original molecule. In cases where MolBuilder would not be able to construct the molecule, perhaps due to unsupported functionality (e.g. 7 membered rings, or single bonds/chains bridging between ring

systems), the tool simply returns 'False' to indicate that the fragmentation was not successful.

The reverse MolBuilder tool can be applied to a database of molecules, with the results being collated to inform users on the frequency of fragment types appearing across species in the set. This is used to extract building blocks from sets of known molecules, for use in genetic algorithms.

## 4.3 Full chemical space exploration

Using the `addition()` and `mutation()` genetic operators from MolBuilder, a full chemical space exploration campaign can be initiated given a set of fragments and mutation types. This was previously accomplished by iterative random generation of molecules:

- A ring type was chosen at random from the initial list of building blocks;
- A ring fragment was also chosen at random from the building blocks list, and added to the molecule. Fragments were added this way until the desired number of rings was found;
- A set of N mutations were performed, with the mutation type and target position being chosen at random;
- The final generated molecule was then checked against a Python set containing all novel species currently discovered; if not present, the new molecule was added to the set.

While this approach is effective given a long enough runtime, dependence on two random outcomes meant that the chance of discovering novel structures dropped as exploration progressed, regardless of the computational resources available.

### 4.3.1 Splitting molecular generation into steps

Rather than generating a molecular 'backbone' of rings and performing random mutations all in one process, the chemical space exploration can optionally be split into two steps; generating unmutated 'backbone' molecules, then performing mutations several times from each 'backbone'.

As previously, in order to generate an unmutated molecule, a ring type is chosen at random and ring fragments are added until the molecule reaches a desired number of rings. There are two approaches that can be taken to performing mutations on molecules, as shown in Figure 4.23; the choice of approach depends on how many unmutated molecules could be generated with the provided ring building blocks and desired range of ring counts.

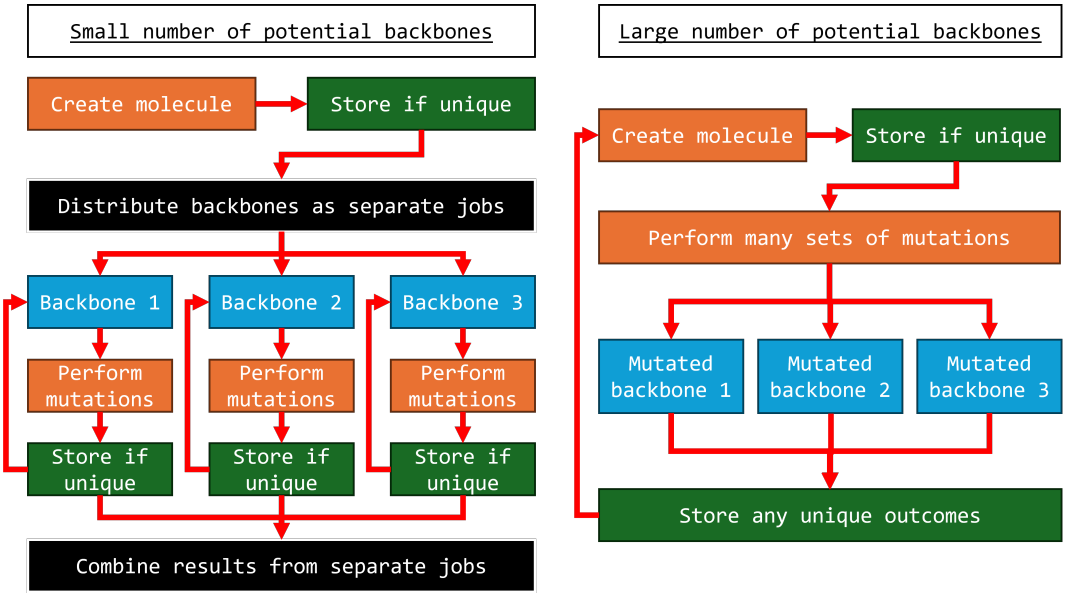



FIGURE 4.23: Two workflows for exhaustive chemical space exploration, chosen dependant on the number of potential unmutated molecular 'backbones'

Although the two 'random' steps in this process are still present in each approach, by splitting them apart the possibility of novel hits is increased. Using the original method, finding new molecules would depend on both finding the relevant molecular 'backbone', then performing a set of mutations leading to a new outcome.

With the newer approaches, the chances of hitting new molecules are increased:

- For small numbers of backbones, each unmutated species can be explored independently of all others, meaning in order to discover a new outcome, only a new set of mutations is needs to be trialled.
- When dealing with larger numbers of backbones, there are multiple chances for a new set of mutations to be found after generating an unmutated species. If a backbone has already been discovered, there are still multiple chances for a new pattern of mutations to be trialled.

### 4.3.2 Parallelisation and distribution of CSE

The full chemical space exploration process can be parallelised and/or distributed to increase the number of molecules discovered during a set period of time; the molecular generation and mutation steps do not incur large computational costs, so processors with *N* cores can be utilised to run *N-1* molecular generation 'worker' processes in parallel, alongside a single 'master' process which collects generated molecules and stores any unique discoveries.

In cases where the number of potential backbones is small, an initial step can be run to generate these backbones, each of which can then be submitted as a separate job focused solely on the mutation step; in these jobs, each worker will repeatedly perform random sets of mutations on the original backbone molecule, and the master process will receive mutated molecules and store any unique discoveries.

For spaces with a large number of backbones, where submitting separate jobs for each potential backbone becomes unfeasible, workers will handle both the molecule generation and mutation steps; at each iteration, a backbone is generated, and multiple sets of mutations are applied. The unmutated molecule and any unique mutation results are all sent to the master process, which stores any unique results.

### 4.3.3 Reaction product enumeration

Libraries of potential reaction products can be generated given a set of desired reaction types and available precursors; this task can be handled with MolBuilder using the 'reactor' molecular generation approach.

When running the 'reactor' code, different arguments are supplied in the configuration. Since molecules are constructed with reagent + reaction pairs, there is no need to define individual ring and side group building block sets. Instead, a list of `reagents` is supplied as SMILES, reaction types are supplied as reactionSMARTS in `reactions`, and the reactive target SMARTS pattern found in each reagent is defined under `rxn_target`. Examples of such parameters are shown in Table 4.2.

| Parameter | Example Input |
|---|---|
| `reagents` | `['Nc1cccc(F)c1N', 'Nc1ccnnc1N']` |
| `reactions` | `['[c:1](-N):[c:2](-N)>N-C(=O)-N>[c:1]1[N&H1]-C(=O)-N[c:2]:1']` |
| `rxn_target` | `'[c:1]([N&H2]):[c:2]([N&H2])'` |

TABLE 4.2: Table describing modified configuration parameters to be set before executing reaction product enumeration with MolBuilder's 'reactor' molecular generation functionality.

This approach to molecular generation also makes use of RDKit's molecule and reaction handling, alongside the ability to identify and count molecular substructures; a function to run reactions with this framework is shown in Listing 4.4.

`reaction()` will apply the chosen reaction to the molecule repeatedly, until the number of remaining reactive patterns (`target`) is equal to the chosen number of free sites (`free_targets`); this allows the option for partial saturation of reactive positions on the molecule, which is useful in cases where the chosen reactant has multiple potential sites where the reaction could occur.

```
def reaction(smi, reaction, target, free_targets):
    # Generate RDKit Mol and Reaction
    mol = Chem.MolFromSmiles(smi)
    rxn = Chem.ReactionFromSmarts(reaction)
    # Perform reaction repeatedly
    while True:
        # Run the reaction on current molecule
        outcomes = rxn.RunReactants((mol, ))
        # Convert random outcome to and from
        # SMILES, sanitizing the molecule
        mol = choice(outcomes)[0]
        mol = Chem.MolFromSmiles(Chem.MolToSmiles(mol))
        # Check how many reactive substructures remain
        n_targets = len(mol.GetSubstructMatches(Chem.MolFromSmarts(target)))
        # Stop reacting if desired number of unreacted targets remain
        if n_targets == free_targets:
            break
        # Otherwise, repeat the loop to run the reaction again,
        # where mol is now the partially reacted molecule
    return mol
    # RDKit Mol converted back to SMILES outside of function
```

LISTING 4.4: The base function used in MolBuilder's 'reactor' molecule generator, which applies a defined reactionSMARTS to a molecular SMILES, returning a single outcome chosen at random

In order to generate all possible outcomes from a combination of reagent + reaction, the `reaction()` function is attempted multiple times, with a selected number of free reaction targets to be left unreacted, as shown in Figure 4.24; this ensures that all possible combinations of reacted and unreacted targets on the reagent are found, including the completely unreacted reagent.

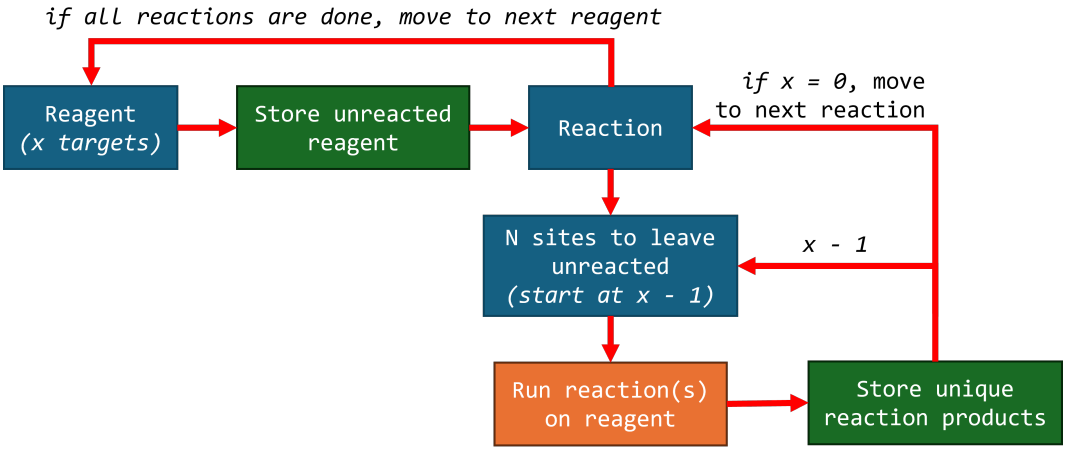



FIGURE 4.24: The workflow used to generate a full library of reaction outcomes, given a list of reagents and reactions, including unreacted and partially reacted species.

## 4.4 Synthetic difficulty prediction

Three SD scoring approaches were chosen for implementation into, or use alongside MolBuilder genetic algorithms; SYBA and SCScore were used to construct fitness functions considering different aspects of synthetic difficulty, while AIZynthFinder was used as a post-hoc analysis tool.

### 4.4.1 SYBA

SYBA is distributed as a Python package, and is readily accessible through the use of Anaconda, dependant only on the presence of an RDKit installation in the same environment.

```
from rdkit import Chem
from syba.syba import SybaClassifier
# Initiate the SYBA classifier
syba = SybaClassifier()
syba.fitDefaultScore()
# Run the prediction given a SMILES string
score = syba.predict('Nc1cccc(F)c1N')
```

LISTING 4.5: Utilising the SYBA classifier to predict a SYBA score from SMILES.

SYBA scores can be can be calculated given either a molecular SMILES, or the equivalent RDKit Mol object; for this project, SMILES is always provided to SYBA when generating scores, as shown in Listing 4.5.

This package is distributed under a GNU General Public License, and is also available to download through GitHub at `https://github.com/lich-uct/syba`. For the prediction of SYBA scores in this work, the package was installed with Anaconda, in a Python environment shared with the CSPy code base.

In order to be used as a fitness function during MolBuilder genetic algorithms, `calculation_type = SYBA` is defined in configuration, with `maximize=True`, as larger SYBA scores indicate classification as easier-to-synthesise. Given a newly generated population of molecules, worker processes will receive individual SMILES strings for each population member, initiate a SYBA classifier, predict the score and return this result to the master process.

Since `SybaClassifier` objects require a fair amount of memory usage, if too many instances are operating at once the entire process can fail due to a lack of required memory; workers can be organised into 'subgroups', where one worker initiates a `SybaClassifier` and predicts the score, while the rest are put on hold, avoiding such issues.

### 4.4.2 SCScore

SCScore is accessible as a Python package through GitHub, at `https://github.com/connorcoley/scscore`, and is distributed under an MIT License. Within the package, three pre-trained models are available, which were generated using 12 million reactions from the Reaxys database; the authors note that there is little advantage to using different models. These models differ by the type of Morgan fingerprints used as input; two use Boolean fingerprints of lengths either 1024 or 2048 bits, and the other uses an integer fingerprint of length 1024 bits.

SCScores are also calculated given a SMILES string, via the `SCScorer()` object, which returns the input SMILES and the associated score; the basic code to achieve this is shown in Listing 4.6.

```
from scscore_master.scscore_funcs import SCScorer
# Initiate the SCScore object
sc_scorer = SCScorer()
# Load in weights from the pre-trained model
sc_scorer.restore(weight_path=None)
# Use the SCScorer to calculate a score
(smi_, result) = sc_scorer.get_score_from_smi(smi)
```

LISTING 4.6: Utilising SCScore to predict a SCScore from SMILES.

A lightweight version of SCScore was implemented into the CSPy code base, containing only the predictive model trained on Boolean 1024 bit fingerprints, and a modified version of the SCScorer class allowing for it to be imported and used as a fitness function during MolBuilder genetic algorithms.

The memory requirements to calculate SCScores are lower than SYBA, and it does not suffer the same issues when several `SCScorer()` objects are running in parallel; the slowest step for this package is loading in pre-trained model weights, so each worker process will initiate an `SCScorer()` at the start of a MolBuilder run, and when these workers receive a SMILES string `sc_scorer.get_score_from_smi(smi)` is applied to calculate the score.

### 4.4.3 AIZynthFinder

The retrosynthesis planning tool AiZynthFinder is available as a Python package, installed either through GitHub or with Anaconda. Alongside the package a set of files containing reagent stock lists extracted from the ZINC database, a pre-trained expansion policy network of reaction templates based on reactions from the USPTO dataset.

AiZynthFinder operates by default with an artificial neural network policy (NNP) guided Monte Carlo tree search (MCTS), recursively breaking down molecules into purchasable precursors based on the library of reaction templates.

Given a target molecule (root node), reaction templates are shortlisted, and one is simulated to split the molecule into a set of precursors (leaf node); these precursors are then subjected to the same steps, if they're not available as reagents in the stock list. This process repeats until a defined number of steps have been applied (maximum tree depth), or when all found precursors are purchasable.

At this point, the results of the simulated reaction pathway are propagated back up the tree, updating each node with information about the quality of the route, which helps to guide further searches; future explorations from the root node can use this information to inform future decisions, with a chosen balance between exploring new routes and exploiting routes which have previously shown good results.

The NNP predicts the outcomes of promising steps using its understanding of chemical transformations learned from the dataset of reactions with which it was trained; given a target molecule either as the root node or a leaf node, the probability of potential precursors produced by reaction templates successfully reacting to produce the target molecule is assessed, and used to inform decisions on which leaf nodes to generate and explore.

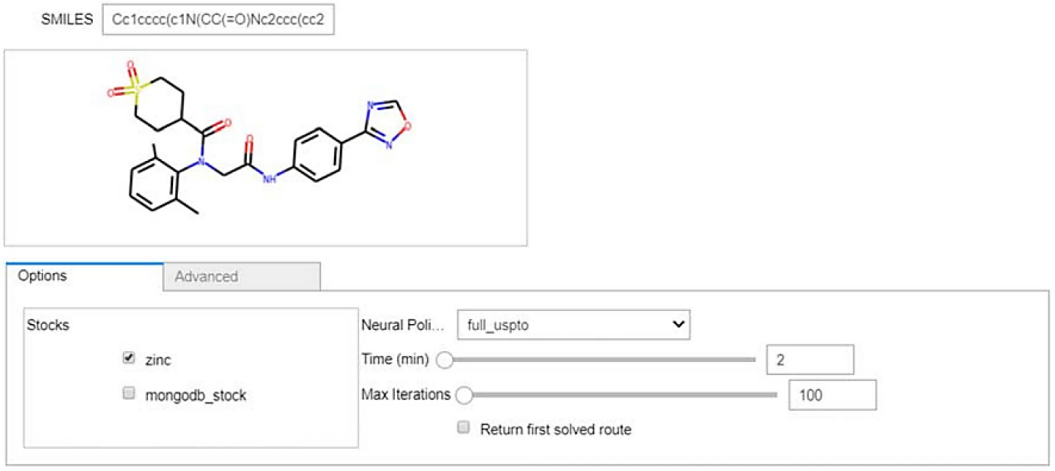



FIGURE 4.25: The Jupyter Notebook GUI available to run AiZynthFinder analysis given a SMILES string.

There are three ways to utilise AiZynthFinder to predict synthetic pathways to a molecule:

1. A graphical user interface built with Jupyter Notebooks, shown in Figure 4.25; this can be used to perform analysis on a single molecule, outputting any precursors needed, the reaction steps used to combine those precursors and produce the target molecule, and statistics found during the process including a score calculated based on the synthetic route found.

2. A command-line interface; this is able to perform analysis on a batch of molecules and store the generated search tree of possible reaction paths, including information on the routes themselves, such as how many are marked as 'solved' (reaching a set of precursors which are all 'in stock' in the reagent list) and the calculated scores for each route.

3. A Python interface, importable into other Python scripts; this allows for an `AiZynthFinder()` object to be initiated, which can perform the tree search of potential synthetic routes, build these routes, then calculate scores and statistics in a more flexible manner.

For this project AiZynthFinder was only used to predict the synthetic routes of molecules and generate scores outside of genetic algorithms, so the command-line interface was chosen to run batches of molecules, generating reaction pathways and associated scores.

```
aizynthcli --config config_file.yml --smiles batch_smiles.txt
```

Custom reagent databases can be used with the AiZynthFinder tool, and a custom neural network policy model can also be trained given a database of reactions; for this project the ZINC-based stock list and USPTO-trained neural network policy provided by the authors were used.

## 4.5 Property prediction

Within CSPy, there are functions which 'wrap around' external packages to handle certain steps within the property prediction process, and of course in-house code to execute crystal structure predictions; these were not developed by the author of this thesis, but are worth discussing due to their importance in the work.

### 4.5.1 Geometry optimization

From the set of 2D coordinates defined upon loading a molecule from SMILES into RDKit, an initial 'rough' geometry can be generated, followed by a quick geometry optimization using the universal force field (UFF), which is a general force field applicable to a wide range of molecules (Rappé et al. (1992)). These 3D coordinates can then be extracted from the RDKit Mol object, and stored as a .xyz file.

```
from rdkit.Chem import AllChem as Chem
# Load 2D molecule from SMILES, add implicit hydrogen atoms
mol_in = Chem.MolFromSmiles(smiles)
molecule = Chem.AddHs(mol_in)
# Generate initial 3D coordinates
Chem.EmbedMolecule(molecule)
# Optimize the 3D coordinates with UFF
Chem.UFFOptimizeMolecule(molecule)
# For each atom, extract [atomic_number, x, y, z]
coords = coordinates_from_rdkit_molecule(molecule)
# Write extracted coordinates to a '.xyz' file
coordinates_to_xyz_file(coords, xyz_file)
# '.xyz' file format:
# [Atom count]
# [Atom type, x, y, z] for each atom
```

LISTING 4.7: Generating 3D molecular geometries from SMILES strings with RDKit UFF optimization.

The coordinates found with RDKit UFF geometry optimization are used as a starting point for higher-level geometry optimization with either Psi4 or Gaussian09; UFF optimization is utilised as a pre-processing step as the generated coordinates may not be fully accurate, but are generally closer to the optimal minimum energy conformer. The more costly optimization at a higher level with Psi4 to reach a more optimum accurate gas-phase conformer can often be accomplished in fewer iterations, if started from a set of coordinates 'pre-optimized' with UFF.

The Day group's CSPy code base contains subprocess functions for the submission of Gaussian09 and Psi4 tasks, and the extraction of results from these tasks on completion. For this project, Psi4 was favoured as it is available as an open-source package, meaning users of the Day group code base can run these optimizations without requiring licenses for external packages.

In most cases, the conformation is assumed to be rigid after this geometry optimization process; there are cases where further optimizations are performed after modification to the molecule, and where multiple potential conformers are generated to trial different molecular geometries.

Geometry optimizations operate by repeatedly adjusting atomic positions and calculating the single point energy of the atomic arrangement, and forces acting on atoms, at each iteration, working to minimize the potential energy of the molecule; as a result of this, whenever geometry optimization is run with Psi4, the final single-point energy of the molecule is also available for use.

### 4.5.2 Crystal structure prediction

The Day group maintains CSPy, a Python-based crystal structure prediction (CSP) software library which handles the sampling of crystal energy landscapes, leading to generation and minimization of possible crystal packing arrangements for a species given its molecular structure.

Before generation of crystal structures can occur, molecular geometries must be determined, typically for the neutral molecule, with the geometry optimisation

process outlined earlier. CSPy has recently been developed to allow for 'flexible-CSP', where multiple molecular geometries are trialled, to simulate systems which could form crystalline systems with molecular geometries other than those found with the optimal coordinates calculated with an isolated molecule.

The optimized geometry `[molecule].xyz` file is then subjected to distributed multipole analysis with GDMA (Stone (1999)), using the command-line app `cspy-dma` available within CSPy, which outputs the remaining set of files required to run crystal structure prediction; `[molecule].mols` defining the molecular axis, `[molecule].dma` containing the calculated multipoles, and `[molecule]_rank0.dma` defining molecular charges.

Given the optimized molecular geometry, and results from distributed multipole analysis, trial crystal structures are quasi-randomly generated using Sobol sequences to select crystal lattice parameters and molecular positions/orientations, evenly sampling the landscape of possible packing arrangements and unit cell dimensions. Sampled crystal structures are then subjected to a clean-up step, attempting to remove any collisions between molecules and reduce the unit cell volume.

These trial structures are then lattice energy minimized with DMACRYS (Price et al. (2010)), allowing the molecules to 'settle' into their optimal positions relative to one another; the program iteratively adjusts molecule positions and orientations within the crystal,using a quasi-Newton-Raphson optimizer, where intermolecular interactions are calculated at each step to eventually produce crystal packing arrangements with minimized lattice energies.

In cases where the species under investigation already exists, the generated crystal structures can be compared to experimental data available from sources like the Cambridge Structural Database (CSD); often the lowest energy structure on a generated landscape will correspond to the experimental results, however this isn't always the case.

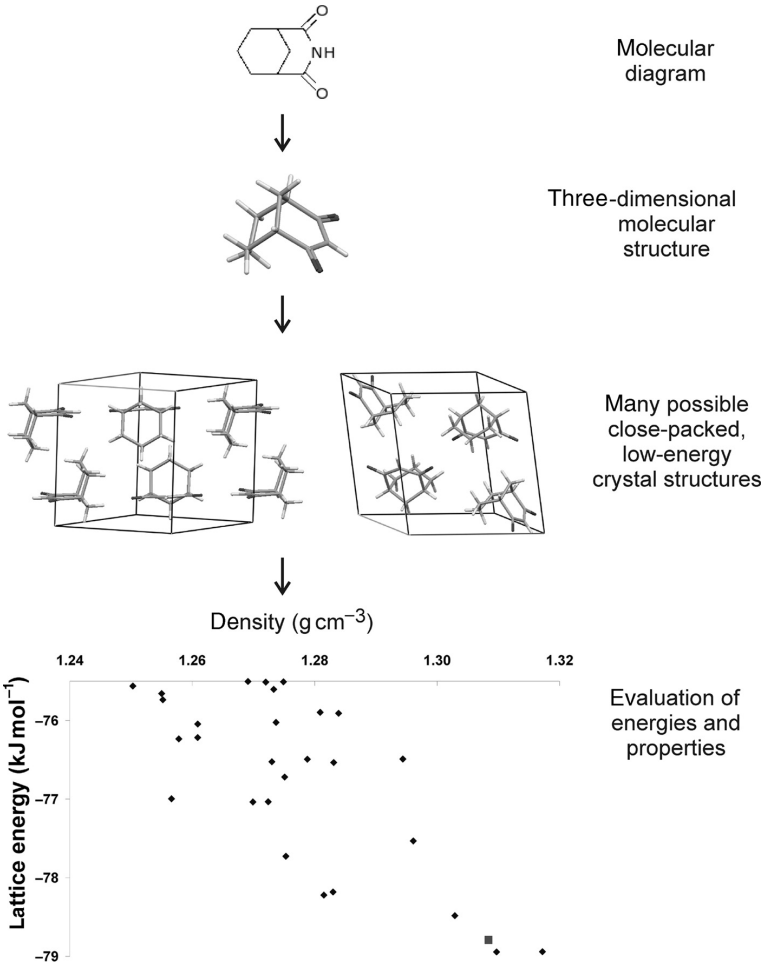



FIGURE 4.26: The general approach used to generate landscapes of predicted crystal structures, from an article by Day (2011).

### 4.5.3 Charge mobility calculations

As explained in the theory section, charge mobility in candidate molecular materials is estimated in two ways; electron reorganisation energies determined through comparison of molecular geometries in neutral and charged states, and electron mobilities which consider both molecular reorganisation energy, and the rate at which electrons can 'hop' between molecular units in the solid state.

#### 4.5.3.1 Reorganisation energy

In order to calculate electron reorganisation energy, the energies of two molecular geometries at two different charge states (neutral and ionic) are combined; this means that four molecular energy calculations need to be performed:

1. $E^-(R_0)$: Anion of the molecule, in the optimal geometry of the neutral form
2. $E^0(R_0)$: Neutral molecule in its optimal geometry
3. $E^0(R_-)$: Neutral molecule, in the optimal geometry of the anionic form
4. $E^-(R_-)$: Anion of the molecule in its optimal geometry

The code inherited in CSPy to calculate reorganisation energies at the start of this project used Gaussian09; this was switched to use Psi4 through development by another member of the Day Group, Jay Johal.

Starting from a SMILES or InChI, geometry optimization is performed with RDKit and Psi4 on the uncharged form of the molecule, which produces both the optimized ground-state coordinates and the associated energy value $E^0(R_0)$; the neutral molecule optimized coordinates are then subjected to another geometry optimization with charge introduced (in this case, an extra electron), which determines the anion's optimal geometry and associated energy value $E^-(R_-)$.

A single-point energy calculation is performed on the optimized neutral molecule coordinates, with an extra charge added, generating the energy value $E^-(R_0)$; the remaining energy value $E^0(R_-)$ is then determined by running a final single-point energy calculation using the optimized anion coordinates, with the molecular charge set to zero.

$$(E^-(R_0) - (E^0(R_0)) + (E^0(R_-) - (E^-(R_-))$$

Once each of these values are calculated, a final step combines them according to the equation defined below to generate an electron reorganisation energy value, which is then stored as a calculated property. The outputs from tasks in this process are also saved as each step completes; in cases where all four steps could not be completed, this allows for the calculation to pick up where it left off, rather than requiring a full restart.

#### 4.5.3.2 Electron mobility

Electron mobility calculations require the solid state packing arrangement, in order to determine how easily electrons can 'hop' between molecular units; this prompted development of 'on-the-fly' crystal structure landscape prediction, from which average electron mobilities can be calculated. Again, the development of code for this task was completed and implemented into MolBuilder by Jay Johal, for use in genetic algorithms.

As stated during the discussion of crystal structure prediction, a given molecule does not necessarily adopt the lowest energy packing arrangement found on the predicted crystal landscape; rather than assuming that the electron hopping rate can be calculated with the lowest energy structure, it makes more sense to base the calculation on average hopping rates across structures in low-energy regions of the landscape.

Kernel density estimation is used to calculate average hopping rates across a crystal landscape, with more weight placed on structures in more densely populated low-energy regions; this allows the calculation of average hopping rates with a bias towards more energetically favoured packing arrangements.

# Chapter 5

# Projects basis and setup

## 5.1 Azapentacenes - testing space

Azapentacenes describes a group of molecules composed of 5 fused benzene rings, with various levels of nitrogen heteroatom substitution. These have been the focus of previous work within the Day Group, such as the publication from Cheng et al. (2020) which entailed full exploration of the space, theoretically listing all possible nitrogen atom substitutions and connectivities of five 6-membered aromatic rings.

### 5.1.1 Reason for working in this space

While changes were made to MolBuilder, this set of molecules was chosen to act as a validation dataset. Since the total number of molecules and the top candidates in this space had already been determined, new approaches to molecular generation could be tested with the same molecular building blocks to ensure that the original set of molecules could still be discovered, before moving on to different or more complex molecular families.

Given that the chemical space is theoretically fully explored, distributions of properties for the azapentacenes should be completely populated, and give insight into what these results should look like when an entire space is described.

As a relatively similar group of molecules, this space can also be used to test out various synthetic difficulty calculations, and see how well they are able to partition a set of molecules with similar appearances.

#### 5.1.1.1 Past work

During previous studies of the azapentacenes within the Day Group, MolBuilder genetic algorithms were set up in a slightly different manner; the set of building blocks and mutations are shown in Figure 5.1.

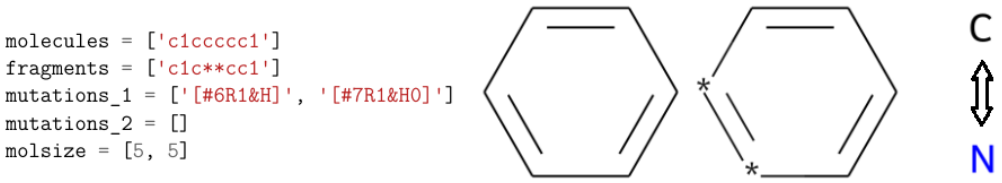



FIGURE 5.1: Definitions of building blocks for the azapentacene chemical space in older versions of MolBuilder.

The 'starting' molecular building block was defined as a simple benzene ring, from which other molecules would be constructed. Molecular building blocks were defined with attachment points (denoted with asterisks *), which dictated where they would be added to molecules during construction.

Molecular size, or the number of rings required in constructed species, was defined by the parameter 'molsize'; in this case only 5-ring species were required. Alongside this was a list of mutations, defining how atoms could be converted into 'mutated groups'; in this case the conversion between aromatic carbons and aromatic nitrogen heteroatoms.

By selective choice of building blocks, the only restriction placed on this exploration was avoidance of fragment addition to cove, bay and fjord patterns within molecules (as in Figure 5.2), which excludes the formation of pyrene substructures. This was done by only providing a building block with two 'attachment points'.

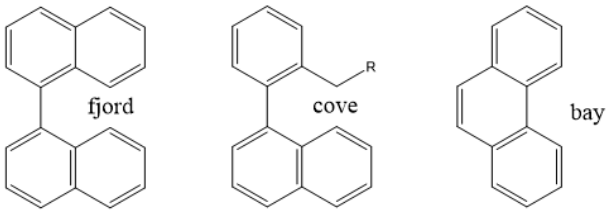



FIGURE 5.2: Illustration of fragment addition to coves, fjords and bays within a aromatic system. These types of additions were forbidden by the chemical space exploration algorithm.

#### 5.1.1.2 Past results of exhaustive and directed exploration

From previous chemical space exploration campaigns, it was determined that 68,064 azapentacene molecules were accessible through combinations of building blocks using the MolBuilder genetic operations. The space was also expanded in a directed manner using the evolutionary algorithm, identifying structural motifs with low reorganisation energies and high electron affinities (Figure 5.3). This demonstrated the efficiency of chemical space exploration with an evolutionary algorithm; rather than performing calculations for every species, the large space can be searched to find promising candidates while only requiring computation of reorganisation energies for roughly 1% of the molecules.

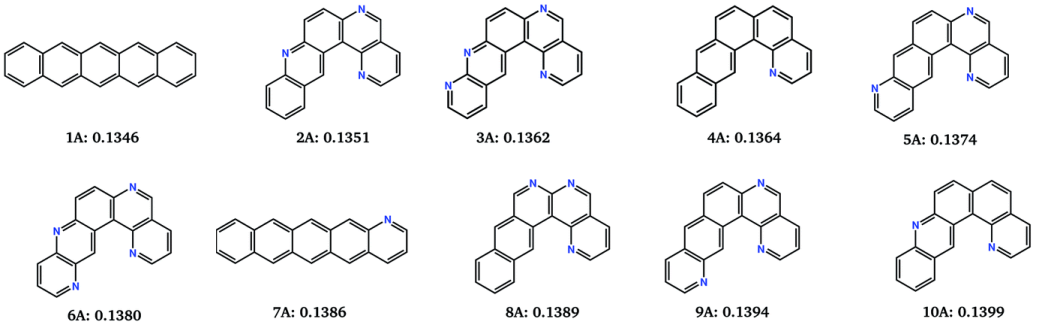



FIGURE 5.3: Promising azapentacene motifs identified during evolutionary chemical space exploration Cheng et al. (2020), labelled with their fitness values of reorganisation energy (eV)

### 5.1.2 Updated chemical space setup

As described in the code development section, building block definition and handling was updated in order to make setup more flexible. For the generation of azapentacene molecules, the set of building blocks is minimal, now requiring only a single ring type (benzene) and single mutation (conversion of aromatic carbon to aromatic hydrogen); all azapentacene molecules contain five rings, so the range of `molsizes` is also minimal.

```
molecules = ['c1ccccc1'] # benzene only
mutations_1 = ['[n:1]'] # convert to N only
molsize = [5, 5] # 5 rings only
```

LISTING 5.1: Building blocks and design rules required to generate azapentacene-type molecules.

Azapentacene generation is the only case in this project which utilises `mutation()` to convert between aromatic carbon and aromatic heteroatoms; as stated in the code development chapter molecules are generated from the defined ring types before mutations occur, and for the azapentacenes in particular there are a limited number of potential unmutated molecules.

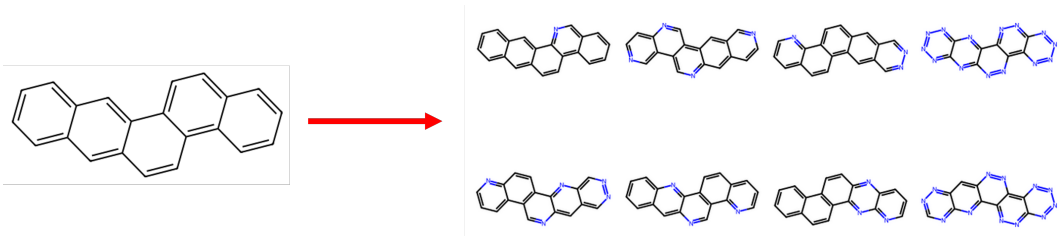


FIGURE 5.4: Illustration of using `mutation()` to convert random sets of aromatic CH positions into nitrogen heteroatoms

For the repeated full exploration of this space, any number of mutations can be performed on a molecule; and example of this is shown in Figure 5.4, following the process outlined in Figure 4.23.

## 5.2 OCELOT - organic semiconductors

The main portion of this project concerns the exploration of a chemical space constructed with MolBuilder-compatible molecules from the OCELOT (Organic Crystals in Electronic and Light-Oriented Technologies) archive, authored by Ai et al. (2021).

The potential size of this space is orders of magnitude larger than the azapentacenes, but aspects of the configuration can be carried over; explorations were still limited to rigid aromatic fused ring systems, which could undergo mutations to allow the addition of side groups rather than adding heteroatoms to rings.

### 5.2.1 Original Dataset

The original set of molecules used in this project were provided in the OCELOT Chromophores v1 dataset, available at `https://oscar.as.uky.edu/datasets/`; this contains, alongside a set of calculated descriptors, 25,251 SMILES strings and CSD REFCODES for pi-conjugated systems in known, experimentally validated molecules.

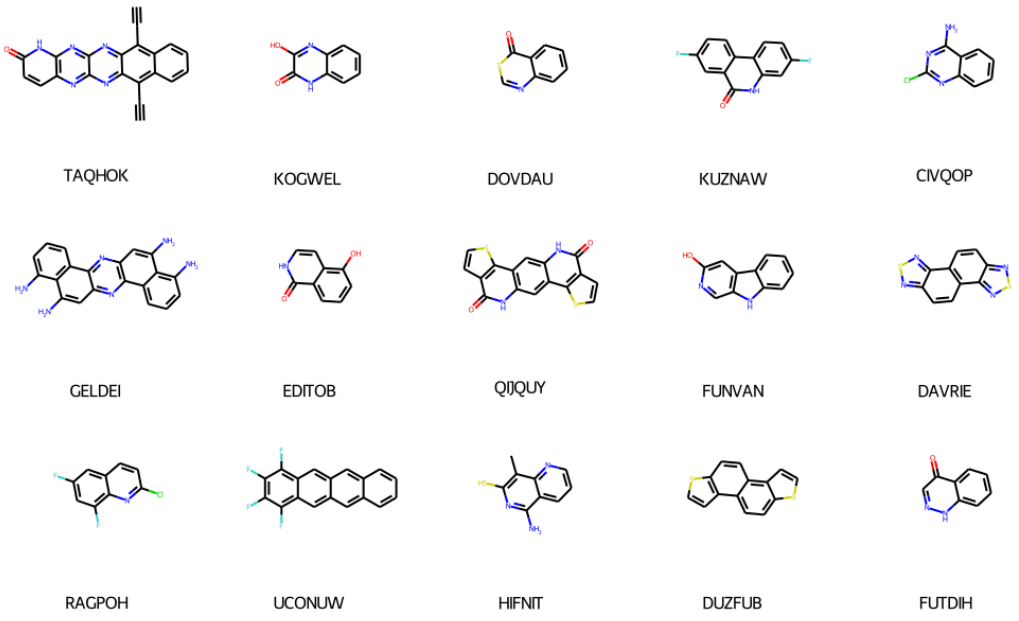



FIGURE 5.5: A selection of OCELOT experimental molecule pi-conjugated systems, with their corresponding CSD REFCODES.

#### 5.2.1.1 Filtering the dataset

Starting from the original dataset, molecules were loaded into Python from their respective SMILES strings with RDKit, and a series of filters were applied:

1. Molecules with any rotatable bonds, either between fused-ring systems, or as part of side chains, were removed, ensuring no flexibility. This reduced the number of molecules to 4,651;
2. Molecules containing rings of size larger than 10 atoms were removed, reducing the molecule count to 4,427;
3. These were then passed through the 'reverse-MolBuilder' process, which identifies any molecules which have atoms bridging between fused ring systems, including any which hadn't been marked as 'rotatable' by the initial step; bringing the final list to 1,258 molecules.

#### 5.2.1.2 Extracting and choosing building blocks

In order to generate molecules with MolBuilder, a set of ring and side group building blocks are required. The 1,258 molecules remaining after filters were passed through the 'reverse-MolBuilder' tool again, this time in order to extract the different ring and side group types present and count the number of times each appears within the dataset. A set of 186 unique ring types and 18 side group types were extracted by this process, however simply using all of these building blocks would result in an overwhelmingly large chemical space to explore.

In order to limit the potential explosion in chemical space size due to a large number of potential building blocks, the ring and side group types were ranked by order of occurrence within OCELOT molecules, with the most common subset being selected for use for MolBuilder explorations. This decision was also made in an effort to introduce bias towards more synthetically available structures, stemming from the idea that common rings and side groups in a dataset of synthetically known molecules should generally be easier to synthesise.

#### 5.2.1.3 Final building block choices

This sequence of filters and choices resulted in a set of 12 ring types, and 5 rigid mutation types, shown in Figure 5.6; it should be noted that bromine was originally included as an additional mutation type, but was removed to avoid the need for more costly calculations with the additional diffuse functions required.

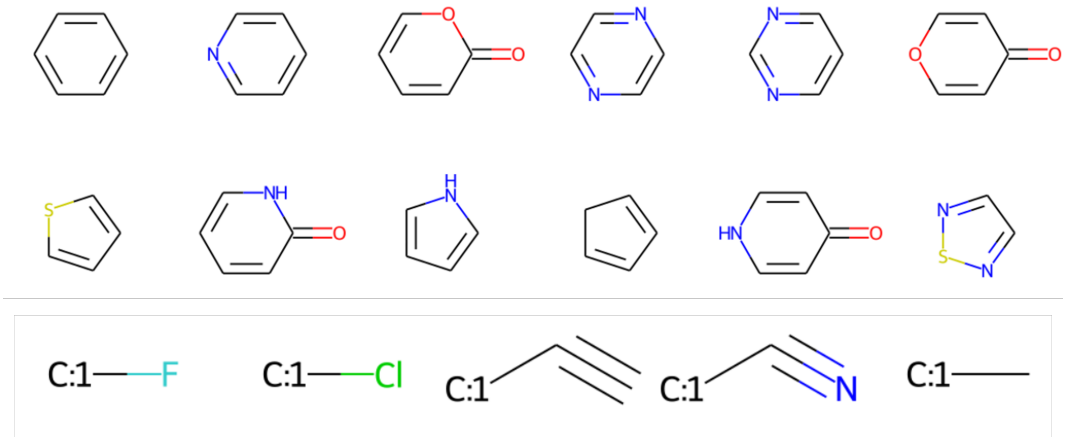



FIGURE 5.6: The selection of building blocks and mutation types generated from the filtered OCELOT experimental dataset.

### 5.2.2 Genetic algorithm setup

After extracting building blocks from the OCELOT dataset and realising the potential size of the chemical space accessible with them, even after limiting to the most common types, further analysis was performed on the filtered set of molecules; this was done to inform additional design rules and modifications to the configuration used in MolBuilder genetic algorithms.

#### 5.2.2.1 Rules on molecular generation

Certain rules are defined in the configuration file submitted to MolBuilder explicitly, while some are 'implied' by the choices of building block types and the current construction of the MolBuilder code; these 'implied' rules are controlled by building block choices, and how the building blocks can be combined.

As an example, by defining only aromatic rings as the building block type, MolBuilder is only able to create or destroy two-atom fused ring bonds; this means that molecular generation before mutation steps is limited to rigid fused ring systems, without access to fjord or bay-type ring additions, highlighted red in Figure 5.7

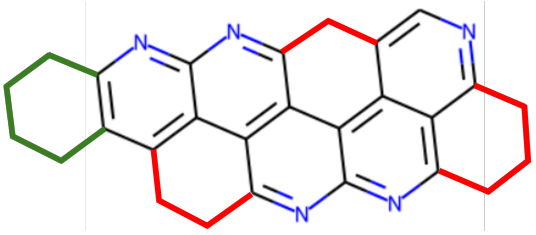



FIGURE 5.7: MolBuilder code is limited to 2-point addition, avoiding the formation of certain fused ring structures (highlighted red), and ensuring only two-atom fused ring bonds can be created (highlighted green)

Several rules are explicitly defined in the configuration, controlling factors such as how many rings and unique ring types can be present in generated molecules, through the use of parameters like `molsize` and `unique_ring_max`.

In order to match up with the original set of molecules when generating new species in the surrounding chemical space, the distribution of ring sizes and frequencies of the experimental set was investigated to inform choices on these rules. Analysis revealed the statistics shown in Figure 5.8, where the majority of molecules in this set contain:

- $2-5$ aromatic rings, with either two or three unique ring types present
- $<3$ rigid side groups attached, with either one or two unique side group types.

In order to replicate the general construction of molecules from the original OCELOT set, these parameter ranges were chosen to be used in MolBuilder genetic algorithm runs. The choices made here were also made as another way to implement bias towards more synthetically accessible molecules; fewer variation in ring types could result in fewer reaction types needed, and a lower ring count means less reaction steps should be needed overall to synthesise a target molecule.

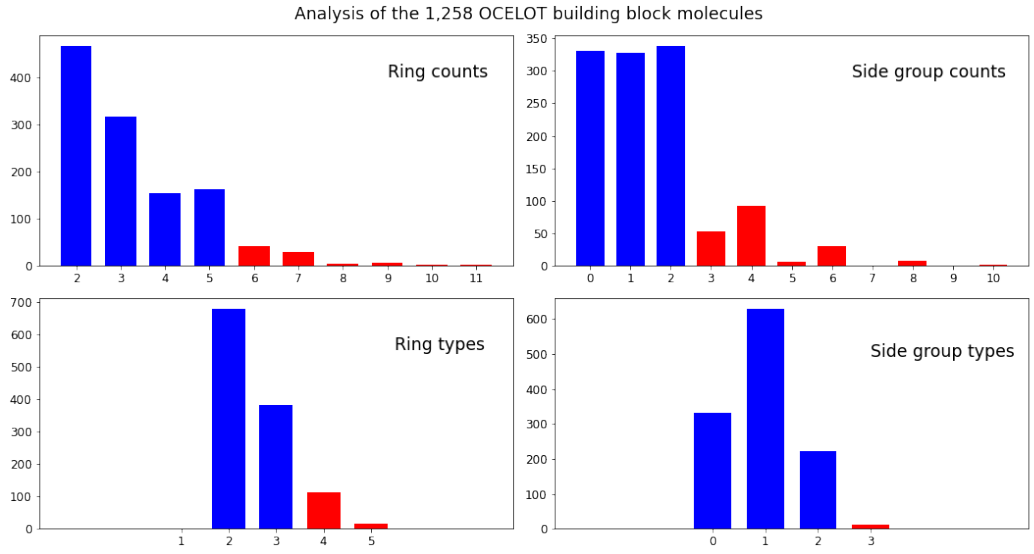



FIGURE 5.8: Analysis of the OCELOT molecules from which building blocks were extracted, used to inform design choices in MolBuilder genetic algorithms

At first, genetic algorithms with the OCELOT building blocks set to run for thirty generations, but this was later extended to 50; as discussed in the analysis of results chapters, convergence of the genetic algorithms was not clearly observed by generation 30, so the jobs were extended, giving the GA more time to find local minima.

#### 5.2.2.2 Properties to optimize

Genetic algorithms were split into four optimization targets, distributed between local machines and supercomputing resources according to the computational costs and considerations associated with each fitness function.

The first calculation type, minimization of electron reorganisation energy, was carried over from the azapentacenes work; a large number of reorganisation energies had already been calculated, and the functionality to use this property as a fitness function were already implemented within MolBuilder. Reorganisation energy genetic algorithms were run on the University of Southampton IRIDIS5 high-performance computing resource (Roe (2018)).

A further physical property calculation, determining charge mobility, requires more complex analysis with crystal structure predictions, which were feasible with this space due to the avoidance of flexibility in generated molecules; while the computational cost associated with CSP at this scale is large, it is reduced where flexibility in molecular units does not need to be considered, and the reduced sampling level implemented for use in genetic algorithms.

CSP-led genetic algorithms were performed on the UK National Supercomputing Service, ARCHER2, as part of a Grand Challenge project under the Materials Chemistry Consortium.

Synthetic difficulty fitness functions were implemented into MolBuilder with the SYBA and SCScore packages, which do not require intense calculations since pre-trained models have been provided to predict their respective scores given only a SMILES string

Genetic algorithms optimizing on these synthetic difficulty scores were run either on local machines, or on IRIDIS5 to take advantage of wider parallelisation of worker processes (local work is split across 6 cores, while IRIDIS5 nodes provide 40 cores each).

Multi-objective fitness functions, combining reorganisation energy calculations with predicted synthetic difficulty scores, were also run on IRIDIS5; at this point in the project, lookup databases of calculated properties had been implemented within MolBuilder, meaning any molecule which had been discovered by for example a reorganisation energy GA could 'skip' it's calculation and retrieve the property value.

### 5.2.3 Selecting initial populations for genetic algorithms

Initial populations of molecules can be set with MolBuilder, and re-used in multiple genetic algorithm runs to determine if repeat behaviour is observed under the same conditions, and how the use of different fitness function impacts the 'direction' in which populations move as the algorithms progress.

A set of 36 initial populations were generated for this project, using the selected building blocks or subsets of them to generate unique molecules. 20 of these populations used all selected building block types, and were named 'full populations' 1-20; the remaining 16 used four subsets of the selected building blocks, labelled as 'reduced populations' 1-16. An example of these populations is shown in Figure 5.16, and all initial populations will be shown in the appendix.

#### 5.2.3.1 Populations with all building blocks

Twenty 'full' populations were generated with the building blocks shown in Figure 5.6, without any molecules shared between populations, ensuring that each was unique and could be considered as separate starting points for genetic algorithm runs. This was achieved through use of MolBuilder's genetic operations `addition()` and `mutation()`; using the method outlined in Figure 4.13, a set of 2000 unique molecules was generated and subsequently partitioned into 20 separate populations.

To confirm that a good level of diversity is present both within each population and between the initial populations, average and maximum pairwise Tanimoto fingerprint similarity analysis was performed, comparing each molecule in one population to each molecule in another. Heatmap plots of these results verify that populations are relatively diverse, with a maximum average similarity below 20%, and only a few instances where the maximum similarity between a given pair of molecules from different populations is above 90%; these plots are shown in Figure 5.9.

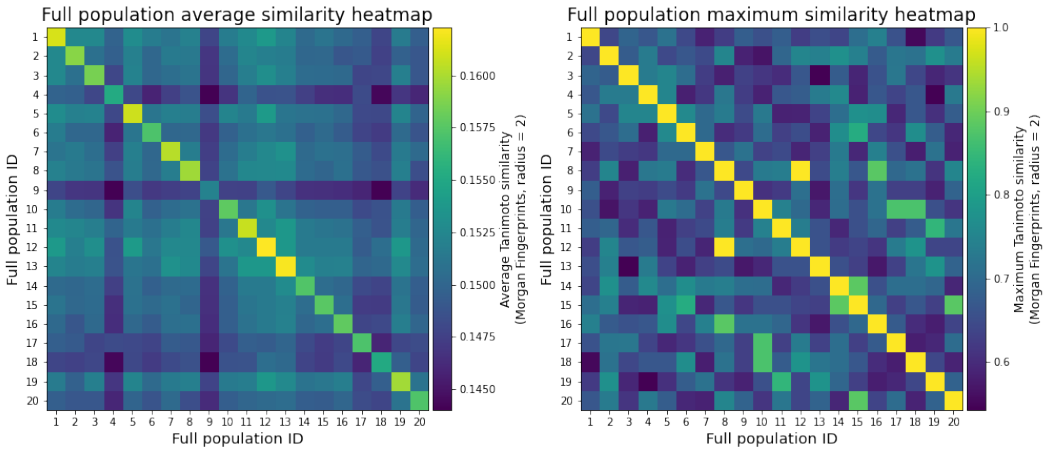



FIGURE 5.9: Heatmaps comparing the Tanimoto fingerprint similarity between 20 initial 'full' populations constructed with the OCELOT building blocks

#### 5.2.3.2 Populations with reduced building block sets

Four different subsets of the chosen OCELOT building blocks were selected for a group of genetic algorithms which work in smaller portions of the overall chemical space available to the full set; each of these four used the same subset of side group building block types, where chlorine and alkyne groups have been removed due to similarity to other side groups in the full set.

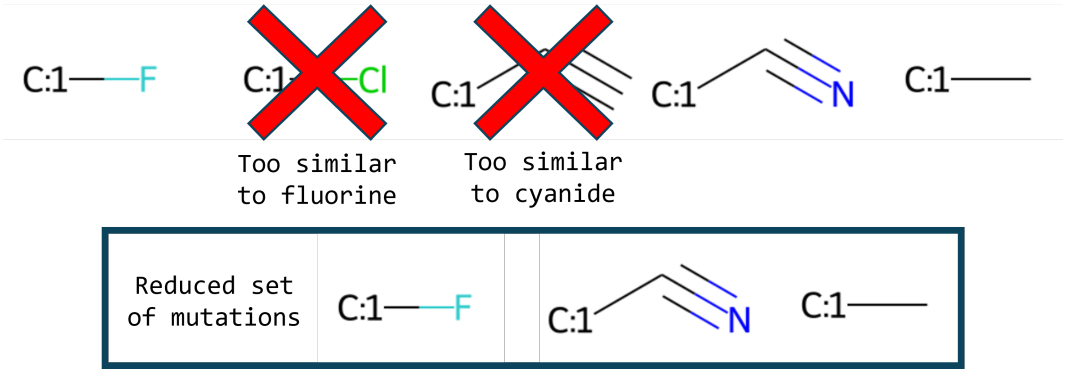



FIGURE 5.10: Three of the five chosen side group types were used in each type of reduced OCELOT building block set.

The first four 'reduced' populations were constructed as a 'step up' from the original azapentacenes work, with access to five- and six-membered rings, some of which have defined nitrogen substitution patterns; this is the simplest of the OCELOT building block sets, and the fragments used are shown in Figure 5.11.

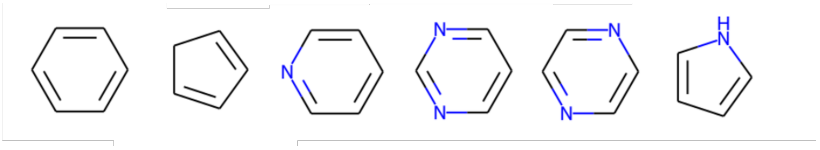


FIGURE 5.11: Ring building blocks used to create and modify the first set of 'reduced' population molecules: aza-substituted ring systems.

As another step up in complexity, four 'reduced' populations were run with the same building blocks, plus two additional ring types which contain sulphur heteroatoms; these are termed as explorations of the chemical space with aza- and thio- substituted ring systems, using the building blocks shown in Figure 5.12.

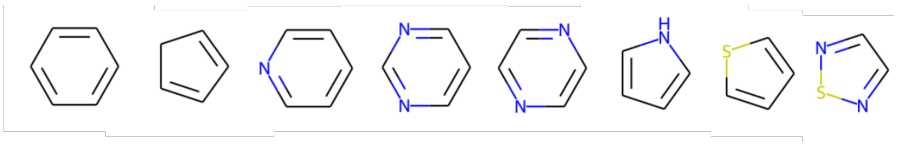


FIGURE 5.12: Ring building blocks used to create and modify the second set of 'reduced' population molecules: aza- and thio-substituted ring systems.

The next four populations were built with a subset of the OCELOT fragments which contain carbonyl, ester and amide-type substructures; these populations are quite different from the other 'reduced' initial populations, as seen in Figure 5.15.

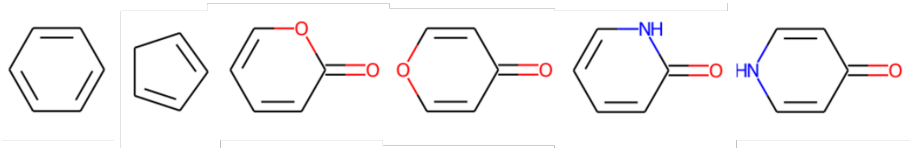


FIGURE 5.13: Ring building blocks used to create and modify the third set of 'reduced' population molecules: acenes with amide/ester-type rings.

The final and most complex 'reduced' building block set aims to capture the variety of building block types by selecting the most common of each 'pair' of similar rings; this set will be named 'varied ring types', and was chosen as an attempt to reduce the potential size of the explorable chemical space while maintaining its variety.

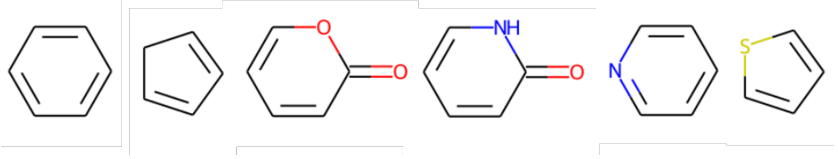


FIGURE 5.14: Ring building blocks used to create and modify the most complex set of 'reduced' population molecules, attempting to capture the variety of ring OCELOT fragments while working in a smaller space.

The 16 'reduced' initial populations were generated with the same approach as the full populations, completed in four groups of 400 species, giving four populations of 100 molecules for each subset of building blocks that was chosen.

Again, with analysis of these populations by Tanimoto similarity, and heatmaps showing only the average pairwise similarity are shown in Figure 5.15, a small level of overlap between some populations built with different subsets of building blocks is seen, with a a maximum of seven shared molecules; some molecules in these populations are also found in the full populations.

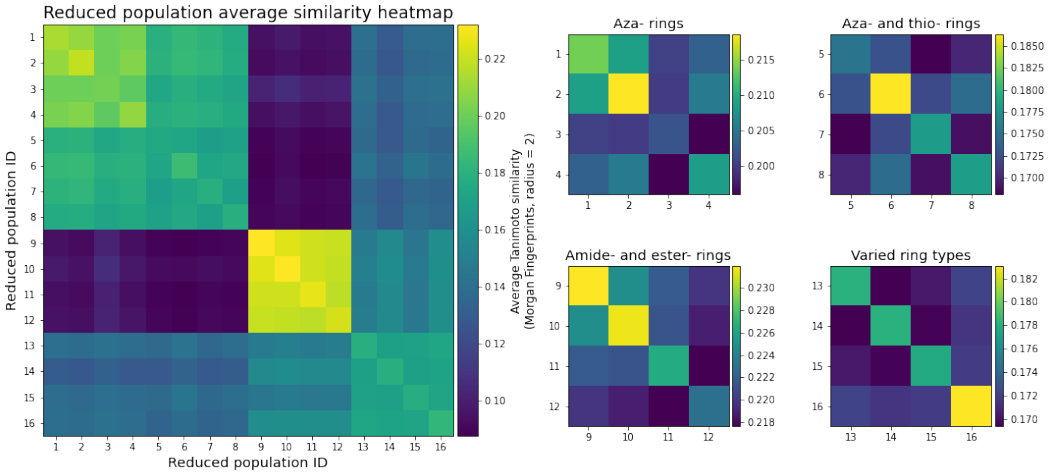



FIGURE 5.15: Heatmaps comparing the Tanimoto fingerprint similarity between 16 initial 'reduced' populations constructed with subsets of the OCELOT building blocks, also separated by the type of subset used.

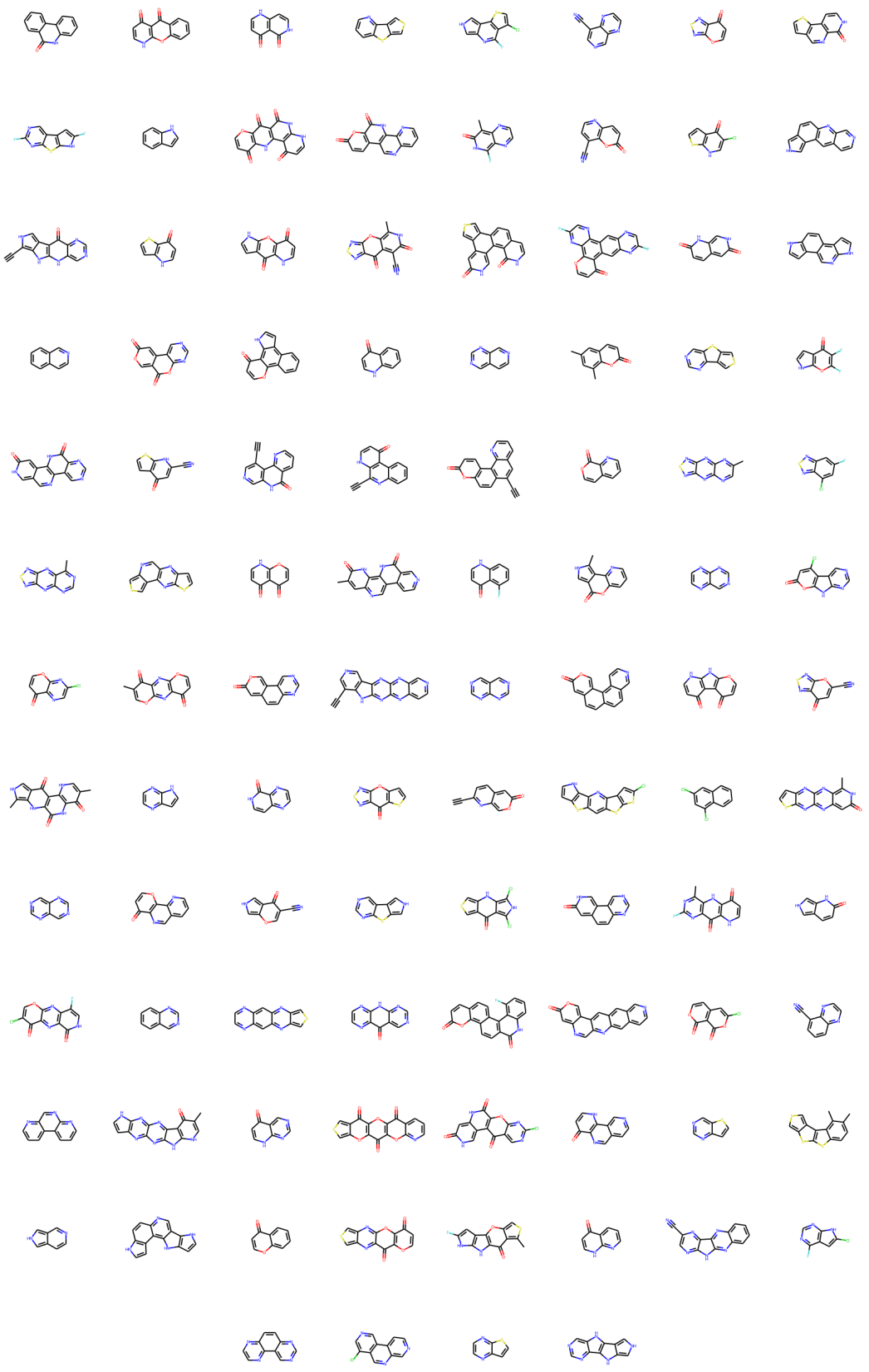


FIGURE 5.16: 100 molecules generated for initial 'full' population 1

## 5.3 Combi-HOFs - porous frameworks

A side project in collaboration with ADAM (https://cordis.europa.eu/project/id/856405), working towards the Autonomous Discovery of Advanced Materials, utilised the 'reactor' approach of MolBuilder in the exploration and discovery of molecules which potentially hydrogen-bonded frameworks (HOF).

This was achieved through the combination of reagents and functionalisation reactions, in order to generate a library of candidates for analysis via computational predictions and in-lab experimentation.

### 5.3.1 Combining cores and end groups

The molecular 'cores', selected to contain ortho-dianiline substructures, are treated as 'reagents' to be combined with functionalisation reactions, converting ortho-dianiline groups into hydrogen-bonding capable ring structures.

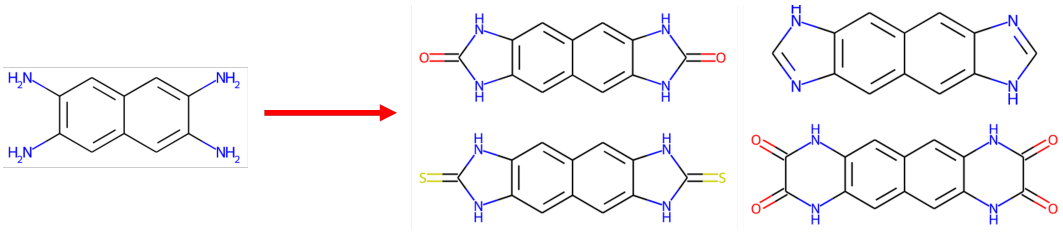


FIGURE 5.17: Given a core with reactive groups (in this case, ortho-dianiline), a set of functionalisation reactions can be applied to generate reaction products.

### 5.3.2 Core molecules

In order to select a set of rigid molecular cores to be used in this project, 9,822 ortho-dianiline containing molecules were first extracted from Reaxys by Filip Szczypinski, a collaborator from the ADAM team.

These were then subjected to a set of filters, aiming to find candidate cores which were both compatible with MolBuilder and easier synthetic targets; the following criteria were applied to remove molecules:

- Cores with any flexibility identified by RDKit's `rdMolDescriptors.CalcNumRotatableBonds()`.
- Cores with non-ring bridges between fused ring systems
- Cores with flexible groups attached which RDKit did not identify, such as tertiary butyl.
- Cores with high molecular weight, any value above 400 units as calculated by RDKit
- Cores with atoms outside of FIT forcefield; only H, C, N, O, S, F and Cl were permitted.
- Any remaining cores which do not contain an aromatic ortho-dianiline substructure, this time detected through the use of SMARTS pattern matching `(c(-[N&H2]):c(-[N&H2]))`.

This process left a set of 740 cores for use in the MolBuilder 'reactor' tool, a selection of which are shown in Figure 5.18.

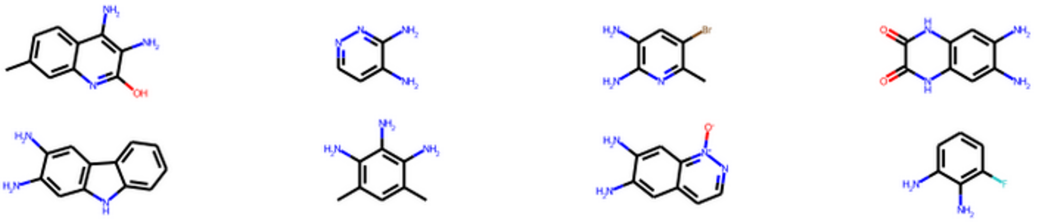


FIGURE 5.18: Example molecular cores which fit the criteria applied, and were subsequently passed through computational 'reactions' to generate potential combi-HOFs

### 5.3.3 Functionalisation reactions

Four functionalisation reactions were selected, which target ortho-dianiline groups and convert them into hydrogen-bonding capable ring structures; these reactions are shown in Figure 5.19.

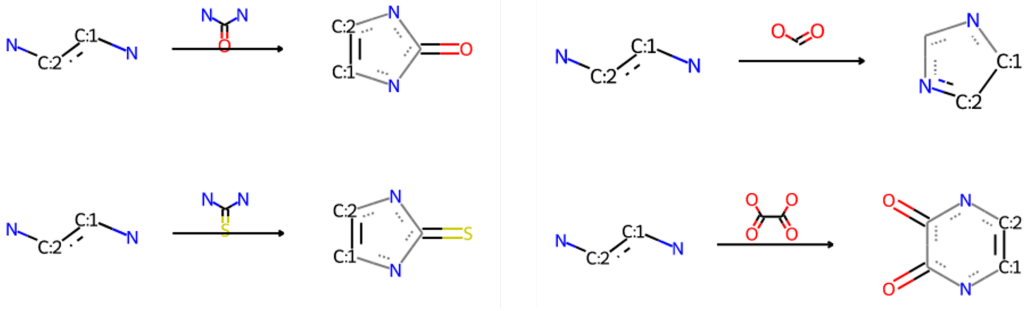



FIGURE 5.19: The four selected functionalisation reactions, which convert ortho-dianiline groups into ring structures.

### 5.3.4 Generating a library of structures

Using the 'reactor' molecular generation process outlined in the coding development chapter (Figure 4.24), the selected cores and reactions were combined to produce a full library of candidate combi-HOF molecules, including both fully and partially 'saturated' products; 3,036 of these were fully rigid molecules, and 2,549 were left with unreacted amine groups.

### 5.3.5 Next steps for the Combi-HOFs

This library of structures was subjected to further computational analysis by other members of the Day Group, using predicted crystal structures to determine if any low-energy packing arrangements of these molecules in the solid state exhibit porosity; the library of candidates was also provided to an experimental team at the University of Liverpool, for synthesis and characterisation.

# Chapter 6

# Exhaustive chemical space exploration

This section describes the results from initial full chemical space exploration campaigns, after changes made during MolBuilder development.

## 6.1 Azapentacenes

The first of these campaigns was conducted using a class of aza-substituted pentacene molecules, referred to as the azapentacenes in this work; the previous iteration of MolBuilder was also used to run exhaustive exploration in this space, so was suitable for use to confirm that all previously accessible species could still be generated.

### 6.1.1 Generating a class of similar molecules

In order to confirm that the new molecular generation approach could reproduce previous datasets, a new full exploration was performed. The set of molecular building blocks required to access this space is minimal, containing only one starting molecule, one fragment type and a single mutation operation.

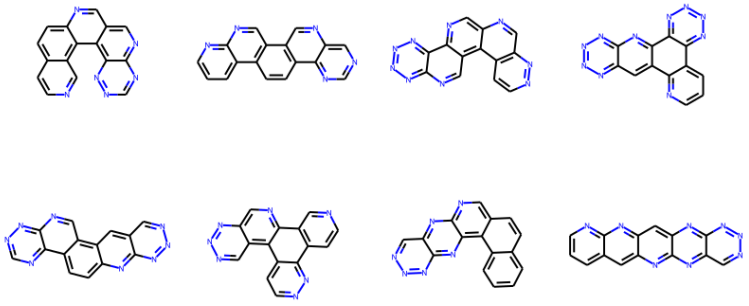


FIGURE 6.1: A selection of azapentacene-type molecules generated using MolBuilder, using the 'frags-in-situ' approach to build an exhaustive list of unique species.

The definitions of these are shown in the project setup chapter, under Listing 5.1. As described in previous sections, building blocks are combined using genetic operations to produce molecules; in a full chemical space exploration campaign, molecules are endlessly generated and checked against a 'master list' of unique discoveries.

In the case of exploring azapentacene chemical space, there are a limited number of unmutated 'molecular backbones' available given only one ring type, and a limit to five-ring molecules. Further ring patterns, such as pyrene-type arrangements, were excluded from this search through the limits on molecular operations available with MolBuilder; as stated earlier, only 'two-point additions' of rings are used, meaning additions to 'cove' and 'bays' are were forbidden (Figure 5.2). Exploration was split up into twelve 'sub-exploration campaigns', each of which only searched for molecules with a certain shape; these backbone shapes are shown in Figure 6.2.

Rate of discovery plots (such as in Figure 6.3) can be used to monitor the progress of these campaigns, where convergence in the number of unique discoveries signals a completed run. The backbones align into three groups where a similar number of mutation patterns are discovered, as a result of the number of aromatic carbons available for mutation; backbones 3 and 11 are of particular note, as due to their symmetry they exhibit lower numbers of unique mutation patterns than other backbones.

This new search revealed that the original azapentacene space from past work by Cheng et al. (2020) only represented roughly half of the feasible molecules which

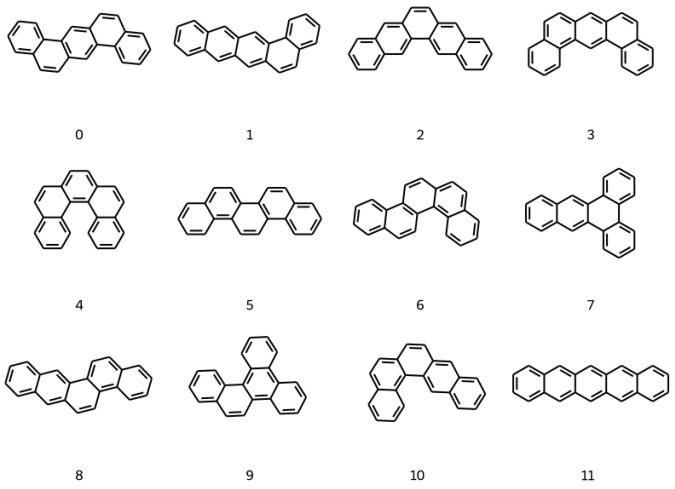



FIGURE 6.2: The 12 molecular backbones which all azapentacene species in the space explored are built from.

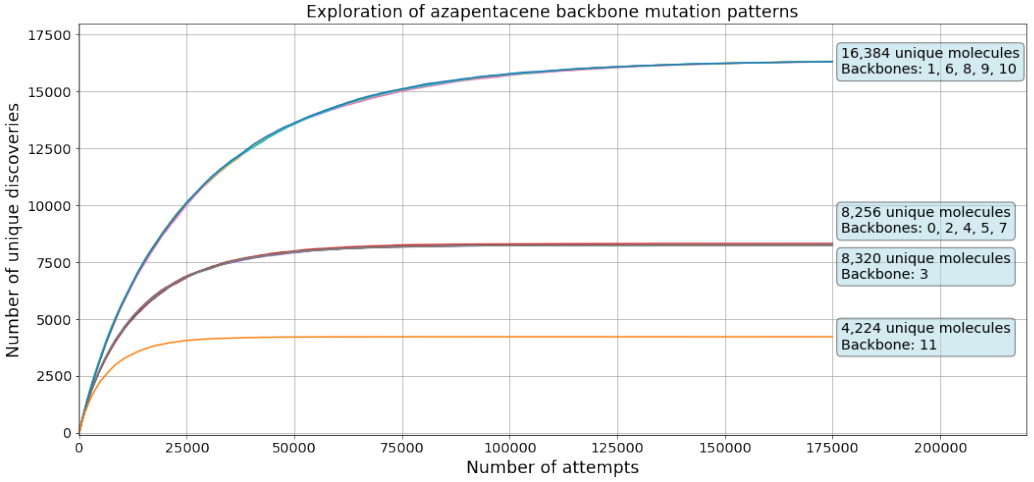



FIGURE 6.3: Discovery rate plot for the exploration campaigns of mutation patterns for each carbon backbone in the azapentacene chemical space.

could be constructed with these building blocks. On further inspection, it was found that the original MolBuilder code used in this work had an additional, unintended limitation. Since the mutation operation was always performed in 'pairs', molecules with an odd number of mutations could not be accessed.

In other words, an azapentacene-type molecule with, for example, 3 nitrogen heteroatoms, would not be discovered in the original exploration campaigns. The updated MolBuilder molecular generation code is able to perform single mutations, avoiding this issue. As a result of this, the azapentacene chemical space is now estimated to contain 135,744 species.

### 6.1.2 Describing and separating a class of similar molecules

Issues arise when it comes to describing and categorising the azapentacenes, as the composition of these molecules is very similar across the entire chemical space. Due to the design rules imposed by the limited set of building blocks:

- Elemental composition is limited to carbon, nitrogen and hydrogen
- The number of aromatic rings is always five
- No side chains or additional functional groups are present

These factors make it more difficult to partition the azapentacenes space into smaller sub-spaces for more focused analysis; This subsection focuses on the ways in which large sets of molecules with similar compositions can be grouped.

#### 6.1.2.1 Elemental composition

A simple metric to categorise azapentacene structures can be found in their empirical formulae, particularly the amount of nitrogen present in the molecule. While this doesn't give as much information as a structural descriptor, the level of nitrogen substitution is a rapid and easy-to-digest approach to clustering the azapentacene set. Table 6.1 illustrates how many molecules fall into each category of nitrogen substitution level.

| Empirical Formula | # Structures | Empirical Formula | # Structures |
|---|---|---|---|
| $C_{22}H_{14}$ | 12 | $C_{14}N_8H_6$ | 24906 |
| $C_{21}N_1H_{13}$ | 117 | $C_{13}N_9H_5$ | 16539 |
| $C_{20}N_2H_{12}$ | 777 | $C_{12}N_{10}H_4$ | 8337 |
| $C_{19}N_3H_{11}$ | 3012 | $C_{11}N_{11}H_3$ | 3012 |
| $C_{18}N_4H_{10}$ | 8337 | $C_{10}N_{12}H_2$ | 777 |
| $C_{17}N_5H_9$ | 16539 | $C_9N_{13}H_1$ | 117 |
| $C_{16}N_6H_8$ | 24906 | $C_8N_{14}$ | 12 |
| $C_{15}N_7H_7$ | 28344 | | |

TABLE 6.1: Azapentacene structure counts after grouping by empirical formula; equivalent to clustering by nitrogen substitution level.

#### 6.1.2.2 Molecular shapes and scaffolds

A key difference between azapentacene structures is the way in which aromatic rings are combined, forming linear, bent and branched aromatic systems. This overall molecular shape can be captured through the use of generic representations like Murcko Scaffolds.

These scaffold representations, which can be considered equivalent to the 'carbon backbone' of a molecule, are generated by converting all atoms to carbon, and removing any attached side groups; a single 'backbone' then describes all molecules which share that molecular shape, regardless of the nitrogen substitution pattern.

The 135,744 azapentacene molecules can be neatly categorised under 12 scaffold types, as shown in Figure 6.2. Given this information, general trends in molecular properties and synthetic accessibility which arise from molecular shape could be determined, if they exist. Partitioning the space like this also allows for batches of similarly shaped molecules to be processed with further calculations.

## 6.2 OCELOT - Organic Semiconductors

As with the azapentacenes, an attempt was made to exhaustively explore the chemical space available given the selected set of building blocks; twelve ring types and five mutation types, constructing molecules containing between two and five rings, and up to two mutations.

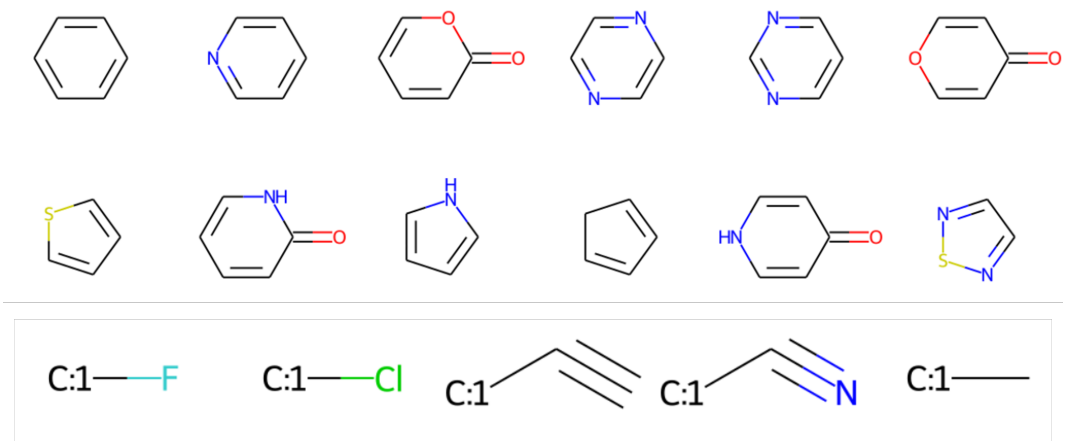



FIGURE 6.4: The selection of building blocks and mutation types used to exhaustively generate molecules 'surrounding' the OCELOT chemical space

This exploration was stopped after just under 13 million unique structures were obtained under the chosen design rules; even at this point, the campaign showed little sign of slowing down, indicating that the actual number of molecules, even with our size limits and restricted ring types, exceeds this count by a long shot.

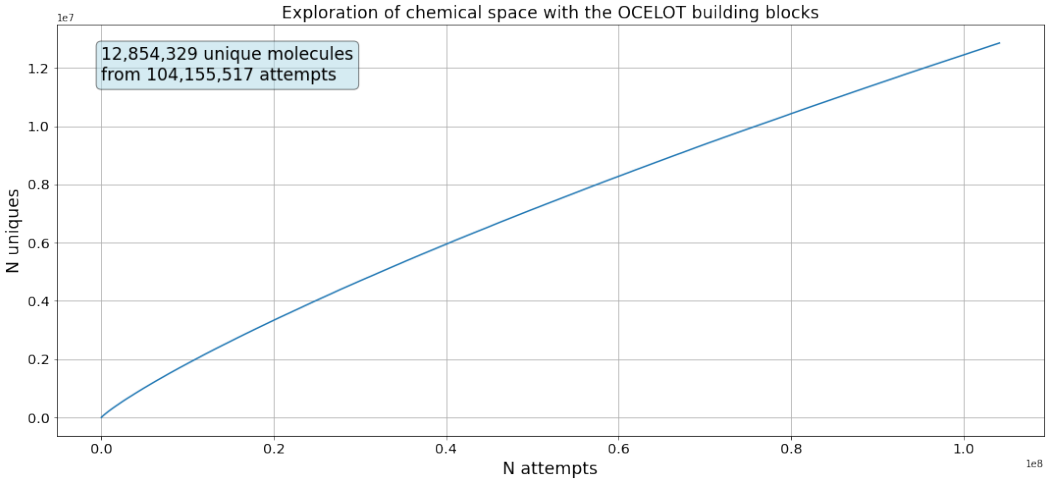



FIGURE 6.5: Discovery rate plot for the exploration campaign using OCELOT building blocks.

The process also started to run into issues with updating and storing the 'master list' of unique discoveries; due to the large number of molecules, checking newly generated species against this list became slower as the exploration progressed. A selection of these discovered molecules is shown in Figure 6.6.

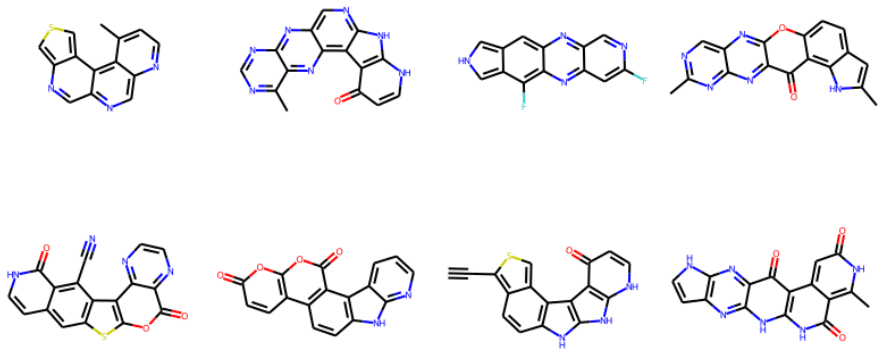


FIGURE 6.6: A selection of molecules generated from the OCELOT building blocks using the MolBuilder 'frags-in-situ' approach to build an exhaustive list of unique species.

### 6.2.1 Exploration using reduced building block sets

A second attempt at the exhaustive exploration of the OCELOT space was conducted by running four separate processes, one for each 'reduced' building block set outlined in the project setup chapter; the exploration progress plots are shown in Figure 6.7.

While these explorations did not reach a convergence in the number of unique molecules either, they provide some insight into the relative sizes of each 'reduced' space, and how the size of the 'master list' of unique species impacts the number of attempts at molecular generation that can be made.

These four explorations were all given the same amount of computational 'runtime', and the same amount of resources; as the number of building block types grows, more unique molecules are discoverable early on in the search, but this also means the list of molecules to check against for new discoveries grows and becomes harder to search.

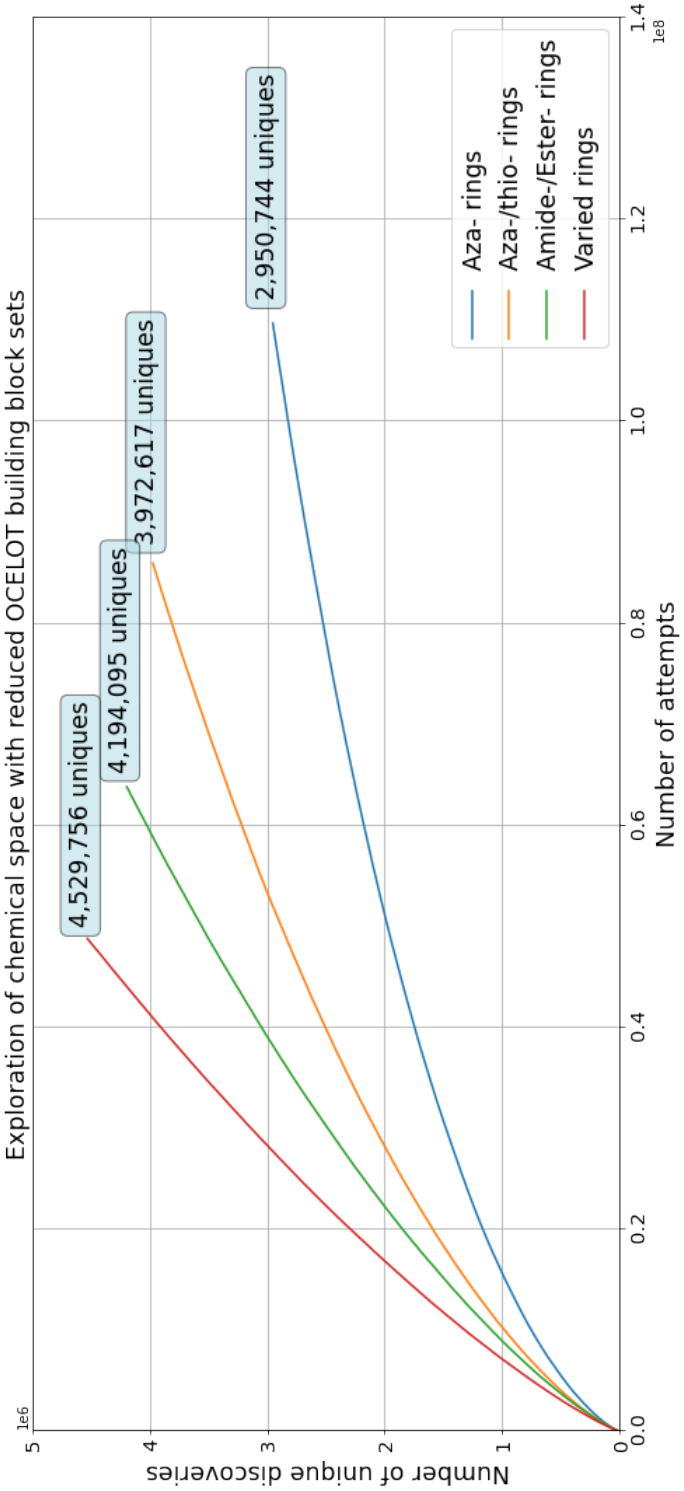



FIGURE 6.7: Discovery rate plot for exploration campaigns using the four 'reduced' sets of OCELOT building blocks.

## 6.3 Conclusions on full chemical space exploration

This chapter has described full exploration of the azapentacene chemical space, and the attempted full exploration of the 'OCELOT' chemical space.

A simple set of aza-substituted 6-membered aromatic ring building blocks leads to 135,744 possible five-ring molecules; increasing the number of building blocks through to 12 common ring types and 5 side group 'mutations' leads to an explosion in chemical space size, which is too large to explore using the same methodology, likely surpassing 13 million unique molecules.

Conducting these campaigns highlights the need to develop more efficient approaches to chemical space explorations, where even limited sets of molecular building blocks and restrictions on design rules lead to overwhelmingly large potential sets of molecules to examine.

This leads on to the following chapters, discussing directed chemical space exploration through the use of a genetic algorithm, where a fitness function is used to optimize one or more properties and guide molecular generation towards species which appear to exhibit more promising scores.

# Chapter 7

# Molecular reorganisation energy GAs

The first set of genetic algorithms performed with building blocks extracted from the OCELOT dataset were set up to minimize molecular electron reorganisation energy.

This fitness function was chosen as a low energy barrier for changes in geometry on gain/loss of an electron is a promising property for organic semiconducting molecules. Reorganisation energy calculations have also been utilised in previous MolBuilder molecular discovery campaigns, and showed success in the azapentacene chemical space, so starting work in the OCELOT space with this calculation made sense. The final dataset of molecules sampled can be found at https://doi.org/10.5258/SOTON/D3745.

## 7.1 Behaviour of genetic algorithms

Before analysing the molecules sampled during the first portion of this project, it's worth discussing the behaviour of the genetic algorithms themselves.

### 7.1.1 General behaviour

As stated in earlier sections, the first 20 genetic algorithms were constructed to start from 20 unique 'initial populations'. There were no molecules shared between these populations, so they should have the opportunity to explore different sections of the available chemical space; this should increase the number of molecules that get sampled, and could potentially highlight promising molecule types if multiple 'isolated' GAs discover similar species, or their populations converge with each other and discover similar classes of molecule.

Overall, nearly 29,000 unique molecules were sampled across the 20 reorganisation energy genetic algorithms, less than 0.3% of the near 13 million molecules discovered through undirected exploration using the same set of building blocks.

When examined separately, there is quite a variation in how many unique molecules each genetic algorithm sampled; at either limit, runs initiated from initial populations 18 and 11 sampled 1,012 and 2354 molecules respectively.

This could be put down to the randomized aspects of molecular generation, where run 11 was 'choosing' more varied outcomes; lower sampling in run 18 could also be a sign that starting from that initial population resulted in less need for exploration of the chemical space, and the genetic algorithm was able to find promising candidates without much variation of the molecules in its population.

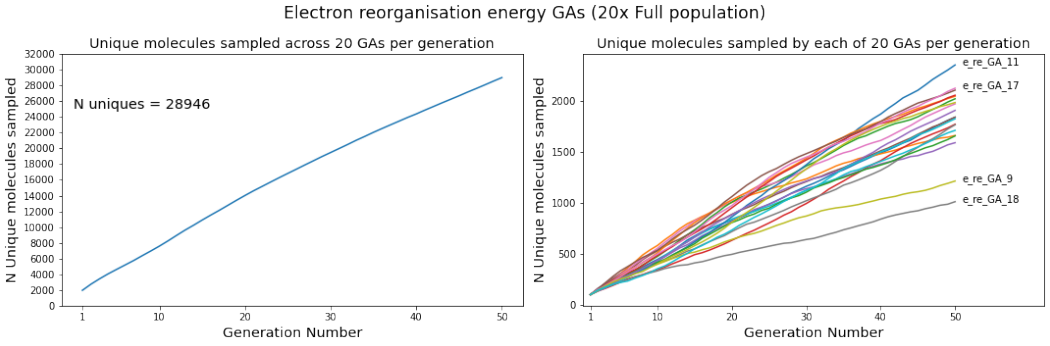



FIGURE 7.1: Plots to show how many unique molecules were sampled: by all 20 GAs (left); by each of the 20 GAs separately (right)

The behaviour of these genetic algorithms can be examined further by determining how often molecules 'stick around' from one population to the next. A certain level of this is expected, due to the 'elitism' approach when constructing a new population; there will also be cases where a crossover operation results in a previously sampled molecule 'reappearing'.

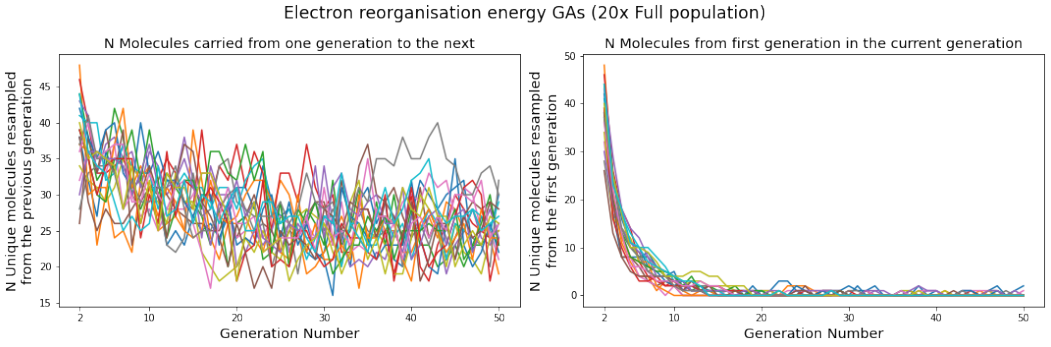



FIGURE 7.2: Plots to show how many molecules get 'carried over' from between generations: counts of how many molecules from the previous generation reappear (left); how many molecules in the current population were also in the initial population (right)

As shown in the left portion of Figure 7.2, anywhere from 20-30% of each population is carried over from one generation to the next at most points during the genetic algorithm runs.

The right portion of Figure 7.2 reveals how often molecules in the initial population of each genetic algorithm run are 'resampled' by later generations. In all cases the reappearance of molecules from the initial populations is very limited, after only a few generations. This is reassuring, as the initial generations were not constructed with optimal reorganisation energies in mind; using the genetic algorithm to optimize this property from essentially 'random' sets of molecules has resulted in mostly different populations as the algorithm progresses.

The diversity of molecule populations changes as genetic algorithms progress, as shown in Figure 7.3; this plot confirms that all 20 initial populations of molecules are relatively diverse, with high average pairwise fingerprint diversity scores.

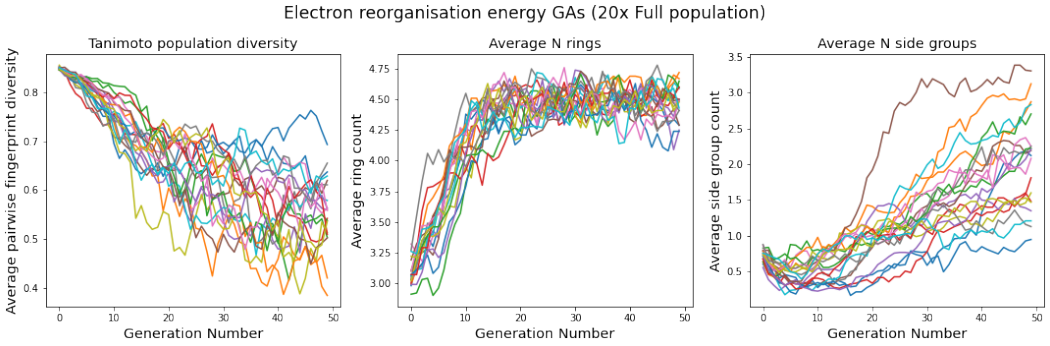



FIGURE 7.3: Plots to show diversity and composition for each population: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right)

As the algorithm progresses, a drop in diversity can be seen in all cases, by various amounts; this is a positive result, as the molecules within each population are becoming more similar, implying the discovery of an area of chemical space with promising properties, since the algorithm is 'exploiting' molecules with good properties, rather than 'exploring' new options.

Examining molecular composition of population members as these genetic algorithms progress can shed light on what kind of molecules perform well according to the desired fitness function; the middle and right plots of Figure 7.3 show clear trends, where molecules with more rings and more side groups are sampled by later generations. Combine this with Figure 7.4, and we can see that molecules with this type of composition are on average exhibiting lower reorganisation energy values.

### 7.1.2 Minimizing electron reorganisation energy

The target of running these genetic algorithms was to discover molecules with low electron reorganisation energies; the performance on this front can be analysed by plotting the average (and minimum) fitness values of each population, as in Figure 7.4 and Figure 7.5.

Overall a clear reduction in the average fitness scores is observed as the genetic algorithms progress, but it is worth noting that from generation to generation, there are several cases where the average increases instead; this is an artefact of the

'randomness' which helps to drive these algorithms. Even if two 'parent' molecules exhibit good reorganisation energies, the 'children' which are generated by crossover of those parents will not necessarily share that behaviour.

It is easier to see the general minimization of properties found by sampling molecules with a genetic algorithm by producing a similar plot, shown on the right portion of Figure 7.4, where the average fitness score from the top 100 sampled molecules up to a given generation is shown.

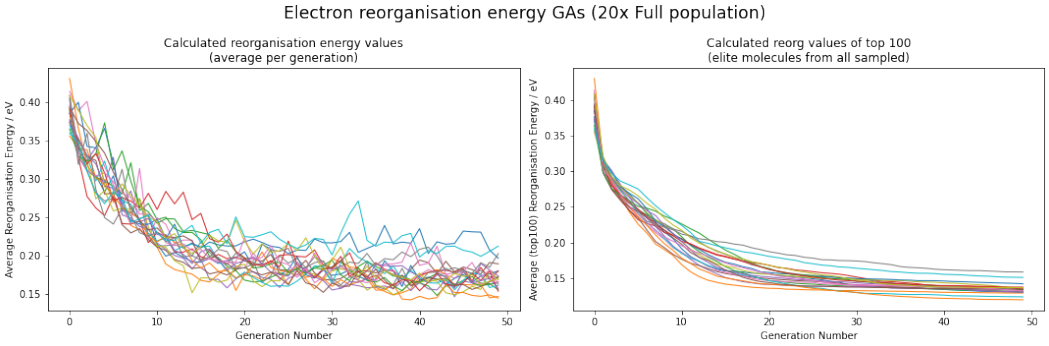



FIGURE 7.4: Plots to show how the fitness values (molecular electron reorganisation energy) change as each of the 20 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right)

Looking at the minimum reorganisation energies reveals cases where the calculated value seems unreasonable, exhibiting negative or very low values. These results likely arise from the approach to reorganisation energy calculations undertaken; neutral and charged geometries were both obtained through optimizations using the same 'rough' starting geometry. For some molecules, this can lead to cases where these optimizations lead to different 'energy basins', relating to overly different geometries. The calculation has since been fixed, instead the charged geometry is determined through optimization using the neutral geometry as a starting point; this more closely matches the process of reorganisation, where a neutral molecule (in the optimal geometry) becomes charged and adjusts its geometry to account for the change.

Setting a threshold (as in the right portion of Figure 7.5) shows that the 20 GAs were all able to find molecules with low, yet reasonable, reorganisation energies within a window of 0.1-0.12 eV.

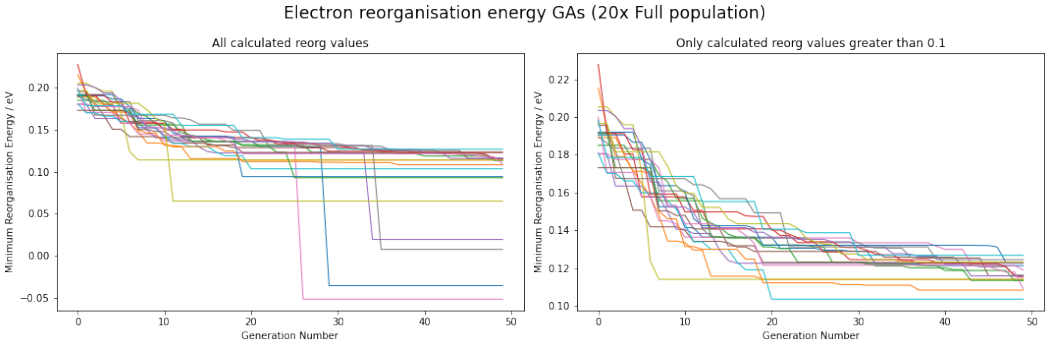



FIGURE 7.5: Plots to show how the fitness values (molecular electron reorganisation energy) change as each of the 20 GAs progress: minimum values per generation (left); minimum values above a 'reasonable' threshold (0.1 eV) of reorganisation energy values (right)

Examining the top-performing portion of molecules sampled by each genetic algorithm reveals some interesting information; while the top performers for most GAs are linear 5-ring systems, those started with initial populations 11 and 18 have sampled more interesting molecules with different compositions.

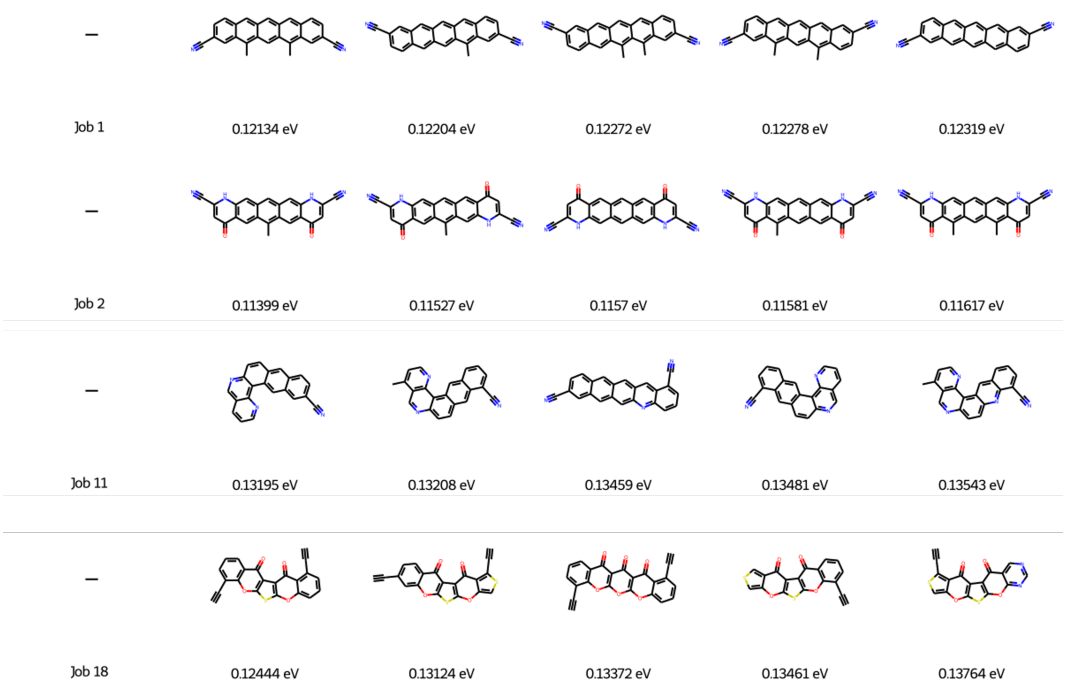



FIGURE 7.6: Top 5 molecules sampled by each of 4 jobs, started from populations 1, 2, 11 and 18.

### 7.1.3 Convergence of GAs from different starting points

Since the 20 genetic algorithms started from unique initial populations, and they are exploring such a large space, any overlap between generated populations is worth investigating. This can be done at two 'levels', as shown in Figure 7.7: overlap per generation, and cumulative overlap (counting how many jobs have sampled a certain molecule at any point up to a given generation).

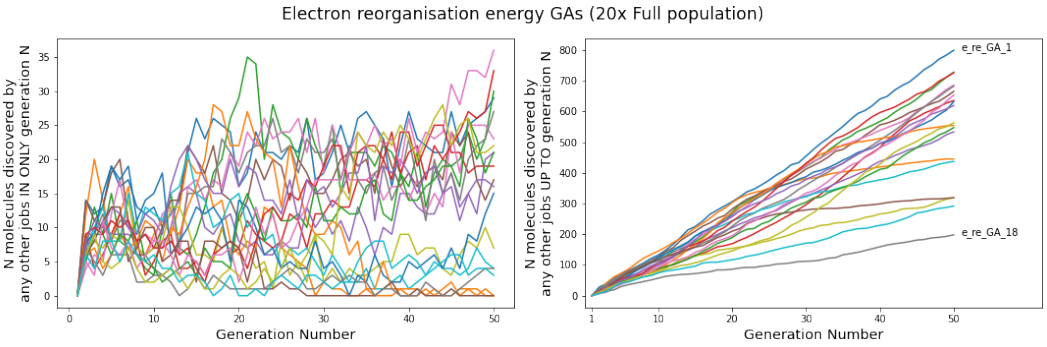



FIGURE 7.7: Plots to show overlap, or molecules shared, between populations (left) and between molecules sampled up to a given generation (right)

Of these 20 genetic algorithms, the run started from initial population 18 consistently sampled populations which had the least overlap with equivalent populations from other jobs; this couples well with earlier analysis such as in Figure 7.1, which indicates that this GA also sampled the fewest number of unique molecules overall.

These results could indicate that starting with initial population 18 leads to a portion of the available chemical space that other initial populations have more trouble reaching. However, there is only one example of a genetic algorithm for each initial population, so the behaviour could also be put down to random chance; in the following section, a selection of these GAs are repeated, to see if this behaviour reoccurs.

## 7.2 Repeating GAs from the same starting point

Of the 20 reorganisation energy genetic algorithms run up to this point, five were selected to undergo repeat runs starting from the same initial population, and with the same configuration. Even when these parameters are kept the same, a given GA is able to explore in a different manner due to the random aspect of genetic operations. For each of these initial populations, five repeats were conducted.

The initial populations to repeat were chosen based on the results gathered so far, in an attempt to gather the most interesting cases for additional runs:

- Populations 2 and 12 exhibited the best average reorganisation energies at their final populations.
- Populations 5 and 18 exhibited the best minimum reorganisation energies at the final population.
- Population 18 was of particular interest as this GA run sampled the lowest number of molecules, and had the least overlap with the other 19 runs (see Figure 7.7).
- Population 20 exhibited the worst average reorganisation energy at the final population.

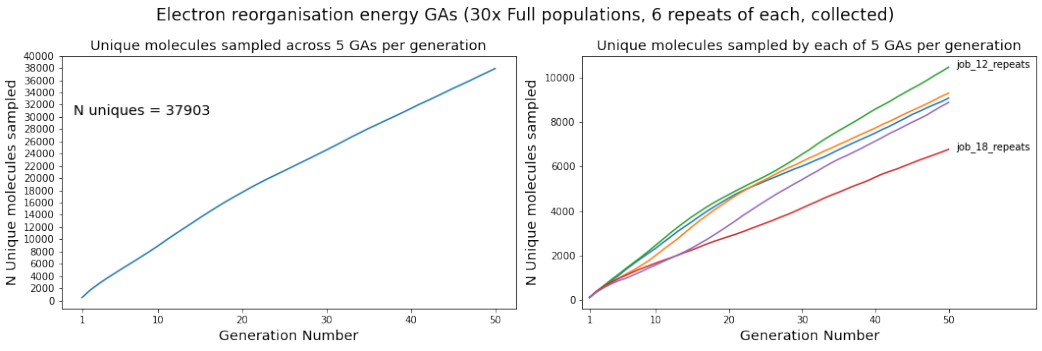



FIGURE 7.8: Plots to show how many unique molecules were sampled: by all 30 GAs using the chosen repeat populations (left); grouped by each 5 initial population types (right).

A summary of the repeat runs is shown in Table 7.1; for each repeated initial population, the total number of unique structures and average reorganisation energy values are listed.

| Population ID | Number of Structures | Average Reorg /eV | Top 100 Average Reorg /eV | Top 100 Average ($> 0.1$) Reorg /eV |
|---|---|---|---|---|
| 2 | 9090 | 0.2425 | 0.1190 | 0.1211 |
| 5 | 9313 | 0.2419 | 0.1166 | 0.1185 |
| 12 | 10486 | 0.2377 | 0.1131 | 0.1131 |
| 18 | 6782 | 0.2868 | 0.1018 | 0.1230 |
| 20 | 8894 | 0.2620 | 0.1189 | 0.1202 |

TABLE 7.1: Summaries of the repeat runs conducted for 'full population' reorganisation energy genetic algorithms.

Genetic algorithm campaigns initiated from population 18 consistently discovered a lower number of molecules than the other populations across all repeats. The 5 new repeats all sampled more than the original run for population 18, effectively exploring different 'routes' through the chemical space, sampling a total of only 6,782 unique molecules between the 6 repeat runs.

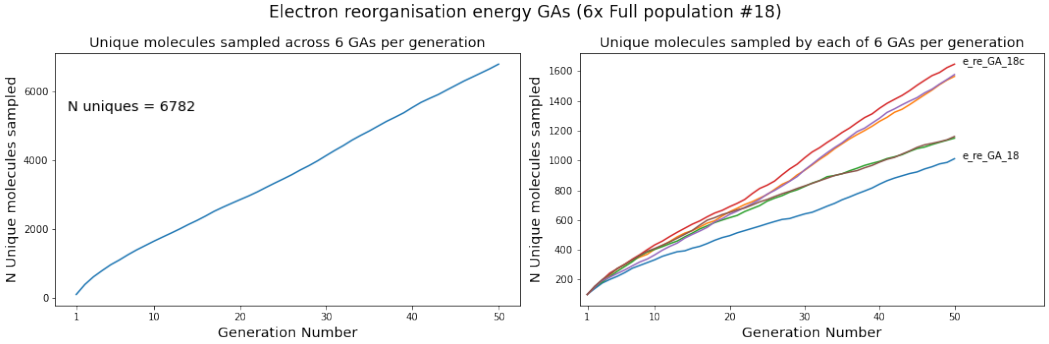



FIGURE 7.9: Plots to show how many unique molecules were sampled: by all 6 GAs using population 18 (left); by each of the 6 GAs separately (right).

These runs exhibit the lowest average reorganisation energy amongst the top 100 sampled molecules, due to the appearance of some more unique structures with low calculated reorganisation energies. These mostly move away from the typical modified pentacene-type structures found in top performing sets that the majority of these genetic algorithms discover, with non-linear fused ring systems frequently containing thiophene rings; the top five sampled molecules from each repeat are shown in Figure 7.10.

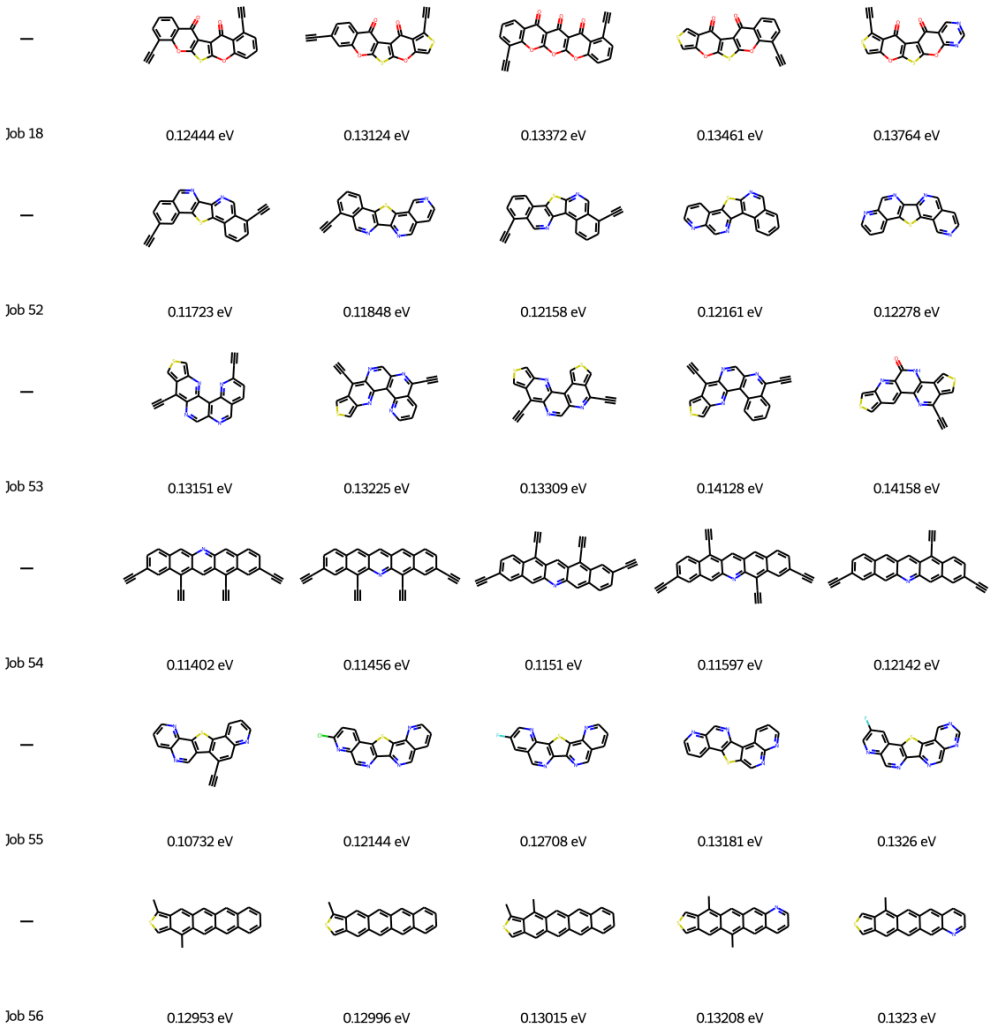



FIGURE 7.10: Grid showing the top 5 molecules discovered in each repeat run from population 18

In contrast, the campaigns started from population 12 sampled the most unique molecules, with 10,337 discoveries across the 6 repeat runs, with the lowest average reorganisation energy value amongst all sampled molecules.

These jobs mostly discovered modified pentacene-type structures as top performers, with varying substitution patterns and ring types in each instance. The grids of top performers in Figure 7.10 and Figure 7.12 also highlight the prevalence of cyano- and ethyne- side groups; for linear patterns, presence of such groups on each end of the molecular core appears to promote a reduction in reorganisation energy.

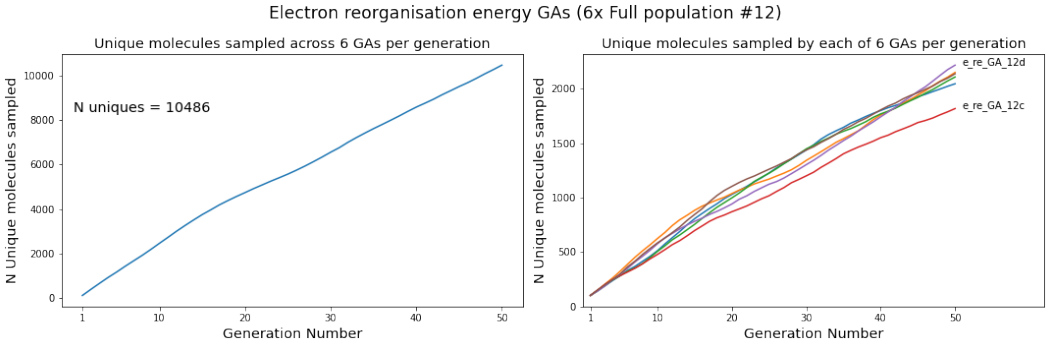



FIGURE 7.11: Plots to show how many unique molecules were sampled: by all 6 GAs using population 12 (left); by each of the 6 GAs separately (right).

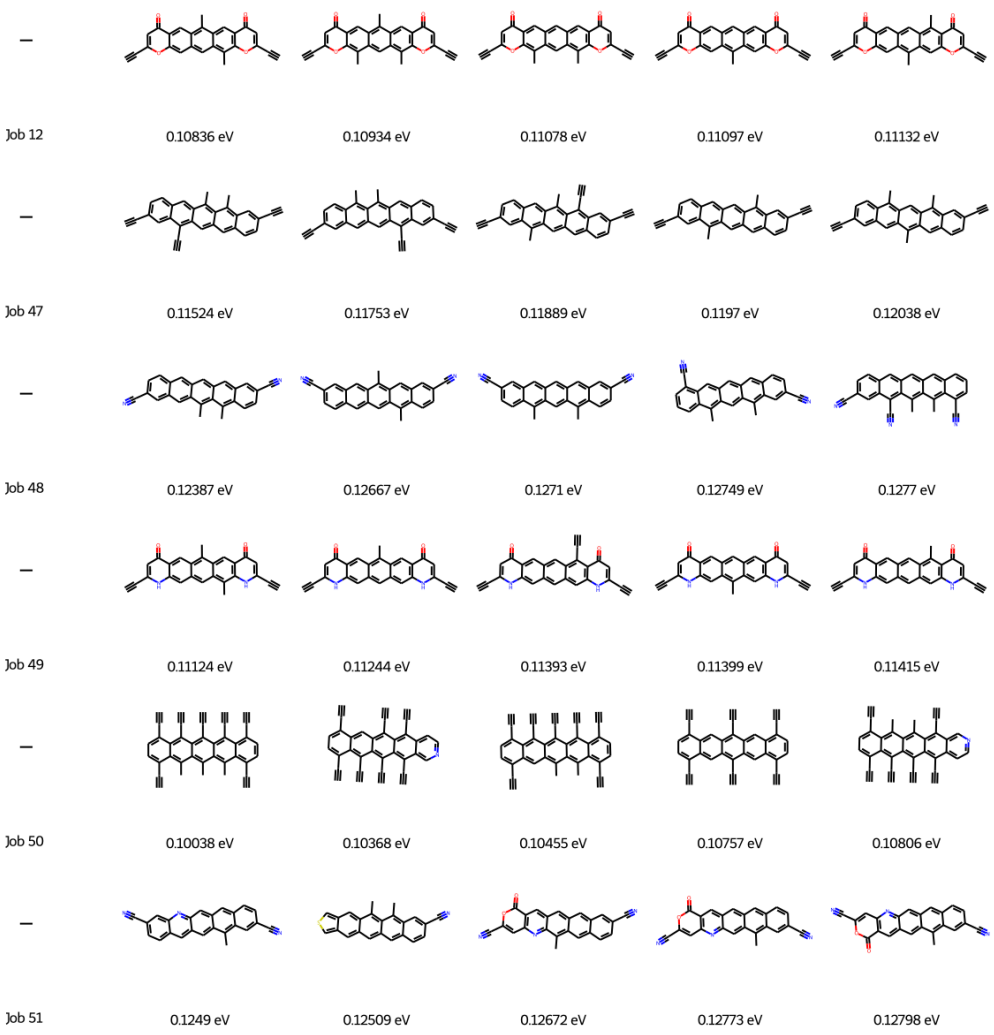



FIGURE 7.12: Grid showing the top 5 molecules discovered in each repeat run from population 12

### 7.2.1 Divergence of runs from the same initial populations

Running these repeat jobs allows analysis of the level of reproducibility that genetic algorithms exhibit, given the same parameters and initial population of molecules. This is achieved by determining the level of overlap between populations, meaning how many molecules are shared between runs for each generation; two approaches to this analysis were used in Figure 7.13 looking at both overlap per generation, and the cumulative overlap between sampled molecules up to each generation.

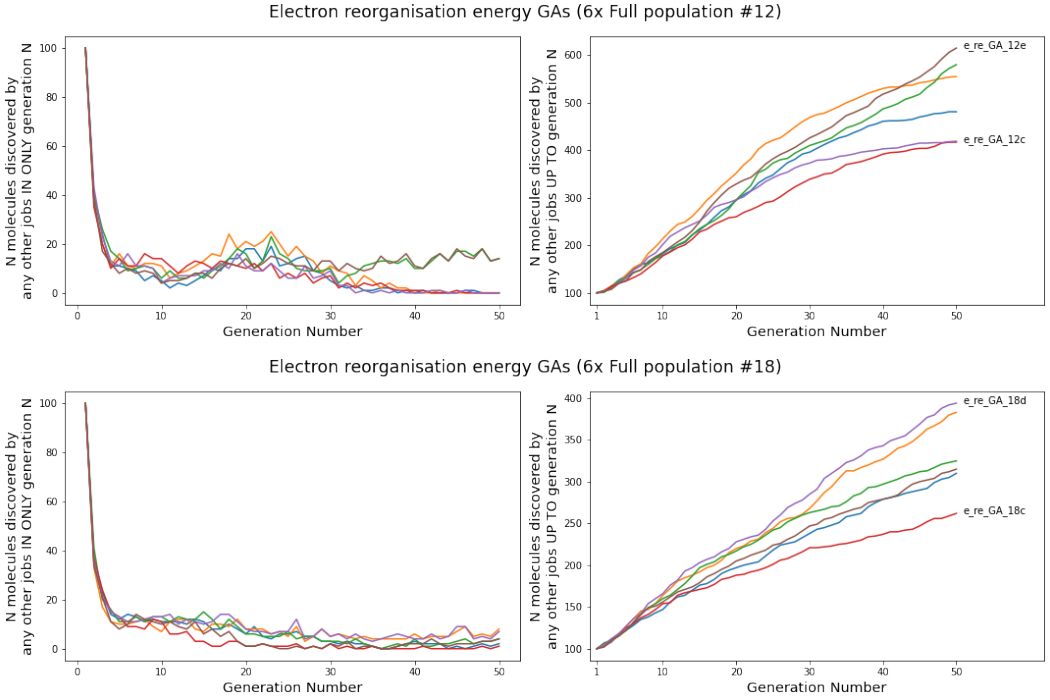



FIGURE 7.13: Overlap plots examining each set of 6 repeats for populations 12 and 18: per generation (left) and cumulative (right)

Each line in these plots corresponds to one job in the group, where the level of overlap is determined by the presence of molecules in any other job for/up to the given generation. Figure 7.13 suggests that for the most part, these genetic algorithms are taking different paths through the chemical space they can access, since the number of molecules shared per generation rarely rises above 20%; after completing, algorithms initiated from the same starting point end up sharing roughly 8%-12% of their overall sampled molecule sets.

## 7.3 Behaviour when the building block set is reduced

Up to this point, the genetic algorithms have all been using the same set of building blocks; 12 types of ring, and 5 types of 'side group'. This section will analyse the impact of reducing this set of building blocks in various ways; four subsets of the building blocks were chosen, with varying levels of complexity, effectively running the genetic algorithms in subsections of the total chemical space accessible with the full building block set. As stated in the project set-up chapter, these reduced sets use different selections of ring building blocks, and share a common set of three mutation types; these are shown in Figure 7.14.

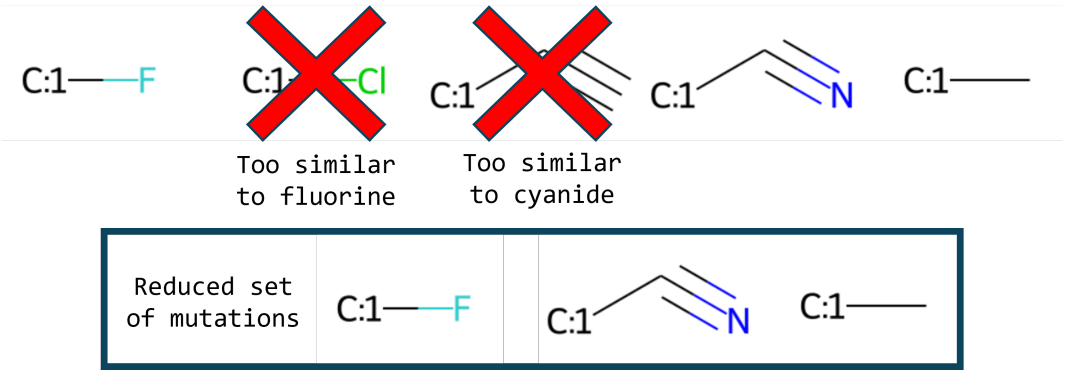



FIGURE 7.14: Three chosen side group types used in each reduced OCELOT building block set.

### 7.3.1 Aza-substituted fused ring systems

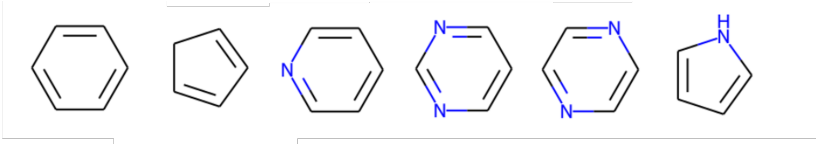



FIGURE 7.15: Ring building blocks used to create and modify the first set of 'reduced' population molecules: aza-substituted ring systems.

The simplest 4 'reduced building block set' GAs sampled a total of 6315 unique molecules, an average of roughly 1,580 each, which is an increase compared to the 20 'full building block set' GAs (which sampled on average roughly 1,450 each).

Reducing the building block set gave a slightly faster drop in reorganisation energy values, with multiple instances approaching 0.15 eV around generation 10, in

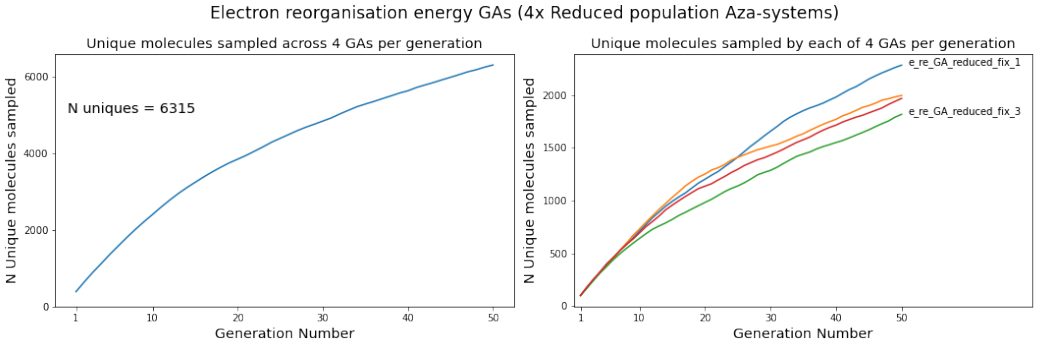



FIGURE 7.16: Plots to show how many unique molecules were sampled: by all 4 GAs using the aza-substituted ring system building blocks (left); by each of the 4 GAs separately (right)

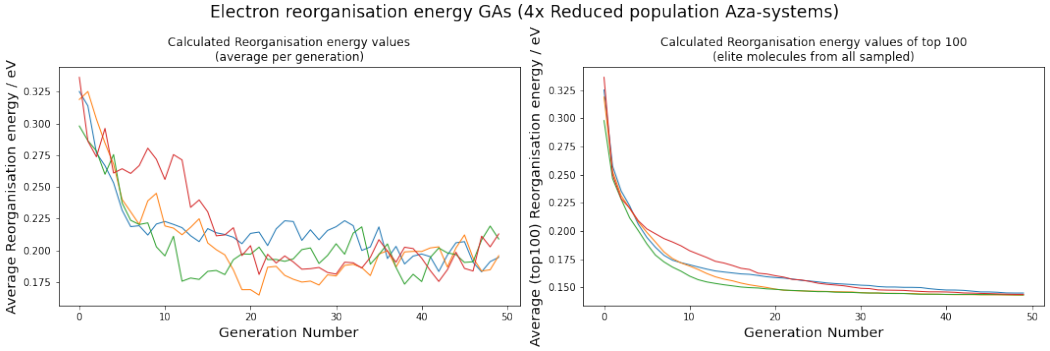



FIGURE 7.17: Plots to show how the fitness values (molecular electron reorganisation energy) change as each of these 4 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right)

comparison to the full populations which tended to hit these values around generation 15.

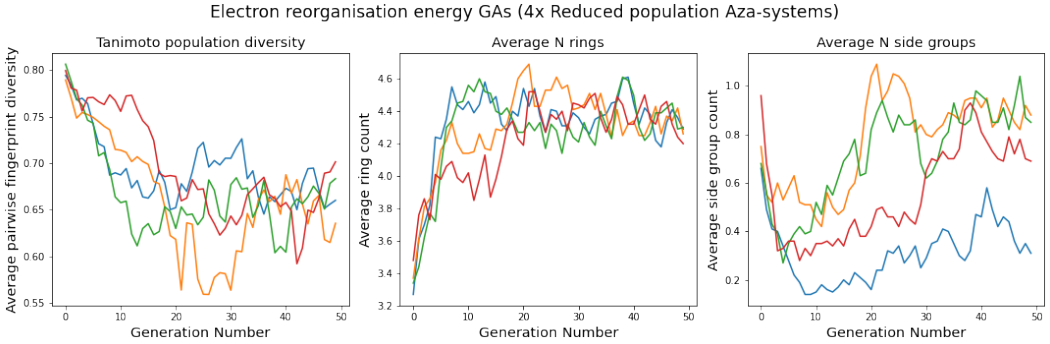



FIGURE 7.18: Plots to show diversity and composition for each population: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right)

Systems tended towards a slightly lower average number of rings of 4.3, compared to full populations 4.5, and a lower presence of side groups with a highest average of 1

compared to range between 1 and 4 for most full population runs; this could be due to the selected side groups not providing as much benefit in terms of reorganisation energy.

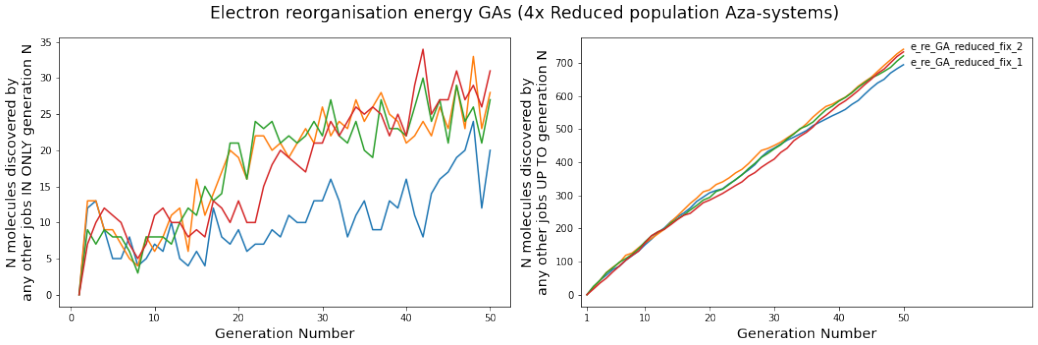



FIGURE 7.19: Overlap plots examining each set of 4 aza-system reduced population genetic algorithms: per generation (left) and cumulative (right)

In contrast to the genetic algorithms which have access to all building blocks, the first 'reduced' set exhibits higher levels of overlap between populations, despite starting from four unique populations; lowering the number of building blocks to this level appears to allow for faster convergence of genetic algorithms, likely due to the reduced number of accessible molecules.

### 7.3.2 Aza- and thio- substituted ring systems

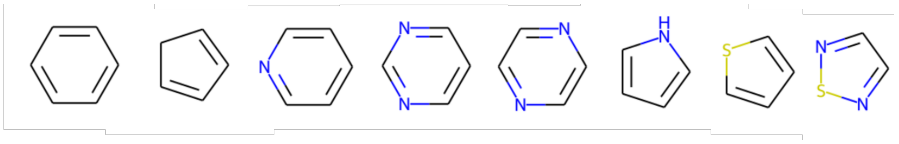


FIGURE 7.20: Ring building blocks used to create and modify the second set of 'reduced' population molecules: aza- and thio-substituted ring systems.

The next step-up in building block complexity, adding sulfur-substituted rings, sampled fewer molecules than the simplest reduced set above, discovering only 5,723 unique molecules across four campaigns; this result is unexpected, since an additional building block should result in a larger accessible chemical space.

Reorganisation energies converged slower than the aza-systems populations, approaching more stable average values around generation 20; this was accompanied by average ring and side group counts similar to the simpler reduced building block

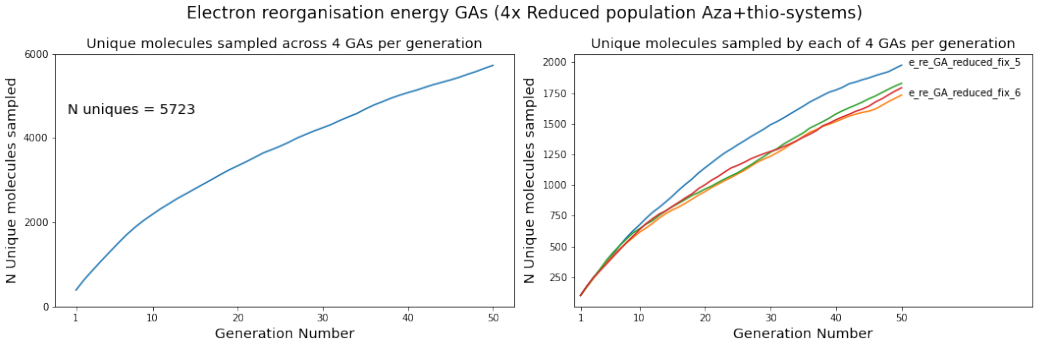



FIGURE 7.21: Plots to show how many unique molecules were sampled: by all 4 GAs using the aza- and thio-substituted ring system building blocks (left); by each of the 4 GAs separately (right)

genetic algorithms. The level of overlap between populations is similar to that of the aza-systems result, with up to 30% of molecules in a given population also being present in parallel jobs, and roughly 13% of all sampled molecules being shared with other jobs. Additional plots for these genetic algorithms can be found in the Appendix.

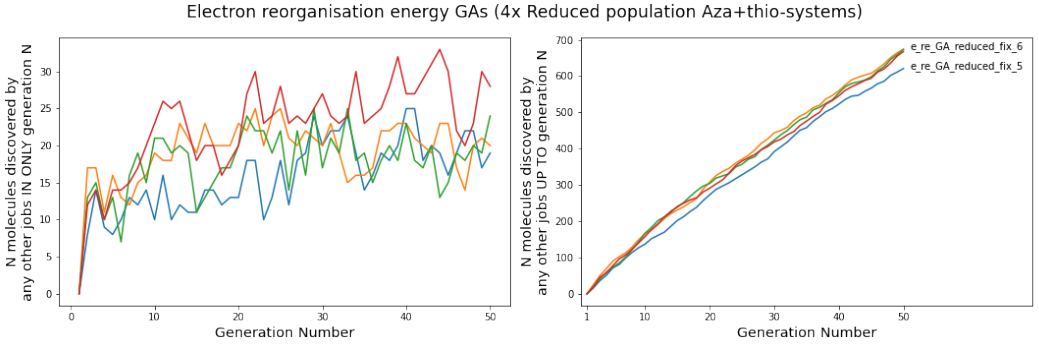



FIGURE 7.22: Overlap plots examining each set of 4 aza-/thio-system reduced population genetic algorithms: per generation (left) and cumulative (right)

### 7.3.3 Acenes with amide/ester-type rings

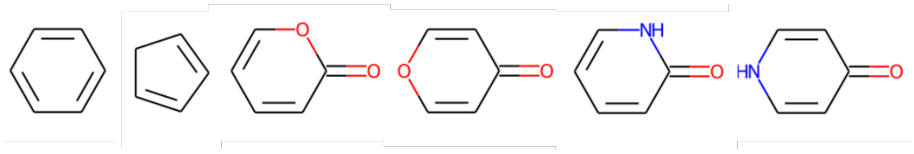



FIGURE 7.23: Ring building blocks used to create and modify the third set of 'reduced' population molecules: acenes with amide/ester-type rings.

Reduced populations with carbonyl, ester and amide-type building blocks tended towards a slightly higher average number of rings, likely due to the lower reorganisation energies found amongst five-ring systems composed with these ring

types; side groups were also more common amongst the final populations, with all average counts between 0.7 - 0.9 groups. Additional plots for these genetic algorithms can be found in the Appendix.

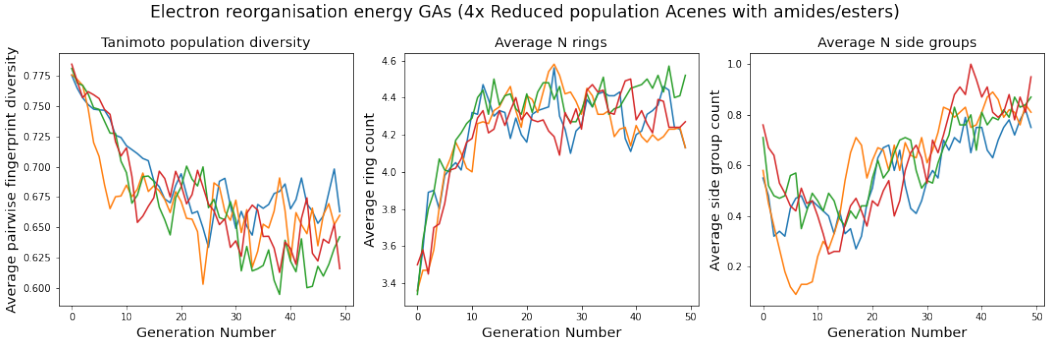



FIGURE 7.24: Plots to show diversity and composition for each population: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right)

### 7.3.4 Varied ring types

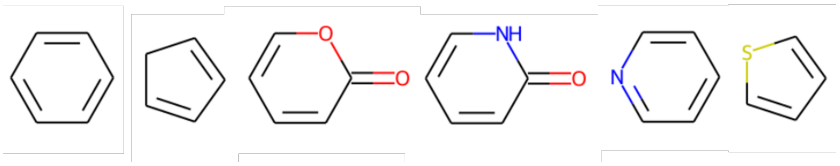



FIGURE 7.25: Ring building blocks used to create and modify the most complex set of 'reduced' population molecules, attempting to capture the variety of ring OCELOT fragments while working in a smaller space.

The most complex reduced building block set, as expected, sampled the highest number of molecules across the four campaigns reaching 6,766 unique species; as the closest analogue to the genetic algorithms with access to all building blocks, some more specific comparisons can be made for these runs.

Reorganisation energies of the top 100 species converged neatly around generation 20 to a tight range of values shown in Figure 7.27, compared to the spread of final values found with the full population runs in the corresponding plot of Figure 7.4.

The average ring count drops slightly in comparison to the full population runs, however the side group count still remains within a similar range to the other reduced building block GAs; this reinforces the idea that the chosen side group types are not able to reduce reorganisation energy values in the same manner as the full set.

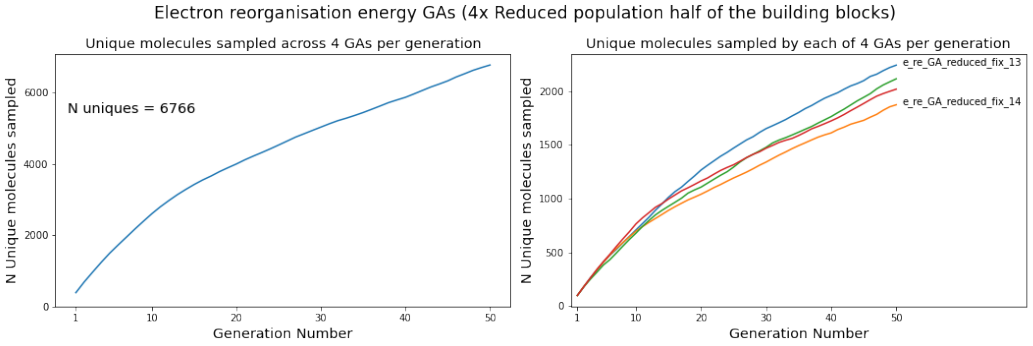



FIGURE 7.26: Plots to show how many unique molecules were sampled: by all 4 GAs using half of the original building block set (left); by each of the 4 GAs separately (right)

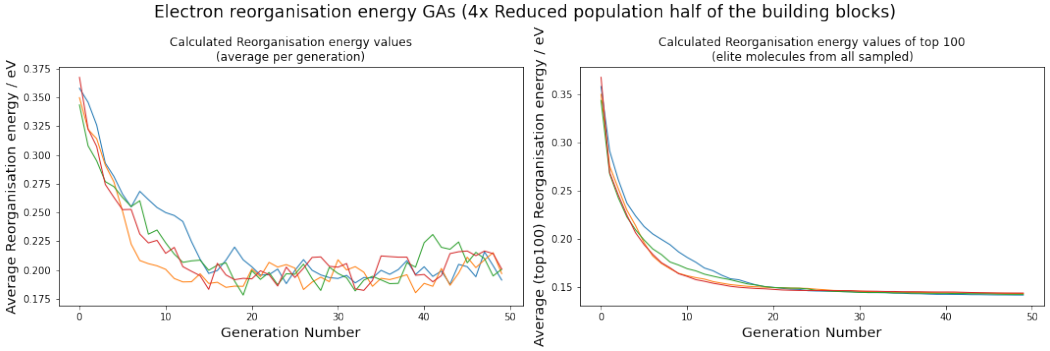



FIGURE 7.27: Plots to show how the fitness values (molecular electron reorganisation energy) change as each of these 4 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right)

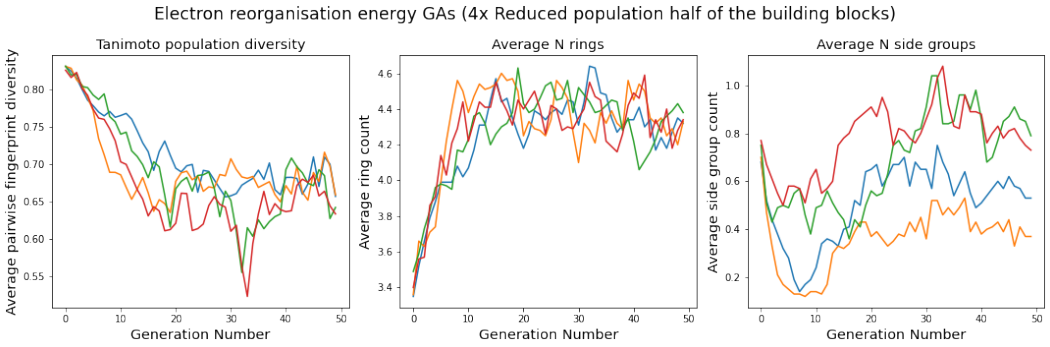



FIGURE 7.28: Plots to show diversity and composition for each population: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right)

## 7.4 Looking at all the GAs together

### 7.4.1 Frequently found molecules

While molecules that appear in numerous separate genetic algorithm runs aren't necessarily the top performers, looking at the subset of molecules with low reorganisation energy which are also frequently found can inform on the promising candidates that MolBuilder can easily access.

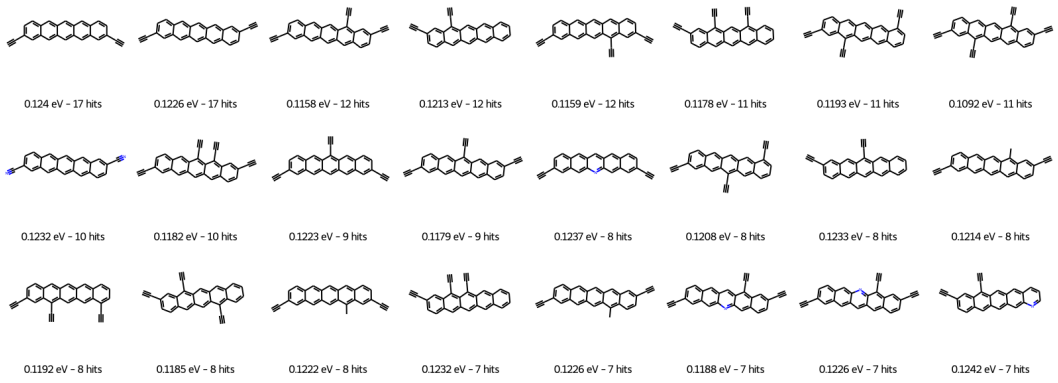



FIGURE 7.29: Grid showing 24 molecules commonly found in the 61 genetic algorithms run so far, where reorganisation energies are below 0.125 eV.

### 7.4.2 Distribution of reorganisation energies

In terms of reorganisation energy, genetic algorithms sampled molecules with a wide range of values; while the most concentrated range covers very low values (0.15-0.2 eV), some species were discovered with rather high values (up to 5.4 eV).

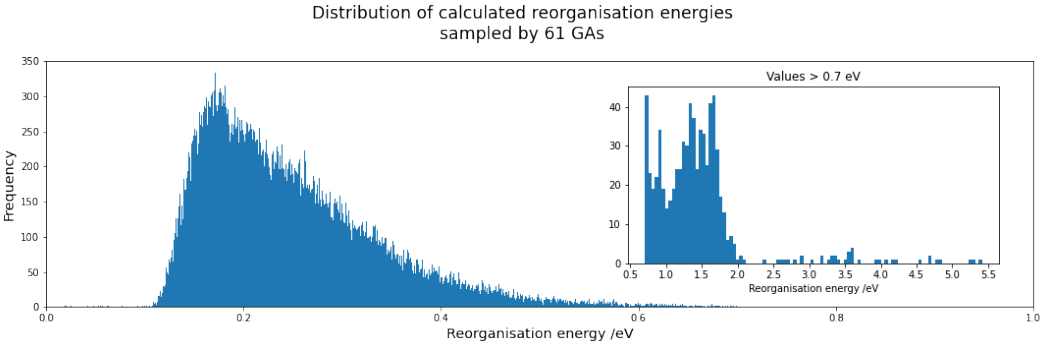



FIGURE 7.30: Plots to show the distribution of reorganisation energy values for all molecules sampled up to this point: majority of the distribution (main) and outliers with values higher than 0.7 eV (subplot).

#### 7.4.2.1 Comparing initial and final populations

The distributions of reorganisation energies can also be used to visualise how genetic algorithms are able to optimize this property, as shown in Figure 7.31. The initial populations exhibit a wide range of values up to 4.5 eV, which isn't unexpected since these populations were generated at random, without direct consideration of minimizing reorganisation energies. After 50 generations in a genetic algorithm run, the final populations fall into a smaller window up to 1.5 eV, with the majority of the distribution centred around lower values.

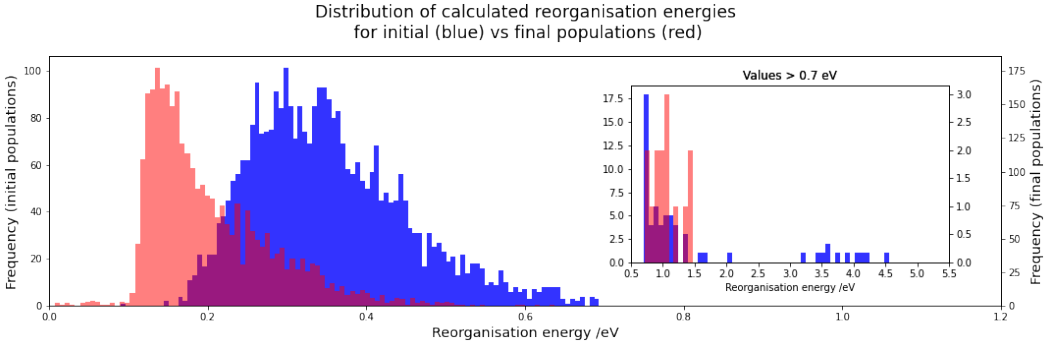



FIGURE 7.31: Plots to show the difference in distributions of reorganisation energies between initial populations (chosen at random, blue) and final populations (optimized via GAs, red).

#### 7.4.2.2 Comparison to known experimental molecules

Calculations on the experimentally known molecules, with the same parameters, gives a distribution centered around slightly higher values, covering a smaller range, as shown in Figure 7.32; by sampling molecules 'around' these experimental species, using extracted building blocks to run a genetic algorithm with MolBuilder, promising species towards the lower end of the reorganisation energy range found in the experimental set were generated.

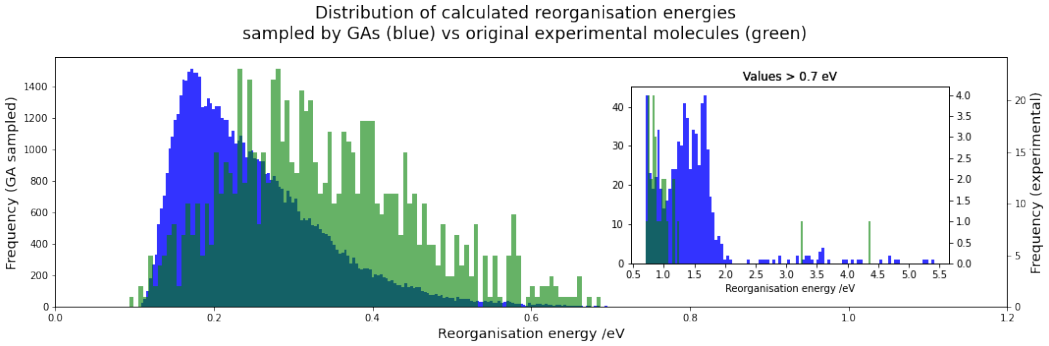



FIGURE 7.32: Plots to show the difference in distributions of reorganisation energies between GA sampled molecules (blue) and the original experimental set (green).

### 7.4.3 Common molecular shapes and substructures

Amongst the 25,736 molecules with low reorganisation energies (currently chosen as between 0.1 - 0.2 eV), a set of 18 'Murcko' molecular scaffolds can be extracted, each of which appears at least 50 times amongst these molecules with low reorganisation energies.

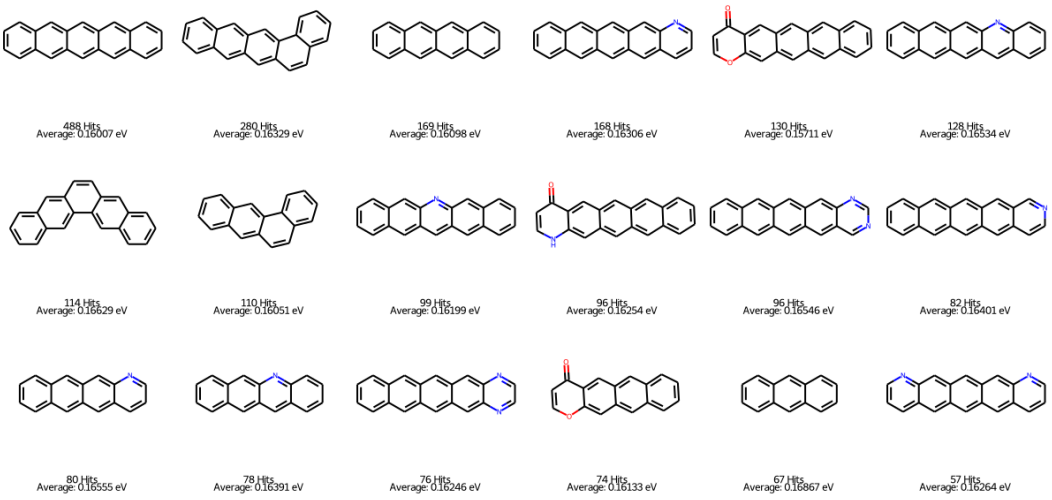



FIGURE 7.33: Drawings of the top 18 'Murcko scaffolds' appearing amongst sampled molecules with reorganisation energies between 0.1 and 0.2 eV. Below each image is a label with the number of appearances the scaffold makes, and the average reorganisation energy value found in molecules sampled with this scaffold.

Pentacene and pentacene-like molecular scaffolds dominate the top-performing portion of molecules sampled by these GAs; linear-pentacene and similar scaffolds make up more than half of these common scaffold types. Linear pentacene itself is the most common Murcko scaffold amongst top performers, with 488 unique substitution patterns appearing.

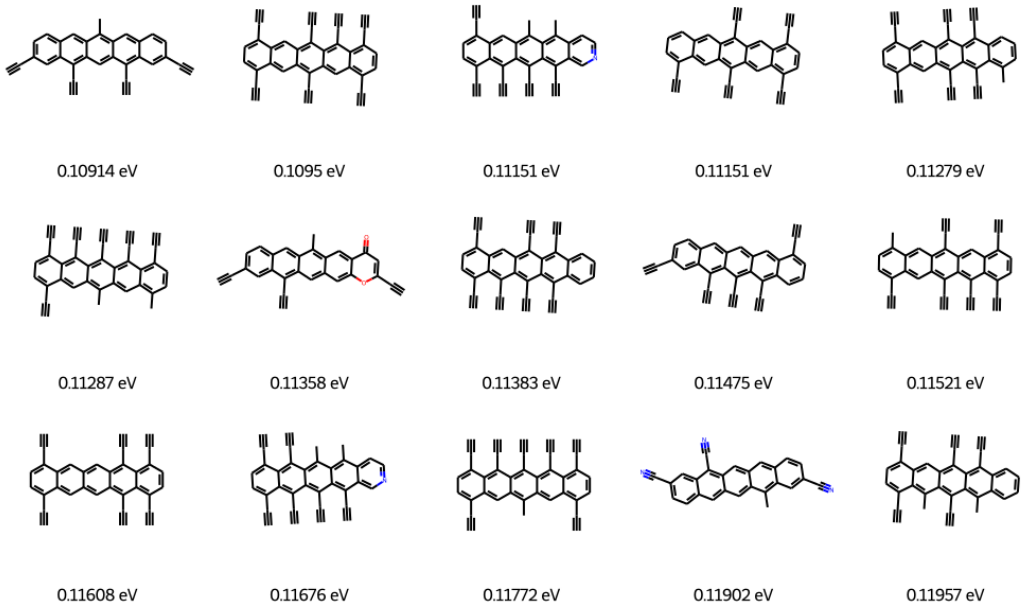



FIGURE 7.34: Drawings of 15 pentacene derivative molecules, ordered and labelled by reorganisation energy values.

Of particular note is the set of molecules highlighted in Figure 7.34. Firstly, it's clear that many of these molecules break the molecular generation rules put in place; there were issues in the code that meant the limits on number of side groups could be exceeded. The information is not without use, however, as high levels of ethyne side group additions at various positions on a pentacene core, seen in this set, leads to molecules with promising calculated reorganisation energies.

## 7.5 Electron mobility as a fitness function

As part of a collaboration within the Day Research group, MolBuilder was used to run genetic algorithms optimizing electron mobilities calculated from predicted crystal landscapes; this was a ambitious project, as CSP on this scale is computationally

expensive. Several members of the group contributed to this project, which required further development of the CSPy code base and the distribution of computational jobs between several users.

ARCHER2 supercomputing resources were used to run calculations, as part of an MCC Grand Challenge project; for the Grand Challenge, the MCC selects a small number of projects for which large allocations of ARCHER2 time are allocated, which fit well with this campaign of genetic algorithms where CSP had to be completed on-the-fly.

Electron mobility was chosen as a 'higher level' property to optimize than reorganisation energy; where reorganisation energies are calculated only accounting for the changes in molecular geometry, electron mobility calculations also consider the arrangement of molecular units in the solid state.

A set of 25 genetic algorithms were conducted with the calculation of electron mobilities through the use of predicted crystal structures. Molecules were again constructed with the full set of OCELOT building blocks, starting from the five initial populations for which repeats of reorganisation energy-led genetic algorithms had been run.

### 7.5.1 Sampling molecules with mobility genetic algorithms

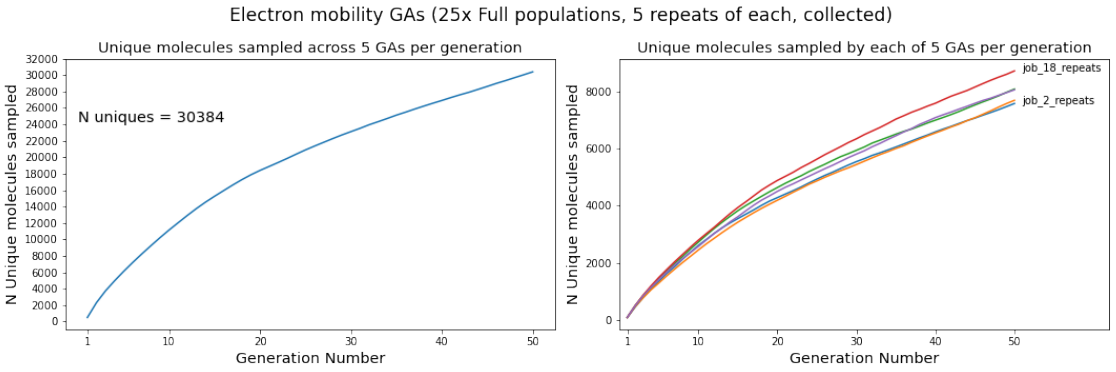



FIGURE 7.35: Plots to show how many unique molecules were sampled: by all 25 mobility-led GAs (left); by separated into groups of five by initial population number (right)

These genetic algorithms sampled fewer molecules than the reorganisation energy runs, with a maximum of 8,708 unique species discovered across five runs started from population 18; overall, these runs sampled roughly 7,000 fewer unique molecules than the 37,903 found in reorganisation energy led genetic algorithms.

### 7.5.2 Convergence of electron mobility values

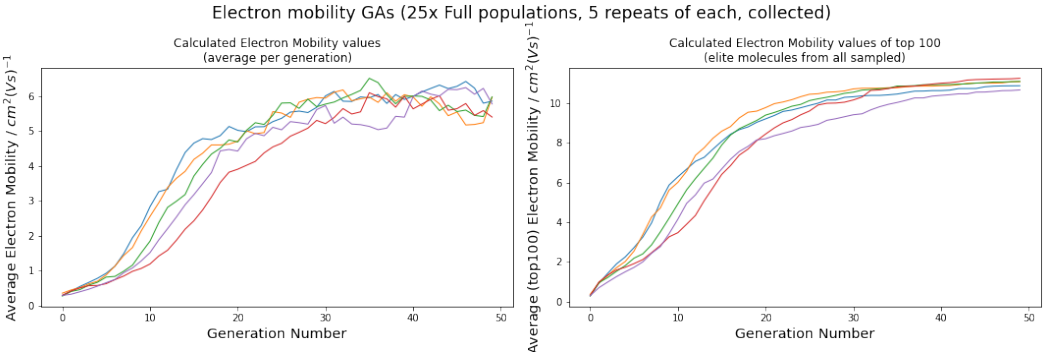



FIGURE 7.36: Plots to show how the fitness values change as each of the 25 mobility-led genetic algorithms progress, grouped by the initial population chosen: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right)

When optimized on electron mobility rather than reorganisation energy, genetic algorithms exhibit relatively quick convergence, where average values across each set of five runs increase rapidly at first, and start to level out after around 25 generations.

The initial populations were predicted to have relatively low electron mobilities, where molecules were not constructed with this property in mind; the genetic algorithm was able to quickly discover species with increased mobilities, approaching values of 6 $cm^2(Vs)^{-1}$.

### 7.5.3 Molecular composition and diversity of populations

Populations of molecules optimized with electron mobility in mind quickly approach the maximum count of five rings on average, and converge around an average of 4.4 rings by generation 20; conversely, the number of side groups attached to these molecules fall into a range of average values, sitting between 0.5 and 0.8.

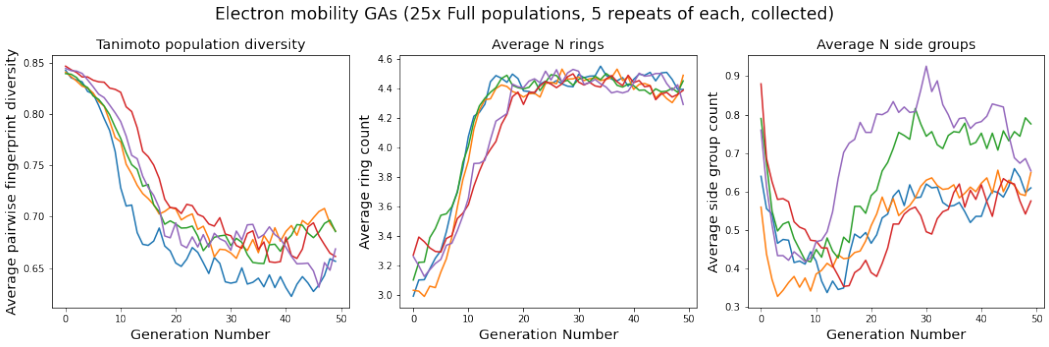



FIGURE 7.37: Plots to show diversity and composition for set of five populations: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right)

### 7.5.4 Overlap between separated populations

The overlap between each set of genetic algorithms per population reveals that by the end of generation 50, roughly 20% of each population is shared with genetic algorithms started from different initial populations.

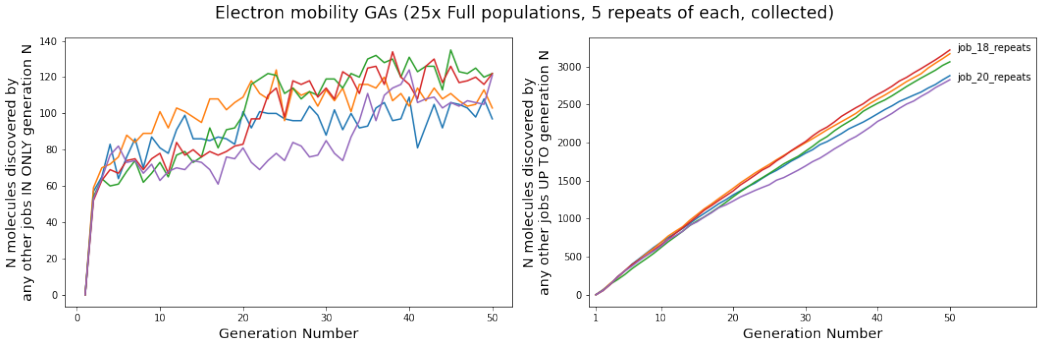



FIGURE 7.38: Overlap plots examining each set of 5 repeats for each initial population run in mobility-led genetic algorithms: per generation (left) and cumulative (right)

### 7.5.5 Comparing mobility to reorganisation energy

By comparing the values of each property with the scatter plot shown in Figure 7.39, the utility of more costly calculations to determine electron mobilities is clear.

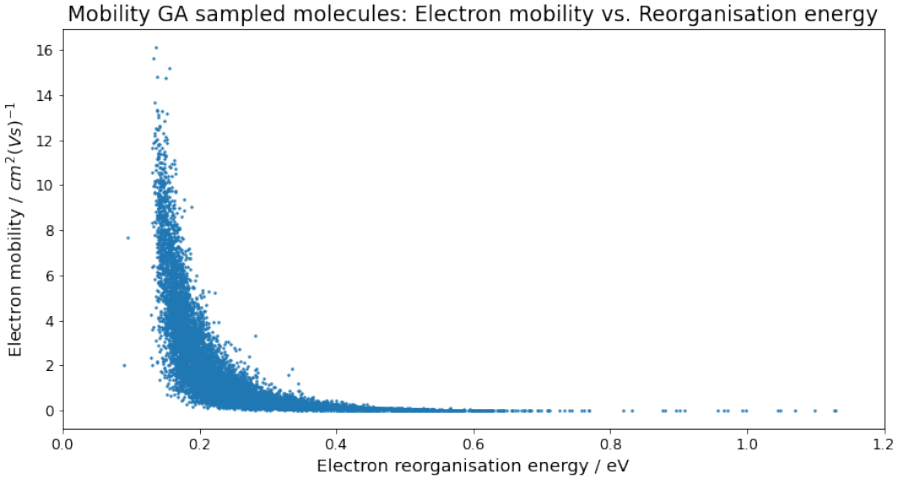



FIGURE 7.39: Scatter plot comparing calculated electron mobilities and electron reorganisation energies of molecules sampled through the 25 electron mobility-led genetic algorithms.

Molecules with higher reorganisation energy values are calculated to have near zero electron mobilities, as expected; however, those with low reorganisation energies achieve a wide range of calculated electron mobilities, as low as zero and reaching values as high as 16 $cm^2(Vs)^{-1}$.

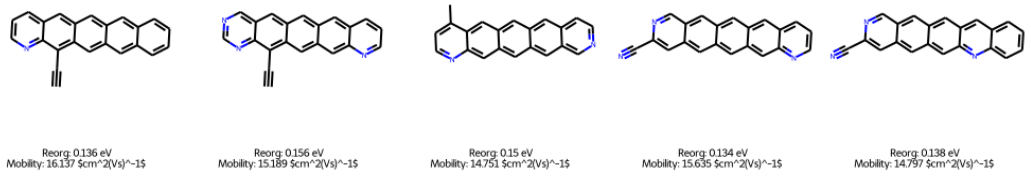



FIGURE 7.40: The top 5 sampled molecules according to electron mobility

Adding consideration of the solid-state packing arrangements is clearly important when determining the charge carrier mobility a given candidate molecule has; linear pentacene-type cores are present in each of the top-performing species shown in Figure 7.40, which exhibit low reorganisation energies but stand out from the rest according to calculated mobilities.

## 7.6 Conclusions after property-led genetic algorithms

This chapter has shown the efficacy of the MolBuilder genetic algorithm to conduct more directed exploration of chemical spaces, by considering reorganisation energy as a fitness function to minimize while iteratively generating populations of molecules.

Across 61 campaigns, these genetic algorithms were able to identify local reorganisation energy minima in the chemical space available with the OCELOT building blocks; this is indicated by plots such as Figure 7.4, where average fitness values level off as the number of generations increases. As the algorithms evolve, molecules within each population become more similar, indicating an exploitation of areas with good properties rather than continued broad exploration; average pairwise fingerprint similarities within each population drop, as shown in Figure 7.3,

Genetic algorithms which start from different initial populations share a portion of discoveries as they progress, likely hitting top performing molecules despite starting with unique populations. This behaviour is observed further in repeat runs from the same initial populations, where between 8% and 12% of the molecules sampled had also been identified by an analogous repeat run; each set of generations takes a different 'route', but a selection of molecules are observed multiple times.

Analysis of molecular composition reveals trends in the sampled species which correlate with lower reorganisation energy values; the movement of statistics such as ring counts and side groups shows what kind of values are preferred to optimize reorganisation energy as the genetic algorithms progress.

Later generations sample molecules with a higher number of rings and side groups, where average ring counts level out near the maximum intended value of 5, and the number of side groups increases, as shown in Figure 7.3. A notable exception to this is seen when the number of side group types is lowered for the 'reduced population' campaigns, where a fewer side groups tends to be preferred; this suggests that the chosen side groups do not favour lower reorganisation energies.

Commonly appearing building blocks and molecular shapes can be observed in the top performing sets, again suggesting the types of molecular composition which appear to promote reduced reorganisation energies. Pentacene-like 'backbones' are frequently found amongst molecules with low reorganisation energies, particularly those with multiple ethyne side groups attached; amongst the top 25,736 molecules with lower reorganisation energies, linear pentacene cores are observed most often, and several other pentacene-like analogues are prominent within this set.

Moving to the more expensive calculation of electron mobilities as a fitness function leads to a similar outcome, where linear pentacene backbones dominate the top-performing subset of candidates; analysis of the molecules sampled in these genetic algorithms shows that considering the solid state packing arrangement of these species is important when trying top optimize charge mobilities.

Overall, these genetic algorithms exhibit how directed chemical space exploration with MolBuilder can provide sets of molecules optimized for a calculable property of interest. Within 50 generations, each campaign was able to significantly lower average reorganisation energies, in comparison to the randomly generated initial populations; as shown in Figure 7.32, the distribution of reorganisation energies in molecules sampled by these genetic algorithms is more tightly centred on lower values when compared to the experimental set from which molecular building blocks were extracted.

# Chapter 8

# Post-hoc synthetic difficulty prediction

## 8.1 Azapentacenes - similar molecules

As described earlier, several approaches to synthetic difficulty prediction are utilised in this work. Applying each approach to the azapentacene chemical space provides insight into how they deal with a class of molecules which are relatively similar to one another. This section covers the analysis of the azapentacene space with each synthetic difficulty estimation method, and will determine how effective each method is in partitioning a large set of similar molecules into 'easy' and 'difficult' synthetic targets.

Methods which only consider the composition of a molecule to estimate synthetic difficulty, such as the fragment-complexity based approach taken by SYBA (Voršilák et al. (2020)) could struggle to differentiate between two given molecules in this set. Due to the limited number of building blocks, it is almost certain that they will share ring types, and the number of rings present will always be equal.

Conversely, methods which predict synthetic routes and base synthetic difficulty scores on them may have a better chance of partitioning this space, given the model has the right information. As an example within the azapentacene space, two molecules with the same set of ring types, but constructed differently, would likely exhibit a difference in estimated synthetic difficulty due to a different order of reactions, stereochemical requirements, etc.

If the azapentacene space can be partitioned according to synthetic difficulty, the clustering performed earlier will be useful in determining whether or not molecular shape and composition have an effect on predicted scores.

### 8.1.1 Distribution of synthetic difficulty scores

Analysis here starts with the distributions of predicted scores; since the full set of 135,744 azapentacene molecules available using the selected molecular building blocks has been generated, smooth distributions of scores are expected.

The highest and lowest scoring portions will be analysed in detail, in particular cases where a molecule scores well on one metric but poorly on another; this aims to give insight into how the methods differ, and what sort of compositions cause them to disagree.

The top performing set according to both metrics will then be further examined using AiZynthFinder, in an effort to determine what kind of synthetic pathways it suggests as well as to see if the retrosynthetic analysis based scoring agrees with molecular complexity and reaction network scoring approaches.

#### 8.1.1.1 SYBA

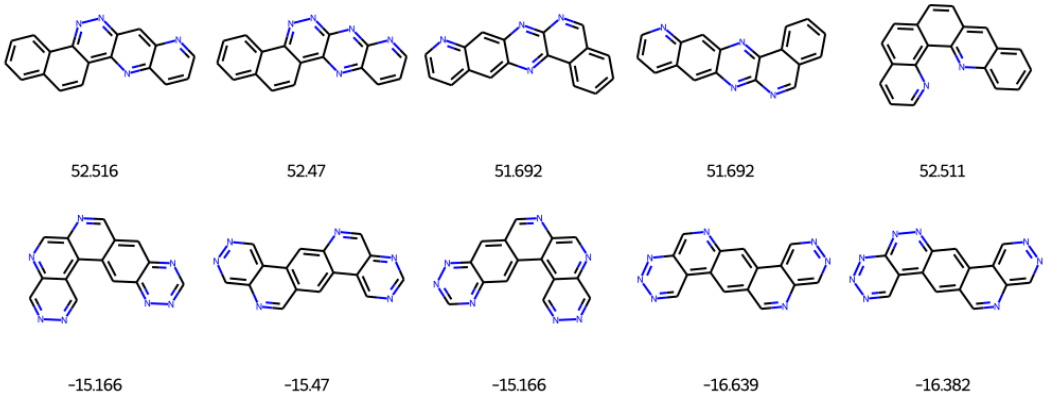



FIGURE 8.1: The best (top) and worst (bottom) scoring azapentacene molecules according to SYBA

Starting with the molecular complexity-based scoring approach, the azapentacenes give a fairly smooth distribution of scores, with the majority being marked as synthetically accessible according to the suggested threshold of 'above zero'.

7,965 molecules fell below said threshold, approximately 6% of the total space sampled. Examples of the best and worst molecules according to SYBA scores are shown in Figure 8.1; due to the similarity between these sets, it's difficult to draw conclusions from visual inspection.

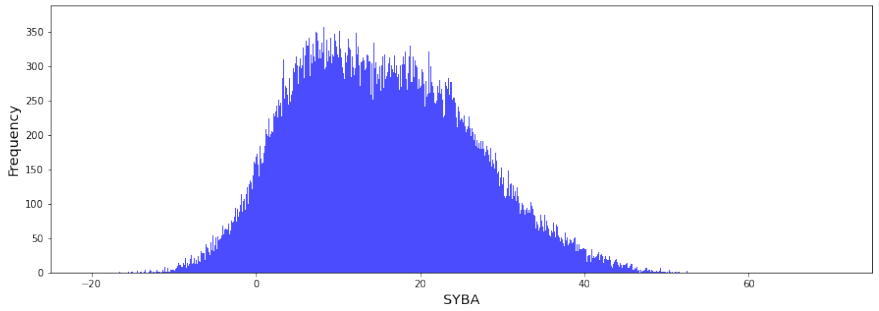



FIGURE 8.2: Distribution of calculated SYBA scores for all molecules sampled in an exhaustive exploration of the azapentacenes; anything scoring below 0 is suggested to be synthetically difficult.

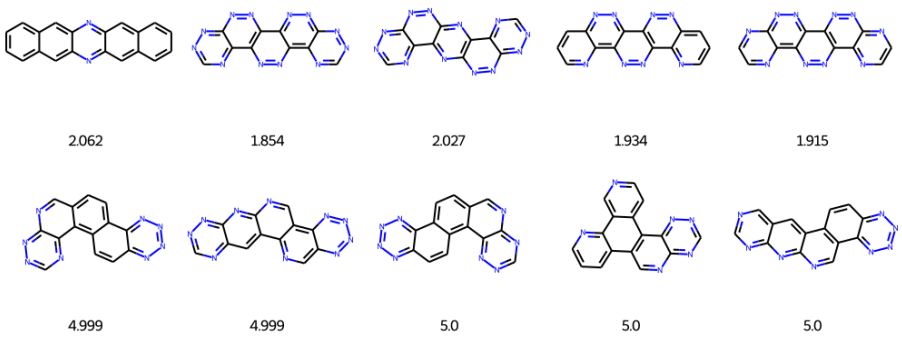



FIGURE 8.3: The best (top) and worst (bottom) scoring azapentacene molecules according to SCScore

#### 8.1.1.2 SCScore

The reaction-network based scoring also exhibits a fairly normal distribution of scores, which cut-off at a value of 5 (the maximum value of SCScores) for those predicted to be synthetically complex; between the top and worst scoring molecules from this analysis, it can be noted that there appears to be higher symmetry amongst molecules with lower SCScores.

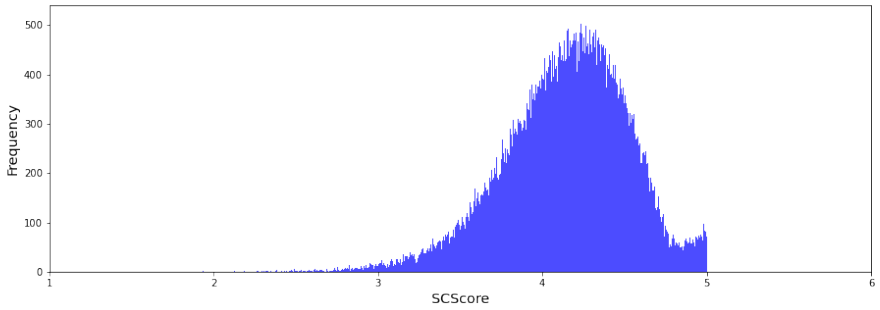



FIGURE 8.4: Distribution of calculated SCScore scores for all molecules sampled in an exhaustive exploration of the azapentacenes; higher scoring molecules are suggested to be synthetically difficult.

#### 8.1.1.3 Analysing clusters

As described in previous sections, clustering the azapentacene space by the shape of the molecular backbone and nitrogen count can be useful to partition this set of similar molecules, which helps to determine possible relationships between these factors and predicted synthetic difficulty scores.

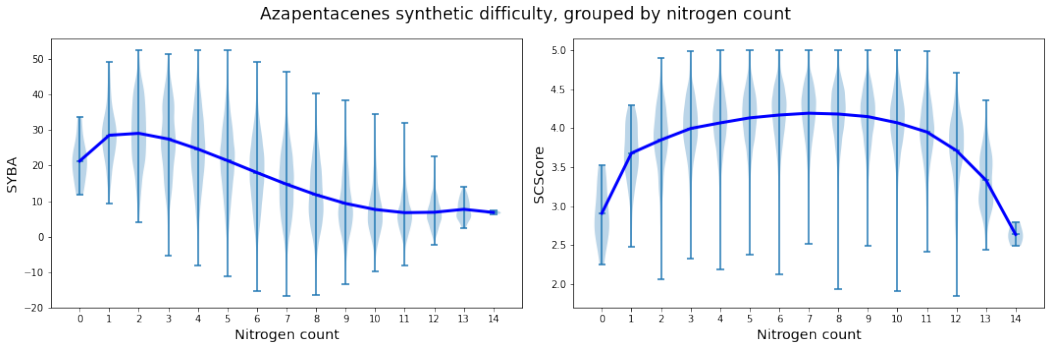



FIGURE 8.5: Violin plots describing synthetic difficulty scores for the azapentacenes, grouped by nitrogen count.

Clustering by nitrogen count shows a variety of scores within each level of nitrogen content; the ranges of these scores become wider as the number of possible substitution patterns increases. There are two trends in the average score, depending on the synthetic difficulty score used. For SYBA scores, higher levels of nitrogen content generally reduce the predicted scores, implying higher molecular complexity; on average, SCScore tends to increase when nitrogen counts are between 1 and 10, implying higher synthetic complexity for species where there is a mixture of carbon and nitrogen on the outer edges of azapentacene structures. Clustering by the generic backbone type shows relatively consistent scores for each group when using SYBA, with linear pentacene structures scoring the best on average; the same is true for SCScore results, where linear pentacene stands out from the other backbones with a lower score on average.

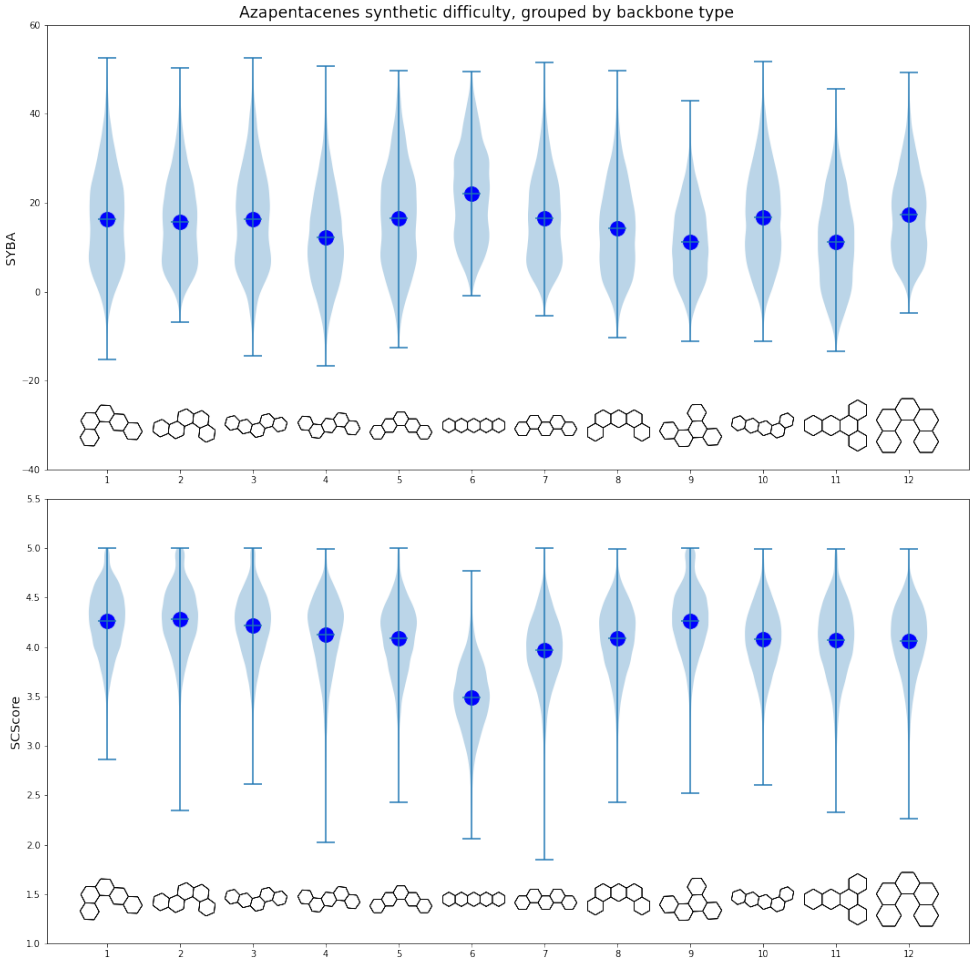



FIGURE 8.6: Violin plots describing synthetic difficulty scores for the azapentacenes, grouped by generic backbone type.

#### 8.1.1.4 Comparing SYBA and SCScore

The two different scores can be scattered against each other to determine how much they 'agree' with each other for each molecule in the azapentacene set.

Figure 8.7 shows a rough agreement between SYBA and SCScore, where in general a higher SCScore (indicating synthetic complexity) is matched by a lower SYBA score (indicating molecular complexity). Molecules found above the 'accessible' threshold of $SYBA > 0$ make up the majority of cases where SCScore falls below 3; when SCScore rises above 3, SYBA scores fall into a larger range, indicating that such molecules appear more like reaction products during SCScore analysis, but are seen by SYBA at varying levels of *molecular complexity*.

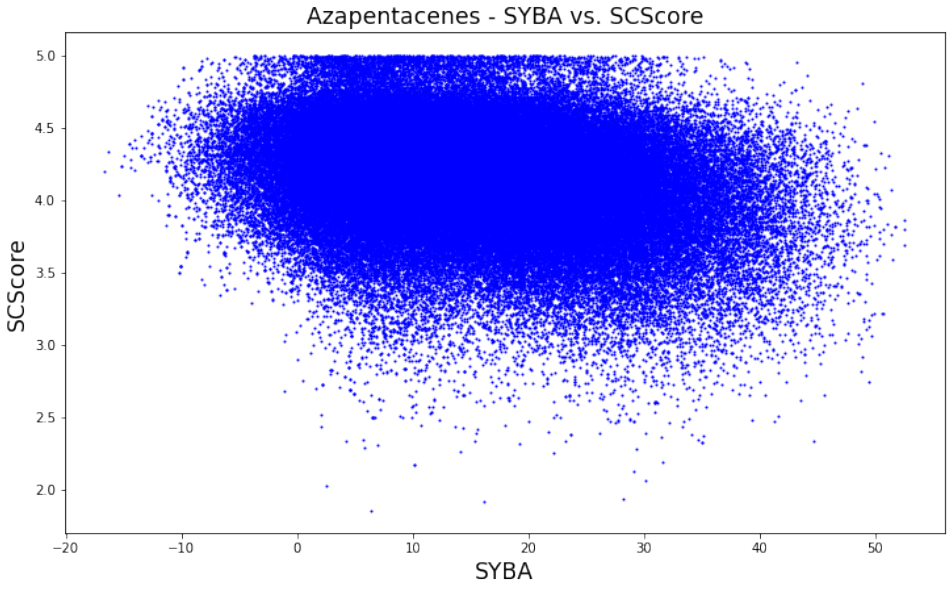



FIGURE 8.7: Scatter plots comparing SYBA and SCScore synthetic difficulty scores.

### 8.1.2 Retrosynthetic analysis with AiZynthFinder

Top performing azapentacene molecules according to both SYBA and SCScore have been extracted and passed through AiZynthFinder, to demonstrate the prediction of retrosynthetic pathways done using this tool to determine the kind of reagents and reaction steps that may be needed to produce these molecules in-lab.

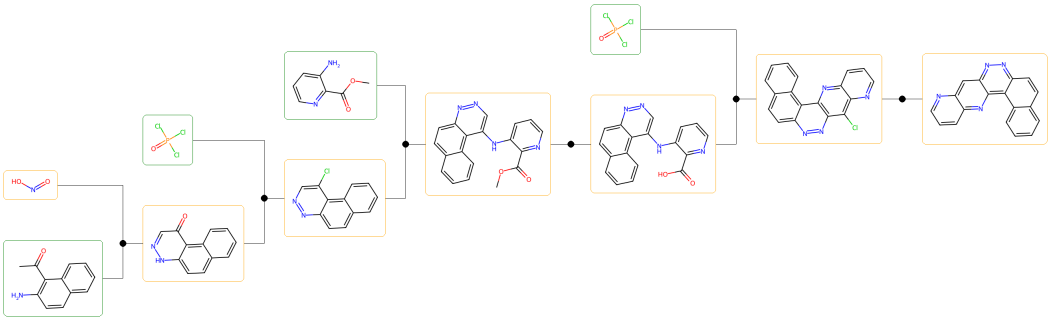


FIGURE 8.8: Predicted synthetic route to an example azapentacene molecule, produced by AiZynthFinder; green borders mean that the molecule is available as a reagent in the ZINC15 database

### 8.1.3 Experimentally known molecules

Given the large number of molecules, and the simplicity of the azapentacene chemical composition, already discovered and synthesised species should be present. By examining this subset of the space, we can start to determine which generated molecules match up with 'real life' results.

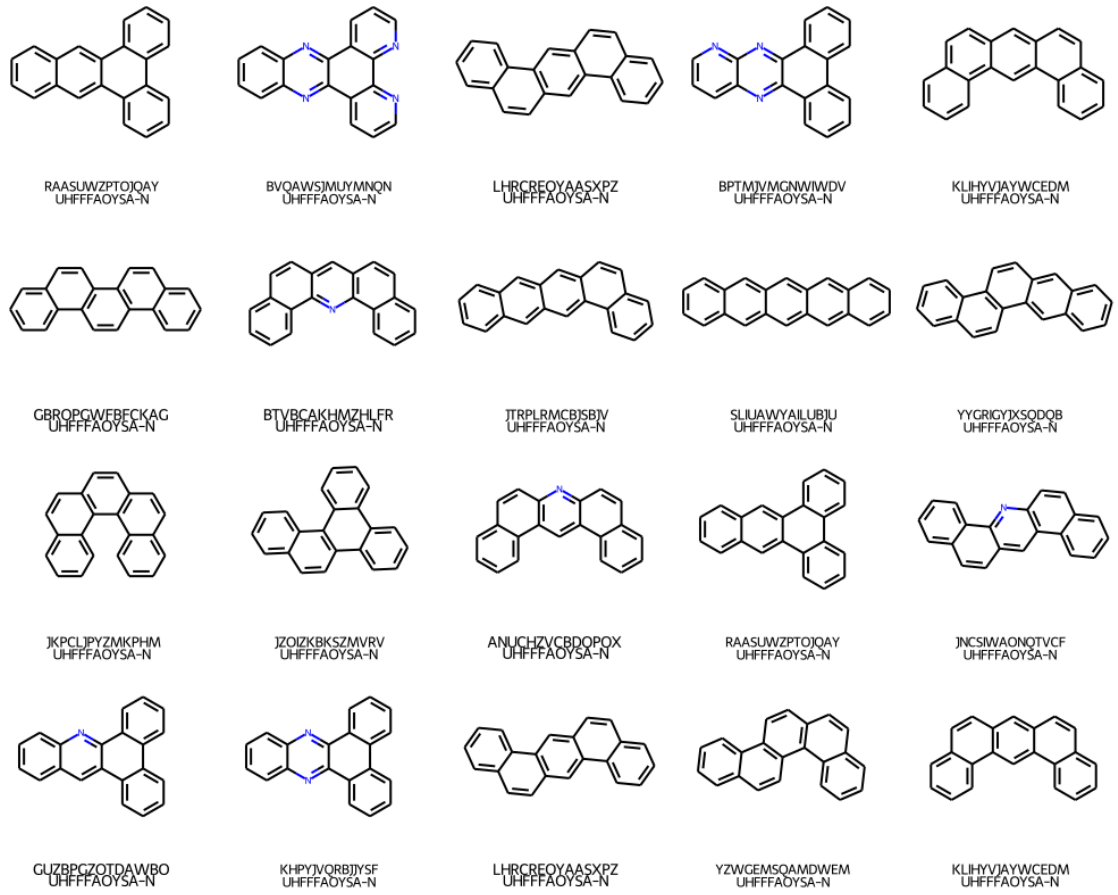



FIGURE 8.9: 5 ChEMBL matches (top) and 15 PubChem matches (bottom)

Only a small number of the sampled molecules were present in the databases searched, including the well-known linear pentacene and multiple other arrangements of pentacene with no nitrogen substitution.

## 8.2 OCELOT - reorganisation energy GA results

As a step up from the azapentacenes, different behaviour is expected from the synthetic difficulty estimation tools when analysing molecules generated from the OCELOT building blocks; there are multiple ring types, including several heteroatom options, and the inclusion of side groups, an option not present when generating azapentacene molecules.

### 8.2.1 SYBA score distributions

Another relatively even distribution is exhibited when looking at the molecules sampled up to this point with reorganisation energy-led genetic algorithms; of the 71,121 molecules generated, only 2,988 fall below the 'synthetically accessible' threshold.

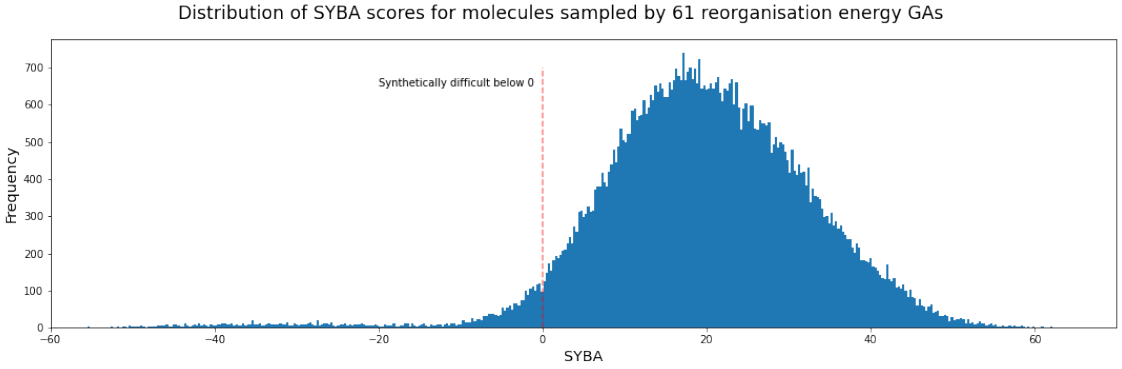



FIGURE 8.10: Distribution of calculated SYBA scores for all molecules sampled with reorganisation energy genetic algorithms; anything scoring below 0 is suggested to be synthetically difficult.

Compared to the azapentacenes, molecules constructed using the OCELOT building blocks were able to score far lower than those sampled in the azapentacene space, with minimum scores of -77 and -16 respectively. This is likely due to some of the OCELOT building blocks being marked as 'complex fragments' in the SYBA scoring scheme.

The peak of the SYBA score distribution has been shifted to a higher value, indicating less molecular complexity amongst these molecules; this is likely a result of the genetic

algorithms being able to access different molecule sizes, as smaller molecules have less rings, and so have fewer fragments to contribute complexity when scoring.

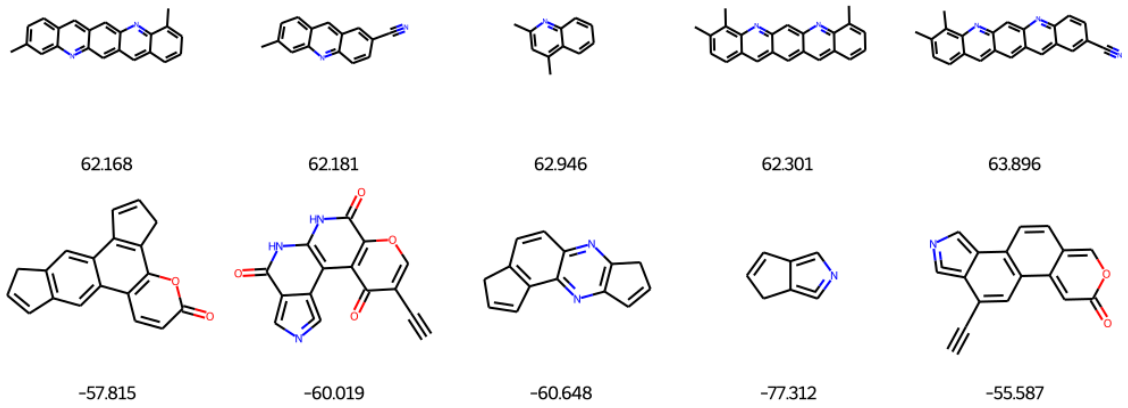



FIGURE 8.11: The best (top) and worst (bottom) scoring reorganisation energy GA sampled molecules according to SYBA

The difference between top and bottom-scoring molecules is much clearer when the set of building blocks is more varied, as shown in Figure 8.11. Benzene and pyridine rings are favoured in the top-5 molecules, with attachments such as methyl and cyano groups. Conversely, the worst 5 sampled molecules are visually more complex, with the presence of amide/ester type rings and frequent appearance of the cyclopentadiene ring (and its aza-substituted equivalent).

The lowest scoring molecule, simply two 5-membered rings attached 'back-to-back' is curious; despite the small molecular size, and lack of 'visual complexity', SYBA has labelled it as incredibly complex, suggesting that such 2-ring fragments have been considered part of the 'complex' dataset used to mark molecules as hard-to-synthesise.

### 8.2.2 SCScore distributions

The distribution of SCScores exhibits a similar shape to the azapentacenes with slightly more spread and a shifted peak towards lower values, indicating that molecules sampled with the genetic algorithms using OCELOT building blocks are generally more synthetically accessible.

This result isn't surprising; once again, with the wider range of molecule sizes, and freedom to pick from a larger set of building blocks, the genetic algorithms are able to sample 'simpler' molecules.

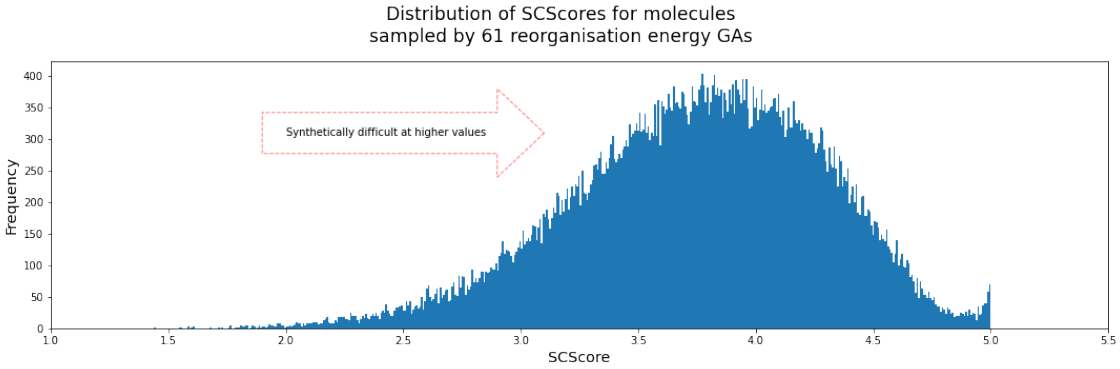



FIGURE 8.12: Distribution of calculated SCScores for all molecules sampled with reorganisation energy genetic algorithms; higher scoring molecules are suggested to be synthetically difficult.

With SCScore in particular, the molecule size (or number of rings) becomes more important. As the scores are based on similarity to either reagents or products in a reaction network, it is almost expected that smaller molecules will score well; common molecules with small numbers of rings are more likely to be prevalent as reagents, rather than reaction targets.

The 'tail' at the high-scoring portion of this distribution appears again; there are likely molecules in this set which are 'too complex' to be considered as reagents of a reaction in any way, or are perhaps similar to natural products in some way.

Compared to both the azapentacene results and the SYBA results for OCELOT, the top and bottom scoring molecules according to SCScore exhibit a stark difference, which sheds more light on how the scores are being generated.

Amongst the top scoring molecules shown in Figure 8.13, SCScore clearly favours smaller molecules, including some with chloride functional groups, which is present in reactants frequently used in the Reaxys database of reactions.

The worst-scoring 5 molecules are visually complex, with several heterocycle types, and attached side groups. As a note, they all received the maximum score available with SCScore, 5.

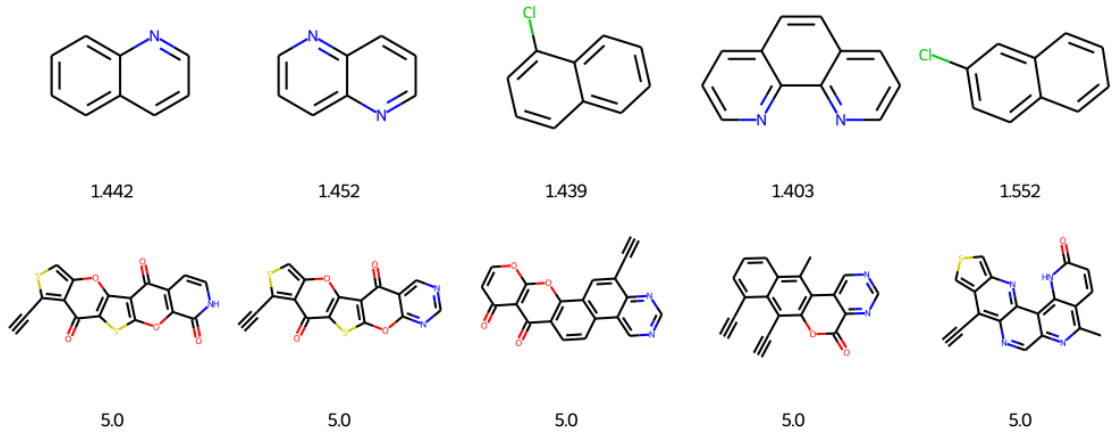



FIGURE 8.13: The best (top) and worst (bottom) scoring reorganisation energy GA sampled molecules according to SCScore

### 8.2.3 Restriction on building blocks to lower synthetic difficulty

One of the ideas that prompted using smaller building block sets was to reduce the synthetic difficulty of generated molecules by reducing the number of ring types available.

Other than a subtle shift in centring of SYBA scores to lower values, this had little effect on the distributions of calculated synthetic difficulty values for molecules sampled by reorganisation energy genetic algorithms, as shown in Figure 8.14 Distributions of these scores partitioned according to the type of reduced population are available in the appendix Figure A.47.

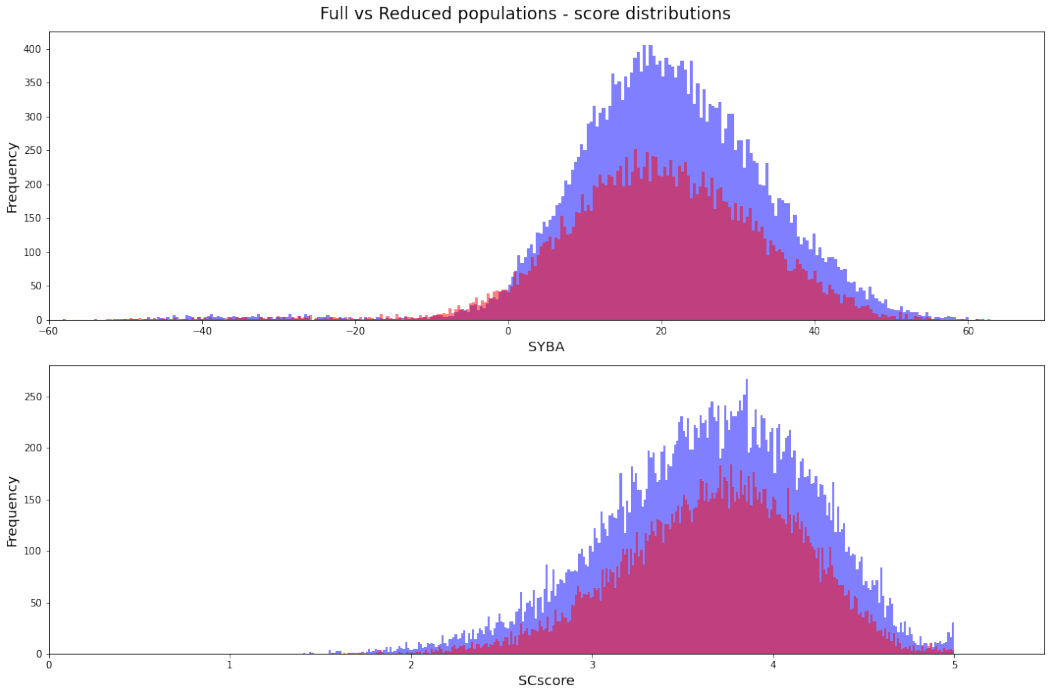



FIGURE 8.14: Distribution of calculated SYBA/SCScores for molecules sampled by reorganisation energy genetic algorithms, partitioned into those with all building blocks (blue) and those without (red).

### 8.2.4 How do SYBA and SCScore match up?

Inclusion of additional building blocks compared to the azapentacenes work results in different behaviour between the synthetic difficulty scoring functions, and highlights cases where SCScore and SYBA do not agree with one another; the scatter plot of Figure 8.15 shows that molecules marked by SYBA as complex receive a range of SCScores across all potential values.

A selection of molecules which produce the biggest 'disagreements' between SCScore and SYBA are shown in Figure 8.16, which SYBA marked as highly complex, while SCScore marked them as synthetically 'easier'.

This highlights the possible reasoning behind this mismatch, as all of these species contain at least one five-membered ring, and only two rings overall; SYBA is likely marking five-membered rings as more complex when attached to other ring systems, whereas SCScore is able to see such molecules used as reagents more often than appearing as products in the Reaxys database of reactions.

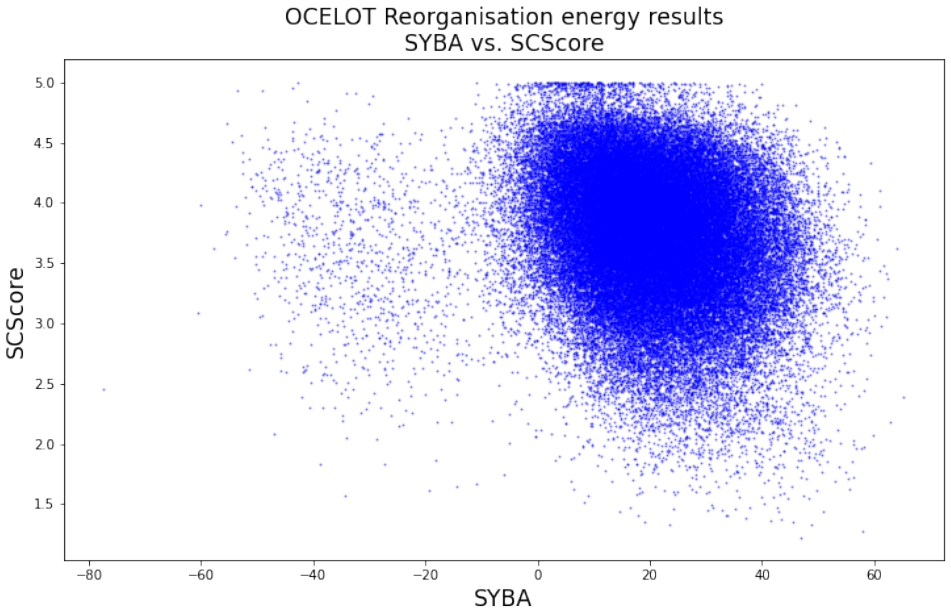



FIGURE 8.15: Scatter plot of SYBA and SCScore results for all molecules sampled with reorganisation energy genetic algorithms.

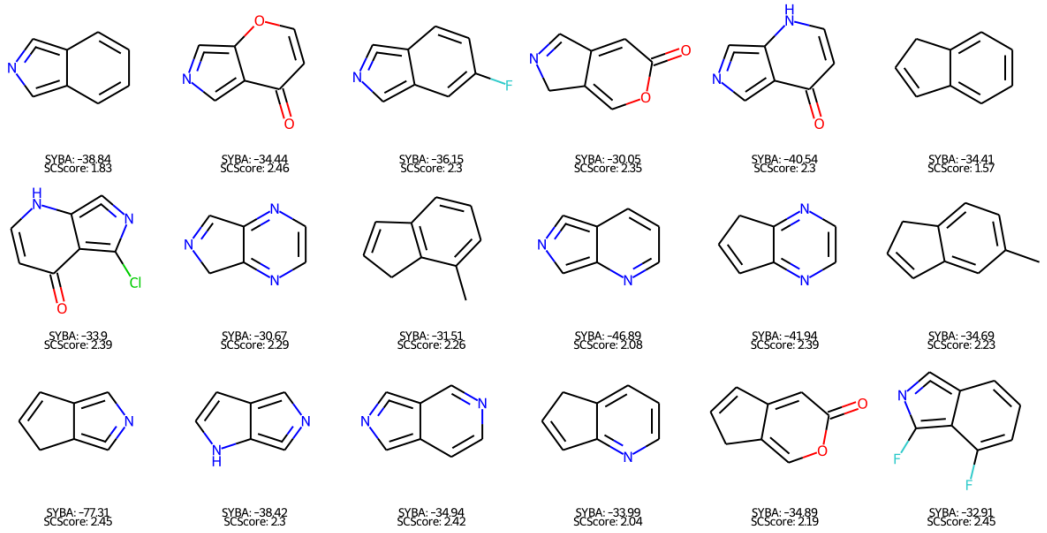



FIGURE 8.16: Grid of species discovered in reorganisation energy genetic algorithms, where post-hoc analysis with SYBA and SCScore results in a disagreement regarding their percieved synthetic difficulties.

### 8.2.5 Comparison to the experimental set

This comparison works to provide validation that generated scores make sense, and inform on how generated molecules compare to the original experimental species from which building blocks were extracted.

Experimentally known molecules should exhibit good synthetic accessibility scores, and fall into a better range of values than the species sampled with reorganisation energy genetic algorithms, where synthetic difficulty wasn't directly taken into account.

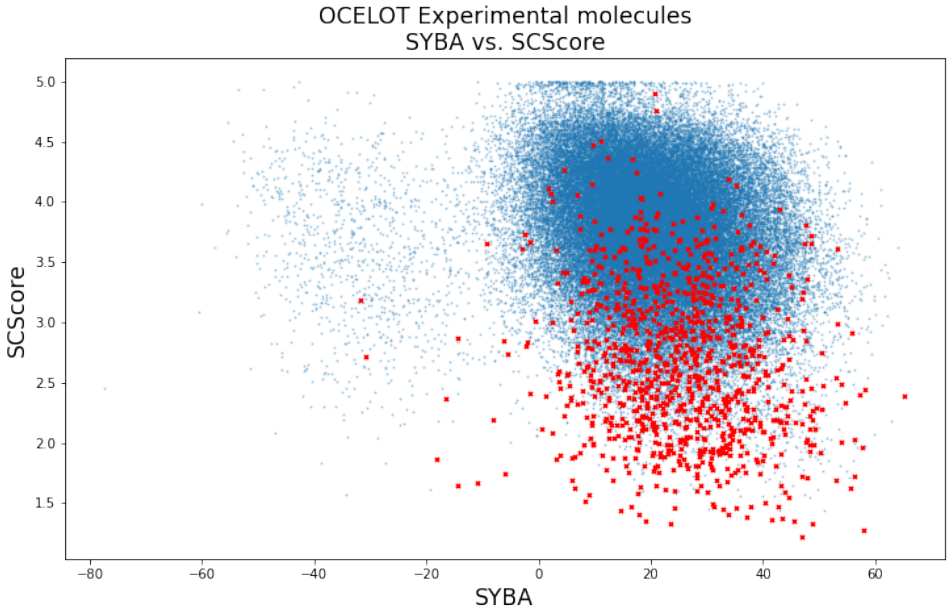



FIGURE 8.17: Scatter plot of SYBA and SCScore results for the original OCELOT molecules (red) used to select molecular building blocks, compared to those sampled through the genetic algorithms (blue).

The scatter plot shown in Figure 8.17 is promising as for the most part, experimental molecules scored above zero with SYBA, indicating lower molecular complexity; values predicted by SCScore fall into a wider range of synthetic complexities, but only a few approach the maximum SCScore of 5.

### 8.2.6 Retrosynthetic analysis with AiZynthFinder

Using AiZynthFinder, synthetic routes to sampled molecules can be predicted and scored according to reagent availability and the number of reaction steps; these scores are described with a violin plot in Figure A.46 according to different categories.

It's worth noting the 9 molecules in initial populations which scored close to zero through AiZynthFinder analysis; these molecules are shown in Figure 8.18. In these cases, where the score is 0.049 no disconnection steps could be found during the retrosynthetic analysis, and the target molecule was not available as a reagent in the ZINC15 database, hence the incredibly low score.

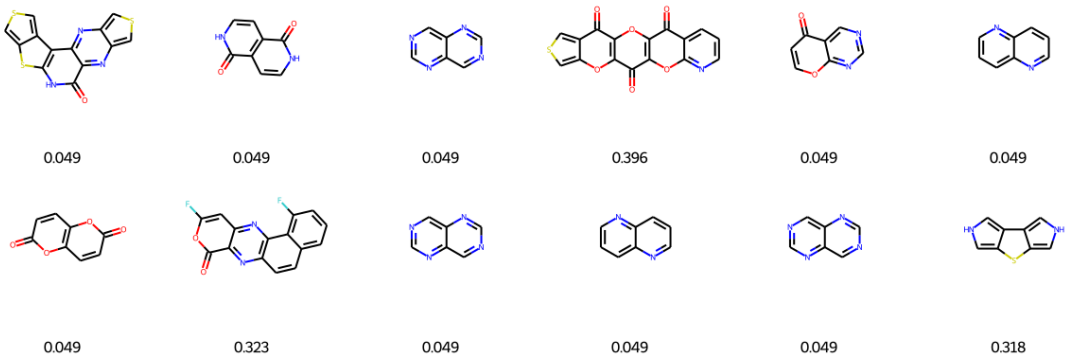



FIGURE 8.18: 12 molecules marked as highly synthetically difficult according to AiZynthFinder.

The initial populations generally exhibited higher scores before being passed through MolBuilder reorganisation energy genetic algorithms; the final populations see a drop in scores of roughly 0.2 on average.

The best target molecules, scoring close to 1 based on AiZynthFinder analysis, have very few steps, or are available directly as precursors in the ZINC15 database; such examples are shown in Figure 8.19.

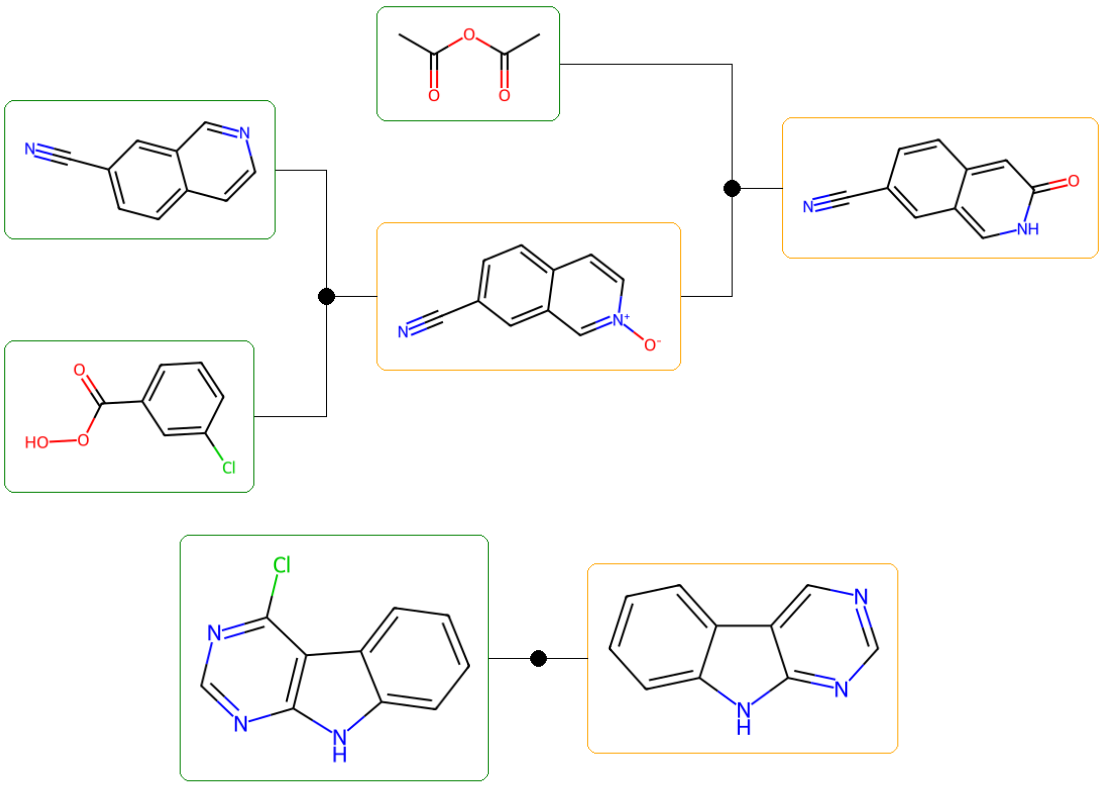



FIGURE 8.19: Examples predicted synthetic routes to targets from the OCELOT space with very high AiZynthFinder scores

### 8.2.7 Accessible molecules with low reorganisation energies

In order to determine the 'best of the best' from molecules sampled so far with reorganisation energy genetic algorithms, which exhibit both promising physical properties and low estimated synthetic difficulty, a subset of 12 species was selected with the following criteria:

- Calculated reorganisation energy below 0.135 eV
- Predicted SYBA score above 30
- Predicted SCScore below 3

The 12 molecules which fit these criteria all share the linear pentacene-type structure, with various levels of nitrogen substitution and side groups attached; these are shown in Figure 8.20.

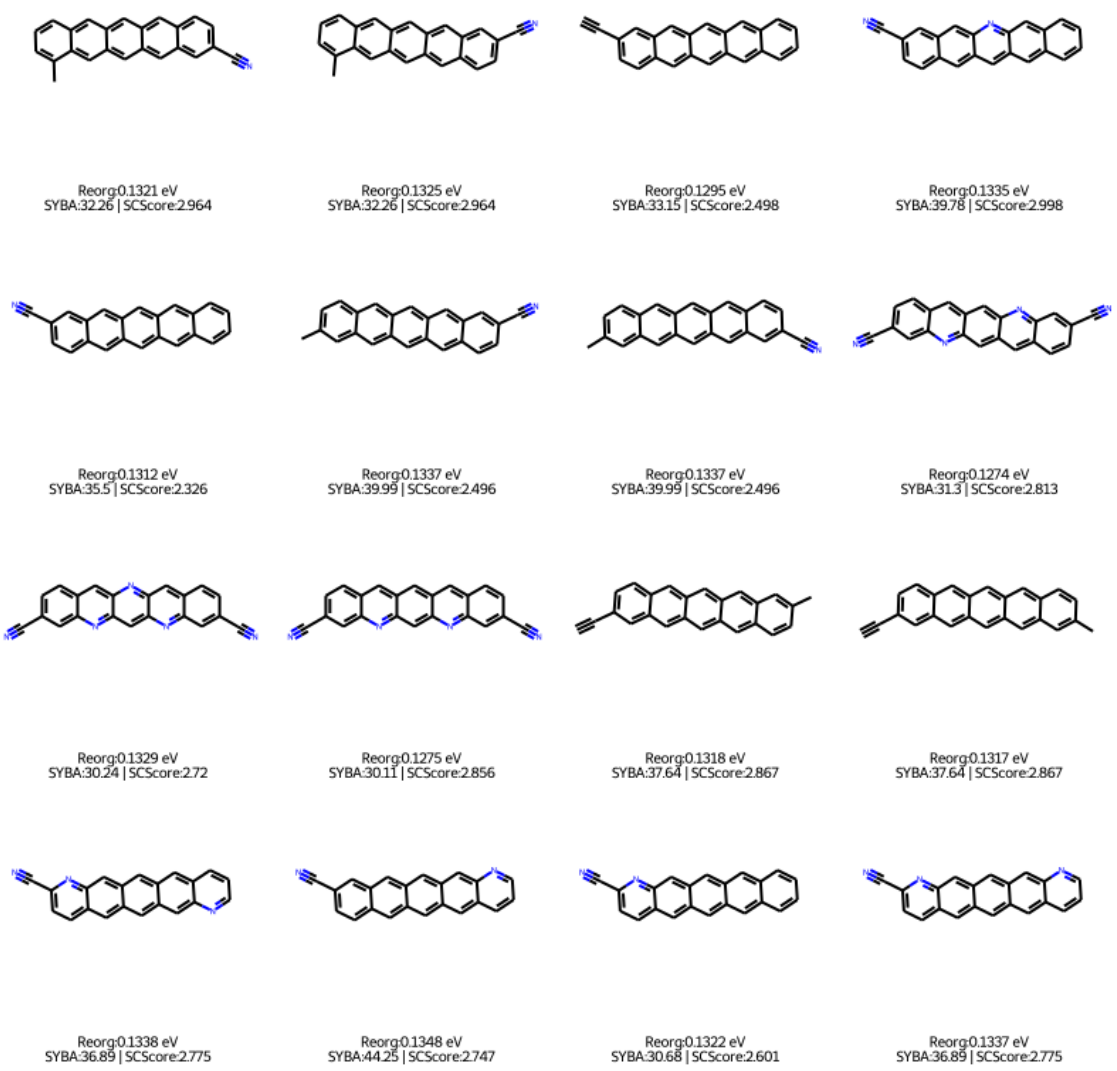



FIGURE 8.20: 12 molecules which exhibit both optimal reorganisation energies and synthetic difficulty scores according to SYBA and SCScore.

This is a reassuring outcome, since as shown in the previous chapter pentacene-like cores appear to promote lower reorganisation energies; this set of top performers can also be identified as top performing molecules in synthetic difficulty scoring through SYBA and SCScore, meaning there is at least some portion of the OCELOT chemical space in which both reorganisation energy and synthetic difficulty can be minimized in tandem.

## 8.3 Conclusions from standalone synthetic difficulty analysis

This chapter served to validate the use of SYBA and SCScore as measures of synthetic difficulty, through analysis of past results and comparisons to scores for experimentally known molecules, before their implementation into MolBuilder for use as fitness functions to optimize in genetic algorithms.

These scores fall into smooth distributions for both the azapentacenes and the GA-sampled OCELOT species; SCScores exhibit a 'tail' at the maximum allowed value of 5, which was noted by Coley et al. (2018) as an indication that molecules are 'substantially more complex' than the species found in Reaxys used to train the model.

Comparison between SYBA and SCScore values shows a general agreement between the metrics, and highlights cases where disagreements are found; the grid of molecules in Figure 12.2 describes such cases, in which small molecules which may frequently appear as precursors (leading to low SCScore values) are considered to be higher in molecular complexity according to SYBA, perhaps due to the combinations of six- and five-membered rings.

The experimentally known OCELOT molecules used to select building blocks for use in the genetic algorithms of this work generally score well in SYBA analysis, falling above the molecular complexity threshold of 0; SCScores are predicted across the possible range of values, which indicates a range of perceived synthetic complexity, however only a few molecules approach the maximum score of 5, and the majority score below 3.5, suggesting that while these experimental species may be more complex to synthesise, they're not impossible to attain.

These outcomes inspire confidence in the use of these synthetic difficulty tools as properties to be optimized in genetic algorithms. SYBA and SCScore tend to agree with one another; in cases where the scores disagree, the reasoning behind such disagreements is clear. The different approaches appear to be capturing different aspects of synthesis, and the scores calculated for experimentally validated species tend to fall within acceptable bounds.

# Chapter 9

# Synthetic difficulty GAs

The next set of genetic algorithms run with MolBuilder used only the OCELOT building blocks, and aim to optimize the synthetic difficulty scoring methods SCScore and SYBA; allowing GAs to run with these as fitness functions provides more information on the type of molecules that are preferred by each approach, and should sample areas of the available chemical space with species easier to produce in laboratories.

20 genetic algorithms were run with the full OCELOT building block set, using the same initial populations as the 20 reorganisation energy runs; these will be analysed in the following sections. An additional four genetic algorithms were run with each of the four reduced building block sets.

## 9.1 Behaviour of SCScore-led genetic algorithms

Genetic algorithms tasked with optimizing SCScore to reduce synthetic complexity show interesting behaviour; the calculated scores drop rapidly for the first few generations, moving away from the initial populations which had not been built with synthetic difficulty directly considered, but consistently rise after around generation 10, reaching average values higher than those found earlier on by the end of the run.

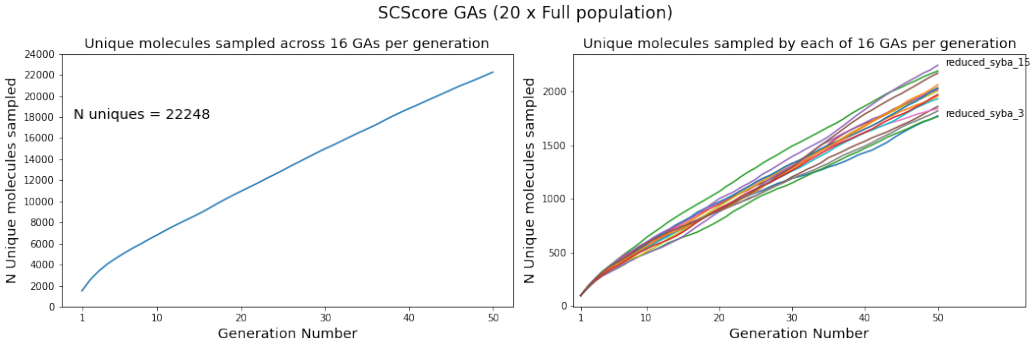



FIGURE 9.1: Plots to show how many unique molecules were sampled: by all 20 SCScore-led GAs (left); by each of the 20 GAs separately (right)

This behaviour may be due to the nature of the genetic operators, as when creating a new generation with parents from the previous population, there's a chance that children can be created with a higher number of rings than their parents; adding more rings to a molecule will likely raise its SCScore, as larger molecules are more likely to be present as reaction products as opposed to reagents.

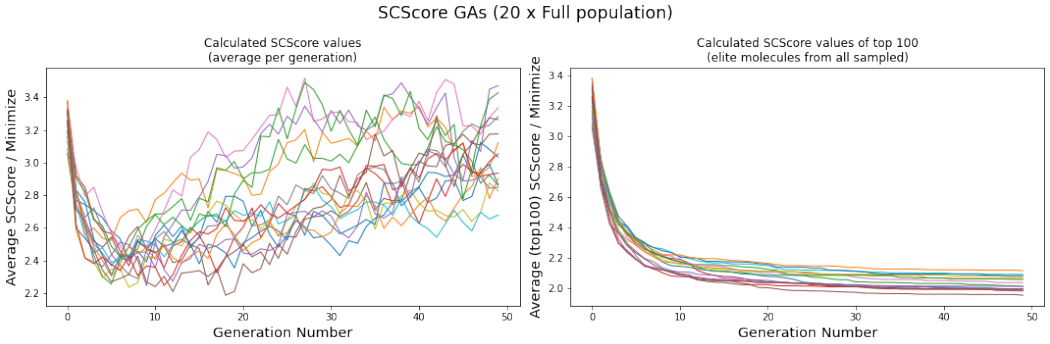



FIGURE 9.2: Plots to show how the fitness values (SCScore) change as each of the 20 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right)

The right portion of Figure 9.2 is reassuring, showing that top-performing molecules get sampled early in each genetic algorithm run, achieving low scores as soon as generation 20; unlike the left potion, which shows an increase in scores at later generations, the top 100 average scores continue to decrease as the algorithms progress.

### 9.1.1 Changes to diversity and molecular composition

The central plot of Figure 9.3 confirms the idea suggested above; at first, the average ring count per population drops to between two and three, achieving low calculated SCScores; after generation 10, the average counts rise again, leading to values between 3.5 and 4.5 rings.

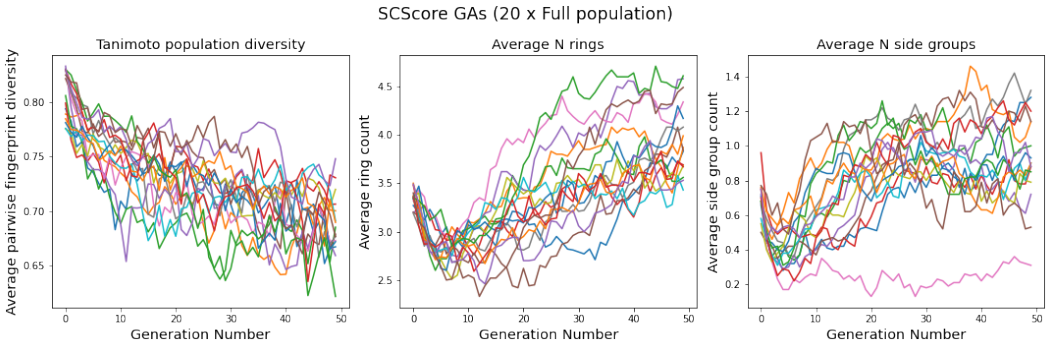



FIGURE 9.3: Plots to show diversity and composition for each population in 20 SCScore-led GAs: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right)

Average side group counts follow a similar pattern, with the exception of a single run, which maintained a low average count throughout the genetic algorithm run; this is likely due to the random aspect of mutations, where the chance of mutations occurring is controlled by a defined mutation rate. This particular run is getting 'unlucky' and not performing mutations as often as the rest.

### 9.1.2 Reducing the number of building blocks

The 16 runs with reduced OCELOT building block sets sampled lower numbers of unique molecules when optimized by SCScore, between 1000 - 1500 per run, compared to the full populations 1600 - 2200; the average fitness converges in a better manner than the 20 full population genetic algorithms, settling into a range of low SCScore values after 10 generations without the increase at later points in the run, as shown in Figure 9.4.

This behaviour is mirrored by the average counts of rings and side groups (see Figure A.50), which also converge to tight ranges of values early on in the genetic

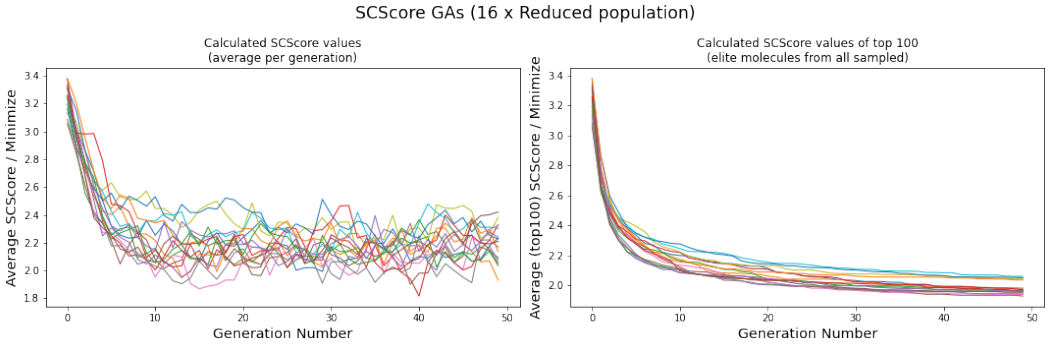



FIGURE 9.4: Plots to show how the fitness values (SCScore) change as each of the 16 reduced-population GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right)

algorithms; new molecules are still being sampled, as shown in Figure A.49, but the average counts are able to remain low, with calculated SCScores staying in a promising range.

## 9.2 Behaviour of SYBA-led genetic algorithms

This set of genetic algorithms seeks to maximise predicted SYBA scores, which should lower the molecular complexity of generated species; this should reveal some more information about the ring types within the OCELOT building block set which are 'preferred' by SYBA, and how combining rings influences the generated scores.

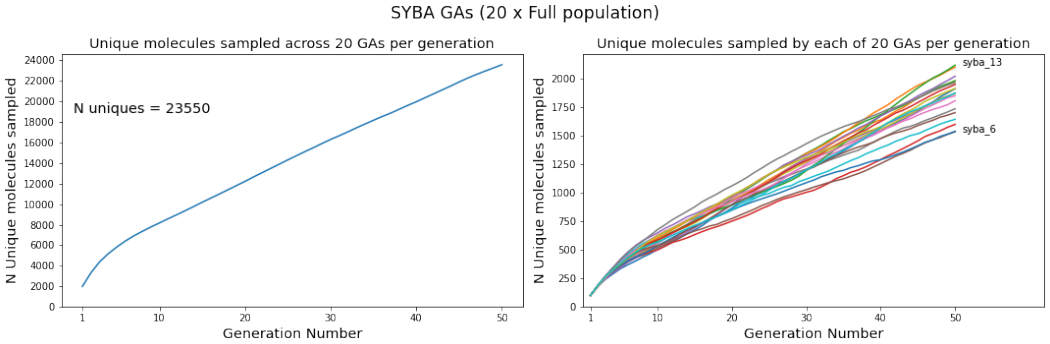



FIGURE 9.5: Plots to show how many unique molecules were sampled: by all 20 SYBA-led GAs (left); by each of the 20 GAs separately (right)

### 9.2.1 Maximization of SYBA scores

SYBA scores show a clean convergence across all 20 runs, with high scores being maintained after generation 20; this is despite the fact that unique molecules are still being sampled after this point, as shown in Figure 9.5. In contrast to the SCScore-led genetic algorithms, the right-hand portion of Figure 9.6 shows that the top-100 sampled molecule set is still being improved by later generations.

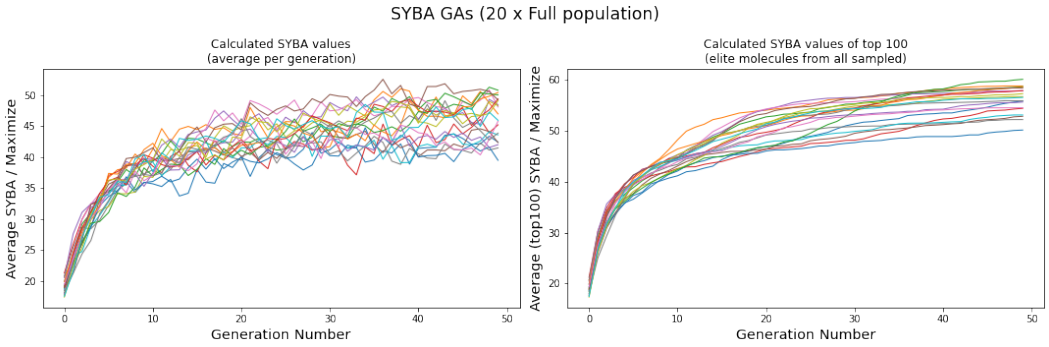



FIGURE 9.6: Plots to show how the fitness values (SYBA) change as each of the 20 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right)

### 9.2.2 Molecular composition and population diversity

The average number of rings rapidly drops at the start of SYBA-led genetic algorithms, then breaking into two 'clusters' of jobs; four runs maintain a low average ring count, while the rest exhibit rising counts, approaching the maximum allowed value of five rings. Side group counts also drop at the start of these runs, before rising to between 0.6 and 1.6 groups at the final generation.

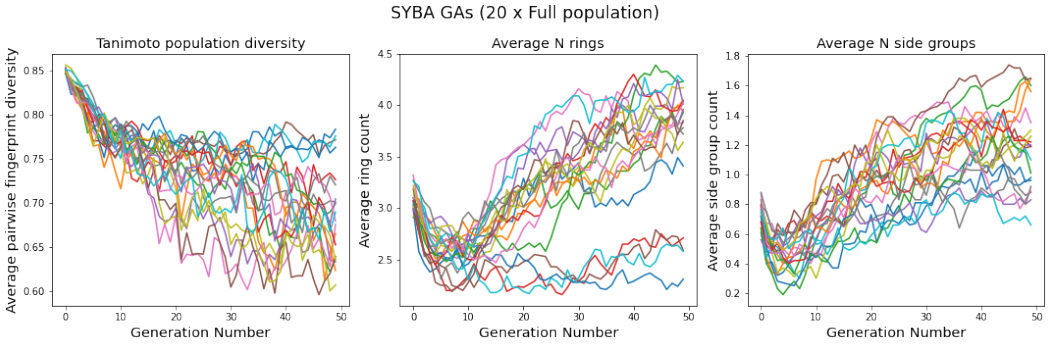



FIGURE 9.7: Plots to show diversity and composition in 20 SYBA-led GAs

### 9.2.3 Reducing the number of building blocks

Reducing the building block set for SYBA-led genetic algorithms had little effect; in some cases, the number of molecules sampled by a given run increases compared to the full population runs.

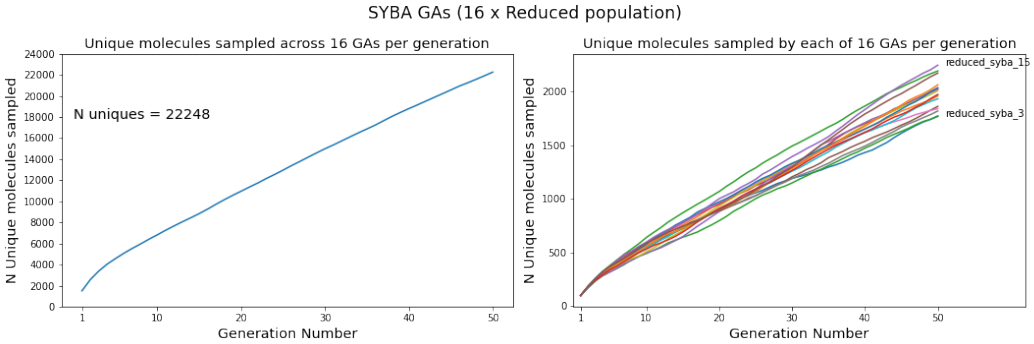



FIGURE 9.8: Plots to show how many unique molecules were sampled: by all 16 reduced population SYBA-led GAs (left); by each of the 20 GAs separately (right)

The average SYBA values behave similarly to the full population runs as genetic algorithms progress, as shown in the appendix Figure A.51; the same can be said for the average ring counts and levels of side group addition, as in appendix Figure A.52.

## 9.3 Analysis of all molecules sampled with synthetic difficulty GAs

A set of 18 molecules were frequently found by at least 50 of the 72 synthetic difficulty led genetic algorithms, with promising scores produced by both SYBA and SCScore; these are shown in Figure 9.9.

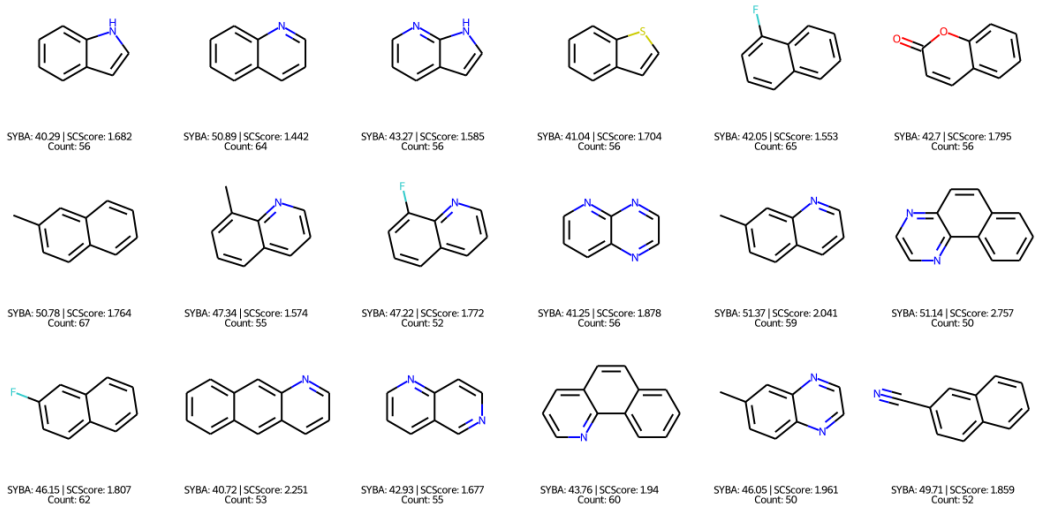



FIGURE 9.9: 18 molecules sampled by synthetic difficulty GAs multiple times, while achieving low SCScores and high SYBA scores.

These molecules illustrate a compromise between the two approaches to synthetic difficulty prediction, containing low ring counts, a maximum of one side group, and relatively simple ring types; it wouldn't be surprising to see these species appear as reagents in reactions.

### 9.3.1 Common motifs in 'synthetically accessible' molecules

A selection of commonly sampled molecular shapes can be extracted through clustering of all sampled molecules with generic Murcko scaffolds; these shapes are shared by molecules discovered with multiple genetic algorithms, and for the most part exhibit promising values in SYBA and SCScore analysis.

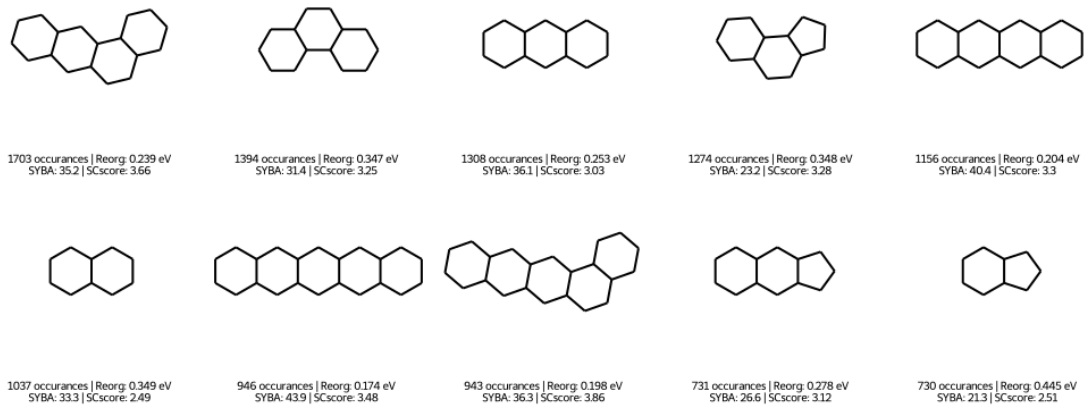



FIGURE 9.10: 10 generic molecular shapes commonly sampled by synthetic difficulty genetic algorithms, with average calculated properties.

### 9.3.2 Synthetically accessible molecules with low reorganisation energies

Similar analysis can be done with an additional filter to examine backbones which also exhibit good reorganisation energies, alongside being sampled multiple times amongst synthetic difficulty; these average values are also shown in Figure 9.10.

The top performing set of molecules once again commonly contains the linear pentacene shape, with various substitutions and six-membered ring types, as shown in Figure 9.11. These can be compared to the top performing set found by genetic algorithms which optimized reorganisation energy directly; the lowest value amongst molecules sampled by synthetic difficulty GAs is 0.1471 eV, which shows a penalty in the best scoring species when compared to the lowest found in reorganisation energy GAs, which tended to be closer to 0.1 eV.

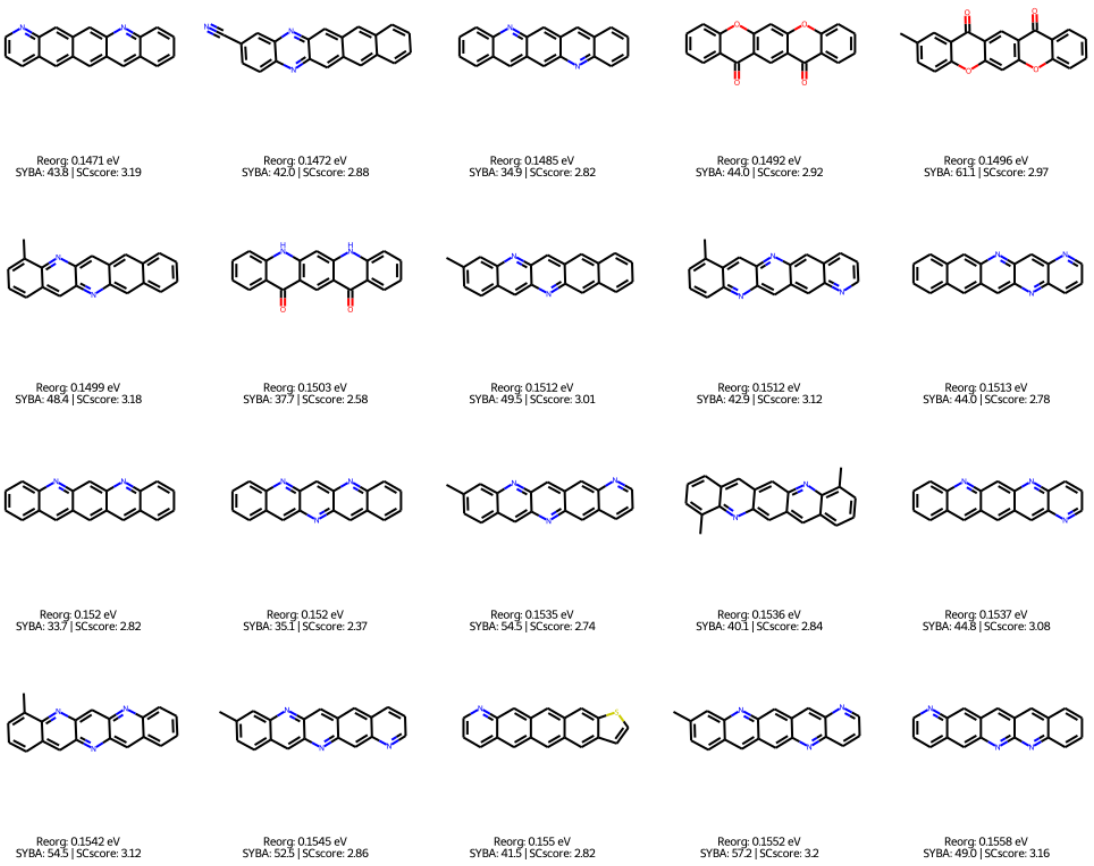



FIGURE 9.11: 15 molecules sampled by synthetic difficulty genetic algorithms, achieving the lowest average reorganisation energies from these runs.

## 9.4 Conclusions from synthetic difficulty-led GAs

This chapter covered the implementation of SYBA and SCScore as fitness functions to be optimized in MolBuilder genetic algorithms; different convergence behaviour is observed when targeting these scores, including unexpected rises in the population average SCScore values.

Top performing molecules according to both measures appear much simpler than those found with reorganisation energy GAs, containing fewer rings and side groups; these species look like reasonable precursors, which matches the expected behaviour of SCScore (where molecules more similar to reaction reagents achieve better scores).

These results show that SYBA and SCScore can be optimized well by genetic algorithms; there is a noticeable 'penalty' in calculated reorganisation energies

amongst the sampled molecules, which is to be expected since these algorithms were built to not consider this property, but it is reassuring to see these campaigns sampling species with higher, but still respectable, reorganisation energy values.

# Chapter 10

# Multi-objective GAs

The final way in which this project assesses synthetic difficulty as part of the computational materials discovery process involves consideration of predicted scores alongside the optimization of target physical properties

This section will cover the effects of minimizing predicted SCScores and electron reorganisation energies within a combined fitness function during MolBuilder genetic algorithms, again constructing molecules with the OCELOT building blocks. The results of these runs will be compared to the single-objective variants of each property, to see what kind of molecules are generated in each case.

## 10.1 Behaviour of multi-objective genetic algorithms

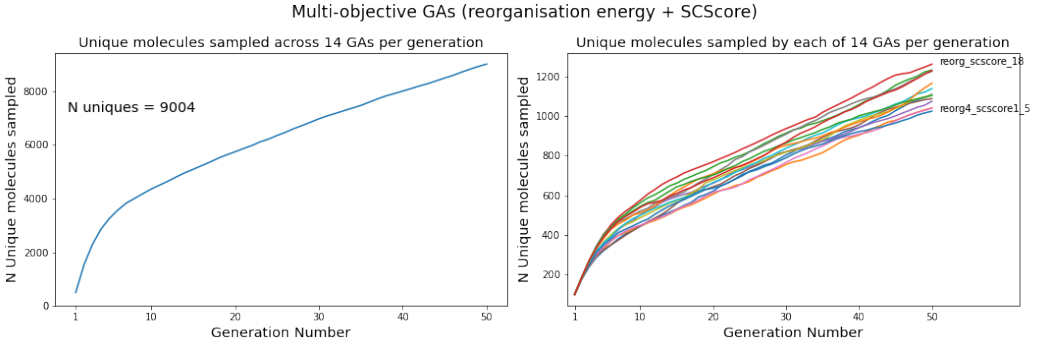



FIGURE 10.1: Plots to show how many unique molecules were sampled: by all 14 multi-objective GAs (left); by each of the 14 GAs separately (right)

14 genetic algorithms were conducted with multi-objective fitness functions, sampling a range of 'weights' placed on each objective, to see what sort of impact different ratios of physical property to synthetic difficulty consideration has on generated species; too much focus on one objective would likely lead to similar results compared to the single-objective genetic algorithms run with that target property.

Across all of these runs, genetic algorithms sampled far fewer molecules than their separate, single-objective counterparts, discovering only 9004 unique species in total.

### 10.1.1 Optimizing reorganisation energy and SCScore

By constructing fitness values from both reorganisation energy and SCScore, genetic algorithms should be aiming to target molecules which look more like reactants according to Reaxys, and have low reorganisation energies; there will likely be a compromise between the best possible values for each property.

A simple multi-objective fitness function was trialled, where reorganisation energy and SCScore were considered at equal weights (i.e. same level of importance during optimization). In order to determine the fitness score, each property was scaled to a value between 0 and 1, according to the range of values sampled so far, then squared; the fitness score is calculated as the square route of the sum of these squares:

$$F_{MULTI} = \sqrt{\lambda^2 + (SCScore)^2}$$

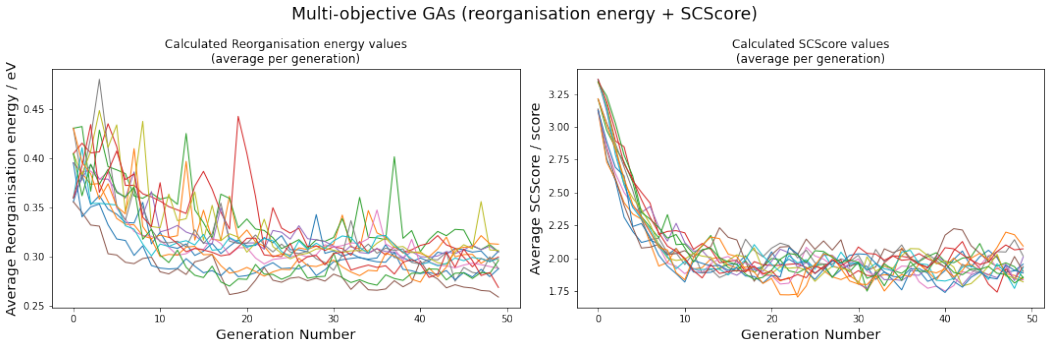



FIGURE 10.2: Plots to show how the individual properties change as each of the 14 GAs progress: reorganisation energy (left) and SCScore (right)

While both properties exhibited convergence, the final value ranges for reorganisation energies at generation 50 were not as optimal as values found in the single-objective reorganisation energy genetic algorithms; SCScore values were able to converge at lower values, and stay in that range until the end of each run, which the single-objective runs did not achieve.

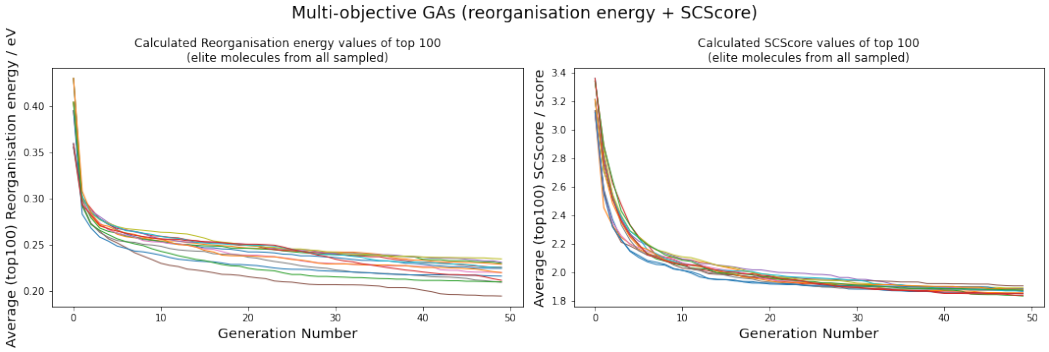



FIGURE 10.3: Plots to show how the individual properties change for the top 100 molecules sampled up to each generation each of the 14 GAs progress: reorganisation energy (left) and SCScore (right)

Looking at the minimum values rather than the averages, Figure 10.4 reveals that while the lowest SCScores were updated over the course of these genetic algorithms, molecules with better reorganisation energies than those in the initial population were rarely found. The genetic algorithms solely optimizing reorganisation energies were able to drop minimum values by up to 0.08 eV (see Figure 7.5); this suggests that optimizing multiple properties as part of the fitness function has required a compromise in the genetic algorithms ability to reduce reorganisation energies in sampled molecules.

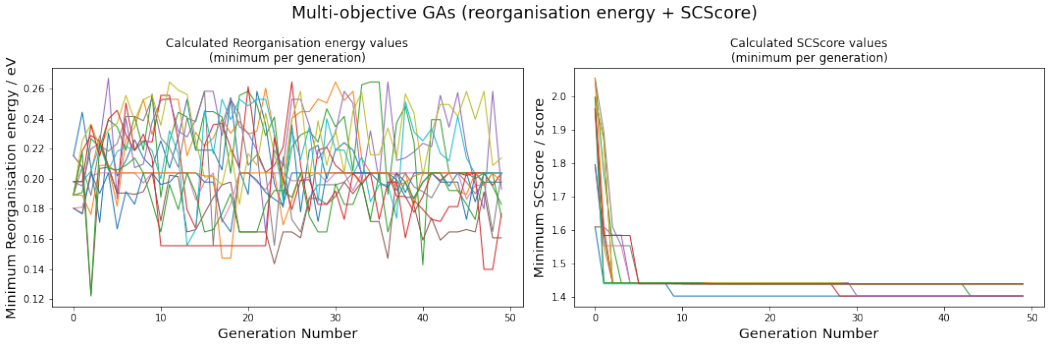



FIGURE 10.4: Plots to show how the minimum values of each property changes as each of the 14 GAs progress: reorganisation energy (left) and SCScore (right)

### 10.1.2 Varying property weights

Across the 14 multi-objective genetic algorithms, jobs were run with fitness functions where the relative importance of each property was modified; the weights ($A : B$) were set at ratios of 1 : 1, 2 : 1 and 4 : 1, with larger weights being placed on reorganisation energy in each iteration.

$$F_{MULTI,WEIGHTED} = \sqrt{(A \times \lambda)^2 + (B \times SCScore)^2}$$

This was an attempt to drop the average reorganisation energies of sampled populations, by placing more importance on the minimization of that portion of the fitness function.

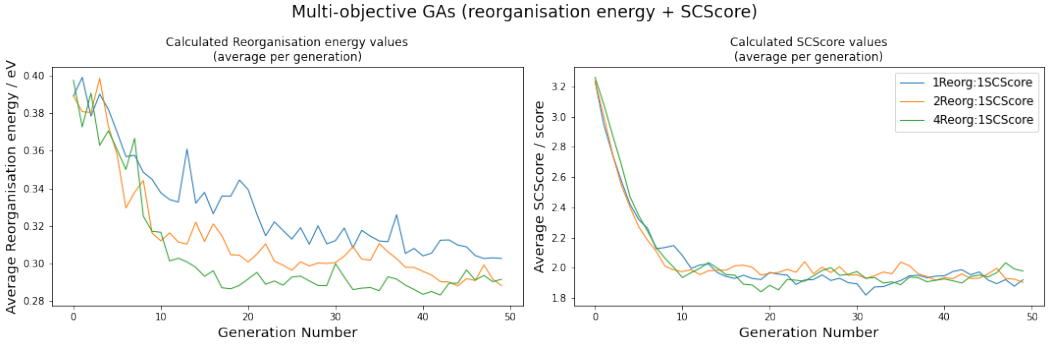



FIGURE 10.5: Plots to show how the individual properties change as each of the 14 GAs progress, grouped by the weights placed on each property in the fitness function: reorganisation energy (left) and SCScore (right)

Modifying these weights to place more importance on reorganisation energies appears to have the mixed effects; looking at the values per population in Figure 10.5 shows that average reorganisation energies typically drop as the weight on this part of the fitness function increases. The average values of the top 100 'elite' sampled molecules shown in the left portion of Figure 10.6 exhibit a less clear trend.

Increasing the weight placed on reorganisation energy in the fitness functions allowed genetic algorithms to achieve lower average reorganisation energy values, with little sacrifice to the calculated SCScores. However, this result is not consistent as the weights are increased further; fitness functions weighted at 2 : 1 sampled elite sets with lower average reorganisation energies than those weighted at 4 : 1.

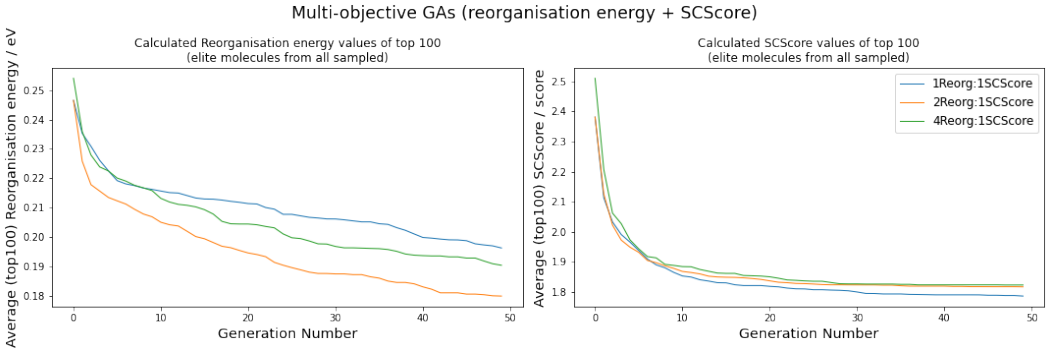



FIGURE 10.6: Plots to show how the individual properties change for the top 100 molecules sampled up to each generation each of the 14 GAs progress, grouped by the weights placed on each property: reorganisation energy (left) and SCScore (right)

This could be a consequence of the random aspect genetic algorithms exploit during chemical space exploration, where the runs at 4 : 1 weighting simply got 'unlucky'; further repeats at different weights would help to confirm this behaviour.

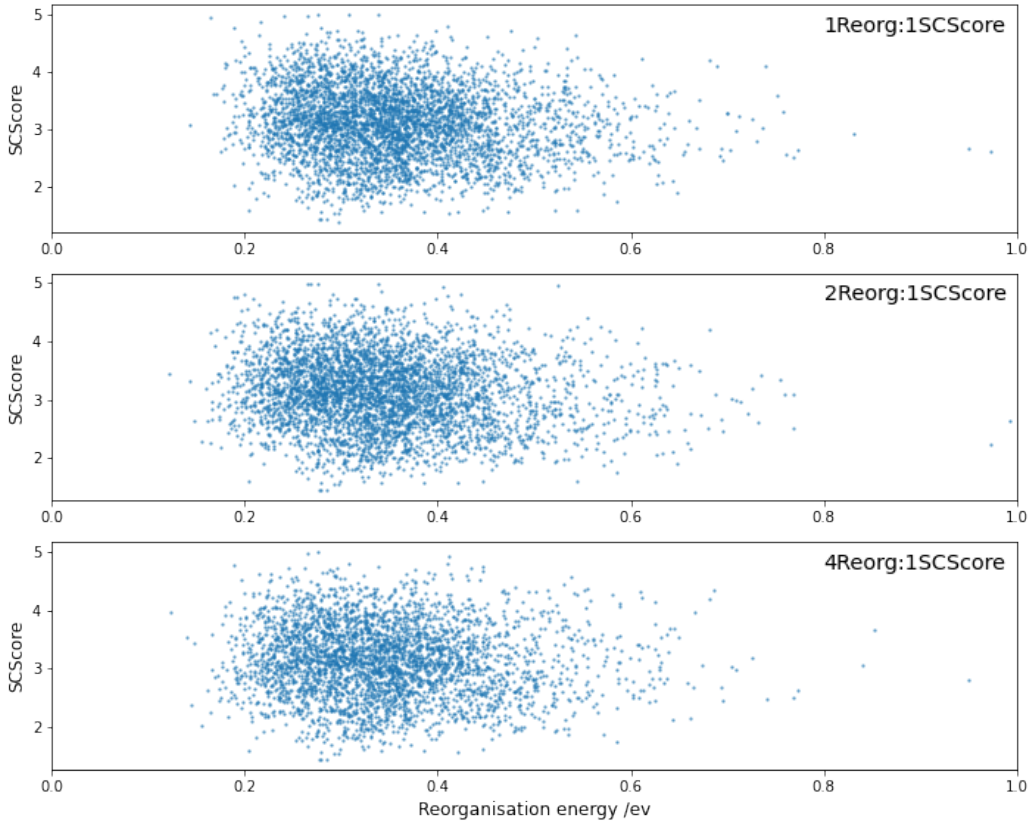



FIGURE 10.7: Scatter plots comparing reorganisation energies and SCScores for molecules sampled with multi-objective algorithms, categorised by the weights placed on each property.

Plotting the properties against one another in Figure 10.7 shows similar distributions for each weight trialled; while lower reorganisation energies tend to appear at higher

weights, the improvement is not great. This suggests that much higher weights should be trialled in future work, to further improve the performance of these multi-objective fitness functions in genetic algorithms.

## 10.2 Top performing molecules

The top performing molecules from multi-objective genetic algorithms appear quite different from past results; rather than all instances being variations on the linear pentacene structure, molecules with different ring counts appear.

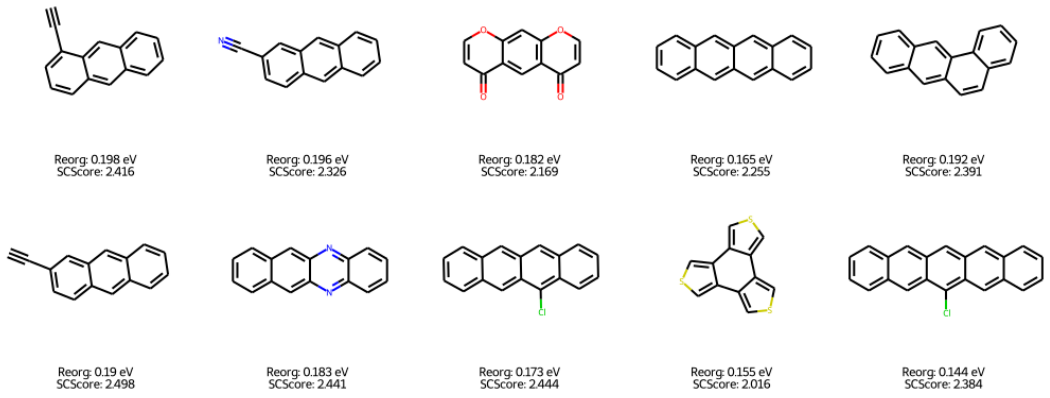



FIGURE 10.8: 10 top performing molecules, with both low reorganisation energies and low SCScores, discovered during multi-objective genetic algorithms.

The influence of including SCScore in fitness functions is clear, where these species appear more like 'reagents', or targets which would be accessible in fewer reaction steps than the top-performing linear-pentacene type structures which dominated top performing sets from single-objective reorganisation energy genetic algorithms.

### 10.2.1 Overlap with single-objective GAs

Of the 9,004 molecules sampled, 1,796 were also sampled by both the reorganisation energy and SCScore-led single objective genetic algorithms; this means roughly 80% of the molecules sampled with multi-objective genetic algorithms were newly discovered. The top molecules, according to reorganisation energy, that were rediscovered by the multi-objective runs are shown in Figure 10.9.

## 10.3 Conclusions from multi-objective genetic algorithms

This chapter has covered the use of genetic algorithms where multiple objectives were optimized as part of a single fitness function, aiming to sample more synthetically accessible molecules which also exhibit promising reorganisation energies.

Inclusion of SCScore in the fitness functions promotes exploration of the chemical space which appear more synthetically accessible, with the top-performing set of species differing well from single-objective genetic algorithms for each property; while linear pentacene type structures still appear, most of the top 40 sampled molecules are constructed with fewer rings and side groups.

The best reorganisation energy values in sampled molecules tend to be higher than those sampled with reorganisation energy led genetic algorithms, indicating a penalty to the best values accessible through multi-objective optimization. Varying the relative weights on each objective allowed for lower reorganisation energies; further increases to the weights should be trialled to promote better reorganisation energy values, while determining if a compromise on SCScores is observed.

The choice of where best to consider synthetic difficulty appears dependant on the requirements of the project and/or collaborators. When applied as a post-hoc filter to molecules sampled with property-led genetic algorithms, a set of similar species can be selected which exhibit good reorganisation energies, and appear to be more synthetically accessible than other species sampled this way (see Figure 8.20). This approach seems more suited to tasks where the best properties are sought after, and synthetic accessibility in candidate species is desired but not essential.

Including synthetic difficulty as part of a multi-objective fitness function *during* genetic algorithm sampling promotes more diversity in the top performing molecule sets, but the best reorganisation energies amongst these species is raised (see Figure 10.8). This approach would likely work better for the earlier stages of a project, where idea generation and the validation of predicted properties is more important; suggesting synthetically accessible species should promote a lower synthetic difficulty 'barrier' to the experimental validation of calculated results.

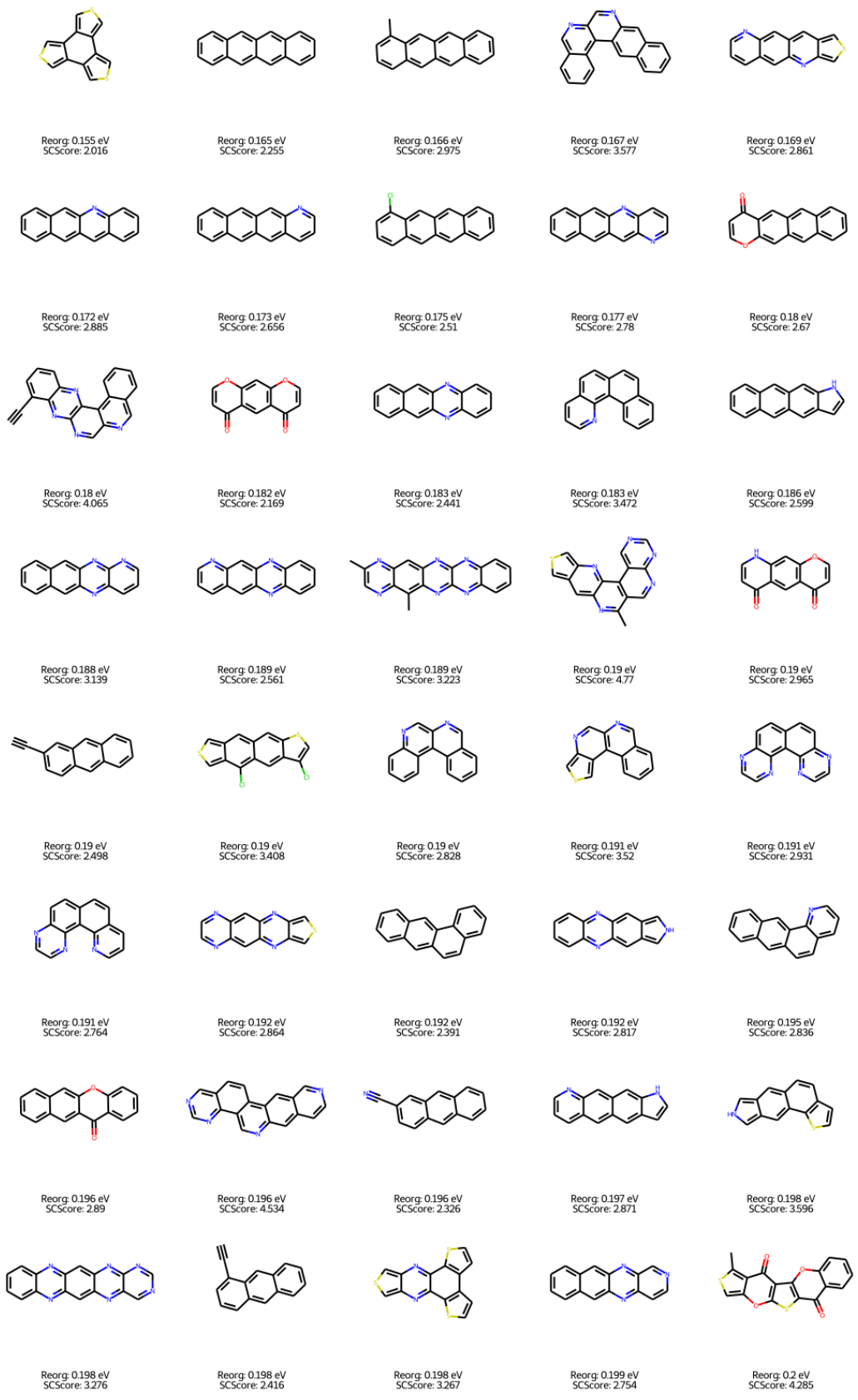



FIGURE 10.9: 40 top performing molecules, with both low reorganisation energies and low SCScores, rediscovered during multi-objective genetic algorithms.

# Chapter 11

# Mixed crystals project

This side project conducted at the start of this PhD involved collaboration with Jim Wuest and Norbert Villenueve, from the University of Montreal; binary mixed crystals were constructed both computationally and experimentally, sampling a range of compositions, to study the crystallisation behaviour of such systems where the two component species are similar.

## 11.1 Binary mixed crystal systems

Four species were chosen for this study; Figure 11.1 shows these molecular structures, which were selected due to a lack of isostructural packing between the species most stable polymorphs despite clear molecular similarity; in each case, a 5-membered cycle is flanked by two benzene rings, with a single atom on the 5-membered ring being changed between species.

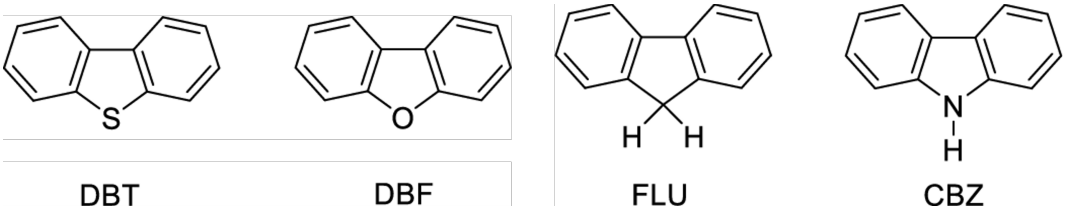



FIGURE 11.1: Species studied as mixed crystals: Dibenzothiophene (DBT), dibenzofuran (DBF), fluorene (FLU), and carbazole (CBZ).

## 11.2 Validation of single-species predicted crystal structures

Starting from 2D sketches of each species, molecular geometries were optimised with Gaussian09, then passed to DMACRYS for distributed multipole analysis. CSP was performed using the optimised structure with the FIT potential, aiming to generate 10,000 valid crystal structures within the ten most common space groups (*fine10*).

The CSD database contains experimental data for each species under study, so once crystal landscapes had been generated with CSPy, COMPACK searches were carried out to confirm that the CSP process had generated experimentally observed packing arrangements.

### 11.2.1 Comparing structures between species

On the predicted crystal structure landscape of dibenzothiophene, the structure of second-lowest energy matches experimental data for the known *P21/n* form. Higher up on the landscape, the highlighted *Pnma* structure matches experimental data extracted from the CSD for DBF, FLU, and CBZ.

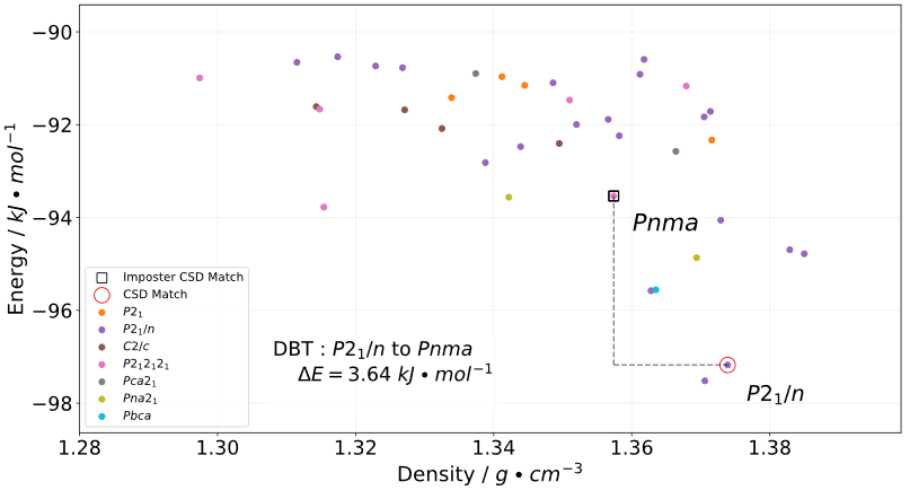



FIGURE 11.2: Predicted crystal-structure landscape for DBT.

This shows that while dibenzothiophene prefers to form a *P21/n* crystal structure, the *Pnma* structure, analogous to the other species in this study, is also present with an associated energy penalty.

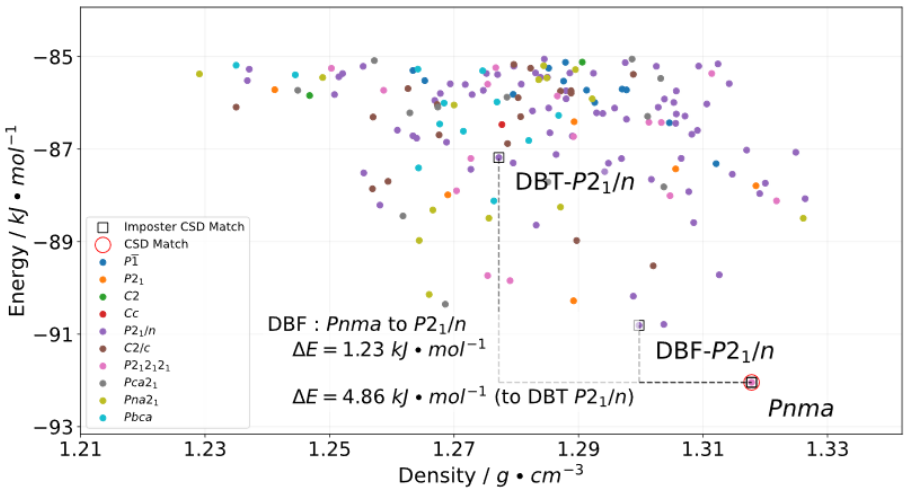



FIGURE 11.3: Predicted crystal-structure landscape for DBF.

The landscape for dibenzofuran provides some more information, with the preferred *Pnma* form once again shared by DBF, FLU and CBZ. The *P21/n* form of dibenzofuran is present on the landscape at a higher energy; this structure was found through energy minimization of the dibenzothiophene crystal structure where all molecules were replaced with dibenzofuran, and the optimized structure matched the marked crystal discovered during crystal structure prediction. Higher up on the landscape still, the crystal matching dibenzothiophene's *P21/n* structure can be found.

## 11.3 Creating binary mixed crystal supercells

The experimental crystal structure of each 'host' was then used to build supercells containing 32 molecules, to be used as starting points for 'mixed crystal supercells'; in cases where supercell dimensions are not equal in each direction, each possible combination was tested.

Starting from the 'host supercell', N molecular positions were chosen at random to be swapped for imposter species, using Python random seeds to ensure reproducibility. Each selected 'host' molecule was then replaced the 'imposter' structure, in the imposter's optimal molecular geometry.

### 11.3.1 Dealing with configurational entropy

In order to account for variation in imposter positions (Figure 11.4), 40 configurations were generated for each level of imposter addition, giving different sets of swapped positions.

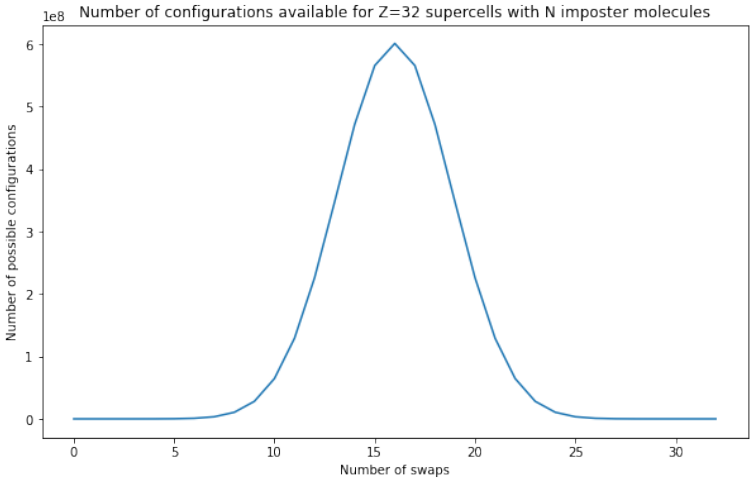



FIGURE 11.4: As the host:imposter ratio reaches 50:50, the number of possible imposter position sets increases rapidly

This many configurations were tested for each system in order to account for the effect of placing imposter molecules in various parts of the supercell, while maintaining a reasonable computational cost; if imposters are present at a different set of positions within the supercell, the optimal packing arrangement can vary, leading to a range of potential energies at a given ratio of host to imposter.

### 11.3.2 Optimizing a mixed crystal supercell

Once the imposter molecules were swapped in, the mixed species supercell was subjected to energy minimization; this allowed the molecules to rotate and shift, settling into their optimal positions. The overall process employed in this mixed crystals study is summarised in Figure 11.5.

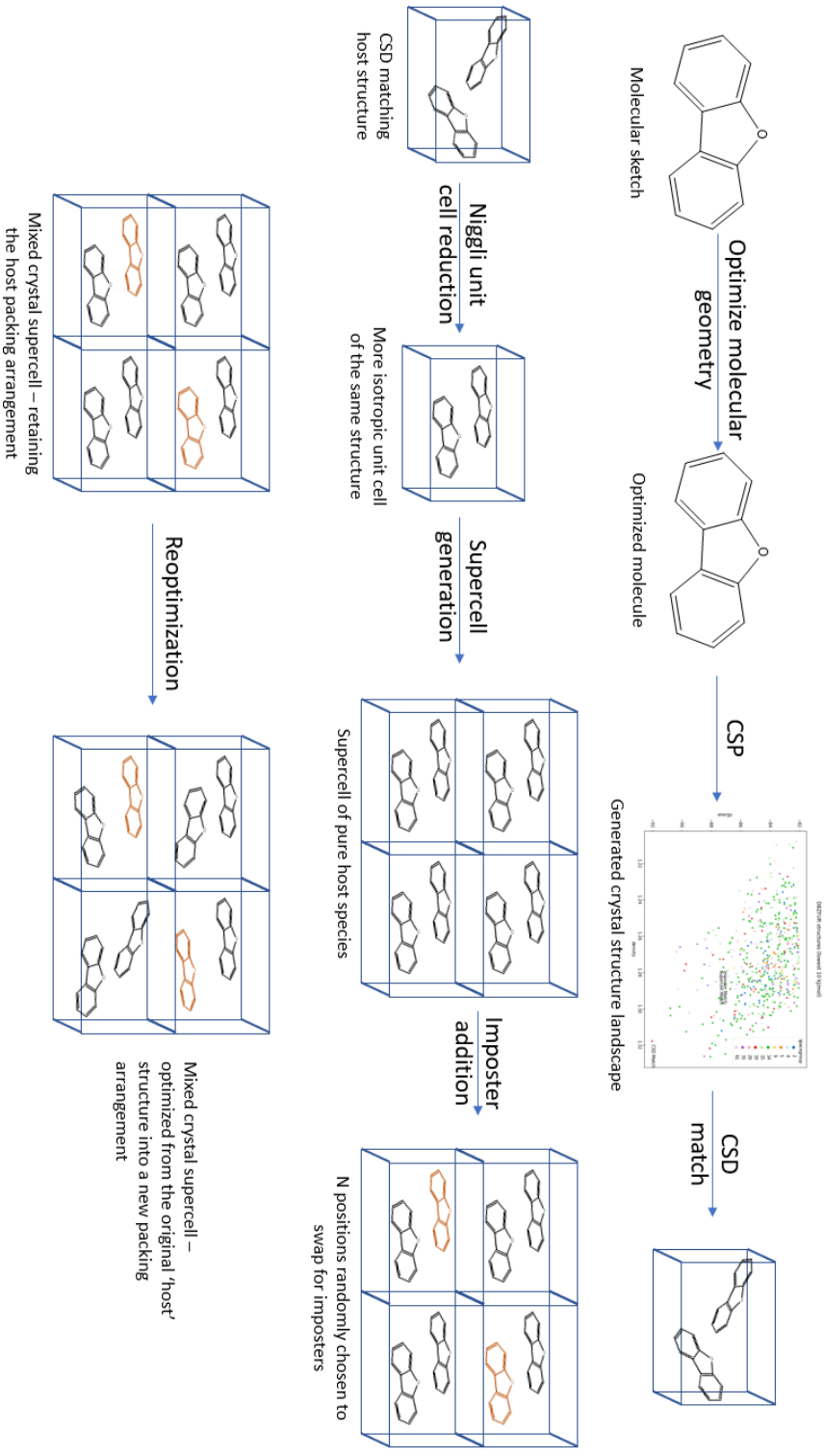



FIGURE 11.5: Overview of computational simulation of mixed crystal systems.

## 11.4 Analysing mixed crystals with varying imposter content

After optimizations are complete, plots can be produced for each system summarising the change in lattice parameters, energy, density and volume as the composition varies. The impact of changing imposter positions can be seen in each system; while they all follow trends in each parameter, there were minor variations between the 40 random configurations.

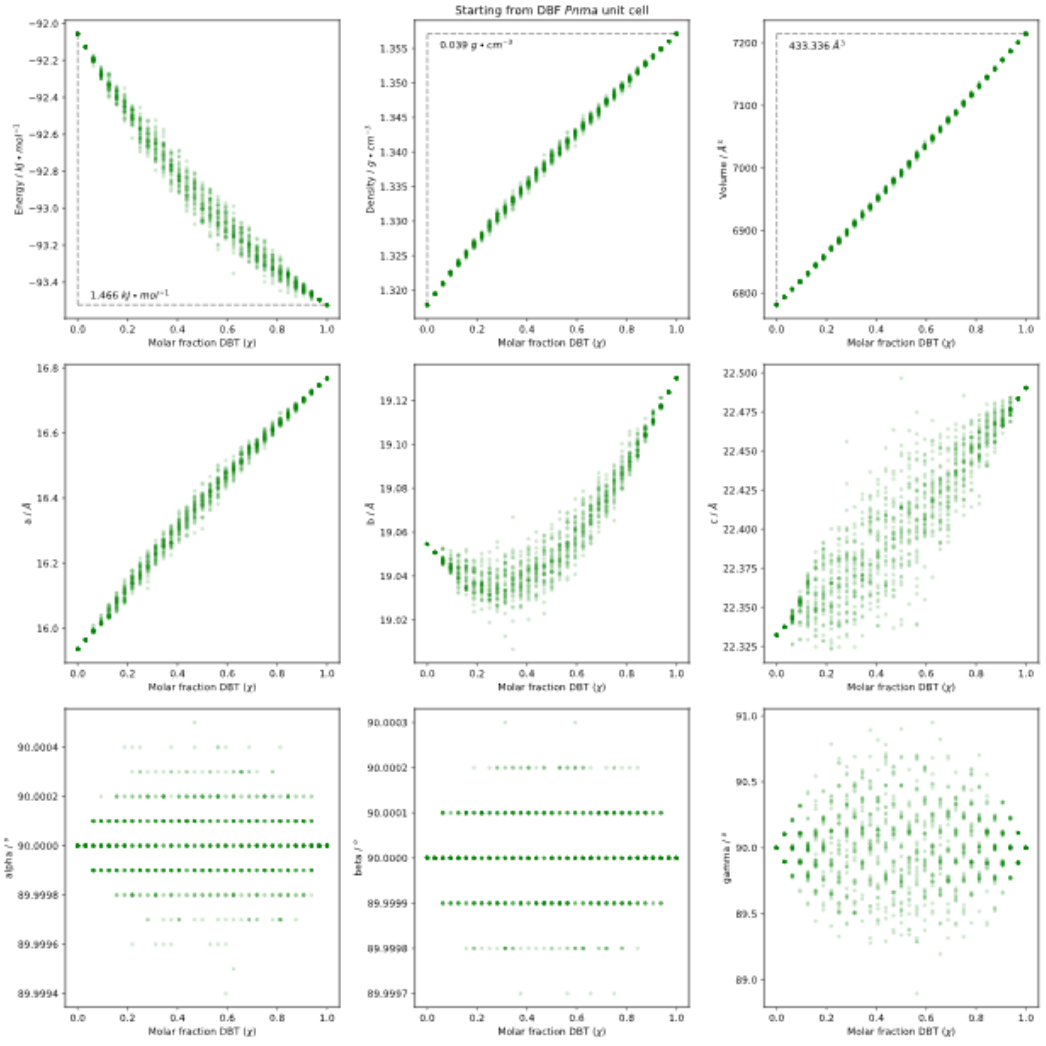



FIGURE 11.6: Plots of the dibenzofuran / dibenzothiophene mixed crystal system, summarising 40 sets of randomly configured supercells with varying compositions, when the initial 'host' supercell was built from dibenzofuran *Pnma* unit cells.

The importance of the chosen 'host' supercell is clear when comparing Figure 11.6 and Figure 11.7; increasing the ratio of imposter molecules prompts vastly different behaviour depending on 'which side' the supercells start from.

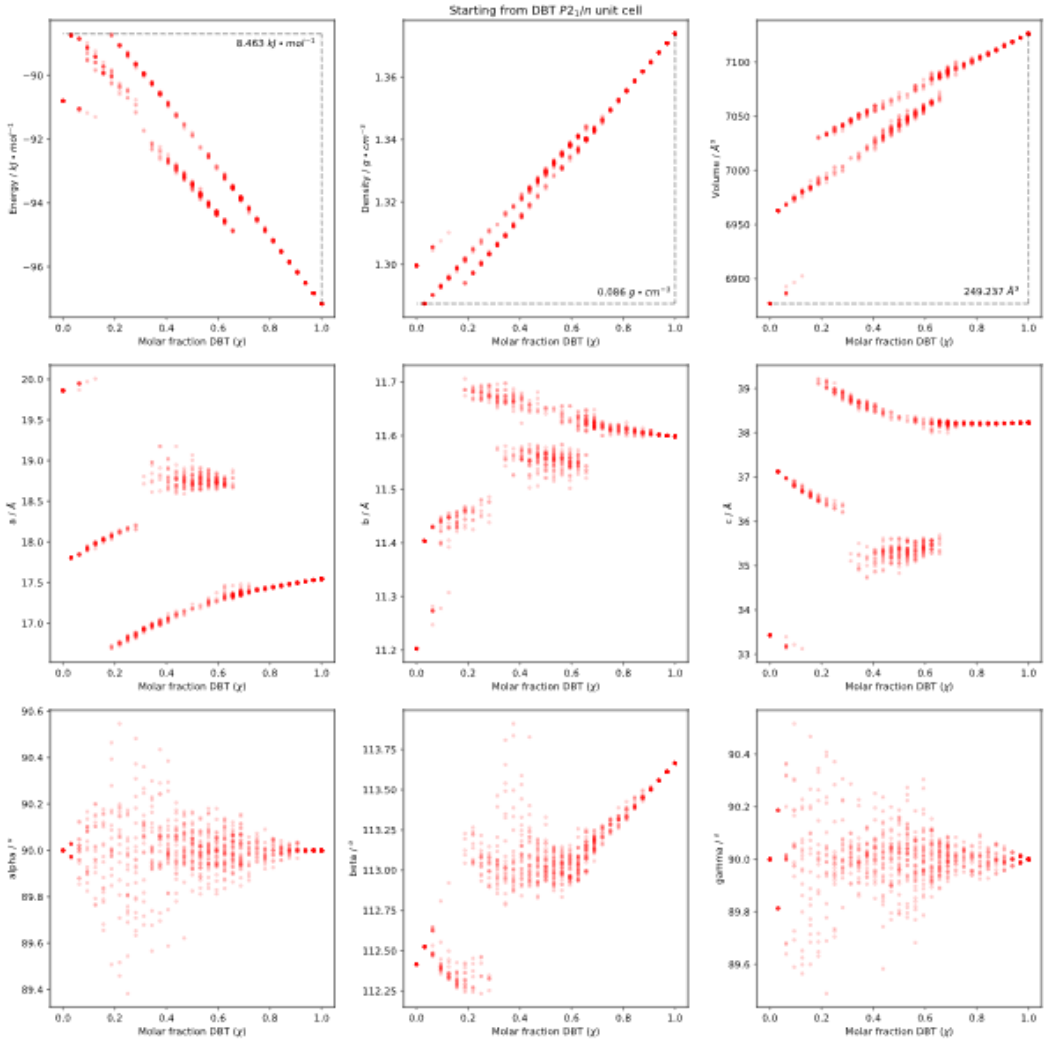



FIGURE 11.7: Plots of the dibenzofuran / dibenzothiophene mixed crystal system, summarising 40 sets of randomly configured supercells with varying compositions, when the initial 'host' supercell was built from dibenzofuran *P21/n* unit cells.

## 11.5 Comparison to experimental findings

Work on the dibenzofuran / dibenzothiophene system mostly validates the computational results (Figure 11.8). Each host-to-imposter ratio examined by Jim Wuest and Norbert Villenueve was found to adopt the *Pnma* packing arrangement of dibenzofuran, in agreement with most predicted results. However, at higher dibenzothiophene levels (above 59%) the computational model predicts that the *Pnma* packing arrangement becomes energetically disfavoured.

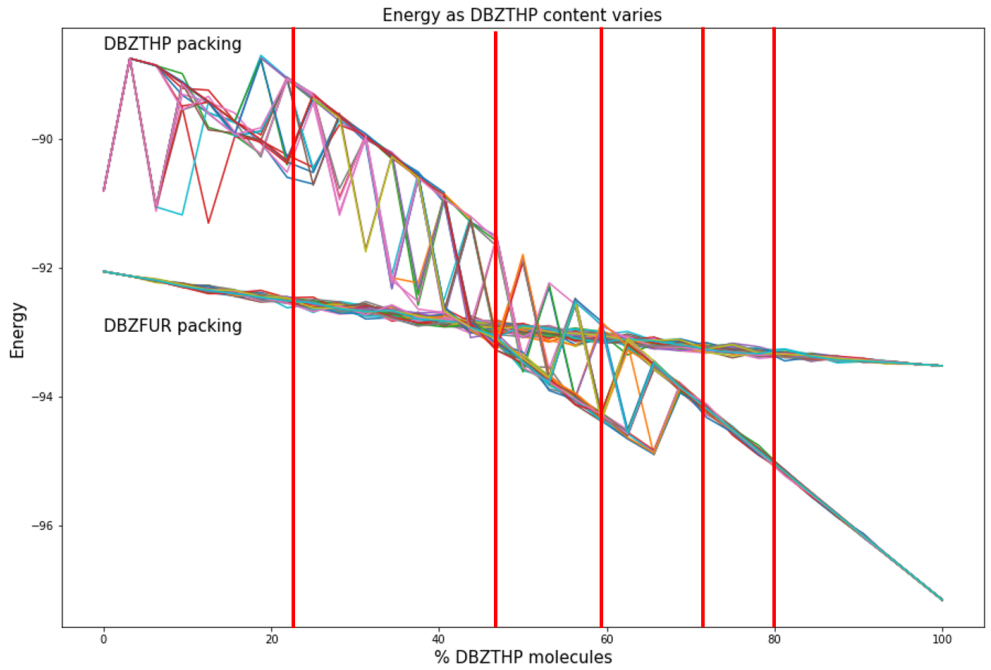



FIGURE 11.8: A 'dual energy' plot, used to compare data from the dibenzofuran / dibenzothiophene system. Red lines indicate compositions produced experimentally; DBF:DBT ratios 77:23, 54:46, 41:59, 27:73 and 20:80

Wuest and Villenueve were able to induce crystallisation of dibenzothiophene in the *Pnma* form through sublimation of dibenzothiophene onto seed *Pnma* crystals of carbazole, fluorene and dibenzofuran, captured as an image in Figure 11.9.

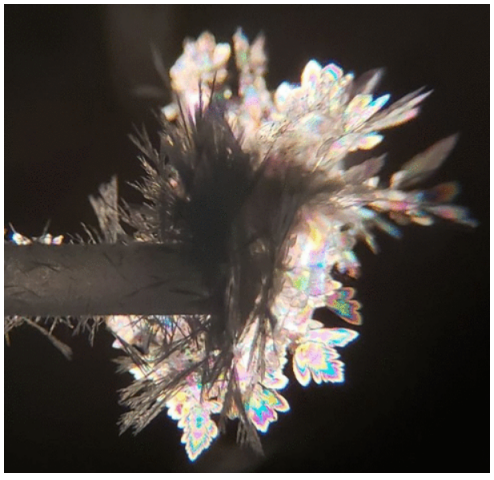


FIGURE 11.9: From the work of Villeneuve et al. (2022); sublimed crystals of pure DBT imaged by optical microscopy under polarized light. The thin plates correspond to the metastable *Pnma* polymorph and the needles to the previously reported *P21/n* form.

# Chapter 12

# Conclusions

Overall, this project examined the use of genetic algorithms to explore defined chemical spaces, aiming to discover molecules which exhibit promising properties for applications in organic semiconductor materials; various approaches were attempted to include biases towards more synthetically accessible species, including the use of multiple computational tools which seek to estimate scores regarding synthesis.

## 12.1 Key findings

The objectives within this project are summarised across three main concepts; generation of molecules while optimizing predicted properties, assessing synthetic difficulty given molecular structure information, and how these ideas can be combined in order to assist larger materials discovery workflows.

Exhaustive chemical space explorations in two chemical spaces highlight the need for more efficient and directed approaches to computational molecular generation; limited sets of molecular 'building blocks' were combined with rigid design rules on composition, revealing a full chemical space of 135,744 aza-substituted pentacene species, and a partially explored space of rigid small molecule organic semiconductors, constructed with building blocks extracted from the OCELOT chemical space, of at least 12 million species.

Previous work in the Day group has highlighted the efficiency of genetic algorithms in exploration of the azapentacene space; in this project, 172 genetic algorithms were conducted around the OCELOT building block sets, optimizing physical properties related to charge mobility, aiming to reduce synthetic difficulty by optimizing estimated scores, or a combination of these two concepts as part of multi-objective fitness functions.

Across these algorithms, several familiar species were discovered including a high level of pentacene-type structures, particularly when algorithms were tasked with the minimization of electron reorganisation energies; when these genetic algorithms were analysed with synthetic difficulty scoring through the SYBA and SCScore tools, the top-performing subset is dominated by aza-substituted linear pentacenes, frequently containing cyano-, methyl- and ethyne side groups.

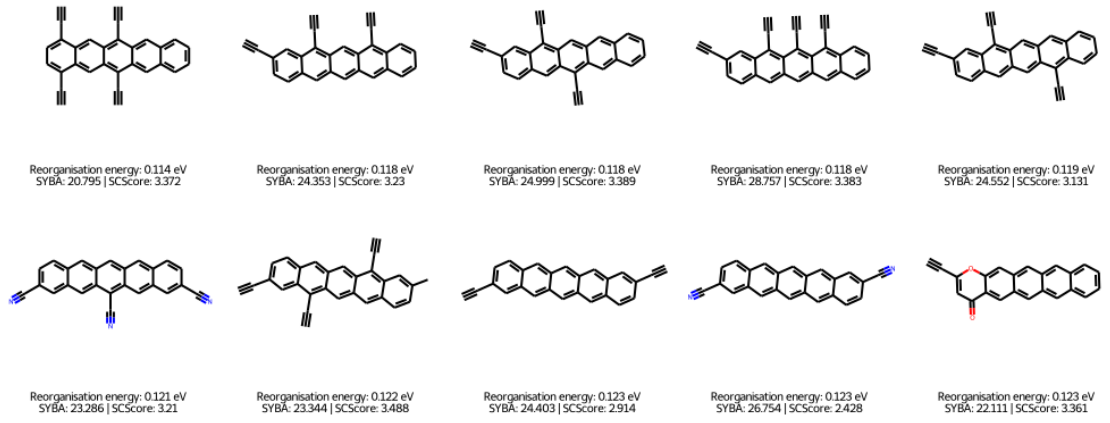



FIGURE 12.1: 10 top performing molecules sampled through genetic algorithms, which exhibit the most promising calculated reorganisation energies alongside low molecular and synthetic complexity scores.

Post-hoc synthetic difficulty analysis of molecules sampled in this way reveals that the scoring methods capture different types of information for use in ranking species; there are many cases where perceived synthetic difficulty through analysis of fragments within a molecule (with SYBA) does not match up with a reduced synthetic complexity based on a species position in a reaction network (with SCScore). The validity of these scoring methods is exhibited through analysis of experimental molecules extracted from the original OCELOT dataset, where the methods evaluate most of these species with reasonable scores.

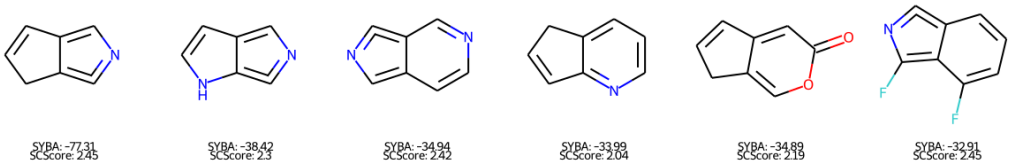



FIGURE 12.2: Sampled species where disagreement arises between scoring of molecular complexity (SYBA) and synthetic complexity (SCScore).

Optimizing genetic algorithms solely to reduce synthetic difficulty returns a rather different set of species, where the top-performing subsets which are identified as synthetically accessible are composed of fewer rings, and appear similar to molecules which would likely be seen as precursors in synthetic pathways; this behaviour is likely driven by the optimization of the Coley et al. (2018) SCScore, where such species are given better scores due to a perceived low number of reaction steps commonly needed to access these smaller and simpler structures.

Reducing the set of building blocks available to construct molecules during genetic algorithms allowed for the restricted exploration of chemical spaces, with the idea of reducing molecular complexity as a proxy to lower synthetic difficulty; targeting more specific areas of the overall chemical space did not result in a large improvement of predicted synthetic difficulty scores, but this approach remains useful for tasks where subsets of a larger potential space can be explored individually.

Multi-objective optimization within genetic algorithms produces the most interesting results; while linear aza-substituted pentacene species still appear in the top-performing subset of discoveries, there is much more variation in the types of molecules suggested through these campaigns, where both reorganisation energy and SCScores work in conjunction to produce synthetically accessible candidates with reasonable and promising reorganisation energies.

Varying the weights placed on each of these properties is essential to gathering such results; equal consideration of synthetic complexity and reorganisation energies produced populations of molecules with higher average reorganisation energies, while increased emphasis on the physical property allowed these averages to drop while maintaining promising calculated SCScores.

## 12.2 Future work

This project produced results only on the computational portion of a larger materials discovery workflow; top performing, newly discovered candidates with low predicted synthetic difficulties would be suggested to experimental researchers for synthesis and analysis, both in an effort to reduce their workloads, and to provide validation of both calculated physical properties and the synthetic difficulty estimation methods.

The MolBuilder application for exploring chemical space with genetic algorithms has many opportunities for further development; as stated previously the algorithm is currently limited to fused aromatic ring systems with optional side groups attached, so the classes of molecules accessible with this approach could be expanded through the implementation of more versatile genetic operations, allowing for the construction of more flexible species, and different potential applications of the molecules to be targeted.

Synthetic difficulty is an important factor to assess when combining computational workflows with materials discovery processes, and many more tools to predict this concept are available outside of the ones used in this project; with increasing availability of both data and computational power, it's vital that further work is done to consider not only the physical properties a candidate may exhibit, but also the way in which they may be synthesised.

Multi-objective fitness functions have been shown to work well in optimizing a combination of these factors, so analysis of synthetic difficulty does not need to be relegated to a post-hoc step during computational molecular generation processes.

## 12.3 Final remarks

I would like to emphasise the importance of collaboration between computational and experimental teams; while such opportunities were limited during the course of this project, this sort of collaboration enables both sides of materials discovery research to 'lift each other up'.

With the rise in availability and use of artificial intelligence, the importance of continued experimental research must be made abundantly clear; these tools work well for studies where previous knowledge exists and can be used, but over-reliance on AI without the creativity that a 'human touch' can provide may limit research to findings that can be extrapolated from historical work.

Computation can predict properties and suggest candidates for lab work, and experimental studies can provide validation of the calculations and further data to aid the training of more versatile predictive models. Closer collaboration between these methodologies can only be beneficial to the increased acceleration of materials discovery processes, and to academic research overall.

# Appendix A

# OCELOT genetic algorithms

## A.1 Initial populations

36 populations of 100 molecules each, generated using the OCELOT building blocks, which were used as starting points for MolBuilder genetic algorithms

### A.1.1 Full building block sets

20 unique populations of molecules, generated using the full set of OCELOT building blocks.

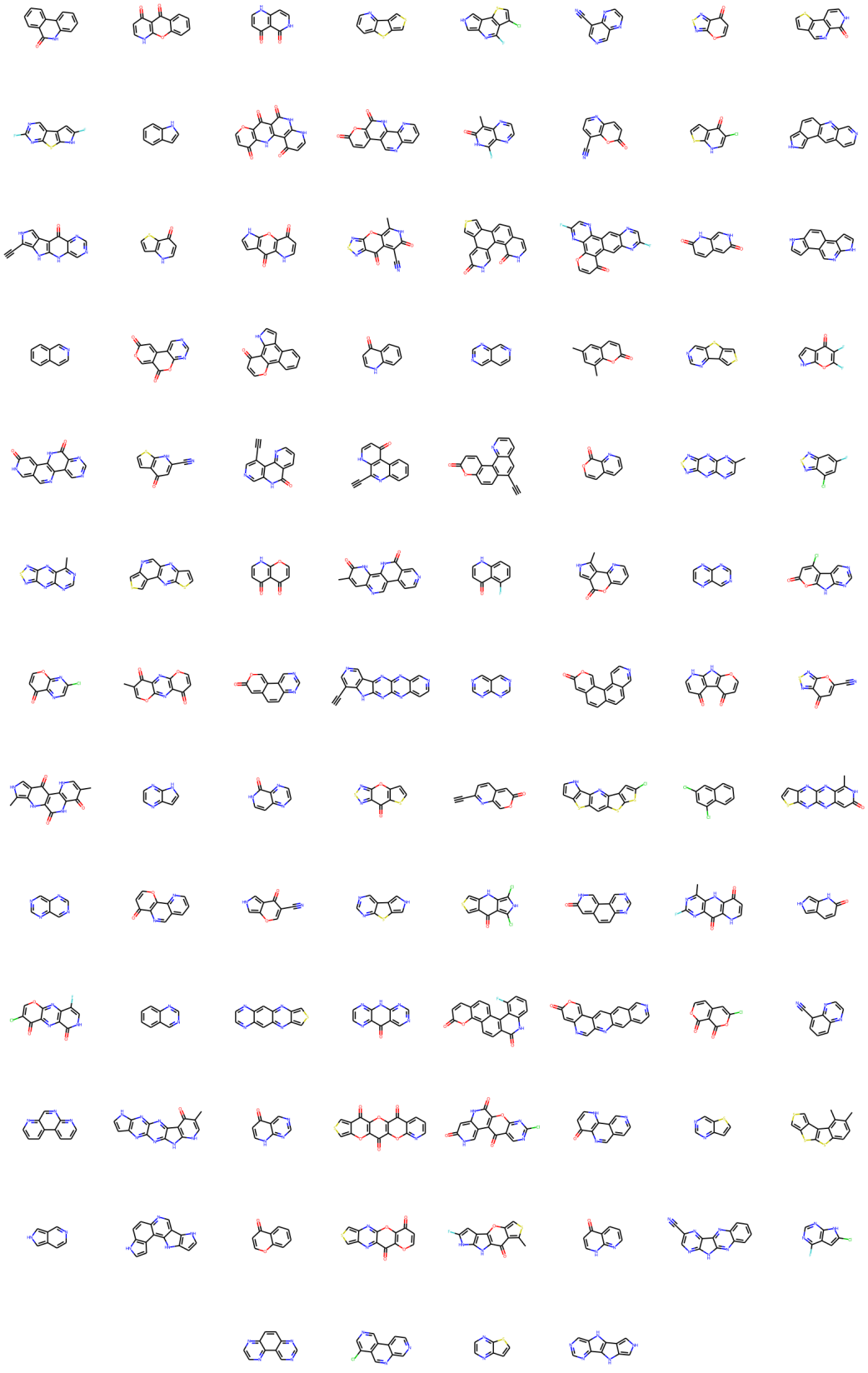


FIGURE A.1: 100 molecules generated for initial 'full' population 1

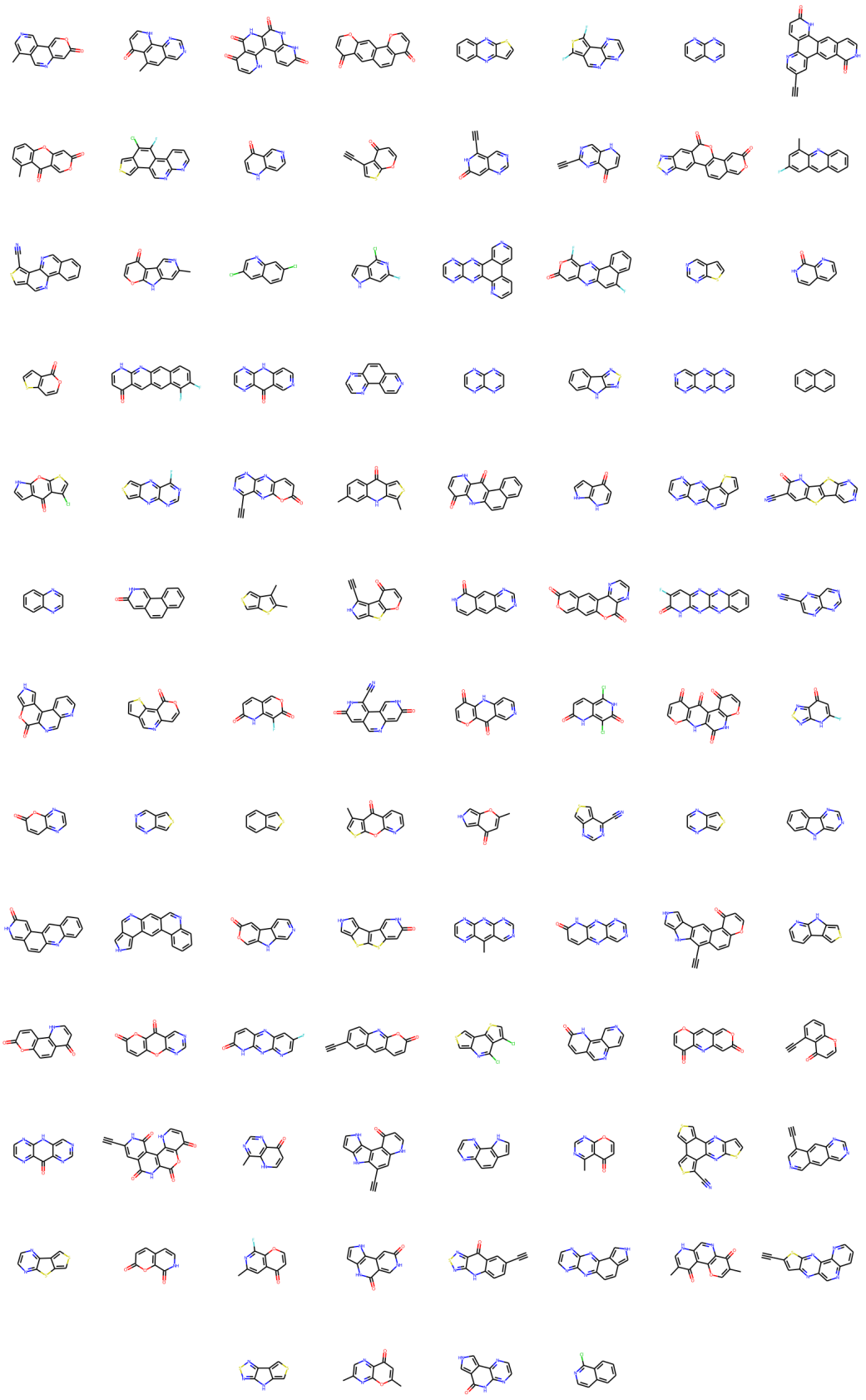


FIGURE A.2: 100 molecules generated for initial 'full' population 2

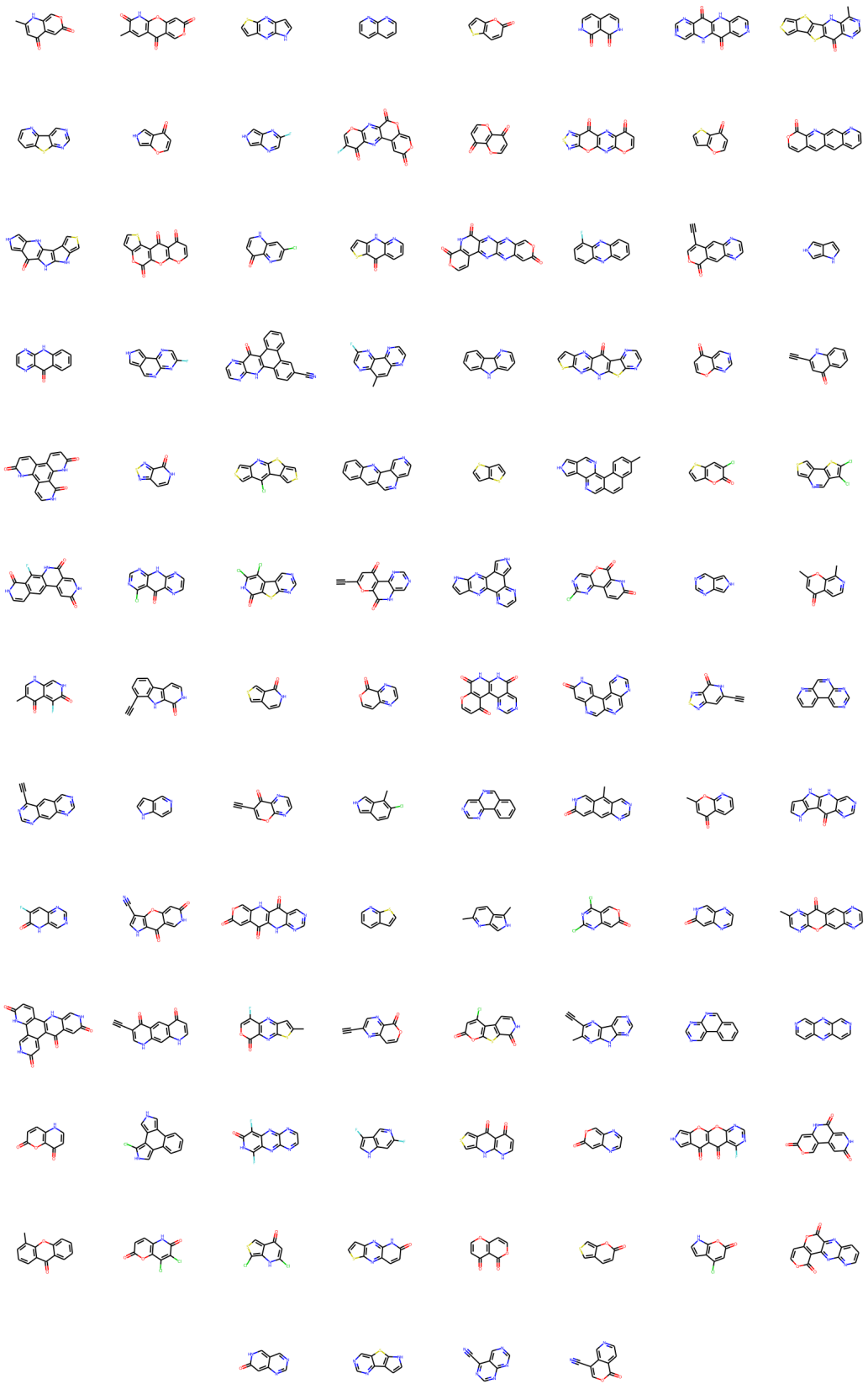


FIGURE A.3: 100 molecules generated for initial 'full' population 3

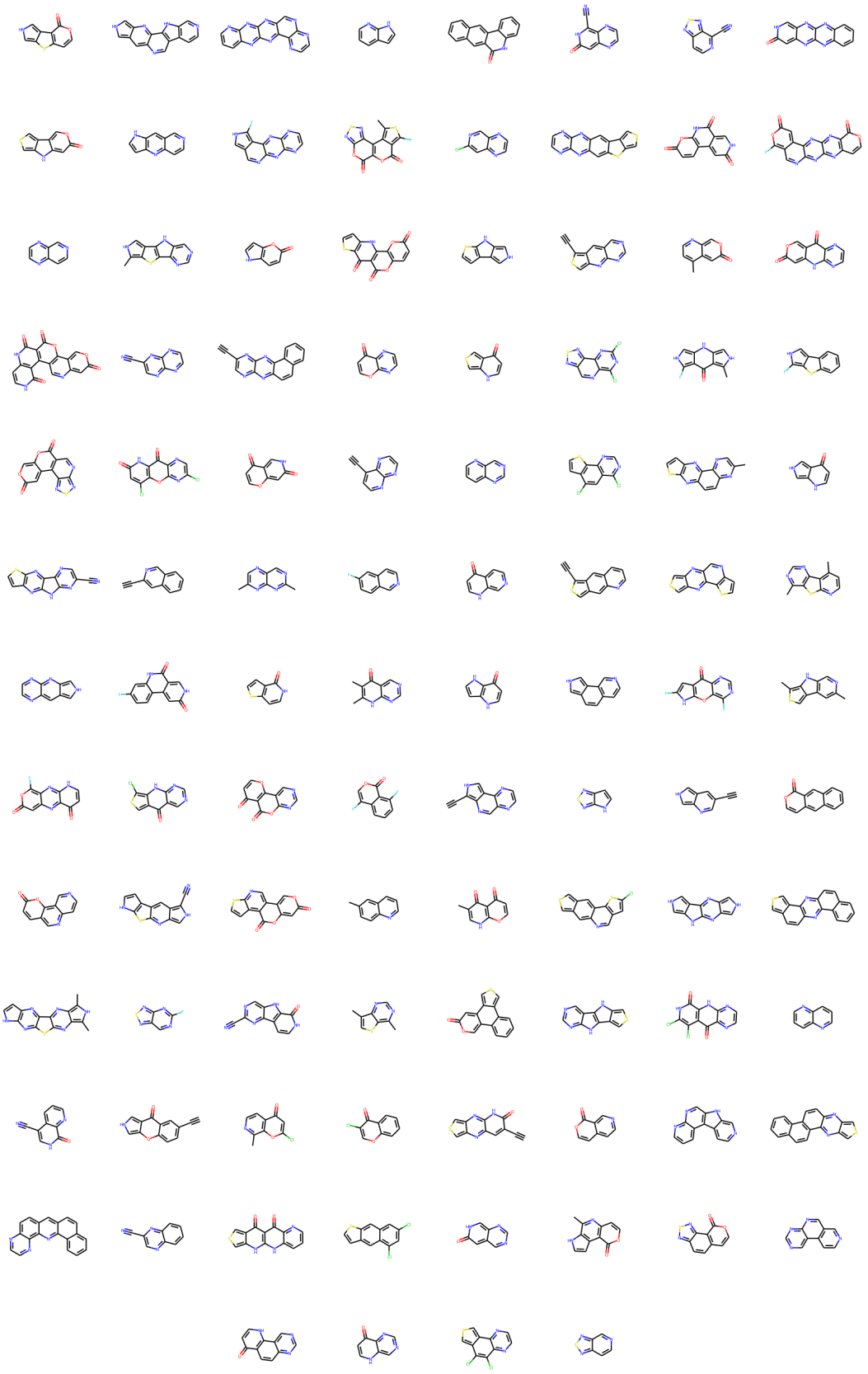


FIGURE A.4: 100 molecules generated for initial 'full' population 4

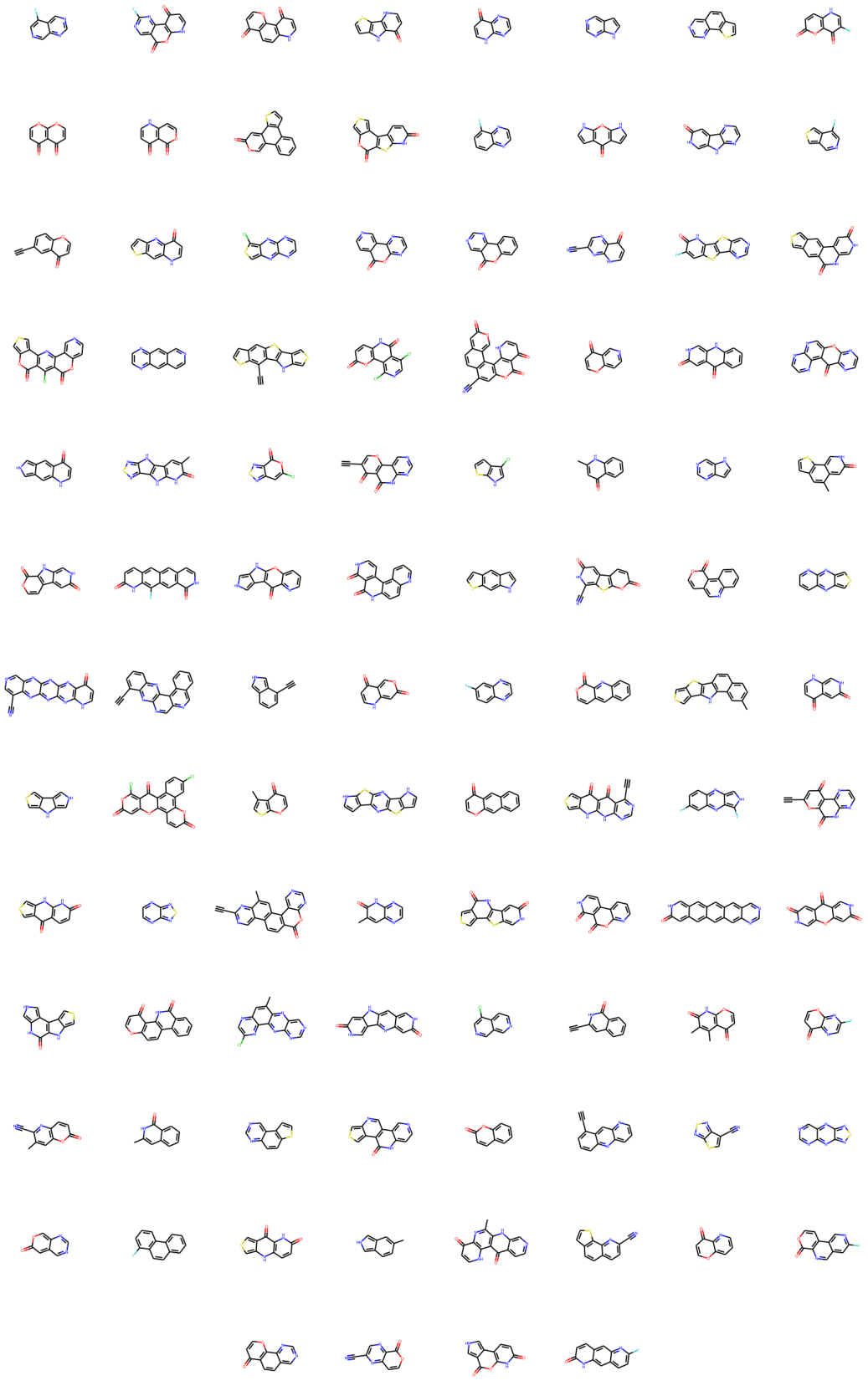


FIGURE A.5: 100 molecules generated for initial 'full' population 5

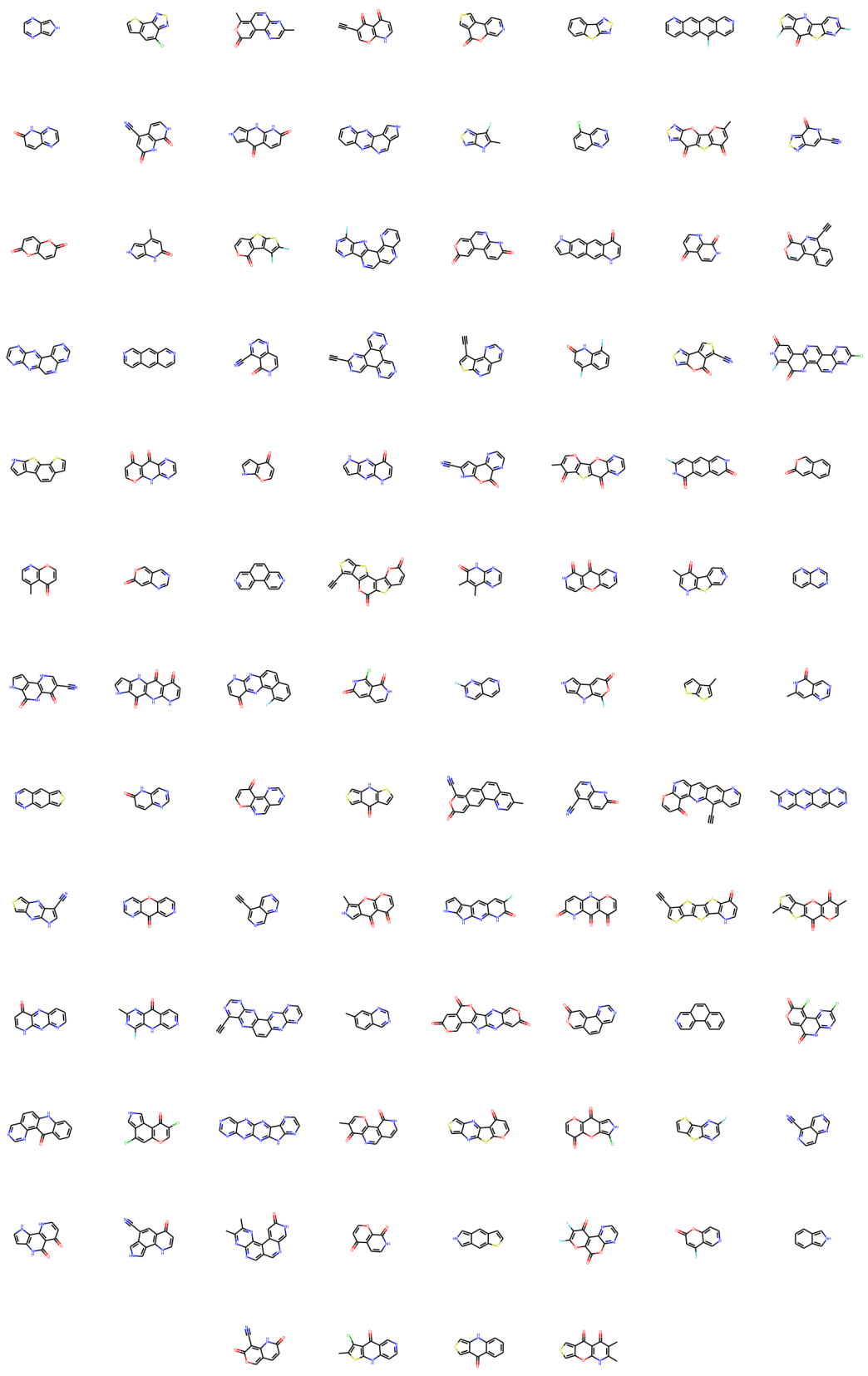


FIGURE A.6: 100 molecules generated for initial 'full' population 6

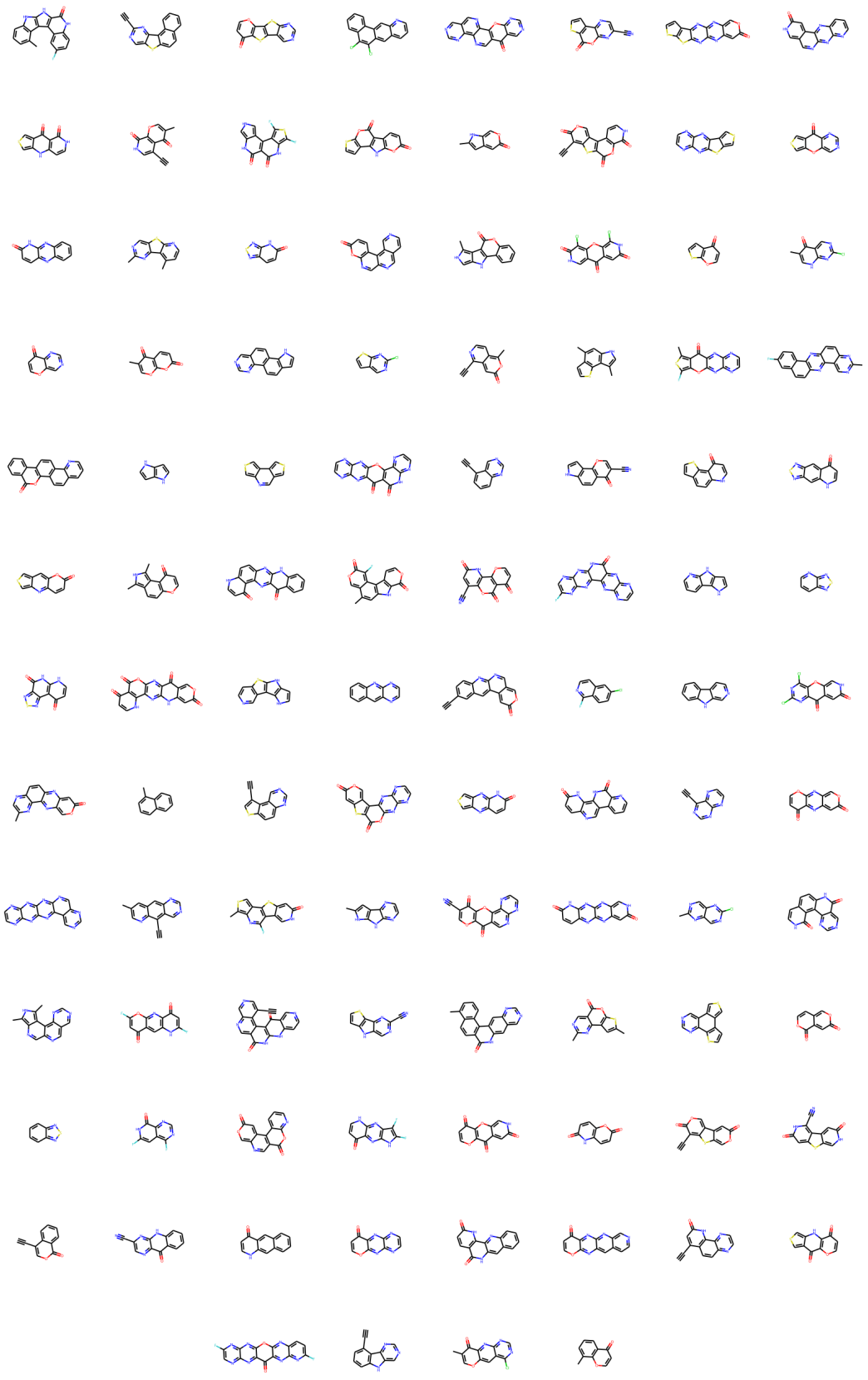


FIGURE A.7: 100 molecules generated for initial 'full' population 7

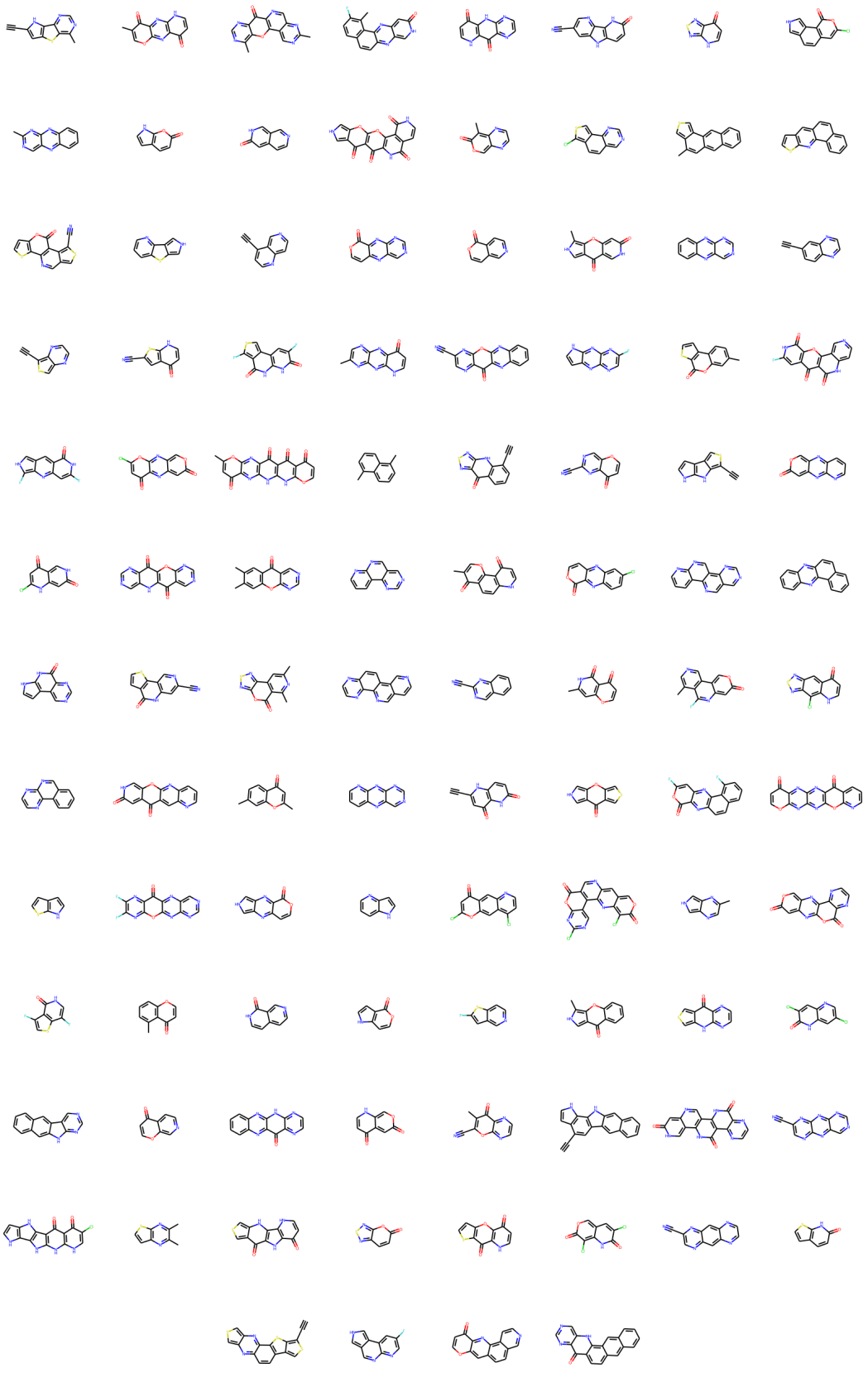


FIGURE A.8: 100 molecules generated for initial 'full' population 8

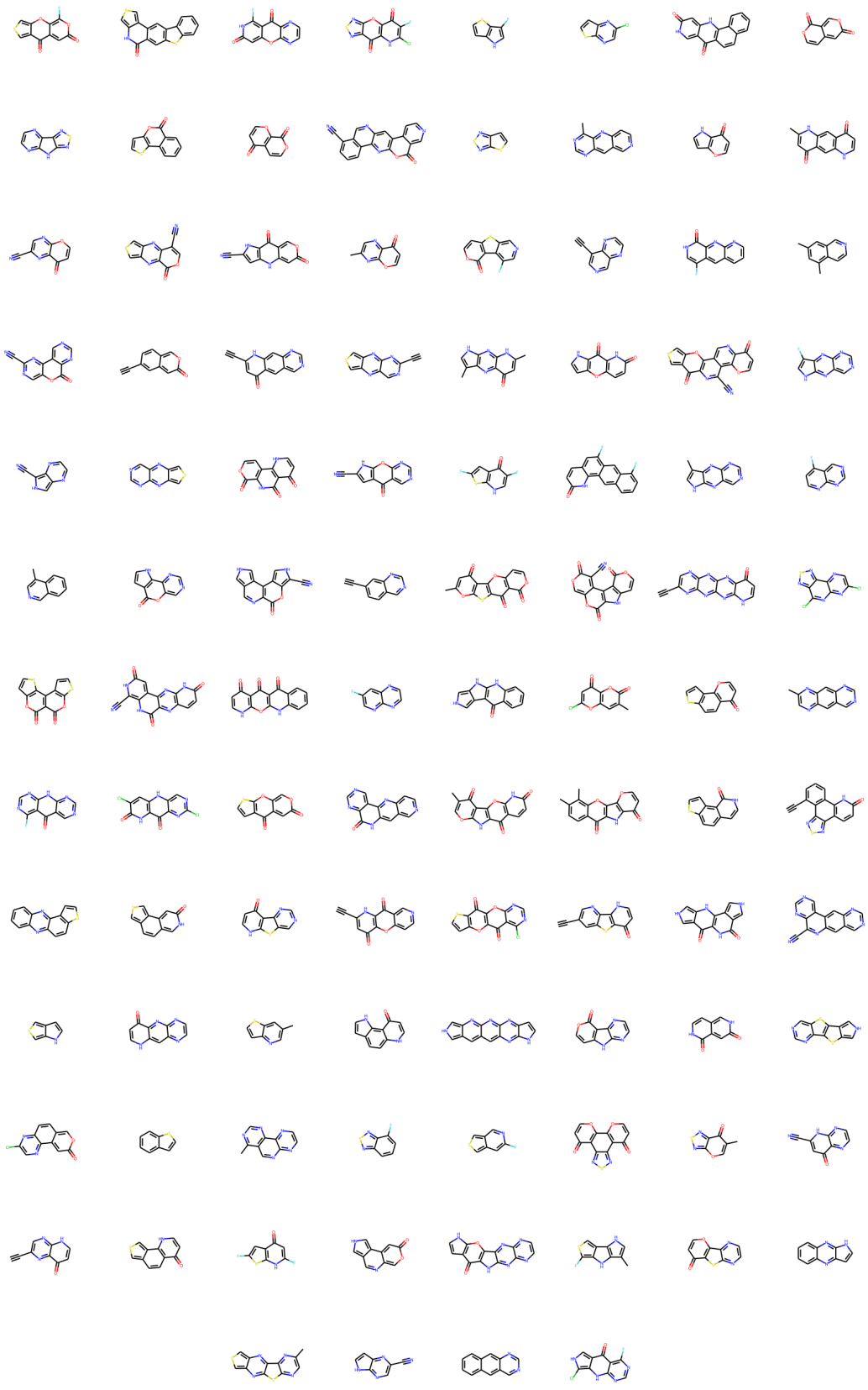


FIGURE A.9: 100 molecules generated for initial 'full' population 9

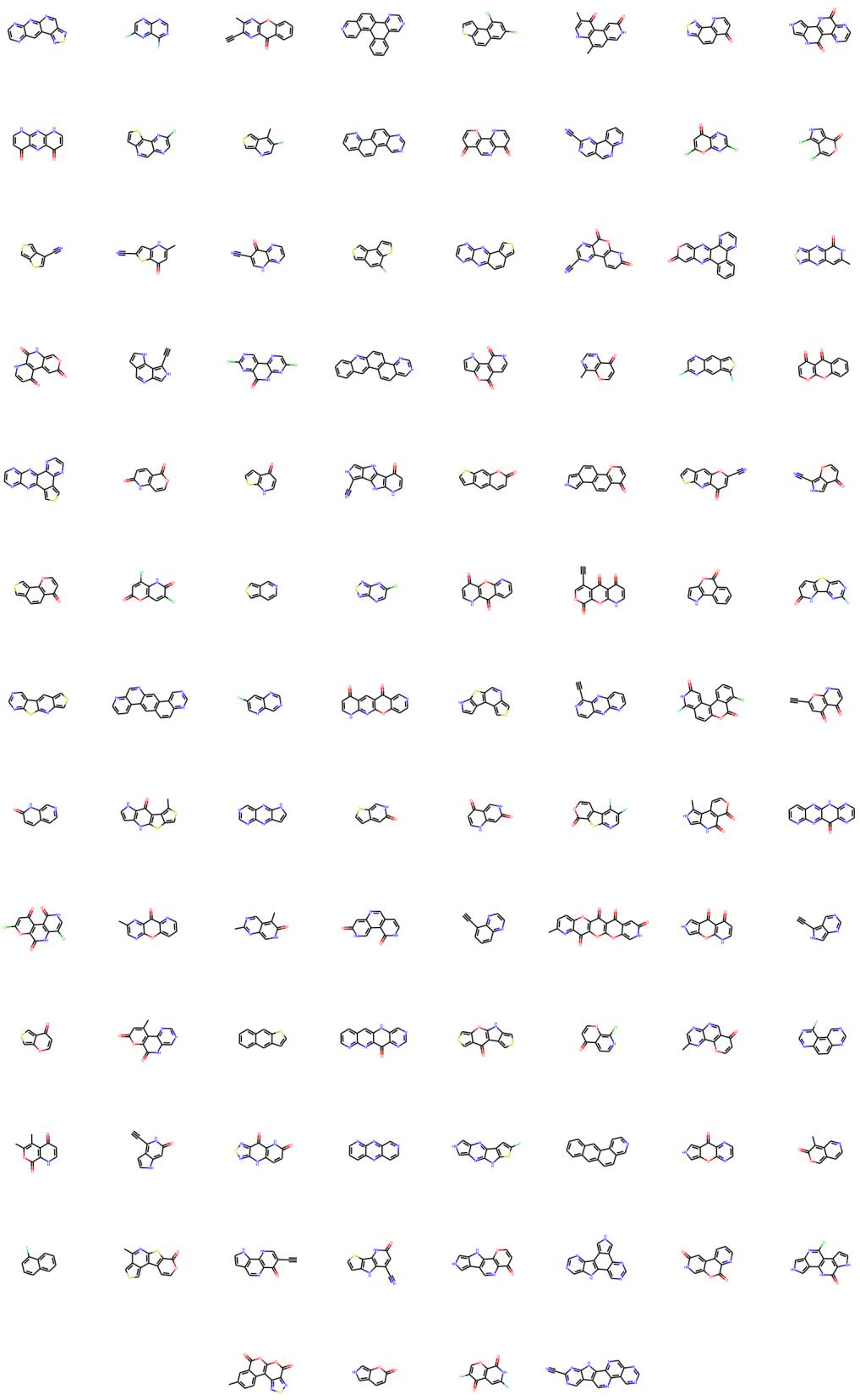


FIGURE A.10: 100 molecules generated for initial 'full' population 10

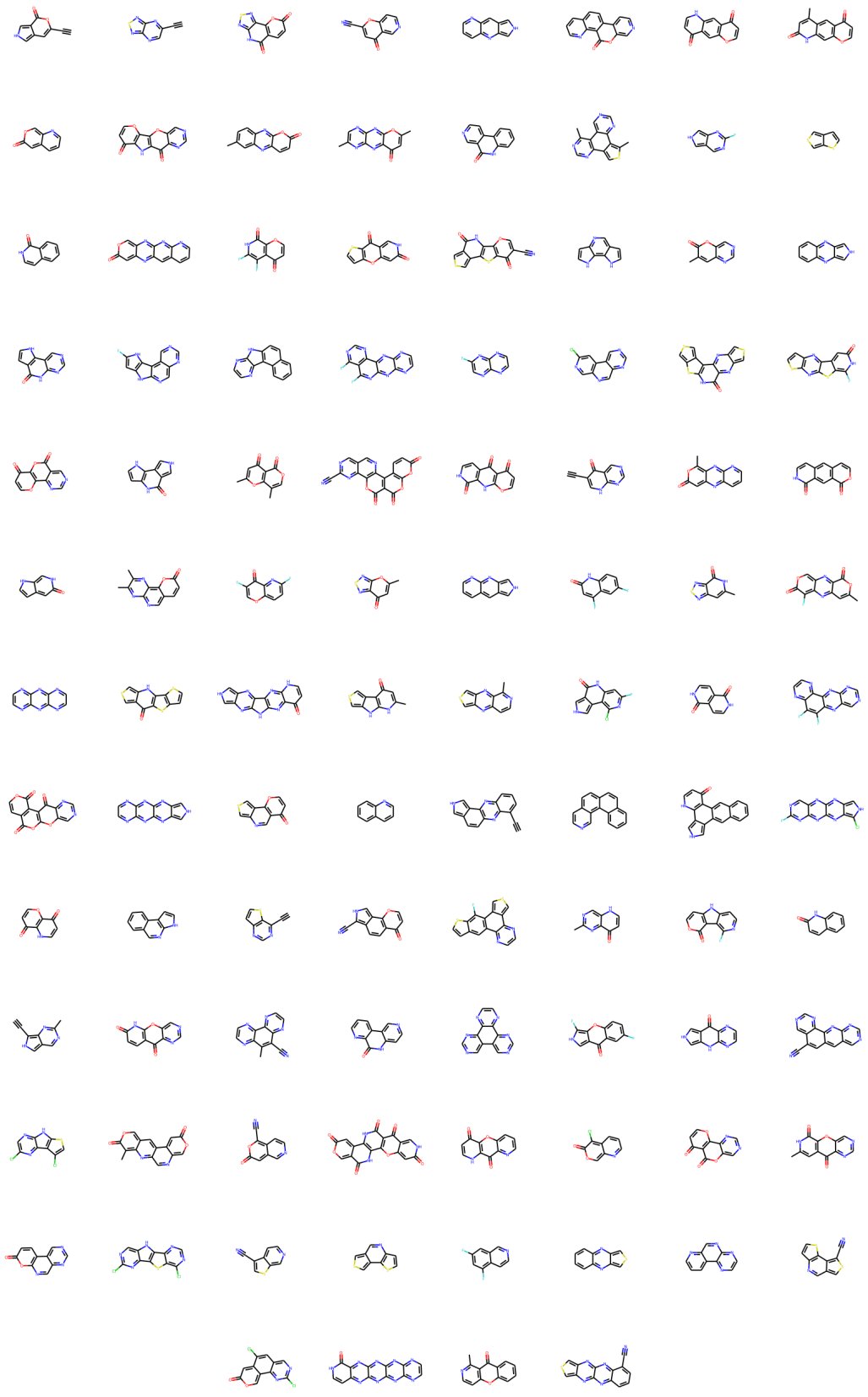


FIGURE A.11: 100 molecules generated for initial 'full' population 11

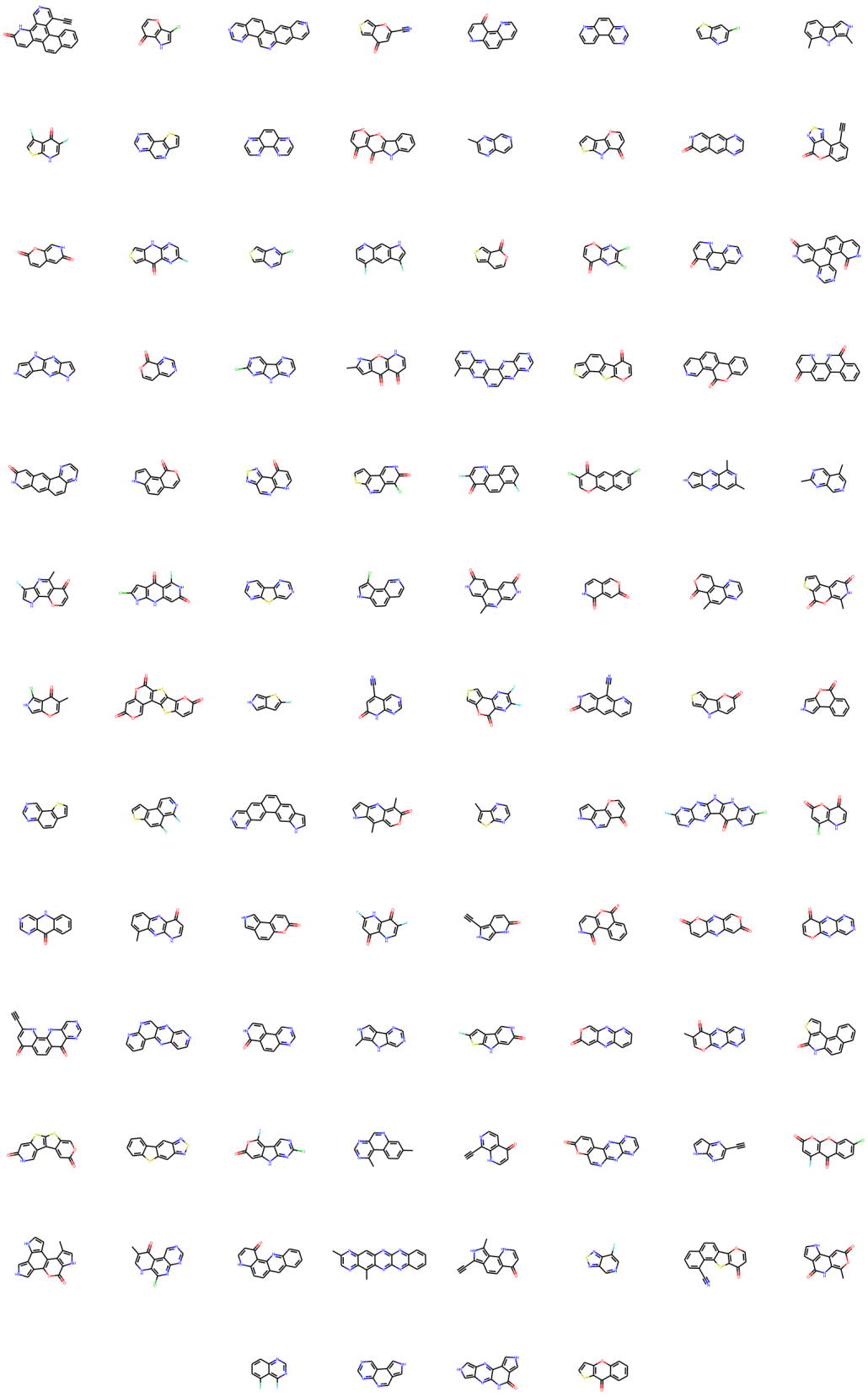


FIGURE A.12: 100 molecules generated for initial 'full' population 12

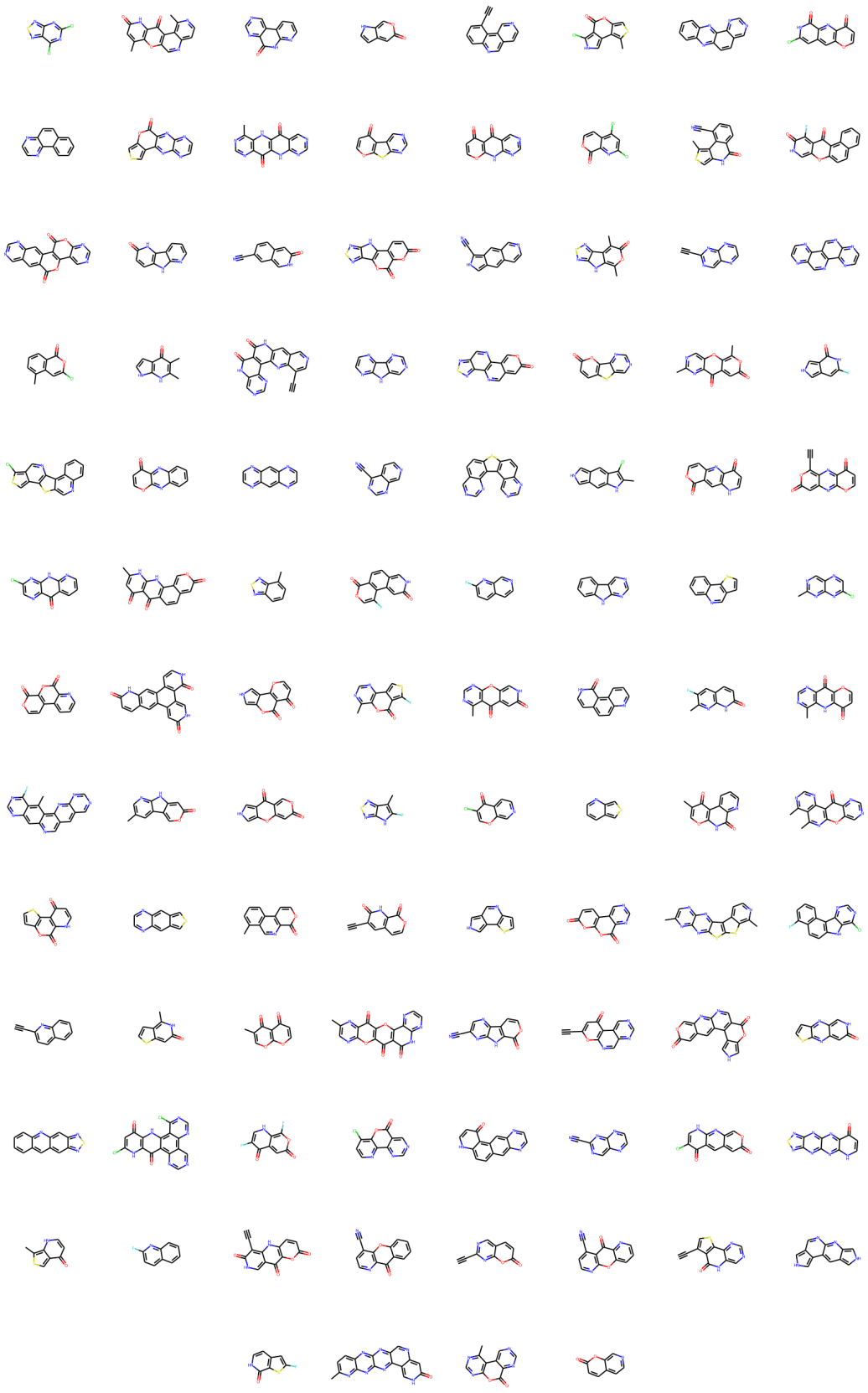


FIGURE A.13: 100 molecules generated for initial 'full' population 13

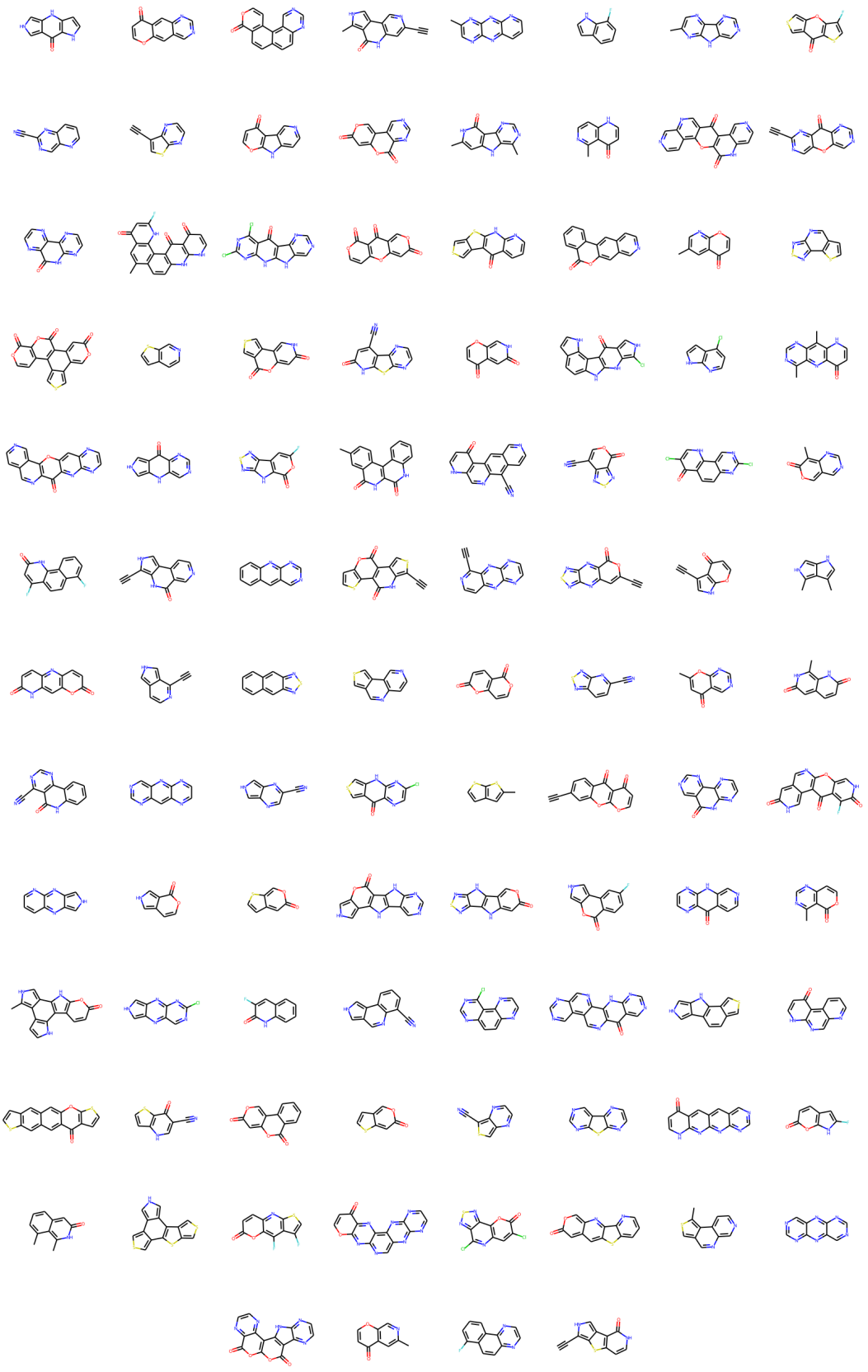


FIGURE A.14: 100 molecules generated for initial 'full' population 14

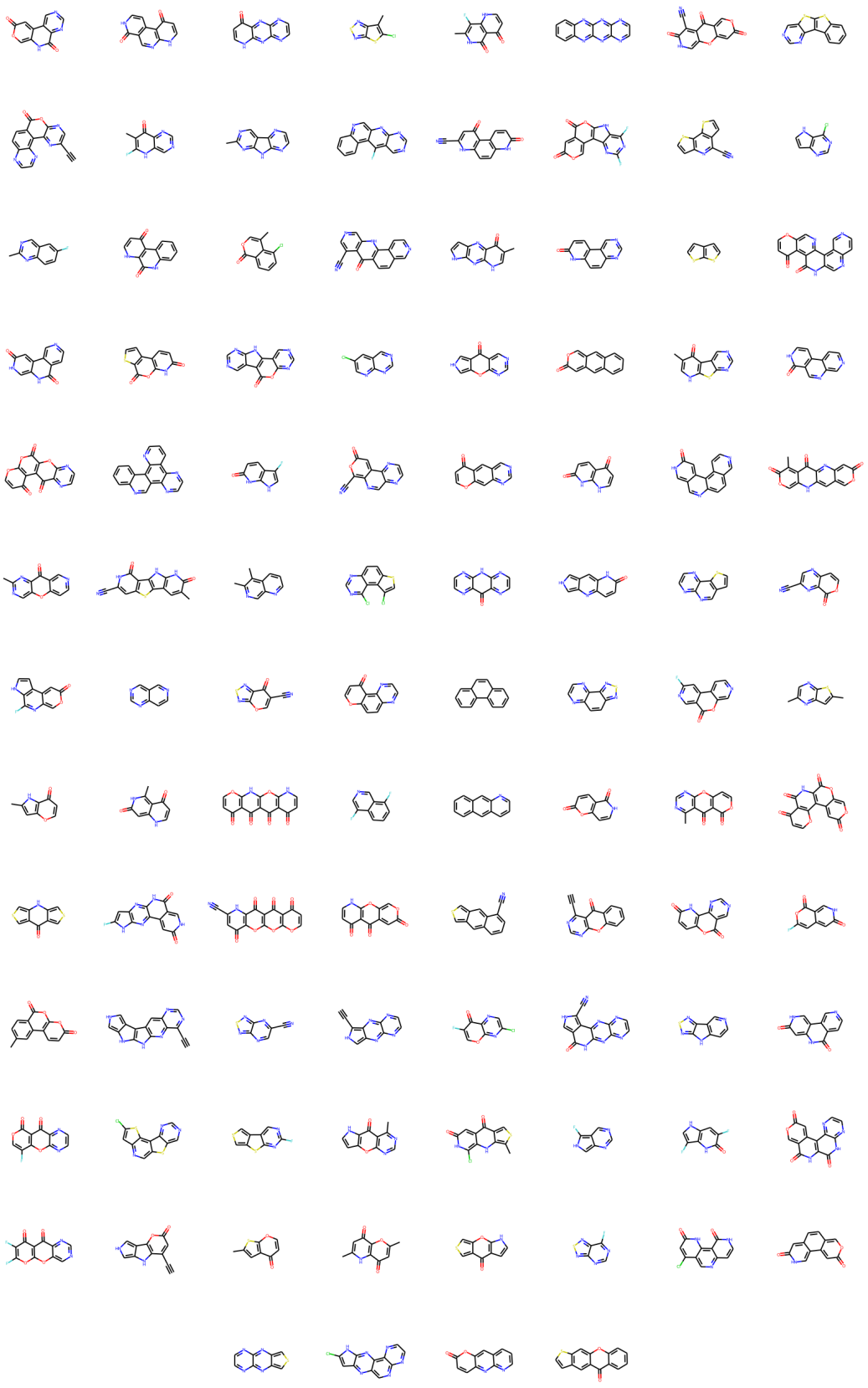


FIGURE A.15: 100 molecules generated for initial 'full' population 15

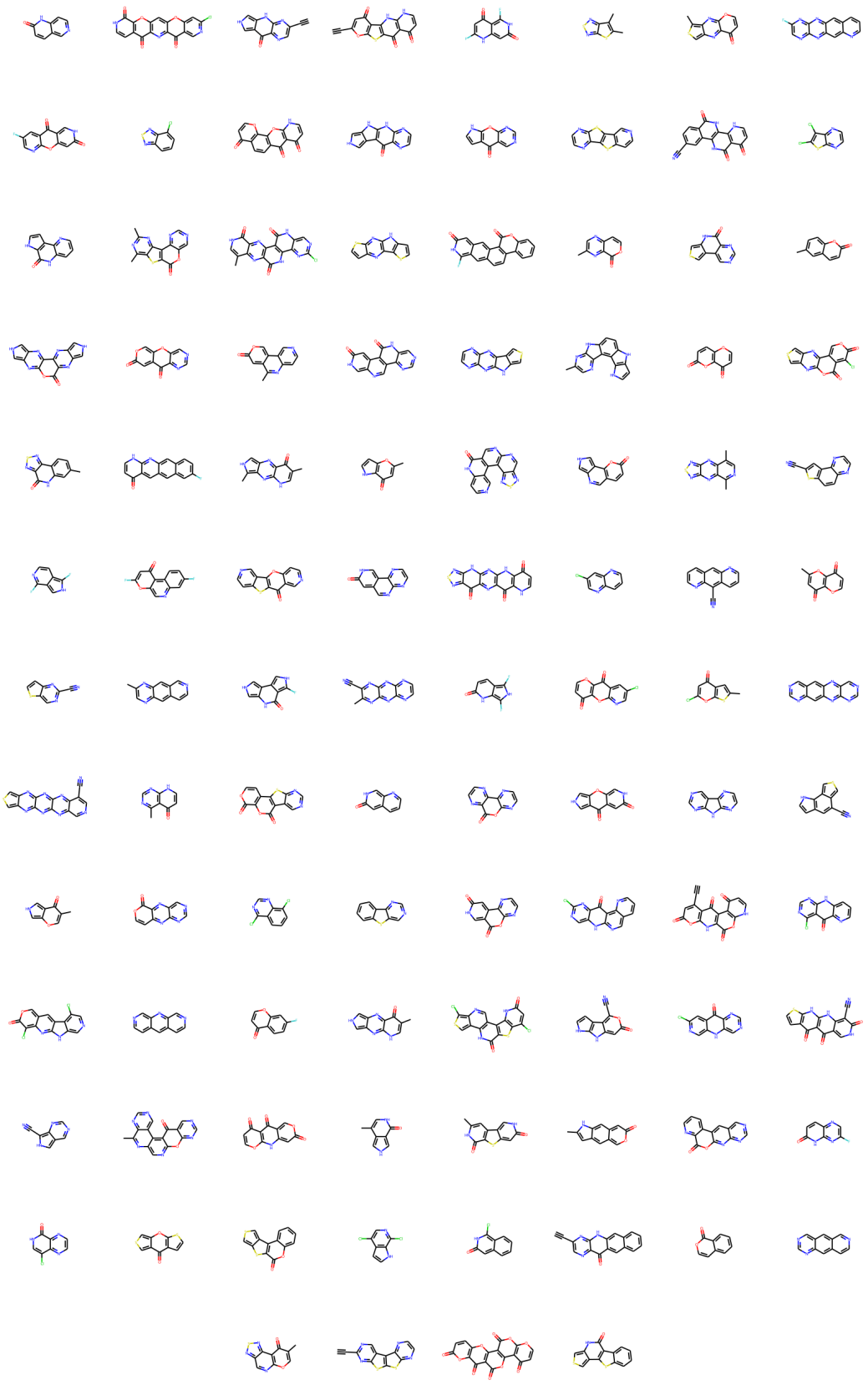


FIGURE A.16: 100 molecules generated for initial 'full' population 16

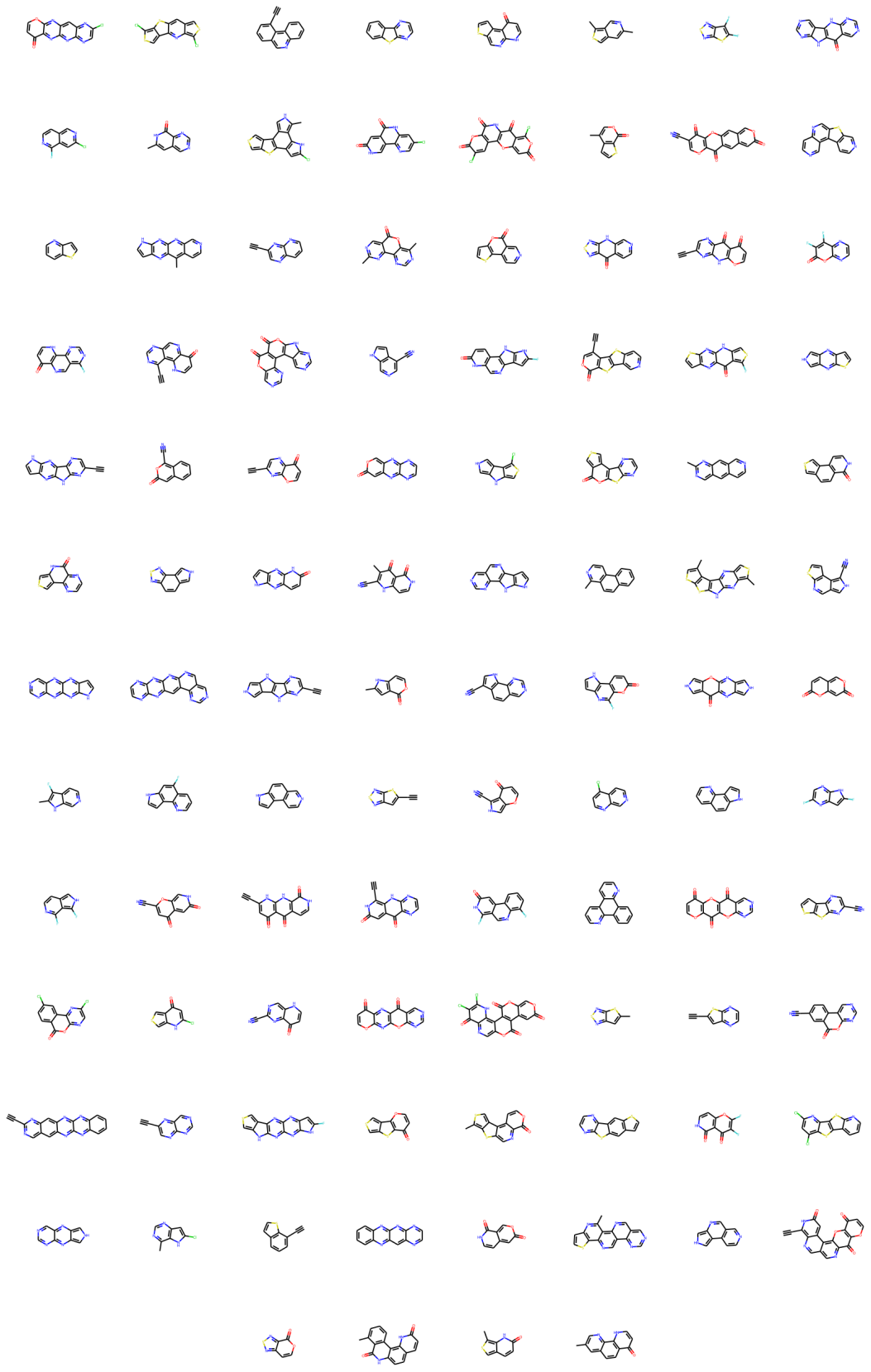


FIGURE A.17: 100 molecules generated for initial 'full' population 17

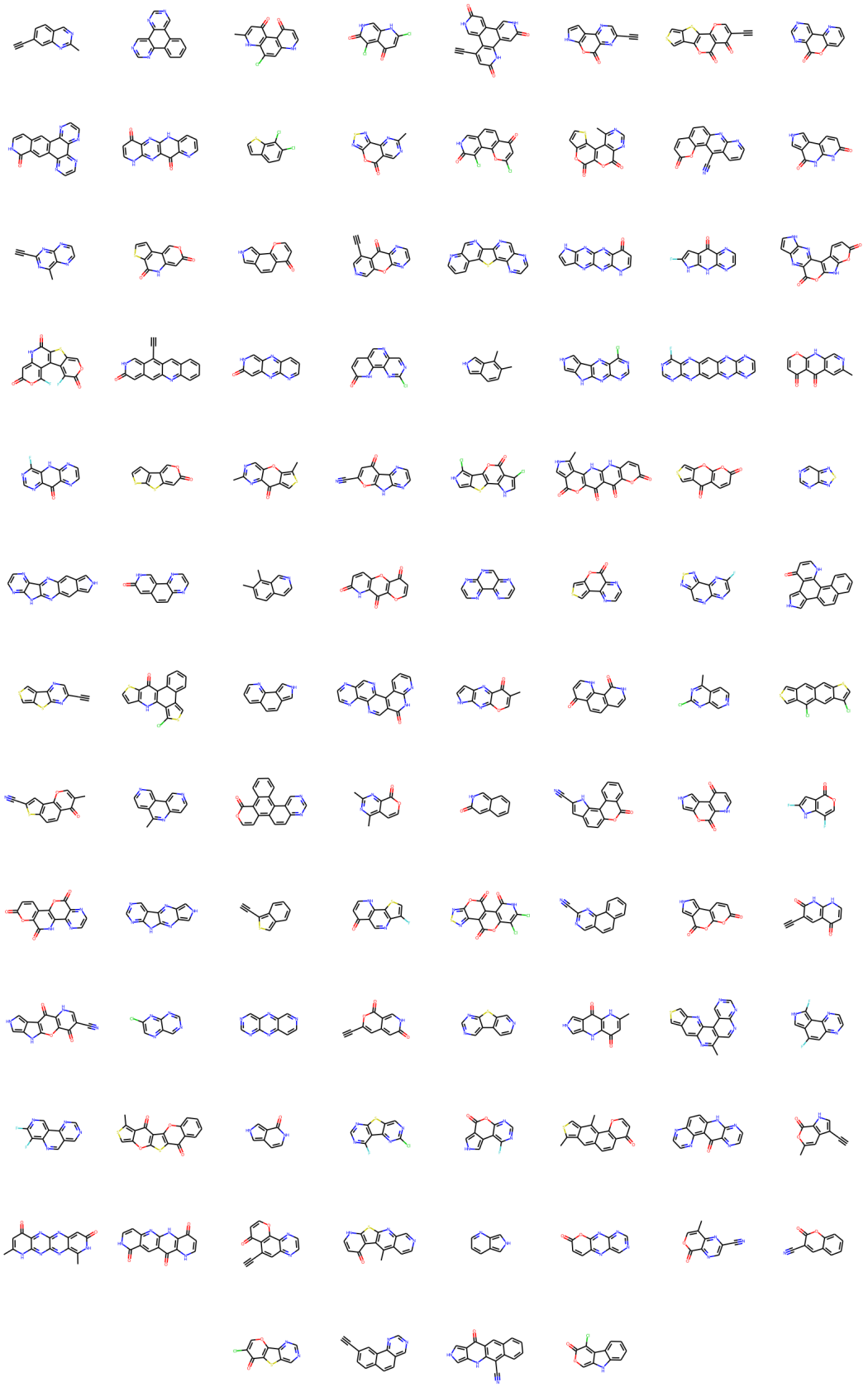


FIGURE A.18: 100 molecules generated for initial 'full' population 18

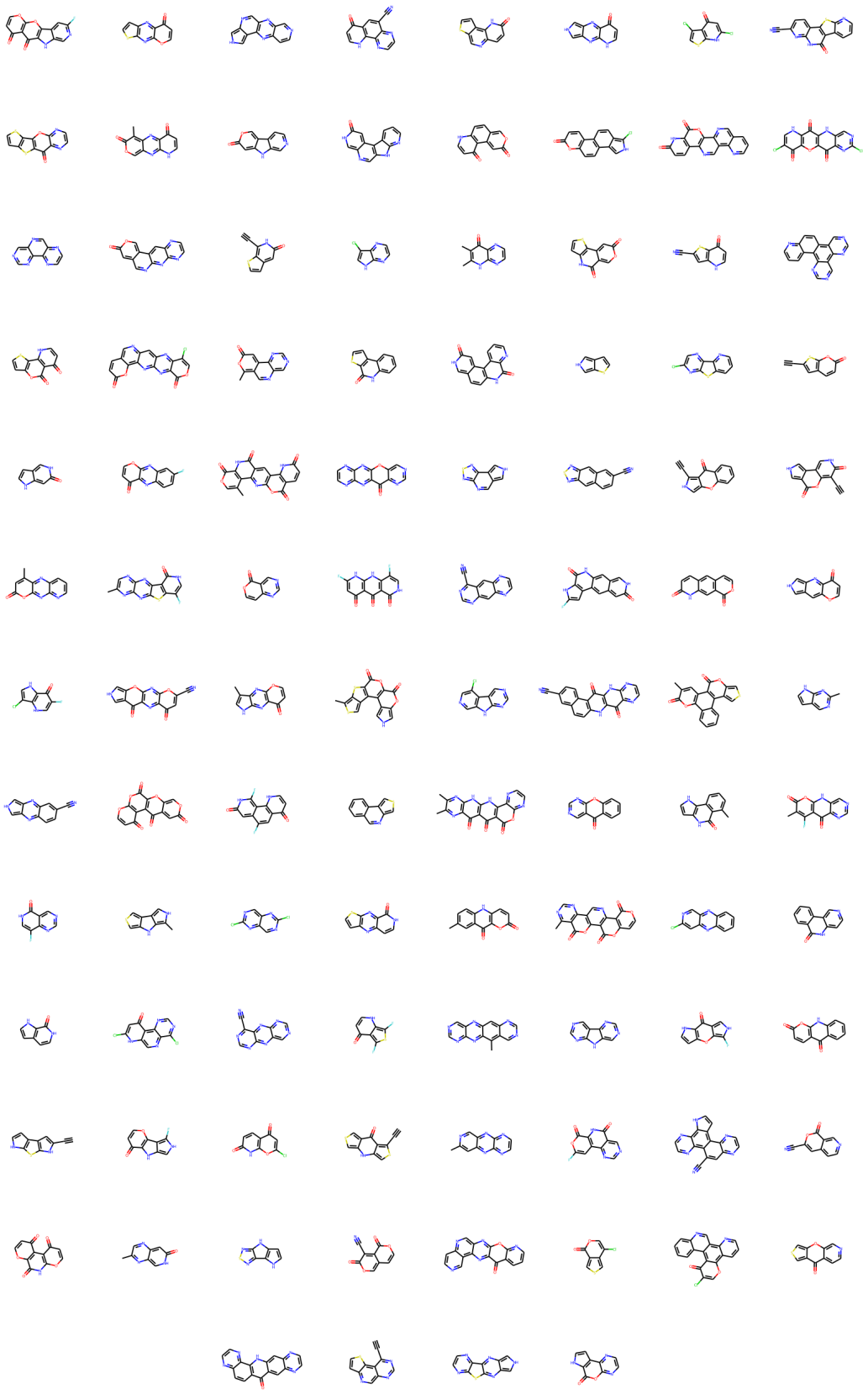


FIGURE A.19: 100 molecules generated for initial 'full' population 19

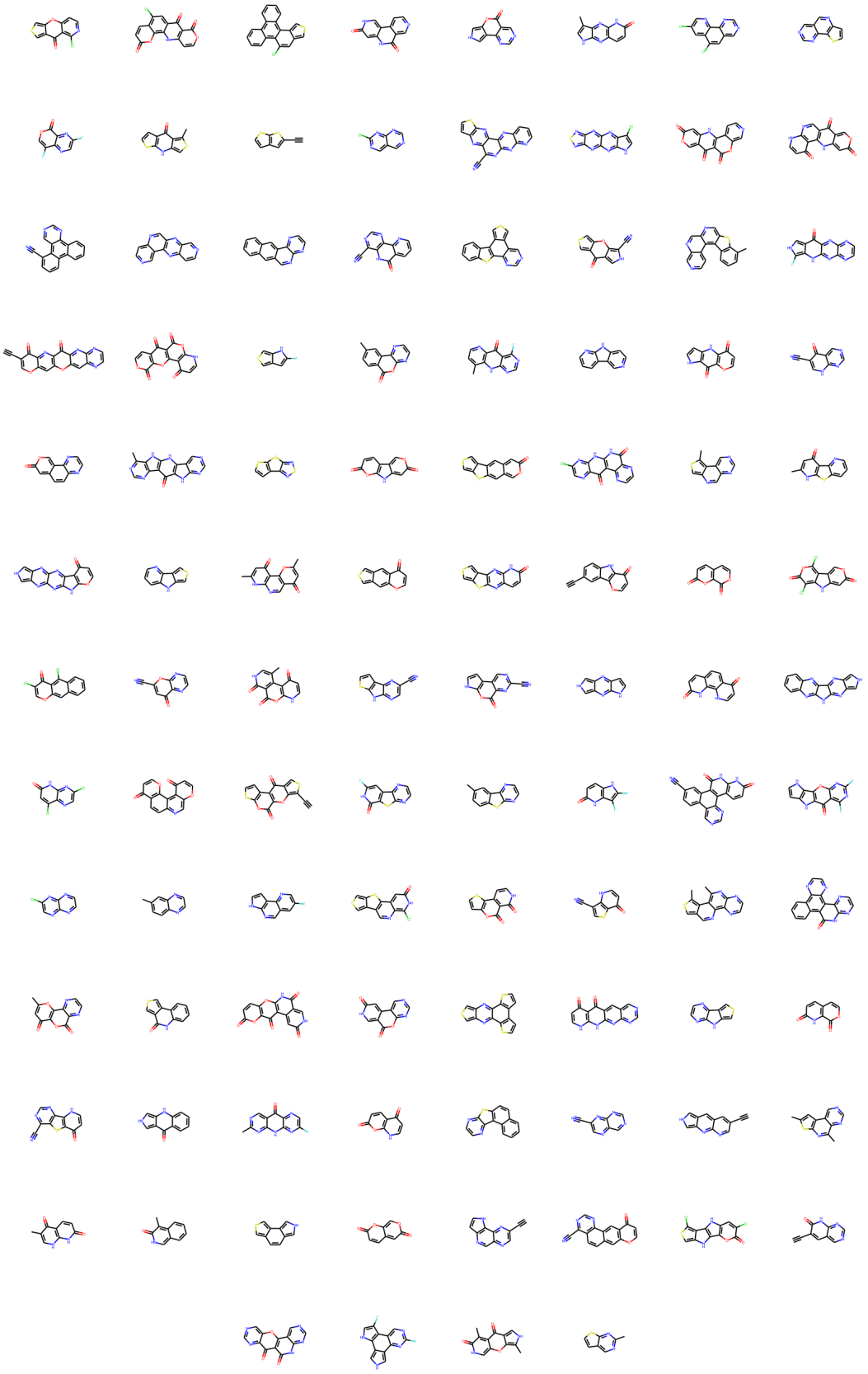


FIGURE A.20: 100 molecules generated for initial 'full' population 20

### A.1.2 Reduced building block sets

16 unique populations of molecules, generated using the reduced sets of OCELOT building blocks.

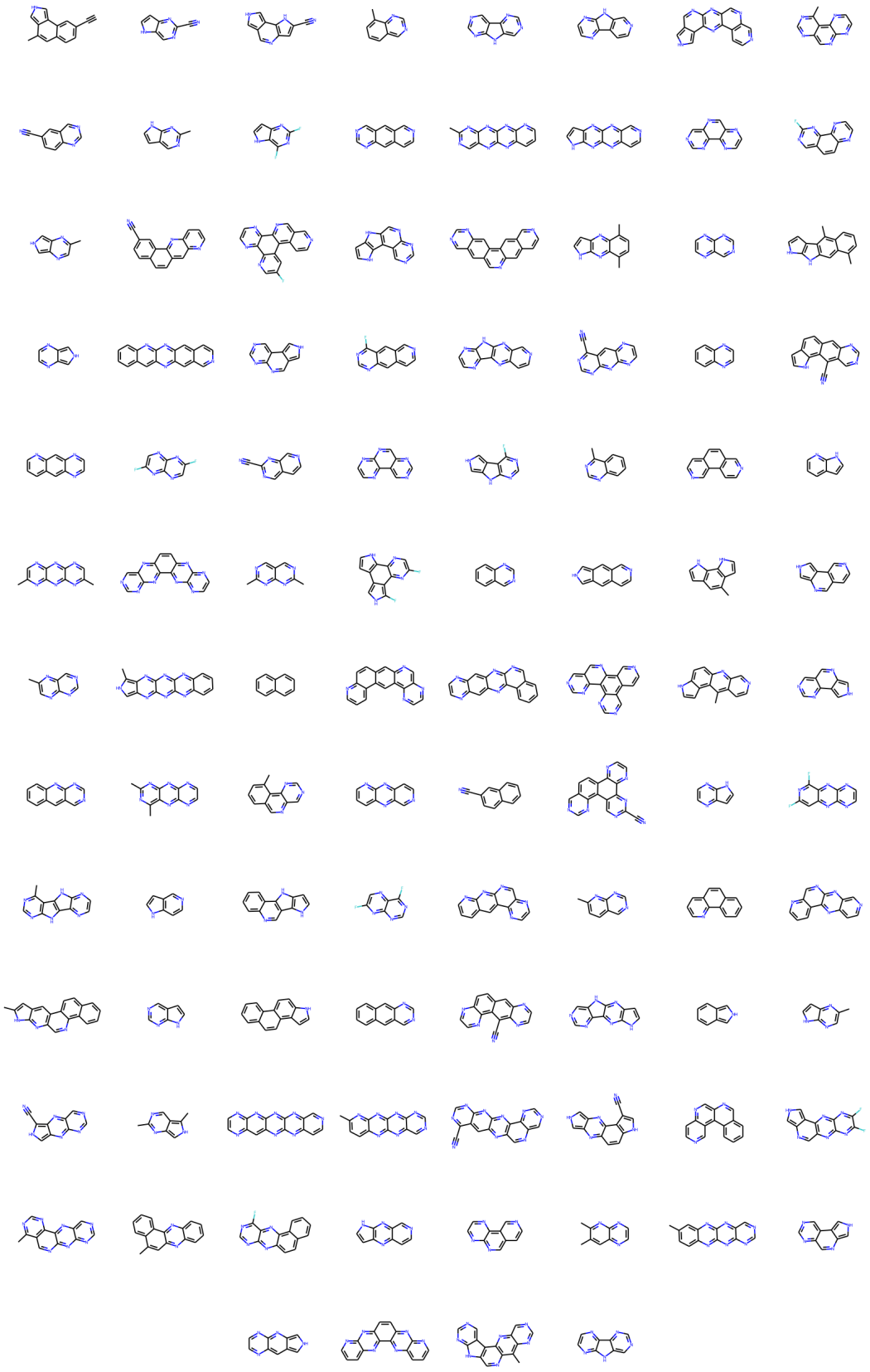


FIGURE A.21: 100 molecules generated for initial 'reduced' population 1

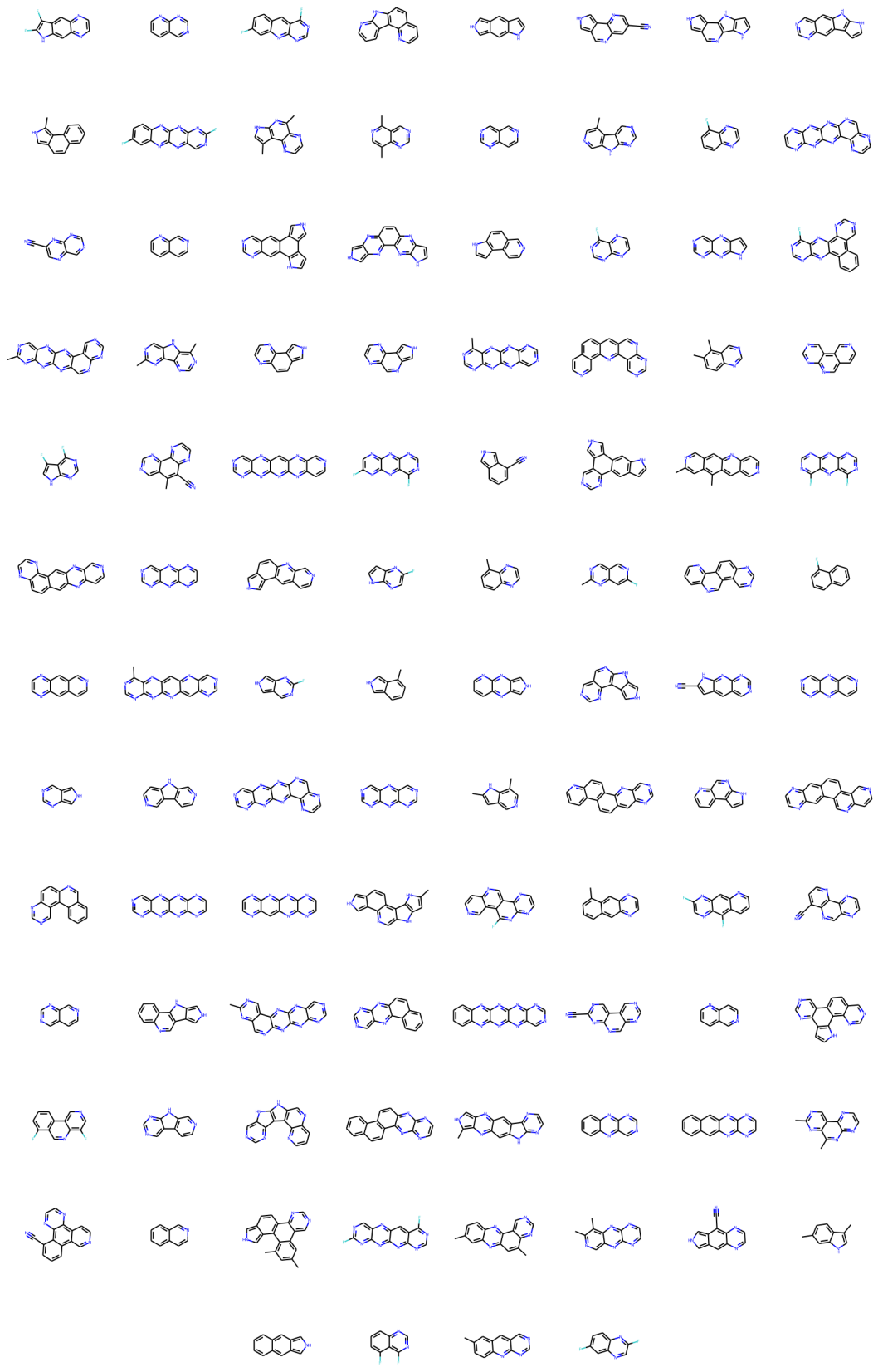


FIGURE A.22: 100 molecules generated for initial 'reduced' population 2

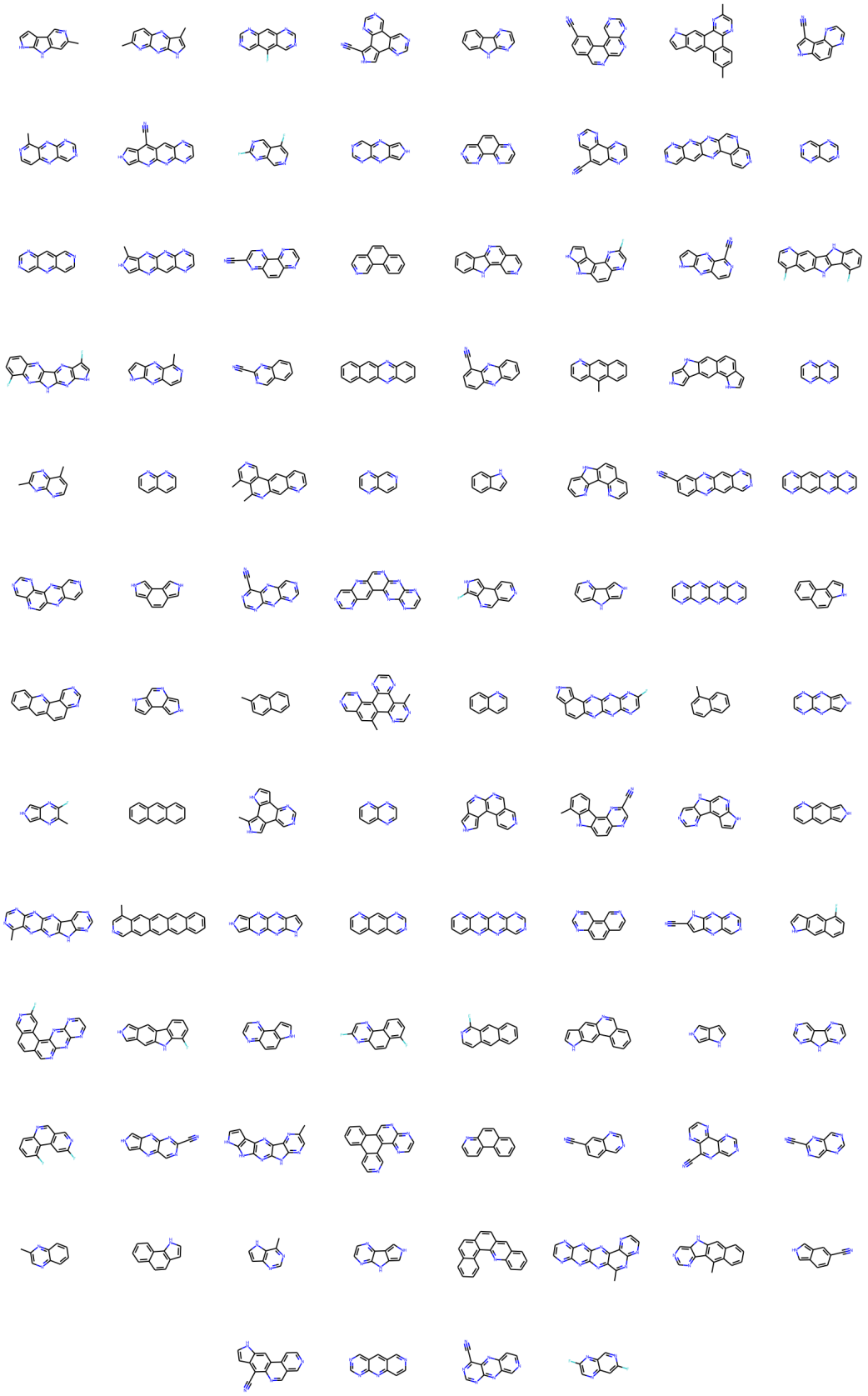


FIGURE A.23: 100 molecules generated for initial 'reduced' population 3

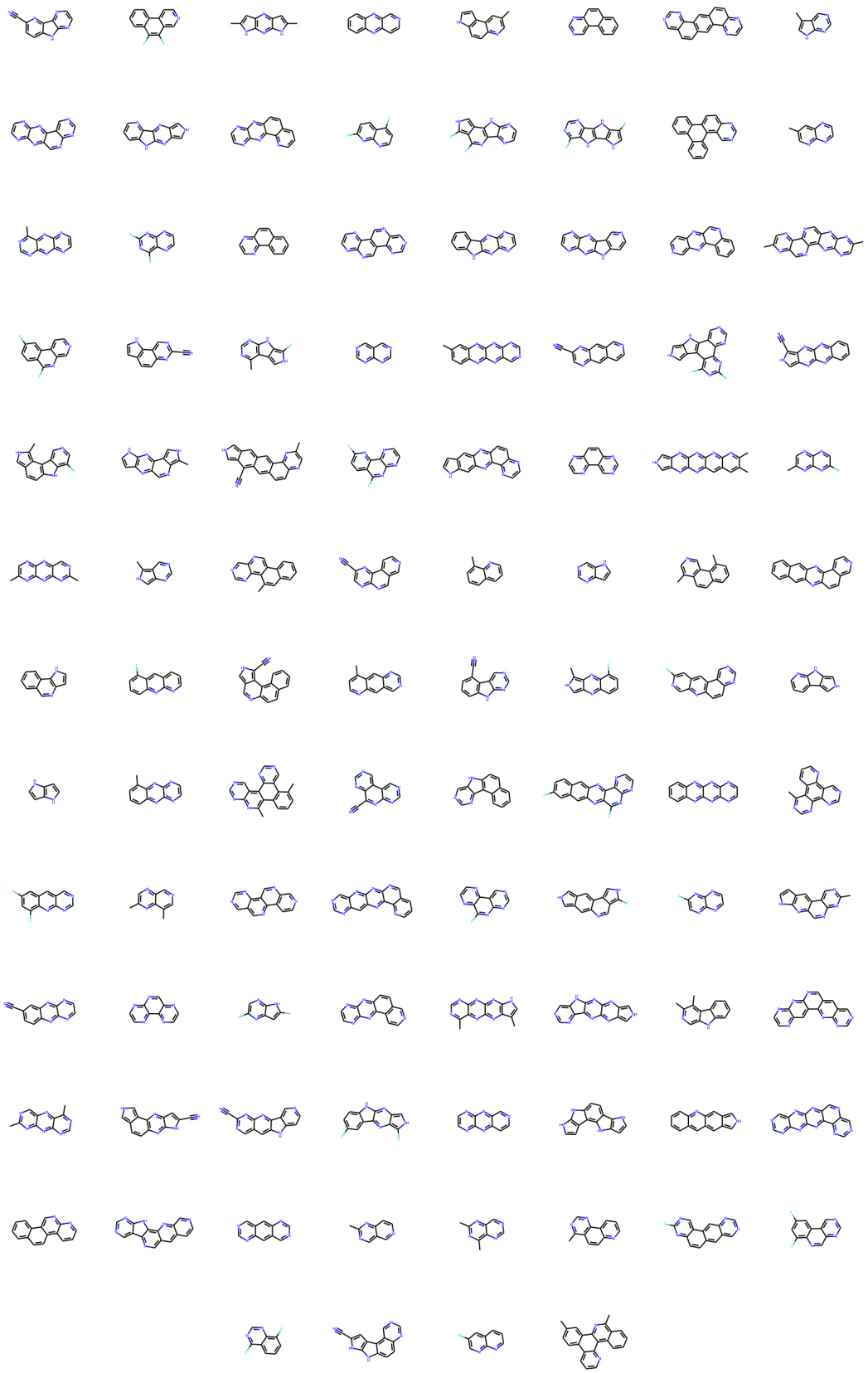


FIGURE A.24: 100 molecules generated for initial 'reduced' population 4

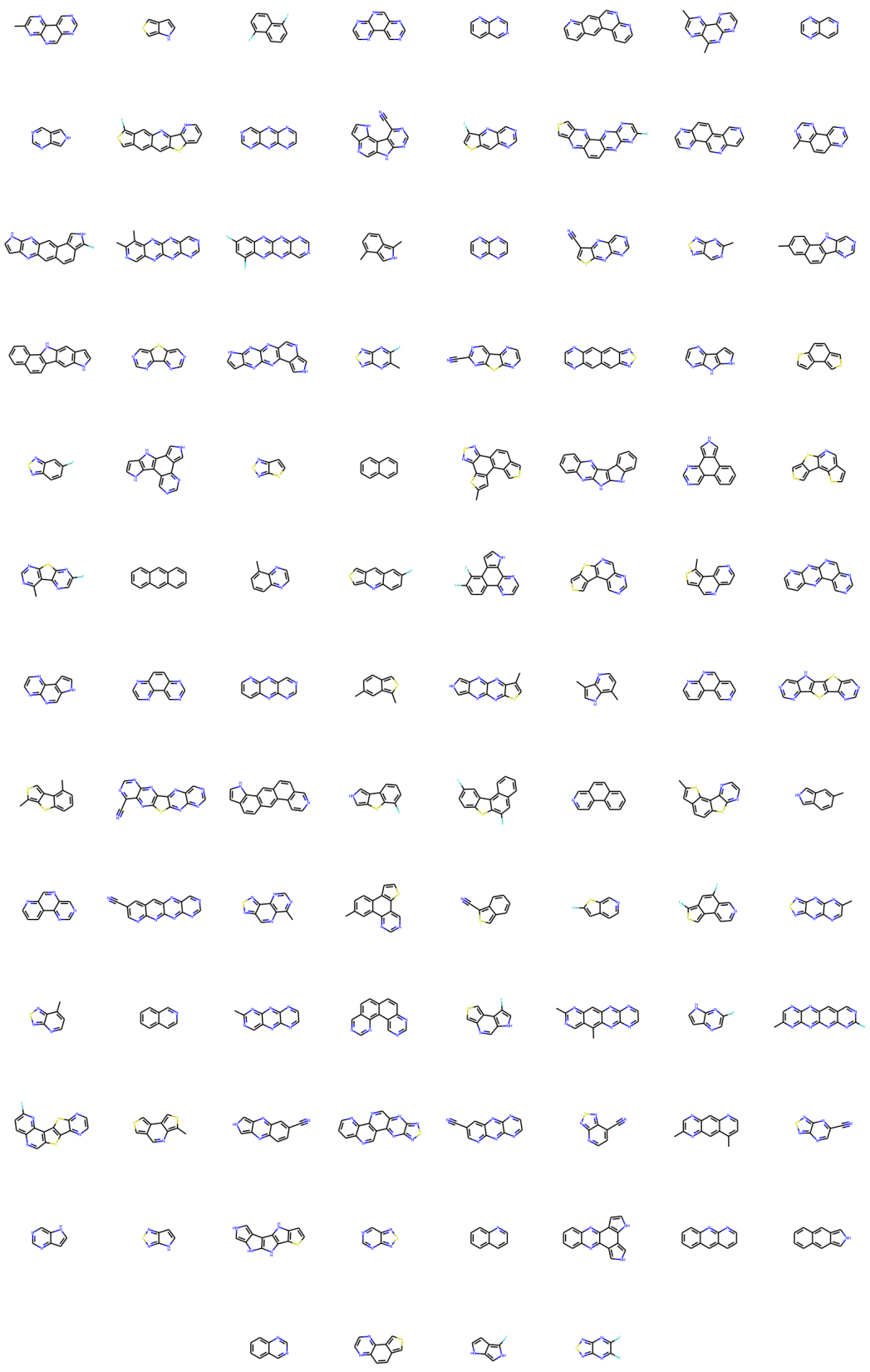


FIGURE A.25: 100 molecules generated for initial 'reduced' population 5

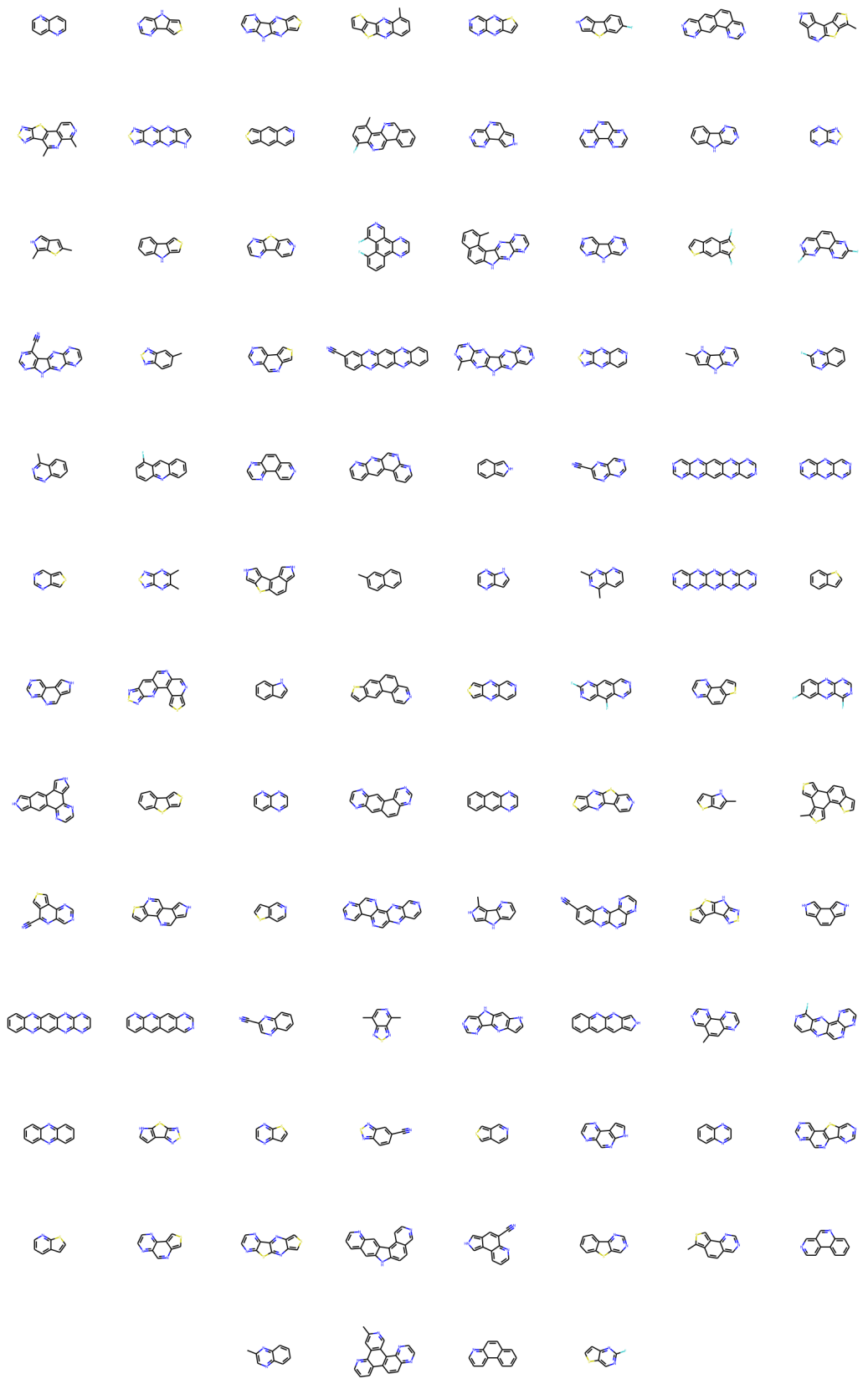


FIGURE A.26: 100 molecules generated for initial 'reduced' population 6

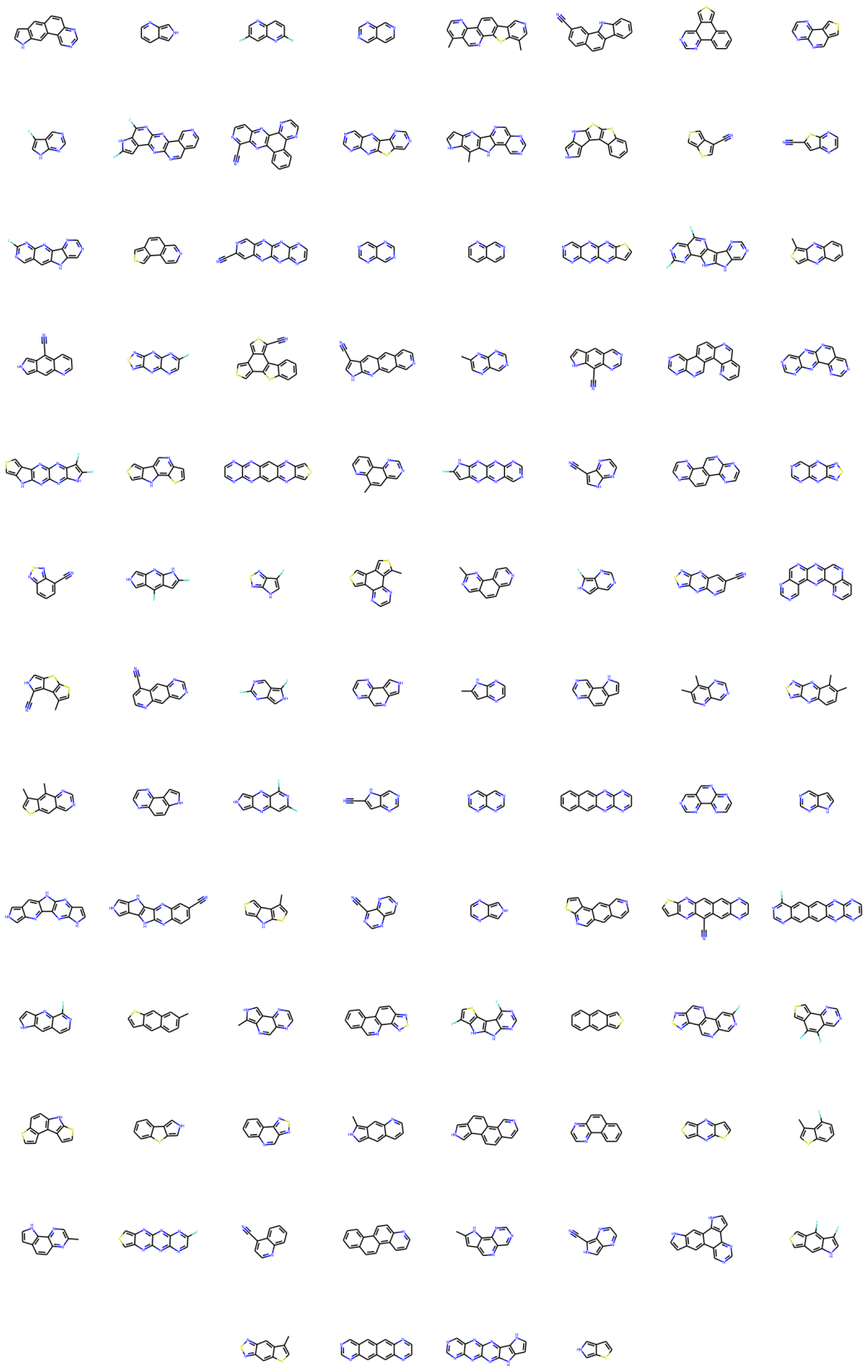


FIGURE A.27: 100 molecules generated for initial 'reduced' population 7

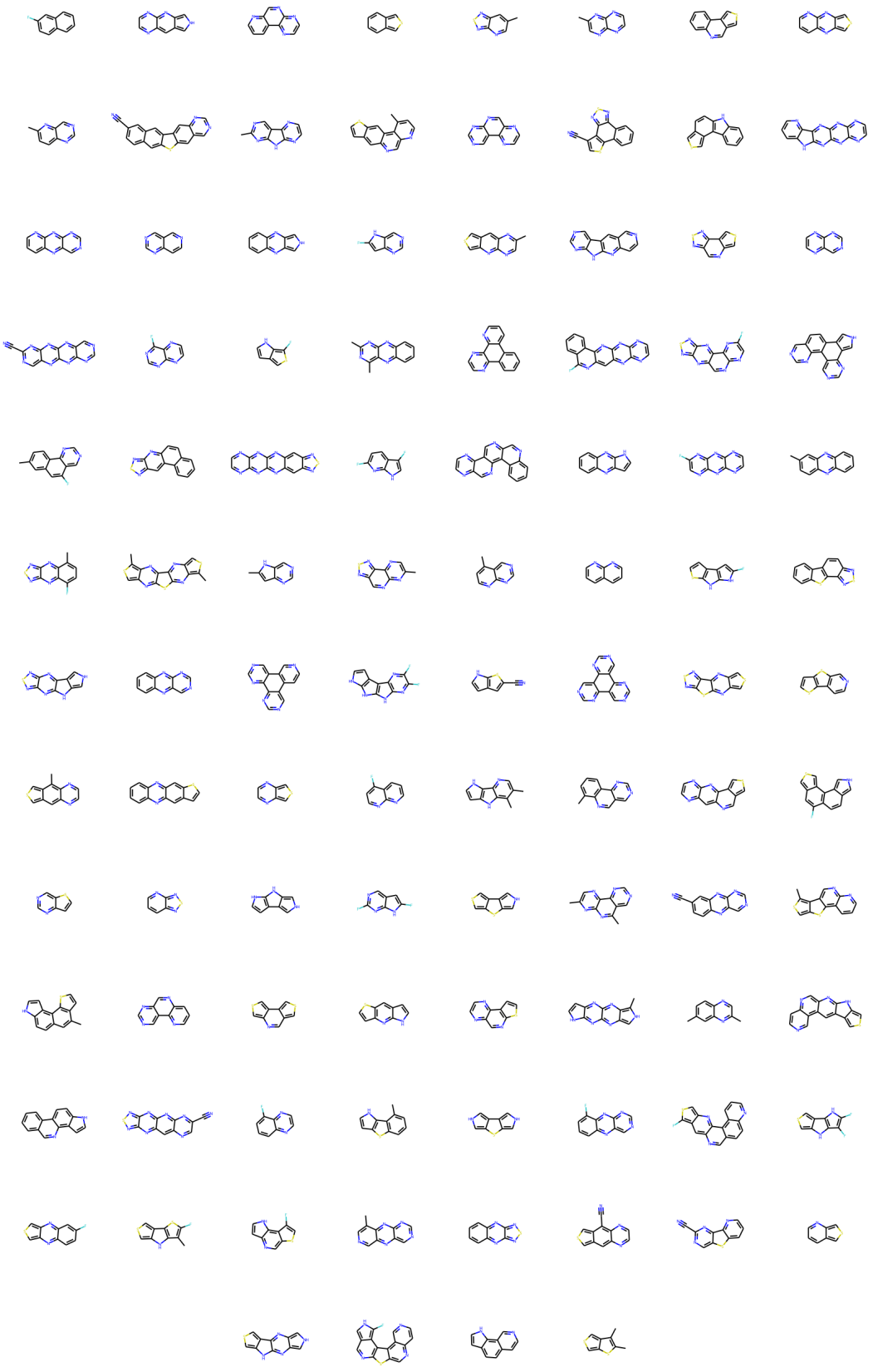


FIGURE A.28: 100 molecules generated for initial 'reduced' population 8

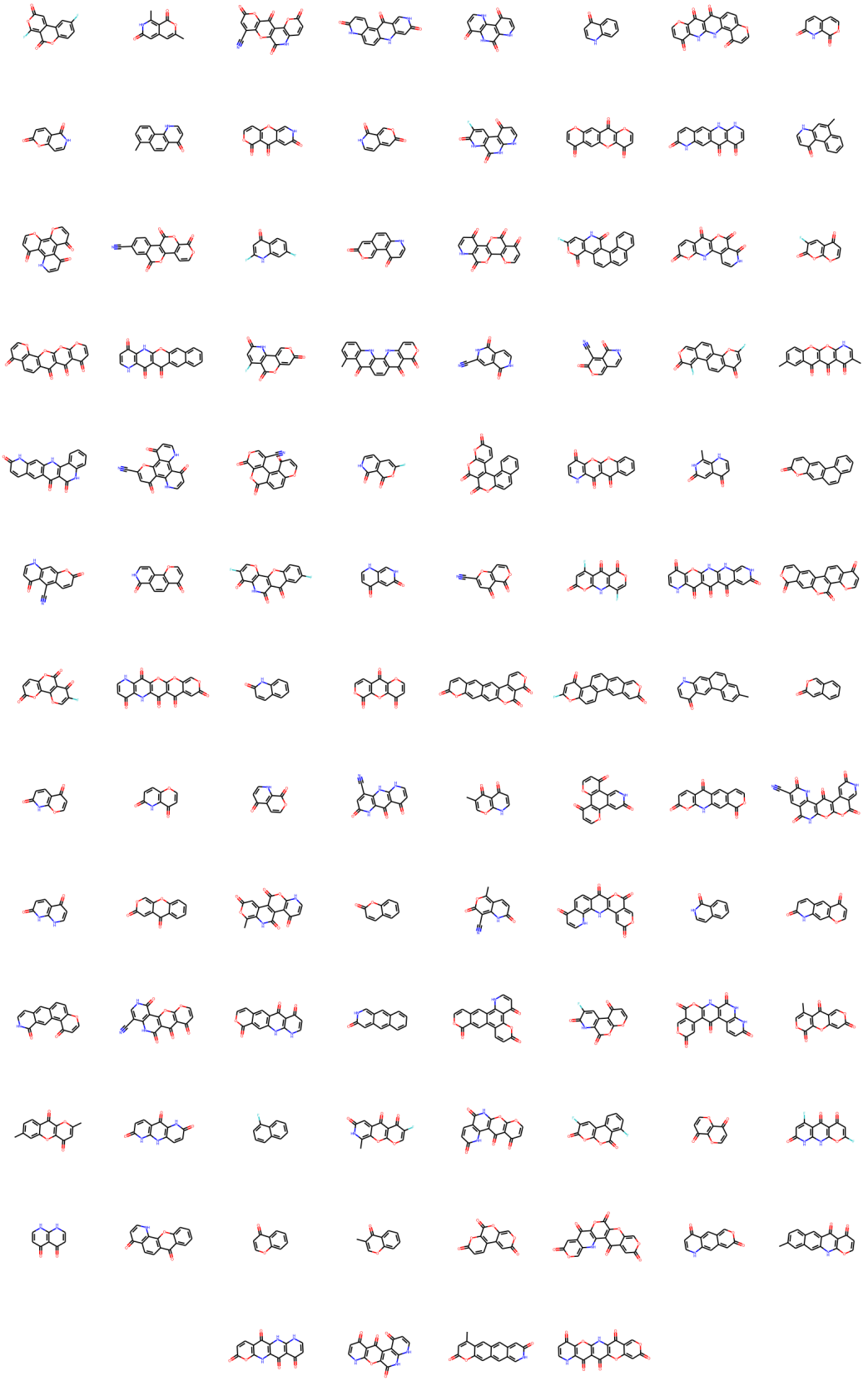


FIGURE A.29: 100 molecules generated for initial 'reduced' population 9

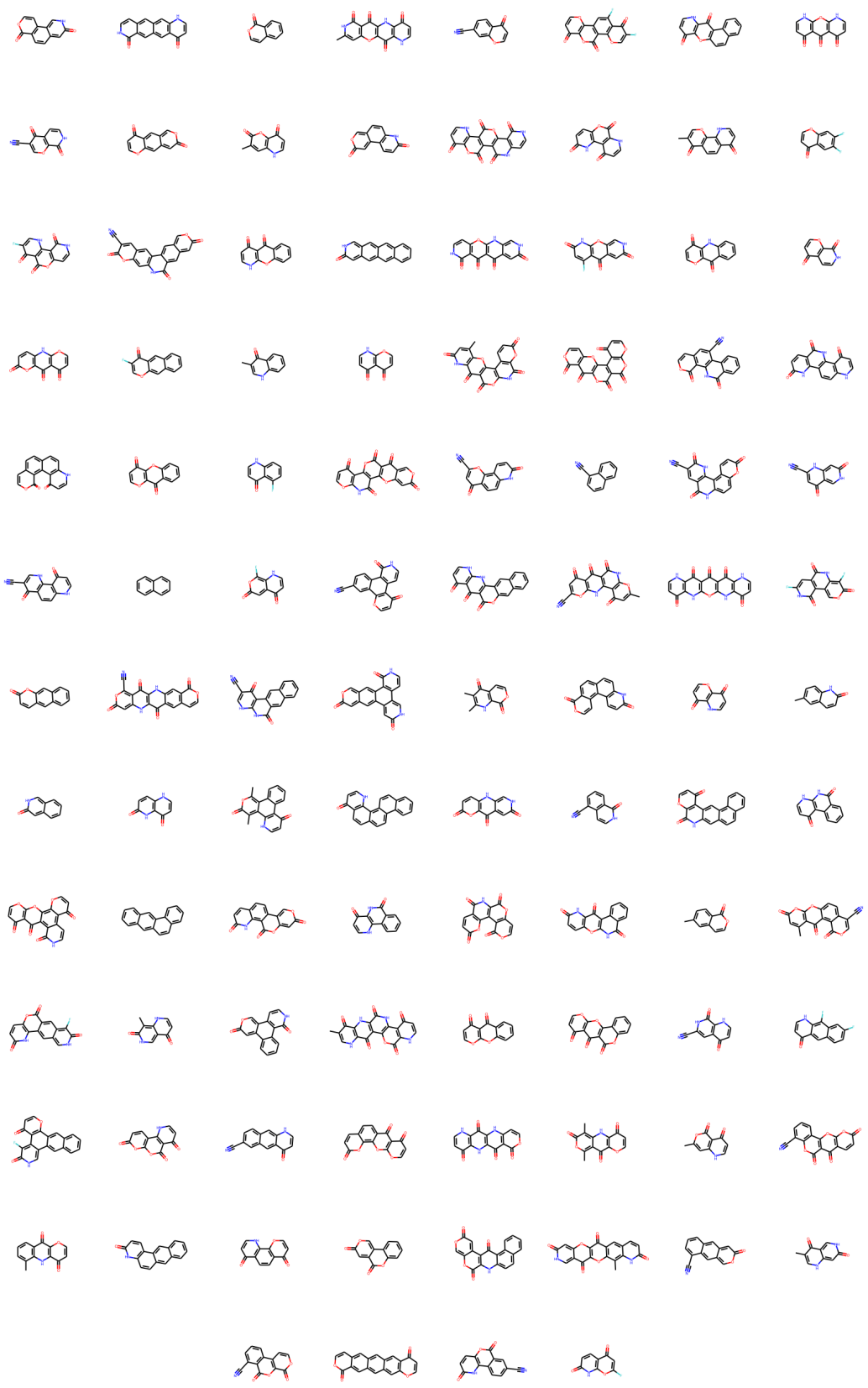


FIGURE A.30: 100 molecules generated for initial 'reduced' population 10

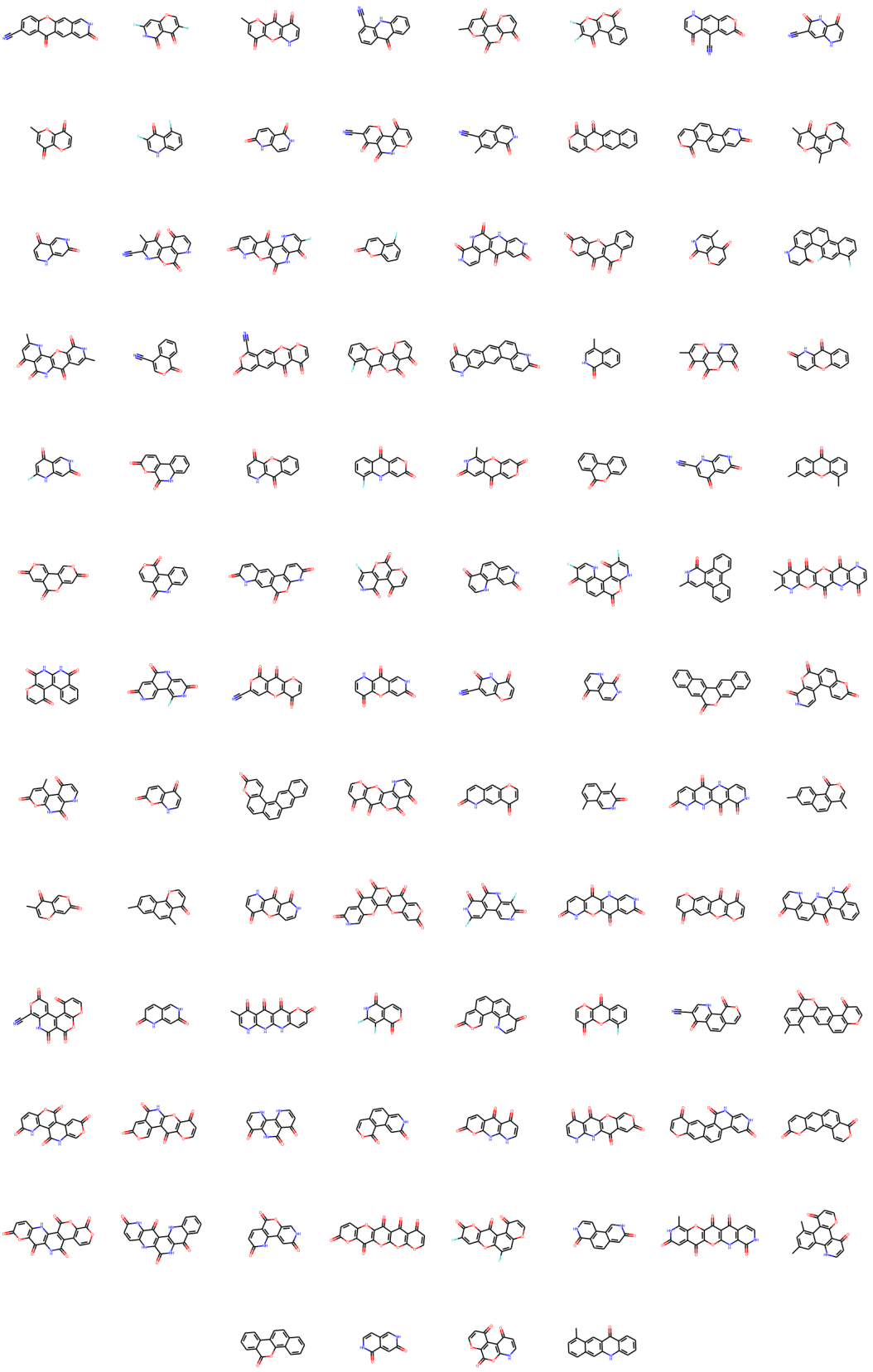


FIGURE A.31: 100 molecules generated for initial 'reduced' population 11

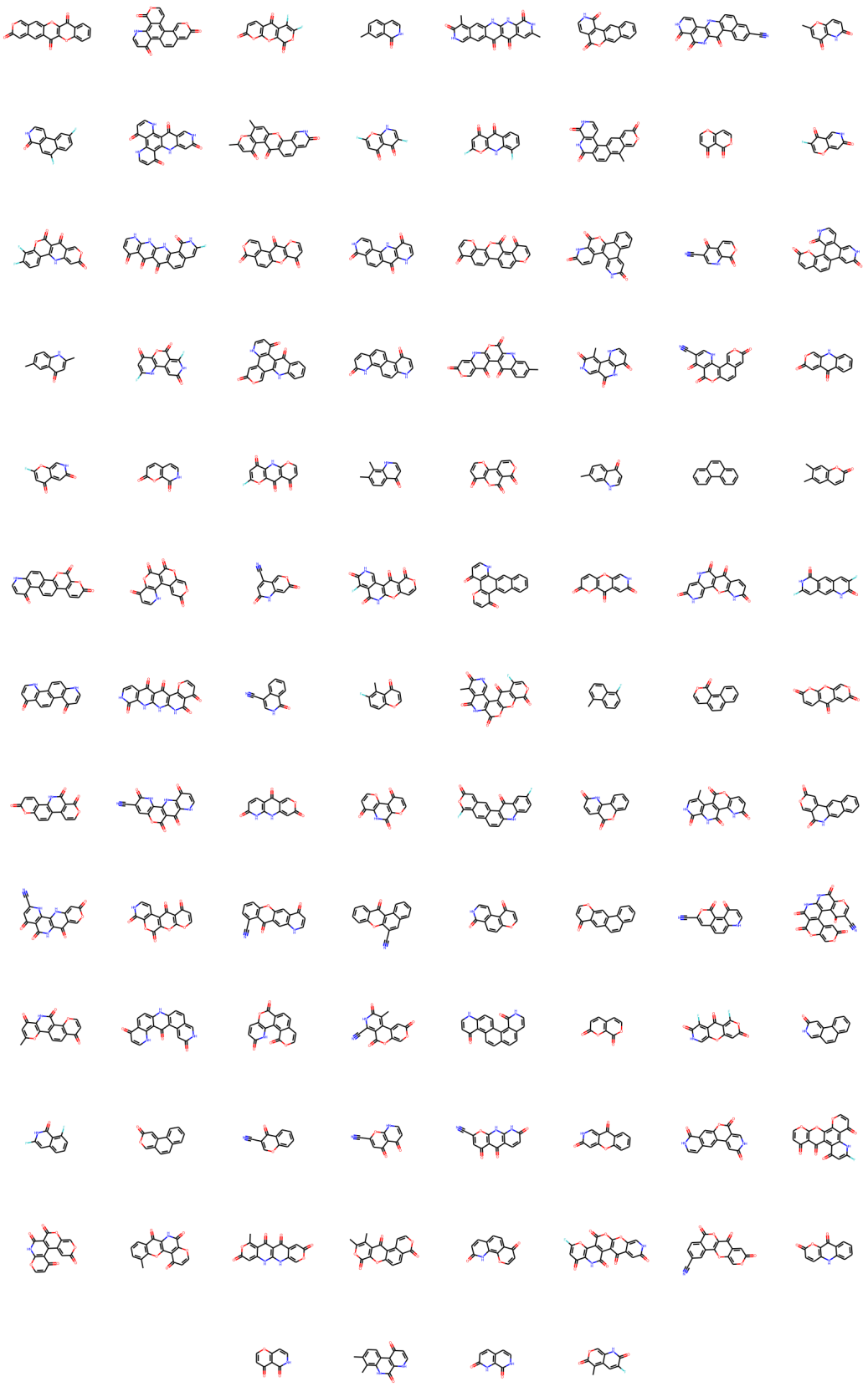


FIGURE A.32: 100 molecules generated for initial 'reduced' population 12

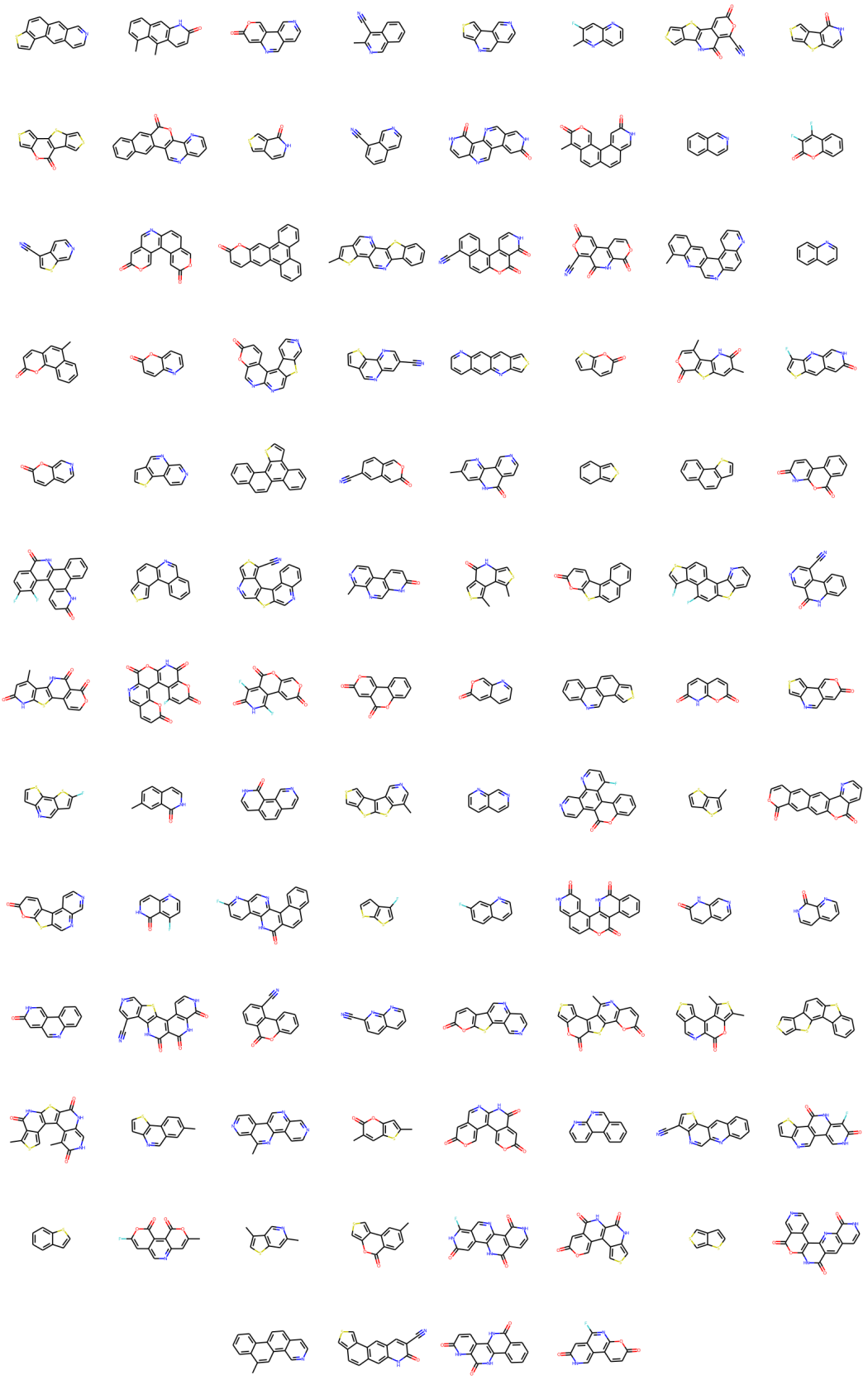


FIGURE A.33: 100 molecules generated for initial 'reduced' population 13

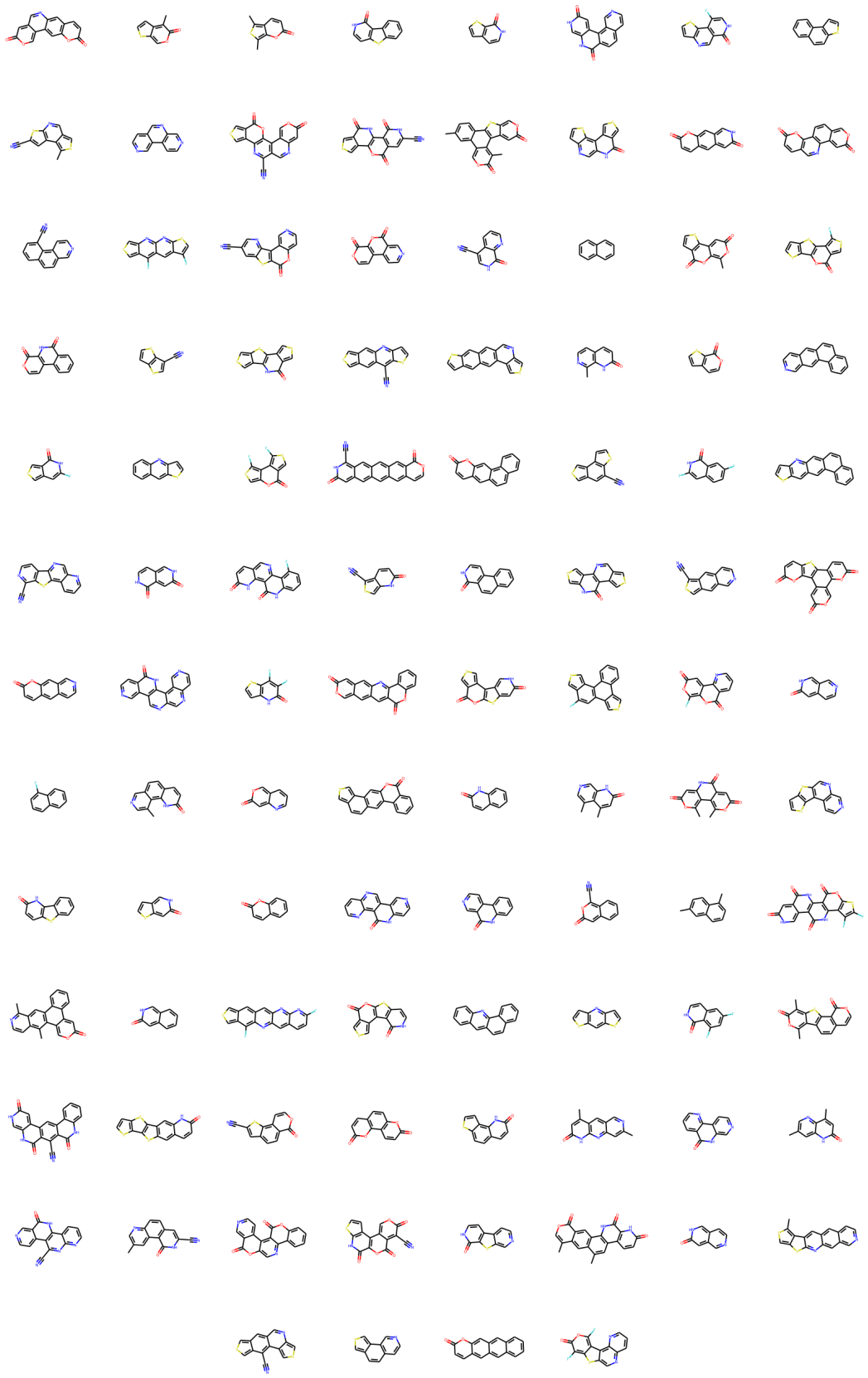


FIGURE A.34: 100 molecules generated for initial 'reduced' population 14

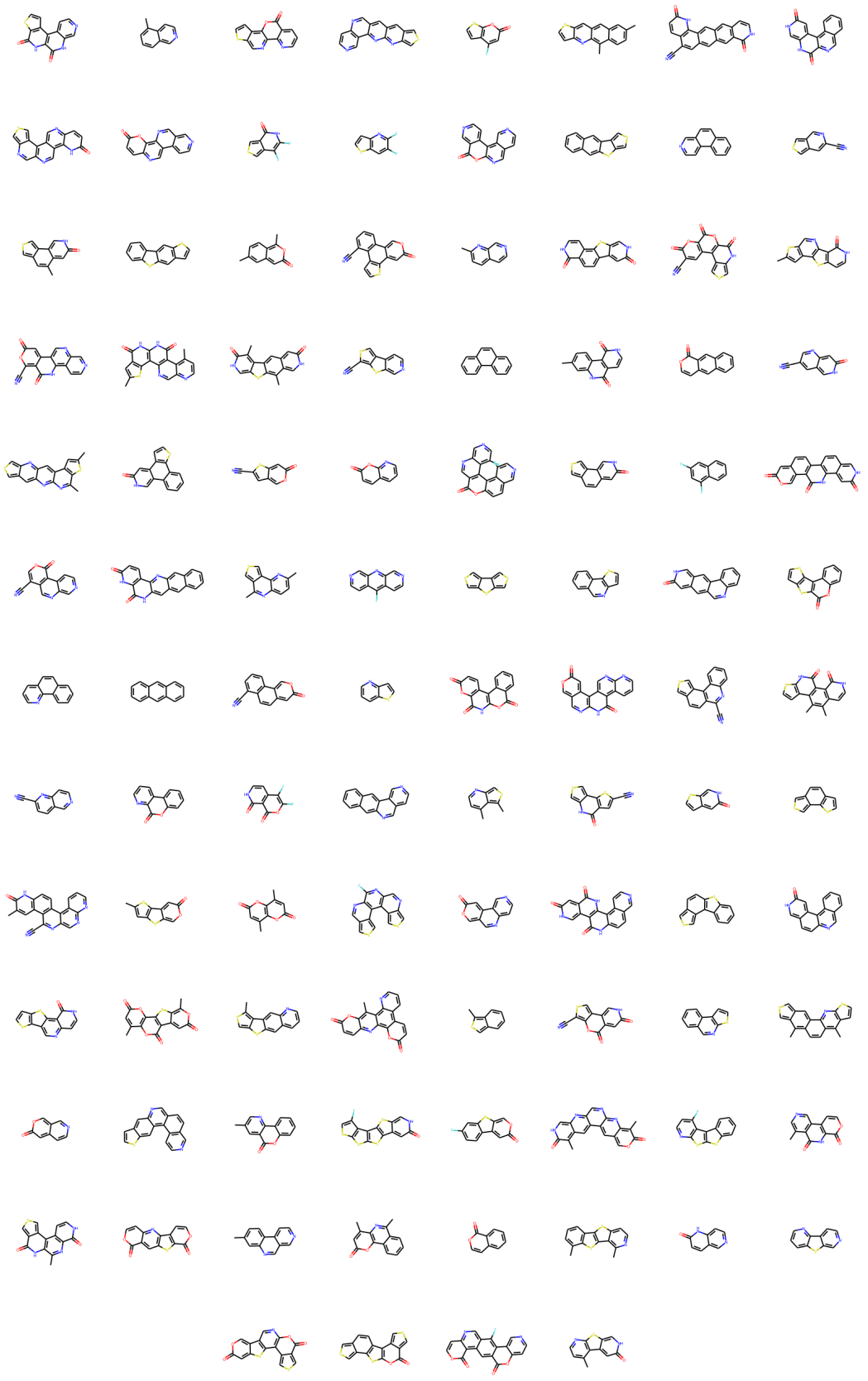


FIGURE A.35: 100 molecules generated for initial 'reduced' population 15

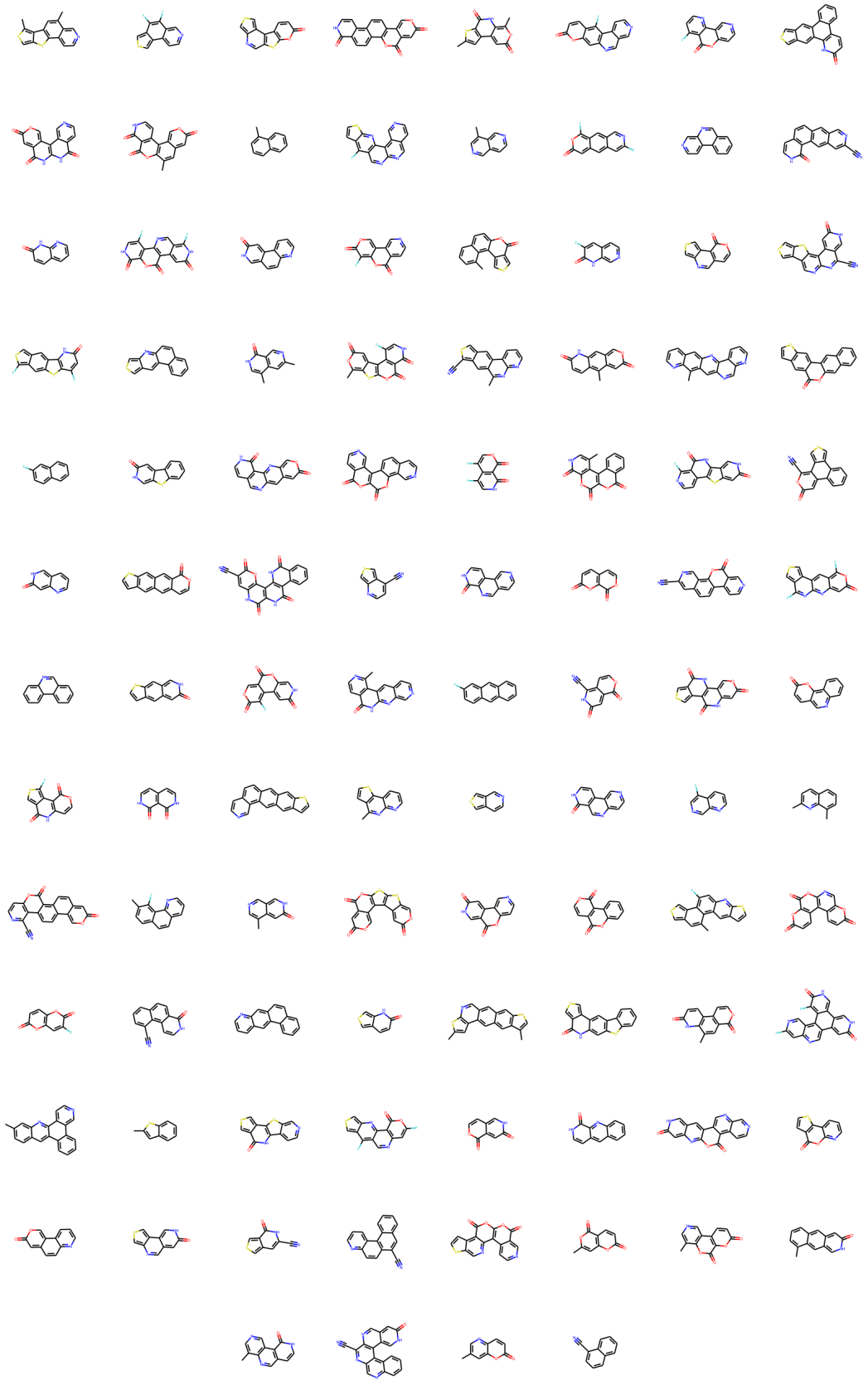


FIGURE A.36: 100 molecules generated for initial 'reduced' population 16

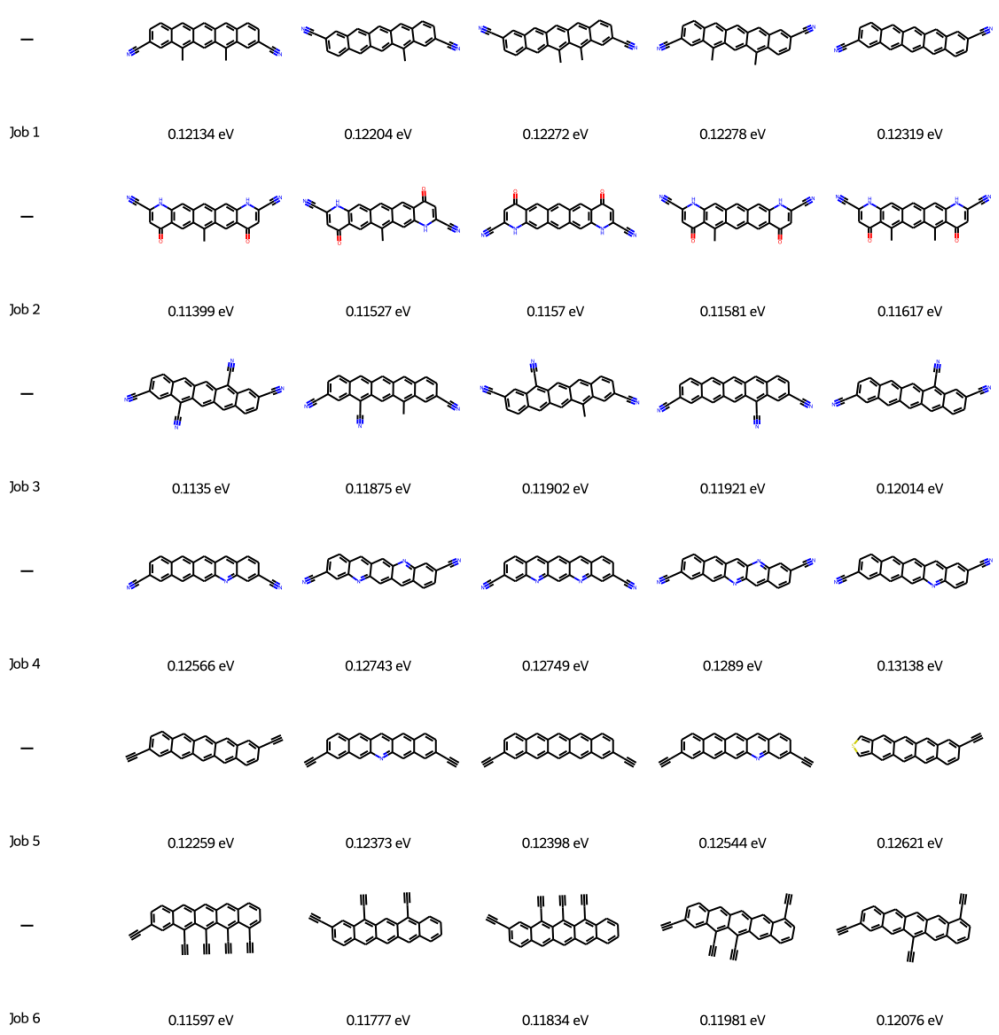



FIGURE A.37: Top 5 molecules sampled in the first round of reorganisation energy GAs, for initial populations 1-6

## A.2 Top performers

Selections of top performing molecules sampled through various genetic algorithms.

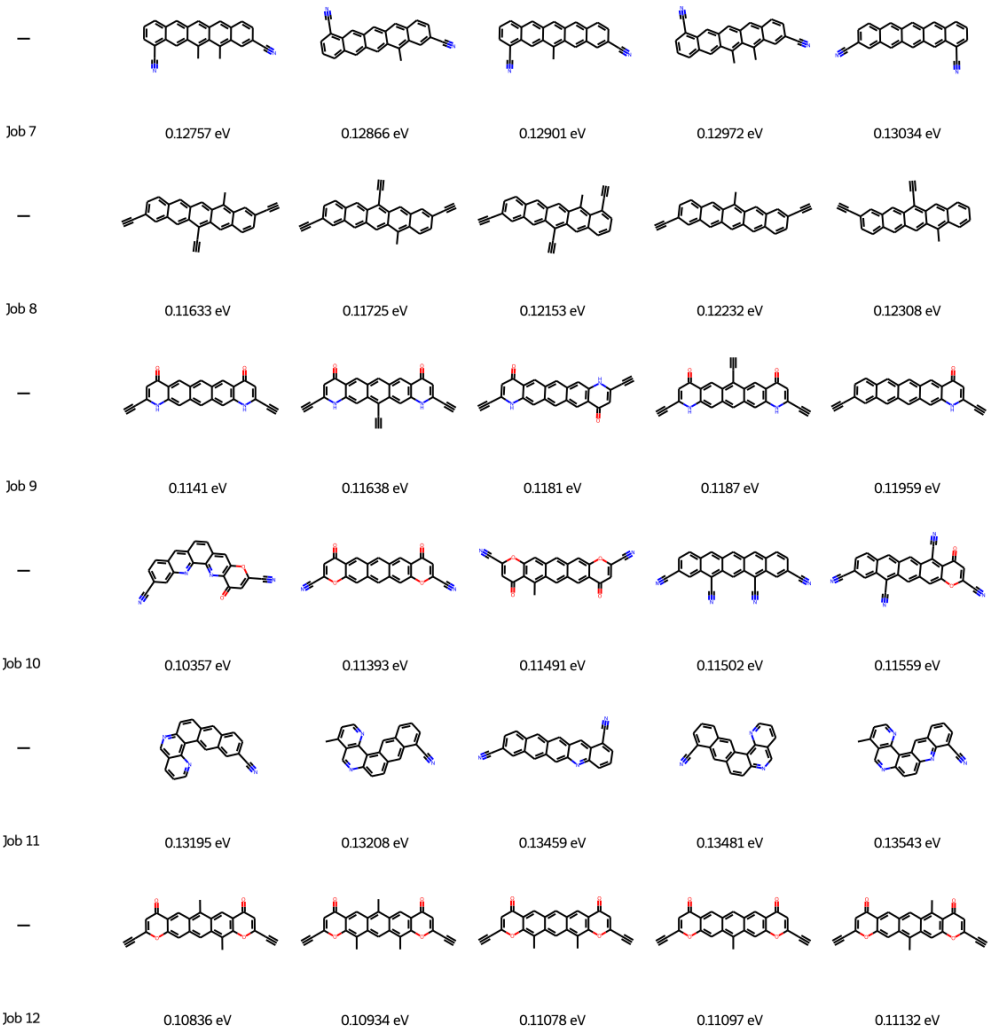



FIGURE A.38: Top 5 molecules sampled in the first round of reorganisation energy GAs, for initial populations 7-12

## A.3 Additional plots

Analysis of the genetic algorithms, mentioned in the main text but not vital to the flow of the thesis.

### A.3.1 Reorganisation energy - reduced populations

## A.4 Synthetic difficulty analysis

Additional plots from synthetic difficulty analysis mentioned in the main text.

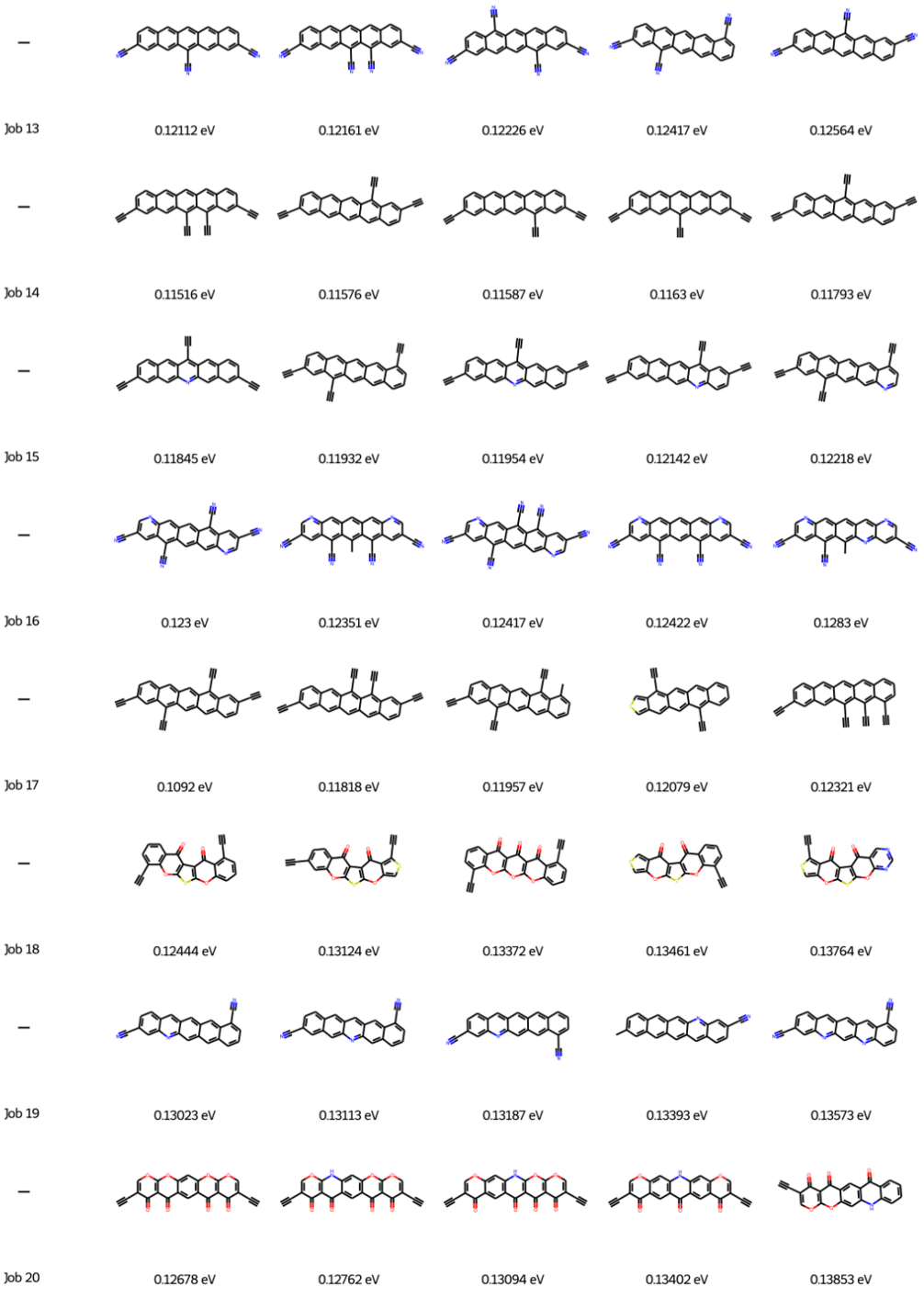



FIGURE A.39: Top 5 molecules sampled in the first round of reorganisation energy GAs, for initial populations 13-20

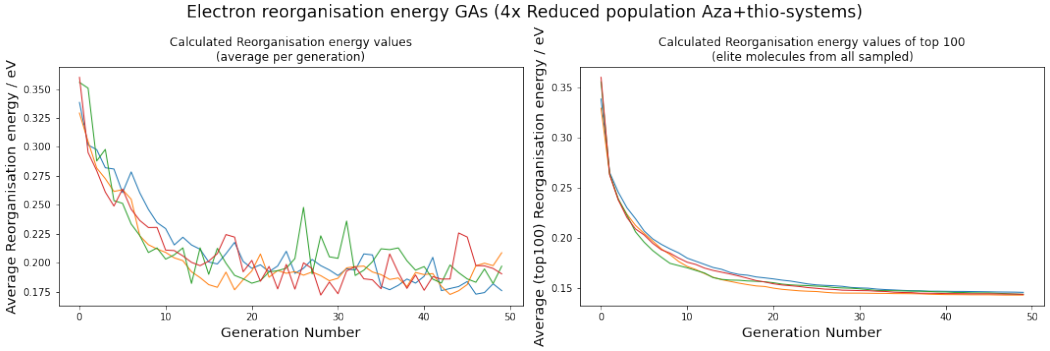



FIGURE A.40: Plots to show how the fitness values (molecular electron reorganisation energy) change as each of these 4 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right)

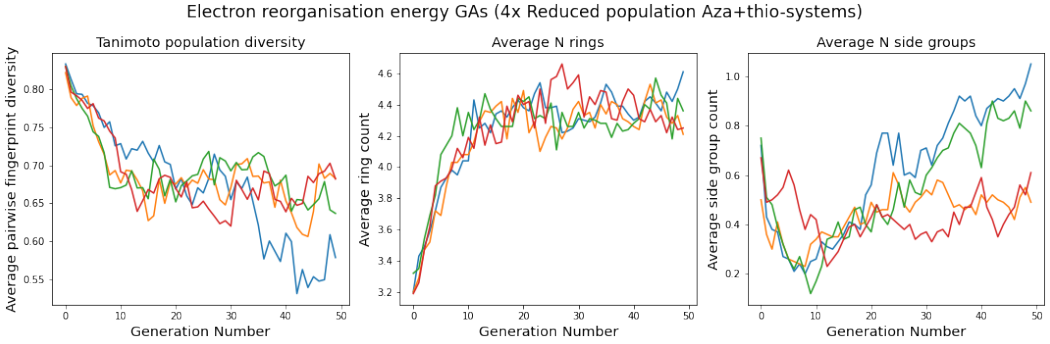



FIGURE A.41: Plots to show diversity and composition for each population: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right)

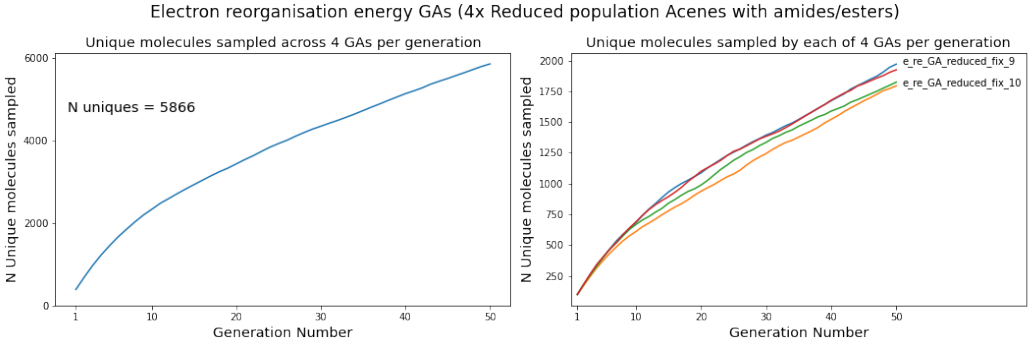



FIGURE A.42: Plots to show how many unique molecules were sampled: by all 4 GAs using the amide/ester ring system building blocks (left); by each of the 4 GAs separately (right)

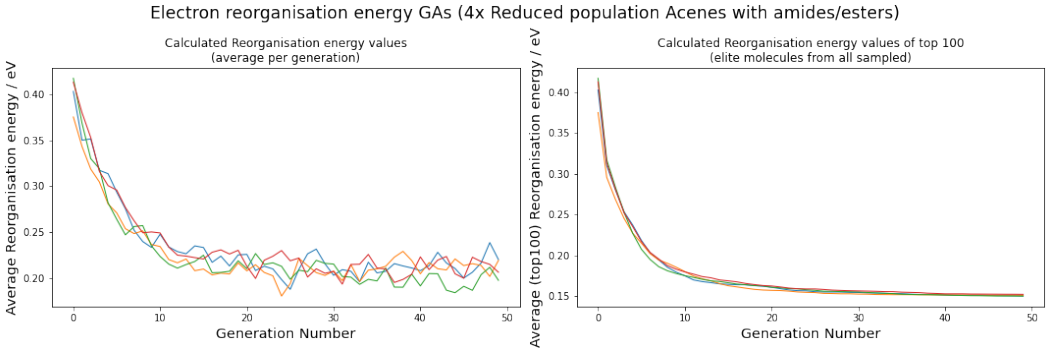



FIGURE A.43: Plots to show how the fitness values (molecular electron reorganisation energy) change as each of these 4 GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right)

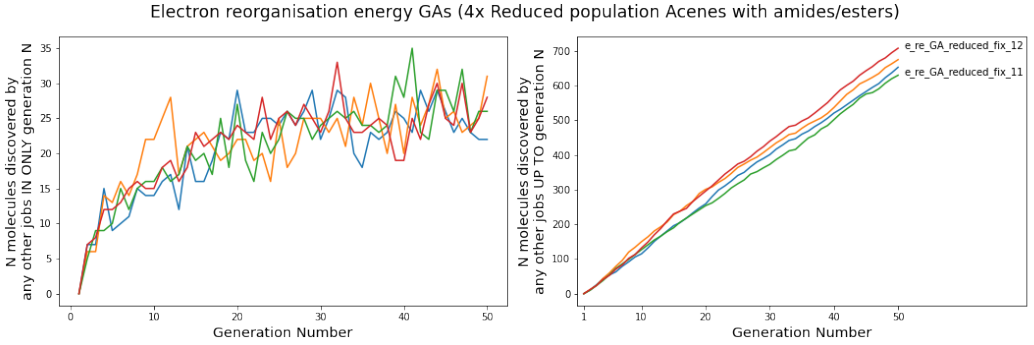



FIGURE A.44: Overlap plots examining each set of 4 amide/ester-type system reduced population genetic algorithms: per generation (left) and cumulative (right)

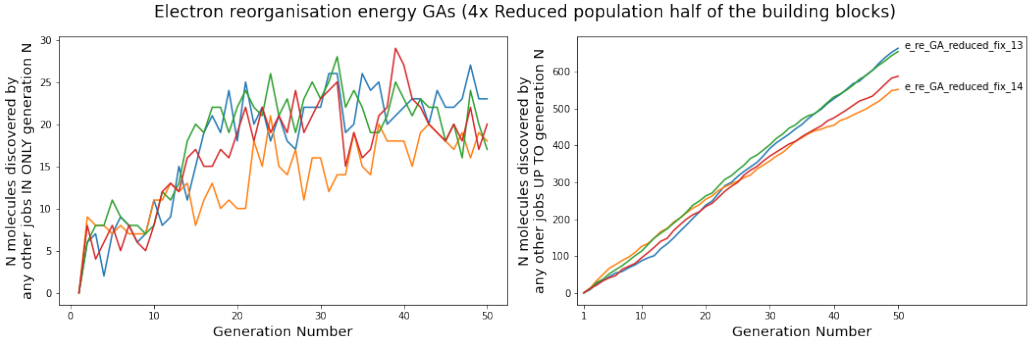



FIGURE A.45: Overlap plots examining each set of 4 half building block reduced population genetic algorithms: per generation (left) and cumulative (right)

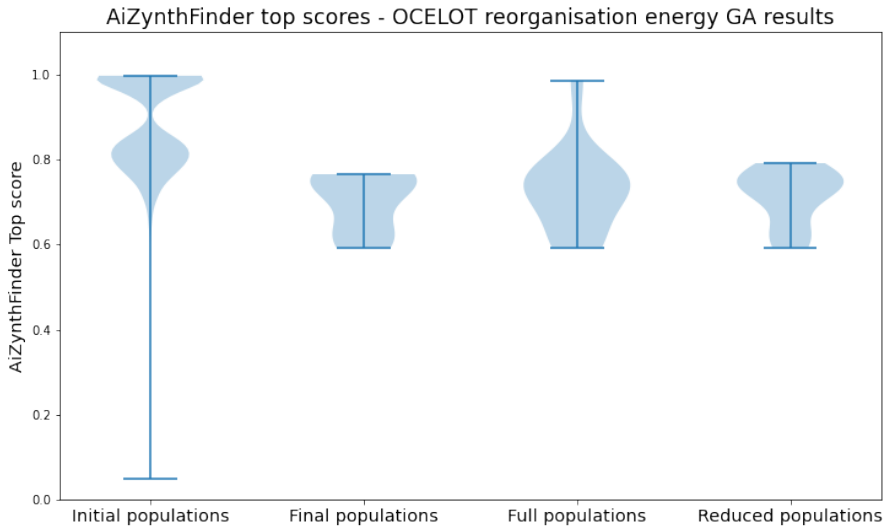



FIGURE A.46: Violin plots describing the AIZynthFinder best scores for predicted synthetic routes to molecules generated with the OCELOT building blocks in reorganisation energy genetic algorithms.

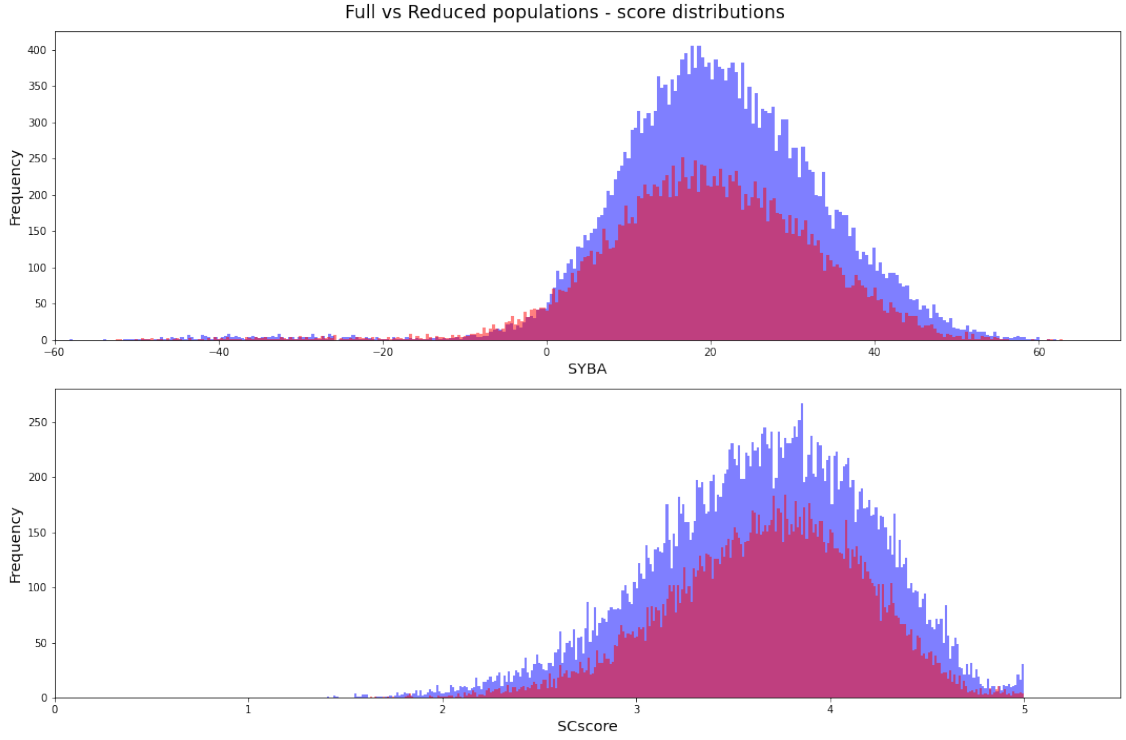



FIGURE A.47: Distribution of calculated SYBA/SCScores for molecules sampled by reorganisation energy genetic algorithms, partitioned according to the type of reduced block building set used.

## A.5 Synthetic difficulty genetic algorithms

Further plotting used in the analysis of genetic algorithms, specifically those optimizing synthetic accessibility.

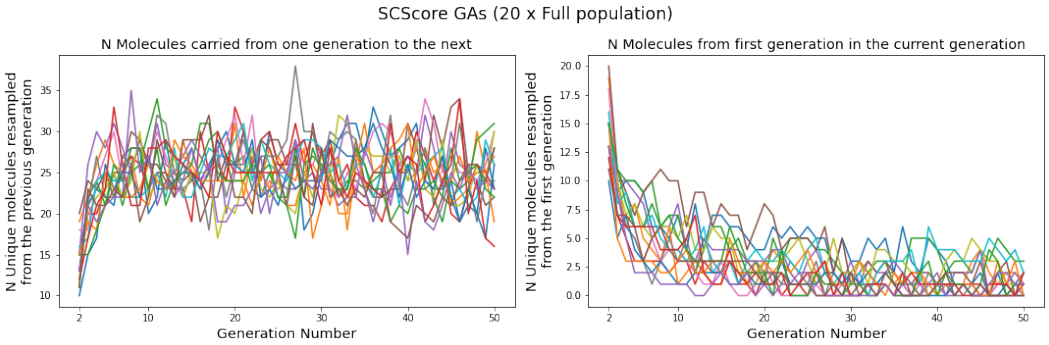



FIGURE A.48: Plots to show how many molecules get 'carried over' from between generations in SCScore-led GAs: counts of how many molecules from the previous generation reappear (left); how many molecules in the current population were also in the initial population (right)

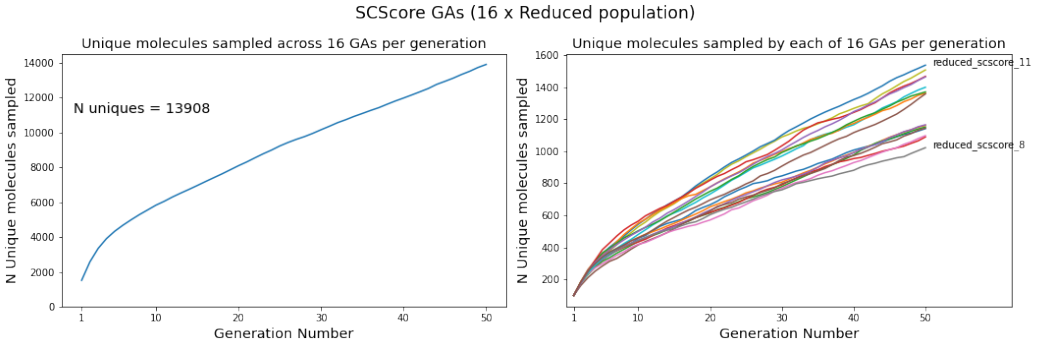



FIGURE A.49: Plots to show how many unique molecules were sampled: by all 16 reduced population SCScore-led GAs (left); by each of the 20 GAs separately (right)

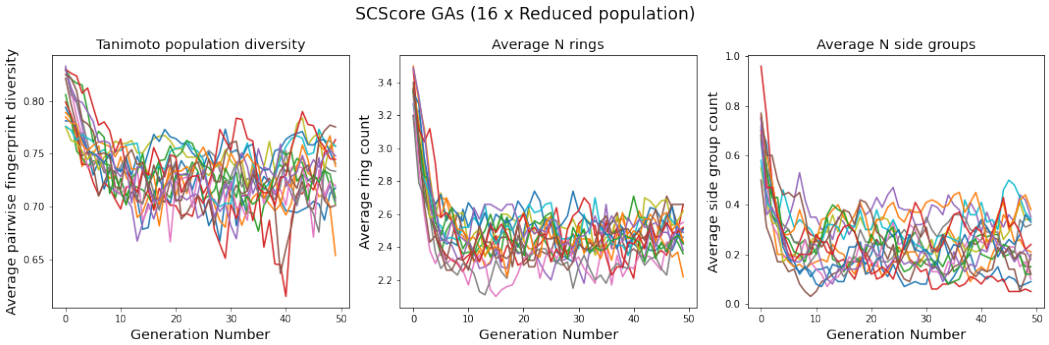



FIGURE A.50: Plots to show diversity and composition for each population in 16 reduced population SCScore-led GAs: average pairwise fingerprint diversity (left); average number of rings (middle); average number of side groups (right)

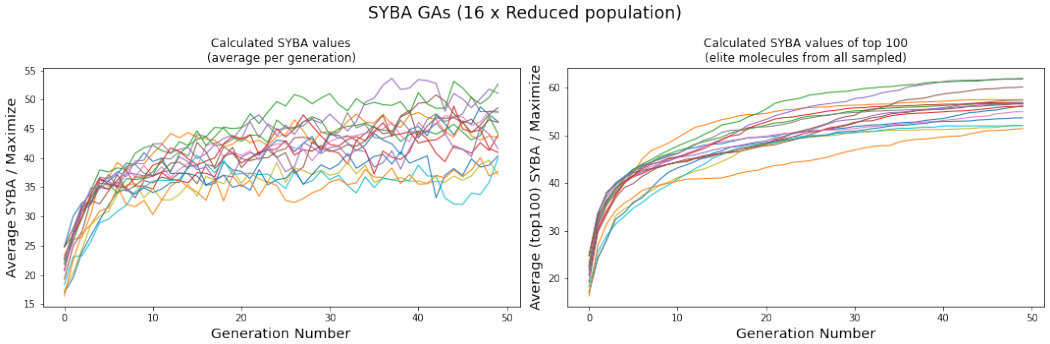



FIGURE A.51: Plots to show how the fitness values (SYBA) change as each of the 16 reduced-population GAs progress: average values per generation (left); average of the top 100 molecules sampled up to a given generation (right)

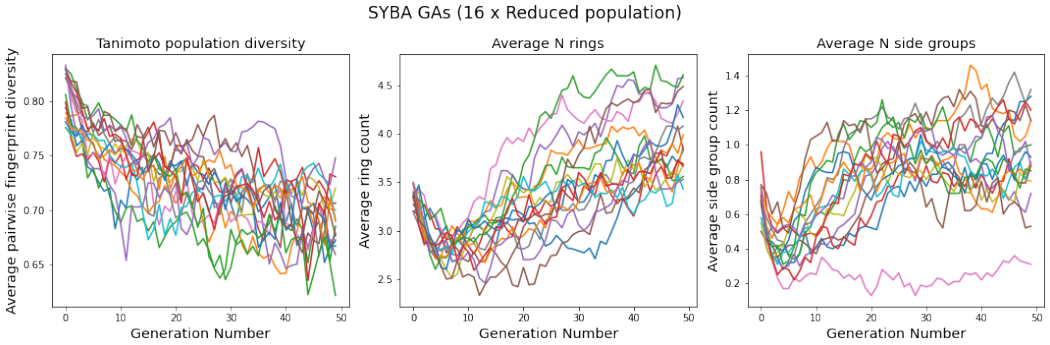



FIGURE A.52: Plots to show diversity and composition in 16 reduced population SYBA-led GAs

# References

Qianxiang Ai, Vinayak Bhat, Sean M Ryno, Karol Jarolimek, Parker Sornberger, Andrew Smith, Michael M Haley, John E Anthony, and Chad Risko. Ocelot: An infrastructure for data-driven research to discover and design crystalline organic semiconductors. *The Journal of Chemical Physics*, 154(17), 2021.

Krisztina Boda, Thomas Seidel, and Johann Gasteiger. Structure and reaction based evaluation of synthetic accessibility. *Journal of computer-aided molecular design*, 21(6): 311–325, 2007.

Pascal Bonnet. Is chemical synthetic accessibility computationally predictable for drug and lead-like molecules? a comparative assessment between medicinal and computational chemists. *European journal of medicinal chemistry*, 54:679–689, 2012.

John Bradshaw, Brooks Paige, Matt J Kusner, Marwin Segler, and José Miguel Hernández-Lobato. Barking up the right tree: an approach to search over molecule synthesis dags. *Advances in neural information processing systems*, 33:6852–6866, 2020.

Nathan Brown. Algorithms for chemoinformatics. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(5):716–726, 2011.

David H Case, Josh E Campbell, Peter J Bygrave, and Graeme M Day. Convergence properties of crystal structure prediction by quasi-random sampling. *Journal of chemical theory and computation*, 12(2):910–924, 2016.

Jiadi Chen, Weifeng Zhang, Liping Wang, and Gui Yu. Recent research progress of organic small-molecule semiconductors with high electron mobilities. *Advanced Materials*, 35(11):2210772, 2023.

Chi Y Cheng, Josh E Campbell, and Graeme M Day. Evolutionary chemical space exploration for functional materials: computational organic semiconductor discovery. *Chemical science*, 11(19):4922–4933, 2020.

Ming Chu, Jian-Xun Fan, Shuaijun Yang, Dan Liu, Chun Fai Ng, Huanli Dong, Ai-Min Ren, and Qian Miao. Halogenated tetraazapentacenes with electron mobility as high as 27.8 cm2 v- 1 s- 1 in solution-processed n-channel organic thin-film transistors. *Advanced materials*, 30(38):1803467, 2018.

Rebecca J Clements, Joshua Dickman, Jay Johal, Jennie Martin, Joseph Glover, and Graeme M Day. Roles and opportunities for machine learning in organic molecular crystal structure prediction and its applications. *MRS Bulletin*, 47(10):1054–1062, 2022.

Connor W Coley, Luke Rogers, William H Green, and Klavs F Jensen. Scscore: synthetic complexity learned from a reaction corpus. *Journal of chemical information and modeling*, 58(2):252–261, 2018.

Graeme M Day. Current approaches to predicting molecular organic crystal structures. *Crystallography Reviews*, 17(1):3–52, 2011.

Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1(1):1–11, 2009.

Eric Fontain. Application of genetic algorithms in the field of constitutional similarity. *Journal of Chemical Information and Computer Sciences*, 32(6):748–752, 1992.

M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross,

V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, Ö. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, and D. J. Fox. Gaussian09 Revision E.01. Gaussian Inc. Wallingford CT 2009.

Fruzsina Gajdos, Siim Valner, Felix Hoffmann, Jacob Spencer, Marian Breuer, Adam Kubas, Michel Dupuis, and Jochen Blumberger. Ultrafast estimation of electronic couplings for electron transfer between $\pi$-conjugated organic molecules. *Journal of Chemical Theory and Computation*, 10(10):4653–4660, 2014.

Wenhao Gao and Connor W Coley. The synthesizability of molecules proposed by generative models. *Journal of chemical information and modeling*, 60(12):5714–5723, 2020.

Samuel Genheden, Amol Thakkar, Veronika Chadimová, Jean-Louis Reymond, Ola Engkvist, and Esben Bjerrum. Aizynthfinder: a fast, robust and flexible open-source software for retrosynthetic planning. *Journal of cheminformatics*, 12(1):70, 2020.

Rebecca L Greenaway and Kim E Jelfs. Integrating computational and experimental workflows for accelerated organic materials discovery. *Advanced Materials*, 33(11): 2004831, 2021.

Geun Ho Gu, Juhwan Noh, Inkyung Kim, and Yousung Jung. Machine learning for renewable energy materials. *Journal of Materials Chemistry A*, 7(29):17096–17117, 2019.

Ichiro Hisaki, Chen Xin, Kiyonori Takahashi, and Takayoshi Nakamura. Designing hydrogen-bonded organic frameworks (hofs) with permanent porosity. *Angewandte Chemie International Edition*, 58(33):11160–11170, 2019.

M Jansen and JC Schön. Rational development of new materials—putting the cart before the horse? *Nature materials*, 3(12):838–838, 2004.

Jan H Jensen. A graph-based genetic algorithm and generative model/monte carlo tree search for the exploration of chemical space. *Chemical science*, 10(12):3567–3572, 2019.

John A Keith, Valentin Vassilev-Galindo, Bingqing Cheng, Stefan Chmiela, Michael Gastegger, Klaus-Robert Muller, and Alexandre Tkatchenko. Combining machine learning and computational chemistry for predictive insights into chemical systems. *Chemical reviews*, 121(16):9816–9872, 2021.

Rui-Biao Lin and Banglin Chen. Hydrogen-bonded organic frameworks: Chemistry and functions. *Chem*, 8(8):2114–2135, 2022.

Daniel Mark Lowe. *Extraction of chemical structures and reactions from the literature*. PhD thesis, University of Cambridge, 2012.

Michael Mastalerz and Iris M Oppel. Rationale herstellung eines extrinsisch porösen molekülkristalls mit einer außergewöhnlich großen spezifischen oberfläche. *Angewandte Chemie*, 124(21):5345–5348, 2012.

Joseph H Montoya, Kirsten T Winther, Raul A Flores, Thomas Bligaard, Jens S Hummelshøj, and Muratahan Aykol. Autonomous intelligent agents for accelerated materials discovery. *Chemical Science*, 11(32):8517–8532, 2020.

Austin M Mroz, Victor Posligua, Andrew Tarzia, Emma H Wolpert, and Kim E Jelfs. Into the unknown: how computation can help explore uncharted material space. *Journal of the American Chemical Society*, 144(41):18730–18743, 2022.

Artem R Oganov, Chris J Pickard, Qiang Zhu, and Richard J Needs. Structure prediction drives materials discovery. *Nature Reviews Materials*, 4(5):331–348, 2019.

Robert M Parrish, Lori A Burns, Daniel GA Smith, Andrew C Simmonett, A Eugene DePrince III, Edward G Hohenstein, Ugur Bozkaya, Alexander Yu Sokolov, Roberto Di Remigio, Ryan M Richard, et al. Psi4 1.1: An open-source electronic structure program emphasizing automation, advanced libraries, and interoperability. *Journal of chemical theory and computation*, 13(7):3185–3197, 2017.

Hitesh Patel, Wolf Ihlenfeldt, Philip Judson, Yurii S Moroz, Yuri Pevzner, Megan Peach, Nadya Tarasova, and Marc Nicklaus. Synthetically accessible virtual inventory (savi). 2020.

Sarah L Price, Maurice Leslie, Gareth WA Welch, Matthew Habgood, Louise S Price, Panagiotis G Karamertzanis, and Graeme M Day. Modelling organic crystal structures using distributed multipole and polarizability-based model intermolecular potentials. *Physical Chemistry Chemical Physics*, 12(30):8478–8490, 2010.

Angeles Pulido, Linjiang Chen, Tomasz Kaczorowski, Daniel Holden, Marc A Little, Samantha Y Chong, Benjamin J Slater, David P McMahon, Baltasar Bonillo, Chloe J Stackhouse, et al. Functional materials discovery using energy–structure–function maps. *Nature*, 543(7647):657–664, 2017.

Jesse TE Quinn, Jiaxin Zhu, Xu Li, Jinliang Wang, and Yuning Li. Recent progress in the development of n-type organic semiconductors for organic field effect transistors. *Journal of Materials Chemistry C*, 5(34):8654–8681, 2017.

Anthony K Rappé, Carla J Casewit, KS Colwell, William A Goddard III, and W Mason Skiff. Uff, a full periodic table force field for molecular mechanics and molecular dynamics simulations. *Journal of the American chemical society*, 114(25):10024–10035, 1992.

Robert Roe. New supercomputer aims to simplify use of hpc: Robert roe talks to southampton university's oz parchment about the decision-making behind installing the latest hpc system at the university. *Scientific Computing World*, (159): 20–22, 2018.

Stephan C Schürer, Prashant Tyagi, and Steven M Muskal. Prospective exploration of synthetically feasible, medicinally relevant chemical space. *Journal of chemical information and modeling*, 45(2):239–248, 2005.

Teague Sterling and John J Irwin. Zinc 15–ligand discovery for everyone. *Journal of chemical information and modeling*, 55(11):2324–2337, 2015.

Anthony J Stone. Distributed multipole analysis of gaussian wavefunctions gdma version 2.2. 02. *GDMA University of Cambridge, UK*, 1999.

Austin Tripp and José Miguel Hernández-Lobato. Genetic algorithms are strong baselines for molecule generation. *arXiv preprint arXiv:2310.09267*, 2023.

Norbert M Villeneuve, Joshua Dickman, Thierry Maris, Graeme M Day, and James D Wuest. Seeking rules governing mixed molecular crystallization. *Crystal Growth & Design*, 23(1):273–288, 2022.

Milan Voršilák, Michal Kolář, Ivan Čmelo, and Daniel Svozil. Syba: Bayesian estimation of synthetic accessibility of organic compounds. *Journal of cheminformatics*, 12:1–13, 2020.