A Computational and Mathematical Approach to Synthesis Planning

Authors:
Carsten Grønbjerg Lützen and Daniel Fentz Johansen
Supervisor:
Daniel Merkle
Department of Mathematics and Computer Science,
University of Southern Denmark, Odense

Abstract

Synthesis planning plays an important role in organic chemistry. The goal of synthesis planning is to find the *best* way to produce certain molecules. The field of synthesis planning has been explored throughly in the literature, with concepts like *retrosynthetic analysis*. Existing approaches and tools rely on user input and/or large databases of compounds.

In this thesis we introduce a concept called HoSP (Hypergraph of Synthesis Plans) and use methods from graph theory to model realistically sized molecules and synthesis plans. In addition we enumerate the K best synthesis plans using different metrics. In our approach we focus on the skeleton of the molecule leaving out details regarding functionalization, when examine the cost of a synthesis plan. This approach does not rely on any in priori knowledge from databases or the user contrary to already existing systems. We introduce different methods for constructing the HoSP, all methods is analyzed and empirically tested. The different methods determine different properties for the HoSP, such as uniqueness of plans and number of reactions in the synthesis plans. Furthermore we present an open source database SynthDB for storing and comparing synthesis plans. We also introduce a new format SPF for storing and documenting synthesis plans. The database enable chemists to find different synthesis plans which includes the same reactions or molecules. The database also provide some core search functionality across synthesis plans. We automatically project a complete real life synthesis plan down to our model, and empirically study the model with respect to different constraint and options.

We conclude that our approach for synthesis planning is feasible and could be used by chemists to devise new skeleton plans. The SynthDB will introduce new functionality to the field.

Contents

1 Introduction and Related Work

Computer-assisted organic synthesis (CAOS) is an established field [10, 19, 24, 24, 27, 43]. But common for most of the programs is the level of human involvement and the use of compound databases. Systems like LHASA (Logic and Heuristics Applied to Synthetic Analysis) and MASA [7] (Molten-Salt-Assisted Self-Assembly) use the human perception to ensure better results. WODCA (Workbench for the Organisation of Data for Chemical Application) uses several databases of compounds to help the synthesis planning [25, 32].

In recent years, as far as we know, no attempts have been made to make a fully automated open system that does not use a database of compounds capable of creating new synthesis plans, although Chematica [39] goes in that direction but is closed source. We believe the recent development within cheminformatics and computer hardware makes it plausible to believe that progress within the field can be made.

In this thesis we will focus on the skeleton of a molecule when creating synthesis plans. The skeleton of the molecule is the single most important thing [26, p. 5].

> The points stressed earlier should be highlighted once more: Construction of the skeleton of the target structure is the prime task in synthesis planning, not the placement of functionalities or stereogenic centers.

The overall objective of this thesis is to introduce, design and implement a set of tools capable of helping chemists creating new synthesis plans. The tools should assist chemists in the important decisions. Given a molecule and a bondset, the tools should be able to recommend a good plan for a given bondset. A useful improvement of this would be to output the K best synthesis plans. Furthermore if no good bondsets are known, it would be beneficial if the tool could suggest bondsets of a given size that are optimal with respect to the metrics. This would make it easy for chemists to specify the number of bonds in the bondset and the tools should then use the *best* bondset. This could also be used to improve already existing plans by proposing new bonds which could be added to the plan.

Another problem in modern synthesis planning is the missing possibility to cross reference molecules and reactions. One of the tools should be able to store and cross reference molecules and reactions. With such a tool chemists should easily find alternative ways of synthesizing a molecule. The problem will be broken down, introduced and discussed later in the thesis.

1.1 Formal Approach to Chemistry

What is *cheminformatics* one might ask? Before computers were invented some people began applying mathematics to chemistry in search for a better understanding of the chemical processes. One of the first mentions of this is Krum that in 1874 wrote:

> Chemistry will then become a branch of applied mathematics; but it will not cease to be an experimental science. Mathematics may enable us retrospectively to justify results obtained by experiment,

> may point out useful lines of research, and even sometimes predict entirely novel discoveries, but will not revolutionise our laboratories; mathematics will not replace chemical analysis.
>
> We do not know when the change will take place, or whether it will be gradual or sudden, but none who believes in the progress in human knowledge, and in the consistence in Nature can doubt that ultimately the theory of chemistry, and of all other physical sciences, will be absorbed into the one theory of dynamics.
>
> - *Alexander Crum Brown, 1874* [13]

Even in 1874 it showed a lot of promise to apply mathematics to chemistry, today even more [44]. This process have been ongoing for many years with various results. In the last 50 years computers have begun to play a more and more important role in many parts of our daily life, from banking to personal computers. This were also the case in chemistry where a new branch was born. It was called *cheminformatics*.

The idea behind cheminformatics is to apply computational and informational techniques to a range of problems all originating from chemistry [24, 27]. In the more recent years the pace has begun to increase due to the fact that faster and more powerful computers have become available. This enables us to simulate complex reaction networks, search for molecules in large online databases and map atoms.

But much work still needs to be done. Things in chemistry can be difficult to model and as Andraos [5] writes:

> Chemistry is a science of compromise. A chemist needs to decide at any given time what are the advantages and what are the liabilities of each process for consideration.

These compromises can be really difficult to formalize and model precisely in mathematical terms. Hence many simple problems in chemistry can be extremely difficult to model and solve.

Since cheminformatics is a rather new field much of the terminology used is ambiguous and require some formal introduction [41]. Some parts of chemistry is considered a mix of magic [5], art [41] and science.

One example is when a chemist talks about the skeleton of a molecule [6]. No clear definition exists in the literature making it difficult to use it without first providing a definition. Hence in the following sections we are going to introduce and formalize some of the concepts used in this thesis.

1.1.1 Molecule

Most people have heard the word *molecule* before. A molecule consists of two or more atoms held together in an electrically neutral group. The force that connects the atoms is called chemical bonds, or just *bonds*. An atom can have different properties, one of these is how many bonds it allows to other atoms. A bond can also have different properties, it can be a *single bond*, *double bond* or *triple bonds*. Bonds can be classified as *strong bonds* or *weak bonds*. Each molecule may have different chemical properties.

Figure 1: Benzene. To the left drawn as a molecule by chemical software and to the right drawn as a graph with alternating bonds.

Another way to think of a molecule is to simply view them as graphs as seen in Figure 1. Each atom in the molecule is translated into a labeled node according to the name of the atom. The bonds are translated into edges. Several possibilities exist for translating the bond types, one could add two bonds for a double bond or just label the edge. The edges do not contain any direction, hence it is an undirected graph. Molecules often contain cycles, this is trivial to modeled in the graph.

In chemistry literature several formats exists for translating a molecule into a textual representation. Most notably: SMILES and InChI. The idea behind these formats is to exploit some of the graph properties and convert these into a string representing for the entire molecule, but they differ in their core approach.

Simplified Molecular-Input Line-Entry System

This format is also known as SMILES. It is one of the most human readable formats from the chemical literature. The core idea is to provide a format that is really simple to read and write. A chemist can very quickly write simple molecules in the SMILES format. The SMILES format is a *hydrogen suppressed* format. This means hydrogen atoms are not written in the SMILES format.

Atoms are denoted with the normal characters known from the periodic table As an example carbon is denoted `C`, nitrogen as `N`, etc.

If we are dealing with aromatic bonds then the atom is written in lowercase. A good example is Benzene. Benzene is written as `c1ccccc1`, see Figure 1. This is a very convenient notation. Single bonds are implicit unless they are implied differently by the adjacent atoms.

This makes it possible to write ethanol as `CCO`. A double bond is noted as `=` and a triple bond `#`. In real life chemistry a lot of molecules contain rings, these are easily modeled by labeling some of the atoms. A good example is cyclohexane which is denoted as `C1CCCCC1`.

Branching is also needed. Branching is required when we can not construct a Hamiltonian Path [12, p. 1066] in the molecule. Representing a new branch in SMILES is done by using parentheses from the atom where the branch should start. One trivial example is fluoroform which is written as `FC(F)F`.

SMILES is not a *canonical* format by design. This is easily seen by looking at ethanol. Ethanol can also be written as: `CCO`, `OCC` and `C(O)C`.

Hence given a molecule as a SMILES string one needs to transform the molecule to be able to search for it. Several algorithms exist for creating

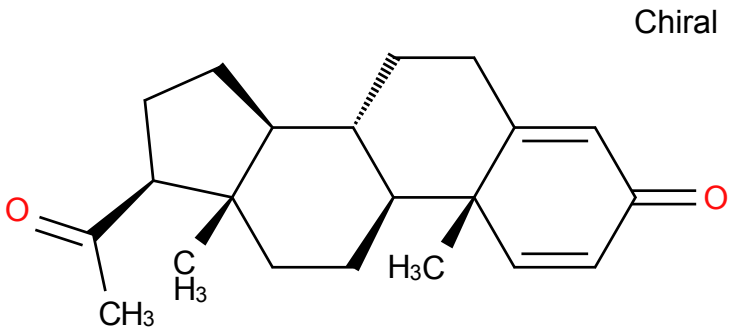


Figure 2: 1-dehydroprogesterone.

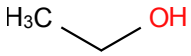


Figure 3: Ethanol.

a canonical SMILES string, and several implementations of each algorithm exists, with their own interpretation.

This has the following consequences: Given a "canonical" SMILES string one cannot be sure that it is the same "canonical" representation that another another canonical SMILES algorithm would generate. Hence one can never trust a "canonical" SMILES string unless it is generated by yourself.

A more advanced example of a molecule and the corresponding SMILES string can be seen in Figure 2 and Table 1. Even with the good intentions of human readability for medium sized molecules, as the one in Figure 2, the SMILES string can be very long and complex.

Molecule	SMILES	InChI
1-dehydroprogesterone, Figure 2	CC(=O)[C@H]1CC [C@@H]2[C@@]1 (CC[C@H]3[C@H] 2CCC4=CC(=O)C =C[C@]34C)C	InChI=1S/C21H28O2/c1-13 (22)17-6-7-18-16-5-4-14- 12-15(23)8-10-20(14,2)19 (16)9-11-21(17,18)3/h8,10, 12,16-19H,4-7,9,11H2,1-3H3 /t16-,17+,18-,19-,20-,21+/m0/s1
Ethanol, Figure 3	CCO, OCC, C(O)C	InChI=1S/C2H6O/c1-2-3/h3H,2H2,1H3

Table 1: Examples of molecules and their identifiers.

International Chemical Identifier

The InChI format has been widely accepted in the chemical society. It provides an easy to understand canonical format. It is not a hydrogen suppressed format. It consists of different layers, each with a specific responsibility. One layer models which atoms are in the molecule, one layer models connections, another layer takes care of stereochemistry, etc. The InChI format contain a lot of advanced features making it a very potent format. One can specify several molecules in one InChI string even if not connected. But the format does not support aromatic bonds per design. An aromatic bond should instead be modeled as an alternating single-double bond.

An InChI string can either be standard or non-standard. This is denoted with an `S` in the beginning of the string: `InChI=1S/...`. The standard mode is the canonical representation that must have been created using the official algorithm. To generate standard InChI an official algorithm exists. This algorithm consists of three iterations: Normalization, canonicalization and serialization. The first iteration removes redundant information from the string, the next iteration creates a canonical labeling of the atoms in the molecule and the last iteration translates this canonical labeling into a string.

An example of a molecule encoded as an InChI string is ethanol, see Figure 3. Ethanol is written as `InChI=1S/C2H6O/c1-2-3/h3H,2H2,1H3`. First we specify that it is a standard InChI string. The first layer, denoted with `/`, tells that the molecule consists of 2 carbon and 6 hydrogen atoms and only 1 oxygen atom. The atoms are labeled in the order they are found in the first layer except hydrogens, hence atom 1 and 2 are carbon and 3 is oxygen. The next layer, denoted by `/c`, defines how the atoms are connected, atom 1 is connected to 2, which is again connected to atom 3. The last layer, `/h`, in ethanol specifies where the hydrogens are attached. Atom 3 has one hydrogen, atom 2 has two hydrogens and atom 3 has three.

1.1.2 Skeleton

The skeleton is not formally defined. hence we make our own definition. We define the skeleton to be a substructure of a molecule. Primarily consisting of carbon. The skeleton is the carbon structure without any functional groups.

1.1.3 Reaction

A chemical reaction is a transformation from one set of molecules, denoted *educts*, to another set of molecules, denoted *products*. This transformation can move existing bonds in the molecules or create entirely new molecules.

Reactions can have different properties such as relocating bonds and/or moving functional groups. Reactions have different yields. Hence, the molecule we are interested in might only be formed in small quantities. To denote this the term *yield* is introduced. Some argue that it makes little to no sense to talk about yield [22]. Often complex reactions have a very low yield, hence chemists tend to work on smaller reactions ensuring a larger yield and postponing complex reactions.

When we talk about molecules as graphs we can think of the reactions as merely relocating bonds/edges. This can be done since in a balanced reaction nothing is removed due to the law of conservation of mass, hence all atoms should be preserved and the total number of bonds should be the same.

An example is the following reaction:
$CH_4 + 2O_2 \longrightarrow CO_2 + 2H_2O$
Here we have conservation of mass, it is a balanced reaction. A visual representation of this reaction can be seen in Figure 4.

With the graph notation we can view reactions as graph grammar rules [2]. As seen in MØD this can be a valuable way of simulating reactions. In this way all relocations of bonds are modeled as edges being removed and new edges being formed.

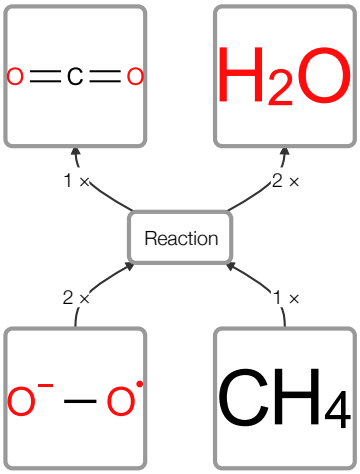


Figure 4: A simple reaction.

Another way of viewing reactions is by mapping atoms in the educts to atoms in the products. This way of looking at reactions has become popular, mainly due to faster computers.

1.1.4 Synthesis Plans

A synthesis plan is a series of reactions that produces a target compound. The synthesis plan can often be visualized as a synthesis tree. Synthesis trees are often depicted as unary-binary trees [26, p. 3] where the root node is the target compound and the leaf nodes are the input compounds, see Figure 5.

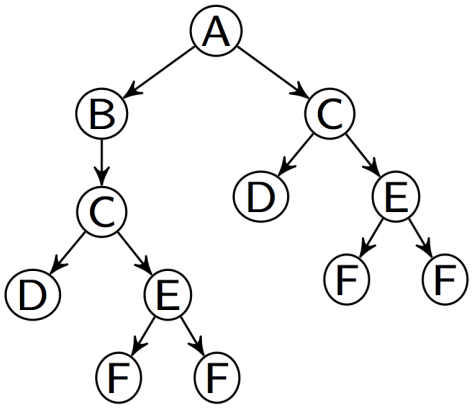


Figure 5: Unary-binary representation of synthesis plan.

We say that a synthesis plan is convergent if the tree is balanced. A convergent synthesis plan is often better than a unbalanced tree. Reactions involving large compounds often have a lower yield, unbalanced trees have more of these reactions.

An example of a synthesis plan is shown in Figure 6.

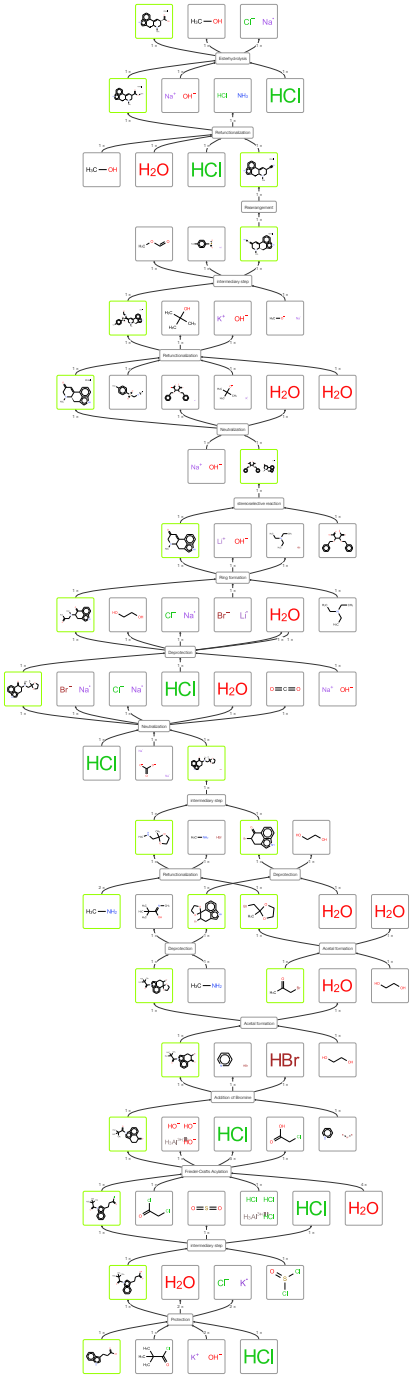


Figure 6: Synthesis plan for Lysergic Acid by Moldvai et al [36].

Bondset

The term *bondset* is widely used when dealing with synthesis plans [9]. A bondset is nothing more than a set of bonds. Several strategies exist for choosing which bonds to break. Two major frameworks exists: MASA [7] and LHASA [43]. These are often mentioned as the current state of art methods within synthesis planning. In more recent literature, a more simple method is introduced. The method is called *total walk count* and recursively determine which bonds to include in the bondset [42].

E. J. Corey introduced several strategies for selecting these bonds [9]. These strategies often aim to break the molecule into smaller parts with lower complexity in a recursive manner.

The selection of the bondset is important [26, p. 5]. A wrong bondset is very likely to be infeasible in the lab where the best bondset might save resources. The selection of a bondset is something most chemists know of. They often rely on previous knowledge instead of formalized methods.

1.1.5 Synthesis Planning

The art of organic synthesis planning is very complex [5, 26] and requires a lot of first hand experience. Acquiring this can be difficult. Andraos introduces the term *chefist* [5, p. 19]. They do things based on previous experiences and not by using an academical approach.

This clearly shows the field of synthesis planning is not formalized. Only a few formalized approaches exists. One proposed by E. J. Corey [11]. He introduced the concept of retrosynthetic analysis. The idea is to break the target compound into simpler and easier to handle compounds. Then one can apply the same procedure to the new molecules and end up with simple input compounds. Each splitting of the molecule represents a reaction.

1.1.6 Retrosynthetic Analysis

The concept *retrosynthetic analysis* is defined and formalized by E. J. Corey [11]. The core idea is to recursively select bonds and break a molecule into simpler molecules and repeat this process until the molecules are commercially available or simple to produce. These ideas are widely recognized as one of the best frameworks for synthesis planning and are still widely used.

As S. J. Danishefsky [45] writes:

> It would be improbable, to say the least, to plan the synthesis of a complex target structure through a cognitive process which is fully progressive in nature. Given the stupefying number of ways in which one might begin and proceed, it would seem unlikely that the human mind would go anywhere but in the retrosynthetic direction wherein, at least generally, complexity is reduced as the planning exercise goes on.

But still this approach have a high complexity. Several strategies exist [26, p. 5] for selecting the bonds in bondset, most notably the following:

- Functional Group Oriented

- Skeleton Oriented
- Building Block Oriented
- Method Oriented

Often one does not only use one strategy, but mixes several together. When chemists use the term *skeleton methods* this still means looking at functional groups and how they are positioned with respect to the skeleton.

1.1.7 Cost Functions

When dealing with synthesis plans we would like to compare them against each other. Several metrics exist in literature [4, 5, 20, 22, 26]. These metrics are often divided into two groups: *tree based* and *tree based with molecular information*.

The tree based approach only look at the structure of the synthesis plan. Most notable is the Hendrickson's Convergency Index [22]. This will be examined in more detail in Section 1.1.8.

The other group also take molecule properties into account. These cost functions are often more complicated, but can take care of more corner cases, by not always preferring a balanced tree. One example of this is the Starting Material Weight Index, taking the weight of the starting compound into account.

In recent years metrics trying to pick the most green synthesis plan have been increasingly popular [5]. These aim to limit the dangerous molecules and expensive catalysts.

1.1.8 Hendrickson's Convergency Index

The core idea with Hendrickson's Convergency Index is to assign low scores to balanced synthesis plans and high scores to linear plans. The way it works is by calculating the *external path length* [29] of the synthesis tree.

The *external path length* is known from computer science to be the sum of the path lengths from the root to all leaves. A small example can be seen in Figure 7. Here each path is given a unique color. If we look at the green path, this contributes with one twice, the red path contributes with two also. Making that subtree contributing with a total of four.

From this example we see that a balanced tree will be assigned a lower cost than an unbalanced tree.

Another way to view this, is to fix the number of nodes and calculate the score for the balanced tree, then no matter what permutation of the nodes we examine the score does not get lower. This is seen in Figure 8.

Lets formalize the method. Let k denote the number of input compounds and l_i denote the path to the ith input compound. The external path length is then calculated as:

$$EPL = \sum_{i=1}^{k} l_i \tag{1}$$

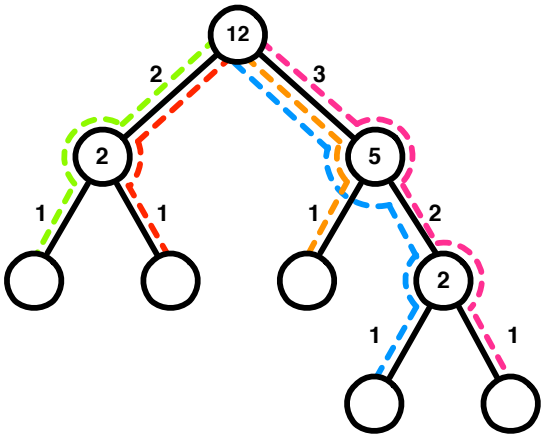


Figure 7: A small tree with each path from the root to the leaves marked. The numbers on the edges count how many paths use that edge. The numbers in the nodes is an accumulated count of children nodes and the outgoing edges.

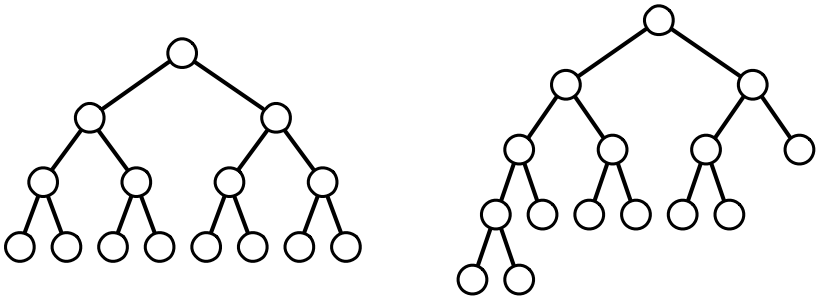


Figure 8: The same number of nodes, but different cost. The left tree has a cost of 24 while the right tree has a cost of 25.

1.1.9 Starting Material Weight

The cost function *Starting Materials Weight*, aims to introduce large molecular structures as late as possible in the synthesis plan. Because chemical reactions involving large molecules have shown to have a low yield. By penalizing each time a carbon atom passes through a reaction, large molecules are penalized more than small molecules when passing through reactions.

Plans where large molecules are introduced late will have a better score than plans where large molecules are introduced early in the synthesis plan.

Formally n_i denotes the number of carbon atoms in input compound i, l_i is the path length from the root to the ith input compound. The cost function then becomes:

$$W = \sum_{i=1}^{k} n_i x^{l_i}$$

Where x is the retro reaction yield, defined as $x = \frac{1}{y}$, where y is the yield, which is often assumed to be 80% [22].

1.2 Algorithms and Methods

With the formal definitions of the chemistry notation used throughout this thesis we also need to define some things from the computer science domain.

1.2.1 Canonical Representation

The notion of a canonical representation is a unique representation of an object. One example is a canonical representation of a graph as a string, is a string that uniquely represents this graph. This is particularly useful when dealing with molecules, since we only want one unique representation of a molecule.

The creation of a canonical form of an object differs from object to object. A canonical representation of a polynomial could for instance be done by writing the terms in decreasing powers. Hence this would be canonical

$$x^3 + 2x^2 + 20$$

While this would not be canonical

$$x^3 + 20 + 2x^2$$

even though they reflect the same polynomial. This is a rather trivial case, but often the canonical form can be quite difficult to construct. This is true when dealing with graphs and molecules.

1.2.2 Isomorphism for Molecules

We say two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijection $f : V \to V'$ in such a way that $(u, v) \in E$ if and only if $(f(u), f(v)) \in E'$. Or said with other words, if we can map all nodes from G to G' while maintaining the edges in G and G' then G and G' is said to be isomorphic. A lot of very powerful tools exists for this problem, one of the best known is the NAUTY package.

The idea behind NAUTY is to exploit the fact that two labeled objects which are isomorphic must be identical if they are canonically labeled. Hence NAUTY constructs a canonical labeling using *equitable partitions* (`http://pallini.di.uniroma1.it/Introduction.html`).

For use in synthesis planning we need to introduce bondsets to the isomorphism check. The usual definition of isomorphic molecules needs to be extended to take bondset bonds into account. Or more generally speaking labeling or coloring of edges.

Hence we can not use NAUTY, since they do not support labeling or coloring of edges. One simple transformation is to add new nodes with a label that corresponds to the labeling of the edge.

The bondset could be transformed into new nodes, there is also a different approach to this, namely VF2.

The core idea behind VF2 is to traverse a tree. The algorithm is given two graphs: V and V'. It then maps one node in V to V', and tries to map another node by following the edges. If no node is found, then the algorithm backtracks and tries another possibility.

For instance it is easy to see that the two molecules in Figure 10 are isomorphic. The two molecules are merely rotated differently.

Figure 9: Converting graph with labeled edges into graph without labeled edges.

Figure 10: Two isomorphic molecules, each with one bond in bondset.

While the molecule in Figure 11 are clearly not isomorphic since there does not exist a bijection.

Figure 11: Two non isomorphic molecules, both with two bonds in bondset.

1.2.3 Yen's Algorithm

Yen's algorithm is a well known algorithm for finding the K shortest loop less paths from the source to the sink in a graph. The algorithm was originally created by Jin Y. Yen in 1971 [46]. This section will cover the theory behind the algorithm, as well as containing a small example for better understanding the algorithm. The algorithm will be used as a basis for the later work done in the thesis.

Problem Definition

There are two main types of the K shortest paths problem. The two main types differ in whether loops are allowed or not.

We focus on the type where loops are not allowed, which is also the version where Yen's Algorithm can be applied.

Given a graph $G = (V, E)$, where each edge $e \in E$ has a cost associated. At this point we distinguish whether or not the cost associated with a edge can be negative or not.

We focus on the version where edges always have a non-negative cost associated. This models the problems we encounter later on in the thesis.

The final problem definition then becomes: Given a graph $G = (V, E)$, where each edge $e \in E$ has a non-negative cost associated, find the K shortest loopless paths, from a source node, s, to a sink node, t, in the graph G.

Notation and Definition

Given a graph $G = (V, E)$ we introduce the following notation.

- $N = |V|$, N is the number of nodes in G.
- (i) where $i = 1, 2, ..., N$, are labels for the nodes in G, where $1 = s$ is the source and $N = t$ is the sink.
- $A^k = (1)—(2^k)—(3^k)—\cdots—(Q_k^k)—(N)$, where $k = 1, 2, ..., J$, is the shortest path from (1) to (N) where (x^k) is the xth node on the kth path and (Q_k^k) is the penultimate node of the kth shortest path.
- d_{ij} where $i \neq j$ is the cost of the edge between (i) and (j). If this edge exists d_{ij} is a finite number, if the arc does not exist $d_{ij} = \infty$.
- A_i^k where $i = 1, 2, \cdots, Q_k$ is a set of alternative paths from A^{k-1} at node (i). Or put differently: the path in A^{k-1} from (1) to (i) coincides with the path in A^k from (1) to (i). Then the path in A_i^k from (i) to (Q_k) is different from any of the $(i+1)$st nodes in A^j where $j = 1, 2, \cdots, k-1$, these must of course have the same paths from (1) to (i) as A^k does.
- R_i^k is the root path of A_i^k, where the root path is defined to be path that coincides in A_i^k and A_i^{k-1}.
- S_i^k is the spur path of A_i^k. The spur path is the path from the last node that coincide with A_i^{k-1} to (N): $(i^k)—\cdots—(N)$ in A_i^k.

Algorithm

The algorithm consists of several iterations. We differentiate between the first iteration and the $2^{nd}, 3^{rd}, ..., K^{th}$ iterations.

The First Iteration

The first iteration is to determine A^1, which is the shortest path from s to t in G. Since finding the shortest path in a graph is another very well known problem, it is not within the scope of this algorithm to solve this problem, it just relies on the problem being solved elsewhere. Any efficient algorithm for finding the shortest path in a graph can be used. In our implementation Dijkstra's Algorithm is used to compute the shortest path between two nodes in the graph [14].

The K-1 Iterations

In the iteration $k = 2, 3, ..., K$ we wish to determine A^k which is the k best path. In order to find A^k, the shortest paths $A^1, A^2, ..., A^{k-1}$, must have been previously determined. The k shortest path A^k is determined in the following way:

1. For i in the sequence $1, 2, ..., Q_{k-1}$.
    a) Check if a subpath consisting of the first i nodes of A^{k-1} match with any of the i first nodes of previously computed shortest paths A^j for j in the sequence $1, 2, ..., k-1$. If so, set $d_{iq} = \infty$ where q is the $(i+1)$ node of the A^j shortest path. This forces the subpath to deviate from A^j. Note that the changes to edge costs done in iteration k only affect iteration k, any changes made to values are reset before iteration $k+1$ starts.
    b) Using a shortest path algorithm, for example Dijkstra's Algorithm, find the shortest path from node (i) to node $(N) = t$, disallowing it to pass through nodes that are already in the path. Note that the subpath from (1) to (i) is R_i^k denoted the root path, and that the subpath from (i) to (N) is S_i^k denoted the spur path. If more than one subpath from node (i) to node $(N) = t$ have a minimum length, pick an arbitrary one.
    c) Find A_i^k by concatenating R_i^k and S_i^k, then add A_i^k to B.
2. B is the list of potentially shortest paths. From B take the path which has the minimum cost, call this plan A^k.
3. If $k \geq K$ stop, else continue to iteration $k+1$.

This step can be envisioned like a branching step, and comes from the observation that A^k is a deviation from the A^j shortest paths, for j in the sequence $1, 2, ..., k-1$. Put in other words A^k must coincide with A^j for the first $m \geq 1$ nodes, then deviate, and not pass through the same node twice. One of these deviations from A^j is then the next shortest path A^j.

Pseudocode

The pseudocode for our implementation of Yen's algorithm is shown in Algorithm 1.

Algorithm 1 Algorithm which calculates the K shortest loopless paths in a graph.

```
1:  function YenKSP(Graph, source, sink, K)
2:      A_0 ← Dijkstra(Graph, source, sink)
3:      B ← ∅
4:      for k from 2 to K do
5:          for i from 1 to |A^{k-1}| − 1 do
6:              rootPath ← A^{k-1}_{1,i}
7:              for all A^j ∈ A do
8:                  if rootPath = A^j_{1,i} then
9:                      d_{iq} ← ∞|q ∈ 1 → A^j_{i+1}
10:                 end if
11:             end for
12:             spurPath ← Dijkstra(Graph, sp)
13:             fullPath ← rootPath + spurPath
14:             B ← B + {fullPath}
15:         end for
16:         if B = ∅ then
17:             Break
18:         end if
19:         Sort(B)
20:         A_k ← B_0
21:         Pop(B)
22:     end for
23:     return A
24: end function
```

Runtime analysis

Since the runtime of the algorithm depends on the algorithm used for computing the shortest path in the graph, we will assume that Dijkstra is used, like in our implementation.

The runtime of the version of Dijkstra's Algorithm we use is $\mathcal{O}(V \cdot \log V + E)$.

Yen's algorithm makes $K \cdot l$ calls to Dijkstra's Algorithm for computing the spur paths, where l is the length of the spur paths. The worst case for l is V. The total runtime of Yen's algorithm then becomes $\mathcal{O}(K \cdot V \cdot (E + V \cdot \log V))$.

Example of Yen's Algorithm

The aim of this section is to familiarize the reader with Yen's algorithm through a small example. This example is inspired from the article about Yen's algorithm on wikipedia (`http://en.wikipedia.org/wiki/Yen's_algorithm`). This example is meant to illustrate each step in the algorithm. Given the graph G, find the 2 shortest paths from node C to H.

See Figure 12 for depiction of the graph G.

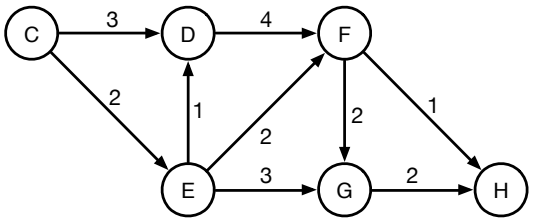


Figure 12: A directed weight graph G.

First Iteration

In the first iteration the shortest path from C to H in G must be determined. This can be done by any classic shortest path algorithm. For this example Dijkstra is used. The call to Dijkstra's Algorithm will be $Dijkstra(G, C, H)$. We will not cover how Dijkstra's Algorithm works in this example. The shortest path in G found by Dijkstra's Algorithm becomes:

$$A^1 = C - E - F - H$$

See Figure 13 for depiction of the shortest path in G.

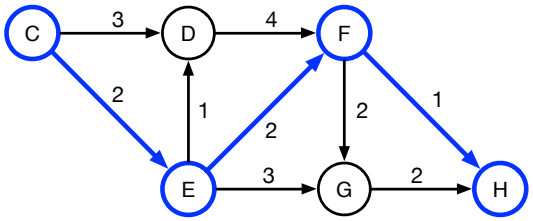


Figure 13: The shortest path from node C to H in G, $C - E - F - H$, marked with blue.

Second Iteration

In the second iteration we wish to find the second shortest path, A^2 in G.

This is done by branching along the previously found shortest path. Formally, we have the previously found shortest path A^1, which we want to branch along. We start in the first branching step with $i = 1$. The root path is determined to the node C, and we need to check if any other paths in A has a subpath that matches. By definition the previously found shortest path always match, and since A does not contain more paths, we only have a match in A^1.

Therefore we set the cost $d_{i,i+1}$ between node (i) and $(i+1)$ in A^1 to ∞, in this example node (i) and $(i+1)$ in A^1 corresponds to edge $C - E$. Setting the cost to ∞ is similar to removing the edge completely. See figure Figure 14 for how the graph looks after the first branching step.

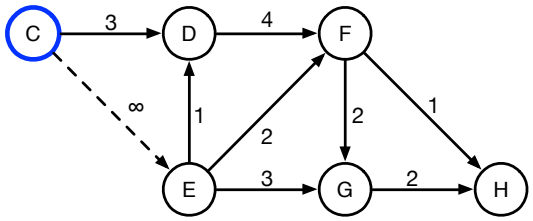


Figure 14: The first branching step in G.

Now Dijkstra's Algorithm is applied to find the shortest path from node (i) to H. The result is called the spur path. The spur path S_1^2 is determined to be $C-D-F-H$. The resulting shortest path is then found by concatenating the root path and the spur path. The shortest path for the first branching step A_1^2 is $C-D-F-H$ which is added to B with cost 8. The second branching step $i=2$, the root path is determined to be the first two nodes in A^1, which is $C-E$.

We search for a subpath $C-E$ in A, again only A^1 match. Again we wish to disallow the use of the edge between the (i) and $(i+1)$'th node in A^1. Therefore the edge cost $d_{i,i+1}$ in A^1 is set to ∞. See the resulting graph in Figure 15.

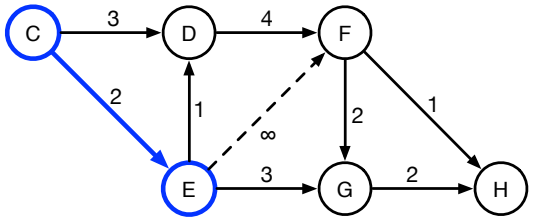


Figure 15: The second branching step in G.

Dijkstra's Algorithm is then applied to find the shortest path from node (i) to H, called the spur path. The spur path S_2^2 becomes $E-G-H$. The total path A_2^2 becomes $C-E-G-H$, which is added to B with cost 7.

The last branching step in this iteration, proceeds like the two previous. First the root path R_3^2 is determined to $C-E-F$. The edge cost between node F and H is set to ∞, as a result of the subpath checking.

See Figure 16 for the graph after the branching step is applied.

The spur path S_3^2 is the shortest path from node F to H, and is found by Dijkstra's Algorithm. S_3^2 becomes $F-G-H$, which is concatenated with the root path, so A_3^2 becomes $C-E-F-G-H$ and is added to B with cost 8.

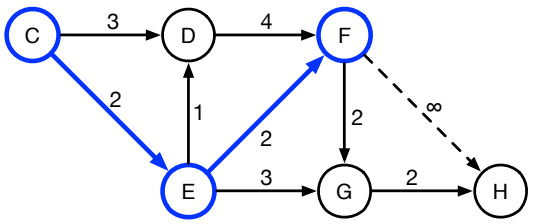


Figure 16: The third branching step in graph G.

At this point B contains the following paths:

Path	Cost
$C-D-F-H$	8
$C-E-G-H$	7
$C-E-F-G-H$	8

Table 2: The 3 best plans in G found by Yen's algorithm.

Now the second shortest path A^2 is chosen to be $C-E-G-H$, which is deleted from B. We have now found the two shortest paths from C to H.

The second shortest path is depicted in Figure 17.

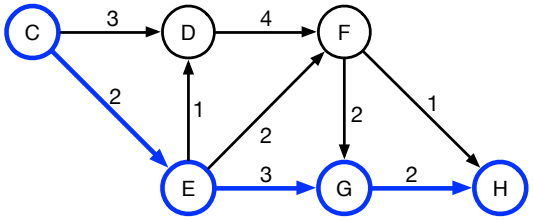


Figure 17: The second shortest path in G.

If we wanted to find the 3 shortest paths, we would do another iteration where the branching was done based on A^2, and the subpath checking was done with respect to both A^1 and A^2.

1.2.4 Petri Nets

Petri nets is a mathematical modeling language, often used within analysis of distributed systems that has a nondeterministic behavior. Petri nets is a classical research area within computer science, with a lot of application areas.

Petri net consists of places, transitions, and arcs. Arcs run between places and transitions, but never between two places or two transitions. We define input places for a transition t_1, to be places which have edges pointing into transition t_1. The edges going out of transition t_1 connects to output places.

Places contain a discrete number of tokens. Any distribution of tokens in the petri net is called a *configuration*.

A configuration can be expressed by a $|P|$-tuple, where P represent the places in the petri net. An element e_i in the tuple, describes how many tokens there are on place P_i, where $i \in 1, 2, ..., |P|$.

A transition determines how tokens can be moved around in the petri net. A transition may fire if the input places have sufficient tokens. If a transition fire, the tokens are removed from the input places, and inserted on the output places. Firing a transition is an atomic action.

The execution of a petri net is nondeterministic. If multiple transitions is enabled, any one of them may fire.

Some general properties regarding petri nets are:

- A petri net have $|P|$ places.
- All places p_i are numbered $i \in 1, 2, ..., L$.
- The output places for t_1 are denoted out(t_1), the input places is denoted in(t_1).
- Place p_1 is input compound for the following transitions in(p_1), likewise the transitions for which p_1 is output place are denoted out(p_1).
- Place p_l is always the root of the petri net.

1.2.5 Petri Net Example

A small example of a petri net is presented below. See Figure 18 for a depiction. The petri net contains 3 places p_1, p_2, p_3, and one transition t_1. p_1 and p_2 are input places for t_1, p_3 is an output place for t_1.

The initial configuration C_1 of the petri net is the following

$$C_1 = (2, 1, 0)$$

Transition t_1 is enabled in configuration C_1. After t_1 is fired, the new configuration C_2 becomes:

$$C_2 = (1, 0, 1)$$

See Figure 18 for before and after depiction of the petri net.

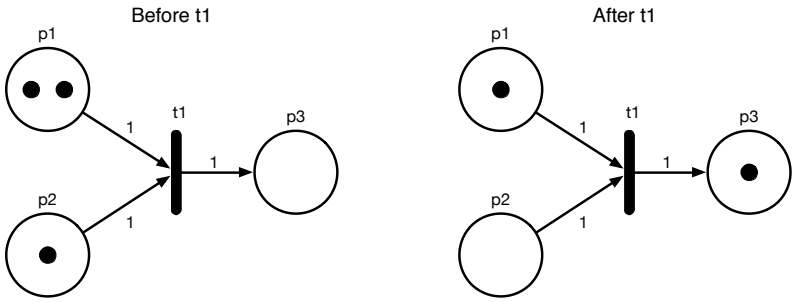


Figure 18: A Petri Net before and after transition t_1 fires.

1.2.6 Hypergraphs

In classical graph theory a graph consists of nodes and edges. It is often written as

$$G = (V, E)$$

where each edge connects two nodes, $e = (u, v)$, where u and v might be the same node. One can generalize this into so called *hypergraphs*. In classical computer science literature hypergraphs are often overlooked, hence we deemed it reasonable to give a short introduction.

A hypergraph, also called an undirected hypergraph, can be thought of as a normal graph with some extra properties. Instead of edges we have hyperedges. A hyperedge can connect any subset of nodes. A hypergraph is often denoted:

$$\mathcal{H} = (\mathcal{V}, \mathcal{E})$$

where $\mathcal{V}$ is the nodes and $\mathcal{E}$ the edges.

A simple example of a hypergraph could be

$$\mathcal{H} = (\{a, b, c, d, e, f\}, \{\{a, b, c\}, \{c, d, e\}, \{b, e, f\}\})$$

This is seen in Figure 19.

A common type of hypergraphs are *directed hypergraphs*. In a directed hypergraph the edges are denoted:

$$e = (\text{Tail}(e), \text{Head}(e))$$

where $\text{Tail}(e) \subseteq E$ and denotes the tail nodes of the hyperedge. The head of the edge is denoted $\text{Head}(e)$ and formally written as $\text{Head}(e) \in E \setminus \text{Tail}(e)$.

The cardinality of a hyperedge is simply $|e| = |T(e)| + 1$. If the cardinality of an edge is 2 it can be viewed as a normal graph edge.

In this thesis we will focus on directed hypergraphs with hyperedges that have a cardinality 2 or 3.

Backward- and Forward Star

The term backward- and forward star is defined for a node v in the following way:

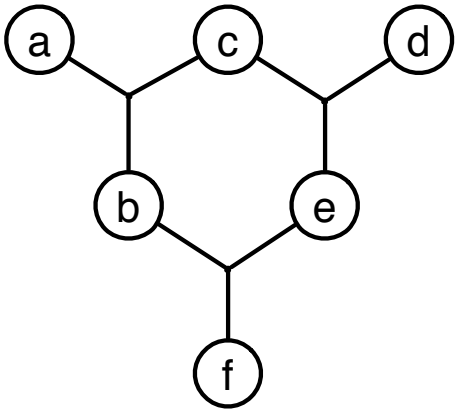


Figure 19: A simple hypergraph.

$$BS(v) = \{e \in E | v \in \text{Head}(e)\}$$

$$FS(v) = \{e \in E | v \in \text{Tail}(e)\}$$

The backward star describes all ingoing hyperedges to v, forward star describes all outgoing hyperedges to v.

1.2.7 Synthesis Plans as Hypergraphs

In order to model synthesis plans in a mathematical way we use hypergraphs. To translate a synthesis plan into a hypergraph a simple conversion is required.

Lets say we have a synthesis plan with a set of reactions, denoted R. All molecules in the synthesis plan can be seen as a set, denoted M. We denote the target compound $t \in M$ and all input compounds as $I \in M$. To translate the synthesis plan into a hypergraph we do the following. First we construct a new hypergraph $H = (V, E)$. Copy all molecules M into H:

$$V = M$$

The internal nodes in the hypergraph are then:

$$M \setminus \{t\} \setminus I$$

where t is the root of the synthesis plan and I are the leaves.

Each reaction will become an hyperedge. The educts of the reactions are the tail of the hyperedge and the product is the head of the hyperedge.

1.2.8 Graph File Formats

For this project we need to store molecules as files. For this several formats exists. One of these would be to build a SMILES or InChI parser. To do this we would need to extend the formats to support bondsets. Another approach

would be use a graph format and just interpret the content as a molecule. This was the approach we decided to use.

Several graph formats exists and we give a short introduction to these and our own format SPM.

Trivial Graph Format etc.

These formats which originates in the computer science literature are not as simple as they would like to be. They are all capable of encoding a lot of information and often this results in more or less obscure syntax that in our case will make it difficult for the chemist to easily identify the molecule and modify the bondset. These formats have open source implementations but often the quality leaves a lot to be wished for, hence we would rather stay clear of these.

Synthesis Plan Molecule

No existing formats had the features that was required to model molecules and a bondset. Hence a new format was devised. The major focus in this format is to make the connections between atoms very clear and the definition of the bondset easy to modify. The SPM is, as the other chemical formats, a hydrogen suppressed representation. Furthermore the SPM format is not a canonical representation as the InChI format is.

The SPM format consists of three sections. Each section has only one responsibility. This makes the format extendable like the InChI format, but with fewer constraints for the sections. One problem with this format is that a molecule encoded as an SPM file is larger than a single string of text.

The sections are the following:

1. Graph
2. Bondset
3. Atoms

The Graph section defines connections betweens the individual atoms. Each atom is represented as an unique number with bonds between. The bond notation is the same as in SMILES; a single bond is written as a dash (`-`), a double bond this is noted as a equal sign (`=`) and a triple bond as a hashmark (`#`). Cycles in the input molecules are easily modeled, a line can start and end with the same atom: `1-2-3-1`, this would create a triangle. One does not need to worry about valence, since hydrogen will just be added where needed.

The Bondset section specifies which bonds are in the bondset. This is done using the same numbers as in the Graph section. Bonds in the Bondset section is always specified as a dash to simplify the input.

The Atoms section specifies the type of the atoms. Atoms defaults carbon if not specified to other type. This ensures that we only need to specify the type of atoms with a different name than carbon.

Ethanol could be encoded like this:

```
Graph:
1-2-3
```

```
Bondset:
1-2
Atoms:
1: O
```

Listing 1.1: Ethanol

And benzene like this:

```
Graph:
1-2=3-4=5-6=1
Bondset:
1-2
Atoms:
```

Listing 1.2: Ethanol

The SPM also have some problems that primarily is a result of the design choices made. To keep the complexity down, no rules are made on how to arrange molecules, this means that the format is not canonical. In the previous ethanol example this is easily seen if we just label atom number 3 with `O` instead of atom number 1. This makes the format unsuitable for uses in databases etc. Since no rules exist on how to encode a molecule in the SPM the files can easily be quite large for medium sized molecules. This size of the input file can make it difficult to figure out what actually does what.

Like InChI SPM does not model aromatic bonds. These bonds must be converted into an single-double-bond alternating cycle.

2 Representing all Synthesis Plans

A synthesis plan consists of reactions, which can be divided into two kinds. *Construction reactions*, which basically create a carbon-to-carbon bond skeletal bond, or *refunctionalization reactions*, which functionalize the molecule without changing the skeleton of the molecule. To simplify the model, we only consider construction reactions, that again can be subdivided into two subcategories, namely *affixations* that merge separate intermediate molecules and *cyclizations*, that create skeletal rings.

This choice of modeling leads to a classic way of representing synthesis plans. Synthesis plans represented by a unary-binary tree [22]. In this unary-binary tree, affixations are modeled by a split (binary edge), and cyclizations are modeled by a single edge. The leafs in the unary-binary tree represent the input compounds, and the root represents the target compound. The bonds which are fixed in the construction reactions are called the bondset. The tree describes in which order the input compounds are merged and cycles are closed.

The number of different synthesis plans are equal to the number of permutations the bonds in the bondset can be fixed. Given a bondset B, it is possible to create $|B|!$ different unary-binary trees.

If unary-binary trees are chosen as the data structure for representing all possible synthesis plans, enumerating the K Best synthesis plans would require to generate all possible synthesis plans. This would render the use of clever K shortest paths algorithms pointless since it would be possible to simply rate each synthesis plan as it is created and keeping track of the K best one.

Another way to represent synthesis plans are hypergraphs, which are formally defined in [15]. A simple example of the projection from a unary-binary tree to a hypergraph is shown in Figure 20 and Figure 21.

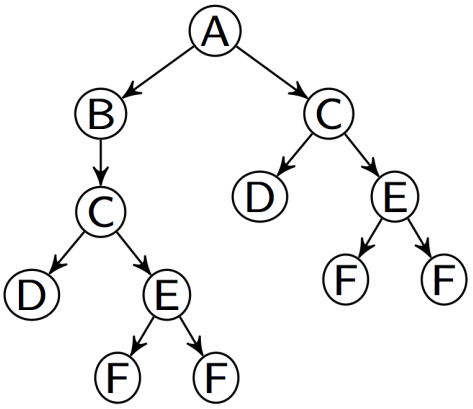


Figure 20: Unary-binary representation of synthesis plan.

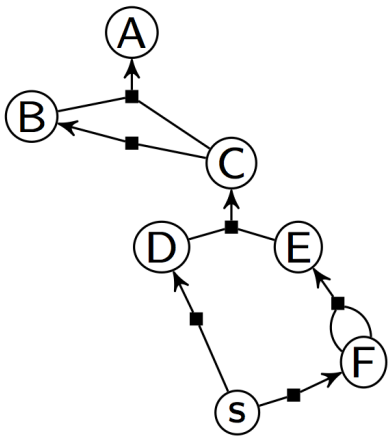


Figure 21: Hypergraph representation of synthesis plan.

The idea for a more effective data structure for representing all synthesis plans is to create a superposition of all synthesis plans, represented as a hypergraph.

This superposition of all synthesis plans is denoted the Hypergraph of Synthesis Plans (HoSP). The aim is to analyse this datastructure and effectively find the K Best Synthesis Plans.

Depending on how the superposition of the synthesis plans are defined, different HoSP are created. In this section different methods for constructing the HoSP are explained and presented.

2.1 Molecular Equality

Recall that the HoSP is defined as the superposition of all synthesis plans. How the equality of molecules is defined, determines how the superposition is constructed, which affects important properties of the HoSP.

Three different methods for defining molecular equality is defined in this section and allow for a distinction between the possible ways to construct the superposition.

Recall that a molecule is represented by an undirected node labeled graph. A bondset is defined as the bonds fixed in the synthesis plan, it defines an edge labeling of the molecule.

All our equality measures are depicted in Figure 26.

Graph Equality

Two molecules `m1` and `m2` are equal if the undirected node labeled graph, disregarding the edge labeling defined by the bondset, for `m1` and `m2` is isomorphic.

Formally we write:

$$\mathrm{m1} =_G \mathrm{m2}$$

Since `m1` and `m2` represent actual molecules, it is possible to create a canonical text representation using canonical SMILES.

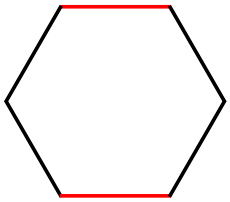


Figure 22: A

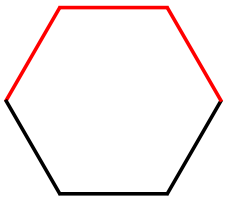


Figure 23: B

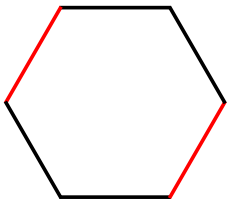


Figure 24: C

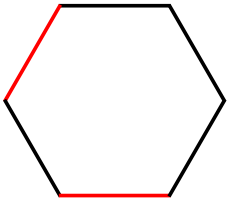


Figure 25: D

Figure 26: This shows the different equality classes for molecular equality in the HOSP, red bonds are in the bondset. A, B, C and D are equal if $=_G$ is used. A and C are equal if $=_E$ is used. A, C and D are equal if $=_N$ is used.

Number equality

Two molecules `m1` and `m2` are equal if the undirected node labeled graph are isomorphic, and the two bondsets `b1` and `b2` are of equal size.

We write:

$$\text{m1} =_N \text{m2}$$

Again it is possible to determine the equality of the two molecules by having a canonical string representation of the two molecules, and simply counting the number of bonds in each bondset.

Edge Equality

Two molecules `m1` and `m2` are equal if the undirected node labeled graph, and the edge labeling defined by the bondsets `b1` and `b2` are isomorphic.

Formally denoted as:

$$\text{m1} =_E \text{m2}$$

Since edge labels are also regarded when determining equality, a canonical string representation like canonical SMILES cannot directly be applied. Instead a algorithm for solving graph isomorphism is used. The current best method [34] is not used since it does not support edge labels. A possible way to work around this is to create an additional node on the edges as a way of representing edge labels. This however means increasing the size of the graph. Instead VF2 is chosen [8].

2.2 Fixing *All* Bonds in the Bondset

A classic approach to synthesizing a molecule for a chemist is the following: Given a target molecule, construct a bondset that defines input compounds. Construct the molecule using *all* input compounds.

The result of this approach is a HoSP where for any of the bonds in the bondset we will have a reaction that fixes that specific bond. This would be represented by a hyperedge that fixes this bond in all possible synthesis plans.

2.2.1 Hypergraph Construction

The exploration of all synthesis plans happens in a top-down manner, starting with the target molecule and recursively breaking bonds. To avoid exploring a molecule twice, a check is made whether or not H contains an equal molecule that has already been explored. Each molecule is represented by a labeled undirected graph. The function *bondset* returns the set of edges which are in the bondset. There is only one bondset per molecule.

For each recursive call to `Expand` for a given molecule m, if the bondset for m is empty, we are done. Otherwise each bond in the bondset is removed one at a time. The next step is to apply connected component analysis on the resulting graph. The idea here is to se how many new smaller graphs were created, each of these graph is called a connected component. The connected components, after the removal of the bond, is denoted as new molecules. For each new molecule $m1$, we check if another molecule $m2$ is equal, with the defined equality operator, and has already been expanded. If there exists such a molecule $m2$, $m1$ is replaced with $m2$.

We then insert the edge $(\text{new molecule}, m)$ into H. Finally, for all new molecules which has not been expanded, call `Expand` on that molecule.

Algorithm 2 HoSP-Edge

Require: $bondset(targetMolecule)$ is set

```
1: function Expand-Edge(m)
2:     if |bondset(m)| = 0 then
3:         return
4:     end if
5:     for all bond b ∈ bondset(m) do
6:         mCopy ← copy of m
7:         Break b in mCopy
8:         newMolecules ← connected components in mCopy
9:         for all m1 ∈ newMolecules do
10:            Check if H has a molecule m2 where m1 =_E m2
11:            if m2 exists then
12:                Replace m1 with m2
13:            end if
14:        end for
15:        Insert edge(new molecules, m) into H
16:        for all m1 ∈ newMolecules do
17:            if m1 has not been expanded then
18:                Expand(m1)
19:            end if
20:        end for
21:    end for
22: end function
```

2.2.2 Uniqueness of Plans

An important question to consider is whether or not the synthesis plans represented in the HoSP are unique.

This is not the case with this method for constructing the HoSP, simply because molecules with the same chemical structure, ie. represented by the same undirected labeled graph, appears multiple places in the HoSP. Hence one of the major drawbacks from the state space approach has been solved.

A small HoSP is depicted in Figure 27, two identical plans are marked with red and blue. The reason the two plans are not merged to one synthesis plan in the HoSP is that the red plan has alternative ways for being synthesized than the blue than, in the intermediate molecule with 4 carbon atoms.

Bonds in the molecules are marked with green.

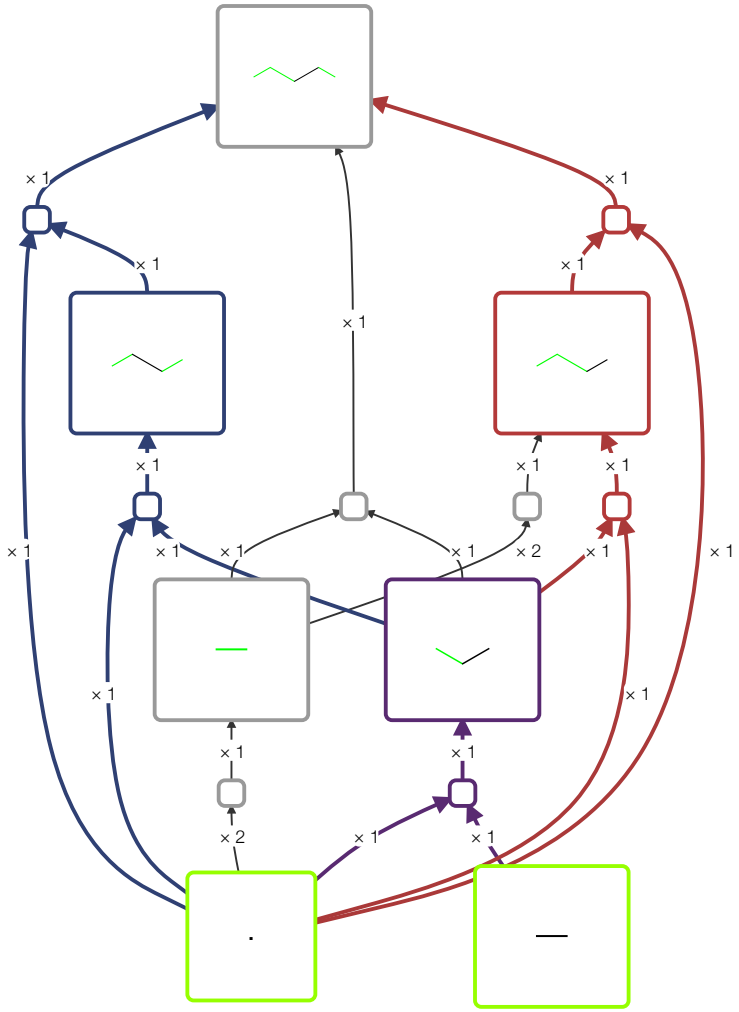


Figure 27: Two chemically identical synthesis plans, marked with blue and red respectively. The input compounds are marked with green. Purple is where the two plans overlap. The bondset of the molecules are highlighted with green.

2.3 Bondset Defining Input Compounds

In the previous method, the bondset defined a set of input compounds where all input compounds were used to create the target molecule. In this method the bondset also defines a set of input compounds, but all input compounds are not necessarily used to create the target compounds. It is allowed to use a subset of the defined input compounds.

In this method the equality of molecules is also defined based on $=_G$. The motivation for doing this is quite clear. In the previous method, similar molecules may be synthesized in different ways, clearly one must be optimal. By merging all similar molecules, all synthesis plans which includes that molecule may use the optimal way to synthesize exactly that molecule.

The fact that similar molecules are only represented once in the HoSP has additional benefits. All synthesis plans represented in the HoSP are per definition unique. Given a synthesis plan in the HoSP, any other path in there must include at least one other molecule, because molecules are only represented once, and hence be another synthesis plan.

Because molecules are merged based on the molecule structure, this has the following implications: Imagine two molecules, A and B, that have the same structure. Molecule A has no bonds in the bondset, i.e: is an input compound. Molecule B on the other hand have 2 bonds left in the bondset. Since A and B have the same structure, they are represented by the same node in the HoSP. Because A is an input compound, it is never necessary to synthesize B. The implication of this observation is that synthesis plans represented by the HoSP width edge doe not necessarily fix all bonds in the bondset.

2.3.1 Hypergraph Construction

The algorithm for constructing the HoSP is similar to the previous method. There are two main differences:

- Each molecule can now have a set of bondsets associated. Therefore the function `bondset` is replaced with `bondsets`, which now returns a set of bondsets.
- Molecule equality is defined by $=_G$.

This method is denoted HoSP-Graph, and is described in Algorithm 3.

Algorithm 3 HoSP-Graph

Require: bondset(targetMolecule) is defined

```
1: function Expand-Graph(m)
2:     bondsets ← bondsets(m)
3:     for all bondset ∈ bondsets do
4:         if |bondset(m)| = 0 then
5:             return
6:         end if
7:         for all bond b ∈ bondset(m) do
8:             mCopy ← copy of m
9:             Break b in mCopy
10:            newMolecules ← connected components in mCopy
11:            for all m1 ∈ newMolecules do
12:                Check if H has a molecule m2 where m1 =_G m2
13:                if m2 exists then
14:                    Replace m1 with m2
15:                end if
16:            end for
17:            Insert edge(newMolecules, m) into H
18:            for all m1 ∈ newMolecules do
19:                if m1 has not been expanded then
20:                    Expand(m1)
21:                end if
22:            end for
23:        end for
24:    end for
25: end function
```

2.3.2 Uniqueness of Plans

As discussed previously, the HoSP constructed by HoSP-Graph per definition represents unique synthesis plans. See Figure 28 for an example of a HoSP built with this method.

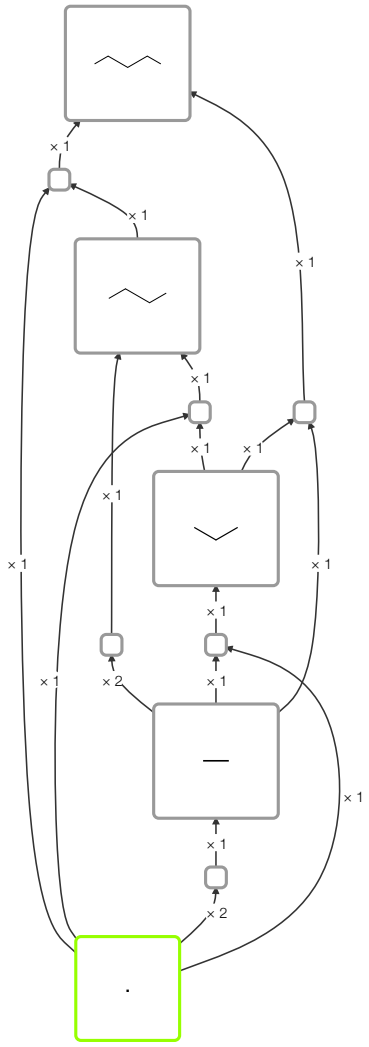


Figure 28: Small example of HoSP created with HoSP-Graph method.

2.4 An In-between Method

A third method was developed, denoted HoSP-Number.

This method is identical to the previous method, except equality of the molecules is defined by $=_N$. However this method is not chemically motivated like the previous method. It is really a method that lies between the two previous methods defined.

HoSP-Number always finds plans using $|b|$ reactions. This does not mean that the bondset specified for the target molecule is respected. Recall that $=_N$ defines equality as two molecules being equal if they have the same molecular structure and the same number of bonds in the bondset.

2.4.1 Hypergraph Construction

Algorithm 4 HoSP-Number

Require: $bondset(targetMolecule)$ is set
1: **function** Expand-Number(m)
2: $bondsets \leftarrow bondsets(m)$
3: **for all** $bondset \in bondsets$ **do**
4: **if** $|bondset(m)| = 0$ **then**
5: **return**
6: **end if**
7: **for all** $bond\ b \in bondset(m)$ **do**
8: $mCopy \leftarrow$ copy of m
9: Break b in $mCopy$
10: $newMolecules \leftarrow$ connected components in $mCopy$
11: **for all** $m1 \in newMolecules$ **do**
12: Check if H has a molecule $m2$ where $m1 =_{graph} m2$
13: **if** $m2$ exists **then**
14: Replace $m1$ with $m2$
15: **end if**
16: **end for**
17: Insert edge($newMolecules$, m) into H
18: **for all** $m1 \in newMolecules$ **do**
19: **if** $m1$ has not been expanded **then**
20: Expand($m1$)
21: **end if**
22: **end for**
23: **end for**
24: **end for**
25: **end function**

2.4.2 Uniqueness of Plans

Synthesis plans found with this method is unique. Although the same chemical compound is represented multiple places in the synthesis plan. The compounds always have a different number of bonds left. Therefore the same chemical compound represented different places in the HoSP can never produce the same subtree in the HoSP, therefore plans are going to be unique.

See Figure 29 for an example of the HoSP generated with HoSP-Number.

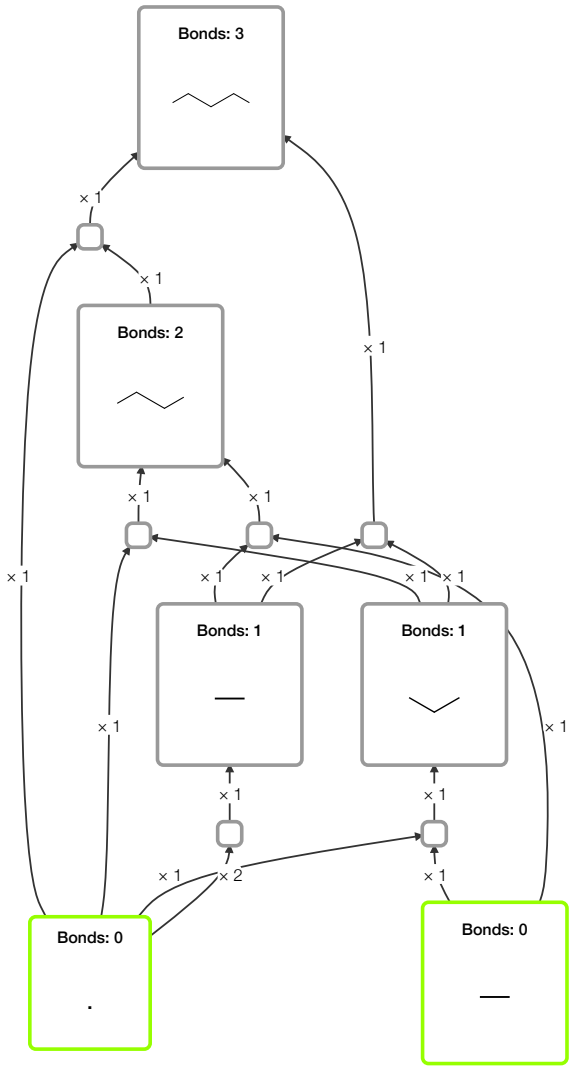


Figure 29: An example of a HoSP created with HoSP-Number.

2.5 All Possible Bondsets

All of the above methods are defined for a single molecule and a single bondset. However when synthesizing a new molecule, a chemist does not always know what a good bondset would be for a specific molecule, hence this task can be extremely daunting. The first idea that comes into mind, is to try all possible bondsets and afterwards try to identify a good bondset.

Let us examine how many possible bondsets there exist for a given molecule `m`. We denote the number of bonds in the skeleton by N.

The number of possible bondsets are:

$$\sum_{i=1}^{N} \binom{N}{i}$$

A real life example is Lysergic Acid Diethylamide (LSD), it has 27 bonds in the skeleton. The number of possible bondsets for LSD is:

$$\sum_{i=1}^{2} 7\binom{27}{i} = 134217727$$

Not only does this imply the HoSP becomes extremely large, it also makes the process of finding a good bondset more difficult, because bondsets of all sizes are compared at the same time.

Therefore we decided to analyse all bondsets of a certain size at a time. The possible bondsets of a certain size b is:

$$\binom{N}{b}$$

All 3 methods, described for exploring a single bondset are also defined for all possible bondsets. All possible bondsets are enumerated for the target molecule, and the HoSP is explored with 1 at a time. Equality of molecules is defined in the same way as previous defined for the individual methods.

We denote the 3 new methods HoSP-Edge-All, HoSP-Graph-All and HoSP-Number-All.

2.6 Comparison of Methods

The 6 different methods for constructing the HoSP are compared below with respect to the following criteria:

- Is it possible to use the cost function *External Path Length*?
- Is it possible to use the cost function *Starting Materials Weight*?
- Are the plans represented by the HoSP unique synthesis plans?
- An upper bound on the HoSP, reference point is HoSP-Graph.

Method	External	Weight	Uniqueness	Size of HoSP
HoSP-Edge	✓	✓	– (2)	$\mathcal{O}\left(\lvert H\rvert \cdot 2^{\lvert B\rvert}\right)$
HoSP-Number	– (1)	✓	– (2)	$\mathcal{O}\left(\lvert H\rvert \cdot \lvert B\rvert\right)$
HoSP-Graph	– (1)	✓	✓	$\mathcal{O}\left(\lvert H\rvert\right)$
HoSP-Edge-All	✓	✓	– (2)	$\mathcal{O}\left(\binom{N}{\lvert B\rvert} \cdot \lvert H\rvert \cdot 2^{\lvert B\rvert}\right)$
HoSP-Number-All	– (1)	✓	– (2)	$\mathcal{O}\left(\binom{N}{\lvert B\rvert} \cdot \lvert H\rvert \cdot \lvert B\rvert\right)$
HoSP-Graph-All	– (1)	✓	✓	$\mathcal{O}\left(\binom{N}{\lvert B\rvert} \cdot \lvert H\rvert\right)$

The size of the HoSP when the construction method HoSP-Edge is bounded by $\mathcal{O}\left(\lvert H\rvert \cdot 2^{\lvert B\rvert}\right)$. Each bond can be in the bondset, and the bondset have size $\lvert B\rvert$. We therefore add a factor of $2^{\lvert B\rvert}$ to $\mathcal{O}\left(\lvert H\rvert\right)$. In HoSP-Number, the size of the HoSP is bounded by $\mathcal{O}\left(\lvert H\rvert \cdot \lvert B\rvert\right)$, since each molecule can have at most $\lvert B\rvert$ different number of bonds left to break.

In the methods where we enumerate over all possible bondsets, we add a factor $\binom{N}{\lvert B\rvert}$ since this is the number of possible bondsets.

Remark 1

Since the cost function External Path Length does not meet the requirements regarding an additive cost function. See 3.1.1.

Remark 2

Recall the discussions about uniqueness in the sections regarding the methods HoSP-Edge, HoSP-Graph and HoSP-Number. The same arguments applies to the *All* version respectively.

2.7 Hypergraph Construction Results

In this section we make experiments, by measuring the size of the HoSP constructed with different methods and varying the size of the bondset. The methods tested are HoSP-Edge-All, HoSP-Number-All and HoSP-Graph-All.

The tests are carried out in the following way. For each of the tested methods, HoSP-Edge-All, HoSP-Number-All and HoSP-Graph-All, we investigate what happens to the size of the HoSP, and examine how many plans there are in it. If there exist more than 1000 plans, we stop. We set a time limit of 10 min. on each test. If the test exceeds 10 minutes, nothing is written in the tables. If the HoSP was generated within the 10 min but the 1000 first plans were not found, the size of the HoSP will be in the tables and the number of plans will not.

2.7.1 Molecules

We present 11 molecules with different skeleton, size and complexity.

The molecules in Figure 30 and Figure 31 are linear in their skeleton. Figure 32, Figure 33, Figure 34 and Figure 35 represent different ring formations often found in organic chemistry.

Figure 36 represent the skeleton of estrone which consists of three six rings and a five ring. Figure 37 represent the skeleton of lysergic acid diethylamide (LSD) which also constsis of three six rings and a five ring. Figure 38 represent the skeleton of longifolene. Figure 39 represent the skeleton of resistomycin which consists of five six rings. Figure 40 represent the skeleton of moenocinol a long chain of carbons.

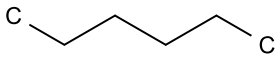


Figure 30: Six carbons connected as a line: `6line`.

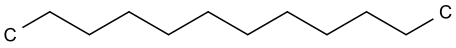


Figure 31: 12 carbons connected as a line: `12line`.

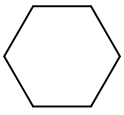


Figure 32: Six carbons connected as a ring: `6ring`.

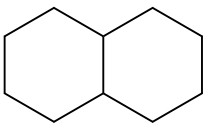


Figure 33: Two six rings: `2x6ring`.

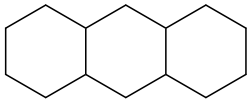


Figure 34: Three six rings: `3x6ring`.

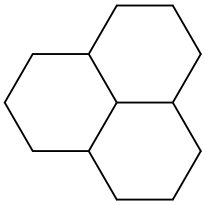


Figure 35: Three six rings in alternative configuration: `3x6ring_alt`.

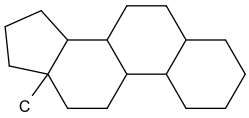


Figure 36: Estrone: `Estrone`.

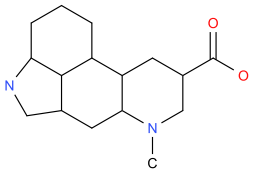


Figure 37: Lysergic acid diethylamide: `LSD`.

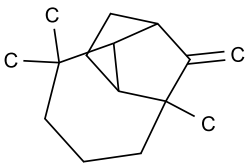


Figure 38: Longifolene: `Longifolene`.

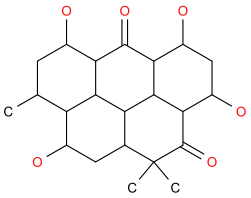


Figure 39: Resistomycin: `Resistomycin`.

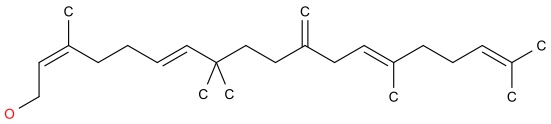


Figure 40: Moenocinol: `Moenocinol`.

2.7.2 HoSP-Edge-All

Recall HoSP-Edge-All defines equality based on $=_E$, which takes into account the edge labels defined by the bondset. By trying all possible bondsets of a given size, this will lead to the largest hypergraphs compared to the two other methods.

Note that the size of the HoSP might decrease in size as the bondset gets larger. This happens for the first two skeletons modeled. As explained

previously this is because of the binomial function which describes how many possible bondsets there are for each molecule. So when we define equality based on $=_E$, all possible configurations of the bondset are represented as a node in the HoSP.

This method is not as efficient as HoSP-Number-All and HoSP-Graph-All. Therefore the largest bondset tested is of size 4.

All tests were carried out on a MacBook Pro, 2.4 GHz Intel Core 2 Duo with 8GB DDR3 RAM.

Table 3 shows the size of the HoSP and number of starting materials for different bondset sizes. Table 4 shows the time required to generate 50 plans.

Bondset size	1		2		3		4	
	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves
6line	$(6, 3)$	5	$(11, 12)$	4	$(15, 24)$	3	$(11, 20)$	2
12line	$(12, 6)$	11	$(41, 60)$	10	$(130, 314)$	9	$(269, 930)$	8
6ring	$(2, 1)$	1	$(9, 6)$	5	$(17, 22)$	4	$(21, 36)$	3
2x6ring	$(5, 4)$	4	$(42, 48)$	17	$(203, 369)$	25	$(659, 1617)$	25
3x6ring	$(6, 5)$	5	$(100, 120)$	39	$(849, 1518)$	107	$(4954, 11660)$	194
3x6ring_alt	$(4, 3)$	3	$(60, 70)$	24	$(532, 905)$	79	$(2985, 6794)$	135
5ring+6rings	$(23, 21)$	22	$(621, 820)$	210	$(8531, 15052)$	1116	$(74459, 169311)$	3453
Lsd	$(33, 28)$	32	$(1151, 1510)$	395	$(22595, 38895)$	3236		
Longifolene	$(18, 16)$	17	$(366, 478)$	126	$(3892, 6680)$	561		
Moenocinol	$(40, 23)$	39	$(729, 1001)$	230	$(7275, 15943)$	539		
Resistomycin	$(34, 31)$	33	$(1407, 1860)$	476	$(31537, 53997)$	4542		

Table 3: Table over size of HoSP when constructing the HoSP with HoSP-Edge. All possible bondsets of size 1-4 tested.

Bondset size	1		2		3		4	
	Time	#Plans	Time	#Plans	Time	#Plans	Time	#Plans
6line	< 1s	3	< 1s	6	< 1s	13	< 1s	14
12line	< 1s	6	< 1s	30	< 1s	50	3s	50
6ring	< 1s	1	< 1s	3	< 1s	10	< 1s	22
2x6ring	< 1s	4	< 1s	24	< 1s	50	5s	50
3x6ring	< 1s	5	< 1s	50	9s	50	3m	50
3x6ring_alt	< 1s	3	< 1s	35	6s	50	2m	50
5ring+6rings	< 1s	21	1s	50	1m	50		
Lsd	< 1s	28	4s	50	13m	50		
Longifolene	< 1s	16	< 1s	50	23s	50		
Moenocinol	< 1s	23	2s	50	3m	50		
Resistomycin	< 1s	31	6s	50	72m	50		

Table 4: The number of possible plans in HoSP from Table 3. We time how long it takes to generate the 50 best plans.

2.7.3 HoSP-Number-All

HoSP-Number-All determine equality based on $=_N$. Since this is less strict than $=_E$. This means the HoSP in general will be smaller then when constructed with HoSP-Edge-All. This means there will also be fewer synthesis plans in the HoSP. However all synthesis plans found with this method will be unique, which might not be the case when using HoSP-Edge-All.

Table 5 shows the size of the HoSP and number of starting materials for different bondset sizes. Table 6 shows the time required to generate 50 plans.

Bondset size	1		2		3		4		5	
	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves
6line	$(6, 3)$	5	$(9, 10)$	4	$(10, 15)$	3	$(9, 14)$	2	$(6, 9)$	1
12line	$(12, 6)$	11	$(21, 40)$	10	$(28, 84)$	9	$(33, 128)$	8	$(36, 162)$	7
6ring	$(2, 1)$	1	$(7, 4)$	5	$(10, 11)$	4	$(11, 16)$	3	$(10, 15)$	2
2x6ring	$(5, 4)$	4	$(22, 28)$	17	$(46, 109)$	25	$(69, 246)$	25	$(83, 392)$	17
3x6ring	$(6, 5)$	5	$(45, 65)$	39	$(151, 417)$	107	$(343, 1544)$	194	$(525, 3664)$	185
3x6ring_alt	$(4, 3)$	3	$(28, 36)$	24	$(106, 260)$	79	$(239, 989)$	135	$(347, 2296)$	111
5ring+6rings	$(23, 21)$	22	$(232, 431)$	210	$(1347, 4055)$	1116	$(4798, 22350)$	3453	$(10749, 78030)$	5954
Lsd	$(33, 28)$	32	$(426, 785)$	395	$(3659, 10493)$	3236	$(22069, 88119)$	18417	$(97371, 516911)$	75314
Longifolene	$(18, 16)$	17	$(143, 255)$	126	$(703, 1920)$	561	$(2340, 8931)$	1641	$(5159, 28488)$	2825
Moenocinol	$(40, 23)$	39	$(269, 538)$	230	$(806, 3679)$	539	$(1659, 12818)$	857	$(2709, 30864)$	1056
Resistomycin	$(34, 31)$	33	$(508, 961)$	476	$(5048, 14470)$	4542	$(36249, 140346)$	31206		

Table 5: Table over size of HoSP when constructing the HoSP with HoSP-Number. All possible bondsets of size 1-5 tested.

Bondset size	1		2		3		4		5	
	Time	#Plans	Time	#Plans	Time	#Plans	Time	#Plans	Time	#Plans
6line	< 1s	3	< 1s	6	< 1s	10	< 1s	9	< 1s	6
12line	< 1s	6	< 1s	30	< 1s	50	< 1s	50	< 1s	50
6ring	< 1s	1	< 1s	3	< 1s	6	< 1s	10	< 1s	9
2x6ring	< 1s	4	< 1s	24	< 1s	50	< 1s	50	< 1s	50
3x6ring	< 1s	5	< 1s	50	< 1s	50	2s	50	10s	50
3x6ring_alt	< 1s	3	< 1s	33	< 1s	50	1s	50	5s	50
5ring+6rings	< 1s	21	< 1s	50	3s	50	31s	50	4m	50
Lsd	< 1s	28	< 1s	50	11s	50	3m	50	32m	50
Longifolene	< 1s	16	< 1s	50	1s	50	10s	50	1m	50
Moenocinol	< 1s	23	< 1s	50	4s	50	33s	50	4m	50
Resistomycin	< 1s	31	< 1s	50	16s	50	5m	50		

Table 6: The number of possible plans in HoSP from Table 5. We time how long it takes to generate the 50 best plans.

2.7.4 HoSP-Graph-All

In HoSP-Graph-All the equality of molecules is based on $=_G$, which is the least strict method. By not considering bondsets in equality, the HoSP will also not decrease in size as the bondset grows. Like HoSP-Number-All, all plans represented by the HoSP are unique. There is however fewer different starting materials than in HoSP-Number-All, which is why the HoSP in general is smaller.

Table 7 shows the size of the HoSP and number of starting materials for different bondset sizes. Table 8 shows the time required to generate 50 plans.

Bondset size	1		2		3		4		5	
	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves	$(\|V\|, \|E\|)$	#Leaves
6line	$(6, 3)$	5	$(6, 5)$	4	$(6, 7)$	3	$(6, 8)$	2	$(6, 9)$	1
12line	$(12, 6)$	11	$(12, 11)$	10	$(12, 16)$	9	$(12, 20)$	8	$(12, 24)$	7
6ring	$(2, 1)$	1	$(7, 4)$	5	$(7, 6)$	4	$(7, 8)$	3	$(7, 9)$	2
2x6ring	$(5, 4)$	4	$(22, 28)$	17	$(39, 84)$	25	$(45, 138)$	22	$(47, 176)$	16
3x6ring	$(6, 5)$	5	$(45, 65)$	39	$(141, 359)$	107	$(268, 971)$	182	$(322, 1655)$	169
3x6ring_alt	$(4, 3)$	3	$(28, 36)$	24	$(102, 239)$	79	$(202, 711)$	132	$(229, 1176)$	107
5ring+6rings	$(23, 21)$	22	$(232, 431)$	210	$(1321, 3739)$	1116	$(4435, 17458)$	3416	$(8613, 47447)$	5689
Lsd	$(33, 28)$	32	$(422, 729)$	395	$(3545, 8740)$	3209	$(20427, 63480)$	17892	$(82872, 310166)$	70002
Longifolene	$(18, 16)$	17	$(143, 255)$	126	$(688, 1784)$	561	$(2171, 7148)$	1628	$(4166, 17911)$	2724
Moenocinol	$(40, 23)$	39	$(149, 157)$	142	$(338, 547)$	313	$(585, 1322)$	523	$(830, 2512)$	709
Resistomycin	$(34, 31)$	33	$(508, 961)$	476	$(5023, 13985)$	4542	$(35590, 126042)$	31110		

Table 7: Table over size of HoSP when constructing the HoSP with HoSP-Graph. All possible bondsets of size 1-5 tested.

Bondset size	1		2		3		4		5	
	Time	#Plans	Time	#Plans	Time	#Plans	Time	#Plans	Time	#Plans
6line	< 1s	3	< 1s	4	< 1s	6	< 1s	6	< 1s	6
12line	< 1s	6	< 1s	10	< 1s	18	< 1s	30	< 1s	50
6ring	< 1s	1	< 1s	3	< 1s	4	< 1s	6	< 1s	6
2x6ring	< 1s	4	< 1s	24	< 1s	50	< 1s	50	< 1s	50
3x6ring	< 1s	5	< 1s	50	< 1s	50	2s	50	9s	50
3x6ring_alt	< 1s	3	< 1s	33	< 1s	50	1s	50	5s	50
5ring+6rings	< 1s	21	< 1s	50	3s	50	27s	50	3m	50
Lsd	< 1s	28	< 1s	50	10s	50	2m	50	29m	50
Longifolene	< 1s	16	< 1s	50	1s	50	9s	50	48s	50
Moenocinol	< 1s	23	< 1s	50	3s	50	28s	50	3m	50
Resistomycin	< 1s	31	< 1s	50	15s	50	4m	50		

Table 8: The number of possible plans in HoSP from Table 7. We time how long it takes to generate the 50 best plans.

3 Finding The Best Synthesis Plan

In this chapter we will define the classic cost functions for synthesis plans on hypergraphs. We will also define a dynamic programming method for finding the shortest path in the HoSP.

3.1 Cost Functions Applied to Hypergraphs

The cost functions discussed for synthesis plans in Section 1.1.7, are defined for synthesis plans represented as a unary-binary tree. Since we represent synthesis plans as a hypergraph, we present a definition of the cost functions which can be applied to hypergraphs. In order for the cost functions to be applied on hypergraphs, they need to be expressed as a additive weight function defined in the following way on a given hyperpath π_{st} from s to t:

$$W(u) = \begin{cases} w(p(u)) + F(p(u)) & \text{if } u \in V \setminus \{s\} \\ C & \text{otherwise} \end{cases}$$

$W(u)$ define the cost of node u. The predecessor function p is used to find the hyperedge $e = p(u)$ which have u as head. For each hyperedge, e, a weight $w(e) \in \mathbb{R}^+$. This weight must be the same for all no matter which path e is part of.

$$F(p(u)) = \sum W(\text{Tail}(p(u)))$$

This is a non-decreasing function of the sum of the weights of the nodes in the tail of Tail$(p(u))$. Each starting material have a constant cost C.

3.1.1 External Path Length on Hypergraphs

Recall external path length is defined as follows:

$$EPL = \sum_{i=1}^{k} l_i$$

Where l_i is the path length from starting material i to the target molecule. In hypergraphs this is defined as follows:

$$EPL(u) = \begin{cases} F(p(u)) + b(p(u)) & \text{if } u \in V \setminus \{s\} \\ 0 & \text{otherwise} \end{cases}$$

$$F(p(u)) = \sum_{v \in \text{Tail}(p(u))} EPL(v)$$

Where $b(p(u))$ is the number of starting materials in the subtree of u. For this scoring function to be well defined, it requires the number of starting materials in each subtree of u to be equal. This is only the case in HoSP-Edge and HoSP-Edge-All.

3.1.2 Starting Material Weight on Hypergraphs

Recall that starting material weight is defined as follows:

$$W = \sum_{i=1}^{k} n_i x^{l_i}$$

Where n_i is the number of carbons in starting material i and x is the retro yield.

Starting material weight is defined on hypergraphs as follows:

$$W(u) = \begin{cases} F(p(u)) + \sum_{v \in \text{Tail}(p(u))} W(u) \cdot x - W(u) & \text{if } u \in V \setminus \{s\} \\ n_i & \text{otherwise} \end{cases}$$

Note that the edge cost is defined as

$$\sum_{v \in \text{Tail}(p(u))} W(u) \cdot x - W(u)$$

The cost of each starting material is equal to the number of carbons.

This method is well defined for all method for constructing the HoSP. This is also the method that chemists would see most reason with.

3.1.3 Finding the Best Plan

After the cost functions is defined for the hypergraph we now wish to find the best best synthesis plan. This is done via a dynamic programming approach and the definition is:

$$\text{Cost}(V) = \min_{e \in BS(V)} e + \sum_{u \in \text{Tail}(e)} \text{Cost}(u)$$

The cost of a node V is the minimum over all the possible ways to synthesize it. The cost of a potential approach to synthesize node V is the sum of cost of the educts involved plus the cost of the reaction required for transforming the educts to the product V. The cost of the reaction can be any cost function.

Algorithm 5 Dynamic programming for finding the best plan.

```
1: function MIN(V)
2:     if V is starting material then
3:         return Cost of V
4:     end if
5:     mincost ← ∞
6:     for all e ∈ BS(V) do
7:         cost ← cost of e
8:         for all u ∈ Tail(e) do
9:             cost ← cost + Min(u)
10:        end for
11:        if mincost ≤ cost then
12:            mincost ← cost
13:            V.minedge ← e
14:        end if
15:    end for
16:    return mincost
17: end function
```

One implementation of the dynamic algorithm approach can be seen in Algorithm 5. Several optimization can be made. One obvious optimization is to check if the cost of V has been calculated before, and not recalculate it.

The runtime is $\mathcal{O}(|V| + |E|)$.

4 Finding the K Best Synthesis Plans

In this chapter we introduce our initial approach to finding the K best synthesis plans using a state space exploration inspired approach.

4.1 K Best Synthesis Plans in HoSP - State Space Approach

In this section we introduce a method to finding the K best synthesis plans in the HoSP. This is equivalent to finding the K shortest hyperpaths in the hypergraph that the HoSP models.

Our first goal was to create an algorithm showing that traversal of the HoSP was a feasible approach. This led us to a state spaced based approach.
The basic idea can be split up into the following parts:

- Project the HoSP to a Petri net, where places in the Petri net model molecules, and transitions model reactions. For each configuration of our Petri net, we build a state space node.
- The state space models how the different configurations of the Petri net is connected. States in the state space, model configurations in the Petri net, and edges in the state space hence model reactions in the HoSP.
- The state space is a weighted directed acyclic graph (WDAG). Therefore is it possible to apply Yen's Algorithm, which is a K shortest paths algorithm for WDAG.
- After the K shortest paths in the state space is found, we project them back to the HoSP, and thereby have the K best synthesis plans.

4.2 Petri Nets

With the concept of our HoSP and Petri nets, we can begin our construction of the state space.

4.2.1 Transformation of HoSP to Petri Net

The overall idea is quite simple: Each place in the Petri net models molecules and each transition models reactions and place tokens on all input compounds.

We are now going to formalize this transformation. Create a Petri net PN.

Given a HoSP, $H = (V, E)$, then create $|V|$ places in PN. For each edge $e \in E$ create a transition t, connect t to places in such a way that $\text{Tail}(e) = \text{in}(t)$ and $\text{Head}(e) = \text{out}(t)$. In other words, the input places must be the educts from reaction e, and the output place must be the product from e.

In Figure 41 we give an example of a transformation from a hypergraph to a Petri net. Note that molecules are not depicted in the hypergraph, but only labeled for readability.

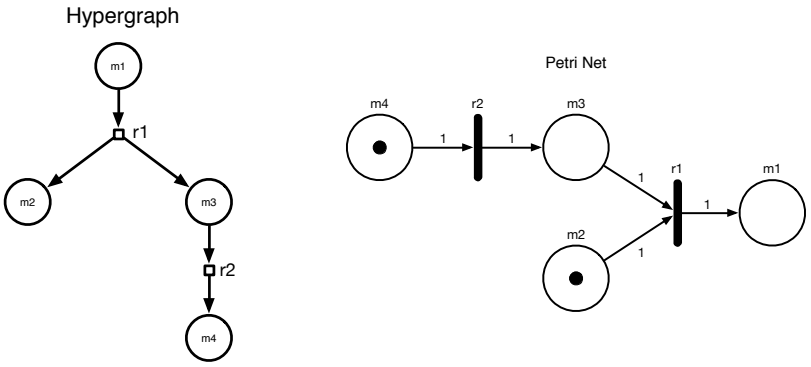


Figure 41: Shows the transformation from Hypergraph to Petri net with labels on both reactions and molecules.

4.2.2 Initial Configuration

The initial configuration of the Petri net needs to reflect the amount of starting materials in the HoSP. Note that there may be more than one token on a place in the initial configuration, i.e. there is a starting material that is used more than once in the final compound.

This adds a limitation to this method, since it indirectly demands that all starting materials must be used in the HoSP, this means that the only combination available is HoSP-Edge.

A simple algorithm for finding the initial configuration is presented in Algorithm 6. After the algorithm has been executed the configuration of the Petri net is the initial configuration needed for the next steps.

Algorithm 6 Algorithm for finding the initial configuration.

```
1:  function Initial Configuration(PN)
2:      root ← p_l
3:      add p_l to queue
4:      while queue not empty do
5:          p ← queue.pop
6:          t ← out(p)
7:          t_1 ← take a transition in t
8:
9:          //Reverse transition t_1
10:         Remove token from p
11:         Create token on in(t_1)
12:         for all p_i ∈ in(t_1) do
13:             add p_i to queue
14:         end for
15:     end while
16: end function
```

4.3 State Space

With the Petri net, PN, constructed and configured, the next step is to construct a graph that models how different configurations of the Petri net look. We call this graph the *State Space*.

The state space $SS = (S, \rightarrow)$ consists of states, S, which represents all possible configurations in the Petri net. For a given state S_1 the configuration, in the Petri net, is denoted $\text{Conf}(S_1)$. $\rightarrow$ models transitions, and describes how states are connected. A connection $S_i \rightarrow S_j$ models a transition $t = \text{Trans}(S_i \rightarrow S_j)$ which transforms $\text{Conf}(S_i)$ to $\text{Conf}(S_j)$.

Two states S_1, S_2 are connected via connection $S_1 \rightarrow S_2$ if there exists a transition t in the Petri net such that the following criteria is met:

1. Transition $\text{Trans}(S_1 \rightarrow S_2) = t_1$ is enabled in configuration $\text{Conf}(S_1) = C_1$.
2. If t_1 is fired in C_1, the resulting configuration is $\text{Conf}(S_2)$.

Observe that each connection in the state space is automatically weighted by the HoSP. Any connection $p \rightarrow q$ models a transition $\text{Trans}(S_p \rightarrow S_q) = t$, a transition t models a reaction in the HoSP which is weighted. Hence the state space is a WDAG.

4.3.1 Building the State Space

Given the Petri net and the initial configuration the idea for constructing the state space is described in Algorithm 7.

Algorithm 7 Algorithm for constructing the state space.

```
1:  function SS Construct(Petri net, Initial Configuration)
2:      PN ← Petri net
3:      SS = (S, →)
4:      unexploredStates ← ∅
5:      S_1(Initial Configuration)
6:      insert S_1 into SS
7:      add S_1 to unexploredStates
8:      while |unexploredStates| > 0 do
9:          S_i ← unexploredStates
10:         for all Enabled transitions t_j ∈ Conf(S_i) do
11:             S_new Fire t_j ∈ Conf(S_i)
12:             insert S_new into SS
13:             Connect S_i with S_new by S_i → S_new
14:             Trans(S_i → S_new) ← t_j
15:             add S_new to unexploredStates
16:         end for
17:     end while
18: end function
```

4.3.2 From State Space to a Synthesis Plan

Since the state space is a WDAG, a K shortest paths algorithm, such as Yen's Algorithm, can be used to find the K shortest paths.

A path in the state space specifies a set of reactions, hence a potential synthesis plan in the HoSP. For any reaction in the set, the reaction along with the educts and product of the reaction, specify the reactions and molecules in the synthesis plan.

4.3.3 Complexity

Analyzing this approach is paramount to understanding it's shortcomings.

The first question to ask is: What is the complexity for building the Petri net? If we again define the HoSP to be: $H = (V, E)$. Then the cost is clearly

$$\mathcal{O}(|V| + |E|)$$

The complexity of the state space is analyzed. The number of states can be expressed as:

$$\mathcal{O}(|V|^{|I|})$$

Where $|I|$ is the number starting materials.

The number of connections is not less than the number of states which is trivially true.

The last thing we need to examine is Yen's Algorithm. Finding the K shortest paths in the state space with Yen's Algorithm is:

$$\mathcal{O}(K \cdot |V|^{|I|} \cdot (|V|^{|I|} + |V|^{|I|} \log |V|^{|I|}))$$

Which can be rewritten to:

$$\mathcal{O}(K \cdot |V|^{2 \cdot |I|} \cdot |I| \cdot \log |V|)$$

4.3.4 Problem with Uniqueness

An important question to ask is: "Are the plans generated from the State Space method unique synthesis plans?".

We will now present a example where this is not the case.

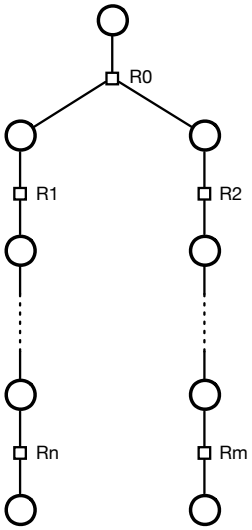


Figure 42: Hypergraph $\mathcal{H}$.

Given the hypergraph $\mathcal{H}$ in Figure 42, it is clear to see that only one hyperpath exists from the target compound to the starting materials, hence there must only be one path in the state space as well.

Unfortunately this is not the case. Lets look at the two chains of reactions in Figure 42. Reactions in the two chains can happen independent of each other. This means that any execution order of the reaction is possible in the two chains. But it is still only one hyperpath.

The obtained state space is depicted in Figure 43.

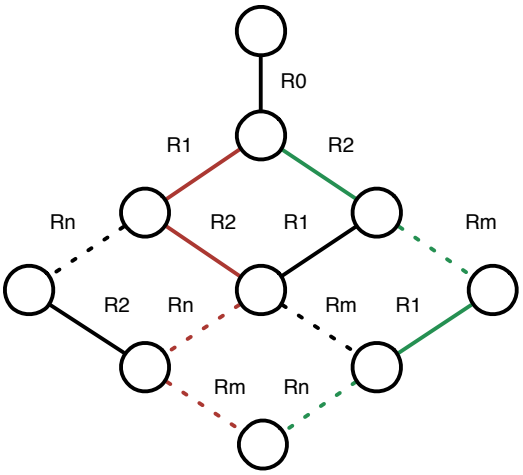


Figure 43: State space for the hypergraph $\mathcal{H}$ as seen in Figure 42.

If we denote the number of reactions in the two chains to n and m, and $n = m$, then there are n^2 different paths in the state space.

This is a common problem within the state space exploration field. Hence methods to reduce this problem are known in the literature. One of the most promising techniques is Monotonic Partial Order Reduction [28].

Applying these techniques wont change the overall complexity but only prune the search space, hence the underlying algorithm needs to be changed.

Example of uniqueness

To demonstrate the problem with uniqueness of plans found by the state space method, a realistic size problem is investigated.

The molecule to be synthesized is Estrone, and the bondset is defined in [23].

HoSP construction	K Best method	Number of plans with cost 29
HoSP-Edge	State Space	503
HoSP-Edge	Hypergraph - Yen	8

Inspecting the 503 plans generated by the state space method, shows there are actually only 8 unique plans, which happens to be `exactly` the 8 plans found by applying Yen's Algorithm to hypergraphs. This is a clear indication that this method should not be preferred when solving the K Best Synthesis plan problem.

4.4 Yen's Algorithm Applied to Hypergraphs

With the fundamental problems presented in the state space approach, we need to devise a new approach that can handle larger instances.

The fundamental realization was: What if we apply Yen's Algorithm directly on the HoSP instead of on a derived graph?

This efficient approach to find the K best synthesis plans in the HoSP will be described below.

Recall that the problem of finding the K best synthesis plans in the HoSP, is equivalent to finding the K synthesis plans with the lowest cost, with the cost functions defined in 3.1.

4.4.1 Motivation

The state space approach for finding the K best synthesis plans, have several problems, i.e. with regard to uniqueness of the solutions generated. The problem imply that the number of shortest paths, in the state space, necessary to find K unique synthesis plans, often exceeds K by a factor 2 in real life examples.

As earlier described this leads to a large overhead which is undesirable. Especially because you often choose a large K. An efficient method that only generates unique plans is therefore desired. The algorithm given in [37] by Nielsen *et al.*, is an algorithm for finding the K shortest hyperpaths in a B-hypergraph.

Since the HoSP is a B-hypergraph, and a hyperpath in the HoSP describes a unique synthesis plan, the algorithm in [37], can be applied directly to our model.

The algorithm is based upon Yen's Algorithm, but with a different branching step. Like in Yen's Algorithm we need an algorithm for finding the shortest hyperpath. The algorithm we use is the dynamic programming approach described earlier in this thesis. The dynamic programming approach, as shown earlier, runs in $\mathcal{O}(|V| + |E|)$.

The algorithm will be described in detail below, but the basic idea is the same as Yen's algorithm. Initially the shortest hyperpath is found, next the algorithm recursively considers all hyperpaths which deviates from one of the shortest hyperpaths found so far. One of the considered hyperpaths is the shortest, and is added to the set of already found shortest hyperpaths. This continues until K hyperpaths have been found.

4.4.2 Theory

Theory and definitions required for understanding how the algorithm works is presented below.

From HoSP to *st* Hypergraph

Since the algorithm by [37], finds the K shortest *st* hyperpaths in a hypergraph. We will present a simple transformation from our HoSP to a *st* hypergraph, see Algorithm 8. Note t is the root in the HoSP.

Algorithm 8 Transformation of HoSP.

```
1: function TransformHosp(Hosp)
2:     Let H be a new hypergraph
3:     H.V ← Hosp.V
4:     H.E ← Hosp.E
5:     add S to H
6:     for all u ∈ Hosp.Input do
7:         add edge from S to u in H
8:     end for
9:     return H
10: end function
```

Note that the transformation is not necessary for an actual implementation, since the only transformation done is adding a virtual starting node to all input compounds.

B-Hyperedges and B-Hypergraphs

Recall that a directed hypergraph is a pair $H = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of directed hyperedges.

A hyperedge is an ordered pair $e = (\text{Tail}(e), \text{Head}(e))$ of subsets of $\mathcal{V}$.

Often $\text{Tail}(e) \cap \text{Head}(e) = \emptyset$ is an assumption. A hyperedge is called a backward hyperedge (B-edge) if $|\text{Head}(e)| = 1$.

If all hyperedge in $\mathcal{H}$ are B-edges, then $\mathcal{H}$ is a B-hypergraph.

Ordering

A valid ordering in $\mathcal{H}$, is a topological ordering of the nodes

$$\mathcal{V} = \{u_1, u_2, ..., u_n\}$$

such that, for any $e \in \mathcal{E} : (u_j \in \text{Tail}(e)) \vee (\text{Head}(e) = u_i) \Rightarrow j \leq i$.

Hyperpath

Given a hypergraph $\mathcal{H}$. A hyperpath π_{st} with origin s and destination t, is an acyclic minimal hypergraph (with respect to deletion of nodes and hyperedges) $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ satisfying the following conditions:

1. $\mathcal{E}_\pi \subseteq \mathcal{E}$
2. $s, t \in \mathcal{V}_\pi = \cup_{e \in \mathcal{E}_\pi}(\text{Tail}(e) \cup \{\text{Head}(e)\})$
3. $u \in \mathcal{V}_\pi \setminus \{s\} \Rightarrow$ is connected to s in $\mathcal{H}_\pi$

If there exist such a hyperpath from s to t in $\mathcal{H}$, we say s is *hyperconnected* to t.

If s is hyperconnected to t in a B-hypergraph via the hyperpath π_{st}, then s is also hyperconnected to all other nodes $u \in \mathcal{V}_\pi$ via hyperpath π_{su} with the node set $\mathcal{V}_u \subseteq \mathcal{V}_\pi$ and hyperedgess $\mathcal{E}_u \subseteq \mathcal{E}_\pi$.

We say π_{su} is a *subhyperpath* of π_{st}.

Since B-hypergraphs only has B-hyperedges, condition 3 guarantees, that for any node $u \in \mathcal{V}_\pi \setminus \{s\}$ there exist a hyperedge $e \in \mathcal{E}_\pi$ where $\text{Head}(e) = \{u\}$.

Because of minimality there exists exactly one such hyperedge, hence $e = p(u)$ is the predecessor hyperedge for the node u.

The predecessor function $p : \mathcal{V}_\pi \to \mathcal{E}_\pi$ defines the hyperpath π_{st}.

Condition 1 guarantees that there exists at least one hyperedge $e \in E_\pi$ with $u \in \text{Tail}(e)$. It follows that there exists an ut-hyperpath.

4.4.3 Finding the K shortest hyperpaths

The definition of the K shortest hyperpaths, is defined in the following way.

Let $H = (V, E)$ be a directed B-hypergraph, $s \in \mathcal{V}$, and $t \in V$, where node s and t is connected. The set of all hyperpaths from s to t is denoted $\mathcal{P} = \{\pi | \pi \text{is a hyperpath from } s \text{ to } t\}$.

Since π_{st} is a hyperpath and per definition acyclic. Hence there exists a valid ordering of the nodes.

$$\{s, u_1, u_2, ..., u_q = t\}$$

Since we have a valid ordering of the nodes, we can easily create a ordering of the hyperedges with the predecessor function.

$$\{p(u_1), p(u_2), ..., p(u_q)\}$$

This ordering of the hyperedges is used to partition the remaining hyperpaths into q disjoint subsets $\mathcal{P}^i, 1 \leq i \leq q$ in the following way:

- $\mathcal{P}^q$ is all st hyperpaths which does not contain the hyperedge $p(u_q)$
- For each hyperpath $\mathcal{P}^i$ where $1 \leq i \leq q$ is the set of all st hyperpaths which contains the hyperedges $p(u_{i+1}), p(u_{i+2}), ..., p(u_q)$, but not the hyperedge $p(u_i)$.

In other words, each set $\mathcal{P}^i$ consists of all the st hyperpaths that have the same $q - i$ hyperedges as π_{st}, and deviates at exactly the ith hyperedge.

All hyperpaths, except π_{st}, must be contained in one of the defined sets.

Idea

The idea behind the algorithm is the following:

- Given the shortest hyperpath π_{st} in H, create the partition $\mathcal{P} \setminus \{\pi_{st}\}$.
- The second shortest hyperpath in H, is the shortest hyperpath in $\mathcal{P} \setminus \{\pi_{st}\}$, hence it must be the shortest hyperpath in one of the sets $\mathcal{P}^i, 1 \leq i \leq q$ which $\mathcal{P}$ represents, lets call the set $\mathcal{P}^j$.
- The shortest hyperpath π_2 in $\mathcal{P}^j$ is used to create the partitions $\mathcal{P}^{j,m}, 1 \leq m \leq |\pi_2|$.
- The third shortest hyperpath in $\mathcal{H}$ will be in $\mathcal{P} \setminus \{\pi_{st}, \pi_2\}$, hence one of the partitions $\mathcal{P}^i, i = 1, ..., j-1, , j+1, ..., q$, or $\mathcal{P}^{j,m}, 1 \leq m \leq |\pi_2|$.
- Continue partitioning until K shortest hyperpaths are found.

Subhypergraphs

Storing $\mathcal{P}$ as sets of hyperpaths requires precomputation of all hyperpaths, making the whole algorithm unnecessary. Instead a set of hyperpaths is represented as a subhypergraph of $\mathcal{H}$.

$\mathcal{H}$ is the representation for all *st* hyperpaths in $\mathcal{H}$, hence a representation of $\mathcal{P}$.

Each $\mathcal{P}^i$ is represented by a hypergraph $\mathcal{H}^i = (\mathcal{V}^i, \mathcal{E}^i)$, which is derived in the following way:

- $\mathcal{V}^i = \mathcal{V}$, i.e all the nodes is included.
- The hyperedges $\mathcal{E}^i \subseteq \mathcal{E}$ in the following way:
  - The hyperedge $p(u_i)$ is removed from $\mathcal{H}^i$
  - The hyperedges $p(u_{i+1}), p(u_{i+2}), ..., p(u_q)$ is fixed in $\mathcal{H}^i$. Formally $u_j, j \geq i : \mathrm{BS}(u_j) = \{p(u_j)\}$, which means that all other ingoing edges to the last $q - i$ nodes will be removed.

The method for creating, the hypergraph $\hat{H}$ representation of a set of hyperpaths P' along the shortest path $\hat{\pi} \in \hat{P}$, is called `BackwardsBranching`.

Algorithm 9 Backward Branching method for a B-hypergraph.

Require: A valid ordering of the nodes $\{s, u_1, u_2, ..., u_q = t\}$.

```
1: function BackwardsBranching(Ĥ)
2:     B ← ∅
3:     for i = 1 to q do
4:         Let Ĥ^i be a new hypergraph
5:         Ĥ^i.V = Ĥ.V
6:         //Remove the i'th hyperedge from Ĥ^i
7:         Ĥ^i.E = Ĥ.E \ {π̂.p(u_i)}
8:         //Fix the backtree of Ĥ^i
9:         for j = i + 1 to q do
10:            Ĥ^i.BS(u_j) = {π̂.p(u_j)}
11:        end for
12:        B = B ∪ {Ĥ^i}
13:    end for
14:    return B
15: end function
```

The algorithm for finding the K shortest hyperpaths `YenHyp`, maintains a priority list L of tuples $(\hat{H}, \hat{\pi})$, where $\hat{H}$ is the hypergraph representation of a set of *st* hyperpaths in $\mathcal{H}$. The list L is sorted with respect to the cost of π.

In each iteration of `YenHyp` the pair (H', π') with the smallest cost is popped off the list, π' is outputted. Next H' is partitioned along π' using `BackwardsBranching`, the shortest path is found in each of the new partitions. All new tuples is inserted into L. The algorithm terminates when the K hyperpaths are found.

Algorithm 10 K Shortest Paths Algorithm in B Hypergraph.

```
1: function YenHyp(H, s, t, K)
2:     Let L be a new priority queue
3:     Let A be new list
4:     π ← ShortestPath(H, s, t)
5:     Insert (H, π) into L
6:     for k = 1 to K do
7:         if L = ∅ then
8:             Break
9:         end if
10:        (H', π') ← L.pop
11:        Insert π' into A
12:        for all H^i in BackwardsBranching(H', π') do
13:            π^i ← ShortestPath(H^i, s, t)
14:            if π^i is complete then
15:                Insert (H^i, π^i) into L
16:            end if
17:        end for
18:    end for
19:    return A
20: end function
```

Note that an algorithm ShortestPath is used to find the shortest hyperpath in a hypergraph. As shown earlier this can be done effectively with dynamic programming, which runs in $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$.

`YenHyp` makes K iterations, in each iteration the length of a hyperpath determines the number of calls to ShortestPath, the worst case for the length of a hyperpath is $\mathcal{O}(|\mathcal{V}|)$. Hence the running time for `YenHyp` becomes:

$$\mathcal{O}(\mathcal{K} \cdot |\mathcal{V}| \cdot (|\mathcal{V}| + |\mathcal{E}|))$$

4.4.4 Example

We will show a small example of how the algorithm works by applying it to the hypergraph shown in Figure 44. The weight will remain the same throughout the example, but will only be depicted in the first figure. Hyperedges depicted as red and dashed are deleted.

The example will cover an entire iteration of the algorithm.

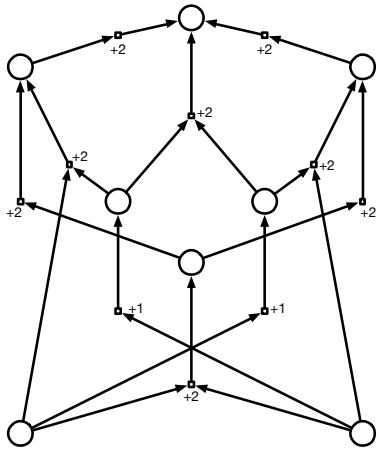


Figure 44: The hypergraph $\mathcal{H}$ with weights.

A method for finding the shortest path in the hypergraph $\mathcal{H}$ is applied, and the shortest path with a cost of 4, is shown in Figure 45.

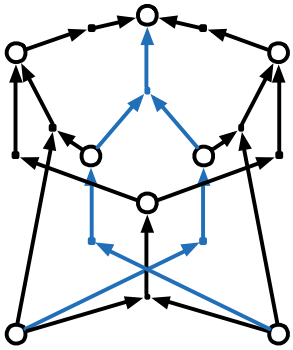
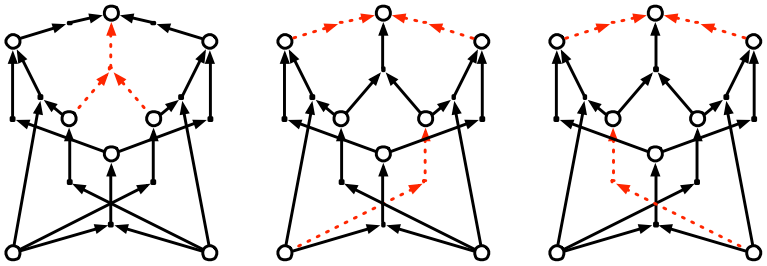


Figure 45: The best path π_1 found in $\mathcal{H}$.

The function `backwardsBranching` is applied on $(\mathcal{H}, \pi_1)$, which yields the hypergraphs $\mathcal{H}^1$, $\mathcal{H}^2$ and $\mathcal{H}^3$.

The shortest path is found in each of the hypergraphs. In the hypergraph $\mathcal{H}^1$ the shortest path is denoted π_2. In the hypergraphs $\mathcal{H}^2$ and $\mathcal{H}^3$ there are no paths.



$\mathcal{H}^1 : \{\pi_2\}$ $\mathcal{H}^2 : \emptyset$ $\mathcal{H}^3 : \emptyset$

The second shortest path is π_2 in $\mathcal{H}^1$ is depicted in Figure 46. All the red edges are deleted from the hypergraphs.

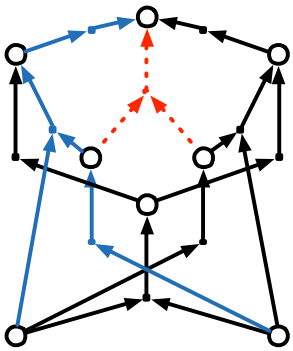
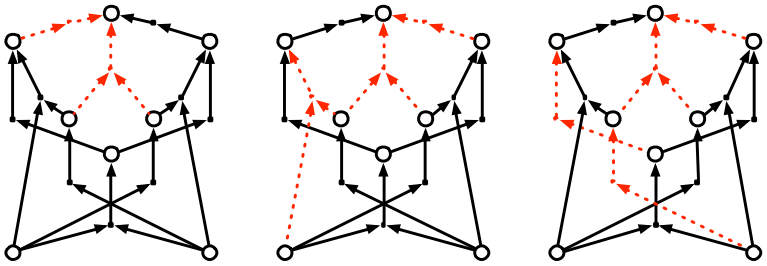


Figure 46: The second best path π_2 found in $\mathcal{H}^1$.

The algorithm will continue by calling `backwardsBranching` on $(\mathcal{H}^1, \pi_2)$, which would yield 3 new hypergraphs $\mathcal{H}^{11}$, $\mathcal{H}^{12}$ and $\mathcal{H}^{13}$.

Again the algorithm for finding the shortest path is applied to the hypergraphs. The path π_3 is found in $\mathcal{H}^{11}$ and π_4 is found in $\mathcal{H}^{12}$.



$\mathcal{H}^{11} : \{\pi_3\}$ $\mathcal{H}^{12} : \{\pi_4\}$ $\mathcal{H}^{13} : \emptyset$

The path π_3 is the third best path in $\mathcal{H}$. We cannot determine if π_4 is the forth best plan at this point. First we must apply `BackwardsBranching`

along π_3, this is left to the reader. See Figure 47 and Figure 48 for π_3 and π_4 respectively.

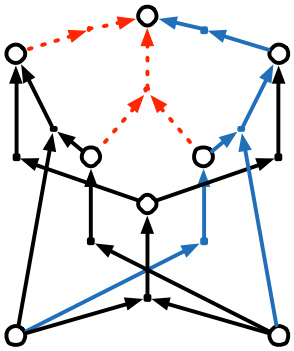


Figure 47: Third best path π_3 in $\mathcal{H}^{11}$.

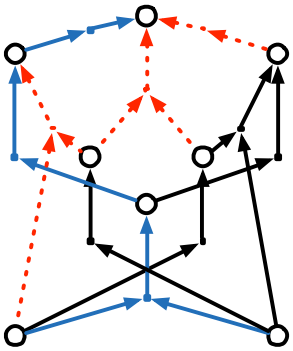


Figure 48: Shortest Path π_4 found in $\mathcal{H}^{12}$.

4.5 Results

In this section we will compare how robust the best plans are, when they are rated with another cost function. Phrased differently, we measure the cumulative distribution of two cost functions.

Comparing different cost functions

The first example is the molecule estrone, with the bondset specified in [23]. The HoSP is constructed with HoSP-Edge, and all plans are outputted.

We now plot how many of the x best plans with respect to Starting Material Weight (Weight) there also is the x best plans with respect to External Path Length (External) and vice versa for various x.

Two lists are created. One that is sorted with respect to Weight (list 1), and one which is sorted with respect to External (list 2).

First list 1 is clustered and each cluster is internally sorted with respect to External. The same thing is done for list two with respect to Weight.

For a given cluster of size x we check how many of the plans contained in the cluster is within the best x in the other list. An example could be that from the best 500 with respect to Weight, 400 of those is contained within the best 500 with respect to External. This would give the point $(500, 400)$.

The further away the points are from the line $x = y$, the less equal the two cost functions are.

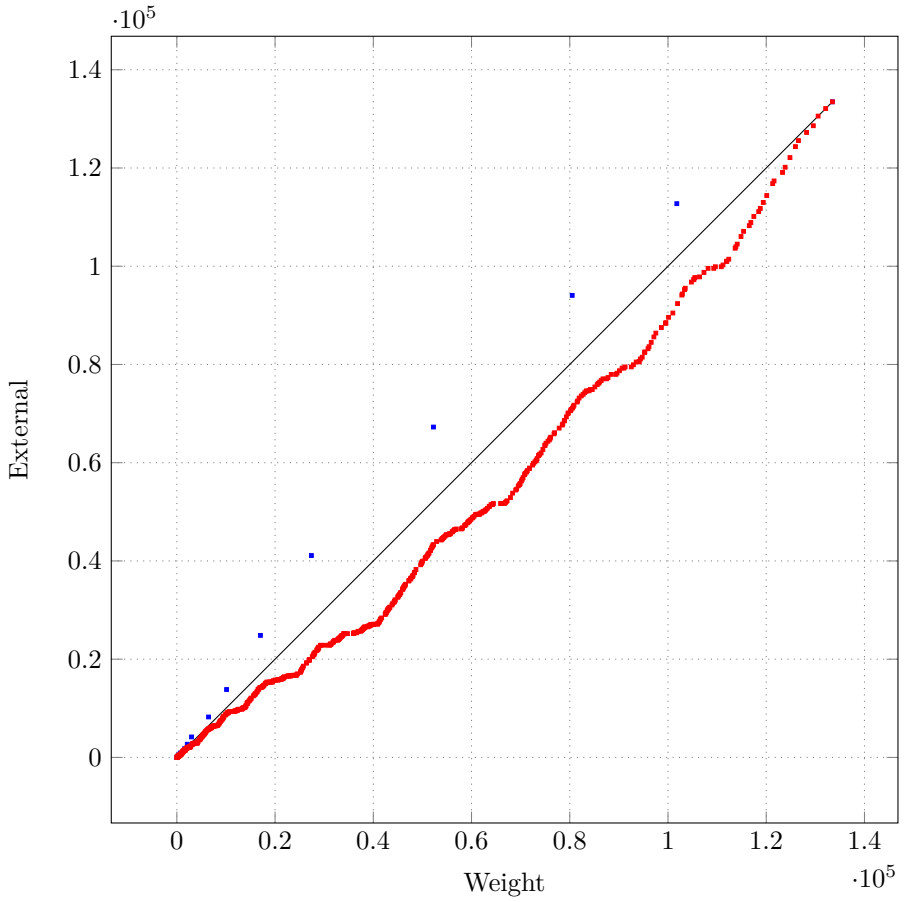


Figure 49: The cumulative distribution of the two cost function, Starting Material Weight and External Path Length. The HoSP is generated with HoSP-Edge, and all plans are found and scored with both objective functions. This graph shows how equal the two cost functions rate the plans found.

As seen in Figure 49, the points lie quite close to the line $x = y$, meaning the two scoring functions rate the 133472 plans found quite similar.

Comparing same cost function

Lets compare the same scoring function, Starting Materials Weight, but with different retro yield. We have chosen a retro yield of 1.25 and 2. The technique for creating the plot is the same, this time however we zoom in on the first 20000 plans, see Figure 50.

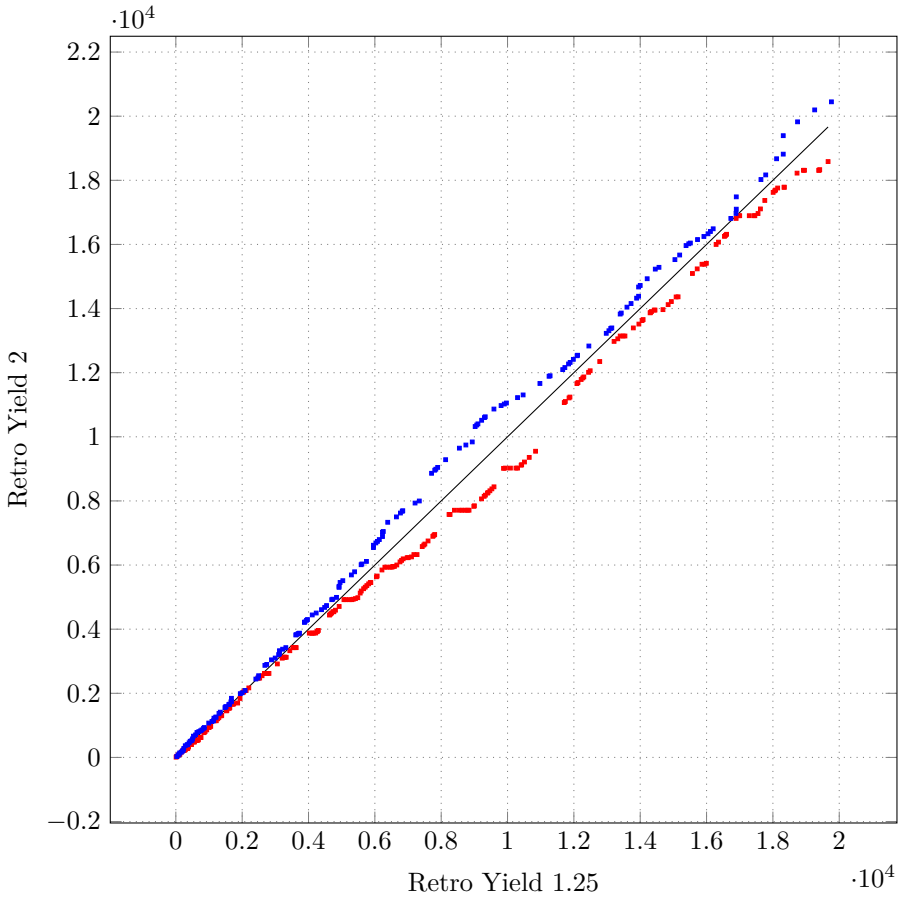


Figure 50: The same cost function, Starting Material Weight, is compared with different retro yield parameters, 1.25 and 2. The HoSP constructed with HoSP-Edge. We examine the first 20000 plans.

In Figure 50, we see the different retro yield values, rate the 20000 plans almost identical. The points lie very close to the line $x = y$. This means the best plans found with one retro yield value, is robust when changing the retro yield value.

Comparing same cost functions with different construction method

Lets examine what the distribution look like if a different construction method for the HoSP is used. This time we construct the HoSP using HoSP-Graph, and still use the retro yield values 1.25 and 2. We plot the first 20000 plans in Figure 51.

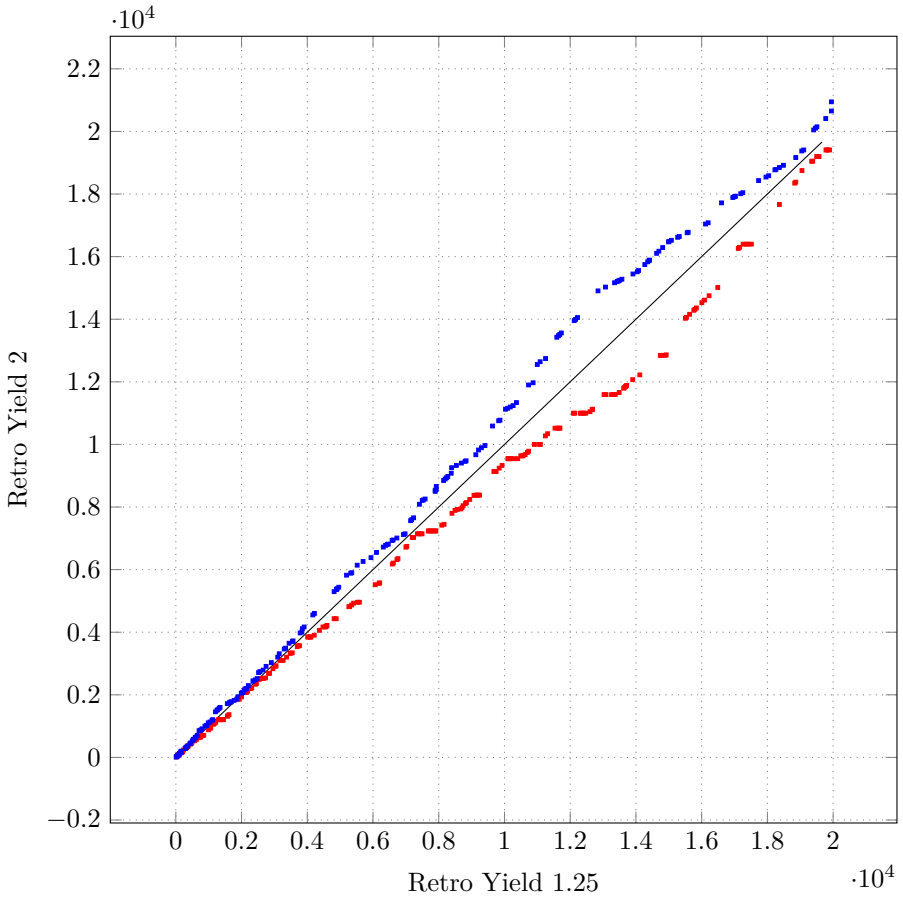


Figure 51: Cost function, Starting Material Weight, with different retro yield of 1.25 and 2, is compared. This time the HoSP is constructed with HoSP-Graph. We plot the first 20000 plans.

If we compare Figure 51 with Figure 50, we notice that the method for constructing the HoSP does not have a large influence on the cumulative distribution of the plans.

Extreme Bondset

Let us examine what happens when the bondset is chosen such that the largest input compound is obtained. We examine how Starting Materials Weight compares to External Path Length. The distribution can be seen in Figure 52. Figure 53 shows the distribution of the first 5500 plans, the two cost functions rate the plans very differently. This might be because External does not take the size of the molecules into account and Weight does.

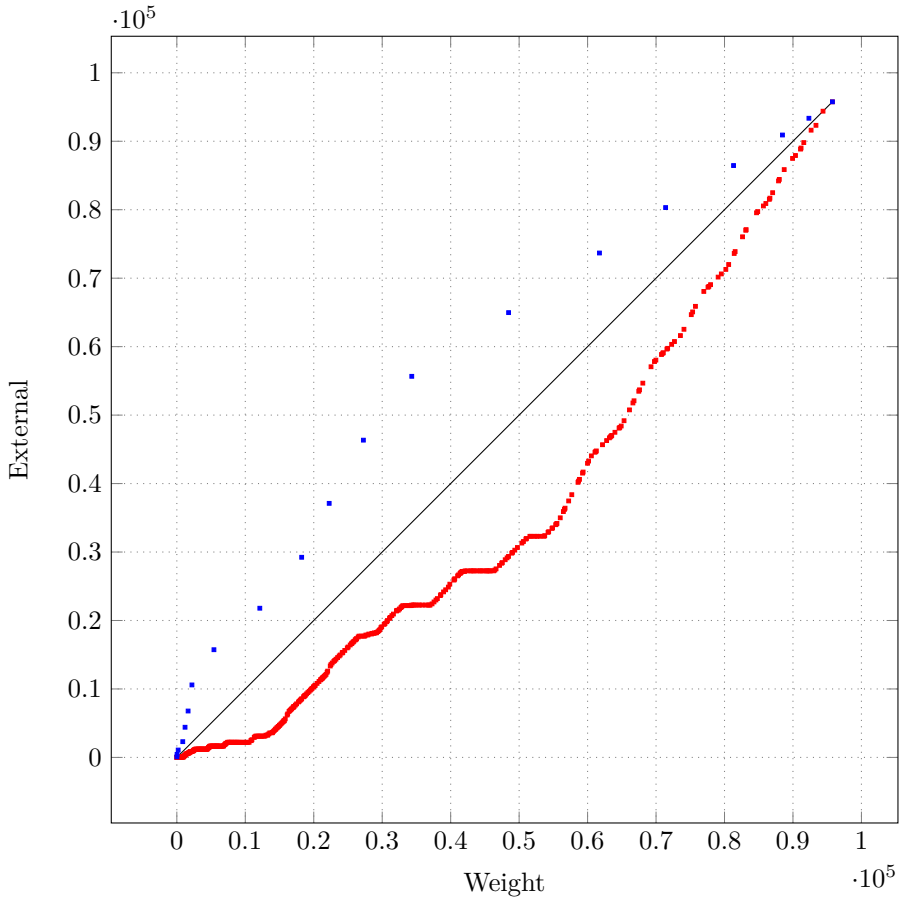


Figure 52: Cost functions, Starting Material Weight and External Path Length is compared. This time the HoSP is constructed with HoSP-Graph.

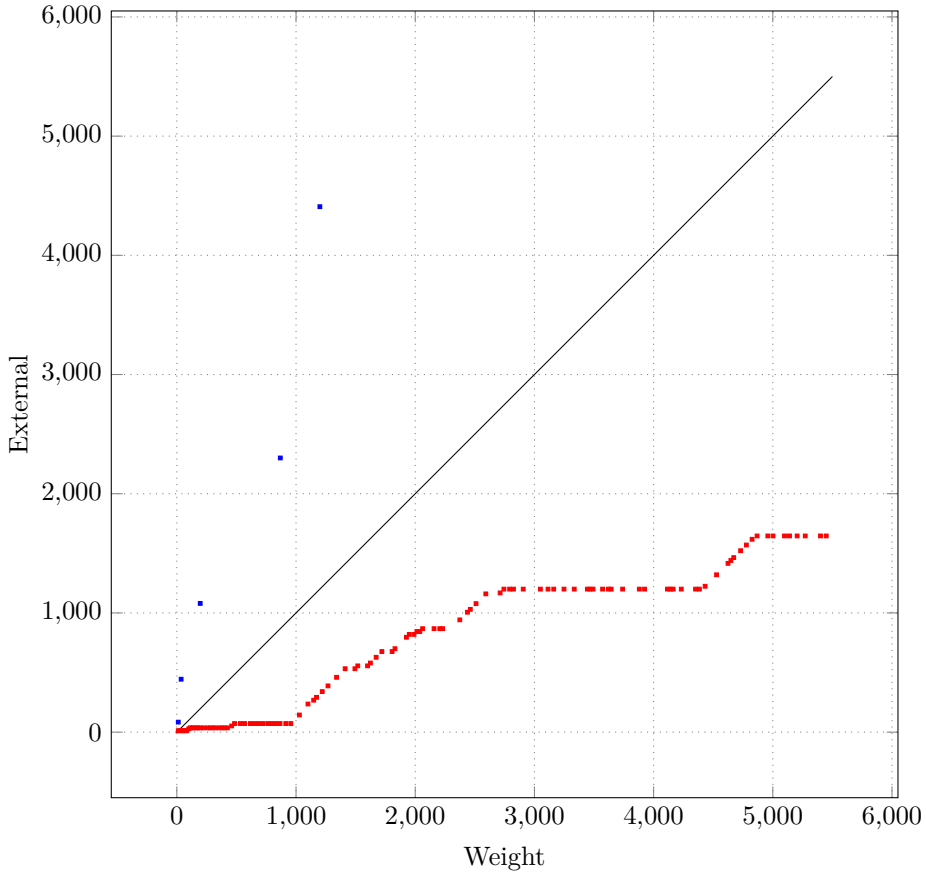


Figure 53: Cost functions, Starting Material Weight and External Path Length is compared. This time the HoSP is constructed with HoSP-Graph. We plot the first 5500 plans.

Analysis of LSD

The analysis is performed on the molecule LSD.

External is compared with Weight, by measuring the cumulative distribution of all plans. This can be seen in Figure 81.

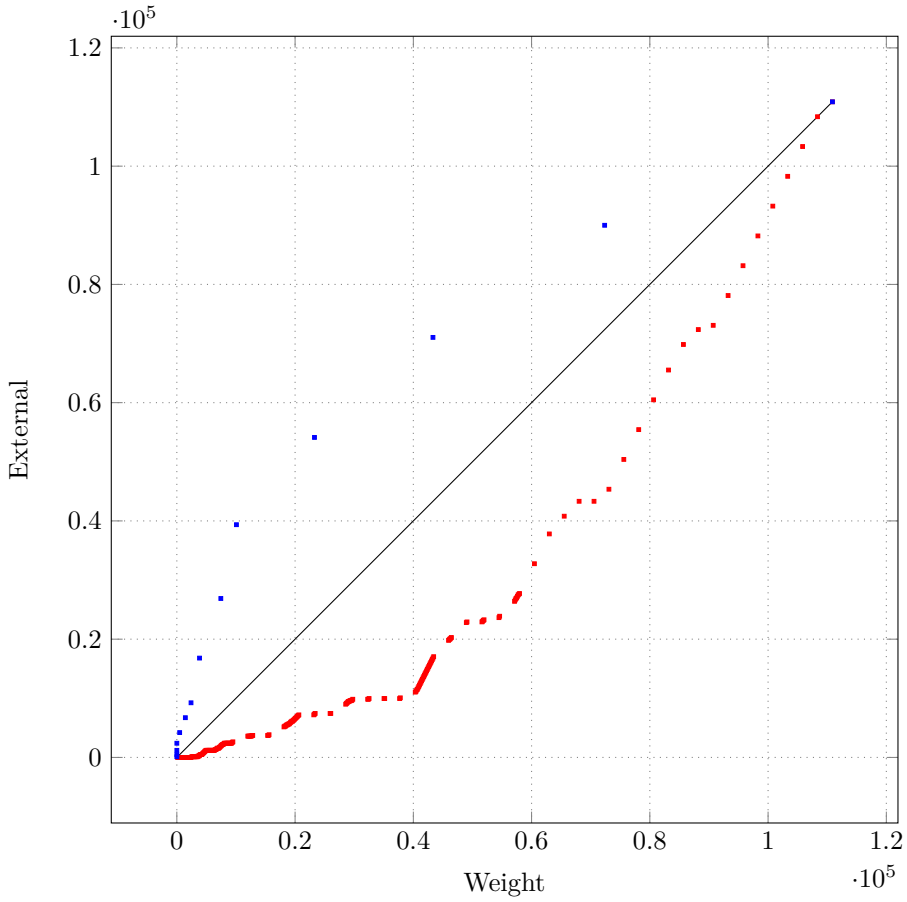


Figure 54: The two cost functions Starting Material Weight and External Path Length is compared for the molecule LSD. All plans are generated with HoSP-Graph.

Figure 81 shows the two cost functions rate the plans different (points are far away from the line $x = y$). The plans found are not robust when changing the cost function.

Different Retro Yield

Two different retro yields 1.25 and 2 will be compared in the cost function Weight. Clusters will be made with respect to both retro yields in a similar manner to Figure 81. The result is seen in Figure 55.

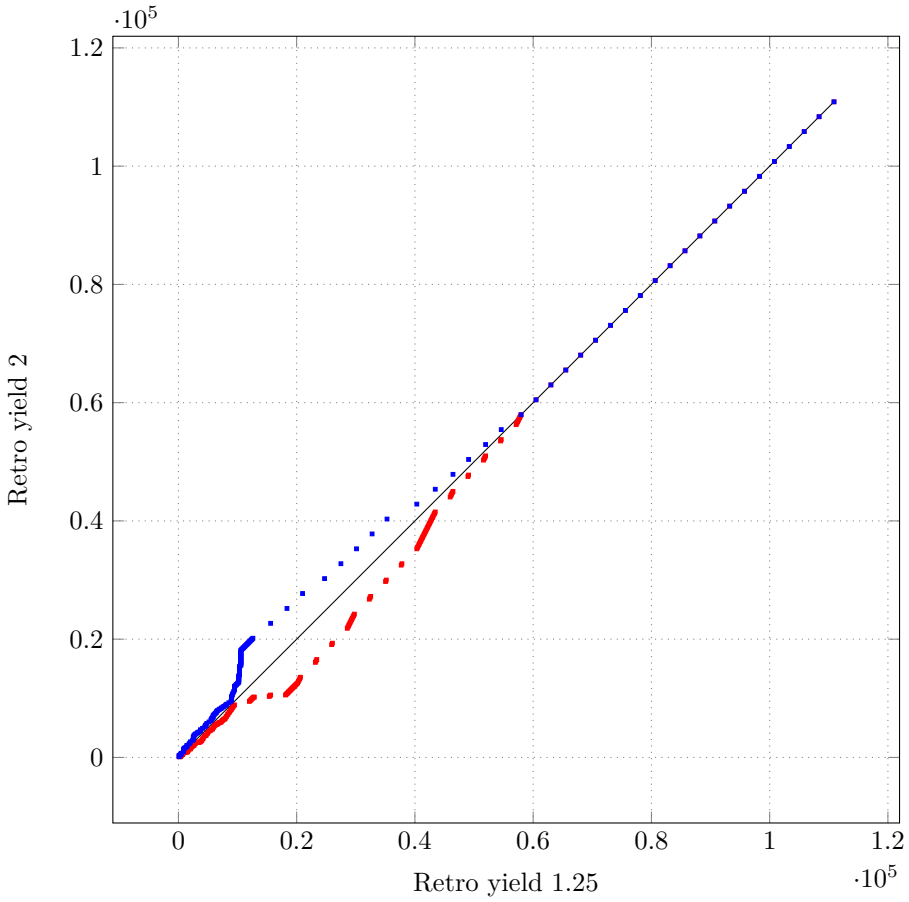


Figure 55: Retro yield 1.25 and 2 is compared in the cost function Starting Material Weight for the molecule LSD. All plans are generated with HoSP-Graph.

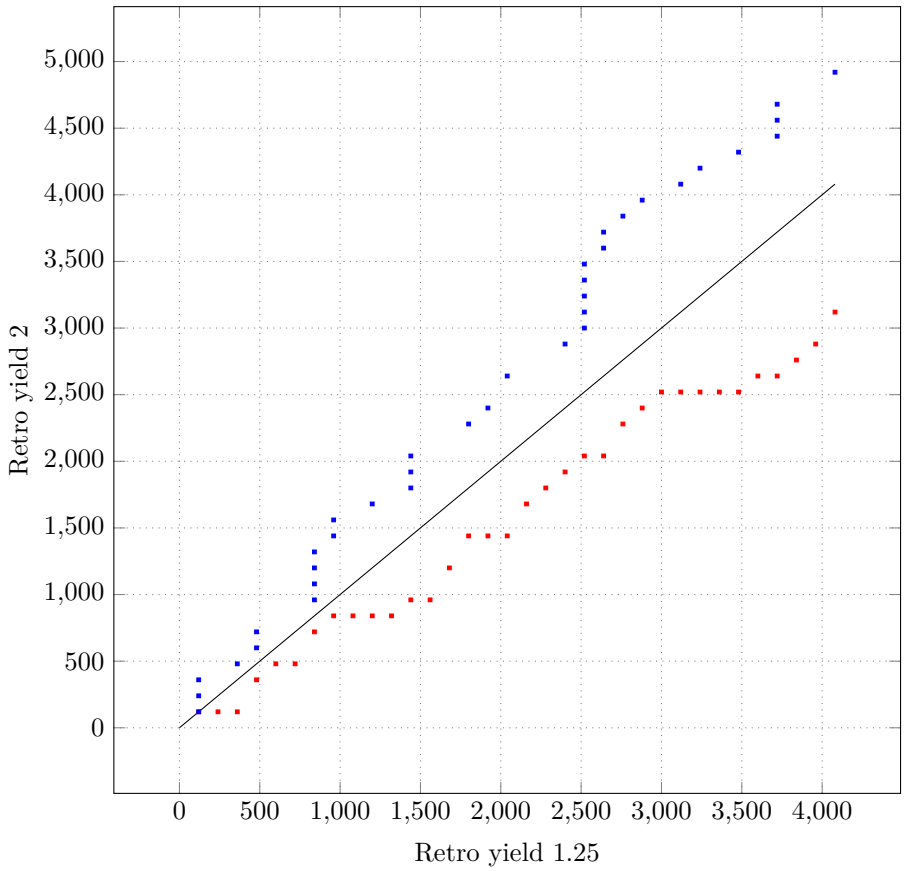


Figure 56: Retro yield 1.25 and 2 is compared in the cost function Starting Material Weight for the molecule LSD. All plans are generated with HoSP-Graph. The first 4000 best plans.

Figure 55 shows the plans found are robust when changing retro yield. Varying the retro yield will result in a very similar ranking of the possible plans. In Figure 55 we show the best 4000 plans.

Extreme Retro Yield

We will examine what happens when the retro yield is high, in this case 10. This means reactions with large molecules have a very bad yield.

In Figure 57 the distribution is shown. Even by changing the retro yield to an extreme value the plans found is still robust. In Figure 58 the best 4000 plans is shown.

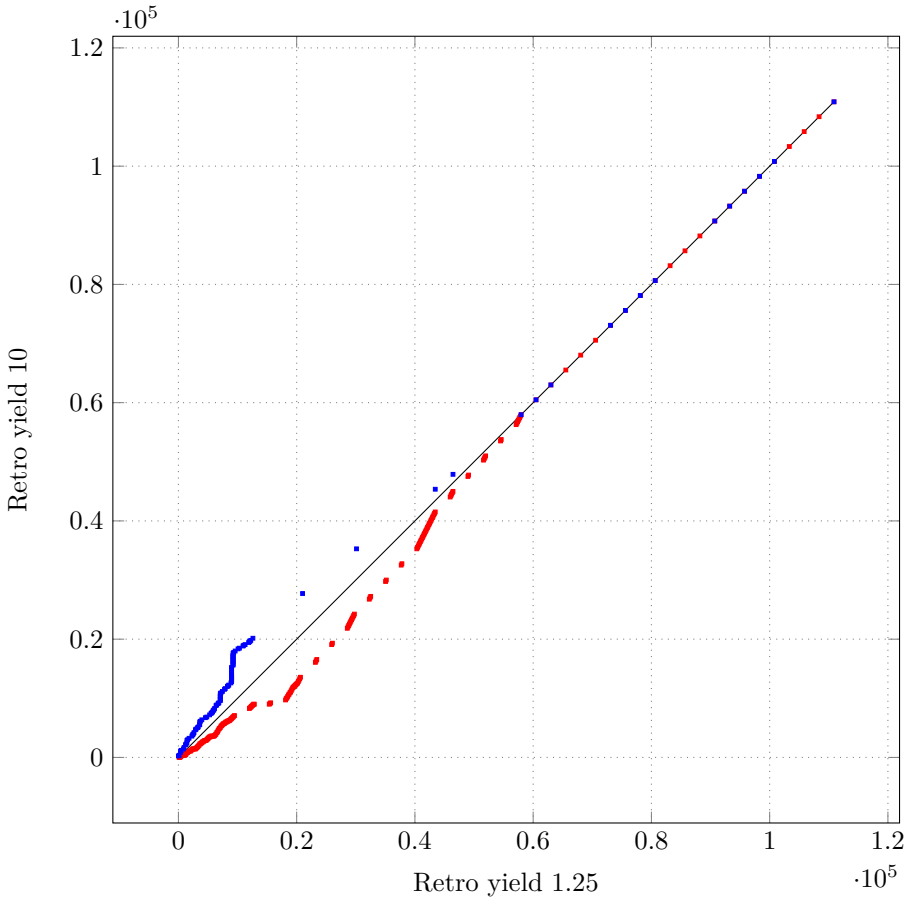


Figure 57: Retro yield 1.25 and 10 is compared in the cost function Starting Material Weight for the molecule LSD. All plans are generated with HoSP-Graph.

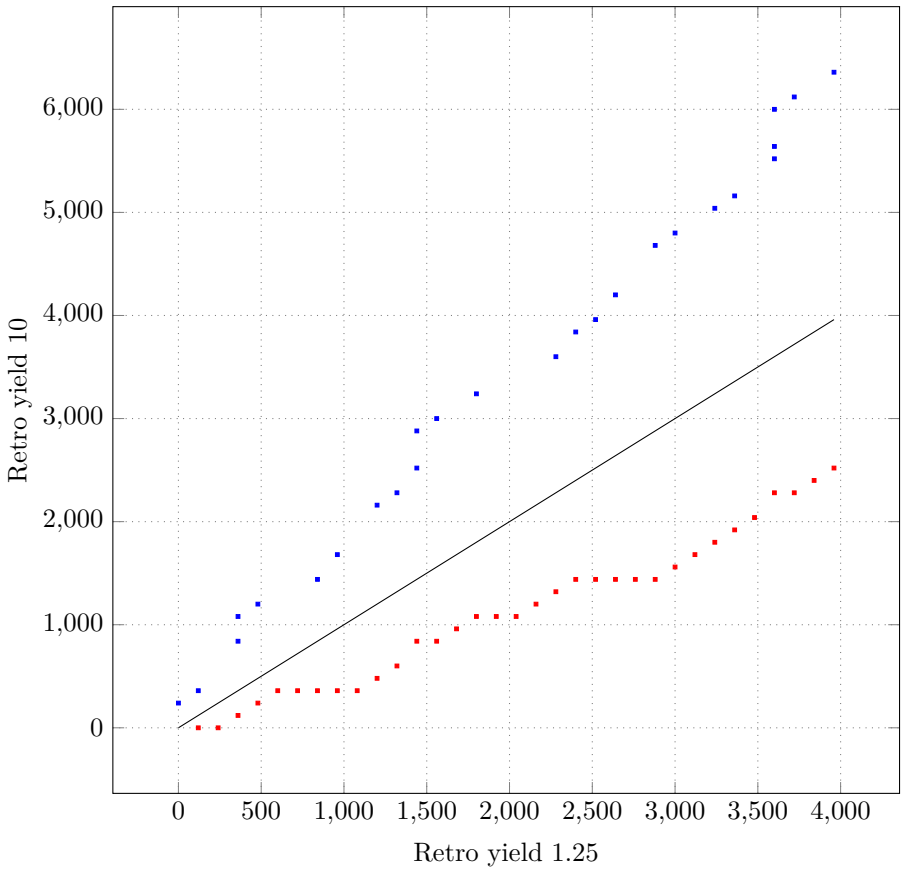


Figure 58: Retro yield 1.25 and 10 is compared in the cost function Starting Material Weight for the molecule LSD. All plans are generated with HoSP-Graph. The first 4000 best plans.

4.6 Constraints

When constructing the HoSP by enumerating over all possible bondsets of a given size, there is no control over the starting materials.

It is however not all starting materials which is desired in synthesis plans. Both very small and very large starting materials is unwanted. Below we introduce two types of constraints limit the number of small and large starting materials.

4.7 Minimum Size Constraint

Small starting materials is unwanted in synthesis plans, simply because larger molecular structures might be synthesized, and available to buy.

It is therefore possible to add a constraint on the minimum size of starting materials in the HoSP.

When disallowing small starting material, all synthesis plans in the HoSP which uses those starting materials will be removed. Hence each time a starting material is disallowed, a set of bondsets is actually removed from the HoSP.

In Figure 59, there is 4 starting materials marked with green. The right most starting material is a single carbon. Lets say we want our starting materials to have minimum size larger than 2 carbons. By introducing this constraint, we would disallow the use of the rightmost starting material. The consequence is that the right subtree would be invalid, since not all leafs are starting materials. In Figure 60 the HoSP after the constraint is applied, is depicted.

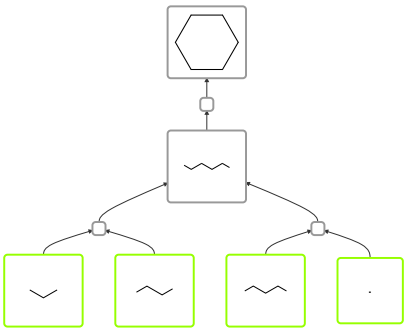


Figure 59: A HoSP before constraint is applied. Starting materials marked with green.

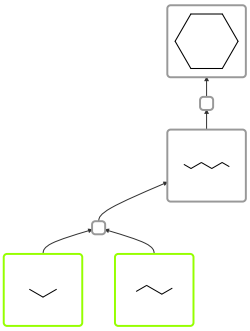


Figure 60: HoSP after the constraint is applied. One Synthesis plan is removed.

4.8 Maximum Size Constraint

It is not only small molecules, which is undesired as starting materials. Large starting materials are just as equally undesired.

The argument against large starting materials, is that they cannot be bought synthesized. If a large compound cannot be bought, it needs to be synthesized, hence additional bonds are actually added to the bondset. Since we only compare bondsets of equal size this is unwanted. Therefore it is possible to create a constraint on the allowed maximum size of starting materials.

Lets again consider the HoSP in Figure 59. Lets say we will disallow starting materials larger then 4 carbons. This would mean, that the starting material second from the right is disallowed. The HoSP in Figure 60 depicts the HoSP after the constraint is applied. Note that the right subtree is removed from the HoSP, because of violation with the constraint.

5 SynthDB – An Open Synthesis Database

During this project it became clear that it would be very beneficial with a user friendly way to present and share synthesis plans. This would greatly aid the chemists to publish new synthesis plans in a uniform manner.

If we were only looking at chemical reactions. Several really good and broadly accepted tools and reaction databases exists, both commercial and open source, like SPRESIweb and Reaxys. They provide accurate information of the reactions and the components and standardized ways of downloading individual reactions.

Several synthesis databases exists online, some free and some commercial. Some of the well known are SynArchive (`http://www.synarchive.com/`) and OrgSyn (`http://www.orgsyn.org/`). These databases store many synthesis plans and you are able to search for compounds and view plans known from the literature. These plans can often be downloaded as PDF, image files etc. This makes it difficult to work with the plans afterwards for other chemists since they need to translate the plan into their own software. Since no standardized way of representing synthesis plans exists, it is quite cumbersome to add new plans to the databases, and it often requires a lot of manual work.

It became clear that a new database using state of the art techniques would be a really valuable asset. The idea of SynthDB was born.

The first goal is to specify a format simple to learn for chemists, human readable but still powerful enough to contain all the intricate details of reactions in a synthesis plan.

5.1 Use Cases

One might ask why a database would be usable at all? We believe it should be able to answer a lot of questions. Some quite obvious, some more advanced. Some of the questions could be:

- Is this synthesis plan in balance? Do atoms appear or disappear?
- Does there exist alternative ways to synthesize compounds in existing plans that could be used in this plan instead?
- In what other plans is this reaction used?
- In what other plans is this molecule used?
- Can we improve a given synthesis plan using information from the database such as alternative reactions?

As far as we know none of the existing tools are capable of answering all of these questions.

Some of these questions can be extended or narrowed down. The last question could easily be almost impossible to answer if taken to extreme degrees.

But answering these questions would significantly help the chemists when devising or looking for new synthesis plans. If the chemist could easily search for a molecule and get plans that produced this molecule, then the best plan could be used to produce input. In this way synthesis plans could very easily help other synthesis plans.

5.2 SPF - Synthesis Plan File

First we began to define a format capable of these things. Several formats were proposed [16], in the end a format reviewed with the eyes of a computer scientist were created. The format is called SPF.

An example of a plan in SPF can be seen in Listing A.1. This is a plan for the synthesis of LSD.

The file format consists of four main sections:

1. An information section, which includes the author, molecule name, etc.
2. A skeleton section, which defines the skeleton of the molecule.
3. A section containing all compounds used in the synthesis plan.
4. A section which specifies the reactions.

5.2.1 The Information Section

The information section is the first section of the file and contains information regarding name, time of last change etc. The section is optional. If no section is present the synthesis plan will be given a name decided by the application.

The keys in this section is case insensitive, hence "`LAST CHANGED`" must be interpreted equal to "`lAsT changed`" or "`last changed`" . All keys must end with a colon (`:`).

This section may contain more keys than specified here, but these will not be parsed by SynthDB.

Each key will be discussed in detail in the following sections.

Last Changed Timestamp

Mandatory: No
Alternate forms: last changed time-stamp
Example:

```
# Last changed timestamp: 2013-03-10 02:19:00
```

Listing 5.1: An example of the last changed key.

This key is used to specify the time of last edit of the synthesis plan. The format of the date is: "`YYYY-MM-DD HH:II:SS`". An example could be: "`2014-03-10 02:19:00`". If the date is not set the current date and time will be used instead.

Last Changed Author

Mandatory: No
Alternate forms: last changed aut
Example:

```
# Last changed author: cl
```

Listing 5.2: An example how to specify the author of last change.

This key is used to specify who did the last edit of this plan. If the plan is imported by command line, the instance of SynthDB will try to link the name to a user in the user database. If the plan is uploaded via the web interface the current user will be posted as last changed author.

Literature

Mandatory: No
Alternate forms: lit
Example:

```
# Lit: Hendrickson, James B and Wang, Jian
#      A New Synthesis of Lysergic Acid
#      Org Lett 6(1):3-5 (2004)
```

Listing 5.3: How to specify supporting literature.

This key is used to specify information about literature concerning this synthesis plan. If the plan was published it would be really valuable to know the corresponding literature.

The first line is the authors, the next the title of the article and the last line is the journal in which the article was published.

DOI

Mandatory: No
Alternate forms: *none*
Example:

```
# DOI: 10.1021/ol0354369
```

Listing 5.4: How to specify supporting literature with DOI.

This is used to give an DOI reference for the literature.

Author

Mandatory: No
Alternate forms: aut
Example:

```
# Author: Carsten Gronbjerg Lutzen <calut09@student.sdu.dk>
```

Listing 5.5: How to specify the author of the plan.

This key specifies the author of the synthesis plan by name and email. If the plan is imported using the web interface the current user will be set to author. If it is imported via command line and the author cannot be found in SynthDB, then the plan wont be imported. SynthDB will use the email to look for the correct user.

5.2.2 Skeleton Section

To facilitate projection of plans, one can specify a skeleton of the target compound. This is specified by writing the skeleton as an InChI string. In this way only bonds in this string will be in the final projection.

```
skeleton: InChI=1S/C6/c1-2-3-4-5-6-1
```

Listing 5.6: Specification of Target Compound Skeleton.

This ignores the functional groups and focuses on the overall structure instead.

5.2.3 Compound Section

The specification of compounds used in the reactions is done by giving each of them a name identifier and an InChI string. Only one compound is allowed on each line. The individual compounds can be named with all characters except a whitespace.

```
compounds:
water InChI=1S/H2O/h1H2
x InChI=X
y InChI=Y
z InChI=Z
```

Listing 5.7: Specification of compounds.

5.2.4 Reactions

The section regarding reactions is a bit more complex than the other sections. We need to tie a lot of information together. An example might clarify things:

```
reactions:
R05 "Acetal formation"
|comp26| + comp27 -> |comp28| + comp29
@yield 81% cat pTsOH
```

Listing 5.8: Specification of a reaction.

We need to specify a unique number in the plan, a name, educts and products and meta data. The reaction is identified by a unique number, i.e. $R1, R2, \cdots, Rn$. Then we specify a "user friendly" name like: "Acetal formation". The unique number in the plan will not be saved in SynthDB but the textual name will.

The next line is a bit more complicated. Here we define the educts and products of the reaction. This is done by using the previously defined compounds. If a compound is surrounded with vertical bars (`|compound|`) this means that it is a primary compound. We define the primary compound to be the main molecule in that reaction. Hence we will have at least one primary compound as educt and exactly one primary compound as product. If we need more than one of a compound this is written as factor times compound, an

example: `3 * water`. This information is used when building the synthesis tree, checking balance and making the projection.

The last line specifies extra information. The line must begin with an "@". In this line the yield must be specified. A simple example is: `@ yield 45.6%`. If the reaction is part of a composite reaction, i.e. a chain of reactions, where we cannot say anything about the yield of the individual reaction we can specify it in the following way: `@ yield R04-R10 10%`, this means that all reactions between reaction 4 and reaction 10 have an overall yield of 10%. This is also how we declare some reactions to belong together.
If one wants to specify more data this is simply done by writing it as `key value`. An example could be:

```
@ solv HCl yield 63% someKey someValue
```

Listing 5.9: Example of Meta Data.

5.3 SynthDB

Our solution was to implement a new database called SynthDB. SynthDB must be very easy to use and wrap advanced functionality, such as making projections of plans, searching for molecules and searching for plans.

SynthDB should be able to make SPF files searchable and visible. To do so a thorough and robust architecture must be made.

5.3.1 Database Layout

The underlaying database structure must be able to handle a lot of things such as groups of reactions, skeletons, reactions used in multiple synthesis plans. To do so we created the layout seen in Figure 61.

The overall idea is normalize all data to a sane state, but keeping some data in cached aggregated form to enable a fast lookup. To do so all of the elements on the SPF have been thoroughly analyzed to determine what is the candidate keys and other identifiers and what is pure data. This coupled with indices and unique columns should make it fast enough for our use. When we take a closer look at the SPF we see that one synthesis plan consists of several groups of reactions. Each reaction group consists of 1 to n reactions. Each of these reactions consists of several compounds and some meta data. All of these relationships adds complexity, but modern Database Management Systems (DBMS) can help a lot and let us define relationship between tables. In this way the DBMS enforces that the data is in a consistent state and we have no inconsistent relations. This helps us keep the database in a normalized state without redundant data.

One example is the synthesis plans. What makes a synthesis plan unique? The name? The reaction groups? The reactions? To ensure most flexibility an artificial key is assigned, making sure that synthesis plans can have the same name without being treated as identical plans. One could argue that the use of the reaction groups could be used to make a key, but this would require a lot of work for almost no gain.

A reaction consists of reactants, also denoted as educts and products. A reaction is actually identified by the reactants. The molecules are unique based

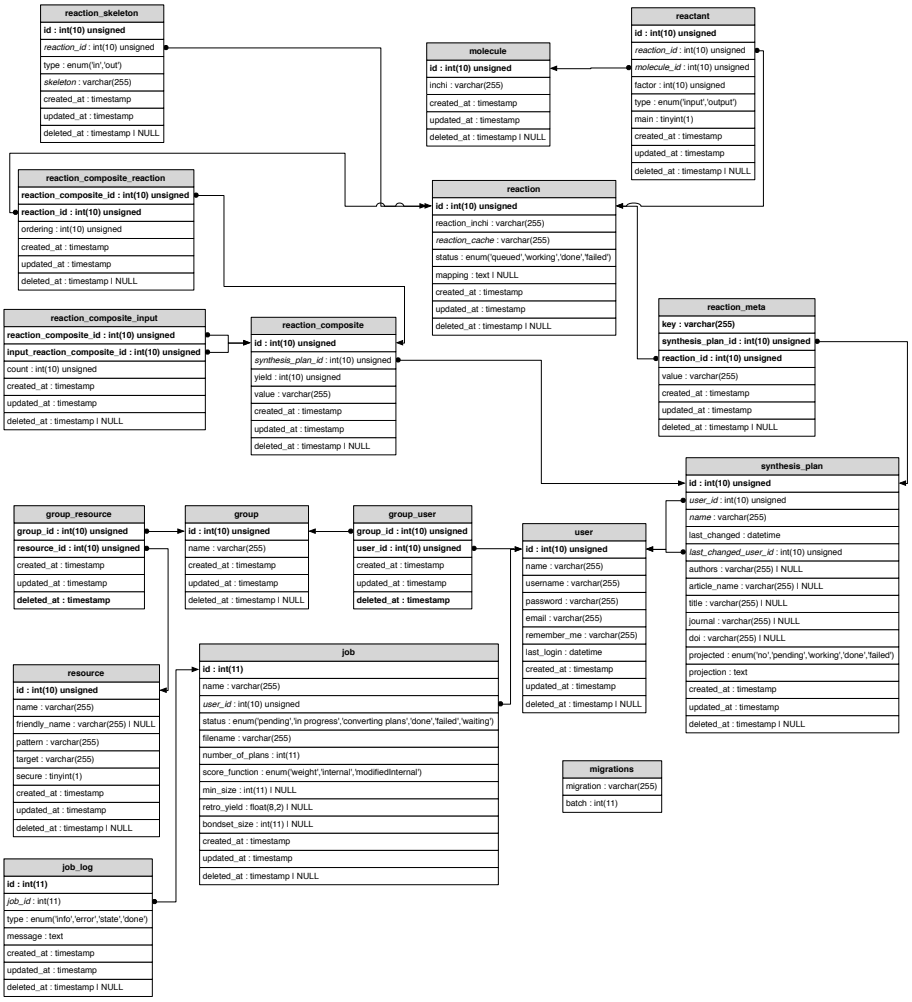


Figure 61: The structure of the database with column types, keys, relations and names.

on their InChI string, hence we have a unique representation usable by foreign key relations. By using the built in features of a DBMS and creating a canonical reactant string we can ensure fast lookup of reactions by using only one query. This, of course introduces some redundant data, but this is a tradeoff between normalization and efficiency. One could argue that the redundant part is in fact even less troublesome since it is only the primary keys.

One example could be a reaction that looks like the following:

```
reactions:
R05 "Aldol Addition"
|ketoneEnolate| + aldehyde -> |aldol|
@yield 100%
```

Listing 5.10: Example of reaction from a SPF file.

The molecule names are only unique in the plan hence we need to translate it into our name of the molecule. We do this by importing the molecules if they do not already exist in the database. We then use their unique id instead of the names from the file and sort it by the molecule id:

```
# 97: aldehyde
# 122: aldol
# 142: ketoneEnolate
reactions:
R05 "Aldol Addition"
97 + |142| -> |122|
@yield 100%
```

Listing 5.11: Example of reaction in canonical format.

All these design decisions have made an extendable and extensible structure suited for large scale data storage.

Of course, this is just the data foundation, and we need to build on top of this to actually perform the queries.

5.3.2 Architecture

With the database layout done, we need to focus on how to make the actual application. The application should be able to scale vertically and horizontally, at least to some degree.

To do so scaleability must be must be a cornerstone in the architecture. Hence the architecture must be a distributed one. Making the architecture capable of spanning multiple layers requires each layer to have a well defined responsibility and purpose. Using the definitions of Renzel et al. [40] it makes sense to use multiple cuts and still keeping a clear responsibility. These cuts could be placed in different places giving different benefits and risks. We want to make a system that many users can use, hence we need some kind of distributed presentation. We could go with the distributed presentation or remote user interface. If we decide to use the remote user interface this would require the users to install an application on each client, this would be a bit more cumbersome, but would enable us to use the power of each client machine. But what would we use this power for? We do not have any clear use cases that would benefit from this. Using the distributed presentation we could make it a web application allowing for easy access but with limited resources and responsibility on each client. Distributed presentation also helps to maintain a good data integrity and high security, since no data is stored on the client.

What about the actual application kernel? Should we keep it on one server or distribute it so several machines? Since some of the work in SynthDB could be quite expensive it might makes sense to create a so called *distributed application kernel*. This poses some problems with security and data integrity, but enables us to distribute the workload onto several servers if need be. This would make the application capable of horizontal scaling. For instance projection of plans and generating new plans can be quite expensive, potentially making simple queries quite slow. This could then be remedied by adding more servers.

We do not want to actually implement an entirely new database engine, hence we use a so called remote database. Whether or not the database is distributed should be transparent for the rest of the application. In this way we can potentially later change the underlying DBMS if new requirements arise.

In the end we stand with an application with the following cuts: Remote User Interface, Distributed Application Kernel and Remote Database.

With these layers we need to find a platform to build upon. Several possibilities exists, depending on the choice of language. Since we have a very strong knowledge of PHP we decided to look for a mature and modern PHP framework with a strong community. An obvious choice would be Drupal. Drupal offers a lot of functionality out of the box, but it does not perform so well due to the age of the system. Another system is Yii. Yii is a strong and sound framework that offers a lot of functionality. But it can be quite cumbersome to work with due to a lot of restrictions. We heard of a new framework called Laravel. Laravel is very popular and trending in the PHP community and we decided to take a closer look.

Laravel is a modern PHP framework that adheres to the SOLID Development Principels [38] and uses a lot of the well known design patterns like: Facade [18, p. 185], MVC, Dependency Injection, Separation of Concerns [30, p. 554] and many more. This greatly helps building maintainable, scaleable and easy to understand and extend applications.
The underlying database in our implementation is a MySQL database, but due to the use of PDO in Laravel this can easily be changed to another database engine like SQLLite or DB2. The complete list of supported database engines can be found here `http://php.net/manual/en/pdo.drivers.php`.

Laravel of course offers remote user interface due to the fact that it creates web applications. The distributed application kernel can be achieved by using the built in queue system in Laravel along with a load balancer. The remote database comes through the use of PDO. All in all Laravel seems like the right platform to use.

A view of the different components can be seen in Figure 62. Here we see that, we can span multiple machines, hence sharing the workload.

Laravel uses a package dependency system called Composer. Our application consists of two Composer(`https://getcomposer.org/`) packages developed by us: Gatekeeper and SynthDB. These packages are then installed into an Laravel installation.

If one needs the functionality of SynthDB one only needs to import the SynthDB package using Composer.

5.3.3 Common Features

All packages are developed using the Service Provider pattern. In this way no hard coded dependencies exists in the code between packages. This makes the system extremely flexible. Many of the classes use interface binding. This ensures that if you want to provide a new implementation of a class, all that is required is to create the new implementation and bind the implementation to the interface name. Then all code will use the new implementation instead. This is done by using Inversion of Control (`http://en.wikipedia.org/wiki/Inversion_of_control`) along with the use of interfaces.

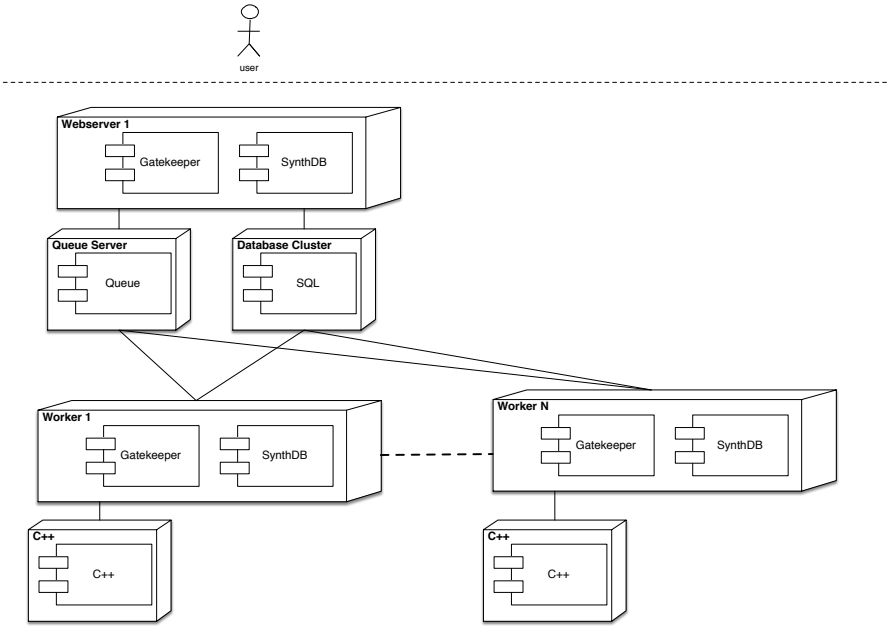


Figure 62: A possible physical deployment strategy.

5.3.4 Gatekeeper

Gatekeeper is an simple Role Based Access Control (RBAC) package responsible for authorizing users, and managing permissions.

This package provides a fine granularity for managing users. But all in all it is a very simple package with limited functionality.

Users are divided into groups, and each group have a set of permissions. A user can be a member of several groups.

5.3.5 Conversion to SPM

Another feature implemented is the possibility to convert InChI, SMILES or CML to our SPM. This makes it easy to convert existing molecules to input to SynthWorker. One can also upload new jobs using these formats and it will then automatically be converted into SPM.

5.3.6 SynthDB

This package does the actual creation of a database for storing the synthesis plans.

The synthesis plans or molecules can be uploaded using the SPF, SPM or text output from SynthWorker. Uploading a SPF file or the text output can be inserted into SynthDB at once. Molecules in InChI or SMILES needs to be converted before being given to SynthWorker hence SynthDB will create a new job, put it in the queue for later processing.

To ensure a high data quality SynthDB employ several validation rules to new synthesis plans. Some of these are balance checks that requires that all

reactions conserve mass. This is done by partially parsing the InChI strings. One could also call external tools, but this would be to slow. Another check is to see if the plan is a connected component, or multiple components. These and other checks play a vital role in keeping the plans correct.

5.3.7 SynthDB and SynthWorker

SynthDB acts as a driver for SynthWorker by giving SynthWorker new input and handling the actual output of the program like information and exit codes. This is done with no link from SynthWorker to SynthDB. In this way there are no dependencies that makes the compiling or running of SynthWorker difficult. SynthDB maintains a queue of jobs not yet processed and processes them one at a time using the Laravel queue feature. When a job is run SynthDB converts and insert the new plans and molecules into the database.

With the system implemented one university faculty can have an installation of SynthDB local on campus. When they have devised a new plan that is ready for publication, they can easily download the file and upload it to the global database. In this way all plans can easily be aggregated into a global database.

5.4 Ideas

It would make sense to integrate the database with Chemspider(`http://www.chemspider.com`) or similar molecule database and reaction databases. This would give the possibility to show a lot of different information for each molecule and reaction.

Another idea is to cache the projection of each reaction plan. This would significantly speed up things. The projection does not depend on the plan, hence it should be possible to cache it. This could also be shared with reaction databases.

At some point it might be required to use more sophisticated access management. It would maybe make sense to implement OAtuh or similar protocol to make it easier for users to be known across databases.

When the database grows to millions of plans the search might become slow. One idea is to introduce SOLR. SOLR is a really fast search engine that is easy to work with. Companies like Netflix, AOL, CNET and many others use SOLR. SOLR provide a lot of really advanced features.

5.5 Results

With the implementation of the database several synthesis plans were constructed and imported into the database.

When a user logs into the database, the user can select different areas: Molecules, Reactions, Synthesis Plans and Jobs. Each of these sections provides different functionality.

5.5.1 Molecules

The molecule section is the simplest of the sections. Here all molecules are displayed and various information is found.

The list of molecules, as seen in Figure 63, displays the InChI string and the number of reactions using this molecule. It is worth noting that the counter counts molecules as input and output of reactions. Hence a molecule that is used in two reactions is often only used in one synthesis plan.

If the users clicks a molecule in the list view then the detailed view seen in Figure 64 is displayed. This shows all reactions that uses this molecule in all the plans.

When the user clicks a molecule the molecule is opened in a modal to display more information, see Figure 65. This is possible throughout SynthDB.

5.5.2 Reactions

The reaction section is responsible for showing all reactions currently in the database, see Figure 66. In this way one can easily find other plans using the same reaction. One example of this is displayed in Figure 67. All educts and products are displayed with multiplicities.

5.5.3 Synthesis Plans

The synthesis plan section is one of the most important sections in the database. Here we can search and view the already imported plans, see Figure 68. Here one can also download the plans and work with them. This is handy when distributing plans to other databases.

When clicking on a plan all literature references are displayed together with a tree showing all main molecules and the yields, this is seen in Figure 69. If one wants to see the entire plan with all educts and products represented as a tree this can also be done in the database, see Figure 70. If one wants to see what molecules are used in multiple reactions in a plan this is done in graph view, see Figure 71. If one wants to create a new plan this is done by just uploading the SPF and name the synthesis plan.

5.5.4 Jobs

Creating a new job allows for importing molecules as InChI, SMILES or SPM. When creating a job all options from SynthWorker can be set as seen in Figure 72. In this way SynthDB works as a graphical front end to SynthWorker. When the job has been created the job is updated in realtime. Then one can easily see what is going on at all times. This is seen in Figure 74.

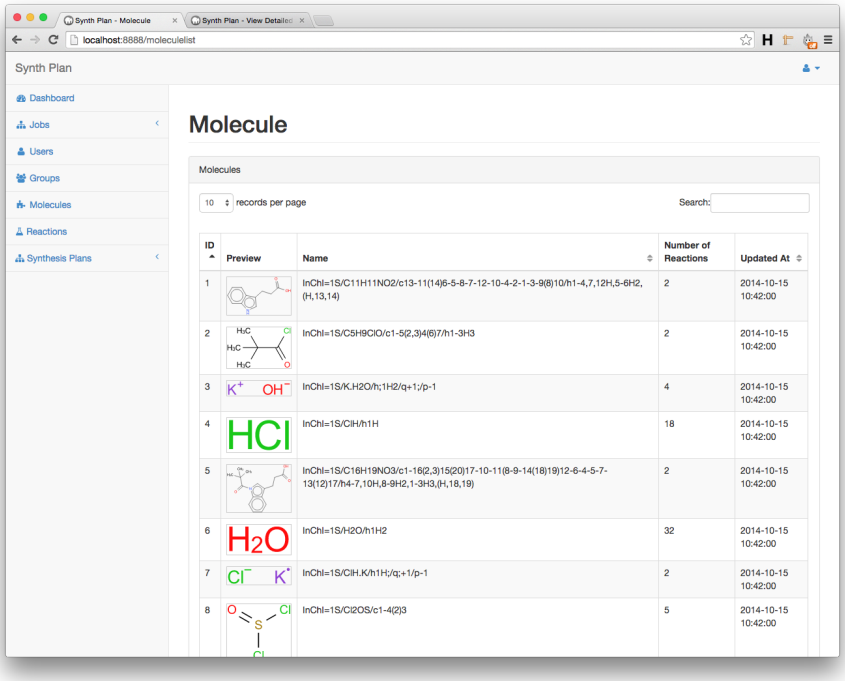


Figure 63: The list of molecules.

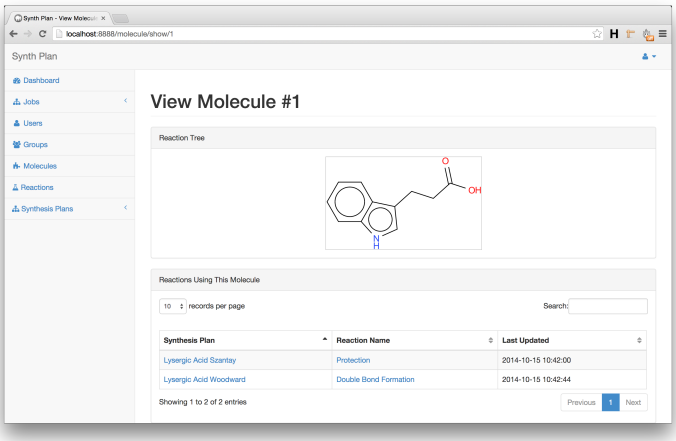


Figure 64: Detailed molecule view, shows all reactions using the molecule and what in what synthesis plans.

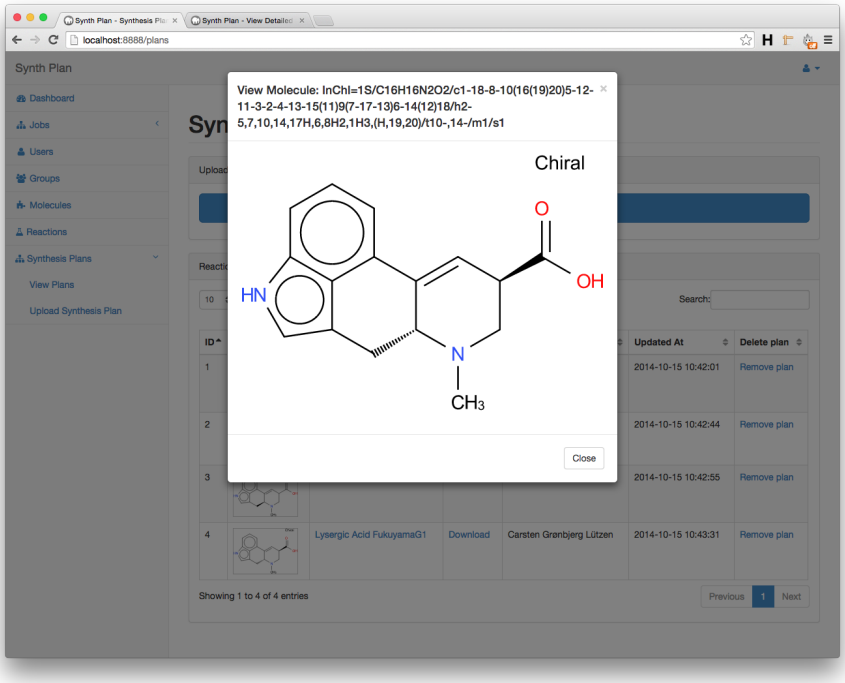


Figure 65: When a user clicks a molecule a modal opens and displays more information if available.

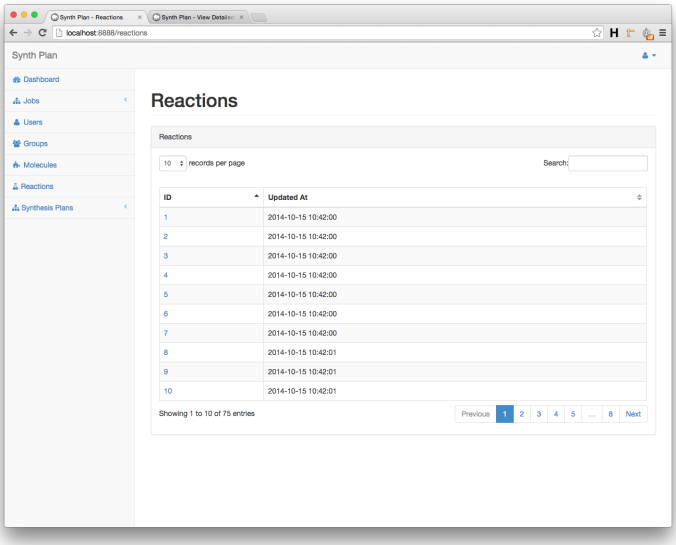


Figure 66: The list of reactions.

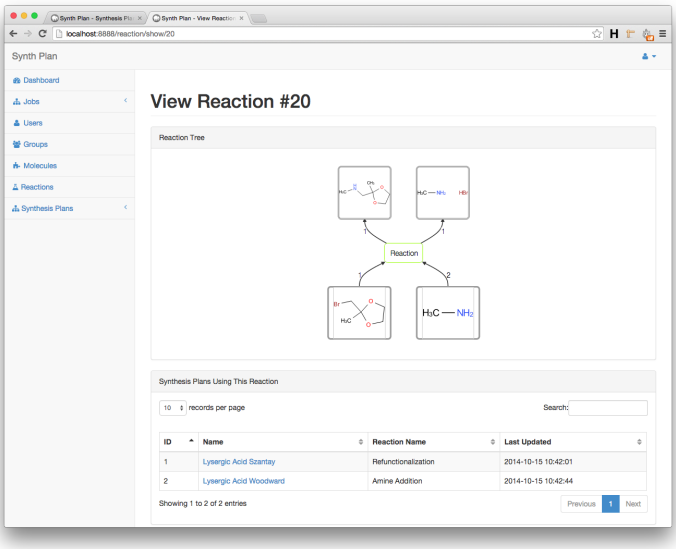


Figure 67: A reaction used in more plans.

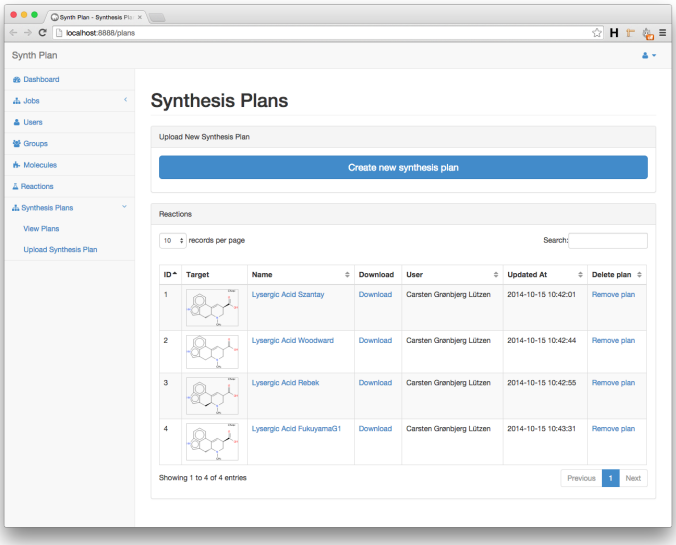


Figure 68: The list of synthesis plans.

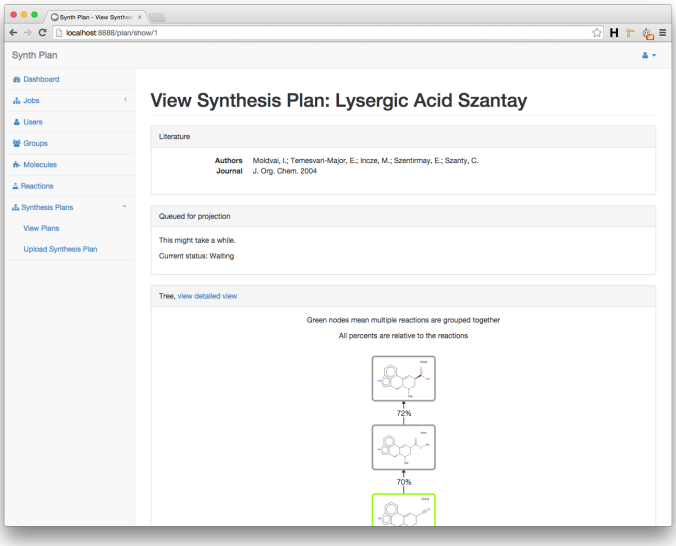


Figure 69: A synthesis plan displayed with literature references and a tree depicting all main compounds.

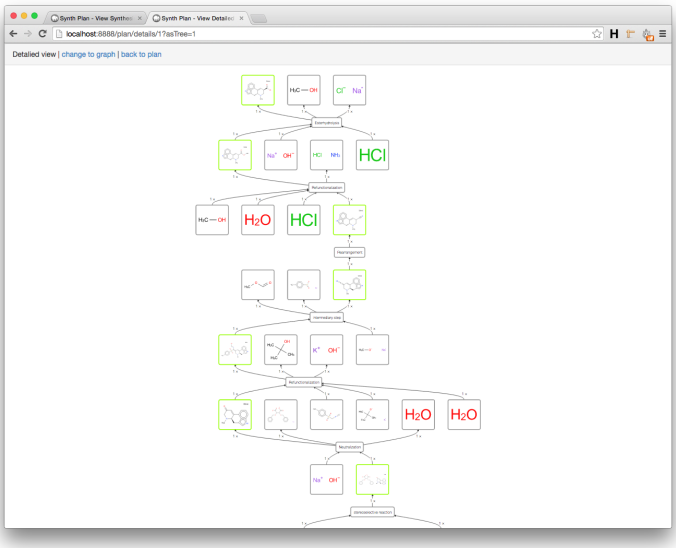


Figure 70: A tree view of a synthesis plan. Here all educts and products are displayed for each reaction.

These jobs can take a lot of time to process hence the user can without any problem close the browser and return later to see the current status of all jobs in the system.

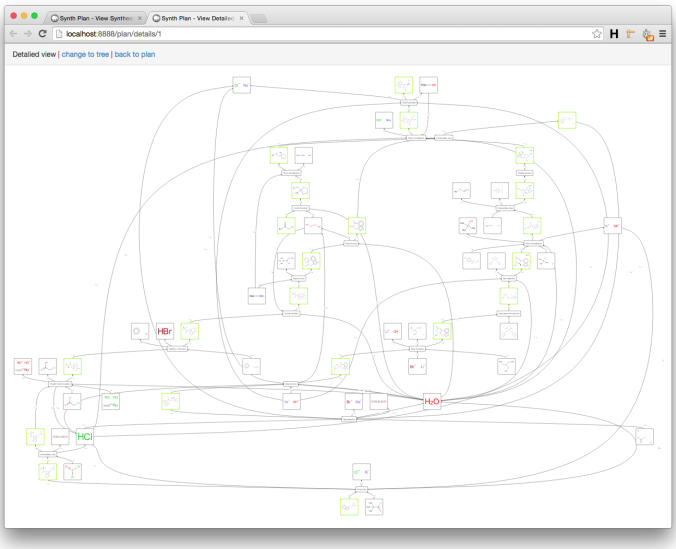


Figure 71: A graph view of a synthesis plan. Here all educts and products are displayed only once and linked to multiple reactions.

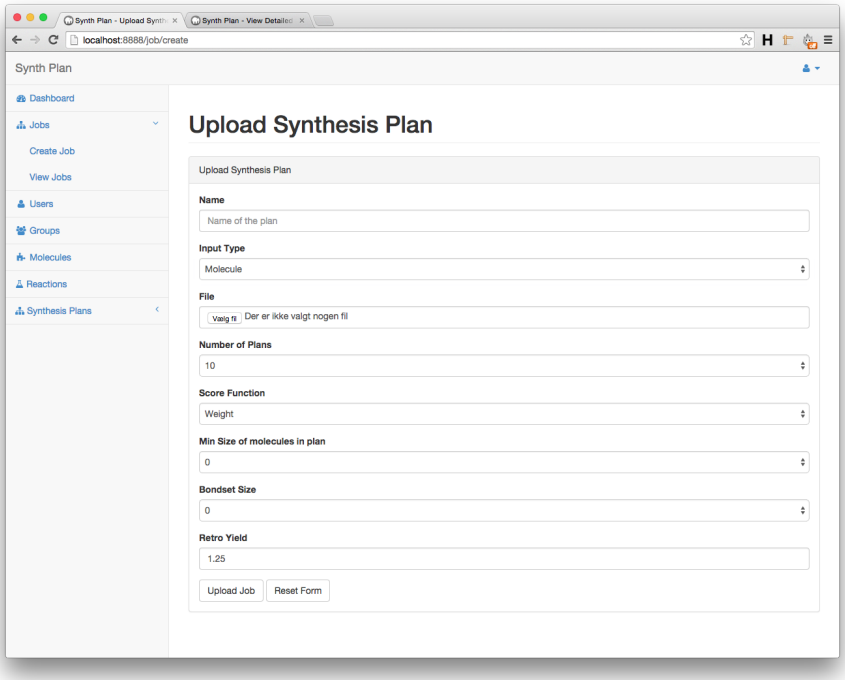


Figure 72: Creation of a new job.

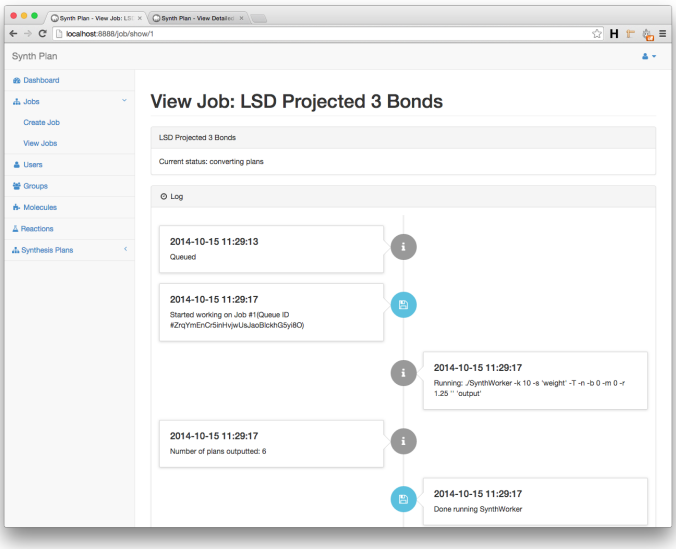


Figure 73: A detailed view a job with the log messages.

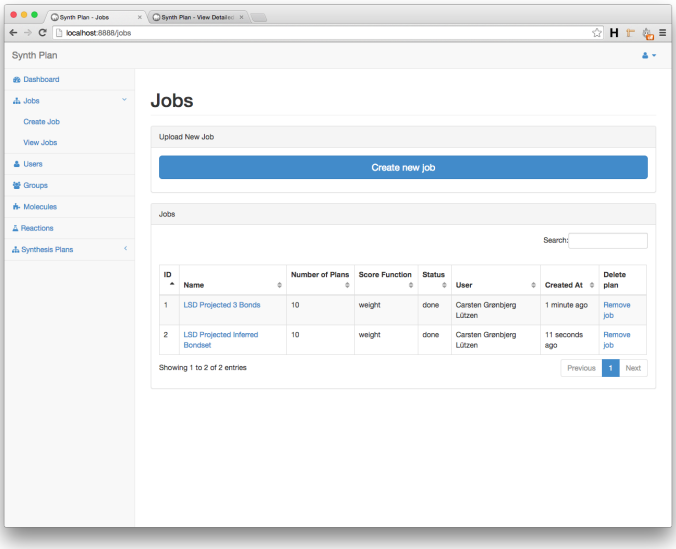


Figure 74: All jobs in the database.

6 Projecting Real Synthesis Plans to Skeletons

With the introduction of both SynthDB and the SynthWorker a valuable asset would be to be able to compare the actual synthesis plans with the generated skeleton plans to determine if they, in theory, could be improved according to the SynthWorker metrics. This would provide the valuable link between real life chemistry and SynthWorker. Another valuable benefit is traceability. With projections one would be able to trace each individual atom in the entire plan. This could then be applied in the lab with marked atoms to see if all reactions react as intended. This would also act as a balance check, hence if we cant make a projection of the reaction, then the reaction is not likely to succeed in the lab.

With a projection and a real synthesis plan, beautiful figures can be made showing all skeleton synthesis plans with the same bondset or same size of bondset compared to the actual bondset from the real plan.

6.1 SynthMapper

To solve this problem we have implemented a simple, yet powerful, tool called SynthMapper. SynthMapper can take a synthesis plan in the SPF format. This specifies a skeleton for the goal compound and all of the educts and products of the reactions. Given this SynthMapper should be able to devise a new SPM for SynthWorker.

Given this SynthWorker can determine how good the plan is and what the best plan for the given bondset is.

In the literature the definition of a skeleton is not firmly established with clear and concise rules, hence the user needs to input the skeleton of the target molecule. This gives the largest degree of freedom and makes SynthMapper more flexible. We give the user the possibility to define a bondset, but we also infer a skeleton and as a result the chemist have absolute freedom when defining the skeleton.

6.1.1 Inferred Skeleton

An interesting feature with the mapping is that we can trace individual atoms through all reactions from the target compound to the input compounds. With this information we can actually construct a skeleton with all atoms that are traceable through the entire synthesis plan.

The inferred skeleton is constructed by taken the parts of the input compounds present in the target compound and use this. In this way we know exactly what is the actual skeleton, and what is functional groups.

This skeleton shows several interesting properties, since we can easily know where the input compounds, or fragments of the input compounds are located in the skeleton. But it is important to remember that the inferred skeleton is only local for the given plan, since it reflects the input compounds which might be different for other plans.

The SynthMapper procedure consists of several steps:

- Parse the SPF file and extract the goal compound skeleton.

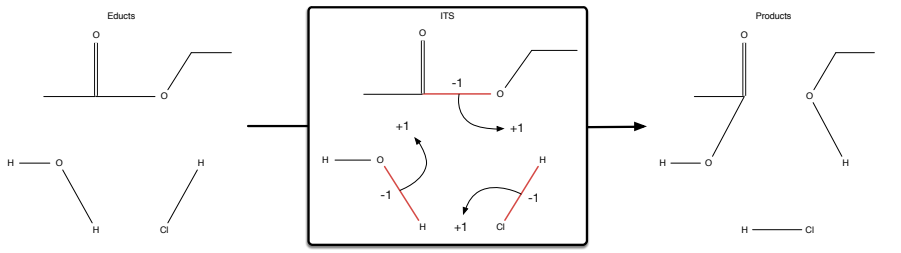


Figure 75: A reaction between ethyl acetate and water with hydrogen chloride as a catalyst. The products are ethanol-acetatic and an acid.

- Use the Atom-Atom mapper to determine which bonds were broken in the reaction for all reactions in the plan.
- Map all broken bonds into the skeleton of the goal compound.
- Construct inferred skeleton from target compound and input compound mapping.

6.1.2 Atom-Atom Mapping

The goal of an Atom-Atom Mapper is to hint whether or not a reaction is possible or not, by looking at how the atoms must relocate to make the transition. In other words one might say that a chemical reaction is nothing else than rearrangement of bonds. With this phrasing it becomes clear that there must exist a one to one correspondence between the atoms in the educts and in the products.

The changes in the molecules can be called imaginary transition state (ITS) [17] and this is exactly what we want SynthMapper to output. This map can give a lot of other information like a classification of the reaction. Previous to Atom-Atom mappers these maps were inferred from partial knowledge. Now these can be generated and later verified from isotope labeling experiments in the lab or by a chemist. In recent time programs capable of mapping atoms have become increasingly popular and powerful.

One example of an ITS can be seen in Figure 75. Here we see the educts, the ITS and the products.

One of the newest and most promising approaches is to formulate the mapping as an ILP problem. This could lead to more than one mapping. Often the first mapping is the correct one, but that is not always true, which can often only be validated in the lab. Hence we selects the first mapping.

Several properties must be satisfied in order to have a valid mapping:

1. The mapping must preserve all atoms and their types in educt and product.
2. The total number of bonds must be preserved.
3. The changed bonds must constitute an alternating path, such that it cycles between adding a bond and removing a bond.

Figure 75 is a valid mapping using these definition since all atoms are preserved and the total number of bonds are preserved. In the ITS it is clear to see that we have an alternating cycle.

6.1.3 Our Approach

The main concept of the projection is shown in Algorithm 11.

Algorithm 11 Output a molecule and a bondset.

```
1:  function SynthMapper(file)
2:      plan ← parse(file)
3:      reactionQueue ← ∅
4:      reactionQueue.push(plan.rootReaction)
5:      bondset ← ∅
6:      while |reactionQueue| > 0 do
7:          reaction ← reactionQueue.pop()
8:          addedBonds ← AAMapper(reaction.educts, reaction.products)
9:          bondset.add(addedBonds)
10:     end while
11:     return {extractSkeleton(plan.goalCompound, bondset}
12: end function
```

In an actual implementation one needs to update the indices to be able to map it back to the skeleton at all times. This is trivial to do, but it will clutter the pseudocode.

The entire procedure takes some time, due to the use of an Atom-Atom mapper (AAM). The AAM we used in our project is designed and implemented by Uffe Thorsen. AAM is based on the model proposed by Latendresse [31], but in this version without weights. Hence all edges are given a weight of one.

6.2 Example of Acetal Formation

An example of a mapping from a set of educts to a set of products can bee seen in Figure 76 where the reaction Acetal Formation from the LSD plan is depicted.

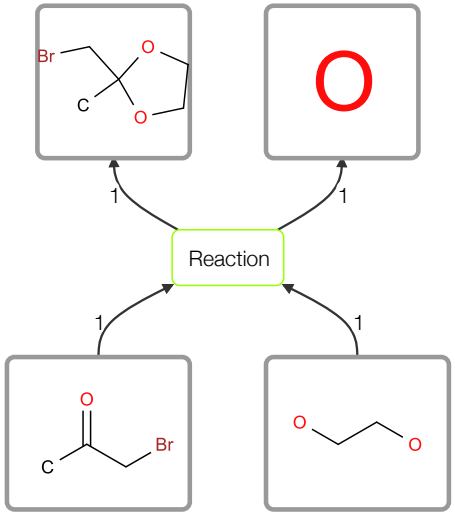


Figure 76: Acetal Formation. Simple reaction with two educts and two products.

We would like to determine what atoms are mapped to what atoms. This is done by calling SynthMapper. SynthMapper outputs a bijectional map. The map is constructed by moving as few bonds as possible. Using this map we can construct the mapping. The mapping can be seen in Figure 77. It is important to keep in mind that the depiction is not chemically valid! The position of the atoms are kept the same to make it clear what atoms goes where.

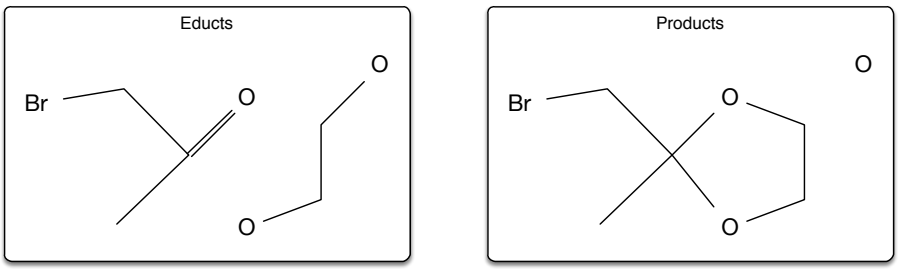


Figure 77: An mapping of the reaction Acetal Formation. The reaction can be seen in Figure 76.

6.3 Empirical Analysis of LSD

Lets look at a synthesis plan and create a projection it. We use the synthesis plan for LSD found in Listing A.1. Here we have 20 reactions. But how many of these are in fact working on the skeleton of LSD? And what reactions are the ones fixing bonds in the skeleton?

These questions are easily answered by SynthMapper. When we ask SynthDB to run a new SynthMapper job we find that three bonds are being fixed in the plan, we also get what the reactions are. This tells us that 3 reactions, out of 20 reaction, is fixing bonds and the rest is functionlization steps. The plan and the inferred-skeleton plan in the same figure can be seen in Figure 78 and the rest of the plan in Figure 79.

This also shows us how far, or near, our skeleton is are to the real world. If we take these three bonds and asks SynthWorker to use this as the bondset we get 6 plans out. The skeleton plan that fixes the bond in this order is ranked the best using the default cost function in SynthWorker.

The Figure 80 shows the best plan one can obtain when using the inferred skeleton from the LSD plan. It is worth to notice that this plan only contains affixation reactions and cyclization reactions.

6.3.1 Real LSD and Skeleton LSD

An interesting thing to look at is how the realistic LSD plan scores with other plans with the same bondset, and with generated plans with a bondset of the same size and input compounds of realistic size.

The best 2000 plans with score function weight are generated, limiting the bondset to size 3 as the real LSD bondset. Furthermore we restricted it to only allow input molecules of a realistic size. We have plotted these in Figure 81. Then we added the actual LSD plan to the plot. We also added the 2 of the possible plans with the same bondset in the plot. The last 4 plans was not among the first 2000 plans.

A empirical study of what the minimum allowed input size matters for the distribution and quality of the plans is presented. Again LSD is the molecule to be synthesized. All possible bondsets of size 3 is generated. The minimum allowed input size is varied. The minimum allowed input size is explained in Section 4.7.

In Figure 82 the cost of the first 10586 plans are plotted. A time limit of 15 minutes is used in this test. In Figure 83 the best plan with a minimum input size of one is shown. The best plan uses input compounds of size one, hence this plan will not be allowed when minimum input size is larger than one. The plan introduces a large input compound late in the plan. The large input compound also needs to be synthesized. Smaller input compounds is therefore wanted. A way to achieve this is increasing the minimum input size allowed. In Figure 82 two solutions clearly dominate the rest.

We increase the required minimum input size and rerun the experiment. In Figure 84 the distribution of the cost is shown, the distribution is different to the distribution found with a required minimum input size of one. One of the best plans is shown in Figure 85. The plan is constraint by a minimum input size of two. Therefore the largest input molecule is smaller than in Figure 83.

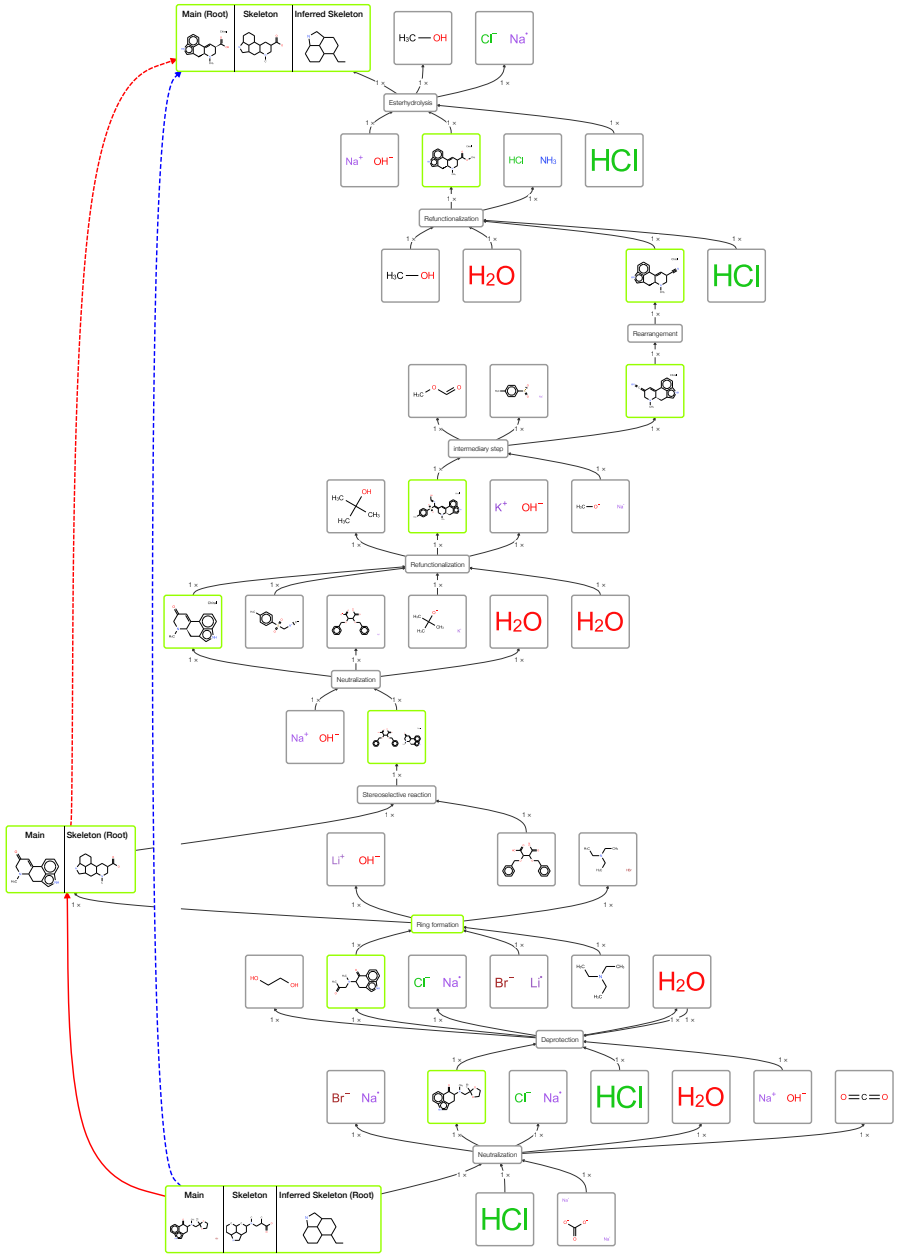


Figure 78: The top of the LSD plan with skeleton molecules added. The skeleton provided by the chemist is called skeleton and the red edges represent this skeleton. The inferred skeleton is shown in the applicable reactions and blue edges.

Increasing the required minimum input size to three leads to the distribution of cost shown in Figure 86. One of the best plans is shown in Figure 87. The largest input compound is smaller the largest input compound in both Figure 83

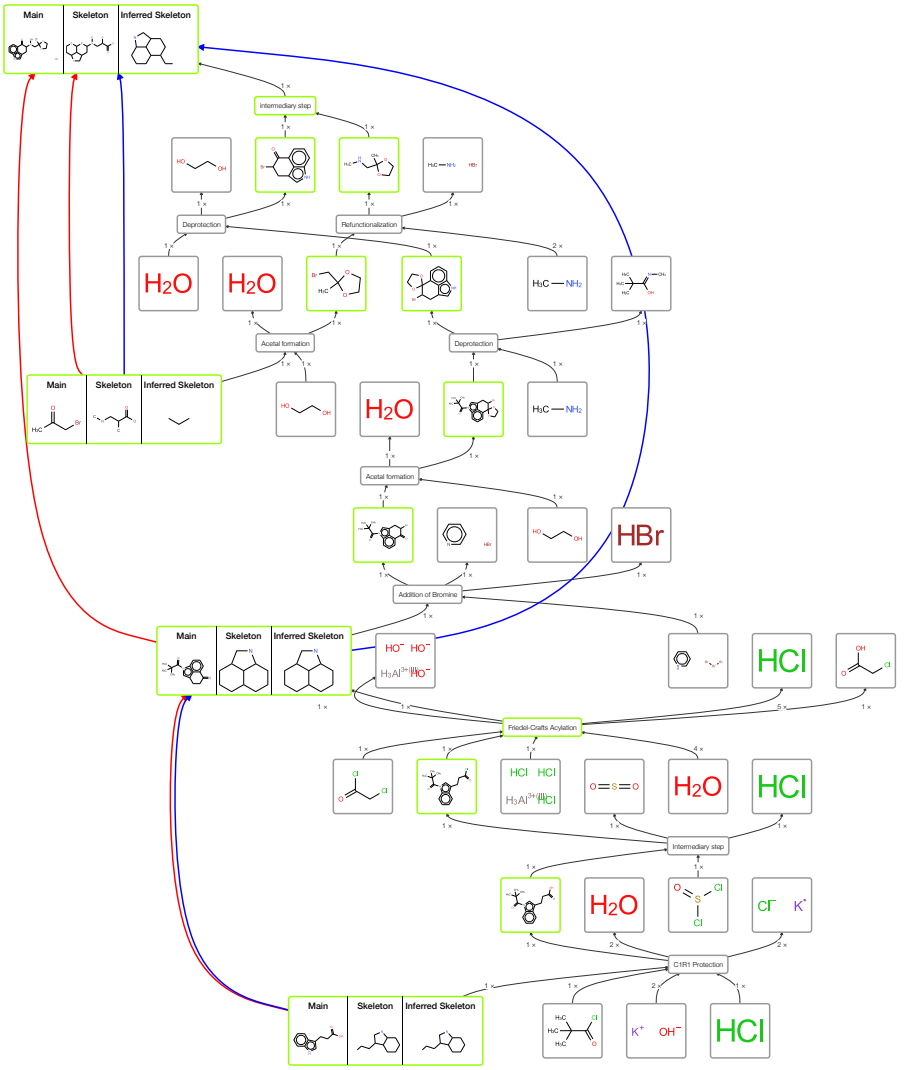


Figure 79: The bottom of the LSD plan with skeleton molecules added. The skeleton provided by the chemist is called skeleton and the red edges represent this skeleton. The inferred skeleton is shown in the applicable reactions and blue edges.

and Figure 85. Note all plans presented so far ends with a cyclization step in the plan.

The distribution of the cost for the plans generated with a required minimum size of four, five, six and seven is shown in Figure 88, Figure 90, Figure 91 and Figure 92 respectively. For all these distributions one of the best plans is shown in Figure 89. The smallest starting material in the plan shown in Figure 89 is of size 9.

The synthesis plan in Figure 89 contains 3 construction reactions. As mentioned above, the smallest starting material is of size 9. However this product

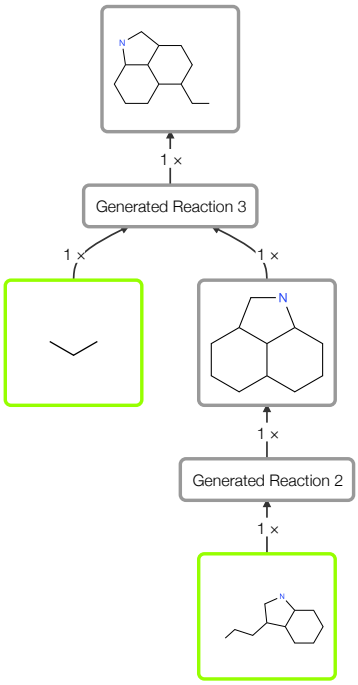


Figure 80: This plan is the best plan that was obtained when using the inferred skeleton from the LSD plan used in Figure 78 and Figure 79. This is the plan marked as a red cross on Figure 81.

is not produced or exists in nature, it has to be synthesized. The chemical compound the starting compound represents is called 3-indolepropionic acid. A synthesis plan for this compound can be found in [1]. The main component of 3-indolepropionic acid is indole which bacteria can produce. This can be modeled in two ways, either to examine how to synthesize the large starting compounds in Figure 89. Another way, is to add more bonds to the bondset. This is not guaranteed to lead to the actual way 3-indolepropionic acid is synthesized. The conclusion is that exploring all possible 3-bondsets does not lead a synthesis plan where all compounds can be found in nature. But it might lead to a synthesis plan which input compounds can be bought synthesized.

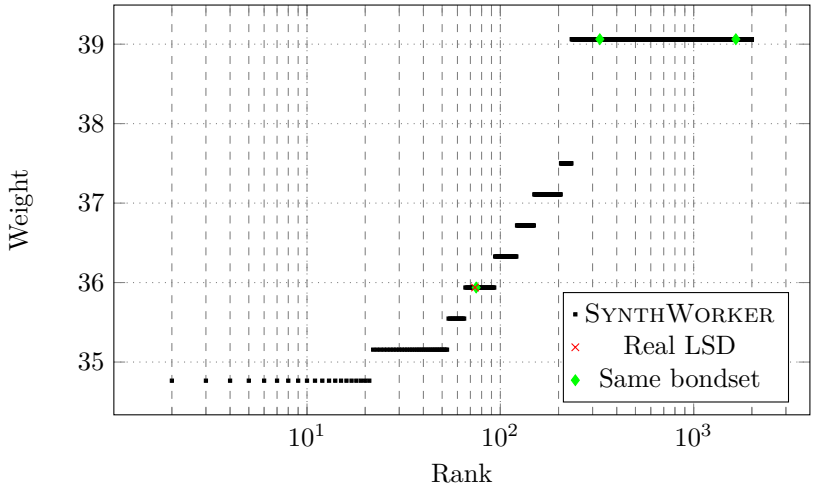


Figure 81: 2000 generated LSD plans with bondset of size 3 plotted with the real LSD plan and the projected skeleton plans.

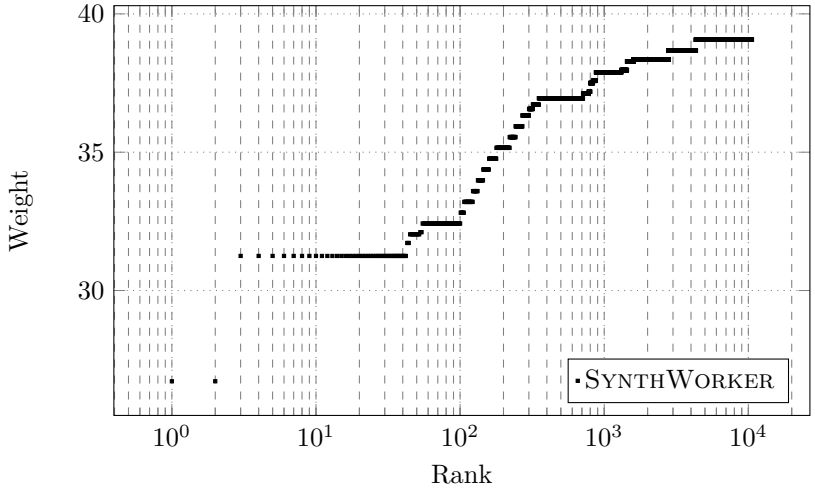


Figure 82: The best 10586 generated LSD plans with bondset of size 3 plotted with a minimum input size of 1, hence not all plans are plotted. The best plan can be seen in Figure 83.

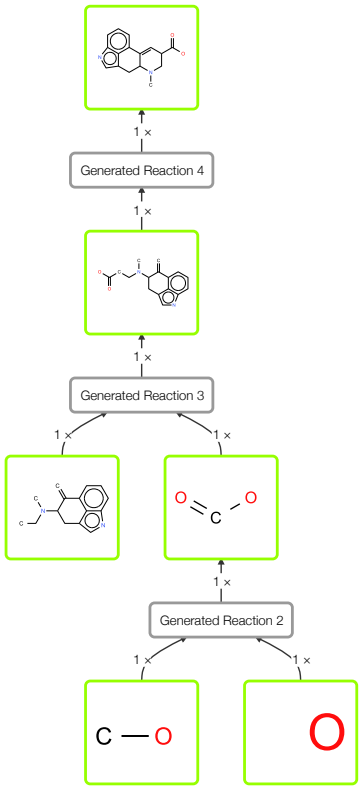


Figure 83: Best plan generated with a bondset of size 3 and a minimum input size of 1.

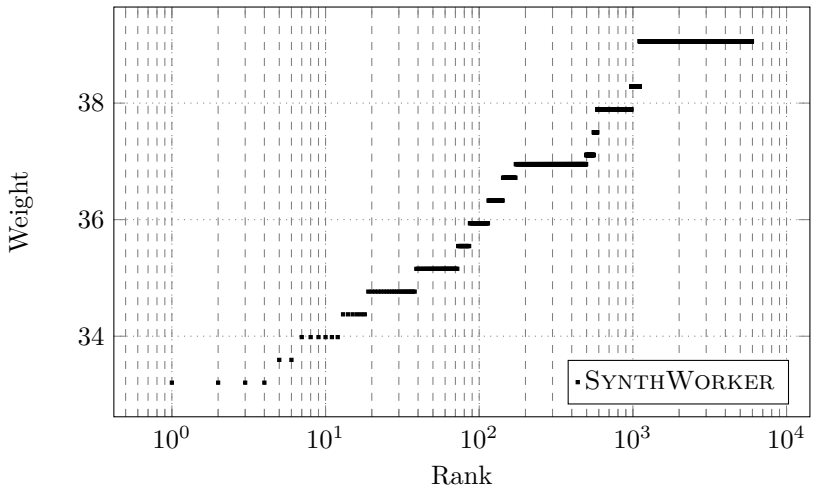


Figure 84: 5952 generated LSD plans with bondset of size 3 plotted with a minimum input size of 2. The best plan can be seen in Figure 85.

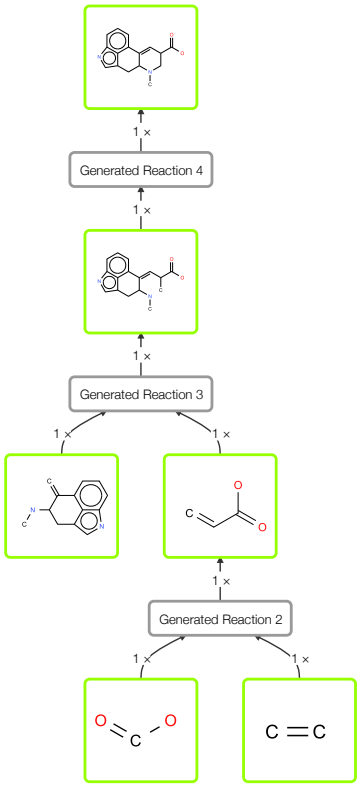


Figure 85: Best plan generated with a bondset of size 3 and a minimum input size of 2.

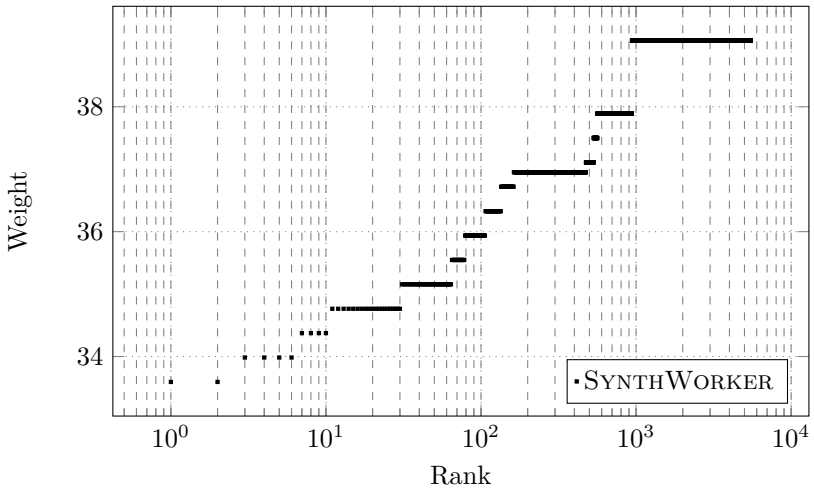


Figure 86: 5502 generated LSD plans with bondset of size 3 plotted with a minimum input size of 3. The best plan can be seen in Figure 87.

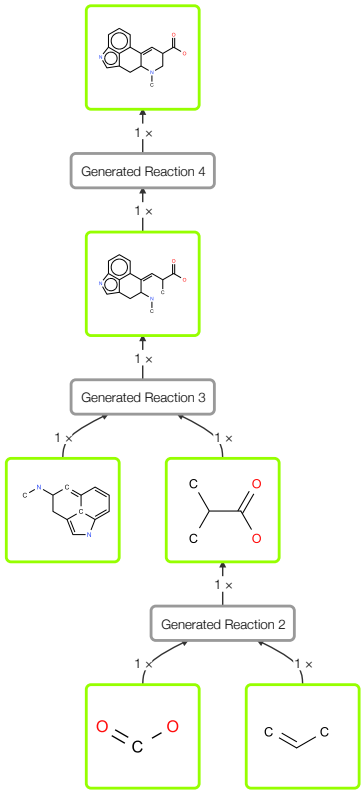


Figure 87: Best plan generated with a bondset of size 3 and a minimum input size of 3.

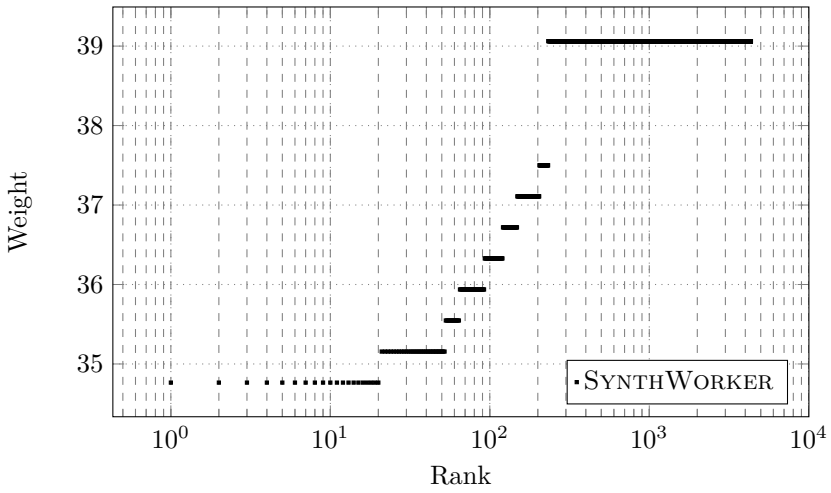


Figure 88: 4338 generated LSD plans with bondset of size 3 plotted with a minimum input size of 4. The best plan can be seen in Figure 89.

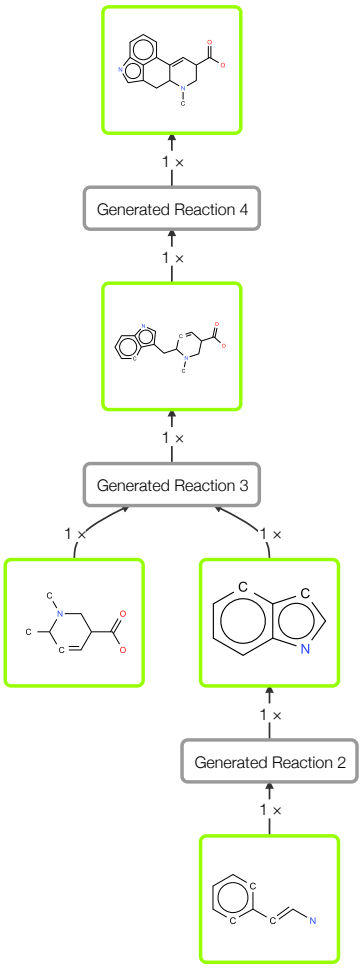


Figure 89: Best plan generated with a bondset of size 3 and a minimum input size of 4, 5, 6 and 7.

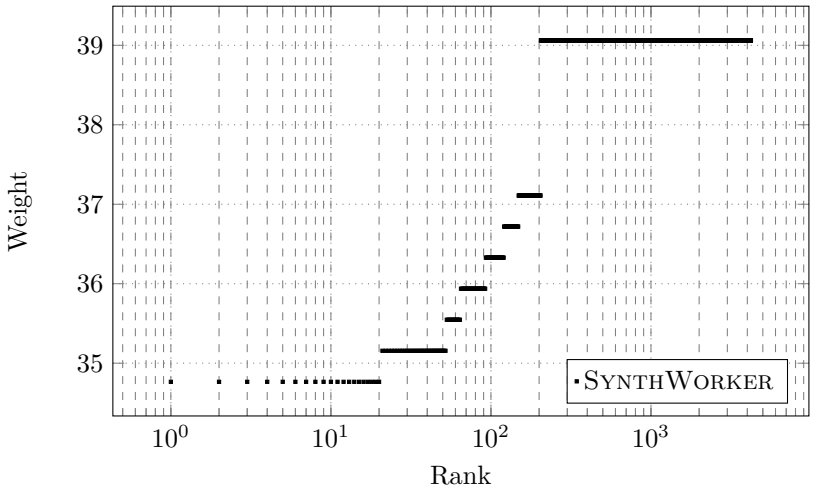


Figure 90: 4212 generated LSD plans with bondset of size 3 plotted with a minimum input size of 5. The best plan is the same as with a minimum input size of 4.

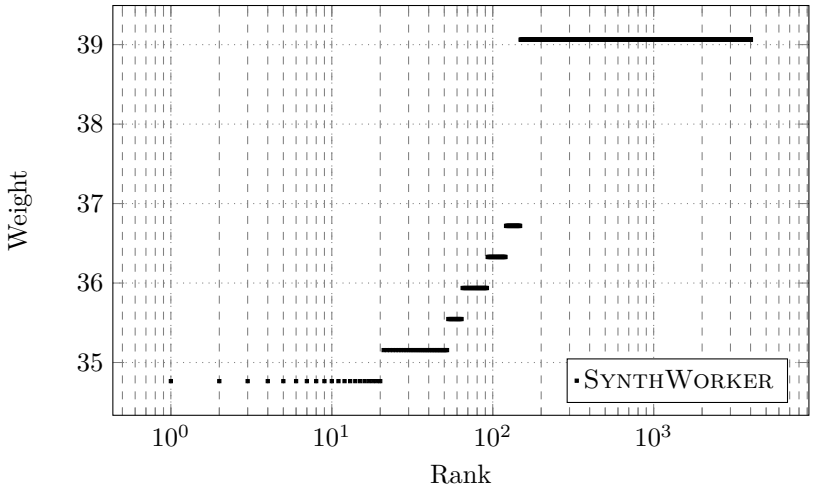


Figure 91: 4020 generated LSD plans with bondset of size 3 plotted with a minimum input size of 6. The best plan is the same as with a minimum input size of 4.

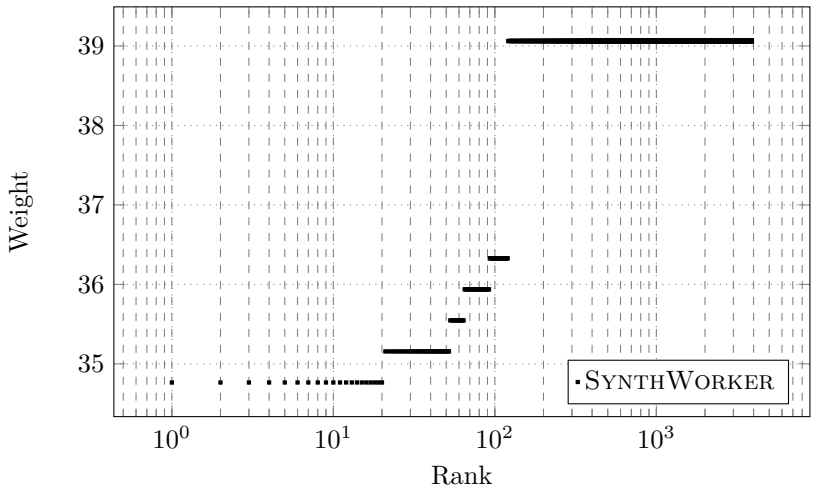


Figure 92: 3906 generated LSD plans with bondset of size 3 plotted with a minimum input size of 7. The best plan is the same as with a minimum input size of 4.

7 Conclusion and Future Work

Within synthesis planning very few formalized approaches exists, this is an attempt to formalize methods within the field synthesis planning.

We constructed a model for representing the superposition of all synthesis plans. This model is called the HoSP (Hypergraph of Synthesis Plans). We define several methods to construct the superposition. Each method has different properties, and tradeoffs. Analyzing the HoSP can support the choice of a bondset.

The K best synthesis plans in the HoSP can be enumerated and analyzed, with several known cost functions for synthesis plans, by SynthWorker.

SynthWorker can analyze the order of construction reactions with respect to optimality. In addition multiple bondsets can be analyzed. By enumerating the K best synthesis plans, the chemist is given alternative synthetic paths if a reaction in the best synthesis plan is infeasible. If a good bondset is not known, SynthWorker suggests good bondsets by enumerating the K best synthesis plans over all possible bondsets.

Several databases for synthesis plans exist, they are closed source and expensive to use. We introduce the first open source format for storing synthesis plans SPF. We introduce SynthDB which is capable of storing and visualizing synthesis plans in SPF. SynthDB enables the chemist to cross-reference molecules and reactions across the database. In SynthDB it is possible to download individual synthesis plans, in SPF.

In order to analyze the quality of already known synthesis plans, we introduce a method for projecting real life synthesis plans into our model of a synthesis plan. This projection is done by SynthMapper. Since all reactions in SynthDB are balanced, an atom-atom mapping is used to automatically infer the mapping of atoms across a reaction. Such a mapping enables us to identify whether a reaction is a construction or functionalization reaction. Hence a skeleton can be automatically inferred for a reactions and molecules and a comparison of real life synthesis plans and our modeled synthesis plans is possible.

Recently it has become more important to invent environmentally friendly reactions [3]. An obvious next step would be to reflected this in the metrics applied to synthesis plans in SynthWorker.

So far the model for synthesis plans does not consider functionalization of molecules. Seen in the empirical analysis of LSD the majority of reactions in real life synthesis plans are functionalization reactions. Supporting functionlization would be a natural addition to the model. Several strategies could examined, one is described by Hendrickson [21]. This methods categorizes each reaction.

Future research direction could be integration of functionalization in the current modeling approach. Protecting a group and unprotecting it later is not a local decision, which opens up for challenging combinatorial problems.

A very recent research line within wet lab approaches for synthesis planning is DNA-templated synthesis planning [33]. DNA-templated synthesis planning is used to combine molecules by using DNA strands to control the vicinity of molecules. Therefore classical functionalization reactions are no longer necessary. Our modeling approach can be used as a direct input for inferring wet lab protocols based on DNA-templated synthesis planning.

This thesis makes important first steps towards formalizing the methods and applies a mathematical and computer science approach to field of synthesis planning.

A Appendix

A.1 LSD plan as SPF

```
# Last changed timestamp:
# Last changed author:
# Lit: Moldvai, I.; Temesvari-Major, E.; Incze, M.;
    Szentirmay, E.; Szanty, C.
#
#      J. Org. Chem. 2004
# DOI:
# Aut: Lukas Bartonek <a0906858@unet.univie.ac.at>

skeleton: InChI=1S/C16H16N2O2/c1-18-8-10(16(19)20)5
    -12-11-3-2-4-13-15(11)9(7-17-13)6-14(12)18/
    h2-5,7,10,14,17H,6,8H2,1H3,(H,19,20)/t10-,14-/m1/s1


compounds:
comp1 InChI=1S/C11H11NO2/c13-11(14)6-5-8-7-12-10-4-2-1-3-9(8
    )10/h1-4,7,12H,5-6H2,(H,13,14)
comp2 InChI=1S/C5H9ClO/c1-5(2,3)4(6)7/h1-3H3
comp3 InChI=1S/K.H2O/h;1H2/q+1;/p-1
comp4 InChI=1S/ClH/h1H
comp5 InChI=1S/C16H19NO3/c1-16(2,3)15(20)17-10-11(8-9-14(18)
    19)12-6-4-5-7-13(12)17/h4-7,10H,8-9H2,1-3H3,(H,18,19)
comp6 InChI=1S/H2O/h1H2
comp7 InChI=1S/ClH.K/h1H;/q;+1/p-1
comp8 InChI=1S/C16H19NO3/c1-16(2,3)15(20)17-10-11(8-9-14(18)
    19)12-6-4-5-7-13(12)17/h4-7,10H,8-9H2,1-3H3,(H,18,19)
comp9 InChI=1S/Cl2OS/c1-4(2)3
comp10 InChI=1S/C16H18ClNO2/c1-16(2,3)15(20)18
    -10-11(8-9-14(17)19)12-6-4-5-7-13(12)18/
    h4-7,10H,8-9H2,1-3H3
comp11 InChI=1S/O2S/c1-3-2
comp12 InChI=1S/ClH/h1H
comp13 InChI=1S/C16H18ClNO2/c1-16(2,3)15(20)18
    -10-11(8-9-14(17)19)12-6-4-5-7-13(12)18/
    h4-7,10H,8-9H2,1-3H3
comp14 InChI=1S/C2H2Cl2O/c3-1-2(4)5/h1H2
comp15 InChI=1S/Al.3ClH/h;3*1H/q+3;;;/p-3
comp16 InChI=1S/H2O/h1H2
comp17 InChI=1S/C16H17NO2/c1-16(2,3)15(19)17-9-10-7-8-13(18)
    11-5-4-6-12(17)14(10)11/h4-6,9H,7-8H2,1-3H3
comp18 InChI=1S/Al.3H2O/h;3*1H2/q+3;;;/p-3
comp19 InChI=1S/ClH/h1H
comp20 InChI=1S/C2H3ClO2/c3-1-2(4)5/h1H2,(H,4,5)
comp21 InChI=1S/C16H17NO2/c1-16(2,3)15(19)17-9-10-7-8-13(18)
    11-5-4-6-12(17)14(10)11/h4-6,9H,7-8H2,1-3H3
comp22 InChI=1S/C5H5N.Br3/c1-2-4-6-5-3-1;1-3-2/h1-5H;/q;-1/p
    +1
comp23 InChI=1S/C16H16BrNO2/c1-16(2,3)15(20)18-8-9-7-11(17)
    14(19)10-5-4-6-12(18)13(9)10/h4-6,8,11H,7H2,1-3H3
```

```
comp24 InChI=1S/C5H5N.BrH/c1-2-4-6-5-3-1;/h1-5H;1H
comp25 InChI=1S/BrH/h1H
comp26 InChI=1S/C16H16BrNO2/c1-16(2,3)15(20)18-8-9-7-11(17)
    14(19)10-5-4-6-12(18)13(9)10/h4-6,8,11H,7H2,1-3H3
comp27 InChI=1S/C2H6O2/c3-1-2-4/h3-4H,1-2H2
comp28 InChI=1S/C18H20BrNO3/c1-17(2,3)16(21)20-10-11-9-14(19
    )18(22-7-8-23-18)12-5-4-6-13(20)15(11)12/
    h4-6,10,14H,7-9H2,1-3H3
comp29 InChI=1S/H2O/h1H2
comp30 InChI=1S/C18H20BrNO3/c1-17(2,3)16(21)20-10-11-9-14(19
    )18(22-7-8-23-18)12-5-4-6-13(20)15(11)12/
    h4-6,10,14H,7-9H2,1-3H3
comp31 InChI=1S/CH5N/c1-2/h2H2,1H3
comp32 InChI=1S/C13H12BrNO2/c14-11-6-8-7-15-10-3-1-2-9(12(8)
    10)13(11)16-4-5-17-13/h1-3,7,11,15H,4-6H2
comp33 InChI=1S/C6H13NO/c1-6(2,3)5(8)7-4/h1-4H3,(H,7,8)
comp34 InChI=1S/C13H12BrNO2/c14-11-6-8-7-15-10-3-1-2-9(12(8)
    10)13(11)16-4-5-17-13/h1-3,7,11,15H,4-6H2
comp35 InChI=1S/H2O/h1H2
comp36 InChI=1S/C11H8BrNO/c12-8-4-6-5-13-9-3-1-2-7(10(6)9)11
    (8)14/h1-3,5,8,13H,4H2
comp37 InChI=1S/C2H6O2/c3-1-2-4/h3-4H,1-2H2
comp38 InChI=1S/C11H8BrNO/c12-8-4-6-5-13-9-3-1-2-7(10(6)9)11
    (8)14/h1-3,5,8,13H,4H2
comp39 InChI=1S/C6H13NO2/c1-6(5-7-2)8-3-4-9-6/
    h7H,3-5H2,1-2H3
comp40 InChI=1S/C17H20N2O3.BrH/c1-17(21-6-7-22-17)10-19(2)14
    -8-11-9-18-13-5-3-4-12(15(11)13)16(14)20;/
    h3-5,9,14,18H,6-8,10H2,1-2H3;1H
comp41 InChI=1S/C17H20N2O3.BrH/c1-17(21-6-7-22-17)10-19(2)14
    -8-11-9-18-13-5-3-4-12(15(11)13)16(14)20;/
    h3-5,9,14,18H,6-8,10H2,1-2H3;1H
comp42 InChI=1S/ClH/h1H
comp43 InChI=1S/CH2O3.2Na/c2-1(3)4;;/h(H2,2,3,4);;/q;2*+1/
    p-2
comp44 InChI=1S/C17H20N2O3/c1-17(21-6-7-22-17)10-19(2)14
    -8-11-9-18-13-5-3-4-12(15(11)13)16(14)20/
    h3-5,9,14,18H,6-8,10H2,1-2H3
comp45 InChI=1S/BrH.Na/h1H;/q;+1/p-1
comp46 InChI=1S/ClH.Na/h1H;/q;+1/p-1
comp47 InChI=1S/H2O/h1H2
comp48 InChI=1S/CO2/c2-1-3
comp49 InChI=1S/C17H20N2O3/c1-17(21-6-7-22-17)10-19(2)14
    -8-11-9-18-13-5-3-4-12(15(11)13)16(14)20/
    h3-5,9,14,18H,6-8,10H2,1-2H3
comp50 InChI=1S/H2O/h1H2
comp51 InChI=1S/ClH/h1H
comp52 InChI=1S/Na.H2O/h;1H2/q+1;/p-1
comp53 InChI=1S/C15H16N2O2/c1-9(18)8-17(2)13
    -6-10-7-16-12-5-3-4-11(14(10)12)15(13)19/
    h3-5,7,13,16H,6,8H2,1-2H3
comp54 InChI=1S/C2H6O2/c3-1-2-4/h3-4H,1-2H2
comp55 InChI=1S/ClH.Na/h1H;/q;+1/p-1
comp56 InChI=1S/H2O/h1H2
```

```
comp57 InChI=1S/C15H16N2O2/c1-9(18)8-17(2)13
    -6-10-7-16-12-5-3-4-11(14(10)12)15(13)19/
    h3-5,7,13,16H,6,8H2,1-2H3
comp58 InChI=1S/BrH.Li/h1H;/q;+1/p-1
comp59 InChI=1S/C6H15N/c1-4-7(5-2)6-3/h4-6H2,1-3H3
comp60 InChI=1S/C15H14N2O/c1-17-8-10(18)6
    -12-11-3-2-4-13-15(11)9(7-16-13)5-14(12)17/
    h2-4,6-7,14,16H,5,8H2,1H3
comp61 InChI=1S/Li.H2O/h;1H2/q+1;/p-1
comp62 InChI=1S/C6H15N.BrH/c1-4-7(5-2)6-3;/h4-6H2,1-3H3;1H
comp63 InChI=1S/C15H14N2O/c1-17-8-10(18)6
    -12-11-3-2-4-13-15(11)9(7-16-13)5-14(12)17/
    h2-4,6-7,14,16H,5,8H2,1H3
comp64 InChI=1S/C18H18O6/c19-17(20)15(23-11-13-7-3-1-4-8-13)
    16(18(21)22)24-12-14-9-5-2-6-10-14/
    h1-10,15-16H,11-12H2,(H,19,20)(H,21,22)
comp65 InChI=1S/C18H18O6.C15H14N2O/c19-17(20)15
    (23-11-13-7-3-1-4-8-13)16(18(21)22)24
    -12-14-9-5-2-6-10-14;1-17-8-10(18)6-12-11-3-2-4-13-15(11
    )9(7-16-13)5-14(12)17/h1-10,15-16H,11-12H2,(H,19,20)
    (H,21,22);2-4,6-7,14,16H,5,8H2,1H3/t;14-/m.1/s1
comp66 InChI=1S/C18H18O6.C15H14N2O/c19-17(20)15
    (23-11-13-7-3-1-4-8-13)16(18(21)22)24
    -12-14-9-5-2-6-10-14;1-17-8-10(18)6-12-11-3-2-4-13-15(11
    )9(7-16-13)5-14(12)17/h1-10,15-16H,11-12H2,(H,19,20)
    (H,21,22);2-4,6-7,14,16H,5,8H2,1H3/t;14-/m.1/s1
comp67 InChI=1S/Na.H2O/h;1H2/q+1;/p-1
comp68 InChI=1S/C15H14N2O/c1-17-8-10(18)6
    -12-11-3-2-4-13-15(11)9(7-16-13)5-14(12)17/
    h2-4,6-7,14,16H,5,8H2,1H3/t14-/m1/s1
comp69 InChI=1S/C18H18O6.Na/c19-17(20)15
    (23-11-13-7-3-1-4-8-13)16(18(21)22)24
    -12-14-9-5-2-6-10-14;/h1-10,15-16H,11-12H2,(H,19,20)
    (H,21,22);/q;+1/p-1
comp70 InChI=1S/H2O/h1H2
comp71 InChI=1S/C15H14N2O/c1-17-8-10(18)6
    -12-11-3-2-4-13-15(11)9(7-16-13)5-14(12)17/
    h2-4,6-7,14,16H,5,8H2,1H3/t14-/m1/s1
comp72 InChI=1S/C9H9NO2S/c1-8-3-5-9(6-4-8)13(11,12)7-10-2/
    h3-6H,7H2,1H3
comp73 InChI=1S/C4H9O.K/c1-4(2,3)5;/h1-3H3;/q-1;+1
comp74 InChI=1S/H2O/h1H2
comp75 InChI=1S/C24H23N3O3S/c1-15-6-8-18(9-7-15)31(29,30)24
    (26-14-28)17-10-20-19-4-3-5-21-23(19)16(12-25-21)11
    -22(20)27(2)13-17/
    h3-10,12,14,22,25H,11,13H2,1-2H3,(H,26,28)/t22-/m1/s1
comp76 InChI=1S/C4H10O/c1-4(2,3)5/h5H,1-3H3
comp77 InChI=1S/K.H2O/h;1H2/q+1;/p-1
comp78 InChI=1S/C24H23N3O3S/c1-15-6-8-18(9-7-15)31(29,30)24
    (26-14-28)17-10-20-19-4-3-5-21-23(19)16(12-25-21)11
    -22(20)27(2)13-17/
    h3-10,12,14,22,25H,11,13H2,1-2H3,(H,26,28)/t22-/m1/s1
comp79 InChI=1S/CH3O.Na/c1-2;/h1H3;/q-1;+1
```

```
comp80 InChI=1S/C16H15N3/c1-19-9-10(7-17)5
    -13-12-3-2-4-14-16(12)11(8-18-14)6-15(13)19/
    h2-5,8,15,17-18H,6,9H2,1H3/t15-/m1/s1
comp81 InChI=1S/C2H4O2/c1-4-2-3/h2H,1H3
comp82 InChI=1S/C7H7O2S.Na/c1-6-2-4-7(5-3-6)10(8)9;/
    h2-5H,1H3;/q-1;+1
comp83 InChI=1S/C16H15N3/c1-19-9-10(7-17)5
    -13-12-3-2-4-14-16(12)11(8-18-14)6-15(13)19/
    h2-5,8,15,17-18H,6,9H2,1H3/t15-/m1/s1
comp84 InChI=1S/C16H15N3/c1-19-9-10(7-17)5
    -13-12-3-2-4-14-16(12)11(8-18-14)6-15(13)19/
    h2-5,8,10,15,18H,6,9H2,1H3/t10?,15-/m1/s1
comp85 InChI=1S/C16H15N3/c1-19-9-10(7-17)5
    -13-12-3-2-4-14-16(12)11(8-18-14)6-15(13)19/
    h2-5,8,10,15,18H,6,9H2,1H3/t10?,15-/m1/s1
comp86 InChI=1S/CH4O/c1-2/h2H,1H3
comp87 InChI=1S/H2O/h1H2
comp88 InChI=1S/ClH/h1H
comp89 InChI=1S/C17H18N2O2/c1-19-9-11(17(20)21-2)6
    -13-12-4-3-5-14-16(12)10(8-18-14)7-15(13)19/
    h3-6,8,11,15,18H,7,9H2,1-2H3/t11?,15-/m1/s1
comp90 InChI=1S/ClH.H3N/h1H;1H3
comp91 InChI=1S/C17H18N2O2/c1-19-9-11(17(20)21-2)6
    -13-12-4-3-5-14-16(12)10(8-18-14)7-15(13)19/
    h3-6,8,11,15,18H,7,9H2,1-2H3/t11?,15-/m1/s1
comp92 InChI=1S/Na.H2O/h;1H2/q+1;/p-1
comp93 InChI=1S/ClH/h1H
comp94 InChI=1S/C16H16N2O2/c1-18-8-10(16(19)20)5
    -12-11-3-2-4-13-15(11)9(7-17-13)6-14(12)18/
    h2-5,7,10,14,17H,6,8H2,1H3,(H,19,20)/t10-,14-/m1/s1
comp95 InChI=1S/CH4O/c1-2/h2H,1H3
comp96 InChI=1S/ClH.Na/h1H;/q;+1/p-1
comp97 InChI=1S/C3H5BrO/c1-3(5)2-4/h2H2,1H3
comp98 InChI=1S/C2H6O2/c3-1-2-4/h3-4H,1-2H2
comp99 InChI=1S/C5H9BrO2/c1-5(4-6)7-2-3-8-5/h2-4H2,1H3
comp100 InChI=1S/H2O/h1H2
comp101 InChI=1S/C5H9BrO2/c1-5(4-6)7-2-3-8-5/h2-4H2,1H3
comp102 InChI=1S/CH5N/c1-2/h2H2,1H3
comp103 InChI=1S/C6H13NO2/c1-6(5-7-2)8-3-4-9-6/
    h7H,3-5H2,1-2H3
comp104 InChI=1S/CH5N.BrH/c1-2;/h2H2,1H3;1H


reactions:
R01 "Protection"
skeletonOut InChI=1S/C16H19NO3/c1-16(2,3)15(20)17
    -10-11(8-9-14(18)19)12-6-4-5-7-13(12)17/
    h4-7,10H,8-9H2,1-3H3,(H,18,19)
|comp1| + comp2 + 2 * comp3 + comp4 -> |comp5| + 2 * comp6 +
     2 * comp7
@yield 78%

R02 "intermediary step"
```

```
skeletonOut InChI=1S/C16H18ClNO2/c1-16(2,3)15(20)18
    -10-11(8-9-14(17)19)12-6-4-5-7-13(12)18/
    h4-7,10H,8-9H2,1-3H3
|comp8| + comp9 -> |comp10| + comp11 + comp12
@yield R02-R03 55%

R03 "Friedel-Crafts Acylation"
skeletonOut InChI=1S/C16H17NO2/c1-16(2,3)15(19)17
    -9-10-7-8-13(18)11-5-4-6-12(17)14(10)11/
    h4-6,9H,7-8H2,1-3H3
|comp13| + comp14 + comp15 + 4 * comp16 -> |comp17| + comp18
     + 5 * comp19 + comp20
@yield R02-R03 55%

R04 "Addition of Bromine"
skeletonOut InChI=1S/C16H16BrNO2/c1-16(2,3)15(20)18
    -8-9-7-11(17)14(19)10-5-4-6-12(18)13(9)10/
    h4-6,8,11H,7H2,1-3H3
|comp21| + comp22 -> |comp23| + comp24 + comp25
@yield 85% cat light

R05 "Acetal formation"
skeletonOut InChI=1S/C18H20BrNO3/c1-17(2,3)16(21)20
    -10-11-9-14(19)18(22-7-8-23-18)12-5-4-6-13(20)15(11)12/
    h4-6,10,14H,7-9H2,1-3H3
|comp26| + comp27 -> |comp28| + comp29
@yield 81% cat pTsOH

R06 "Deprotection"
skeletonOut InChI=1S/C13H12BrNO2/
    c14-11-6-8-7-15-10-3-1-2-9(12(8)10)13(11)16-4-5-17-13/
    h1-3,7,11,15H,4-6H2
|comp30| + comp31 -> |comp32| + comp33
@yield 88%

R07 "Deprotection"
skeletonOut InChI=1S/C11H8BrNO/c12-8-4-6-5-13-9-3-1-2-7(10(6
    )9)11(8)14/h1-3,5,8,13H,4H2
|comp34| + comp35 -> |comp36| + comp37
@yield 97% cat HCl

R08 "intermediary step"
skeletonOut InChI=1S/C17H20N2O3.BrH/c1-17(21-6-7-22-17)10
    -19(2)14-8-11-9-18-13-5-3-4-12(15(11)13)16(14)20;/
    h3-5,9,14,18H,6-8,10H2,1-2H3;1H
|comp38| + |comp39| -> |comp40|
@yield R08-R09 56%

R09 "Neutralization"
skeletonOut InChI=1S/C17H20N2O3/c1-17(21-6-7-22-17)10-19(2)
    14-8-11-9-18-13-5-3-4-12(15(11)13)16(14)20/
    h3-5,9,14,18H,6-8,10H2,1-2H3
|comp41| + comp42 + comp43 -> |comp44| + comp45 + comp46 +
    comp47 + comp48
```

```
@yield R08-R09 56%

R10 "Deprotection"
skeletonOut InChI=1S/C15H16N2O2/c1-9(18)8-17(2)13
    -6-10-7-16-12-5-3-4-11(14(10)12)15(13)19/
    h3-5,7,13,16H,6,8H2,1-2H3
|comp49| + comp50 + comp51 + comp52 -> |comp53| + comp54 +
    comp55 + comp56
@yield 100%

R11 "Ring formation"
skeletonOut InChI=1S/C15H14N2O/c1-17-8-10(18)6
    -12-11-3-2-4-13-15(11)9(7-16-13)5-14(12)17/
    h2-4,6-7,14,16H,5,8H2,1H3
|comp57| + comp58 + comp59 -> |comp60| + comp61 + comp62
@yield 60%

R12 "stereoselective reaction"
skeletonOut InChI=1S/C22H26O2.C15H14N2O/c1-17(2)21
    (23-15-19-11-7-5-8-12-19)22(18(3)4)24
    -16-20-13-9-6-10-14-20;1-17-8-10(18)6
    -12-11-3-2-4-13-15(11)9(7-16-13)5-14(12)17/
    h5-14,21-22H,1,3,15-16H2,2,4H3;2-4,6-7,14,16H,5,8H2,1H3/
    t;14-/m.1/s1
|comp63| + comp64 -> |comp65|
@yield 79%

R13 "Neutralization"
skeletonOut InChI=1S/C15H14N2O/c1-17-8-10(18)6
    -12-11-3-2-4-13-15(11)9(7-16-13)5-14(12)17/
    h2-4,6-7,14,16H,5,8H2,1H3/t14-/m1/s1
|comp66| + comp67 -> |comp68| + comp69 + comp70
@yield 38%

R14 "Refunctionalization"
skeletonOut InChI=1S/C24H23N3O3S/c1-15-6-8-18(9-7-15)31
    (29,30)24(26-14-28)17-10-20-19-4-3-5-21-23(19)16
    (12-25-21)11-22(20)27(2)13-17/
    h3-10,12,14,22,25H,11,13H2,1-2H3,(H,26,28)/t22-/m1/s1
|comp71| + comp72 + comp73 + comp74 -> |comp75| + comp76 +
    comp77
@yield 77%

R15 "intermediary step"
skeletonOut InChI=1S/C16H15N3/c1-19-9-10(7-17)5
    -13-12-3-2-4-14-16(12)11(8-18-14)6-15(13)19/
    h2-5,8,15,17-18H,6,9H2,1H3/t15-/m1/s1
|comp78| + comp79 -> |comp80| + comp81 + comp82
@yield R15-R16 70%

R16 "Rearrangement"
skeletonOut InChI=1S/C16H15N3/c1-19-9-10(7-17)5
    -13-12-3-2-4-14-16(12)11(8-18-14)6-15(13)19/
    h2-5,8,10,15,18H,6,9H2,1H3/t10?,15-/m1/s1
```

```
|comp83| -> |comp84|
@yield R15-R16 70%

R17 "Refunctionalization"
skeletonOut InChI=1S/C17H18N2O2/c1-19-9-11(17(20)21-2)6
    -13-12-4-3-5-14-16(12)10(8-18-14)7-15(13)19/
    h3-6,8,11,15,18H,7,9H2,1-2H3/t11?,15-/m1/s1
|comp85| + comp86 + comp87 + comp88 -> |comp89| + comp90
@yield 72%

R18 "Esterhydrolysis"
skeletonOut InChI=1S/C16H16N2O2/c1-18-8-10(16(19)20)5
    -12-11-3-2-4-13-15(11)9(7-17-13)6-14(12)18/
    h2-5,7,10,14,17H,6,8H2,1H3,(H,19,20)/t10-,14-/m1/s1
|comp91| + comp92 + comp93 -> |comp94| + comp95 + comp96
@yield 54%

R19 "Acetal formation"
skeletonOut InChI=1S/C5H9BrO2/c1-5(4-6)7-2-3-8-5/h2-4H2,1H3
|comp97| + comp98 -> |comp99| + comp100
@yield 60%

R20 "Refunctionalization"
skeletonOut InChI=1S/C6H13NO2/c1-6(5-7-2)8-3-4-9-6/
    h7H,3-5H2,1-2H3
|comp101| + 2 * comp102 -> |comp103| + comp104
@yield 80%
```

Listing A.1: A synthesis plan for LSD

Bibliography

[1] Mauro F. A. Adamo and Vivekananda R. Konda. Multicomponent synthesis of 3-indolepropionic acids. *Organic Letters*, 9(2):303–305, 2007. doi: 10.1021/ol0627698.

[2] Jakob Andersen, Christoph Flamm, Daniel Merkle, and Peter Stadler. Inferring chemical reaction patterns using rule composition in graph grammars. *Journal of Systems Chemistry*, 4(1):4, 2013. doi: 10.1186/1759-2208-4-4.

[3] John Andraos. Unification of reaction metrics for green chemistry: Applications to reaction analysis. *Organic Process Research and Development*, 9(2):149–163, 2005. doi: 10.1021/op049803n.

[4] John Andraos. On using tree analysis to quantify the material, input energy, and cost throughput efficiencies of simple and complex synthesis plans and networks: towards a blueprint for quantitative total synthesis and green chemistry. *Organic Process Research amp; Development*, 10(2):212–240, 2006. doi: 10.1021/op0501904.

[5] John Andraos. *The Algebra of Organic Synthesis: Green Metrics, Design Strategy, Route Selection, and Optimization*. CRC Press, November 2011.

[6] Steven H. Bertz. The bond graph. *J. Chem. Soc., Chem. Commun.*, pages 818–820, 1981. doi: 10.1039/C39810000818.

[7] Steven H. Bertz, Christoph Rücker, Gerta Rücker, and Toby J. Sommer. Simplification in synthesis. *European Journal of Organic Chemistry*, 2003 (24):4737–4740, 2003. doi: 10.1002/ejoc.200300558.

[8] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(10):1367–1372, Oct 2004. doi: 10.1109/TPAMI.2004.75.

[9] E. J. Corey, W. Jeffrey Howe, H. W. Orf, David A. Pensak, and George Petersson. General methods of synthetic analysis. strategic bond disconnections for bridged polycyclic structures. *Journal of the American Chemical Society*, 97(21):6116–6124, 1975. doi: 10.1021/ja00854a026.

[10] E. J. Corey, A. K. Long, and S. D. Rubenstein. Computer-assisted analysis in organic synthesis. *Science*, 228(4698):408–418, 1985.

[11] E.J. Corey and X.M Cheng. *The Logic of Chemical Synthesis*. Wiley, 1989.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[13] Sir William Crookes. *The Chemical News and Journal of Industrial Science*, volume 29–30. Chemical news office, 1874.

[14] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. doi: 10.1007/BF01386390.

[15] Rolf Fagerberg, Christoph Flamm, Rojin Kianian, Daniel Merkle, and Peter F. Stadler. Finding the k best synthesis plans. 2014. submitted.

[16] Christoph Flamm and Lukas Bartonek. Personal communication.

[17] Shinsaku Fujita. Description of organic reactions based on imaginary transition structures. 8. synthesis space attached by a charge space and three-dimensional imaginary transition structures with charges. *Journal of Chemical Information and Computer Sciences*, 27(3):111–115, 1987. doi: 10.1021/ci00055a004.

[18] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[19] H. L. Gelernter, A. F. Sanders, D. L. Larsen, K. K. Agarwal, R. H. Boivie, G. A. Spritzer, and J. E. Searleman. Empirical Explorations of SYNCHEM. *Science*, 197(4308):1041–1049, 1977.

[20] Steven H. Bertz and Toby J. Sommer. Rigorous mathematical approaches to strategic bonds and synthetic analysis based on conceptually simple new complexity indices. *Chem. Commun.*, pages 2409–2410, 1997. doi: 10.1039/A706192G.

[21] James B. Hendrickson. Systematic synthesis design. iv. numerical codification of construction reactions. *Journal of the American Chemical Society*, 97(20):5784–5800, 1975. doi: 10.1021/ja00853a023.

[22] James B. Hendrickson. Systematic synthesis design. 6. yield analysis and convergency. *Journal of the American Chemical Society*, 99(16):5439–5450, 1977. doi: 10.1021/ja00458a035.

[23] James B. Hendrickson. Approaching the logic of synthesis design. *Accounts of Chemical Research*, 19(9):274–281, 1986. doi: 10.1021/ar00129a003.

[24] James B. Hendrickson. Organic synthesis in the age of computers. *Angewandte Chemie International Edition in English*, 29(11):1286–1295, 1990. doi: 10.1002/anie.199012861.

[25] James B. Hendrickson, David L. Grier, and A. Glenn Toczko. A logic-based program for synthesis design. *Journal of the American Chemical Society*, 107(18):5228–5238, 1985. doi: 10.1021/ja00304a033.

[26] R.W. Hoffmann. *Elements of Synthesis Planning*. Springer, 2009.

[27] Wolf-Dietrich Ihlenfeldt and Johann Gasteiger. Computer-assisted planning of organic syntheses: The second generation of programs. *Angewandte Chemie International Edition in English*, 34(23-24):2613–2633, 1996. doi: 10.1002/anie.199526131.

[28] Vineet Kahlon, Chao Wang, and Aarti Gupta. Monotonic partial order reduction: An optimal symbolic partial order reduction technique. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, pages 398–413, Berlin, Heidelberg, 2009. Springer-Verlag. doi: 10.1007/978-3-642-02658-4_31.

[29] D. E. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. 1997.

[30] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[31] Mario Latendresse, Jeremiah P. Malerich, Mike Travers, and Peter D. Karp. Accurate atom-mapping computation for biochemical reactions. *Journal of Chemical Information and Modeling*, 52(11):2970–2982, 2012. doi: 10.1021/ci3002217.

[32] James Law, Zsolt Zsoldos, Aniko Simon, Darryl Reid, Yang Liu, Sing Yoong Khew, A. Peter Johnson, Sarah Major, Robert A. Wade, and Howard Y. Ando. Route designer: A retrosynthetic analysis tool utilizing automated retrosynthetic rule generation. *Journal of Chemical Information and Modeling*, 49(3):593–602, 2009. doi: 10.1021/ci800228y.

[33] X. Li and D. R. Liu. DNA-templated organic synthesis: nature's strategy for controlling chemical reactivity applied to synthetic molecules. *Angew. Chem. Int. Ed. Engl.*, 43(37):4848–4870, 2004.

[34] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, 60(0):94 – 112, 2014. doi: http://dx.doi.org/10.1016/j.jsc.2013.09.003.

[35] Mireya L. McKee, Phillip J. Milnes, Jonathan Bath, Eugen Stulz, Rachel K. OâReilly, and Andrew J. Turberfield. Programmable one-pot multistep organic synthesis using dna junctions. *Journal of the American Chemical Society*, 134(3):1446–1449, 2012. doi: 10.1021/ja2101196.

[36] István Moldvai, Eszter Temesvári-Major, Mária Incze, Éva Szentirmay, Eszter Gács-Baitz, and Csaba Szántay. Enantioefficient synthesis of -ergocryptine: first direct synthesis of (+)-lysergic acid. *The Journal of Organic Chemistry*, 69(18):5993–6000, 2004. doi: 10.1021/jo049209b.

[37] Lars Relund Nielsen, Kim Allan Andersen, and Daniele Pretolani. Finding the k shortest hyperpaths. *Computers Operations Research*, 32(6):1477 – 1497, 2005. doi: http://dx.doi.org/10.1016/j.cor.2003.11.014.

[38] Taylor Otwell. *Laravel: From Apprentice to Artisan*. LeanPub, 2013.

[39] Mark Peplow. The Robo-Chemist. *Nature*, 512:20–22, 2014.

[40] Klaus Renzel and Wolfgang Keller. Client/server architectures for business information systems - a pattern language, 1997.

[41] C. Rücker, G. Rücker, and S. H. Bertz. Organic synthesis–art or science? *J Chem Inf Comput Sci*, 44(2):378–386, 2004.

[42] Gerta Rücker and Christoph Ruecker. Counts of all walks as atomic and molecular descriptors. *Journal of Chemical Information and Computer Sciences*, 33(5):683–695, 1993. doi: 10.1021/ci00015a005.

[43] Matthew H. Todd. Computer-aided organic synthesis. *Chem. Soc. Rev.*, 34:247–266, 2005. doi: 10.1039/B104620A.

[44] Ivar Ugi, Johannes Bauer, Klemens Bley, Alf Dengler, Andreas Dietz, Eric Fontain, Bernhard Gruber, Rainer Herges, Michael Knauer, Klaus Reitsam, and Natalie Stein. Computer-assisted solution of chemical problems—the historical development and the present state of the art of a new discipline of chemistry. *Angewandte Chemie International Edition in English*, 32(2):201–227, 1993. doi: 10.1002/anie.199302011.

[45] Rebecca M. Wilson and Samuel J. Danishefsky. Pattern recognition in retrosynthetic analysis: snapshots in total synthesis. *The Journal of Organic Chemistry*, 72(12):4293–4305, 2007. doi: 10.1021/jo070871s.

[46] Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):pp. 712–716, 1971.