Two Optimal Path Problems in Synthetic Biology

Matthew Fong

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-126
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-126.html

May 15, 2015

Two Optimal Path Problems in Synthetic Biology

by Matthew Fong

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Sanjit A. Seshia

Research Advisor

(Date)

* * * * * * *

Professor J. Christopher Anderson

Second Reader

(Date)

Two Optimal Path Problems in Synthetic Biology

Matthew Fong

May 2015

Abstract

We examine two optimal path problems that arise in the context of ranking enzymatic pathways to synthesize a target compound. First, we present a survey of exact and approximation algorithms for the multiobjective shortest path (MOSP) problem. Second, we formalize the problem of finding stoichiometrically minimal source pathways (SMSP), which seeks to find paths that use a non-dominated amount of native chemicals to reach a target compound. We show that the SMSP problem is NP-complete and present two approaches to solve it based on model checking. For both problems, we provide an experimental evaluation and discussion of results.

Contents

Chapter 1

Introduction

1.1 Background

As increasing amounts of data are being collected in the field of synthetic biology, the need has arisen for graph-theoretic approaches to analyze and process them in a meaningful manner. Several problems in synthetic biology involve systems that have a natural graph structure, such as metabolic networks, regulatory networks, and protein-protein interaction (PPI) networks [11]. In this study, we aim to formalize and propose solutions to solve two such problems, as well as give the biological context that they are founded upon.

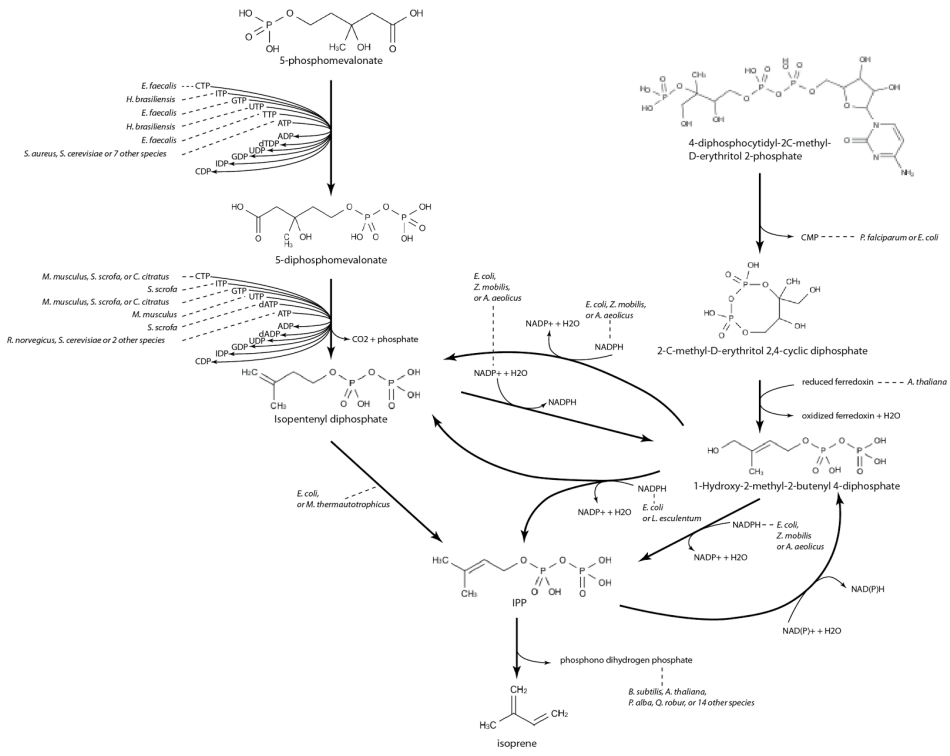

Figure 1.1: Metabolic cascade to the compound isoprene.

This project is based upon the work from Act Ontology [16], a formalism for describing the molecular function of any entity that participates in a biochemical reaction. One of its primary objectives is building an enzymatic pathway synthesizer, which can construct pathways based on both naturally-known and predicted reactions. To make this idea concrete, we present a cascade (showing multiple pathways) to isoprene in Fig 1.1 above. From this example and Fig 1.2, one can see the numerous possible pathways to a target.

To translate this biological entity into a theoretical problem, we will interpret each chemical as a vertex and each reaction as a directed hyperedge in our metabolic network. In this work, we will specifically look at the design and applications of graph algorithms to solve an open problem from the Act Ontology project. More specifically, our goal is to be able to answer the following query: *Given a metabolic graph and a target, what is the best pathway to that target?*. There are many possible criteria for ranking paths, such as O_2 consumption, free energy change, and stoichiometric yield. Without knowing the relative importance of these different metrics, this study investigates two optimal path algorithms aimed at solving this problem.

1.2 Problems Studied

First, the **Multiobjective Shortest Path** (MOSP) problem aims to find shortest paths in a graph, where the cost of each edge is a vector, instead of a scalar. The features of these values are those criteria that we discuss above, such as O_2 consumption and free energy change. The goal is to return all non-dominated paths, as there is no currently known order of importance for our features. As this is a NP-complete problem, we investigate the runtime and effectiveness of exact and approximation algorithms.

Second, the **Stoichiometrically Minimal Source Pathway** (SMSP) [6] problem attempts to find pathways in a metabolic graph that use the least amount of the native (starting) compounds. This can be used as another component of our ranking problem above, as stoichiometric yield is definitely an impactful factor in the efficiency of a metabolic pathway. We provide a formalization and an algorithm for solving this NP-complete problem.

1.3 Contributions

We conduct a survey of three algorithms for the MOSP problem: Martins' [10], Warburton's [18], and Tsaggouris' [17]. We also implement these algorithms and discuss their results on randomly-generated test graphs.

We propose a formalization for the SMSP problem, as well as two model-checking approaches for solving it. We perform an extensive experimental evaluation of our model-checking algorithms and hypothesize possible improvements in terms of their tractability using the NuSMV [4] and UCLID [2][3] model checkers.

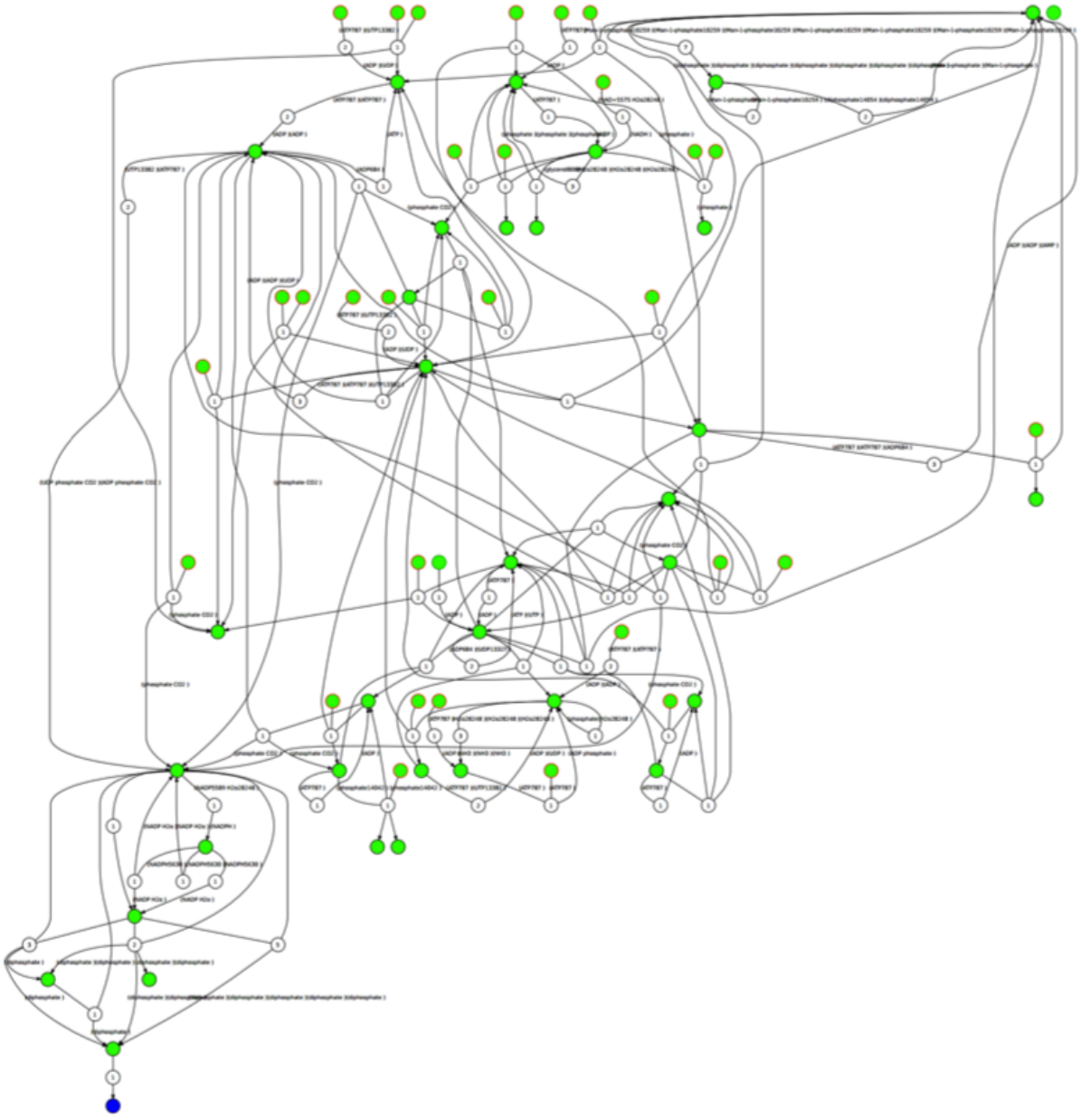

Figure 1.2: Sample output from the Act [16] pathway synthesizer to the compound tryptophol. Green nodes are chemicals, white nodes are reactions, and the blue node is the target. Nodes that are depicted as source nodes are the native compounds. This thesis is the first effort to systematically address the problem of ranking the possible paths to a target.

Chapter 2

Multiobjective Shortest Path

2.1 Introduction

The multiobjective shortest path (MOSP) problem is one that has been studied extensively, with numerous approximation algorithms devised since it was proved to be an NP-complete problem by Pierre Hansen in in 1980 [7]. While there is a linear time solution to the single-objective shortest path problem (Djikstra's algorithm), there exists an instance of the bi-objective shortest path problem that has an exponential number of possible unique solutions [7]. In other fields, the MOSP problem has been used to model situations such as a road network (time, distance), as well as in operations research. In this study, we attempt to optimize this problem for the use in the Act Ontology enzymatic pathway synthesizer.

As mentioned in Ch. 1, our goal is to rank pathways to a given target, in terms of how likely it is to be successful. There are many possible criteria on which to rank pathways, such as oxygen consumption, stoichiometric yield, and number of reactions. However, there is no widely accepted ordering of the relative importance of these criteria. Therefore, we will represent our edges as having multi-dimensional cost and find the set of all non-dominated paths, paths that are not strictly worse than another path to the target.

Let $x = (x_1, ..., x_r)$ and $y = (y_1, ..., y_r)$ represent real r-vectors, where r is the number of features.

Definition 2.1.1. *We define $x \leq y$, or x **dominates** y, if $x_k \leq y_k$ for $1 \leq k \leq r$ and $x_k < y_k$ for at least one such k.*

Definition 2.1.2. *We define the multiobjective shortest path problem as the following: Given a graph $G(V, E, c)$ with associated n vertices V, m edges E, and r-dimensional cost function c over all of the edges, the solution to MOSP is the set of all s-t paths that are not dominated by another path, where the cost of a path p is $c(p) = \sum_{e \in p} c(e)$.*

We also provide the linear programming formalization below for the 2-dimensional case (which can be easily

generalized to r dimensions):

$$\min \qquad c(x) = \begin{cases} c_1(x) = \sum_{(i,j)\in E} c_1(i,j)x_{ij} \\ c_2(x) = \sum_{(i,j)\in E} c_2(i,j)x_{ij} \end{cases} \tag{2.1}$$

$$s.t. \qquad \sum_{i:(i,j)\in E} x_{ij} - \sum_{i:(j,i)\in E} x_{ji} = \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

$$x_{ij} \in \{0,1\} \qquad \forall \qquad (i,j) \in E \tag{2.3}$$

Definition 2.1.3. *We define an* ϵ***-approximation*** *for MOSP equivalently, except the dominance definition is relaxed to* $x_k \leq (1+\epsilon)y_k$.

2.2 Martins' exact algorithm

The first algorithm investigated was an exact exponential algorithm, designed by Ernesto Martins in 1982 [10]. It follows a multiple labeling approach, where a label consists of the feature vector for a given edge, as well as a pointer to the previous label from which it was extended from.

Precisely, a label is a tuple with the following fields: (*costs, prevLabels, reactionID, chemicalID*). Here, *costs* is a r-dimensional vector, *prevLabels* is a set of labels that were used on the path to this current label, *reactionID* represents the last reaction to this current label, and *chemicalID* represents the chemical for this label. Once a label is set as permanent, it will never be deleted, while temporary labels may be removed during the dominance test on line 13. This algorithm iterates through the entire set of temporary labels until there are none left.

Algorithm 1 Martins' algorithm

```
1:  procedure MOSP(G, starting, r)
2:      for all chemical ∈ starting do
3:          add [(0, 0, ..., 0), null, null, chemical] to tempLabels
4:      while tempLabels is not empty do
5:          currentLabel ← tempLabels.poll()
6:          permanentLabels.add(currentLabel)
7:          for all reactions from currentLabel.node with fully visited reactants do
8:              for all products from reaction do
9:                  for all combinations ∈ cartesianProduct( reactant costs ) do
10:                     newCosts ← sum(combination) + reaction.costs
11:                     tempLabels.add([newCosts, currentLabel.prevLabels + currentLabel, reaction, product])
12:                     for all labels in tempLabels that create product do
13:                         if newLabel dominates label then
14:                             delete label
15:                         if label dominates newLabel then
16:                             delete newLabel
```

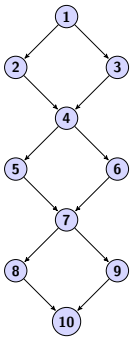

Figure 2.1: N-diamond graph, where N = 3. Edges are composed of randomly generated integers in the range of 0 to 1000.

In this algorithm, *permanentLabels* is a list of labels that are marked as permanent, and *tempLabels* is a priority queue of temporary labels that is ordered by the lexicographical ordering of the distances (eg. from smallest cost vector to largest cost vector, starting with the first value and then looking at the second, third, etc to break ties). By examining each edge in this order, we ensure that once a label is marked as permanent, it is not possible for it to be dominated by another vector. In Alg. 1, we have made the modifications to Martins' algorithm that are listed in the following section.

The N-diamond graph formulation is shown in Fig 2.1, which our early benchmarking studies were performed on. Fig 2.2 shows the performance of the exact algorithm on that graph, while plotting edges vs. time. The feature values are generated randomly for the N-diamond graph in Fig 2.1. From these results, we can see the worst-case exponential behavior in terms of the input size.

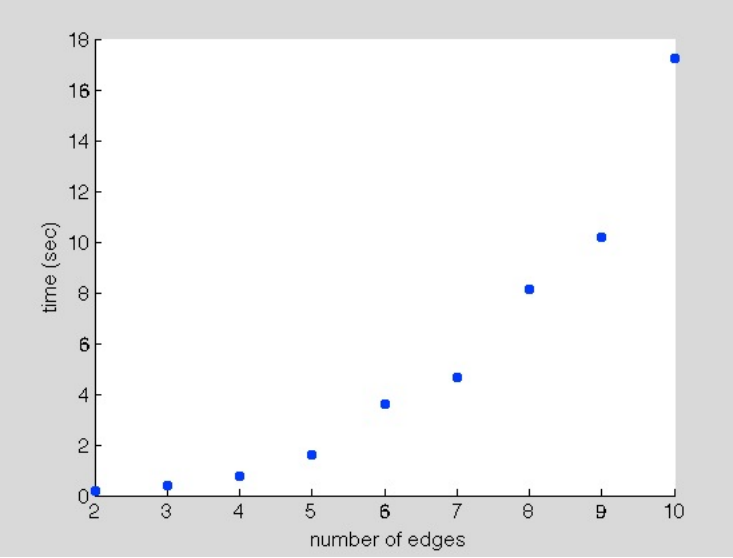

Figure 2.2: Edges vs time for Martins' algorithm with N-diamond graph, with 3 features. Edge vectors are randomly generated integers in the range of 0 to 1000.

2.2.1 Modifications for metabolic graphs

To ensure this algorithm's correctness in our Act system, several modifications needed to be made to the published algorithm. A proof to the original algorithm can be found in the Martins paper [10], and the modifications for Act are justified below.

1. Multiple sources
   Martins algorithm only addresses the single source case, but just initializing a zero-cost vector for each native chemical keeps the key invariants true.
2. Hypergraph vs simple graph
   We needed to account for the fact that most of the reactions required multiple reactants and would yield multiple products. Our edges, reactions, connect a set of reactants to a set of products. To accommodate for this, we had to augment our algorithm in the following two ways:
   (a) Reaction Filtering
   For line 7 of Algorithm 1, a filtering function was used to ensure that all of the other reactants for that reaction had already been reached. A set of reached chemicals is kept, and all of the auxiliary reactants of a reaction must be in that set for that reaction to be considered.
   (b) Cartesian Product
   Because there are multiple reactants that can contribute to a reaction, all possible combinations of those labels must be kept in mind when finding the cost of the label. Therefore, a Cartesian product of the sets of all current cost vectors for each reactant is taken and added to the reaction cost.

2.2.2 Warburton's approximation: scaling and rounding algorithm

Warburton's algorithm [18] is an example of a scaling and rounding approximation [14], in which each edge weight is scaled to an appropriate integer and an exact algorithm is used to solve the problem. It follows an iterated approach, where the final approximated solution is the union of all of the paths that are returned by each of the smaller subproblems. The last index for the arrays seen in the algorithm are indexed up to $r-1$ because no rounding needs to be done on the final feature value.

Partial Solution: Approximation of C_L

Given a positive integer vector $L = (L_1, ..., L_{r-1})$ and a positive vector $\theta = (\theta_1, ..., \theta_{r-1})$, with $\theta_k > 1$, we will find C_L, the set of all paths satisfying

$$L_k \leq c_k(P) \leq \theta_k L_k, \qquad 1 \leq k \leq r-1 \tag{2.4}$$

One iteration of this algorithm will determine an ϵ-efficient set for C_L. This set is found by scaling the arc lengths to appropriate values, as seen below, where $d_k(e)$ is the new value for the kth feature of arc e and $c_k(e)$ is the old cost.

$$d_r(e) = c_r(e) \tag{2.5}$$

$$d_k(e) = \left\lfloor \frac{c_k(e)}{T_k} \right\rfloor, \qquad 1 \leq k \leq r-1 \tag{2.6}$$

$$T_k = \frac{\epsilon L_k}{n-1}, \qquad 1 \leq k \leq r-1 \tag{2.7}$$

Theorem 2.2.1. *Using the new arc lengths $d(e)$ above with an exact algorithm will find an ϵ-approximation for C_L.*

Note: below, we will use x^* to denote the set of all Pareto optimal members in x, and $\mathcal{P}$ to denote the set of all s-t paths. When the cost operator $c()$ or $d()$ acts upon a set of paths, it returns a set of the corresponding costs.

Proof From equation (2.6), we get, for any $p \in \mathcal{P}$ and arc e:

$$\frac{c_k(e)}{T_k} - 1 \quad \leq d_k(e) \quad \leq \frac{c_k(e)}{T_k}, \qquad 1 \leq k \leq r-1 \tag{2.8}$$

$$c_k(p) - (n-1)T_k \quad \leq T_k d_k(p) \quad \leq c_k(p), \qquad 1 \leq k \leq r-1 \tag{2.9}$$

Consider the set $p \in Q_L$ of s-t paths for which $d_k(p) \leq (n-1)\frac{\theta_k}{\epsilon}$. Construct $R_L \subset Q_L$ such that for each $u \in (d(Q_L))^*$, there is exactly one $p \in R_L$ satisfying $d(p) = u$.

Claim: R_L is ϵ-efficient for C_L.

Reasoning: $(d(R_L))^* = d(R_L) = (d(Q_L))^*$, from the construction above. Therefore, $c(R_L) = (c(R_L))^*$. Now, we just need to show that for each $v \in C_L$, there is a $u \in c(R_L)$ that satisfies $u_k \leq (1+\epsilon)v_k$.

First, let V be a path that satisfies equation (2.4), and U be a path that satisfies equation (2.9). We can easily see that $d_k(U) \leq d_k(V)$. We also can assume $c_r(U) \leq c_r(V)$. Taking the equations above, we have

$$c_k(U) \leq c_k(V) + (n-1)T_k \tag{2.10}$$

$$= c_k(V) + \epsilon L_k, \tag{2.11}$$

$$\leq (1+\epsilon)c_k(V). \tag{2.12}$$

$u = c(U)$ and $v = c(V)$ shows what we need, concluding the proof. □

We will complete the algorithm by detailing the iterative step below (lines 2-5 of the algorithm). The decreased runtime comes from scaling the edge weights to smaller integers (and limiting the number of total number of partial paths that need to be kept as the algorithm progresses). The extra parameter B in the algorithm is set to be large enough to ensure that $1 \leq c_k(p) \leq B$. **MOSP_exact**(G, r, c) is any algorithm that solves the MOSP problem exactly.

Algorithm 2 Warburton's algorithm

1: **procedure** MOSP(G(V,E), r, c, B)
2: $m \leftarrow \lceil \log B \rceil$
3: $I \leftarrow$ **cartesianProduct**$([0, 1, \ldots, m-1])$
4: **for all** $(m_1, m_2, \ldots, m_{r-1}) \in I$ **do**
5: $L \leftarrow (2^{m_1}, 2^{m_2}, \ldots, 2^{m_{r-1}})$
6: $T_k \leftarrow \dfrac{\epsilon L_k}{n-1}, \qquad 1 \leq k \leq r-1$
7: $d_k(e) \leftarrow \left\lfloor \dfrac{c_k(e)}{T_k} \right\rfloor, \qquad 1 \leq k \leq r-1$
8: $d_r(e) \leftarrow c_r(e)$
9: $sol \leftarrow$ **MOSP_exact**(G, r, d)
10: $allSolutions \leftarrow allSolutions \cup sol$

Complete solution: Iterative Step

The previous approximation now must be directly related to an ϵ-approximation for $\mathcal{P}^*$, the Pareto-optimal set of paths with the original edge lengths. One appropriate choice of L and θ is as follows. First, compute B such that B is larger than the largest cost value of any possible path through the graph.

Then, let $m = \lceil \log B \rceil$ and let I denote the $(r-1)$-fold Cartesian product of $[0, ..., m-1]$. For each $(m_1, ..., m_{r-1}) \in I$, $L = (2^{m_1}, ..., 2^{m_{r-1}})$ and $\theta = (2, ..., 2)$.

Theorem 2.2.2. *The union of all C_L that are found with the exact algorithm for each L compose an $\epsilon-$approximation for $\mathcal{P}^*$.*

Proof From above, we find all efficient sets of paths that meet this criteria:

$$L_k \leq c_k(p) \leq \theta_k L_k, \qquad 1 \leq k \leq r-1 \tag{2.13}$$

Given our definitions of L and θ, it is easy to see that the smallest value of L_k is 0, and the largest value of $L_k\theta_k$ is B, which is the largest cost value. Because we are taking a Cartesian product, every possible combination of L_k and $L_k\theta_k$ will be taken, ensuring that all paths are considered. Therefore, the union of all efficient sets of C_L will include every efficient path with the original path lengths. We can simply do a dominance check after the union to find the ϵ-approximate Pareto optimal solutions. □

Theorem 2.2.3. *The runtime of this algorithm is $O(rn^3(\frac{n^2\lceil logB \rceil}{\epsilon})^{r-1})$.*

Proof Details can be found in Warburton's paper [18]. □

2.2.3 Tsaggouris/Zaroliagis approximation: interval partitioning algorithm

The algorithm designed by Tsaggouris and Zaroliagis [17] is an example of an interval partitioning approximation [14], in which there is only one path kept per defined interval of features, so that the number of paths retained is not bounded exponentially by the problem size. The $(r-1)$-dimensional arrays Π_v^i are used to store these paths, and the function $pos(p)$ below is used to index appropriately into the arrays.

$$c_i^{min} = \min_{e \in E} c_i(e) \qquad c_i^{max} = \max_{e \in E} c_i(e) \tag{2.14}$$

$$C_i = \frac{c_i^{max}}{c_i^{min}} \tag{2.15}$$

Definition 2.2.1. *$P^i(s, v)$ denotes the set of all $s-v$ paths in G with no more than i edges.*

The algorithm proceeds in rounds. For each round i and for each node v the algorithm computes a set of labels Π_v^i, which is an ϵ-approximation of all possible $P^i(s, v)$ paths. We implement these sets of labels by using $(d-1)$-dimensional arrays Π_v^i, where the bounds for each dimension are

$$\lfloor \log_{1+\epsilon}(nC_i) \rfloor \qquad 1 \leq i \leq r-1 \tag{2.16}$$

and index these arrays using $(d-1)$-vectors. This is done with a $pos(p)$ function, which gives us the position in Π_v^i corresponding to p (essentially, which interval it is in):

$$pos(p) = [\log_{1+\epsilon} \frac{c_1(p)}{c_1^{min}}, \log_{1+\epsilon} \frac{c_2(p)}{c_2^{min}}, ..., \log_{1+\epsilon} \frac{c_{r-1}(p)}{c_{r-1}^{min}}] \tag{2.17}$$

From this $pos(p)$ function, we can see why the bounds in Equation 2.16 apply – C_i is the biggest possible cost ratio. The resulting paths stored in Π^{n-1}_{target} are the final possibilities for the approximated set (because the longest path will be at most length $n-1$). After removing any dominated sets from that matrix, we are left with the solution to this problem. The runtime improvement from this algorithm comes from the discretization of the state space (from continuous to discrete intervals), meaning that we do not need to store an exponential number of paths.

Theorem 2.2.4. *For all $v \in V$ and for all $i \geq 0$, after the i-th round, Π^i_v is an ϵ-approximation for $P^i(s,v)$.*

Proof We can prove this theorem by showing that for all $p \in P^i(s,v)$, there exists $q \in \Pi^i_v$ such that $c_k(q) \leq (1+\epsilon)c_k(p)$ for $1 \leq k \leq r$. We will use a proof by induction.

For the base case, consider a path with only one edge. After the first round, the position $pos(p)$ of Π^1_v will contain a path q for which $pos(q) = pos(p)$ and $c_r(q) \leq c_r(p)$. That first condition implies that $log_{1+\epsilon}\frac{c_k(q)}{c^{min}_k} - 1 \leq log_{1+\epsilon}\frac{c_k(p)}{c^{min}_k}$ for $1 \leq k \leq r$. Therefore, this implies that we have $c_k(q) \leq (1+\epsilon)c_k(p)$ for $1 \leq k \leq r$.

Now, we want to look at the inductive step. Consider a path $p = (e_1, e_2, \ldots, e_k) \in P^i(s,v)$, with $k \leq i$. If we look at subpath p' with the first $k-1$ edges of p, we know there exists a path $q' \in \Pi^{i-1}_u$ such that $c_k(q') \leq (1+\epsilon)c_k(p')$ for $1 \leq k \leq r$.

Now, let q^* be the addition of edge e_k to path q'. We have $c_k(q^*) \leq (1+\epsilon)c_k(p)$ for $1 \leq k \leq r$. Then, similar to the base case, position $pos(q^*)$ of Π^i_v will contain a path q that has $pos(q) = pos(q^*)$ and $c_r(q) \leq c_r(q^*)$. This implies that $log_{1+\epsilon}c_k(q) - 1 \leq log_{1+\epsilon}c_k(q^*)$ for all $1 \leq k \leq r$, which brings us to the statement that we are trying to prove. □

Theorem 2.2.5. *The runtime of this algorithm is $O(nm\prod^{r-1}_{j=1}(\lfloor log_{1+\epsilon}(nC_j)\rfloor + 1))$.*

Proof This algorithm terminates after at most $n-1$ rounds, because there can be at most $n-1$ edges in a path. In each round, all m edges are examined and an ExtendMerge operation is performed. The ExtendMerge operation runs in time proportional to the size of the arrays, which is $\prod^{d-1}_{j=1}(\lfloor log_{1+\epsilon}(nC_j)\rfloor + 1)$. Combining these observations leads to our desired runtime. □

Algorithm 3 Tsaggouris' algorithm

```
1:  procedure MOSP(G(V,E), r, c)
2:      for all v ∈ V do
3:          {Π^0_v = ∅}
4:      Π^0_s[0] = {(0, null, null)}
5:      for i = 1 to n − 1 do
6:          for all v ∈ V do
7:              Π^i_v = Π^{i−1}_v
8:              for all e = (u, v) ∈ E do
9:                  Π^i_v = ExtendMerge(Π^i_v, Π^{i−1}_u, e)
10: procedure ExtendMerge(R, Q, e)
11:     for all p ∈ Q do
12:         q = (c(p) + c(e), p, e)
13:         pos(q) = [⌊log_{1+ε} c_1(q)/c^min_1⌋, ⌊log_{1+ε} c_2(q)/c^min_2⌋, ..., ⌊log_{1+ε} c_{r−1}(q)/c^min_{r−1}⌋]
14:         if R[pos(q)] = null or c_d(R[pos(q)]) > c_d(q) then
15:             R[pos(q)] = q
```

2.3 Experimental Evaluation

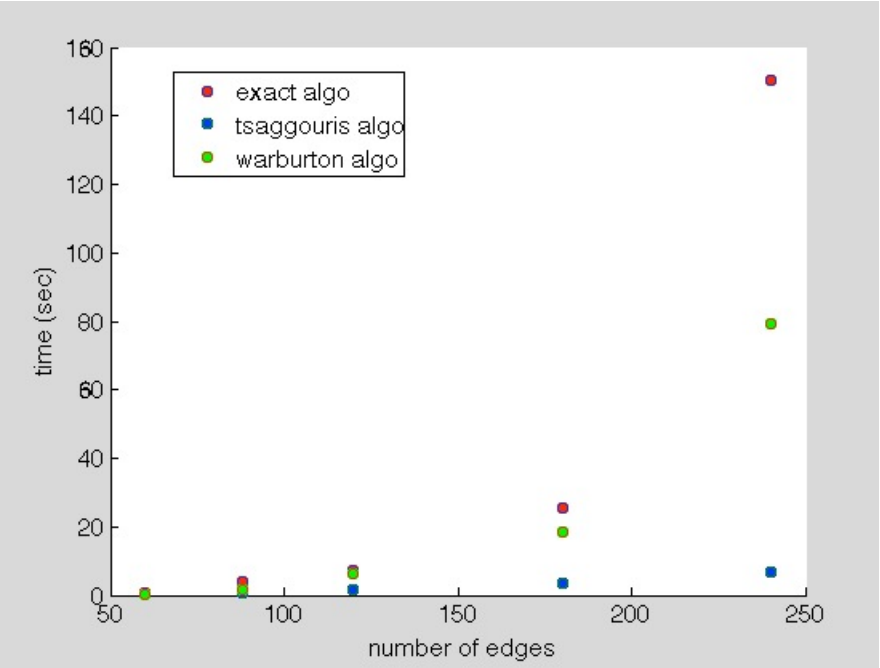

Figure 2.3: Edges vs time for MOSP algorithms, with 3 features and ϵ=0.1.

For our evaluation of these algorithms, we again generate an N-diamond graph with randomly chosen feature values in the range of 0 to 1000. We looked specifically at $\epsilon = 0.1$, with three features in the cost vector.

Fig 2.3 shows the results of all three algorithms on the graph described above. Although Warburton's algorithm does not perform significantly better than the exact algorithm, Tsaggouris' algorithm definitely shows major improvement. The one downside with the Tsaggouris algorithm is that its implementation is memory intensive - it runs out of memory with 5 features and 180 edges. Both approximations, without additional modifications, are intractable for these parameters with more than 5 features, as both of their runtimes are exponential in the number of features.

Because the actual metabolic graphs will not necessarily have the same exponential structure of the N-diamond graph, the approximation results are a promising sign that we can still make a good tractable approximation to the problem. However, optimization would be needed to fit in the Tsaggouris algorithm with its large memory footprint. While the proper weightings of the feature values are not known, the best approach would probably be a combination of heuristic and shortest path search methods.

Chapter 3

Stoichiometrically Minimal Source Pathways

3.1 Introduction

We formulate the problem of finding stoichiometrically minimal source pathways (SMSPs) in biochemical metabolic graphs and present a model checking approach to solve it. SMSPs are paths that, when flattened up to the source nodes corresponding to native compounds, use a non-dominated amount of those compounds. Our approach allows one to eliminate inefficient pathways when selecting the best path to a target. We also investigate the impact of the choice of model checking technique on the runtime for our procedure.

Definition 3.1.1. *We define a* ***metabolic graph*** *as $G = (V, E, U, t)$, where V is the set of all chemicals, $U \subseteq V$ is the set of native chemicals (also called starting or source chemicals), t is the target, and E is the matrix expressing the coefficients of the reactions, where E_{ji} is the coefficient for chemical i in reaction j.*

Let $|V| = n$ and $|U| = k$. The cost vector $c = [c_1, \ldots, c_k, \ldots, c_n]$ represents the amount of each chemical that is used (positive) or produced (negative) in any path. For each hyperedge j that is traversed, increment c_i by E_{ji}.

Definition 3.1.2. *We define a* ***final cost vector*** *as the vector $c = [c_1, \ldots, c_k, 0, \ldots, 0]$, which must have a zero entry for all indices that refer to non-native chemicals.*

Definition 3.1.3. *Path x* ***dominates*** *path y for cost vector c if for all $i \in U$, $c_i^{(x)} \leq c_i^{(y)}$ and $c_i^{(x)} < c_i^{(y)}$ for at least one such i. $c_i^{(x)}$ and $c_i^{(y)}$ are the costs for the ith chemical in paths x and y respectively.*

Definition 3.1.4. *We define the* ***SMSP problem*** *as follows: Given a metabolic graph, find all U-t paths that are non-dominated for their final cost vectors.*

As this graph is completely connected, there is some ordering of reactions that can force the cost vector to be zero for the non-native chemicals. We define the problem in terms of the final cost vector so that we can compare the costs of different pathways relative to the native chemicals.

Finding SMSPs can easily be shown to be NP-complete with a reduction from the set partition problem, as seen in the next section, so we take a model checking approach to solve this SMSP problem. We will also outline the results from an *E. coli* system, provided by the Act Ontology pathway synthesizer [16].

3.2 Combinatorial Complexity

Definition 3.2.1. *The decision version of the SMSP problem is as follows: Given a metabolic graph $G(V, E, U, t)$, is there a reaction pathway that utilizes less than $x_1, x_2, \ldots, x_k$ units of the corresponding chemical in U to reach the target t?*

Theorem 3.2.1. *Decision-SMSP is NP-complete.*

Proof We will prove that the SMSP problem is NP-hard by showing a reduction from the bicriterion shortest path (BSP) problem, and then showing a reduction from the partition problem to the bicriterion shortest path problem.

In the BSP problem, we are given a graph G(V,E) with positive integer weights and lengths of each edge. We then try to solve the problem, "Does there exist an $s - t$ path with weight $\leq W$ and length $\leq L$?"

There is a straightforward encoding of the BSP problem as an SMSP problem. Each of the two criteria are represented by one starting chemical (for instance, X and Y). Each edge in the BSP problem can be encoded as one extra reaction in the SMSP system, by treating every non-s,t node in my BSP graph as an intermediate chemical in my SMSP system. For example, if we have two nodes A and C connected by an edge with weight 5 and length 2, we can write the equation $5\,\mathrm{X} + 2\,\mathrm{Y} + \mathrm{A} \longrightarrow \mathrm{C}$. Then, the solution to the BSP problem is any path that uses less than W units of X and L units of Y. Therefore, there is a reduction from the BSP problem to the SMSP problem.

For the second half of this proof, we will use a reduction from the partition problem, which asks, "Given a mulitset S of positive integers with total sum s, can it be partitioned into two subsets S_1 and S_2 so that the sum of numbers in S_1 equals the sum of numbers in S_2?" We can encode this into a BSP instance. If there are n integers in the original multiset, construct a linear graph with $n + 1$ nodes. For each integer x_i in S, connect nodes i and $i + 1$ with two edges: one with weight x_i and length 0, and one with weight 0 and length x_i. Then, finding a path from node 1 to node $n + 1$ with weight $\leq s/2$ and length $\leq s/2$ will be a solution to the partition problem.

We can show that the SMSP decision problem is in NP because you can verify in polynomial time whether the returned path is an s-t path, as well as whether each value is smaller than the $x_1, x_2, \ldots x_n$ limits that were set in the original problem.

Thus, we prove that the SMSP decision problem is NP-complete with a reduction from the partition problem and the BSP problem. □

3.3 Model checking

3.3.1 Background

Model checking is a method for formally verifying whether a system satisfies a specification expressed in temporal logic [5]. It is effective in systems that have extremely large state spaces, and symbolic model checking can be used to explore many states efficiently. In this work, we utilize bounded model checking, which is a variant that only extends the state space search to a limited number of steps. We also express our specifications as linear temporal logic (LTL) [13], which has modalities referring to time. For example, there are operators such as G (for always, or globally) and F (for eventually, or in the future) which are used in conjunction with a prepositional clause to verify whether a certain property holds true over an execution trace.

3.3.2 Formal Model

For model checking, the state of the system is defined by $[N_1, \ldots, N_n, rxn]$, which is a vector $S \in$ [$\mathbb{N}$ x ... x $\mathbb{N}$ x RXN_T] where RXN_T is an enum type whose elements are all possible reactions. Informally, N_i represents the net number of units produced of compound i, and rxn represents the reaction in the metabolic graph that was previously traversed ("fired"). The transition system is defined as (V, δ, I), where V is the set of state variables, δ is the transition relation, and I is the set of initial states.

$I = [0, \ldots, 0]$, where the last 0 denotes that no previous reaction was traversed. The edges of the graph define the transition relation δ, which increments N_i when compound i is produced and decrements N_i when compound i is consumed in the reaction rxn. The target t defines the entry of S which must become positive within a finite number of steps (eg. it is reached).

There are limits on each of the N_i variables, where M is set to be the maximum number of units of a compound (and should be large enough so that it is never reached). On each step of the transition system, rxn is non-deterministically assigned the value of any reaction.

No precautions need to be taken for possible cycles in the reaction pathway that the model checker returns. All that matter is the net consumption and not whether the pathway is actually viable independent of other reactions. For example, the pathway below (with A, B being native compounds, and G being the target) is valid:

1. $\mathrm{A} + \mathrm{D} \longrightarrow \mathrm{C}$
2. $\mathrm{B} + \mathrm{C} \longrightarrow \mathrm{D} + \mathrm{G}$

This is equivalent to a pathway that also reduces to $\mathrm{A} + \mathrm{B} \longrightarrow \mathrm{G}$, even though there is no ordering of those reactions that could produce G independently.

3.3.3 Transitions

Chemical coefficients (N_1, ..., N_n) will increase when a reaction is fired that produces the chemical, and will decrease when a reaction uses the chemical. Below, we define E, U, and t for a simple example (refer to Section 3.1 for the formal problem statement and definitions of these variables):

$$2\mathrm{A} + \mathrm{B} \rightarrow \mathrm{C}\ (\mathrm{rxn1}) \qquad \mathrm{B} + 2\mathrm{C} \rightarrow \mathrm{D}\ (\mathrm{rxn2})$$

$$U = \{\mathrm{A}, \mathrm{B}\}, \qquad t = \{\mathrm{D}\}, \qquad E = \begin{bmatrix} 2 & 1 & -1 & 0 \\ 0 & 1 & 2 & -1 \end{bmatrix}$$

The number of units of A, B, C, D are denoted by N_1, N_2, N_3, N_4, respectively. We encode this transition relation in the input language of NuSMV, a symbolic model checker [4].

This is the corresponding transition relation for N_3 (we flip the signs of the coefficients in the E matrix so we can have a positive coefficient when we produce the product). The first three clauses stop our search once N_3 goes beyond its bounds, or if the target is already produced. The next two clauses represent how N_3 changes in reaction 1 and 2. Otherwise, N_3 stays the same.

```
next(N3):=
case
    N3 > M : N3;        //stop if at upper bound
    N3 < -M : N3;       //stop if at lower bound
    N4 > 0 : N3;        //stop if target is produced
    rxn = 1 : N3 + 1;   //transition for rxn1
    rxn = 2 : N3 - 2;   //transition for rxn2
    TRUE : N3           //stays the same, otherwise
esac;
```

3.3.4 Termination

An appropriate linear temporal logic (LTL) [13] specification must then be checked to generate a counterexample that will provide one possible path. We iteratively add a constraint to the LTL formula to eliminate this generated counterexample and generate a new counterexample that is not dominated by this original one (or it will return "no counterexample found").

1. Positive coefficient for target
2. Non-negative coefficients for all non-starting chemicals
3. Better than any previously generated counterexample

Here is an example on another small graph, with target N51. This first LTL specification has characteristics (1) and (2), and is the initial specification that is evaluated.

```
G (N51 = 0 | (N51 = 0 | (N7 < 0) | (N20 < 0) | (N28 < 0)))
```

The following counterexample is found: N3 = -1, N5 = -1, N51 = 1, implying that one unit each of Chemicals 3 and 5 can be consumed to produce one unit of 51. The next specification is then:

```
G (N51 = 0 | (N51 = 0 | (N7 < 0) | (N20 < 0) | (N28 < 0)) | (N3 <= -1 & N5 <= -1))
```

This continues until no further counterexamples are found.

3.4 Results

First, we will examine a concrete solution to this problem. We are using an Act-generated metabolic graph for *E. coli*, which has the characteristics below. A pair of possible targets at different depths will be analyzed in the following section, one that is at depth 4 and one that is at depth 7.

Total reactions	1253
Balanced	1154
Total chemicals	762
Starting compounds	131
Reachable chemicals	175
Reachable reactions	196
Max depth	10

Table 3.1: Metrics for metacyc_ecol316407c network

3.4.1 Sample output

In Figure 3.1, we see an example of one non-dominated pathway to d-allose-6-phosphate. Although this example only has compounds with coefficient 1, our system accounts for non-unitary coefficients as well. In this case, the cost of this pathway would be the total of the compounds in red on the reactants side. All other reactants in this pathway come from the products of some other reaction in the pathway.

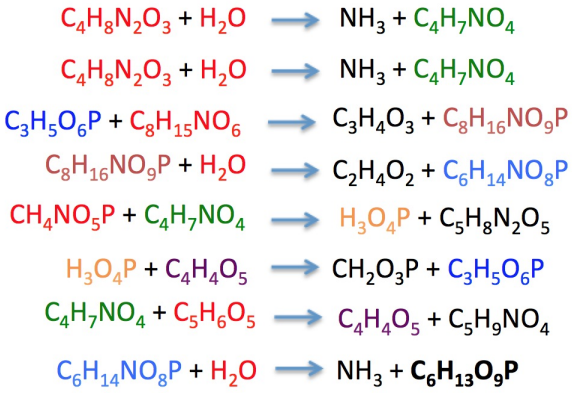

Figure 3.1: Sample pathway to $C_6H_{13}O_9P$. Red compounds are native.

3.4.2 Sample runtime

For our first implementation of this model, we used NuSMV, a symbolic model checker [4]. In Figure 3.2, we see how the runtime of this SMSP algorithm scales with the maximum reaction depth. Unfortunately, it appears to grow exponentially, and runs for nearly 8 hours when the maximum depth is set to 12. In addition, we will see in Table 3.2 that there are still paths that are found at those depths. From these preliminary results, we can definitely see a need for improving the runtime of this algorithm.

To frame this idea of increasing the maximum depth more concretely, we will look at the some of the costs found at different bounds.

Maximum Depth	Non-dominated cost
4	1 $C_5H_6O_5$, 2 H_2O, 1 CH_4NO_5P, 2 $C_4H_8N_2O_3$
5	1 $C_5H_6O_5$, 1 H_2O, 1 CH_4NO_5P, 2 $C_4H_8N_2O_3$
10	1 $C_4H_6O_4$, 1 CH_4NO_5P, 2 $C_4H_8N_2O_3$, 2 $C_5H_6O_5$
11	1 $C_4H_9NO_3$, 1 $C_5H_6O_5$, 1 H_2S, 1 CH_4NO_5P, 2 $C_4H_8N_2O_3$

Table 3.2: Various source node costs for SMSP with different maximum depths, for production of $C_3H_5O_6P$.

3.5 Optimization

3.5.1 Heuristic simplifications

From looking at the previous example, we see that the algorithm returns still-improving pathways as the bound is increased past length 10, but the runtime of larger bounds causes the algorithm to be infeasible if

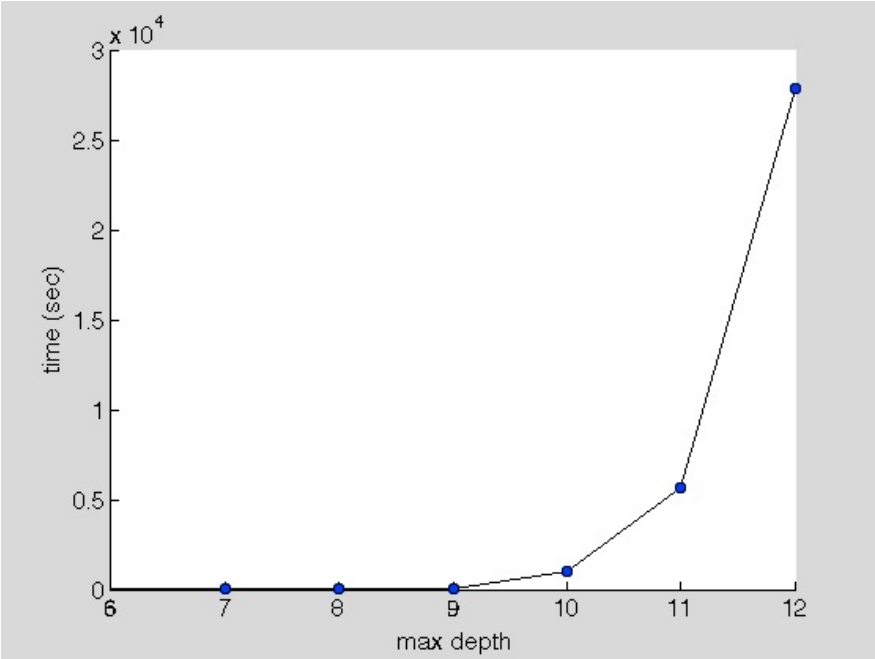

Figure 3.2: Runtime vs. max depth for finding paths for $C_3H_5O_6P$ with NuSMV using the simple model.

running on many targets. Our first attempt to reduce the runtime was with heuristic simplifications, which all served to decrease the state space of the model. We describe two of them below.

1. Do not allow regeneration of native compounds

Although this problem aims to find the paths that use up the least amounts of the native compounds, there are certain reactions that regenerate chemicals in this starting set. By allowing native chemicals to be the product of a reaction, we are artificially increasing the search space of this problem by returning extra paths that are not biochemically interesting. For example, if one possible path requires 2 A, 2 B, and 2 C (and there is a reaction A + B $\rightarrow$ C), then another originally returned path is 3 A, 3 B and 1 C (as well as 4 A, 4 B, and 0 C). These additional paths are not as interesting because they contain additional reactions that solely manipulate the counts of the native chemicals. One possible downside however, is a reaction that produces an intermediate as well as a native compound as a side product - those reactions do not get the added benefit of producing the native compound in this scheme. We can balance these two concerns by preventing reactions that *only* generate native compounds.

2. Limit constant M to be the maximum coefficient in the metabolic graph

There is no need for any chemical to be allowed to have a value greater than the maximum coefficient in the metabolic graph. If a particular reaction needs to be fired multiple times, then the counterexample can order itself in a way such that the reactions can occur without needing to maintain a value larger than the maximum coefficient. However, this change must also be combined with more careful logic in the transition statements to prohibit a reaction from taking place if one if its reactants or products is already at the minimum or maximum value that it can attain.

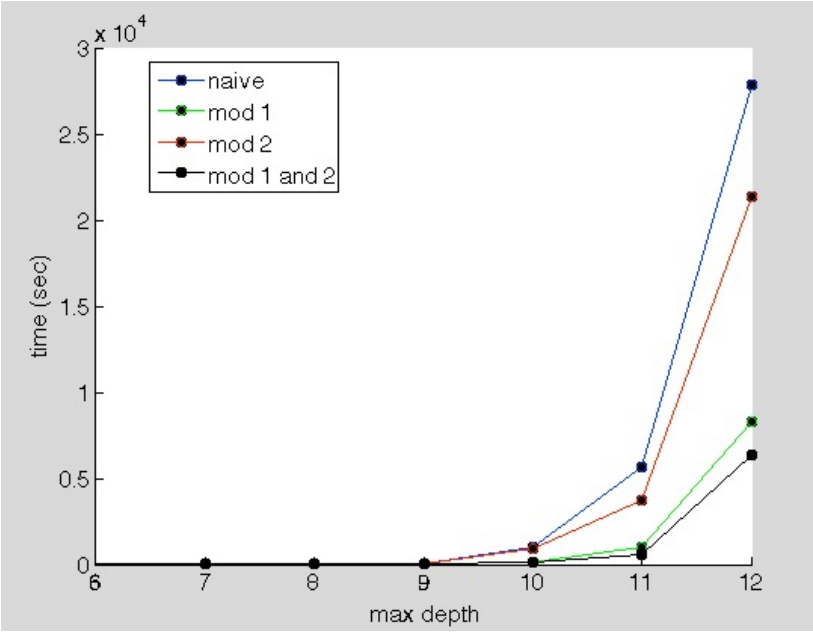

Figure 3.3: Runtime vs. max depth for finding paths for $C_3H_5O_6P$ with NuSMV and the modifications above. Notice that the runtimes are still in the range of 10^4, even after the heuristic modifications.

3.5.2 Software comparison

As mentioned earlier, we first encode our model with NuSMV. There are two important points to note in this analysis. First, we must impose a maximum search depth for our model checker. In practice, the most interesting (and best) paths occur within a relatively low maximum depth, slightly greater than the depth of the target to account for pathways that are not completely in series. The main tradeoff that we work with is the exponential nature of runtime vs. search depth, as we can see in Figures 3.2 and 3.3. In addition, we report the amount of time that it takes for a given model to reach a result of "no counterexample found", as that signifies the end of the search of the entire state space.

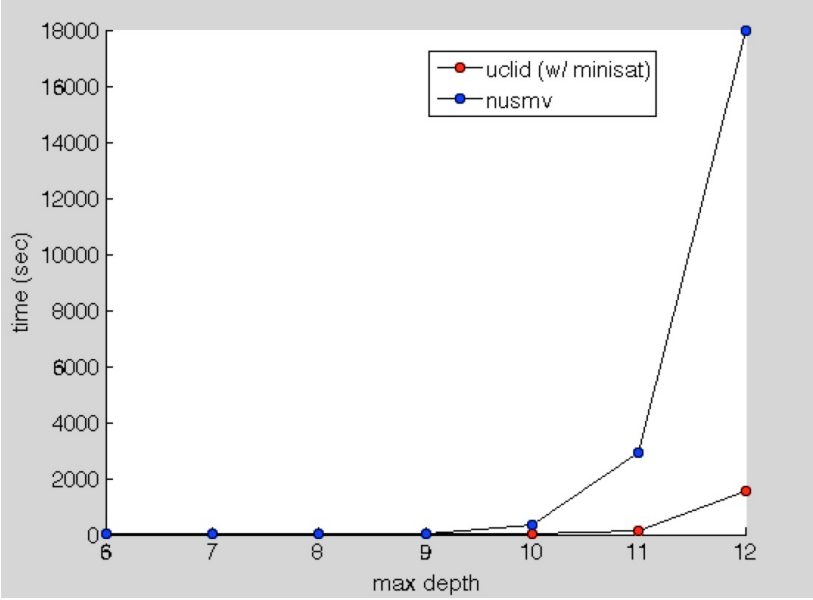

Figure 3.4: Runtime vs. max depth for finding paths for $C_6H_{13}O_9P$

With the slow performance of NuSMV at bounds greater than 10, we attempted to use other model checking techniques to solve this problem. In particular, we used UCLID [2][3][9], a model checker based on satisfiability modulo theories (SMT) solving [1]. As shown in Figure 3.4, UCLID dramatically improved the runtime. Moreover, UCLID can be used with any back-end Boolean satisfiability (SAT) solver, and varying the SAT solver paired with UCLID yields further improvements in runtime as shown in Figure 3.5.

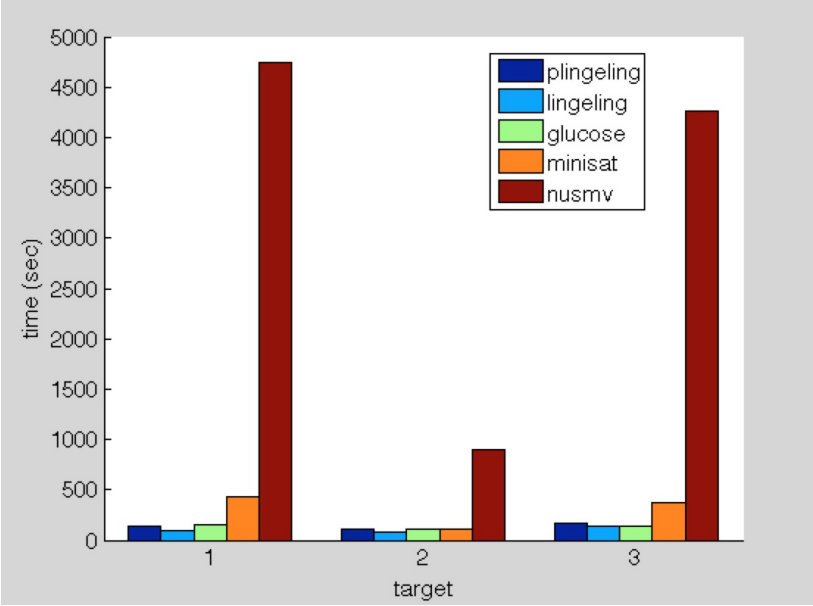

Figure 3.5: Comparison of SAT solvers after encoding with UCLID, depth 11 (targets: (1) $C_5H_{11}O_7P$, (2) $C_7H_5NO_4$, (3) $C_3H_7O_6P$).

NuSMV average	2048s
UCLID (w/ MiniSAT) average	161s
average speedup	24.9x
best speedup	59x
worst speedup	8.1x

Table 3.3: Comparison of NuSMV to UCLID (with MiniSAT) for 19 targets with a maximum depth of 11.

We have run this procedure on 19 different targets with UCLID, and for a bound of 11, the average runtime for these is 161 seconds, with a minimum of 11 seconds. Other metrics can be seen in Table 3.3. Some examples of other chemical targets we analyzed are d-glucosamine-6-phosphate and 3-hydroxypropionaldehyde, which is a component of the antimicrobial compound Reuterin. We are further investigating the quality of pathways past a certain reaction threshold, as well as other methods to speed up our runtime.

3.6 Future Work

From our early experiments, we already see what a large impact our software choices have on the runtime of the problem, which indicates that further optimization could make greater depths more tractable. The main topic of focus is now the variance among the average runtimes for different targets, and what causes such wide differences between them. In addition, it is interesting to consider limiting the expansions of certain chemicals. For many targets, the bulk of the complexity of the model comes from having thousands of ways of creating an early, precursor compound. For example, in Figure 3.6 below, three non-dominated pathways to producing (2Z)-2-hydroxypenta-2,4-dienoate are displayed. These generated minimal pathways involve reactions that actually produce a water molecule, and they are treated as non-dominated pathways because they don't involve the use of water as a starting cost. This doesn't just apply for water, but other small (and common) products as well.

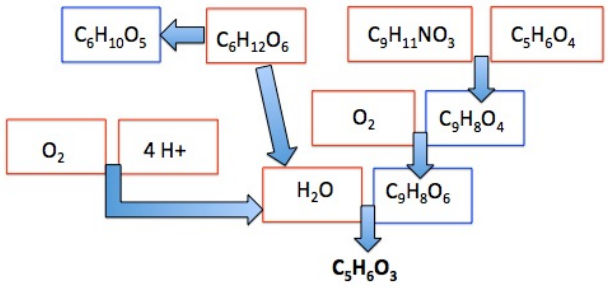

Figure 3.6: Pathway to produce (2Z)-2-hydroxypenta-2,4-dienoate. Native compounds boxed in red.

Non-dominated cost
1 $C_9H_{11}NO_3$, 1 $C_5H_6O_4$, 1 O_2, 1 H_2O
1 $C_9H_{11}NO_3$, 1 $C_5H_6O_4$, 2 O_2, 4 H+
1 $C_9H_{11}NO_3$, 1 $C_5H_6O_4$, 1 O_2, 1 $C_6H_{12}O_6$

Table 3.4: Non-dominated costs from the pathways depicted in Fig 3.6.

3.7 Conclusion

We have shown a viable method of finding SMSPs, with the important distinction of generating all possible SMSPs (bounded by an input search depth). This differs from previous works which only take into account carbon flux [12], provide a subset of "good" paths with a genetic algorithm [15], or focus on just modeling the pathway [8]. Although runtime is currently the biggest concern with this algorithm, we suggest many directions in which this procedure may be modified to help alleviate the problem. Overall, we have provided the first formalization of the SMSP problem, as well as a complete algorithm, utilizing model checking, to solve it.

Bibliography

[1] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 4, chapter 8. IOS Press, 2009.

[2] R. E. Bryant, S. K. Lahiri, and S. A. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *CAV*, LNCS 2404, pages 78–92, July 2002.

[3] R. E. Bryant and S. A. Seshia. UCLID. `http://uclid.eecs.berkeley.edu/index.html`.

[4] A. Cimatti, E. M. Clarke, E. Giunchiglia, and et al. NuSMV 2: An opensource tool for symbolic model checking. In *CAV*, pages 359–364, 2002.

[5] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[6] M. Fong and S. A. Seshia. Stoichiometrically minimal source pathways via model checking. Manuscript submitted for publication, August 2015.

[7] P. Hansen. Bicriterion path problems. *Multiple Criteria Decision Making Theory and Application*, 177:109–127, 1990.

[8] I. Koch, B. H. Junker, and M. Heiner. Application of petri net theory for modelling and validation of the sucrose breakdown pathway in the potato tuber. *Bioinformatics*, 21(7):1219–1226, 2005.

[9] S. K. Lahiri and S. A. Seshia. The UCLID decision procedure. LNCS 3114, pages 475–478. Springer-Verlag, July 2004.

[10] E. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16:236–245, 1984.

[11] G. A. Pavlopoulos, M. Secrier, C. N. Moschopoulos, T. G. Soldatos, S. Kossida, J. Aerts, R. Schneider, P. G. Bagos, et al. Using graph theory to analyze biological networks. *BioData mining*, 4(1):10, 2011.

[12] J. Pey, J. Prada, J. E. Beasley, and F. J. Planes. Path finding methods accounting for stoichiometry in metabolic networks. *Genome Biol*, 12(5):R49, 2011.

[13] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977.

[14] L. T. Rosario Garroppo, Stefano Giordano. A survey on multi-constrained optimal path computation: Exact and approximate algorithms. *Computer Networks*, 54:3081–2107, 2010.

[15] A. Seressiotis and J. E. Bailey. MPS: an artificially intelligent software system for the analysis and synthesis of metabolic pathways. *Biotechnology and bioengineering*, 31(6):587–602, 1988.

[16] S. Srivastava, J. Kotker, S. Hamilton, and et al. Pathway synthesis using the Act ontology. In *Proceedings of the 4th International Workshop on Bio-Design Automation (IWBDA)*, June 2012.

[17] G. Tsaggouris and C. Zaroliagis. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. *Algorithms and Computation*, 4288:389–398, 2006.

[18] A. Warburton. Approximation of pareto optima in multiple-objective, shortest-path problems. *Operations Research*, 35:70–79, 1987.