Enzymatic target identification with dynamic states

Bin Song, Sanjay Ranka, Tamer Kahveci
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611, USA
{bsong,ranka,tamer}@cise.ufl.edu

ABSTRACT

As a metabolic network reaches from a state to a steady state, a subset of its fluxes gradually change. The sequence of intermediate states, called the *dynamic state*, shows the pattern that the underlying network follows to reach the steady state Understanding this pattern is crucial for metabolic engineering as the intermediate states can lead to undesired effects such as toxicity although the final steady state is a desired one. In this paper, we consider the problem of enzymatic target identification in metabolic networks. Unlike existing strategies, we consider the dynamic behavior of the state changes of the networks. Given a goal pattern for the fluxes of a given network, we aim to find the set of enzyme knockouts that will produce a dynamic state similar to that goal pattern. We consider three distance functions to measure the difference between two dynamic states. These are the Euclidean distance, time-warping distance and pattern distance. Euclidean distance restricts the solution space to the exact goal state. Time-warping distance allows for stretching of the goal pattern in the time domain. Pattern distance allows scaling and shifting of the goal flux in addition to stretching in the time domain. We provide a branch and bound method to solve this problem. We also develop a partitioning strategy to reduce the running time of our method. This strategy avoids constructing the entire dynamic state by computing a lower bound to the distance between two dynamic states when the entire dynamic state is not available. Our experiments on the purine metabolism show that our method runs accurately. They also show that our partitioning strategy reduces the number of time intervals computed for dynamic states by a factor of 2 to 6.

1. INTRODUCTION

Metabolic networks show how enzymes and compounds interact through reactions. Enzymes catalyze reactions that transform a set of compounds into another set of compounds. The *state* of a metabolic pathway can be expressed as a vector, where each entry denotes the concentration of a compound [18] or a flux [9] in the network at a given time. *Steady state* is the state that remains unchanged over time. A number of methods have been developed to compute the steady state of a given metabolism for different network models (See Voit's book for an overview of these models and methods [19]).

When an enzyme is inhibited, it cannot catalyze the reactions it is responsible from. As a result, the production of a set of compounds can change. Following from this observation, the *enzymatic target identification problem* aims to identify the set of enzymes whose knockouts lead the steady state of the metabolism close to a given goal state [14]. In the literature, this problem has been considered for a number of network models including Boolean [17, 16, 15] and stoichiometric [14] models.

The enzymatic target identification problem above and the existing solutions for this problem have a serious shortcoming. This is because they consider only the steady state of the given metabolic network. However, the biological system reaches to the steady state over a period of time, after a sequence of changes to its current state. In other words, reaching to a steady state is a process that involves observing many other states. Inhibition of two different sets of enzymes can lead to the same steady state in two different ways. Figure 1 illustrates this on a hypothetical example by focusing on the concentration of a single compound. The pattern of states observed while reaching a steady state is a critical information that is ignored by current enzymatic target identification methods. For instance, if the blood sugar concentration increases (or decreases) too rapidly or if it becomes over (or below) a threshold, this pattern may cause dangerous side-effects. Thus, it is necessary to consider the dynamic process while solving the enzymatic target identification problem.

We use the term *dynamic state* to describe the sequence of states observed over time. We consider the dynamic state as a vector where the ith entry denotes the state of the network at the ith time instance. There are several models to describe the dynamic state of the biological system. We discuss them in Section 2. However, to the best of our knowledge, there is no published study that considers the dynamic states for the enzyme target identification problem. Following from this observation, we focus the following problem in this paper.

> **Problem statement.** (*dynamic enzymatic target identification problem*) Given a goal dynamic state of a set of compounds in a metabolic network, we aim to identify the set of enzymes whose inhibition leads to the dynamic state of these compounds as close to the goal dynamic state as possible.

Contributions: We address the dynamic enzymatic identification problem in this paper. In order to solve this problem, it is necessary to provide a measure to evaluate the difference between two

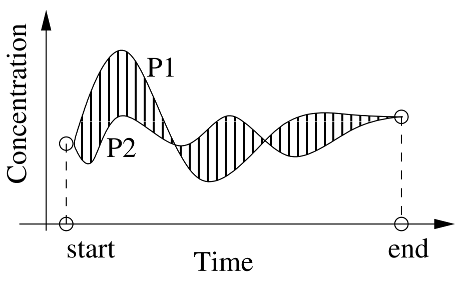

Figure 1: The patterns P_1 and P_2 show the sequence of concentration of a compound resulting from two alternative manipulations to the metabolic network. Both manipulations lead to the same steady state in the same time, however they do this through different sequence of states. The area between the two plots show the difference between two dynamic states P_1 and P_2 of one compound.

dynamic states. We consider three alternative measures to cover a broad range of distance functions. The first one measures the area between the curves defined by the two dynamic states. For example, in Figure 1, the shaded region corresponds to the distance between dynamic states P_1 and P_2. This measure makes the simplifying assumption that the metabolism reaches to the steady states in the same amount of time when different enzyme sets are inhibited. Our second measure eliminates this assumption by allowing the dynamic states stretch along the time dimension by arbitrary amounts. Figure 2 depicts this on a simple pattern representing a hypothetical dynamic state. We build a dynamic programming solution to compute this distance function. Our third measure further generalizes the distance measure by allowing scaling and shifting of dynamic states (see Figure 2). This generalization is motivated by the fact that two different patterns can have similar trends while having significantly different values. We build an iterative algorithm that finds the distance between two dynamic states when one of them is allowed to stretch along the time domain as well as shift/scale on the dimensions corresponding to the flux values.

We develop a branch and bound strategy that uses these distance measures to solve the dynamic enzymatic target identification problem. The basic search strategy follows the standard OPMET algorithm for Boolean networks [17]. It considers the search space as a hierarchical tree where each node of this tree corresponds to a set of enzymes to be inhibited. The fundamental difference is that each node now corresponds to a dynamic state. This difference introduces additional computational cost on the basic OPMET algorithm in two different ways. First, computing the dynamic state can be significantly harder than just computing the steady state. Second, the time complexity of computing the distance between two dynamic states can be quadratic in the number of time instances, while that for the steady states only is constant. We deal with these two challenges by develop a partitioning strategy as follows. Instead of creating the entire dynamic state, we quickly create a short prefix of it. We then use this prefix to compute a lower bound to the distance between the entire dynamic states. This bound helps us to prune unpromising solutions in the search space. We extend this prefix by computing more values in the dynamic state if needed. As longer prefix of the dynamic state becomes available, we improve the lower bound using the new values for further pruning of the search space. Our experiments demonstrate that our method is

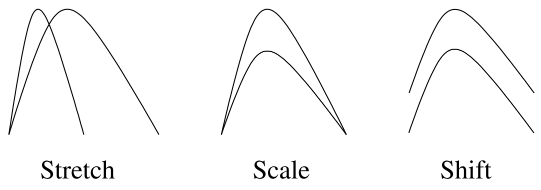

Figure 2: Three transformations, stretch, scale and shift applied on a hypothetical dynamic state.

85-100% accurate when a single enzyme is inhibited. It is 65-75% accurate when two enzymes are inhibited. Furthermore, our partitioning strategy reduces the number of time intervals computed for dynamic states by a factor of 2 to 6.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 defines three dynamic distance functions to measure the difference between two dynamic states. Section 4 discusses how we search the set of possible solutions and our partitioning strategy. Section 5 presents experimental results. Section 6 concludes the paper.

2. RELATED WORK

Existing methods for enzymatic target identification use three models to explain how metabolism works, namely Boolean, linear and non-linear models. A detailed discussion of these methods is available at Song et.al., [14]. We briefly summarize some of them here. Sridhar et. al and Song et. al considered a Boolean model of the enzymatic target identification problem [16, 17, 15]. In their version, each entry of the state denotes whether the corresponding enzyme is present or not. *Flux Balance Analysis, (FBA)* [5, 1, 4, 13] uses a set of linear equations to describe a given metabolic network. Methods, that use this system often use Linear Programming (LP), Integer Linear Programming (ILP) or genetic algorithms to solve the enzymatic target identification problem. OptKnock [2] and the method by Patil et.al., [10] are two examples to the algorithms in this class [2]. Though the linear model works well for some cases, there exist more complex non-linear models to describe the metabolism. These non-linear models can simulate the cell system better than the linear model. For example, S-systems [12, 19] and Generalized Mass Action (GMA) model [11, 19] belong to this category. Most of the existing methods are suited well for linear models. Thus they do not work when these more complex models are used to compute the steady state of the metabolic network. Song et. al proposed a genetic algorithm solution for non-linear models [14]. *All the above mentioned methods consider only the steady state of the metabolism. They ignore the sequence of states the underlying network visits while reaching the steady state. As a result, although their solution may be optimal at the steady state, the intermediate states of their solutions can be undesirable.*

There are several models that simulate the dynamic state of a given metabolic network. For example, Dynamic Flux Balance Analysis (DFBA) extended the traditional FBA to describe the change rate of the fluxes over a period of time [8, 7]. DFBA incorporates the time parameter which can predict the metabolite concentrations. It considers the entire time period and builds a non-linear programming problem. It separates the time into several intervals. For each interval it employs a linear-programming method to estimate the flux values during that interval. Integrated dynamic FBA (idFBA) simulates the integrated system including signaling, metabolic and regulatory networks [6]. Similar to DFBA,, idFBA separates the time into several intervals. For each interval, it applies FBA to compute the flux values. From these values, it decides which reactions will take place during the next interval. Integrated FBA (iFBA) model builds a dynamic simulation among metabolic, regulatory and signaling networks [3] along the same lines as DFBA and idFBA. It first separates the time to several intervals. It then applies ordinary differential equations (ODEs) and Boolean regulatory model to constrain the FBA linear programming problem. It updates the biomass and external metabolite concentrations for use in subsequent time steps. All these methods aim to find the dynamic state of a given metabolic network. *They however do not consider the dynamic enzymatic target identification problem, which is the focus of this paper.*

In this paper, we use Generalized Mass Action (GMA) model [11, 19] to model metabolic networks as this is one of the most accurate models. It is a generalization of the S-systems of equations. We employ DFBA to compute the dynamic state of the given metabolic network. It is worth noting that one can replace these with another mathematical model for computing the dynamic state with little or no change to the rest of this paper.

3. COMPARING DYNAMIC STATES

In order to solve the dynamic enzymatic target identification problem, the first task is to measure the distance between two dynamic states. We first present the basic notation we use in this paper in Section 3.1 We then describe three distance measures. Section 3.2 discusses the exact distance. Section 3.3 discusses how we can measure the distance when we allow flexibility to the amount of time it takes for the metabolic network to reach the steady state. Section 3.4 describes the pattern distance that allows the dynamic state to be scaled of shifted by arbitrary amounts.

3.1 Notation

We start by describing the notation that will be used in the rest of the paper. Assume that a given metabolic network contains M fluxes. Let use denote the fluxes in the given metabolic network with $X_1, X_2, \cdots, X_M$. Let us denote the amount of instantaneous change in X_i with $\dot{X}_i$. Assume that X_i takes part in N reactions. The GMA model expresses the flux of each X_i using the equation

$$\dot{X}_i = \sum_{k=1}^{N} \gamma_{i,k} \prod_{j=1}^{M} X_j^{f_{i,j,k}}.$$

Here, the constant $\gamma_{i,k}$ is the speed of the kth reaction and the constant $f_{i,j,k}$ is the rate of contribution of the jth flux to the kth reaction. For a given set of equations of this form, when the initial values (i.e., the value at time zero) of all the X_is are provided, we can compute the value of each X_i at a given time t by simulating this process or by solving these equations.

We represent the dynamic state of X_i using two vectors $X_{i,T}$ and $X_{i,V}$. The first one, $X_{i,T}$ = $[xt_{i,1}, xt_{i,2}, \cdots, xt_{i,n}]$, shows the time points at which me compute the value of X_i until the metabolic network reaches to the steady state ($\forall j, xt_{i,j} < xt_{i,j+1}$). The second one, $X_{i,V}$ = $[xv_{i,1}, xv_{i,2}, \cdots, xv_{i,n}]$, shows the values of X_i at these time points. The collection of $X_{i,T}$ and $X_{i,V}$ for all X_is is the dynamic state of the given metabolic network.

In order to simplify our notation, in the rest of the paper we will focus on a particular X_i and represent its dynamic state with the vector pair X = (X_V, X_T), where X_V = $[xv_1, xv_2, \cdots, xt_n]$ and X_T = $[xt_1, xt_2, \cdots, xt_n]$ by dropping the subscript i.

We use a similar notation to denote the goal dynamic state for X_i using a vector pair G = (G_V, G_T). The first vector G_V = $[gv_1, gv_2, \ldots, gv_m]$ denotes the goal values for X_i until it reaches to steady state. The second vector G_T = $[gt_1, gt_2, \ldots, gt_m]$ stores the ideal time for X_i to have each of these values.

3.2 Exact distance

Our first distance function measures the area between the two curves defined by the two given dynamic states. We name this measure the *Exact distance* The area of the shaded region in Figure 1 shows the exact distance on a hypothetical example.

Given a goal dynamic state (G_V, G_T) and the dynamic state (X_V, X_T), this measure assumes that G_T and X_T has exactly the same entries. Although this is a strong assumption, we ensure that this is satisfied as follows. For all the values in G_T that are missing in X_T we insert those values in X_T and compute the value of X_V at those new time points using the GMA model as described in Section 3.1. For all the values in X_T that are missing in G_T we insert those values in G_T. We follow a different procedure to compute the G_V values for the new time points as there is no metabolic network available for the goal state. We simply use linear interpolation of the existing time points and values in G_T and G_V to compute the new entries for G_V.

Once we ensure the equality of the vectors G_T and X_T, we compute the exact distance between the two dynamic states as:

$$\sum_i \frac{(|gv_i - xv_i| + |gv_{i+1} - xv_{i+1}|) \times |xt_{i+1} - xt_i|}{2}.$$

Geometrically, this equation approximates to the area between the two curves by splitting the region between them into trapezoids between consecutive time intervals. Each summation term in the above formulation corresponds to the area of one trapezoid.

The exact distance measure can be computed efficiently in O(n) time, where n is the number of time points. It however is restrictive as it returns a small value only if the two dynamic states have the similar values at similar times. In the following sections, we will discuss how we relax this constraint.

3.3 Time-warping distance

The exact distance function requires the two states to have similar flux values for the same period of time. This restriction, however, will fail to find the similarities between two pathways under the following scenario. It is possible to have two different metabolisms that have similar sets of reactions but the speed at which the reactions take place differ. In such scenarios, the dynamic states of the two metabolisms can have similar values but the time it takes to reach these values will differ. We say that such dynamic states are *stretched* along the time axis. The left-most dynamic states in Figure 2 illustrate a pair of dynamic states with one stretched.

Our second distance function uses time-warping distance function to address problems caused by the metabolic manipulations that change the speed of the reactions. Assume that we are given a goal dynamic state G = (G_V, G_T) and the dynamic state X = (X_V, X_T) as described in Section 3.1. Time-warping distance aligns the two dynamic states to find a mapping between their time points. Figure 3 illustrates the alignment of two hypothetical dynamic states and how the time-warping distance stretches them to bring their similar values close to each other. Once the two dynamic states are aligned, this measure computes the distance as the area between their curves after the dynamic state (X_V, X_T) is stretched along the time dimension to match its time points to those of the goal dynamic state.

We use dynamic programming method to align the two dynamic states as follows. Let us denote the distance between the ith value of G and the jth value of X with $d(G, i, X, j)$. We discuss computation of $d(G, i, X, j)$ later in this section.

Let us also denote the distance between the first i values of G and the first j values of X after their optimal alignment (i.e., alignment with smallest distance) with $\gamma(i, j)$. We compute $\gamma(i, j)$ as

$$d(G, i, X, j) + \min\{\gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1)\}.$$

The first of the three scenarios in the min function correspond to the case that the first $i-1$ values of G is aligned to the first $j-1$ values of X. The second scenario corresponds to the case that the first $i-1$ values of G is aligned to the first j values of X. In other words X is stretched along the time axis to match its jth entry to both ith and $(i-1)$th entries of G. The last scenario corresponds to the case that the first i values of G is aligned to the first $j-1$ values of X. That is X is contracted along the time axis to match both of its $(j-1)$th and jth entries to the ith entry of G. We compute the time-warping distance between G and X as

$$Dis(G, X) = \sqrt{\gamma(m, n)}.$$

We skipped two important details in computation of the time-warping distance so far. The first one is the distance between two time points of the two dynamic states $d(G, i, X, j)$. We compute this value as

$$(gv_i - xv_j)^2 \cdot (\frac{gt_{i+1} - gt_{i-1}}{2(gt_m - gt_1)} + \frac{xt_{j+1} - xt_{j-1}}{2(xt_n - xt_1)})/2.$$

Briefly, this function approximates to the area between the two curves when the jth interval of X is moved to the ith interval of G at that time interval. It approximates this area as the sum of two triangles. The first triangle defines the area in the time interval $[gt_{i-1}, gt_{i+1}]$. The second one defines that in the time interval $[xt_{j-1}, xt_{j+1}]$.

The second detail we omitted is the initialization of the $\gamma(i, j)$ matrix. We initialize this matrix for the cases when at least one of the two dynamic states contain no values as follows.

- *Case 1:* $\gamma(0, 0) = 0$
- *Case 2:* $\gamma(i, 0) = \gamma(i-1, 0) + gv_i^2 \cdot \frac{(gt_{i+1} - gt_{i-1})}{4(gt_m - gt_1)}$
- *Case 3:* $\gamma(0, j) = \gamma(0, j-1) + xv_j^2 \cdot \frac{(xt_{i+1} - xt_{i-1})}{4(xt_n - xt_1)}$

The first case indicates that both dynamic states have no value. The last two cases indicate that one of the two dynamic states have values. In this case the distance is the area under the nonempty curve.

3.4 Pattern distance

We have discussed how to deal with the differences in the dynamic states due to differences in the reaction speeds using time warping distance. Two similar networks, however, can have different dynamic states even under the time warping distance when the values of one dynamic state is shifted or scaled. The dynamic states in the middle and on the right of Figure 2 illustrate this event. This kind of alterations often happens due to external factors, such as increasing or decreasing the concentrations of a set of compounds. It can also be caused by inaccuracies in measurements. Our last distance measure computes the smallest distance between two dynamic states when one of them is shifted, scaled and stretched to make as close to the other as possible. We call this measure the *pattern distance*.

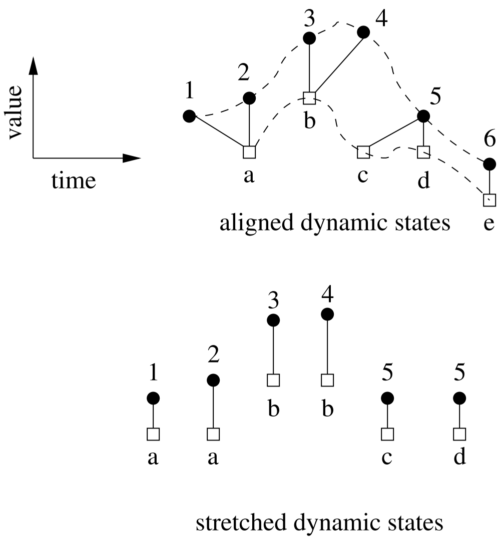

Figure 3: An illustration of two dynamic states. Each square (circle) shows a time point. The time points of one of the dynamic states is labeled with numbers 1 to 6. Those of the other are labeled with letters a to e. One of the dynamic states is shifted down slightly to make it easier to visualize. Top figure: Dashed lines show the actual pattern of the dynamic states. Solid lines show the time-warping alignment of the two states. Bottom figure: The time points that are aligned with multiple time points from the other state are duplicated and stretched to match them along the time axis.

Assume that we are given a goal dynamic state G = (G_V, G_T) and the dynamic state X = (X_V, X_T) as described in Section 3.1. Let α and β be two real numbers. Scaling the goal state with α moves G to $(\alpha \cdot G_V, G_T)$. Similarly, shifting G by β moves G to $(G_V + \beta, G_T)$.

If we are given the optimal scaling and shifting coefficients α and β, one can find the distance between G and X as the time warping distance between X and $(\alpha \cdot G_V + \beta, G_T)$ using the dynamic programming method in Section 3.3. This, however, is not feasible as the values of α and β however are not available. Inversely, given an alignment of the time points of G and X, through algebraic manipulations, we can compute the scaling and shifting coefficient α and β that will minimize the distance between them. We discuss how this can be done later in this section. However, this is not feasible as well as their alignment is unknown.

We develop an iterative algorithm to solve this problem (see Algorithm 1). Each iteration of our algorithm contains two phases. In the first phase, we fix the values of α and β and optimize the alignment. In the second phase, we fix the alignment and compute the optimal α and β values for that alignment. Each phase guarantees that the resulting distance is less than or equal to that in the previous phase/step. Thus, the distance value our algorithm computes decreases monotonically throughout iterations. Thus, the algorithm is guaranteed to converge to a minimal distance. Next, we focus on Phase 2 of our algorithm and explain how to compute the optimal values of α and β.

At the end of Phase 1 (Step 2) of Algorithm 1, we have an align-

Algorithm 1 *Iterative Algorithm for Pattern Distance*

```
Input: Dynamic states $G$ and $X$
1: Initialize $\alpha = 1$ and $\beta = 0$.
2: Phase 1: Use the dynamic programming method of Section 3.3 to find
   the alignment of $G$ and $X$.
3: Phase 2: Compute the $\alpha$ and $\beta$ values that minimize the distance for
   the current alignment. Update $G_V$ as $\alpha G_V + \beta$.
4: Go to step 2 until a given number iterations or no distance improve-
   ment.
```

ment between G and X. Recall that, this alignment can map one time point of one dynamic state to several consecutive time points of the other. For instance in Figure 3, time point "a" is mapped to time points "1" and "2". To make our notation for the rest of this section simple, we duplicate such time points (see the bottom figure in Figure 3). Let us call the resulting dynamic states as $G' = (G'_V, G'_T)$ and $X' = (X'_V, X'_T)$, where G' and X' both have same number of (n) time points. In other words, the ith time point of G' is aligned to the ith time point of X'. Formally, we will use the following notation for G' and X':

- $X'_V = [xv'_1, xv'_2, \cdots, xv'_n]$
- $X'_T = [xt'_1, xt'_2, \cdots, xt'_n]$
- $G'_V = [gv'_1, gv'_2, \ldots, gv'_n]$
- $G'_T = [gt'_1, gt'_2, \ldots, gt'_n]$

We define two functions over time intervals of G' and X' as

$$\omega(gt'_i) = \frac{gt'_{i+1} - gt'_{i-1}}{2(gt'_n - gt_1)} \text{ and } \omega(xt'_i) = \frac{xt'_{i+1} - xt'_{i-1}}{2(xt'_n - xt'_1)}.$$

The pattern distance between G and x is

$$Dis(G, X) = \sqrt{\sum_i (\alpha \cdot gv'_i + \beta - rv'_i)^2 \cdot (\omega(gt'_i) + \omega(rt'_i))/2}.$$

We solve the following equations to get the values of α and β.

$$\frac{\partial Dis(G, X)}{\partial \alpha} = \frac{\partial Dis(G, X)}{\partial \beta} = 0.$$

We skip the individual algebraic steps on how we solve these equations and present the final result here. The values of α and β are

$$\alpha = \frac{C - B \cdot D}{A - B^2} \text{ and}$$
$$\beta = D - \alpha \cdot B,$$

where

$$A = \sum_i (\omega(gt'_i) + \omega(xt'_i)) \cdot (gv'_i)^2,$$
$$B = \sum_i (\omega(gt'_i) + \omega(xt'_i)) \cdot gv'_i,$$
$$C = \sum_i (\omega(gt'_i) + \omega(xt'_i)) \cdot xv'_i \cdot gv'_i, \text{ and}$$
$$D = \sum_i (\omega(gt'_i) + \omega(xt'_i)) \cdot xv'_i.$$

As we mentioned earlier, this algorithm is guaranteed to converge to a minimal distance. This is because at the end of each of the two phases we gurantee that the new distance is less than or equal to the earlier distance values. It is worth mentioning that this algorithm, however, does not guarantee to find the overall minimum distance. It may end up in a local minima. That said, it is possible to avoid current local minima using standard techniques such as simulated annealing.

4. FINDING THE ENZYMATIC TARGETS

So far we have discussed how to compute the distance between two dynamic states. The next step is to identify the set of enzymes whose elimination leads to the dynamic state as close to the goal dynamic state as possible with respect to this distance function. One way to solve this problem is to exhaustively traverse all possible subsets of enzymes, examine the distance value and pick the lowest one. However, this is not feasible as the number of subsets is exponential in the number of enzymes.

OPMET solves this problem using a branch and bound algorithm for a simplified Boolean model of metabolic networks [17]. At this paragraph, we take a detour and summarize the OPMET algorithm as we use it in this paper. Unlike the problem considered in this paper, OPMET considers only the steady state (i.e., the last value of the dynamic state) instead of the entire dynamic state. It systematically searches subsets of enzymes. Each node in the branch-and-bound search tree is a candidate solution (i.e., a set of enzymes to be inhibited). The root node contains the empty set. As the search space is traversed, OPMET keeps the current best node with the minimum distance so far as the *current best solution* and the associated distance as the *global cut-off threshold*. If the current node has distance less than the current threshold, it saves that node as the new best solution and updates the global threshold with the current distance value. It then selects another enzyme to insert into the current set of inhibited enzymes if the insertion improves the current solution. Otherwise, it backtrack to the previous solution.

We use the OPMET algorithm to search subsets of enzymes systematically to find the one with the closest dynamic state to the goal state. Using the dynamic state, however, introduces a new challenge that does not exist in OPMET. The computational cost for computing the dynamic state X of a metabolic network often dominates the time it takes to compute the distance between two dynamic states. The gap between the two depends on the number of equations and variables that explain the fluxes of the metabolic network as well as the dependencies between those variables. Therefore, it is essential that we avoid computing the dynamic state for the non-promising enzyme subsets. We achieve this for each subset of enzymes as follows. We do not compute the entire dynamic state X. Instead, we compute only a short prefix of the values in X. Using this prefix, we compute a lower bound to the distance between G and the entire X. Computation of the lower bound varies for the distance measures we defined in Section 3. We discuss how to compute the lower bound for each of the three distance measures next.

Exact distance. Our solution follows from the observation that this distance function is additive over the time domain. That is the exact distance between two dynamic states over a time period is simply the sum of their distances of the shorter time intervals that make that time period.

We split the entire time span till the metabolic network reaches steady state into non-overlapping intervals. We compute the dynamic state X only for the first time interval. We compute its distance to G inside this time interval by considering only those values of G that are within this interval as described in Section 3.2. If the distance is greater than the global cut-off threshold we filter that en-

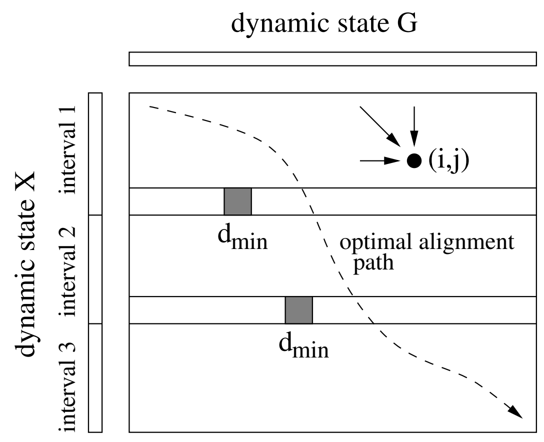

Figure 4: An illustration of how partitioning the dynamic state X into shorter intervals improves the running time. Solid arrows show the entries in the dynamic programming matrix that can contribute to the computation of entry $(i,\, j)$. The dashed arrow shows the path corresponding to the optimal alignment of G and X. $d_{\min}$ (i.e., value of the shaded entry) is the minimum value at each row. If the value of $d_{\min}$ at the end of an interval is more than the cutoff we do not need to compute the rest of the dynamic state X.

zyme set. Otherwise, we compute the values of X for the next time interval and update the distance with these new values. As the distance is additive and nonnegative, it monotonically increases with the number of time intervals. We repeat updating the distance with new intervals until the distance exceeds the threshold or we compute the entire X. This way, we can avoid computing the entire X if the distance grows beyond the cutoff quickly.

Time-warping/pattern distance: Both time-warping and pattern distances use dynamic programming to compute the alignment of two dynamic states. As a result, these distances are not additive. We develop a different strategy for them. Next, we explain only the time-warping distance as both distances use the same dynamic programming method.

Figure 4 illustrates our strategy. Similar to the exact distance, we compute the dynamic state X only for a short prefix of X. Assume that this prefix contains m_1 values we compute the values of $\lambda(m_1, j)$ for all $j \in [1, n]$. If the smallest value among $\lambda(m_1, j), \forall j$ is greater than the global cut-off threshold we filter that enzyme set. Otherwise, we compute the values of X for the next time interval and continue filling the dynamic programming matrix. We repeat this process until the minimum distance in a row of the dynamic programming matrix exceeds the threshold or we compute the entire X. This way, we can avoid computing the entire X if the minimum distance grows beyond the cutoff quickly.

5. EXPERIMENTAL RESULTS

In this section, we evaluate our algorithms on real datasets.

Dataset: We test our three distance functions and the search on the purine metabolism system as all the reaction coefficients needed to compute the dynamic states are available in literature for this network. Purine metabolism is an important metabolic network. It synthesizes and breaks down purines. We use Voit's computational model for purine metabolism [19]. We compute the dynamic state of the purine metabolism by solving the GMA system equations. The kinetic orders and rate constants of the corresponding GMA system equations are available in the literature [19]. Briefly this dataset contains 36 fluxes and 16 ODEs that describe the relationship between different variables.

Query sets: We created four query sets, namely Q_1, Q_2, Q_3 and Q_4 from the purine metabolism dataset [19]. Each query set contains 10 goal dynamic states. We created each query in Q_1 as follows. We randomly selected one enzyme from the purine metabolism system. We then computed the the dynamic state of that metabolism after inhibiting that enzyme and used it as the goal state. Similarly, we created each query in Q_2, Q_3 and Q_4 by inhibiting two, three and four randomly selected enzymes respectively. This ensures that there is a solution to the dynamic enzymatic target identification problem for each query with zero distance. In order to simulate different kinds of mutations on the goal dynamic state, we also created three more query sets for each of Q_1, Q_2, Q_3 and Q_4 as follows.

- *Stretch.* The first one simulates stretching in the time domain. For this, we stretched each query by randomly shifting its time value at each time point by a random amount. We denote these query sets by including the prefix S- to the query set (e.g., S-Q_1).
- *Shift/Scale.* This query set simulates scaling and shifting mutations. We obtained this by scaling and shifting the dynamic states in the original query sets by random α and β values. We denote these query sets by including the prefix SS- to the query set (e.g., SS-Q_1).
- *Shift/Scale/Stretch.* This query set simulates scaling and shifting mutations as well as stretching in the time domain. We obtained this by scaling and shifting the dynamic states in the stretched query sets by random α and β values. We denote these query sets by including the prefix SSS- to the query set (e.g., SSS-Q_1).

Implementation and system details: We implemented the developed algorithms in C++ and MATLAB. In our experiments, we apply the ordinary differential equation functions (e.g. ode23t, ode45) of MATLAB to compute the GMA system equations. Then, we created a C++ shared library from MATLAB M-file for dynamic state computation. Thus, we can use our traversal method to find the results. We ran our experiments on a system with Intel Core i7-920 2.66GHz Processor, 4 gigabytes of RAM, and a 64-bit Windows7 operating system.

Experimental setup: We evaluate the speed and the accuracy of our method for each of the three distance measures in terms of several metrics.

- *Execution time:* This indicates the average total time taken by our algorithms to find the enzyme set whose inhibition leads the metabolic network to the closest dynamic state.
- *Percentage of time intervals:* This indicates the percentage of the dynamic state our method computes after splitting it to smaller intervals. A small percentage is desirable as it often indicates lower running time.
- *Percentage of success:* Each goal dynamic state in our query set has a matching enzyme subset that has the smallest distance. This metric reports the percentage of queries for which

our algorithm could identify the optimal result. It is worth noting that this measure is biased against our method. This is because even when our methods does not find the optimal enzyme subset, the result it find can be very close to the optimal one. This metric considers those results as failed results. However, as we present later in this section, our method has high accuracy for this metric despite this bias.

5.1 Evaluation of the performance

Our first experiment evaluates the performance of our method with and without splitting the dynamic states in the time domain. While searching the possible enzyme sets, recall that, our algorithm splits the dynamic state into short intervals and incrementally computes these intervals. The first questions that we need to answer are: (i) What is the typical running time of our algorithm without splitting and (ii) what is the impact of splitting on the performance of our algorithm? To answer these questions, we ran queries without splitting and with splitting when we split the dynamic state into 2, 4 and 8 intervals.

Table 1 shows the performance results for the exact and the time-warping distance. When the underlying distance is the exact distance, the percentage of intervals generated tends to decrease as we split the the dynamic state into smaller pieces. This indicates that, our algorithm can often filter an enzyme subset after considering a small prefix of it. Thus by splitting each dynamic state into a larger number of (shorter) intervals we avoid computing a large portion of the dynamic state. We observe the largest improvement after splitting the dynamic states into two intervals. Further splittings gradually improves the percentage of intervals generated. As the number of intervals grows beyond four, we observed that the performance gain is negligible. The largest average running time we observed in our experiments was around two minutes (when the number of intervals is = 1 and we use the Q_4 query set). The improvement in the running time as we increase the number of intervals changes greatly from one query to another depending on the complexity of the set of ordinary differential equations and the corresponding time interval. On the average, we observed the best running times for four to six intervals.

The results for the time-warping distance follows a similar pattern. The percentage of intervals generated drops as we split the dynamic state into shorter intervals. As the number of intervals increases to four, we filter enzyme subsets after considering a small fraction (around 27%) of the of the values of the dynamic state. As we split the dynamic state further into smaller intervals, the amount of additional intervals filtered grows very slowly. Similar to the exact distance, the largest average running time we observed in our experiments was around two minutes (when the number of intervals is = 1 and we use the Q_4 query set). Filtering time intervals improved the average running time of our algorithm by a factor of two to four. However, it is worth mentioning that the running time does not necessarily drop linearly along with it number of time intervals as we discussed in the previous paragraph. *In summary, we observe that partitioning the dynamic state to a small number of intervals (such as four intervals) improves the running time. Further splittings do not help significantly. They can even increase the running time.*

5.2 Evaluation of the accuracy

This experiment evaluates the accuracy of our method using the three distance functions we defined in Section 3 on query sets with different characteristics. We measure the accuracy in terms of the *percentage of success*, i.e., the percentage of queries for which our method finds the optimal result correctly. We used all four classes of query sets we generated for this purpose. Table 2 presents the results. A set of interesting observations follow from these results.

Table 1: Comparison of the average percentage of the intervals our algorithm generates for the exact and time-warping distance with and without splitting the dynamic states for query sets Q_2, Q_3 and Q_4 into different number of intervals. 1 interval indicates that the dynamic state is not split. K intervals ($K > 1$) indicates that the dynamic state is split into K equal sized non-overlapping intervals along the time axis.

Exact distance			
Number of intervals	Q_2	Q_3	Q_4
1	100	100	100
2	72.5	60.7	64.3
4	58.7	43.9	49.0
6	58.0	40.7	47.5
8	54.9	36.8	43.7

Time-warping distance			
Number of intervals	Q_2	Q_3	Q_4
1	100	100	100
2	52.3	51.8	52.1
4	27.6	27.8	27.7
6	18.7	19.4	19.0
8	14.9	15.1	15.1

Table 2: Accuracy of our algorithm using the three distance measures on datasets with different characteristics. The accuracy values are reported in terms of percentage of success.

	Distance measure		
Query Set	Exact	Time-warping	Pattern
Q_1	**100**	**100**	**100**
Q_2	70	**75**	**75**
S-Q_1	10	**100**	**100**
S-Q_2	20	**75**	**75**
SS-Q_1	5	5	**85**
SS-Q_2	10	10	**75**
SSS-Q_1	0	0	**85**
SSS-Q_2	0	5	**65**

- When the query dynamic state can be achieved without any stretch/shift/scale transformation, our method has similar accuracies for all the three distance measures. This is expected as this is a special case of the transformations (e.g., α = 1, and β = 0).
- As the goal states are transformed through stretching along the time domain, the accuracy of the exact distance drops rapidly, while that of the other two distance functions remain identical. This justifies the need for the dynamic programming solution of the time-warping distance. This is because even small perturbations in the network can alter the result of the exact distance greatly.
- When the goal state is shifted or scaled, both the exact and the time-warping distance measures fail while the pattern dis-

tance remain to have high accuracy. The high accuracy values of the pattern distance suggests that our iterative algorithm for optimizing the pattern distance is highly accurate.

- We observe that the best accuracy values we observe in each row of this table can be less than 100% even when the query states are not transformed (i.e., consider the results for Q_2). This is caused because of the heuristic filtering strategy of OPMET [17] while searching possible enzyme subsets. This problem can be alleviated by using a more stringent filtering strategy at the expense of increased running time.

We conclude from these observations that the pattern distance is the most promising distance measure among the three. The exact distance is faster than the pattern distance. However, the gap between their accuracies when goal states are transformed justifies their small performance difference.

6. DISCUSSION

The classic enzymatic target identification problem aims to identify the set of enzymes whose knockouts lead the steady state of the metabolism close to a given goal state. This definition is problematic for a biological system reaches to the steady state over a period of time, after a sequence of changes to its current state. These sequence of states, called the dynamic states can be crucial as they can lead to serious side effects. We addressed a new variant of the enzymatic target identification problem, named the dynamic enzymatic identification problem in this paper. Unlike the existing problem we considered the entire trajectory, the given network's state follows to reach the steady state. To the best of our knowledge, this problem has not been considered in the literature so far.

We considered three alternative distance measures to compute the dissimilarity between two dynamic states, namely exact, time-warping and pattern distance. The first one measures the area between the curves defined by the two dynamic states. The second one allows the dynamic states stretch along the time dimension by arbitrary amounts. The last measure further generalizes the distance by allowing scaling and shifting of dynamic states. We exploited the OPMET algorithm to develop a branch and bound strategy that uses these distance measures to solve the dynamic enzymatic target identification problem. In order to improve the running time of this algorithm for the dynamic states, developed a partitioning strategy as follows. Instead of creating the entire dynamic state, we quickly created a short prefix of it. We then used this prefix to compute a lower bound to the distance between the entire dynamic states. If this lower bound exceeds the distance of the best solution found so far we prune that solution without generating the rest of the dynamic state. Our experiments demonstrated that our method is 85-100% accurate when a single enzyme is inhibited. It was 65-75% accurate when two enzymes are inhibited. Furthermore, our partitioning strategy reduced the number of time intervals computed for dynamic states by a factor of 2 to 6.

7. ACKNOWLEDGMENTS

This work was supported partially by NSF under grants CCF-0829867 and IIS-0845439.

8. REFERENCES

[1] H. P. J. Bonarius, G. Schmid, and J. Tramper. Flux analysis of underdetermined metabolic networks: The quest for the missing constraints. *Trends Biotechnology*, 15, 1997.

[2] Anthony P. Burgard, Priti Pharkya, and Costas D. Maranas. Optknock: A bilevel programming framework for identifying gene knockout strategies for microbial strain optimization. *Biotechnology and Bioengineering*, 84, 2003.

[3] M. W. Covert, N. Xiao, T. J. Chen, and J. R. Karr. Integrating metabolic, transcriptional regulatory and signal transduction models in Escherichia coli. *Bioinformatics*, 24(18), 2008.

[4] J. Forster, I. Famili, P Fu, B. O. Palsson, and J Nielsen. Genome-scale reconstruction of the saccharomyces cerevisiae metabolic network. *Genome Research*, 13, 2003.

[5] K. J. Kauffman, P. Prakash, and J. S. Edwards. Advances in flux balance analysis. *Current opinion in biotechnology*, 14(5), 2003.

[6] J. M. Lee, E. P. Gianchandani, J. A. Eddy, and J. A. Papin. Dynamic analysis of integrated signaling, metabolic and regulatory networks. *PLoS computational biology*, 4(5), 2008.

[7] R. Y. Luo, S. Liao, G. Y. Tao, Y. Y. Li, S. Zeng, Y. X. Li, and Q. Luo. Dynamic analysis of optimality in myocardial energy metabolism under normal and ischemic conditions. *Molecular systems biology 2*, (2006.0031), 2006.

[8] R. Mahadevan, J. S. Edwards, and F. J. Doyle III. Dynamic flux balance analysis of diauxic growth in Escherichia coli. *Biophysical Journal*, 83, 2002.

[9] B. O. Palsson. *Systems biology: Properties of reconstructed networks*. Cambridge University Press, 2006.

[10] K. R. Patil, I. Rocha, J. Forster, and J. Nielsen. Evolutionary programming as a platform for in silico metabolic engineering. *BMC Bioinformatics*, 6(308), 2005.

[11] M. Peschel and W. Mende. *The predator-prey model: do we live in a volterra world?* Akademie-Verlag, Berlin, 1986.

[12] M.A. Savageau and E.O. Voit. Recasting nonlinear differential equations as S-systems: a canonical nonlinear form. *Math. Biosci*, 87, 1987.

[13] T. Shlomi, O. Berkman, and E. Ruppin. Regulatory on/off minimization of metabolic flux changes after genetic perturbations. *Proc. Natl. Acad. Sci. USA*, 102, 2005.

[14] B. Song, I. E. Buyuktahtakin, T. Kahveci, and S. Ranka. Manipulating the steady state of metabolic pathways. , *IEEE/ACM Transactions on Computational Biology and Bioinformatics (IEEE TCBB)*, accepted for publication.

[15] B. Song, P. Sridhar, T. Kahveci, and S. Ranka. Double iterative optimization for metabolic network-based drug target identification. *International Journal of Data Mining and Bioinformatics*, (2):145–159, 2009.

[16] P. Sridhar, T. Kahveci, and S. Ranka. An iterative algorithm for metabolic network-based drug target identification. *Pacific Symposium on Biocomputing*, 2007.

[17] P. Sridhar, B. Song, T. Kahveci, and S. Ranka. OPMET: A metabolic network-based algorithm for optimal drug target identification. *Pacific Symposium on Biocomputing*, 2008.

[18] A. I. Vogel, A. R. Tatchell, B. S. Furnis, A. J. Hannaford, and P. W. G. Smith. *Vogel's textbook of practical organic chemistry*. Prentice Hall, 5 edition, 1996.

[19] Eberhard O. Voit. *Computational analysis of biochemical systems: a practical guide for biochemists and molecular biologists*. Cambridge University Press, 2000.