Accelerating the Computation of Elementary Modes Using Pattern Trees

Marco Terzer and Jörg Stelling

ETH Zurich, Department of Computer Science, 8092 Zurich, Switzerland
{marco.terzer, joerg.stelling}@inf.ethz.ch

Abstract. Elementary flux modes (EFMs)—formalized metabolic pathways—are central and comprehensive tools for metabolic network analysis under steady state conditions. They act as a generating basis for all possible flux distributions and, thus, are a minimal (constructive) description of the solution space. Algorithms to compute EFMs descend from computational geometry; they are mostly synonymous to the enumeration of extreme rays of polyhedral cones. This problem is combinatorially complex, and algorithms do not scale well. Here, we introduce new concepts for the enumeration of adjacent rays, which is one of the critical and stubborn facets of the algorithms. They rely on variants of k-d-trees to store and analyze bit sets representing (intermediary) extreme rays. *Bit set trees* allow for speed-up of computations primarily for low-dimensional problems. Extensions to *pattern trees* to narrow candidate pairs for adjacency tests scale with problem size, yielding speed-ups on the order of one magnitude relative to current algorithms. Additionally, fast algebraic tests can easily be used in the framework. This constitutes one step towards EFM analysis at the whole-cell level.

1 Introduction

Metabolic networks are characterized by their complexity. Even in simple bacteria, they involve ≈2.000 metabolites and ≈1.000 proteins that catalyze reactions converting external substrates to metabolites and products. For their computational analysis, in particular, stoichiometric or constraint-based approaches have gained popularity because the necessary reaction stoichiometries and reversibilities are usually well–characterized, in contrast to reaction kinetics and associated parameters [1]. For example, genome–scale stoichiometric models have been constructed for several organisms to predict flux distributions in metabolic networks in normal or perturbed conditions as well as optimality and control thereof [2].

Conceptually, the analysis starts from the $m \times q$ stoichiometric matrix $\mathbf{N}$, where m is the number of (internal) metabolites and q the number of reactions. As metabolism usually operates on faster time–scales than other cellular processes, we can assume (quasi) steady–state for the metabolic reactions to derive the fundamental metabolite balancing equation:

$$\boldsymbol{N} \cdot \boldsymbol{r} = \mathbf{0} \tag{1}$$

P. Bücher and B.M.E. Moret (Eds.): WABI 2006, LNBI 4175, pp. 333–343, 2006.

where the $(q \times 1)$-vector $\boldsymbol{r}$ represents a *flux distribution*. Additionally, the reaction rates $\boldsymbol{r}$ are subject to thermodynamic feasibility constraints for irreversible reactions (into which any reversible reaction can be decomposed):

$$\boldsymbol{r} \geq \boldsymbol{0} \tag{2}$$

Eqs. (1) and (2) constrain the solution space for valid reaction fluxes to a *convex polyhedral cone* P (see Section 2 for formal definitions). Hence, comprehensively analyzing metabolic network behavior amounts to characterizing P [3]. Metabolic pathways such as elementary flux modes (EFMs) or extreme pathways, which are minimal, linearly independent flux vectors unique for a given network, allow for this because they correspond to extreme rays of P [3].

Thus, computation of EFMs is equivalent to the enumeration of the extreme rays of P, a problem from computational geometry known to be hard for the general case. Current algorithms are variants of the *double description method* (DDM) introduced by Motzkin *et al.* in 1953 [4]. In particular, the *canonical basis approach* [5] and the more efficient *nullspace approach* [6] are used for EFM computation. However, no efficient algorithm is known with time complexity polynomial in the input and output size [7], which currently restricts metabolic pathway analysis to networks of ≈100 reactions and metabolites [1].

Here, we propose improved algorithms for EFM computation that address the most critical feature of the DDM, namely the independence tests for (preliminary) extreme rays. We focus on the nullspace approach, but the concepts are readily applicable to the canonical form. After giving fundamental definitions (Section 2) and a detailed description of current algorithms (Section 3), we will present our new approaches relying on k-d trees (Section 4) and experimental results showing their significant impact on performance (Section 5).

2 Fundamentals

Definition 1. *A nonempty set C of points in an Euclidean space is called a* (convex) cone *if $\lambda \boldsymbol{x} + \mu \boldsymbol{y} \in C$ whenever $\boldsymbol{x}, \boldsymbol{y} \in C$ and $\lambda, \mu \geq 0$.*

Definition 2. *A cone P is* polyhedral *if $P = \{\boldsymbol{x} \mid \boldsymbol{A}\,\boldsymbol{x} \geq \boldsymbol{0}\}$ for some matrix $\boldsymbol{A}$, i.e. P is the intersection of finitely many linear half-spaces.*

Note that $\boldsymbol{A} = [\boldsymbol{N}^T; -\boldsymbol{N}^T; \boldsymbol{I}]^T$ and $\boldsymbol{x} = \boldsymbol{r}$, with the stoichiometric matrix $\boldsymbol{N}$, identity matrix $\boldsymbol{I}$ to ensure irreversibility constraints, and the flux distribution $\boldsymbol{r}$, define the cone in the context of EFM analysis as given by eqs. (1) and (2).

Theorem 1 (Minkowski's Theorem for Polyhedral Cones). *For every cone $P = \{\boldsymbol{x} \mid \boldsymbol{A}\,\boldsymbol{x} \geq \boldsymbol{0}\}$ there exists some $\boldsymbol{R}$ such that $P = \{\boldsymbol{x} \mid \boldsymbol{x} = \boldsymbol{R}\,\boldsymbol{c}$ for some $\boldsymbol{c} \geq \boldsymbol{0}\}$ is generated by $\boldsymbol{R}$.*

$\boldsymbol{A}$ is called a *representation matrix* of the polyhedral cone P, $\boldsymbol{R}$ is the *generating matrix* for P. Because both $\boldsymbol{A}$ and $\boldsymbol{R}$ describe the same object P, the pair $(\boldsymbol{A}, \boldsymbol{R})$ is called *double description pair* or *DD pair* [4,7].

Definition 3. *For any vector* $\boldsymbol{x} \in P$*, the set* $Z(\boldsymbol{x})$*, containing the indices* i *such that* $\boldsymbol{A}_i\,\boldsymbol{x} = 0$*, is called the* zero set *of* $\boldsymbol{x}$*.*

Definition 4. *A vector* $\boldsymbol{r}$ *is said to be a* ray *of* P *if* $\boldsymbol{r} \neq \boldsymbol{0}$ *and* $\alpha\,\boldsymbol{r} \in P$ *for every* $\alpha > 0$*. Two rays* $\boldsymbol{r}$ *and* $\boldsymbol{r'}$ *are said to be* equivalent*, i.e.* $\boldsymbol{r} \simeq \boldsymbol{r'}$*, if* $\boldsymbol{r} = \alpha\,\boldsymbol{r'}$ *for some* $\alpha > 0$*.*

Definition 5. *Let* $\boldsymbol{r}$ *be a ray of* P*. If one of the following holds, both hold and* $\boldsymbol{r}$ *is called an* extreme ray*:*

(a) $rank(\boldsymbol{A}_{Z(\boldsymbol{r})}) = rank(\boldsymbol{A}) - 1$

(b) there is no $\boldsymbol{r'} \in P$ *with* $Z(\boldsymbol{r'}) \supseteq Z(\boldsymbol{r})$ *other than* $\boldsymbol{r'} \simeq \boldsymbol{r}$*.*

If all columns of $\boldsymbol{R}$ are *extreme rays*, $\boldsymbol{R}$ is called a *minimal generating set* for P.

3 Existing Algorithms

3.1 Double Description Method (DDM)

The DDM relies on the definition of *adjacent rays* that is derived from the *extreme ray definition (5)*. Thus, there exist two options to ensure adjacency, (a) sometimes referred to as *algebraic* adjacency test, (b) as *combinatorial* test:

Definition 6. *Let* $\boldsymbol{r}$ *and* $\boldsymbol{r'}$ *be two extreme rays of* P*. If one of the following holds, both hold and* $\boldsymbol{r}$ *and* $\boldsymbol{r'}$ *are said to be* adjacent*:*

(a) $rank(\boldsymbol{A}_{Z(\boldsymbol{r})\cap Z(\boldsymbol{r'})}) = rank(\boldsymbol{A}) - 2$

(b) if $\boldsymbol{r''} \in P$ *with* $Z(\boldsymbol{r''}) \supseteq Z(\boldsymbol{r}) \cap Z(\boldsymbol{r'})$ *then* $\boldsymbol{r''} \simeq \boldsymbol{r}$ *or* $\boldsymbol{r''} \simeq \boldsymbol{r'}$*.*

The algorithm constructs $\boldsymbol{R}$ from $\boldsymbol{A}$ iteratively as follows:

1. *Initialization Step:* Since P is pointed, i.e. $\boldsymbol{0}$ is an extreme point of P, $\boldsymbol{A}$ has full rank d, a nonsingular square sub-matrix $\boldsymbol{A}_d$ exists, and $(\boldsymbol{A}_d, \boldsymbol{A}_d^{-1})$ is an initial DD pair. As we will see for the *nullspace approach*, other initial pairs are possible.
2. *Iteration Step:* Assume the DD pair $(\boldsymbol{A}_j, \boldsymbol{R}_j)$ with j inequality constraints from $\boldsymbol{A}\,\boldsymbol{x} \geq \boldsymbol{0}$ already considered. The next DD pair $(\boldsymbol{A}_{j+1}, \boldsymbol{R}_{j+1})$ is achieved by fulfilling an additional inequality $\boldsymbol{a}_{j+1} := \boldsymbol{A}_{j+1}\,\boldsymbol{x} \geq 0$.
   (a) The hyperplane $H_{j+1}^0 = \{\boldsymbol{x} \mid \boldsymbol{A}_{j+1}\,\boldsymbol{x} = 0\}$ separates $\boldsymbol{R}_j$ into 3 parts:
      i. $\boldsymbol{R}_j^0$, the extreme rays of $\boldsymbol{R}_j$ fulfilling inequality $\boldsymbol{a}_{j+1}$ with equality,
      ii. $\boldsymbol{R}_j^+ \subseteq \boldsymbol{R}_j$ fulfilling $\boldsymbol{a}_{j+1}$ with strict inequality and
      iii. $\boldsymbol{R}_j^- \subseteq \boldsymbol{R}_j$ not fulfilling $\boldsymbol{a}_{j+1}$.
   (b) The matrix $\boldsymbol{R}_{j+1}$ is constructed as the union of
      i. those extreme rays that still fulfill the new condition $(\boldsymbol{R}_j^0 \cup \boldsymbol{R}_j^+)$
      ii. together with the rays resulting from the intersection of the separating hyperplane H_{j+1}^0 with the hyperplane through the pair of rays $(\boldsymbol{r}^-, \boldsymbol{r}^+)$ where $\boldsymbol{r}^- \in \boldsymbol{R}_j^-$, $\boldsymbol{r}^+ \in \boldsymbol{R}_j^+$ and $\boldsymbol{r}^-$ is *adjacent* to $\boldsymbol{r}^+$, i.e. the newly created ray is an extreme ray. This step is also known as *Gaussian elimination* with the newly constructed ray $\boldsymbol{r'}$ in H_{j+1}^0: $\boldsymbol{r'} = (\boldsymbol{A}_{j+1}\boldsymbol{r}^+)\boldsymbol{r}^- - (\boldsymbol{A}_{j+1}\boldsymbol{r}^-)\boldsymbol{r}^+$.
3. Continue with 2 until all inequalities are considered.

3.2 Binary Nullspace Algorithm

Nullspace approach. Wagner [6] proposed to use a well defined form of the kernel matrix $\boldsymbol{K}$ of $\boldsymbol{N}$ as an initial minimal representation matrix, where $\boldsymbol{K} = [\boldsymbol{I}; \boldsymbol{K^*}]^T$. If $\boldsymbol{N^{m \times q}}$ has full rank, i.e. $d = rank(\boldsymbol{N}) = m$, the kernel matrix has dimensions $q \times (q - m)$ and $\boldsymbol{K^*}$ consequently $m \times (q - m)$. Thus, this initialization results in $(q - m) + 2m = q + m$ resolved constraints, leaving m inequalities to be solved in the iteration phase. It can be shown [3] that $(\boldsymbol{A}, \boldsymbol{K})$ form an initial DD-pair with $\boldsymbol{K}$ being a minimal generating matrix and $\boldsymbol{A^{(q+m) \times q}} = \left[\boldsymbol{I_{q-m}} \boldsymbol{0^{(q-m) \times m}}; \boldsymbol{N}; \boldsymbol{-N}\right]$. The *nullspace approach* proved to be faster than the original version, removes redundancies (by the nature of the kernel matrix), and simplifies the Gaussian elimination step.

Bit sets. Adjacency tests are the most expensive parts of the algorithm. However, as we only need to know whether or not a ray fulfills a specific inequality with equality, we can use *bit sets* to store this information. Corresponding to the *zero sets* in definition 3, the *bit set zero set* of a given vector $\boldsymbol{x}$ at iteration step j is defined as follows, complementary to [3]:

Definition 7. *For any $\boldsymbol{x} \in P_j$, P_j being the polyhedral cone at iteration step j represented by the double description pair $(\boldsymbol{R}_j, \boldsymbol{A}_j)$, the set*

$$B_j(\boldsymbol{x}) = \{r_1 r_2 \dots r_j \,|\, r_i \in [0,1], 1 \le i \le j\} \quad \textit{with} \quad r_i = \begin{cases} 1 & \textit{if } \boldsymbol{A}_i \, \boldsymbol{x} = 0 \\ 0 & \textit{otherwise} \end{cases}$$

is called the bit set representation *of the zero set of $\boldsymbol{x}$.*

We will use the shorter term *zero set* subsequently for *bit set representation of the zero set.*

The *bitwise and* operation for zero sets corresponds to the intersection of sets, because for every bit-position in the bit set, the position in the resulting set is 1 iff the position was 1 in both source sets. Accordingly, the subset (or superset) operation can be performed by:

$$B(x) \subseteq B(y) \iff B(x) \wedge B(y) \equiv B(x) \tag{3}$$

Proposition 1. *To derive the zero set of a vector at iteration $j+1$, the following operations are performed:*

$$B_{j+1}(\boldsymbol{x}) = \begin{cases} B_j(\boldsymbol{x}) + 1 & \textit{if } \boldsymbol{x} \in \boldsymbol{R}^0_{j+1} \\ B_j(\boldsymbol{x}) + 0 & \textit{if } \boldsymbol{x} \in \boldsymbol{R}^+_{j+1} \end{cases} \tag{4}$$

for extreme rays which still fulfill the new equation and are kept, and

$$B_{j+1}(\boldsymbol{x}, \boldsymbol{y}) = \{B_j(\boldsymbol{x}) \wedge B_j(\boldsymbol{y}) + 1 \,|\, \boldsymbol{x} \in \boldsymbol{R}^+_{j+1}, \, \boldsymbol{y} \in \boldsymbol{R}^-_{j+1}, \, \boldsymbol{x} \textit{ adj. to } \boldsymbol{y}\} \tag{5}$$

for newly combined rays, where + stands for concatenation, *$\wedge$ for the* bitwise and *operation.*

Proof. The proof for (4) and (5) immediately emanates from definition (3).

The bit set representation of zero sets has two main advantages: It demands little space in memory, and set operations (*bitwise and*, subset tests) for adjacency can be performed efficiently. Moreover, storing only one bit for vector elements concerning rows in $\boldsymbol{A}$ which have already been processed is sufficient. The number of zero positions in extreme rays is maximized and the combination of zeros and non-zeros is unique; thus, the original real-valued rays can be reconstructed from the bit set extreme rays after the final iteration step [3].

4 New Approaches

4.1 Bit-Set Trees

The bit sets in definition 7 can be seen as k-tuples of $[0, 1]$ values, and thus search operations on a set of bit sets coincide with queries on a collection of k-dimensional records. For this purpose, k-d-trees have been invented as a structure for storage and retrieval of multidimensional (k-dimensional) data [8].

In the context of EFM-computation, we need to test for the existence of a superset for a given bit set. For 2 adjacent rays $\boldsymbol{r}$ and $\boldsymbol{r'}$ with corresponding zero sets $B(\boldsymbol{r})$ and $B(\boldsymbol{r'})$, the combinatorial adjacency test as defined in 6(b) bars the existence of a zero set that is superset of $B(\boldsymbol{r}) \cap B(\boldsymbol{r'})$ other than $B(\boldsymbol{r})$ and $B(\boldsymbol{r'})$. This type of queries can operate on a binary k-d-tree and works similar to the *partial match queries* given in [8].

Tree construction. Given a set of bit sets (our zero sets), the algorithm returns a binary k-d-tree or *bit set tree*. The input of the algorithm is a *set* of bit sets, that is, a collection without duplicates, which conforms to the actual problem. This simplifies step 2 of the algorithm below, where the bit sets are split into two newly created leafs, and we can assure that infinite loops are avoided.

main Create a leaf node containing all bit sets and invoke `sub` with it. The returned node is the tree's root r.

sub
1. If the leaf node contains not more elements than some threshold (the maximum leaf size), return it and continue at invoker.
2. Choose some bit j that has not yet been used on prior levels. Separate the leaf's bit sets and create two new leaf nodes *zero* and *one* containing the bit sets with $bit_j = 0$ and $bit_j = 1$, respectively.
3. Recursively invoke `sub` with *zero* and *one*.
4. Create a new intermediary node i with two children *zero* and *one*, the nodes returned by `sub` in 3. Return i and continue at invoker.

Superset existence. Given the root r of a bit set tree t constructed as described above and a bit set s to be tested, where $s = s^+ \cap s^-$ with $s^+ \in t$ and $s^- \in t$, the algorithm returns *true* if a super set of s is contained in t (other than s^+ and s^-), $false$ otherwise (i.e. it returns *true* iff s^+ is adjacent to s^-). The functionality of the algorithm is illustrated in Fig. 1.

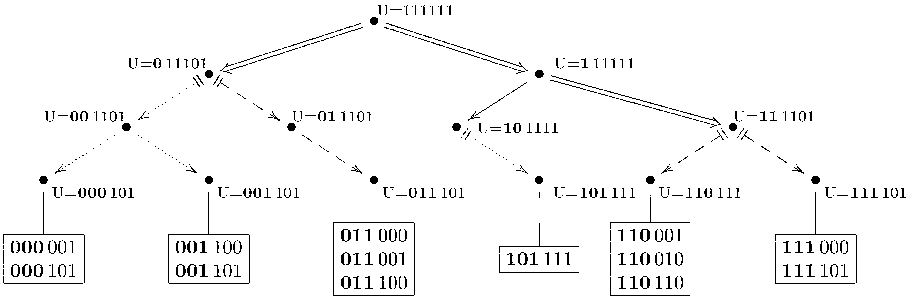

Fig. 1. Superset-Existence algorithm on a *bit-set tree/pattern set tree* with ternary leafs and a test bit set $s = 010011$. Double-lines indicate pointers to child nodes which are traversed in both tree-variants, dotted lines are traversed in neither of them. Dashed lines are only traversed in the bit-set tree, single solid lines only in the pattern-tree. Double-bar arrow-heads highlight truncation by the pattern.

main Invoke `sub` with the root node r and return the result from that call.

sub
1. If the current node is a leaf, iterate through the leaf's bit sets and return *true* if any of them is a superset of s (not being s^+ or s^-), $false$ otherwise.
2. Let s_i be the bit i of s where i is the bit position corresponding to the current node (this bit has been used to separate the bit sets in child node *zero* from those in *one*).
3. Invoke `sub` with *one*. If *true* is returned, pass it to the invoker.
4. If $s_i = 0$, call `sub` with *zero* and return the result, else return $false$.

Correctness and complexity. By the way of constructing the tree, the *zero* child of an intermediary node with selective bit j contains those bit sets that have $bit_j = 0$. Thus, if the set s to be tested contains j, that is, $bit_j = 1$, the bit sets in *zero* cannot be supersets of s and only the bit sets in *one* are superset candidates, conforming with the recursion condition in step 4.

We cannot estimate the number of intermediary modes and, thus, the overall time complexity of the DDM. However, for each step, at least $d - 1$ inequalities are fulfilled with equality, where $d = rank(\boldsymbol{A})$ (definition 5(a)). Since $\boldsymbol{A}$ contains $\boldsymbol{I}$, $d = q$ equals the number of irreversible reactions, and due to the nature of the nullspace, all equality constraints are fulfilled. They correspond to $2m$ rows in $\boldsymbol{A}$ with rank m (assuming independent rows in $\boldsymbol{N}$), thus $q - 1 - m$ positions are left to be fulfilled with equality. That is, the bit sets in t have at least $q - m - 1$ 1-bits, and due to definition 6(a) s at least $q - m - 2$ respectively. With bit set length l ($q - m \leq l \leq q$), the probabilities of a 1 in s and in the tree's bit sets can be estimated:

$$\begin{cases} n \cdot \frac{q-m-1}{l} & \text{remaining bit sets with probability} \quad \frac{q-m-2}{l} \\ n & \text{remaining bit sets with probability} \quad 1 - \frac{q-m-2}{l} \end{cases} \tag{6}$$

We assume a well balanced tree of depth $log_2(n)$ and set $\epsilon_1 = \frac{g-m-1}{l}$ and $\epsilon_2 = \frac{g-m-2}{l}$. The time complexity at step j is proportional to the number of considered bit sets per adjacency test, approximated by

$$n \cdot (1 - \epsilon_2 + \epsilon_1\epsilon_2)^{log_2(n)} = n^{1+log_2(1-\epsilon_2+\epsilon_1\epsilon_2)} \tag{7}$$

Note that the sublinear function in eq. 7 has an optimum at $\epsilon_{1/2} \approx 1/2$. It is relatively insensitive to perturbations in $\approx [0.2, 0.8]$, especially for large n. For real problems, eq. 7 is a good (and conservative) approximation.

In a well-balanced tree, we have $n/2$ nodes holding n unary leafs, requiring $c \cdot 2n$ additional memory space for a total of n intermediary nodes and n leafs, where c is a small constant. Optimizations could be applied, but these memory demands are far from being critical for our purposes. In [8], an algorithm is presented which constructs a balanced tree based on the median of a collection of elements. With binary values, this approach cannot be applied, but we can adjust the selective bit at step 2 of the tree construction. Either a *static* bit order is calculated before constructing the tree, or the most selective bit is chosen *dynamically* when the leaf's bit-sets are subdivided. We get closer to optimally balanced trees with dynamic choice, but loose the property of having the same selective bit for nodes on the same level. Here, we used *static* and *dynamic* heuristics, leaving space for subsequent explorations.

4.2 Pattern Trees

The general idea of *pattern trees* ties up to the *bit set trees*, where bit sets are separated into two child nodes in every intermediary node, taking some designated *selective bit* as criterion for partitioning. In pattern trees, additionally, all intermediary and leaf nodes account for the bit sets of their children by a *union pattern* of all bit sets contained in the subtree. At least the selective bits of the node and its predecessors are common for all bit sets in the subtree. However, since the actual bit sets constitute only a small fraction of all possible values, it is likely that other common 0's will occur in the pattern. This allows a more restrictive pre-rejection of test sets.

Proposition 2. *Let s be a set, E a collection of sets and $U = \{\bigcup e \,|\, e \in E\}$ the union of all sets in E. Then $s \subseteq U$ is a necessary condition for $\{e \,|\, s \subseteq e, e \in E\} \neq \emptyset$, i.e. that a superset of s exists in E.*

Proof. If $s \not\subseteq U$, s contains at least some $j \notin U$. Thus, for all $e \in E$, $j \notin e$ and consequently $s \not\subseteq e$ hold.

Tree construction. In addition to the algorithm for constructing *bit set trees*, we calculate the *union pattern* U when a new leaf node (containing the set E of bit sets) is created (in `main` and at step 2) as $U = (\vee e \,|\, e \in E)$ where $\vee$ stands for the *bitwise or* operation.

Superset existence. The algorithm works very similarly to that given for *bit set trees*, passing the root node r of a pattern tree. Note that no bit tests are

performed in step 4 and the recursion is always invoked with both children. Step 1 avoids descending the *zero*-subtree if the test bit of s was 1 since the pattern of the *zero*-child contains 0 at the respective bit position (Fig. 1).

main Invoke `sub` with the root node r and return the result from that call.

sub
1. If $s \not\subseteq U$, U being the *union pattern* of the node, return $false$.
2. If the node is a leaf, iterate through the leaf's bit sets and return *true* if a superset of s exists (not being s^+ or s^-), $false$ otherwise.
3. Invoke `sub` with *one*. If *true* is returned, pass it to the invoker.
4. Invoke `sub` with *zero* and return the result.

Correctness and complexity. The only point where we decide to disregard some superset candidates is at step 1, where sets are excluded if their union pattern does not fulfill the condition introduced in proposition 2 necessary for the existence of supersets in the corresponding subtree.

Both time and memory demands for pattern trees are very similar to those of bit set trees, and it is beyond the scope and objectives of this work to achieve more precise estimates for time complexity.

4.3 Narrowing Adjacent Pair Candidates

In the previous sections, we have addressed *adjacency testing*. We will now focus on narrowing the ray-pairs being candidates for adjacency even before testing.

Proposition 3. *Let E_1, E_2 and E_3 be collections of sets with corresponding union patterns $U_j = \{\bigcup e \,|\, e \in E_j, 1 \leq j \leq 3\}$, and let $S = \{s_1 \cap s_2 \,|\, s_1 \in E_1, s_2 \in E_2\}$. Then*

$$\exists s_3 : s_3 \in E_3 \text{ with } U_1 \cap U_2 \subseteq s_3 \Longrightarrow \exists s_3 : s_3 \in E_3 \text{ with } s \subseteq s_3 \tag{8}$$

holds for every $s \in S$.

Proof. By definition, $U_1 \cap U_2 = \big((s_{11} \cup \cdots \cup s_{1n}) \cap (s_{21} \cup \cdots \cup s_{2m})\big)$. Applying the distributive law, we get $\big((s_{11} \cap s_{21}) \cup (s_{11} \cap s_{22}) \cup \cdots \cup (s_{11} \cap s_{2m}) \cup \cdots \cup (s_{1n} \cap s_{2m})\big)$ being the union of all elements of S. Thus, $s \subseteq U_1 \cap U_2$, and consequently $U_1 \cap U_2 \subseteq U_3 \Longrightarrow s \subseteq U_3$. From this, eq. (8) follows by replacing U_3 by s_3. If an s_3 exists (left hand of 8), the same s_3 exists on the right hand side.

We can use eq. 8 as necessary preconditions for the all-pair combinations. The success of this shortlisting of candidates highly depends on the relations between the sets or their union patterns. Higher similarities of patterns U_1 and U_2 enhance the probability of the precondition being true, while larger sets E_1 and E_2 are more desirable since more pairs could be eliminated. *Pattern trees* comply with these requirements well since they constitute subtrees with union patterns. Nodes in the upper part of the tree have many, but barely similar entries; descending the tree means lowering the number of entries and increasing the similarity. This characteristic can be used to find the optimal balance between number and similarity of entries in a set.

Here, we implemented an algorithm that tests the *cut-pattern* $U_1 \cap U_2$ for two leaf nodes. We used heuristics to calculate the optimal leaf size, that is the number of bit sets per leaf, in a manner of *statically* balancing similarity and exclusion. Future development should also consider *dynamic* balancing by calculating the *cut pattern* for all nodes, not only leaf nodes, to reject candidates at different tree levels.

In Fig. 1, the pattern combination of the left-most leaf nodes leads to the *cut-pattern* of 000101 ($000101 \wedge 001101$), for which we find a superset 111101 in the right-most leaf node of the tree. Thus, no pair from the left-most leafs form an adjacent pair, which has been found by one test instead of four for all combinations.

Algorithmic extensions. At iteration j of the double description algorithm, we construct a *pattern tree* t_j as described above with the following extensions:

1. The collections of zero sets in the leafs of the pattern tree are divided into three subsets S^0, S^+, and S^- corresponding to the separation of the rays by the hyperplane H_j^0 at step 2a of the DD-algorithm.
2. We calculate three *union patterns* for every leaf l:

$$\begin{aligned} l.U &= \{\bigvee s \,|\, s \in l.S^0 \cup l.S^+ \cup l.S^-\} \\ l.U^+ &= \{\bigvee s \,|\, s \in l.S^+\} \\ l.U^- &= \{\bigvee s \,|\, s \in l.S^-\} \end{aligned}$$

To create the extreme rays for the next iteration step, we iterate through the tree's leafs L, and initialize $L_A = L$.

loop$_A$
1. Choose some leaf l_A from L_A and initialize $L_B = L_A$.
2. Collect the zero sets in S^0 and S^+ of l_A and apply eq. 4.

loop$_B$
- i. Pick some leaf $l_B \in L_B$ (possibly again l_A) and calculate the *cut-patterns* $\begin{cases} C^{+-} = l_A.U^+ \wedge l_B.U^- \\ C^{-+} = l_A.U^- \wedge l_B.U^+ \end{cases}$
- ii. If a superset s for C^{+-} exists in the tree with $s \notin l_A.S^+, s \notin l_B.S^-$, no pair $(s_A^+, s_B^-) \in (l_A.S^+, l_B.S^-)$ is an *adjacent* pair according to eq. 8, thus continue at (iv).
- iii. Test every pair $(s_A^+, s_B^-) \in (l_A.S^+, l_B.S^-)$ as usual, i.e. by testing the intersection $s_A^+ \wedge s_B^-$, and apply eq. 5 for adjacent pairs.
- iv. Repeat (ii) and (iii) with C^{-+} accordingly.
- v. Remove l_B from L_B and continue at `loop`$_B$ if L_B is nonempty.

3. Remove l_A from L_A and continue at `loop`$_A$ if L_A is nonempty.

5 Experimental Results

As realistic examples, we used variants of a stoichiometric model for the central metabolism of *Escherichia coli* [9]. Network compression techniques mostly

Table 1. Computation of the elementary modes for variants of the central metabolism of *Escherichia coli*. Abbreviations: Glc = glucose, Ac = acetate, Form = formiate, Eth = ethanol, Lac = lactate, CO2 = carbon dioxide, Succ = succinate, Glyc = glycerol. The products considered were Ac, Form, Eth, Lac, CO2. Relative speed figures relate to the bit set tree version. Note that due to the current implementation, the absolute time measurements are somewhat above those given in [10].

	S0		S1		S2	
substrates	Suc		Glc		Glc, Succ, Glyc, Ac	
network size	97 × 114 (28 rev.)		97 × 114 (28 rev.)		97 × 118 (28 rev.)	
compressed size	34 × 45 (16 rev.)		35 × 46 (17 rev.)		37 × 52 (17 rev.)	
iteration steps	28		29		31	
elementary modes	7,055		27,100		507,632	
	time	rel. speed	time	rel. speed	time	rel. speed
iterate all	6.6s	0.5	453s	0.1	-	-
bit set tree	3.6s	1.0	38s	1.0	451min	1.0
pattern tree	3.4s	1.1	28s	1.4	431min	1.1
candidate narrowing	2.6s	1.4	14s	2.7	28min	16.1

identical to those presented in [3] were applied. The algorithm was entirely implemented in Java and the tests were run on a Linux machine with an AMD Opteron(tm) 250 processor with 2.4 GHz, using a Java 5 virtual machine with max. 4 GB memory. The results summarized in table 1 show major improvements from the primitive combinatorial adjacency test to those with bit-set or pattern trees for small problem sizes, where adjacent candidate pair narrowing yields lower advances. With higher dimensional problems, candidate narrowing scales much better than merely improving testing and in fact becomes essential with regard to whole-cell metabolic networks.

6 Conclusions and Prospects

Determining elementary flux modes constitutes an important problem for bioinformatics. In addition, it is relevant for other domains of computer science / applied mathematics due to the nature of the underlying problem: the enumeration of all extreme rays of convex polyhedral cones. In this work, we focused on one aspect of the double description method that is critical for its performance, namely the independence tests for (preliminary) extreme rays. Conceptually, we introduced variants of k-d trees—*bit-set trees* and *pattern trees*—to implement the search for a superset of a given test set in the combinatorial adjacency test, and for effectively restricting the search scopes. Implementations and applications of the algorithms to real-world metabolic networks confirmed performance gains on the order of one magnitude compared to currently employed algorithms. In particular, refined searches using pattern trees scale well with problem size, which is important for ultimately analyzing whole-cell networks.

In perspective, further improvements are expected by exploiting pattern trees for pre-rejection of test candidate pairs in a more sophisticated manner, by adaptive methods for tree-balancing, and by employing the more efficient rank test for adjacency that does not depend on the number of modes to be tested. It is quite simple to combine rank test and candidate narrowing with pattern trees, by demanding that *cut patterns* pass the rank test. All these aspects require further efforts in theory, for instance, to determine optimal balancing schemes. Porting efficiency-sensitive code parts from Java to a high-performance language will certainly help fully realize the algorithms' potential; another relevant and attractive topic regarding applicability to larger networks is parallelization. Overall, we anticipate these approaches to finally enable enumeration of elementary modes for genome-scale metabolic networks. This, of course, still has to be proven. The subsequent interpretation of huge sets of metabolic pathways is yet another challenging and interesting problem.

Acknowledgments

We thank Gaston Gonnet for algorithmic ideas and for comments on the manuscript.

References

1. Klamt, S., Stelling, J.: Stoichiometric and constraint-based modeling. In Szallasi, Z., Stelling, J., Periwal, V., eds.: System Modeling in Cellular Biology. MIT Press (Cambridge / MA) (2006) 73–96
2. Price, N., Reed, J., Palsson, B.: Genome-scale models of microbial cells: Evaluating the consequences of constraints. Nat. Rev. Microbiol. **2** (2004) 886–897
3. Gagneur, J., Klamt, S.: Computation of elementary modes: A unifying framework and the new binary approach. BMC Bioinformatics **5** (2004) 175
4. Motzkin, T.S., Raiffa, H., Thompson, G., Thrall, R.M.: The double description method. In Kuhn, H., Tucker, A., eds.: Contributions to the Theory of Games II. Volume 8 of Annals of Math. Studies., Princeton University Press (Princeton / RI) (1953) 51–73
5. Schuster, S., Hilgetag, C.: On elementary flux modes in biochemical reaction systems at steady state. J. Biol. Syst. **2** (1994) 165–182
6. Wagner, C.: Nullspace approach to determine the elementary modes of chemical reaction systems. J. Phys. Chem. B **108** (2004) 2425–2431
7. Fukuda, K., Prodon, A.: Double description method revisited. In: Combinatorics and Computer Science. (1995) 91–111
8. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18** (1975) 509–517
9. Stelling, J., Klamt, S., Bettenbrock, K., Schuster, S., Gilles, E.: Metabolic network structure determines key aspects of functionality and regulation. Nature **420** (2002) 190–193
10. Klamt, S., Gagneur, J., von Kamp, A.: Algorithmic approaches for computing elementary modes in large biochemical reaction networks. IEE Proc. Systems Biol. **152** (2005) 249–55