# Parallel Extreme Ray and Pathway Computation

Marco Terzer and Jörg Stelling

Biosystems Science and Engineering, ETH Zürich, CH-8092 Zürich, Switzerland
{marco.terzer,joerg.stelling}@bsse.ethz.ch

**Abstract.** Pathway analysis is a powerful tool to study metabolic reaction networks under steady state conditions. An Elementary pathway constitutes a minimal set of reactions that can operate at steady state such that each reaction also proceeds in the appropriate direction. In mathematical terms, elementary pathways are the extreme rays of a polyhedral cone—the solution set of homogeneous equality and inequality constraints. Enumerating all extreme rays—given the constraints—is difficult especially if the problem is degenerate and high dimensional. We present and compare two approaches for the parallel enumeration of extreme rays, both based on the double description method. This iterative algorithm has proven efficient especially for degenerate problems, but is difficult to parallelize due to its sequential operation. The first approach parallelizes single iteration steps individually. In the second approach, we introduce a *born/die* matrix to store intermediary results, allowing for parallelization across several iteration steps. We test our multi-core implementations on a 16 core machine using large examples from combinatorics and biology.

**Availability:** The implementation is available at `http://csb.ethz.ch` in the tools section (Java binaries); sources are available from the authors upon request.

**Keywords:** Extreme ray enumeration, metabolic pathway analysis, elementary modes, double description method, born/die algorithm.

## 1 Introduction

Biochemical reaction networks are typically characterized by a matrix $\boldsymbol{S}^{m\times r}$ containing the stoichometric coefficients, where a negative (positive) entry $(i, j)$ stands for the consumption (production) of metabolite $i$ in reaction $j$; zero entries indicate that the metabolite does not participate in the reaction. Concentration changes are expressed by

$$\frac{d\boldsymbol{c}}{dt} = \boldsymbol{S}\,\boldsymbol{v} \tag{1}$$

where $\boldsymbol{v}$ represents a flux vector, assigning a flux value to each of the $r$ reactions. For many considerations, (quasi) steady state is assumed, meaning that (internal) metabolite concentrations remain constant. To comply with thermodynamical feasibility, flux values have to be nonnegative (reversible reactions

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 300–309, 2010.

are usually split into a forward and backward part). Hence, flux values are constrained as follows:

$$\boldsymbol{S}\,\boldsymbol{v} = \boldsymbol{0} \tag{2}$$

$$\boldsymbol{v} \geq \boldsymbol{0} \tag{3}$$

The resulting flux space is a (pointed) polyhedral cone. Its extreme rays are called *elementary modes*, and every possible flux vector $\boldsymbol{v}$ can be constructed as nonnegative combination of the elementary modes [1,2].

Enumerating extreme rays is computationally challenging, especially if the problem is degenerate. Polynomial algorithms exist for nondegenerate cases [3], where no point satisfies more than $d$ inequalities for a $d$-dimensional cone. Unfortunately, many practical problems are degenerate, like most biological examples, and no efficient algorithm is known [4].

Here, we introduce two approaches to parallelize the *double description method* initially invented in [5]. The standard sequential algorithm is described in section 1.1. The parallel approaches are presented in section 2, where we also derive our new *born/die* approach (2.2) and prove the correctness of the algorithm. Finally, we test our multi-core implementation with difficult problems from combinatorics and biology (2.3).

### 1.1 Double Description Method

The double description method starts with an initial cone and adds remaining constraints iteratively. Each constraint, represented by a halfspace, is intersected with the cone of the previous step (Fig. 1A). The cone is defined by equalities and nonnegativity constraints:

$$P = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{x} \geq \boldsymbol{0}\} \tag{4}$$

This form readily reflects the steady state (eq. 2) and irreversibility constraints (eq. 3) in biochemical reaction networks. Note that any other cone can be transformed into this form.

For the initial cone, we use a special form of the kernel matrix $\boldsymbol{K}$, a basis for the nullspace of $\boldsymbol{A}$. We can assume that $\boldsymbol{A}^{\,a\times d}$ has full row rank $a$, and hence we have $d-a$ columns in $\boldsymbol{K}$. Using the row-echelon form for $\boldsymbol{K}$ as proposed in [6], we get:

$$\boldsymbol{K} = [\tilde{\boldsymbol{K}}^T, \boldsymbol{I}]^T \quad \text{with} \quad \tilde{\boldsymbol{K}} \in \mathbb{R}^{a\times(d-a)} \tag{5}$$

Since $\boldsymbol{A}\,\boldsymbol{K} = \boldsymbol{0}$, and since all entries in $\boldsymbol{K}$ are nonnegative for rows $i > a$, the remaining work is to ensure that they are nonnegative for all rows. By iteratively processing the rows $i \leq a$, each column with a negative entry at row $i$ is removed at iteration $i$. New columns are derived by combining removed columns with other columns with positive entry at row $i$, such that the value at row $i$ vanishes for the new column.

It is important that columns are only added if they are extreme rays. Thus, every new column is tested for extremity, or equivalently, only adjacent ray pairs

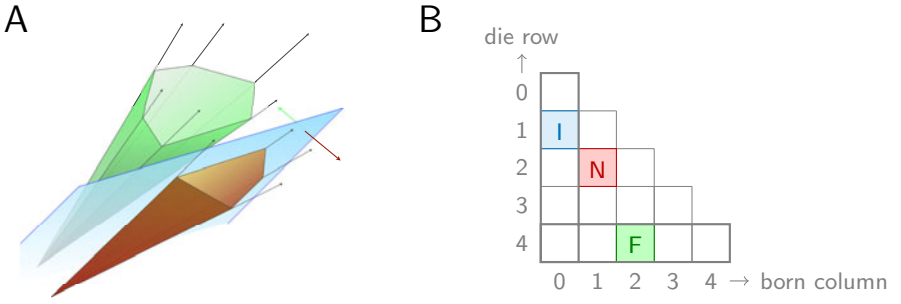


**Fig. 1. A:** Iteration step of the double description method. The added constraint represented by a halfspace cuts the preliminary cone, possibly generating new extreme rays lying in the seperating hyperplane. **B:** Born/die matrix with sample cells. The highlighted blue cell (I) contains initial rays dying at iteration $t = j + 1$ (here: $t = 2$), that is, rays $\boldsymbol{r}$ in the cell have $r_k \geq 0$ for $k < t$ and $r_t < 0$. The green cell (F) contains final rays, meaning that $r_k \geq 0\ \forall k$. Rays in the red cell (N) are born during the first iteration $i$, and they die in the third iteration $t = j + 1$.

are used to generate new columns. In fact, most of the computation time is spent for extreme ray or adjacency testing, and we presuppose an efficient method to generate new extreme rays [7,8].

## 2 Results

### 2.1 Per-step Parallelization

A first approach to exploit multi-core CPUs parallelizes each iteration step individually. When we intersect the previous cone with the halfspace, new extreme rays are generated and every ray is tested for extremity. This generation and testing process can be executed with multiple threads. We have proposed this technique in [8] and compare it here with an alternative approach applying parallelization across multiple iteration steps.

### 2.2 The Born/Die Approach

*Born/die matrix and basic idea*

At iteration $t$, intermediary extreme rays $\boldsymbol{r}$ with negative value $r_t < 0$ are removed, that is, they *die* at step $t$. New extreme rays $\boldsymbol{q}$ are created from adjacent ray pairs $(\boldsymbol{r}, \boldsymbol{s})$ with $r_t < 0$ and $s_t > 0$, that is, they are *born* at iteration $t$. We can thus assign a *born/die* index pair $(i, j)$ to each extreme ray, and for iterations $1, \ldots, n$, we get:

- $i = 0$: initial ray, a column in the initial kernel matrix $\boldsymbol{K}$
- $i \in [1, \ldots, n]$: ray born during iteration $t = i$
- $j \in [0, \ldots, (n-1)]$: ray dies during iteration $t = j + 1$
- $j = n$: ray never dies, final extreme ray, part of algorithm output

Note that $j \geq i$, since a ray can only die *after* being born. Each ray can be associated with a cell in a lower triangular *born/die matrix* (Fig. 1B). The $(n+1)$ columns and rows account for born and die indices, respectively. Column 0 contains initial, row $n$ final extreme rays. The die index $j$ of a ray $\boldsymbol{r}$ is determined as follows:

$$j = \begin{cases} n & \text{if } r_k \geq 0 \quad \forall k \in [1,\, n] \\ \min\{k : r_k < 0,\ k \in [1,\, n]\} - 1 & \text{otherwise} \end{cases} \tag{6}$$

The general idea of the algorithm exploits that every cell in the born/die matrix contains rays dying at a certain iteration step (except for cells in the last row). The matrix is filled up by pairing the dying rays with rays from other cells. The pairing jobs can be run concurrently, but we have to be careful: newly generated rays are again stored in the born/die matrix, and a pairing job should not be started before the involved cells have gathered all rays.

**Definition 1.** *The generation process of new rays* $\boldsymbol{q}$ *from rays* $\boldsymbol{r}$ *in cell* $(i,\, j)$*, dying at iteration* $j+1$*, and rays* $\boldsymbol{s}$ *adjacent to* $\boldsymbol{r}$ *from a cell* $(k,\, l)$ *with* $s_{j+1} > 0$*, is called* pairing job *or* pairing task $\theta_{(i,\,j)}\Big((k,\, l)\Big)$*. We call the cell* $(i,\, j)$ pairing cell*, and* $(k,\, l)$ partner cell *of* $\theta$*, and we say that pairing cell* $(i,\, j)$ initiates *pairing jobs* $\theta_{(i,\,j)}(\star)$*.*

*Cell stages*

Before we can use the rays of a cell, we must ensure that the cell has already collected all rays to be stored in it ((A)ccumulating stage). When the cell has collected all rays, it enters the (B)earing stage. After actively triggering pairing jobs, it is still involved in pairing jobs triggered by other bearing cells ((C)ollaborating stage). Finally, if the rays in the cell are not used anymore, they are dropped and the cell is in the (D)one stage. The cell stages (A)-(D) are summarized in the table below, and an exemplarily born/die matrix with cells at different stages is shown in Fig. 2A.

**(A)ccumulating:** The cell is collecting rays generated by pairing jobs, but is not involved in pairing jobs in a constitutive way.
**(B)earing:** The cell is initiating pairing jobs as pairing cell. It might also be involved in other pairing jobs as partner cell.
**(C)ollaborating:** The cell is involved in pairing jobs as partner cell, but has completed the initiation of pairing jobs.
**(D)one:** The cell is not involved in pairing jobs anymore. If it is not a final row cell, the rays previously stored in the cell have been dropped.

*Conditions for cell stage transitions*

**Lemma 1 (Pairing Lemma).** *Let* $t = j + 1$*, and* $\boldsymbol{r}$ *be a ray in cell* $(i,\, j)$ *with* $i \in [0,\, n-1]$*,* $j \in [0,\, n-1]$*. Then,* $\boldsymbol{r}$ *is paired with adjacent rays* $\boldsymbol{s}$ *with* $s_t > 0$ *found in cells*

$$\mathit{\Pi}_{(i,\,j)} = \{(k,\, l) : k \in [0,\, j]\ ,\ l \in [j+1,\, n]\}$$

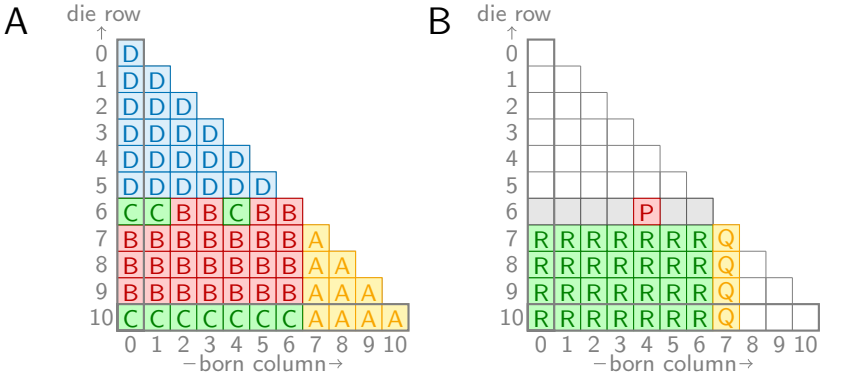


**Fig. 2. A:** Hypothetical intermediary cell stages (A)ccumulating, (B)earing, (C)ollaborating and (D)one. Collaborating cells can only occur in the final row $n$, and in the first bearing row $\rho$ (here: $\rho = 6$). Rays in cells of the final row never die, and are thus never bearing; cells in rows between $\rho$ and $n$ have pending partner cells in column $\rho + 1$, which are still accumulating. Cells above row $\rho$ are always in the Done stage, cells with column index $> \rho$ are always accumulating. **B:** Born/die matrix with an exemplarily red pairing cell (P) and yellow destination cells (Q) for pairing jobs involving P. All green cells (R) are partner cells for P. Note that any other pairing cell in row 6 has the same destination and partner cells as P.

*Proof.* From the main lemma of the double description method (see [9]), we know that at iteration $t$, rays $\boldsymbol{r}$ with $r_t < 0$ are paired with adjacent rays $\boldsymbol{s}$ with $s_t > 0$. Here, we also have $r_t < 0$ since $\boldsymbol{r}$ is stored in row $j$, that is, it dies at iteration $t = j + 1$. It remains to prove that partner rays $\boldsymbol{s}$ can only be found in the cells $\mathit{\Pi}_{(i,\,j)} = \{(k,\,l)\}$. Clearly, $k \leq j$, otherwise, the partner ray would not yet be born at iteration $t$. Furthermore, $\boldsymbol{s}$ must die *after* step $t$, that is, $l \geq j + 1$. □

**Lemma 2 (Dependency Lemma).** *Every extreme ray* $\boldsymbol{q}$ *stored in cell* $(i,\,j)$ *with* $i \in [1,\,n]$, $j \in [1,\,n]$ *has been generated from an ancestor ray in*

$$\mathit{\Psi}_{(i,\,j)} = \{(k,\,l) : k \in [0,\,i-1],\; l = i-1\}$$

*Proof.* Since $i \geq 1$, a ray $\boldsymbol{q}$ stored in cell $(i,\,j)$ must have been generated from a dying ray $\boldsymbol{r}$ with $r_t < 0$ and a surviving ray $\boldsymbol{s}$ with $s_t > 0$. The dying ancestor must be dying during the same iteration step as ray $\boldsymbol{q}$ is born, that is, at iteration $t = i$, and we have $l = i - 1$. It is not relevant when ray $\boldsymbol{s}$ was born, and hence $k \in [0,\,i-1]$. □

The dependency lemma 2 not only tells us when to start pairing jobs, but also where the new rays created from those jobs will be stored. The rays stored in cell $(i,\,j)$ are connected to ancestor cells only through column index $i$, and all ancestor cells with dying rays are contained in row $i-1$ (Fig. 2B). This leads to the following corollary:

**Corollary 1 (Destination Cells).** *Rays* $\boldsymbol{q}$ *generated from a pairing job* $\theta_{(\star,\,i)}(\star)$ *initiated by a pairing cell from row* $i$ *are placed in column* $i+1$.

Stages (B) and (C) are *active* since cells are involved in pairing jobs during those stages. The following corollary from lemma 2 yields the condition for the activation of a cell:

**Corollary 2 (Activation Condition: transition A–B).** *Cells in column* $i$ *with* $i \in [1, n]$ *have collected all rays if all pairing jobs* $\theta_{(\star,\,k)}(\star)$ *involving a pairing cell from row* $k = i-1$ *have completed.*

The sequential algorithm keeps rays which are feasible at least until the next iteration. Infeasible rays are dropped after the generation of new rays with adjacent partner rays. To the same effect, we derive a deallocation condition for rays stored in the born/die matrix:

**Proposition 1 (Deallocation Condition: transition C–D).** *Rays stored in cells of row* $j$ *with* $j \in [0, n-1]$ *can be dropped if all pairing jobs* $\theta_{(\star,\,k)}(\star)$ *initiated by a pairing cell from row* $k \leq j$ *have completed.*

*Proof.* As a precondition, the cell in row $j$ is not used as pairing cell, and thus also not as partner cell as implied by pairing lemma 1. Consequently, the rays in the cell are not used in any pairing job, and because $j < n$, they are also not final rays, that is, they can be dropped. □

To account for the last transition from (B) to (C), we count the completed pairing jobs initiated by a cell, and introduce a *remaining job counter* $\Theta$ as follows:

$$\begin{aligned}\Theta\Big((i,\,j)\Big) &= \text{remaining jobs, initiated by } (i,\,j)\\ &= |\theta_{(i,\,j)}(\star)|_{\text{total}} - |\theta_{(i,\,j)}(\star)|_{\text{completed}}\\ &= (j+1)(n-j) \;- |\theta_{(i,\,j)}(\star)|_{\text{completed}}\end{aligned} \tag{7}$$

The counter $\Theta\Big((i,\,j)\Big)$ is initialized with $(j+1)(n-j)$ according to pairing lemma 1, and decremented whenever a pairing job $\theta_{(i,\,j)}(\star)$ terminates. We can also derive the activation and deallocation conditions from $\Theta$ by computing the minimum index $\rho$ of a row with at least one cell with remaining jobs:

$$\rho = \begin{cases} n & \text{if} \quad \Theta\Big((i,\,j)\Big) = 0 \quad \forall\, i,j \\ \min\{j : \exists i : \Theta\Big((i,\,j)\Big) > 0\} & \text{otherwise} \end{cases} \tag{8}$$

Our observations are summarized in Table 1.

*Implementation aspects*

For the concurrent execution of pairing jobs, operations modifying the cell counter $\Theta$ must be thread-safe. In our Java implementation, we use variables from

**Table 1.** Conditions for cell stages and algorithm termination

| *Object* | *Stage* | *Condition* |
|---|---|---|
| $(i, j)$ | (A)ccumulating | $i > \rho$ |
| $(i, j)$ | active, (B) or (C) | $i \leq \rho \leq j$ |
| $(i, j)$ | (B)earing | $(i, j)$ is active and $\Theta\Big((i, j)\Big) > 0$ |
| $(i, j)$ | (C)ollaborating | $(i, j)$ is active and $\Theta\Big((i, j)\Big) = 0$ |
| $(i, j)$ | (D)one | $j < \rho$ |
| algorithm | terminates | $\rho = n$ |

the atomic package of the standard library. The classes `AtomicInteger` and `AtomicIntegerArray` offer atomic, thread-safe and lock-free `decrementAndGet` and `compareAndSet` operations. An additional helper variable $\Omega(j)$ for $j \in [0, n-1]$ is initialized with $j + 1$, accounting for the number of cells in row $j$ which are still (or not yet) bearing. The implementation of the conditions in Table 1 is illustrated in pseudo-code procedure `PairingJobTermination`. In addition to the state variables $\Theta\Big((i, j)\Big), \Omega(j)$ and $\rho$ used to control cell stage conditions and algorithm termination, also adding rays to an accumulating matrix cell and job queue operations must be thread-safe.

**Procedure** `PairingJobTermination`

```
desc: Called before terminating a concurrently executed pairing job θ_(i,j)(⋆)
begin
    jobsLeft ← decrementAndGet Θ((i, j))
    if (jobsLeft = 0) then
        cellsLeft ← decrementAndGet Ω(j)
        while (cellsLeft = 0) do      /* no bearing cells left in row j  */
            if (ρ ← compareAndSet ρ = j, j + 1) then      /* j was ρ  */
                if (ρ = n) then
                    terminate;
                else
                    queue new pairing jobs involving a cell of the activated
                    column ρ
                    j ← j + 1
                    cellsLeft ← Ω(j)
                end
            else
                cellsLeft ← −1            /* j was not ρ ⇒ break loop */
            end
        end
    end
end
```

We have now all elements for the born/die algorithm:

1. Initialize: $\Theta\Big((i,\, j)\Big) = (j+1)(n-j)$ and $\Omega(j) = j+1$
2. Add rays from the initial kernel matrix $\boldsymbol{K}$ to column 0 in the born/die matrix.
3. Setup an empty job queue.
4. Start concurrent working threads, processing queued jobs if available.
5. Activate column 0 by setting $\rho = 0$ and queue new pairing jobs involving only cells from column 0. All further pairing jobs will be added to the queue by terminating jobs as illustrated in procedure `PairingJobTermination`.
6. Await termination condition $\rho = n$.
7. Terminate working threads and return resulting extreme rays stored in matrix row $n$.

### 2.3 Experimental Results

We have tested the two approaches on five large problems from combinatorics and systems biology. Two examples are part of the `cddlib` sample files [10]: `ccp7` (116,764 facets of a 0/1 polytope in 21 dimensions) and `mit71-61` (3,149,579 vertices, 60 dimensions). Three examples concern elementary modes (EMs) of central metabolism of *E.coli*, namely `coli-S2` (507,632 EMs corresponding to extreme rays of a polyhedral cone in 64 dimensions, configuration S2 in [8]), `coli-S3` (2,450,787 EMs, 68 dimensions) and `coli-L1` (26,381,168 EMs, 76 dimensions).

The born/die approach seems to have a larger overhead especially with few threads (Table 2), which is not surprising since the work is split into many (possibly fine-grained) pairing jobs. The scale-up behaviour of the methods is comparable, and for some examples, both algorithms scale poorly beyond 8 threads.

**Table 2.** Computation times for *per-step* and *born/die* approach with 1,2,4,8 and 16 threads (arbitrary precision integer arithmetic for `ccp7` and `mit71-61`, double precision arithmetic for `coli` examples, adjacency test with rank updating and modulo arithmetic), and sequential single-threaded times for `cddlib` (V0.94f with double arithmetic, [10]) and `4ti2` (V1.3.2 with 64 bit integers, [11]). The tests are run on a 64bit linux 2.6.18 machine with 4 Intel Xeon X7350 Quad Core CPUs with 2.93GHz. We used a Sun Java 64-Bit Server VM (1.6.0_15-b03) with at most 8GB memory, except for coli-L1 (32GB). All times (in seconds unless otherwise indicated) are the mean of the median 6 measurements of 10 runs, except for `cddlib` and `4ti2`, where the timing is taken from a single test run. Supplementary information for all tests is available at `http://csb.ethz.ch` in the publications section.

| *Problem* | *Per-step* | | | | | *Born/die* | | | | | *cddlib* | *4ti2* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Threads* | *1* | *2* | *4* | *8* | *16* | *1* | *2* | *4* | *8* | *16* | *(1)*[1] | *(1)*[1] |
| ccp7 | 806 | 418 | 253 | 151 | 97 | 1,278 | 626 | 313 | 178 | 138 | 22,652 | 525 |
| mit71-61 | 481 | 270 | 158 | 113 | 116 | 539 | 294 | 158 | 116 | 117 | >4d | 13,253 |
| coli-S2 | 38 | 22 | 15 | 14 | 15 | 60 | 29 | 17 | 16 | 14 | – | – |
| coli-S3 | 338 | 199 | 144 | 121 | 115 | 579 | 319 | 187 | 145 | 138 | – | – |
| coli-L1 | 5,204 | 3,229 | 2,359 | 1,884 | 1,925 | 10,291 | 5,146 | 2,989 | 2,496 | 2,448 | – | – |

On the other hand, we have solid speedup for `ccp7`, indicating that scalability depends strongly on the problem instance. Both variants also compete well with the sequential implementations `cddlib` [10] and `4ti2` [11], even if a comparison is difficult[1]. The systems biology examples did not terminate with `cddlib` and `4ti2`, possibly due to missing pre-processing e.g. to compress input matrices [7].

## 3 Conclusions

We have presented and compared two different approaches to parallelize the inherently sequential double description method. In a per-step approach, we apply parallelization to the individual iteration steps. The second and novel born/die algorithm stores intermediary results in matrix cells, which enables for a cross-step parallelization. Numerous pairing jobs can be run concurrently if certain dependency conditions are considered. Both approaches compete well with alternative implementations if run on a single-core system, and scale comparably well on multi-core architectures. The multi-threaded implementation runs within the same virtual machine. However, we are confident that the born/die approach is also a promising candidate for distributed computation – in particular through its sophisticated storage model for intermediary rays.

## Acknowledgment

This work was carried out under the HPC-EUROPA project (contract No. RII3-CT-2003-506079). It was hosted by the Centrum Wiskunde & Informatica (CWI) and supported by SARA Computing and Networking Services, Amsterdam. We thank the scientific hosts, F.J. Bruggeman and L. Stougie.

## References

1. Schuster, S., Hilgetag, C.: On elementary flux modes in biochemical reaction systems at steady state. J. Biol. Syst. 2, 165–182 (1994)
2. Stelling, J., Klamt, S., Bettenbrock, K., Schuster, S., Gilles, E.: Metabolic network structure determines key aspects of functionality and regulation. Nature 420, 190–193 (2002)
3. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Appl. Math. 65(1-3), 21–46 (1996)
4. Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V.: Generating all vertices of a polyhedron is hard. Discrete Comput. Geom. 39(1), 174–190 (2008)
5. 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. Annals of Math. Studies, vol. 8, pp. 51–73. Princeton University Press, Princeton (1953)
6. Wagner, C.: Nullspace approach to determine the elementary modes of chemical reaction systems. J. Phys. Chem. B 108, 2425–2431 (2004)

[1] Computation times are not directly comparable, because (i) the sequential tools use only one thread, and since (ii) internal pre-processing steps and (iii) input ordering have a significant impact on the running time. In particular (ii) and (iii) affect the order of the constraint processing and hence the number of intermediary rays.

7. Gagneur, J., Klamt, S.: Computation of elementary modes: A unifying framework and the new binary approach. BMC Bioinformatics 5, 175 (2004)
8. Terzer, M., Stelling, J.: Large scale computation of elementary flux modes with bit pattern trees. Bioinformatics 24(19), 2229–2235 (2008)
9. Fukuda, K., Prodon, A.: Double description method revisited. In: Combinatorics and Computer Science, pp. 91–111 (1995)
10. Fukuda, K.: cddlib-094, C–library for polyhedral computations, `http://www.ifor.math.ethz.ch/~fukuda/cdd_home/index.html`
11. 4ti2 team: 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces, `www.4ti2.de`