Parallel Out-of-core Algorithm for Genome-Scale Enumeration of Metabolic Systemic Pathways*

Nagiza F. Samatova[1], Al Geist[1], George Ostrouchov[1], and Anatoli V. Melechko[2]

*[1]Computer Science and Mathematics Division, Oak Ridge National Laboratory**,*
P.O. Box 2008, Oak Ridge, TN 37831, {samatovan, gst, ost}@ornl.gov
[2]Center for Environmental Biotechnology, University of Tennessee, Knoxville, TN 37996

Abstract

Systemic pathways-oriented approaches to analysis of metabolic networks are effective for small networks but are computationally infeasible for genome scale networks. Current computational approaches to this analysis are based on the mathematical principles of convex analysis. The enumeration of a complete set of "systemically independent" metabolic pathways is at the core of these approaches and it is computationally the most demanding component. An efficient parallel out-of-core algorithm for generating a complete set of systemically independent metabolic pathways, termed "extreme pathways", is presented. These pathways represent the edges of a high-dimensional convex cone and can be used to derive any admissible steady-state flux distribution (or phenotype) for a specified metabolic genotype. The algorithm can be used for computing "elementary flux modes" that are different but closely related to extreme pathways. The algorithm combines a bitmap data representation, search space reduction, and out-of-core implementation to improve CPU-time and memory requirements by several orders of magnitude. Augmented with a parallel implementation, the algorithm provides extremely scalable performance. No previous parallel and/or out-of-core algorithms for the enumeration of systemically defined metabolic pathways are known.

Keywords: metabolic pathways, parallel out-of-core algorithms, convex analysis, metabolic modeling, extreme pathways, and elementary modes

1. Introduction

The elucidation of genome-scale metabolic networks (Karp *et. al*, 1996; Selkov *et. al*, 1997, 1998; Overbeek *et. al*, 2000) necessitates the development of more efficient and effective methods for the analysis of their integrated properties and the comparison of these properties amongst different organisms. Systemic pathways oriented approaches (Fell, 1990; Mavrovouniotis *et. al*, 1990; Schuster *et. al*, 1994, 1999, 2000; Schilling *et. al*, 1998, 1999, 2000.a, 2000.b, 2000.c) centered around properties of metabolic networks at steady state imposed by their stoichiometric structure have been successfully applied to various metabolic systems in defining their limitations and production capabilities (Schuster *et. al*, 2000; Schilling *et. al*, 2000.c). Various algorithms and mathematical tools have been developed (Pfeiffer *et. al*, 1999; Schuster *et. al*, 1999; Schilling *et. al*, 2000.a). However, while efficient in analyzing reactions

* This work has been supported by the MICS Division of the U.S. Department of Energy
** Oak Ridge National Laboratory is managed by UT-Battelle for the LLC U.S. D.O.E. under Contract No. DE-AC05-00OR22725.

systems of small size, the computational complexity of these algorithms limits their practical applicability to organism-scale metabolic networks (Schilling *et. al*, 2000.b, 2000.c), especially in the context of comparative analysis of metabolic networks among different organisms.

In recent years, two related systemic pathways oriented approaches are advocated for defining and comprehensively describing all metabolic routes (phenotypes) that are both stoichiometrically and thermodynamically feasible for a given metabolic genotype. Both of them share a common underlying mathematical framework capitalized on the principles of convex analysis. These principles include closely related concepts of "elementary flux modes" (Schuster *et. al*, 1994, 1999, 2000; Pfeiffer *et. al*, 1999) and convex basis (Nozicka *et. al*, 1974), or "extreme pathways" (Schilling *et. al*, 2000.a, 2000.c) (for a recent review, see Schilling *et. al*, 1999).

Both elementary modes and extreme pathways are systemically independent flux vectors that lie in the null space of the stoichiometric matrix. When non-negativity constraints are imposed on flux vectors, this null space takes the shape of a convex polyhedral cone, called steady-state flux cone. An extreme pathway represents an edge of this steady-state flux cone whereas an elementary flux mode is a steady-state flux vector that cannot be decomposed into two flux vectors that would have additional zero components. Any vector within the cone can be represented as a nonnegative linear combination of extreme pathways (elementary modes). The set of elementary modes (extreme pathways) is unique. The extreme pathways (elementary modes) are systemically independent because none of them can be written as a nonnegative linear combination of the others. For a more detailed explanation of their similarities and dissimilarities refer to Pfeiffer *et. al,* 1999 and Schilling *et.al.*, 2000.a).

An algorithm for constructing a convex basis (or a complete set of extreme pathways) has been developed (Nozicka *et. al*, 1974; Schuster *et. al*, 1993; Schilling *et. al*, 2000.a). An algorithm for computing both convex basis and elementary modes has been implemented as part of METATOOL available at ftp://bmsdarwin.brookes.ac.uk/pub/software/ibmpc/metatool (Pfeiffer *et. al*, 1999). For simplicity's sake, we will focus on the common procedure of enumerating these pathways that is the core of both algorithms. It is computationally the most expensive. Without loss of generality, our discussion will be presented in the context of extreme pathways in what follows.

For small networks, generating the set of extreme pathways is simple. However, for genome-scale networks, the calculation of extreme pathways poses a significant computational challenge. The computational time as well as the number of extreme pathways grows exponentially as the size of the metabolic network grows linearly. The problem of enumerating the extreme pathways can be reduced in polynomial time to the problem of enumerating all vertices of an n-dimensional convex polyhedron that is known to belong to the class of NP problems [Garey *et. al*, 1979]. Currently, this bottleneck of computational intractability has been addressed by either considering the reduced reaction network (with the enzyme subsets taken as combined reactions, or monofunctional "super-enzymes" (Kholodenko *et. al*, 1995; Rohwer *et. al*, 1996; Pfeiffer *et. al*, 1999) or decomposing the network into computationally feasible subsystems to generate pathways in each subsystem that may be pieced together (Schilling *et. al*, 2000.c).

In this paper, we present a parallel out-of-core algorithm for the enumeration of metabolic systemic pathways. The algorithm combines a bitmap data representation, search space reduction, and out-of-core implementation to improve CPU-time and memory requirements by several orders of magnitude. Augmented with a parallel implementation, the algorithm results in extremely scalable performance. No previous parallel and/or out-of-core algorithms for the enumeration of systemically defined metabolic pathways are known.

2. Metabolic Extreme Pathways within the Context of Convex Analysis

This section briefly describes the underlying mathematical theory for the systemic definition of metabolic extreme pathways presented by Shilling *et. al*, 2000.a.

To study structural and functional properties of metabolic networks, the metabolic system is often placed into a steady-state:

$$\mathbf{S} \cdot \mathbf{v} = 0 \quad (1)$$

where $\mathbf{S}$ is a $m \times n$ *stoichiometric matrix* of m metabolites (rows) and n reactions (columns) and $\mathbf{v}$ is a vector of reaction rates, called the *flux vector*. The stoichiometric coefficient S_{ij} corresponds to the number of moles of metabolite i produced (or consumed) in reaction j. The complete set of flux vectors satisfying the homogeneous system of linear equations (1) lies in the null space of $\mathbf{S}$ (Golub *et. al*, 1996) spanned by basis vectors that can be calculated by standard linear algebra methods, e.g. the Gaussian elimination algorithm (Golub *et. al*, 1996). Note that the basis vectors are not unique but their number equals the dimensionality of the null space, which is $n - rank(\mathbf{S})$. In many cases, only biochemically meaningful null space vectors are of interest.

Thus, solutions to a system of equations (1) have additional constraints imposed by principles of thermodynamics and by systemic (input/output) characteristics of the network. In particular, all reversible internal reactions are split into forward and reverse fluxes that are constrained to be non-negative. Exchange fluxes (with respect to hypothetical boundaries of the system under study) or pseudoreactions (Clarke, 1981) are left unconstrained or constrained to some lower and upper limits depending on the ability of corresponding metabolites to enter or exit the system. (For more details see Shilling *et al*, 2000.a, 2000.b). The constraints on internal ($\mathbf{v}$) and exchange ($\mathbf{b}$) fluxes can be expressed mathematically as:

$$\mathbf{v}_i \geq 0, \ \forall i \text{ and } \alpha_j \leq \mathbf{b}_j \leq \beta_j \ \ \forall j \quad (2)$$

If exchange fluxes are unconstrained, then equations (1) and (2) are reduced to a system of homogeneous linear equations and inequalities:

$$\mathbf{S} \cdot \mathbf{v} = 0, \ \mathbf{v} \geq 0 \quad (3)$$

From convex analysis (Rockafellar, 1970), all solutions to system (3) of equations and inequalities form a convex polyhedral cone. Every point within this cone $\boldsymbol{K}$ can be represented by a non-negative linear combination of the extreme rays $\mathbf{p}_i$:

$$\boldsymbol{K} = \{\mathbf{v} \in R^n \mid \mathbf{v} = \sum_{i=1}^{k} w_i \mathbf{p}_i, w_i \geq 0, \forall i\} \quad (4)$$

where k denotes the number of extreme rays. In the context of metabolic systems, these extreme rays are called *extreme pathways* as each extreme ray corresponds to a particular pathway in a reaction network. The set of extreme pathways is *conically* or *systemically independent*. This means that none of the extreme pathways can be formed as a non-negative linear combination of the other extreme pathways. Based on equation (4) and systemic independence, this set of extreme pathways is referred to as the *conical basis* of the convex space (3). From a network function perspective, this means that any attainable steady-state flux vector (or phenotype) allowable by the constraints placed on the metabolic system can be reached by controlling (switching on/off) the activity levels of the extreme pathways for a defined metabolic genotype. Unlike a linear basis of the null space, the conical basis is unique up to scalar multipliers. Moreover, the number of extreme pathways is usually much greater than the dimensionality of the cone.

If exchange fluxes are constrained, then the inhomogeneous system of inequalities (2) can be converted to an equivalent homogeneous system in the form of (3) by introducing auxiliary flux variables and applying simple algebraic operations. Thus, without loss of generality we may assume a homogeneous system of equations and inequalities in what follows.

3. The Extreme Pathways (EP) Algorithm

This section presents the core of the serial version of the algorithm for calculating the complete set of extreme pathways for a reaction network (Schilling *et. al.* 2000.a). The procedure is based on the principles of finding extreme solutions to a homogeneous system of equations and inequalities developed in convex analysis (Nožička *et. al*, 1974) and further extended to an inhomogeneous system (Schuster

et.al, 1993). The underlying convex cone is similar to that encountered in linear programming problems that can be solved by the simplex algorithm. As a result, some terminology, such as the tableau, is similar to the simplex algorithm. However, where the simplex algorithm traverses a few edges of this cone with a fixed size tableau to get to an optimal solution, the extreme pathways algorithm enumerates all the edges with a growing tableau.

Our description of EP algorithm mostly follows Schuster *et. al*, 1993. The algorithm begins with the construction of an initial tableau $\mathbf{T}^{(0)}$ containing the transpose of an $m \times n$ stoichiometric matrix $\mathbf{S}$ augmented on the left by an $n \times n$ identity matrix $\mathbf{I}$:

$$\mathbf{T}^{(0)} = (\mathbf{I}\,\mathbf{S}^{\mathrm{T}}) \tag{5}$$

The consecutive tableaux $\mathbf{T}^{(1)}$, $\mathbf{T}^{(2)}$,…, $\mathbf{T}^{(m)}$ are constructed as follows. $\mathbf{T}^{(j+1)}$ is obtained from $\mathbf{T}^{(j)}$ by a series of steps including:

1) Selecting a pivot column of the right-hand side of $\mathbf{T}^{(j)}$ (originating from the transposed stoichiometry matrix). This is the column that will be zeroed out during the j^{th} iteration. The choice of a pivot column may affect the performance of the algorithm. However, a selection strategy based on sparse matrix computation ideas is beyond the scope of this paper. For simplicity, the $(n+j+1)^{\text{th}}$ column for the j^{th} iteration is chosen in this paper.
2) Copying all rows containing a zero in the pivot column to the next tableau, $\mathbf{T}^{(j+1)}$.
3) Taking all possible positive linear combinations of pairs of rows with an opposite sign pivot column element so that the combination produces a zero in the pivot column. This is somewhat like Gaussian elimination restricted to positive linear combinations. For example, given a pair of rows, $\mathbf{r}_i^{(j)}$ and $\mathbf{r}_k^{(j)}$, of opposite sign in the pivot column of $\mathbf{T}^{(j)}$, the new row $\mathbf{r}^{(j+1)}$ is generated as:
$$\mathbf{r}^{(j+1)} = \left|\mathbf{r}_{k,n+j+1}^{(j)}\right| \cdot \mathbf{r}_i^{(j)} + \left|\mathbf{r}_{i,n+j+1}^{(j)}\right| \cdot \mathbf{r}_k^{(j)} \tag{6}$$
where $\mathbf{r}_{k,n+j+1}^{(j)}$ and $\mathbf{r}_{i,n+j+1}^{(j)}$ are the $(n+j+1)^{\text{th}}$ elements of the corresponding rows, $\mathbf{r}_k^{(j)}$ and $\mathbf{r}_i^{(j)}$.
4) Transferring each row $\mathbf{r}^{(j+1)}$ obtained in the previous step to the next tableau, $\mathbf{T}^{(j+1)}$, if the following *conical independence constraint* is satisfied:
$$Z(\mathbf{r}_i^{(j)}) \cap Z(\mathbf{r}_k^{(j)}) \not\subseteq Z(\mathbf{r}_l^{(j)}) \text{ for all } l \neq i,k \tag{7}$$
where the set $Z(\mathbf{r}_q^{(j)})$ contains all the column indices, c, for which the elements of row q of the left-hand side part of $\mathbf{T}^{(j)}$ (originating from the identity matrix) equal zero:
$$Z(\mathbf{r}_q^{(j)}) = \{c : c \leq n, \mathbf{r}_{q,c}^{(j)} = 0\} \tag{8}$$

Note that the number of rows may increase, decrease, or remain the same with each iteration. The number of rows in the final tableau, $\mathbf{T}^{(m)}$, corresponds to the number of extreme pathways. The left-hand side of $\mathbf{T}^{(m)}$ formed by the first n columns contains all the extreme pathways for system (3). A pseudo-code for this algorithm is given in Figure 1.

Inspection of the pseudo-code in Figure 1 shows that there can be two parts in the optimization of the performance of the EP algorithm: one in the outer **for** loop (line 1) when the next pivot column is selected (*global optimization*), and the other in the step of zeroing out the selected pivot column (lines 2-12), when the conical independence condition (7) is checked for each pair of rows of opposite signs (*local optimization*). Our experiments with different stoichiometric matrices show that the overall computation time can change drastically depending on the column order of the stoichiometric matrix. Thus, the global optimization can be achieved by using the optimized routine for zeroing the pivot column and by substituting the right-hand side of $\mathbf{T}^{(0)}$ (equation (5)) with the $\mathbf{S}^{\mathrm{T}} \cdot \mathbf{P}$ matrix where $\mathbf{P}$ is a permutation matrix. For the purpose of this paper, we assume that the permutation matrix $\mathbf{P}$ is given and focus on the local optimization part only. Since the cumulative cost of checking for conical independence (condition 7)

is the most time consuming operation (about 99% of the total execution time based on profiling results obtained by a *gprof* UNIX utility), a speedup can be achieved by minimizing the number of rows to be checked per each combined pair of rows (lines 6-10) (see "Search space reduction" section) as well as by improving the efficiency of an individual check (line 7) (see "Bitmap data representation" section). The next section describes the details of the local optimization part in the improvement of the EP algorithm's performance as well as presents a parallel out-of-core version of the locally optimized EP algorithm.

4. Parallel Out-of-core Computation Model

To improve the performance of the EP algorithm in Figure 1 we have developed a parallel out-of-core version of this algorithm. Our strategy is to transform a large problem into a set of small sub-problems and to perform these sub-problems almost concurrently with reasonable data transfers, latency, and synchronization so that the cumulative computational cost is much less than the cost of the aggregate problem. The key idea of the algorithm is based on:

1) The reduction of memory requirements via:
   - Using a bitmap data representation scheme with a high data compression rate;
   - Deploying an out-of-core strategy;
2) The reduction of computational time via:
   - Performing efficient bitwise logical operations without decompression overheads;
   - Minimizing search space to check for conical independence by maintaining descriptive statistics about data with cost effective updates;
   - Storing all critical information in memory thus minimizing I/O access;
   - Maintaining almost even load balance between processors;
3) The reduction of communication cost via:
   - Having one-time synchronization per long-running iteration followed by initialization of some minimal global information;
   - Partitioning the data to minimize data transfer needs;

4.1. Bitmap data representation

The concepts of *Row Feature* and *RF* hash table are at the core of our algorithm. *Row Feature* is a triple summarizing the information about a row vector in the left part of the tableau.

Definition 1. Given a row vector $\mathbf{r} = (r_1, r_2, ..., r_n)$, the *Row Feature* (*RF*) vector of $\mathbf{r}$ is defined as a triple $RF(\mathbf{r}) = (B(\mathbf{r}), First(\mathbf{r}), Last(\mathbf{r}))$ where

1) $B(\mathbf{r})$ is a non-negative integer number whose binary (base 2) representation gives the non-zero structure of $\mathbf{r}$. More formally $B(\mathbf{r})$ is defined as:

$$B(\mathbf{r}) = \{ j \in \mathbf{N} : \; j = \sum_{i=0}^{n-1} 2^i \cdot \chi(r_{n-i}) \text{ and } \chi(r_i) = \begin{cases} 1, \text{if } r_i \neq 0 \\ 0, \text{if } r_i = 0 \end{cases} \} \tag{9}$$

2) $First(\mathbf{r})$ is the index of the first non-zero component of $\mathbf{r}$ defined as:

$$First(\mathbf{r}) = \{ j : \; r_j \neq 0 \text{ and } r_i = 0 \text{ for all } i < j \} \tag{10}$$

3) $Last(\mathbf{r})$ is the index of the last non-zero component of $\mathbf{r}$ defined as:

$$Last(\mathbf{r}) = \{ j : \; r_j \neq 0 \text{ and } r_i = 0 \text{ for all } i > j \} \tag{11}$$

Theorem 1 (RF Additivity Theorem). *Assume that* $RF(\mathbf{r}_1) = (B(\mathbf{r}_1), First(\mathbf{r}_1), Last(\mathbf{r}_1))$ and $RF(\mathbf{r}_2) = (B(\mathbf{r}_2), First(\mathbf{r}_2), Last(\mathbf{r}_2))$ *are the RF vectors of two rows. Then the RF vector of the row that is formed by combining the two rows as in equation (6), is:*

$$RF(\mathbf{r}_1) + RF(\mathbf{r}_2) = (B(\mathbf{r}_1) \,|\, B(\mathbf{r}_2), \min\{First(\mathbf{r}_1), First(\mathbf{r}_2)\}, \max\{Last(\mathbf{r}_1), Last(\mathbf{r}_2)\}) \quad (12)$$

Proof. The proof consists of simple algebra. ❑

From the *RF* definition and the additivity theorem we know, that the *RF* vectors need to be computed only for the initial tableau, $\mathbf{T}^{(0)}$. For the consecutive matrices $\mathbf{T}^{(1)}$, $\mathbf{T}^{(2)}$,…, $\mathbf{T}^{(m)}$, they are updated as rows are combined. Thus, we do not need to store the entire tableau, but only the *RF* vectors of its rows as summary. This *RF* summary not only takes much less space ($p(n+m)$ double precision numbers where p is the current number of rows in the tableau vs. $3p$ integer numbers) but it is much more efficient for checking the conical independence condition (7) due to efficient bitwise logical operations as shown below.

Definition 2. Given three row vectors $\mathbf{r}_1, \mathbf{r}_2$, and $\mathbf{r}_3$, their *characteristic function F* is defined as a logical function:

$$F(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) = !(B(\mathbf{r}_1) \,|\, B(\mathbf{r}_2)) \,\&\, B(\mathbf{r}_3) \quad (13)$$

Theorem 2 (Conical Independence Theorem). *For a pair of rows, $\mathbf{r}_i^{(j)}$ and $\mathbf{r}_k^{(j)}$, of the tableau $\mathbf{T}^{(j)}$, the conical independence constraint (7) holds if and only if the characteristic function F of $\mathbf{r}_i^{(j)}$, $\mathbf{r}_k^{(j)}$, and $\mathbf{r}_l^{(j)}$ (see (13)) is true for all $l \neq i,k$, i.e.:*

$$Z(\mathbf{r}_i^{(j)}) \cap Z(\mathbf{r}_k^{(j)}) \not\subseteq Z(\mathbf{r}_l^{(j)}) \text{ for all } l \neq i,k \;\Leftrightarrow F(\mathbf{r}_i^{(j)}, \mathbf{r}_k^{(j)}, \mathbf{r}_l^{(j)}) \equiv true \text{ for all } l \neq i,k \quad (14)$$

Proof. The proof follows from the principles of Boolean algebra and definition (8) of $Z(\mathbf{r}_q^{(j)})$. ❑

Note that there is a problem in a computer implementation of condition (14) due to hardware limitations on the largest integer. For example, if the number of reactions *n* is greater than 64 than on a 64-bit hardware architecture an integer overflow exception will be generated. To fix this problem, we partition the *n* bits of the row vector **r** into a minimal number, *s*, of consecutive segments so that each segment contains as many bits as is supported by the hardware architecture. Then a generalized form of the conical independence theorem holds.

Theorem 3 (Generalized Conical Independence Theorem). *Let a row vector **r** be partitioned into the minimum number, s, of consecutive segments: $\mathbf{r} = (\mathbf{r}(1), \mathbf{r}(2), ..., \mathbf{r}(s))$, where **r**(i) is a vector of length b in a b-bit hardware architecture. Then for a pair of rows, $\mathbf{r}_i^{(j)}$ and $\mathbf{r}_k^{(j)}$, of the tableau $\mathbf{T}^{(j)}$, the conical independence constraint (7) holds if and only if the characteristic function F of $\mathbf{r}_i^{(j)}(p)$, $\mathbf{r}_k^{(j)}(p)$, and $\mathbf{r}_l^{(j)}(p)$ (see (13)) is true for all p=1,…,s and $l \neq i,k$, i.e.:*

$$Z(\mathbf{r}_i^{(j)}) \cap Z(\mathbf{r}_k^{(j)}) \not\subseteq Z(\mathbf{r}_l^{(j)}) \text{ for all } l \neq i,k \;\Leftrightarrow F(\mathbf{r}_i^{(j)}(p), \mathbf{r}_k^{(j)}(p), \mathbf{r}_l^{(j)}(p)) \equiv true \text{ for all } p \text{ and } l \neq i,k \quad (15)$$

Proof. The proof follows from the properties of Boolean functions and theorem 2. ❑

4.2. Search Space Reduction

Definition 3. The *RF* hash table *H* of tableau **T** is defined as:

$$H[first][last] = \{i : First(\mathbf{r}_i) = first \text{ and } Last(\mathbf{r}_i) = last\} \quad (16)$$

Thus, the hash table *H* defines a partition of the rows of **T** such that two rows are in the same partition if their *RF* vectors have the same *First*() and *Last*() values. From such a partition of tableau **T** it follows that for any pair of rows, it is sufficient to check the conical independence constraint (7) only for the subset of rows defined by the *First*() and *Last*() values of their combination (6). This is summarized by the next theorem.

Theorem 4 (Reduced Conical Independence Theorem). *For a pair of rows,* $\mathbf{r}_i^{(j)}$ *and* $\mathbf{r}_k^{(j)}$, *of the tableau* $\mathbf{T}^{(j)}$, *the conical independence constraint (7) holds if and only if the characteristic function F of* $\mathbf{r}_i^{(j)}$, $\mathbf{r}_k^{(j)}$, and $\mathbf{r}_l^{(j)}$ *(see (13)) is true for all* $l \in H[p][q]$ *where* $p \geq First(\mathbf{r}^{j+1})$ and $q \leq Last(\mathbf{r}^{j+1})$ *provided* $\mathbf{r}^{(j+1)}$ *is given by equation (6).*

Proof. The proof consists of showing that conical independence holds for all other *p* and *q* combinations. ❑

4.3. Out-of-core Computation Model

The exponential (for the entire program run) and quadratic (for a single iteration) space complexity of the EP algorithm (see "Complexity Analysis" section) results in huge data structures that cannot fit into memory. This necessitates "out-of-core" calculations where data are stored on disk and repeatedly brought into memory a section at a time, processed, and written to disk. Because disk access is so much slower than main memory access, efficient out-of-core methods focus on reducing disk I/O costs. Disk I/O costs are typically reduced in three ways: reduce the cost of each access (e.g., by using parallel disk systems (Bordawekar *et. al*, 1995), reduce the number of accesses, or reduce both. In this paper, we concentrate on reducing the number of accesses.

Our out-of-core programming model is based on several key aspects. Due to space limitation, we limit ourselves here to an enumeration of the issues and illustration of the simplified pseudo-code in Figure 2:

- *Storing small (linear vs. non-linear) size critical (most frequently accessed) information in memory*. Frequent checks for conical independence drive our choice of in-core data structure. Based on the reduced conical independence theorem (Theorem 4), it is sufficient for performing these checks to have the in-core data structure that consists of: 1) the *RF array* $\mathbf{A}^{(j)}$ of *RF* vectors corresponding to the rows of the left-hand side of the current tableau, $\mathbf{T}^{(j)}$; and 2) the *RF* hash table $\mathbf{H}^{(j)}$ of $\mathbf{T}^{(j)}$. Thus, only the summary information about the tableau is stored in memory; the current and next tableaux, $\mathbf{T}^{(j)}$ and $\mathbf{T}^{(j+1)}$, respectively, are stored on disk.
- *Providing a flexible trade off between computational cost and I/O access cost*. We do not need to store (hence, read and write) the right-side ($\mathbf{S}^{(j)}$) of the tableau, $\mathbf{T}^{(j)}$ (originating from the $\mathbf{S}$ matrix). We can always compute it provided the left-side ($\mathbf{I}^{(j)}$) is known:
$$\mathbf{S}^{(j)} = \mathbf{I}^{(j)} \cdot \mathbf{S}^{\mathrm{T}} \qquad (17)$$
- *Deploying a data-parallel programming paradigm in partitioning data into in-core and out-of-core that will naturally lead to the parallelization of the algorithm*. Every iteration, the rows of tableau $\mathbf{T}^{(j)}$ are partitioned into three parts with each part stored in a separate file: the file *j.zero* stores all the "zero" rows of $\mathbf{T}^{(j)}$ that have a zero in the pivot column and the file *j.pos* (*j.neg*) stores all the "positive" ("negative") rows with a positive (negative) entry in the pivot column. The partition is done by the previous $(j\text{-}1)^{\text{th}}$ iteration: before a new row of $\mathbf{T}^{(j)}$ is stored on disk, the value in the j^{th} pivot column is checked and the row is saved into the appropriate file. Such partitioning of the matrix has several advantages:
  1) It reduces the computational complexity of finding the next positive-negative pair of rows (lines 2-5 in Figure 1) from $O(p^2)$ to $O(p)$ where *p* is the number of rows in $\mathbf{T}^{(j)}$);
  2) It allows for restructuring the code in such a way that the decision on what should be in-core and out-core becomes very deterministic. In particular, the positive and negative rows are processed by two nested loops (lines 5-12 in Figure2) with the internal loop (of higher I/O access rate) over the smaller file. The in-core vs. out-core decision is made based on the priority determined by the number of accesses: entries of $\mathbf{A}^{(j)}$ and $\mathbf{H}^{(j)}$ (line 2), entries of the internal loop (line 6), and entries of the external loop (line 5) in decreasing order. The out-of-core entries of the matrix are brought in-core in big blocks of size determined by the available memory. Thus, this strategy allows us to fully utilize the entire in-core memory;

3) It easily expands to the parallel implementation of the out-of-core code (see "Parallel Computation Model" section);

4.4. Parallel Computation Model

In this paper, we will assume an abstract machine model in which a number of processors are interconnected via a high-speed network. Each processor is connected to a local disk of its own that will be used as out-of-core scratch space.

Given the partitioning of data described in the previous section, the parallel algorithm becomes as follows. For every j^{th} iteration, each processor initializes a *global RF* array and a global *RF* hash table of the matrix $\mathbf{T}^{(j)}$. Initialization is done by merging all *local RF* arrays stored by other processors on their scratch disks (*/tmp* partition is assumed by default). This data is transferred by the requesting processor using the *rcp* UNIX utility. Uniform file naming convention and *pid-hostname* (*pid* is a process id) mapping allow for such transfers without requesting it from the owner process. Processor-specific offset number for row indices obtained from the *master* processor is used to maintain a global indexing scheme in both the *RF* array and the *RF* hash table. Each processor is processing its own local data files (*j.mypid.zero, j.mypid.pos*, *j.mypid.neg*) as well as remote negative (or positive depending on global size) files (*j.pid.neg*) from all the other processors. Note that with this scheme the collection of all possible pairs of rows of opposite sign is completely covered. On completion of the j^{th} iteration, a processor synchronizes with the master on whether to continue with the next iteration (if all processors finished this iteration). It also sends the master some information required for local-global mapping of row indices. The pseudo-code of the parallel out-of-core algorithm is given in Figure 3.

Maintaining reasonable load balance is another issue in the design of our parallel code. We expanded our computational model above by making each processor operate in a concurrent server-client mode using multi-threading. As a client, a processor executes the parallel out-of-core code described above. As a server, it accepts requests from other clients that have finished their execution and are ready to help. A server-thread makes a decision on which of the data files are unprocessed, sends the requester the name of the file, and updates the list of locally processed files. The requester ("helper") gets information on which processor needs help from the master. Thus, the master is responsible for assigning helpers, maintaining global mapping information, and interprocess synchronization.

5. Performance Evaluation

While experiments showed differences among the serial and parallel out-of-core algorithms in CPU-time and memory up to several orders of magnitude, a theoretical analysis is difficult. One major source of difficulty is predicting the resulting number of extreme pathways that is largely determined by the structure of the stoichiometric matrix. This number can vary from one to $(const \cdot n)^{\lfloor n-rank(\mathbf{S}) \rfloor}$. Another difficulty is introduced by the iterative structure of the algorithm; it is not only the structure of the stoichiometric matrix, but the structure of all intermediate tableaux (including the number of rows in the current tableau, the number of positive-negative pairs and the number of pairs that meet the conical independence constraint) generated in the solution process determine the complexity. Considering these difficulties we give complexity estimates for a given iteration expressed in terms of its parameters as a first step in this direction. In this state, experiments will prove useful for a better understanding of the stoichiometric structure and the behavior of different algorithmic concepts.

5.1. Complexity Analysis

To allow a theoretical comparison of the serial and out-of-core EP algorithms we sketch some space and time complexity analyses not for the entire program but for a given iteration. Even for this case it seems to be very difficult to obtain tight complexity results. That is why general upper bounds will be

provided although these bounds might be far from being tight. Let $p^{(j)}$ and $q^{(j)}$ denote the number of rows and positive-negative pairs in the tableau $\mathbf{T}^{(j)}$, respectively. Referring to the pseudo-codes of the serial EP algorithm in Figure 1 and of the out-core algorithm in Figure 2, the following space and time complexities can be provided:

- *Space complexity*. Since both $\mathbf{T}^{(j)}$ and $\mathbf{T}^{(j+1)}$ are stored in memory, the space complexity of the serial code is comprised of the space requirements to store each of the double precision matrices:

$$Space^{(j)} = Space(\mathbf{T}^{(j)}) + Space(\mathbf{T}^{(j+1)}) = (p^{(j)} + p^{(j+1)}) \cdot (n+m) \cdot sizeof(double) \quad (18)$$

Substituting the upper bound for $p^{(j+1)}$, this results in the following worst case space cost:

$$Space^{(j)} = O((p^{(j)})^2 \cdot n) \quad (19)$$

The out-of-core (OOC) algorithm needs to store the *RF* array, $\mathbf{A}^{(j)}$, and *RF* hash table, $\mathbf{H}^{(j)}$, in memory for its efficient execution. This results in the following space requirement for *b*-bit hardware:

$$Space_{OOC}^{(j)} = Space(\mathbf{A}^{(j)}) + Space(\mathbf{H}^{(j)}) = \left(\left\lfloor \frac{n}{b} \right\rfloor + 2\right) \cdot p^{(j)} \cdot sizeof(integer) + (n^2 + 3p^{(j)}) \cdot sizeof(int*) \quad (20)$$

In a big-O notation, equation (20) is equivalent to:

$$Space_{OOC}^{(j)} = O(p^{(j)} \cdot n) \quad (21)$$

Thus, from equations (19) and (21) the out-of-core algorithm has linear space complexity compared to quadratic space complexity of the serial code for a given iteration.

- *Time complexity*. Note that processing of positive-negative rows takes most of the execution time. Computational time for processing zero rows is linear and will be ignored in what follows. Since in the serial EP algorithm every positive-negative pair is checked for conical independence constraint (7) against all the rows of $\mathbf{T}^{(j)}$, the upper bound for the time complexity can be estimated as follows:

$$Time^{(j)} = q^{(j)} \cdot p^{(j)} \cdot O(n^2) = O((p^{(j)})^3 \cdot n^2) \quad (22)$$

It seems to be difficult to evaluate the time complexity of the OOC algorithm due to the difficulty of predicting the actual number of rows of $\mathbf{T}^{(j)}$ (after search space reduction) against which the conical independence constraint will be checked. Experiments and heuristic properties of metabolic networks indicate that on average this number should be very low. Based on the pseudo-code in Figure 2, time complexity of the OOC algorithm can be evaluated as:

$$Time_{OOC}^{(j)} = \sum_{i=1}^{q^{(j)}} \alpha_i \cdot p^{(j)} \cdot O(1) \quad (23)$$

where $\alpha_i, 0 \le \alpha_i \le 1$, determines the proportion of the search space reduction for the i^{th} positive-negative pair.

5.2. Empirical Evaluation Results

We have implemented the ideas described here in portable ANSI C. Interprocess communication within the same machine is through UNIX POSIX threads. Interprocessor communication between different machines is based on a concurrent server-client model through a simple API of our design built on top of TCP/IP UNIX sockets. The results presented here are for a cluster of 8 450MHz Pentium Pro systems running Linux each with 1GB of memory and a 500MB SCSI temp disk partition devoted to our out-of-core scratch space. The communication system is very modest, consisting of a single 100baseT Ethernet switch.

To demonstrate the performance of our algorithm, we provide results for a real metabolic subsystem of *E. coli* with 66 metabolites and 118 reactions, of which 24 are reversible. Our results on several other real metabolic subsystems show similar behavior. Figure 4 shows the analysis of the number of pathways and the number of positive-negative row pairs for the first 16 iterations. Figure 4.a. and 4.b. are plots of

the actual number of pathways, $p^{(j)}$, and the ratio of these numbers in two consecutive iterations, $p^{(j)}/ p^{(j-1)}$ (giving the base of exponent if number of pathways is expressed as a power sequence), respectively. The number of pathways grows exponentially with the number of iterations (giving a fluctuating base of exponent with an average value of 2). Almost quadratic dependence of the number of positive-negative row pairs (Figure 4.c) on the number of rows in the tableau (Figure 4.a) is observed. This is consistent with an approximately even split between positive and negative rows. The ratio of the maximum possible versus actual number of positive-negative row pairs for a given number of rows is plotted in Figure 4.d. This ratio eventually stabilizes around a constant. Thus, for this example, the time complexity per iteration is close to a theoretical upper bound given in equation (22).

Figure 5 shows comparative results for overall memory utilization of the in-core (Figure 1) and modified out-of-core (Figure 2) algorithms. The memory cost for the in-core algorithm grows exponentially with the number of iterations (giving a fluctuating base of exponent with an average value of 2). The base of exponent average value for the growth of memory cost in the out-of-core algorithm is 1.3 (Figure 5.a). A sustained memory improvement of one to two orders of magnitude with each iteration is observed (Figure 5.b). When the number of pathways reaches 118551 (iteration 16), the in-core algorithm core-dumps on a 256MB PC because the memory requirement becomes 276MB.

Table 1 shows comparative results for the overall execution time cost for the in-core algorithm (Figure 1) and parallel out-of-core algorithm (Figure 3) on one, two, four and eight processors. The comparison of the in-core algorithm and the parallel algorithm on a single processor demonstrates the improvement of two to three orders of magnitude. The parallel algorithm scales up almost linearly with the number of processors. Finally, the total execution of the parallel algorithm on 20 processors of a general-purpose cluster of SUN SPARC workstations completes within 3 hours whereas the in-core algorithm does not finish within 4 days.

Acknowledgements

The authors wish to acknowledge Professor Bernhard O. Palsson and his “Genetics Circuit Research Group” at University of California, San Diego, for bringing our attention to the importance of the extreme pathways enumeration problem and the need for more efficient algorithms to compute them and for providing real data sets to test the performance of our algorithm. We also thank Dr. Ed D'Azevedo of the Oak Ridge National Laboratory for fruitful discussions on sparse linear algebra problems.

References

Bordawekar, R., Choudhary, A., Kennedy K., Koelbel C., & Paleczny, M. (1995). A model and compilation strategy for out-of-core data parallel programs. *In Proceedings of the Fifth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming* (PPOPP), pp. 1-10. ACM Press, July 1995. ACM SIGPLAN Notices, Vol. 30, No. 8.

Clarke, B.L. (1981). Complete set of steady states for the general stoichiometric dynamical system. *J. Chem. Phys.* **75**: 4970-4979.

Fell, D.A. (1990). Substrate cycles: theoretical aspects of their role in metabolism. *Comm. Theor. Biol.* **6**: 1-14.

Garey, M.R. & Johnson D.S. (1979). *Computers and intractability: a guide to the theory of NP-completeness*, W. H. Freeman.

Golub, G.H. & Van Loan, C.F. (1996). *Matrix Computations*, The Johns Hopkins University Press.

Karp, P. D., Ouzounis, C., et al. (1996). HinCyc: A Knowledge Base of the Complete Genome and Metabolic Pathways of *H. influenza*e. *Proceedings of the ISMB-96 Conferenc*e: 9.

Kholodenko, B.N., Shuster, S., Rohwer, J.M., Cascante, M., & Westerhoff, H.V. (1995). Composite control of cell function: Metabolic pathways behaving as single control units. *FEBS Lett.* **368**: 1-4.

Mavrovonouniotis, M.L., Stephanopolous, G., & Stephanopolous, G. (1990). Computer-aided synthesis of beiochemical pathways. *Biotechnol. Bioeng.,* **36**: 1119-1132.

Nozicka, F., Guddat, J., Hollatz, H., & Bank, B. (1974). *Theorie der linearen parametrischen optimierung.* Akademie-Verlag, Berlin.

Overbeek, R., Larsen, N., et al. (2000). WIT: integrated system for high-throughput genome sequence analysis and metabolic reconstruction. *Nucl. Acids Res.* **2**8(1): 123-125.

Pfeiffer, T., Sanchez-Valdenebro, I., Nuno, J.C., Montero, F., & Shuster, S. (1999). METATOOL: for studying metabolic networks. *Bioinformatics* **15**(3): 251-257.

Rockafellar, R. (1970) *Convex Analysis.* Princeton University Press, Princeton, NJ.

Rohwer, J.M., Shuster, S., & Westerhoff, H.V. (1996). How to recognize monofunctional units in a metabolic system. *J. Theor. Biol.* **179**: 213-228.

Selkov, E., Jr., Grechkin, Y., et al. (1998). MPW: the Metabolic Pathways Database. *Nucl. Acids Res.* **2**6(1): 43-5.

Selkov, E., Maltsev, N., Olsen, G.J., Overbeek, R., & Whitman, W.B. (1997). A reconstruction of the metabolism of *Methanococcus jannaschii* from sequence data. *Gene* **197**: GC11-GC26.

Schilling, C. H., Letscher, D., and Palsson, B.O. (2000.a). Theory for the Systemic Definition of Metabolic Pathways and their use in Interpreting Metabolic Function from a Pathway-Oriented Perspective. *J. Theoret. Biol.* **20**3: 229-248.

Schilling, C. H., Edwards, J., Letscher, D., & Palsson, B.O. (2000.b). Combining pathway analysis with flux balance analysis for the comprehensive study of metabolic systems. *Biotechnology and Bioengineering* **71**(4): 286-306.

Schilling, C. H. & Palsson, B.O. (1998). The underlying pathway structure of biochemical reaction networks. *Proc. Natl. Acad. Sci. U.S.A* **95:** 4193-4198.

Schilling, C. H. & Palsson, B.O. (2000.c). Assessment of the Metabolic Capabilities of *Haemophilus influenzae Rd* through a Genome-scale Pathway Analysis. *J. Theoret. Biol.* **20**3: 249-283.

Schilling, C. H., Schuster, S., Palsson, B.O., & Heinrich, R. (1999). Metabolic pathway analysis: basic concepts and scientific applications in the post-genomic era. *Biotechnol. Prog.* **1**5(3): 296-303.

Schuster, S., Dandekar, T., & Fell, D.A. (1999). Detection of elementary flux modes in biochemical networks: a promising tool for pathway analysis and metabolic engineering. *Trends. Biotechnol.* **1**7(2): 53-60.

Schuster, S., & Hilgetag, C. (1994). On elementary flux modes in biochemical reaction systems at steady state. *J. Biol. Syst.* **2**: 165-182.

Schuster, R. & Schuster, S. (1993). Refined algorithm and computer program for calculating all non-negative fluxes admissible in steady states of biochemical reaction systems with or without some flux rates fixed. *Comput. Appl. Biosci.* **9**: 79-85.

Schuster, S., Fell, A.F., & Dandekar, T. (2000). A general definition of metabolic pathways useful for systemic organization and analysis of complex metabolic networks. *Nature Biotechnology* **18**: 326-332.

```
1: for each col=SelectColumn() {
2:     for each irow in T^col  {
3:       if (S^col[irow][col] == 0)  Copy irow to T^col+1
4:       else for each krow > irow {
5:            if (S^col[irow][col] * S^col[krow][col] < 0) {
6:                for any jrow {
7:                    if(w^i·irow+w^k·krow is independent on jrow in I^col) {
8:                        Copy w^i·irow+w^k·krow into T^col+1
9:                    } // end-if
10:               } // end-for-jrow
11:      } // end-for-krow
12:   } // end-for-irow
13: } // end-for-col
```

Figure 1. Pseudo-code of the serial in-core EP algorithm.

```
1: for each col {
2:     Initialize A^col and H^col
3:     Copy each row from col.zero into col+1.zero, col+1.pos or col+1.neg file;
4:     Update A^col+1
5:     for each irow in col.pos {
6:         for each krow in col.neg {
7:             if (w^i·irow+w^k·krow is independent on any jrow in A^col) {
8:                 Copy w^i·irow+w^k·krow into col+1.zero, col+1.pos or col+1.neg
9:                 Update A^col+1
10:            } // end-if
11:        } // end-for-col.neg
12:    } // end-for-col.pos
13: } // end-for-col
```

Figure 2. Pseudo-code of the serial out-of-core EP algorithm.

```
1: for each col {
2:     Initialize Global_A^col and Global_H^col
3:     Copy each row in col.mypid.zero into col+1.mypid.* file;
4:     Update Local_A^col+1
5:     for each irow in col.mypid.pos {
6:         for each pid {
7:             for each krow in col.pid.neg {
8:                 if (w^i·irow+w^k·krow is dependent in Global_A^col) {
9:                     Copy w^i·irow+w^k·krow into col+1.mypid.* file;
10:                    Update Local_A^col+1
11:                } // end-if
12:            } // end-for-col.pid.neg
13:        } // end-for-pid
14:    } // end-for-col.pos
15: } // end-for-col
```

Figure 3. Pseudo-code of the parallel out-of-core EP algorithm.

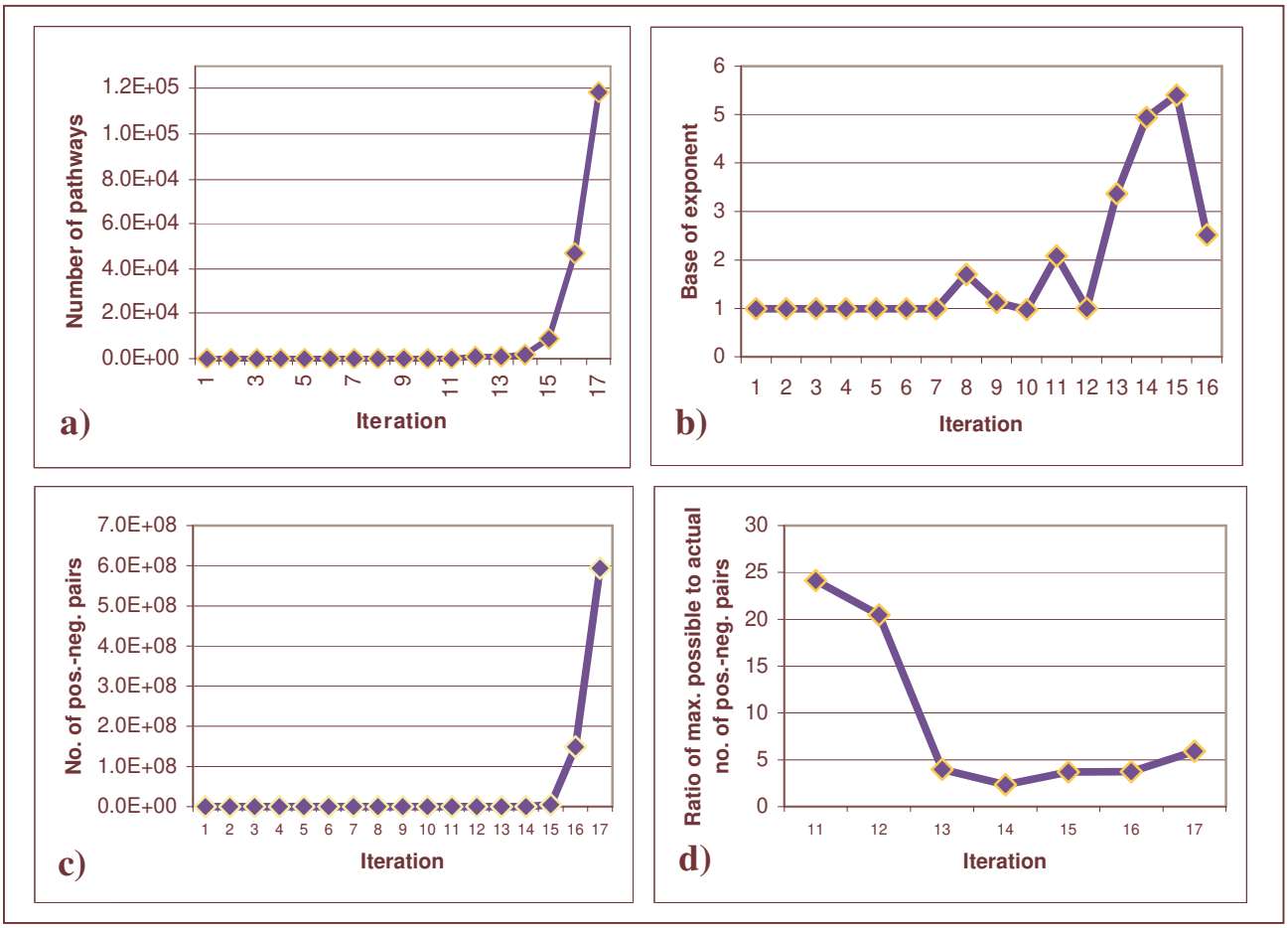

Figure 4. Comparative results on the number of pathways and number of positive-negative row pairs for a metabolic subsystem of *E. coli* with 66 metabolites and 118 reactions. a) The number of pathways at each iteration. b) The ratio of the actual number of pathways in two consecutive iterations. c) The number of positive-negative row pairs at each iteration. d) The ratio of maximum possible to actual number of positive-negative row pairs.

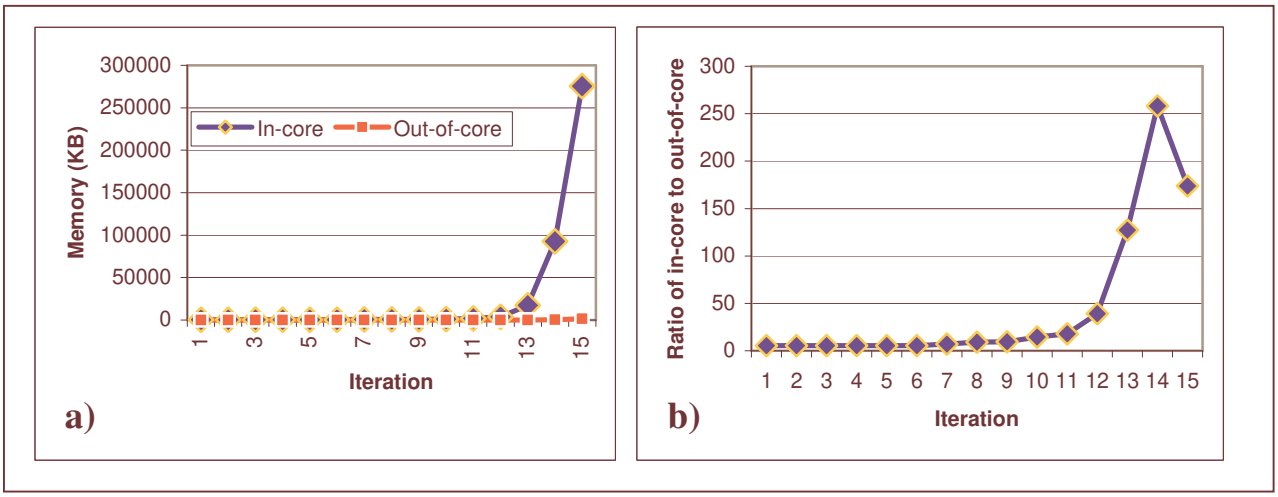

Figure 5. Comparative results for overall memory utilization of the in-core and modified out-of-core algorithms. a) Memory cost for in-core (diamonds) and out-of-core (squares) algorithms. b) The ratio of in-core to out-of-core memory costs.

Iteration	In-core	Parallel out-of-core			
		proc=1	proc=2	proc=4	proc=8
33	1	0	1	0	1
34	7	1	0	1	0
35	56	3	3	2	2
36	5784	115	64	35	20
38	90	4	2	2	2
39	151	4	4	3	3
40	Not finished	926	483	260	145
41	Not finished	1	1	1	1
42	Not finished	1	1	1	1
43	Not finished	171	90	52	33
44	Not finished	20	11	9	8
45	Not finished	1237	704	370	194
46	Not finished	1	1	1	1
47	Not finished	596	273	179	98

Table 1. Comparative results for overall execution time cost for the in-core algorithm and parallel out-of-core algorithm on one, two, four, and eight processors.