Data-driven, Data-intensive Computing for Modelling and Analysis of Biological Networks: Application to Bioethanol Production

Byung-Hoon Park, Nagiza F. Samatova, Tatiana Karpinets, Andrew Jallouk, Scott Molony, Scott Horton, and Steven Arcangeli

Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN

Contact: samatovan@ornl.gov

Abstract. Modelling biological networks is inherently data-driven and data-intensive. The combinatorial nature of this type of modelling, however, requires new methods capable of dealing with the enormous size and irregularity of the search. Searching via "backtracking" is one possible solution that avoids exhaustive searches by constraining the search space to the subspace of feasible solutions. Despite its wide use in many combinatorial optimization problems, there are currently few parallel implementations of backtracking capable of effectively dealing with the memory-intensive nature of the process and the extremely unbalanced loads present. In this paper, a parallel, scalable, and memory-efficient backtracking algorithm within the context of maximal clique enumeration is presented, and its applicability to large-scale biological networks aimed at studying the mechanisms for efficient bioethanol production is discussed.

1. Introduction

Unlike scientific simulations from "first principles," where underlying models are described by a system of equations, biological network modelling is often *data-driven*. This process frequently begins with a comprehensive enumeration of the "parts" (e.g., transcription factors or co-regulated proteins) derived from transcriptomics, proteomics, or genomics data. Discovery of putative interactions between these "parts" can then be used to design *in silico* models of biomolecular interactions, such as protein-protein complexes, metabolic or regulatory pathways.

Data-driven model construction is often considered to be a combinatorial optimization problem in which a search for a particular object or *enumeration* of all the objects with given properties is being sought. The data-intensive nature of this problem, however, causes existing methods to fail due to their inability to conform to the required scale of data size, heterogeneity, and dimensionality. Table 1 highlights some of the differences between simulation and combinatorial optimization techniques in terms of input size and structure, memory access, disk access, output size, communication requirements, and types of arithmetic operations used. Such differences create a need for novel architectural designs and algorithms using an intelligent balance of memory, disk storage, and communication trade-offs. In this paper, we present a memory-efficient, scalable, algorithmic approach to the enumeration of maximal cliques in biological networks.

Table 1. Distinct data access in running simulations and building models.

	Simulation	Search	Enumeration
Input	medium	huge	medium
Memory access	local (2 time steps)	global (entire database)	exponential irregular
Output to disk	iterative	irregular	irregular, huge
Communication	intensive	for scoring	load balancing
Arithmetic	float	integer (float)	integer (float)

2. Parallel Backtracking Maximal Clique Enumeration

Biological pathways and networks can be mathematically represented as graphs where nodes represent genes or metabolites and edges represent some type of relationship between them, such as regulation, physical interaction, or catalytic activity for the conversion of a metabolite to a substrate. Using these graphs, questions about biological networks can be translated into combinatorial problems, which can then be grouped into three categories: search (e.g. subgraph isomorphism), optimization (e.g. minimum vertex cover), and enumeration (e.g. all maximal cliques). "Backtracking" is a common underlying technique for traversing the search space for these problems. As the most computationally intensive part of the modelling process, this technique is vitally in need of memory-efficient and scalable parallelization. The rest of this paper focuses on the design and implementation of backtracking parallelization in the context of maximal clique enumeration.

2.1. *Breadth First-based and Depth First-based Algorithms*. The most widely used maximal clique enumeration algorithms are based on backtracking [1]. Unlike a brute force approach, which exhaustively searches through the input graph for maximal cliques, a backtracking method avoids exploring unpromising paths in the graph by applying the property of clique completeness. In other words, all vertices in a given path must be pair-wise connected in order for the algorithm to further explore the path. An efficient backtracking algorithm must also adopt additional constraints to further limit unnecessary traverses. In this work, we applied the backtracking strategy of Bron and Kerbosh [2] (BK). Using this approach, all possible paths that a backtracking algorithm can explore are represented as a single tree, called the search tree. A path (from the root) in a search tree corresponds to a clique, and the tree is expanded as the algorithm traverses the graph. The algorithm stops expanding a path and returns to its previous level (backtracks) if no further expansion in the current direction leads to new cliques.

Paths in a search tree can be grown in parallel by either a Breadth First Search (BFS) or a Depth First Search (DFS). At first glance, the parallelization of a backtracking algorithm may seem to favor the BFS approach. By forcing every participating processor to expand all paths to the same level of the search tree at any given step, coordinating the overall procedures becomes straightforward. Additionally, this strategy causes maximal cliques to be produced in order of increasing size, a property that can be an invaluable asset to certain types of applications. Since every branch at the current deepest level of the search tree needs to be kept in core memory, however, the amount of memory required by a BFS-based implementation can easily exceed the capacity of available resources. For example, *pClique* [3], the first and only backtracking, parallel, maximal enumeration algorithm, implements the BFS strategy, and, indeed, illustrated a huge memory requirement. With two gigabytes of available memory, our implementation could not operate on even relatively small graphs (approximately 5,000 vertices and less than 10,000 edges). Considering the fact that a graph constructed from biological data can easily include million of vertices and edges, such BFS-based methods are impractical.

2.2. *Sequential Implementation of BK-based Backtracking Algorithm.* In order to create an efficient parallelization, it becomes necessary to convert BK, an originally recursive algorithm, into a non-

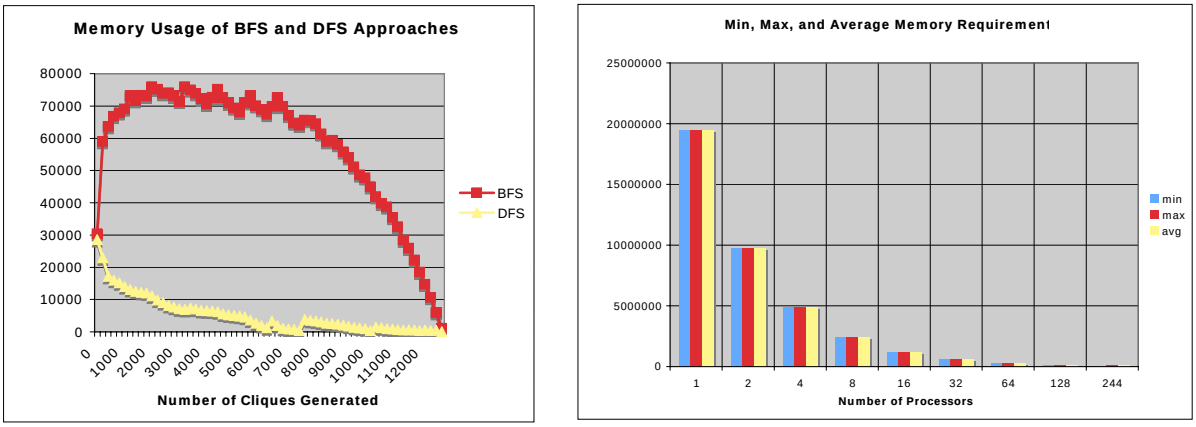

Figure 1: (a) Memory usage pattern of BFS-based and DFS-based maximal clique enumeration algorithms. Memory usage (in bytes) is measured every time 200 cliques are found. (b) Minimum, maximum, and average memory requirements per processor when up-to 244 processors are used for pDFC.

recursive version by making each recursion step independent of any previous steps. When a recursive backtracking algorithm expands a search path at a certain node, all information needed for expansion (or backtracking) is stored in the system stack. Such information is stored and retrieved in a strict Last In First Out order. Since a sequential algorithm does not operate on the system stack, all information should be supplied during the expansion, regardless of the order of expansion steps. This requirement necessitates a representation of such information for expansion. We devised one such representation that embeds:

- The clique represented by the path to the current node.
- All eligible vertices to the path (common neighbors).
- Vertices covered earlier in expanding the parent path (to avoid redundant coverage).

We call a data structure that includes all this information a *candidate path*. Note that a candidate path is self-sufficient in expanding the path. A candidate path is said to be of order k if the clique represented by the path includes k vertices.

Since a candidate path is self-sufficient in expanding the path, the enumeration process of a sequential implementation is not affected by the order in which these candidate paths are stored and retrieved. Breadth first- or depth first-based sequential algorithms can be implemented by employing different strategies in storing and retrieving candidate paths in a given memory queue, i.e., First In First Out (FIFO) for BFS-based and Last In First Out (LIFO) for DFS-based. We will refer to the BFS- based, BK-Improved variant as Breadth First Clique (BFC) and the DFS-based, BK-Improved variant as Depth First Clique (DFC).

2.3. *Parallelization of BFC and DFC.* Parallelization of a backtracking, maximal clique enumeration algorithm is based on the idea of "dynamic search tree decomposition," a technique where each branch of the search tree is allocated to a single processor for further expansion. Various assignment strategies are possible, depending on the priority of parallelization schemes. If fully balanced loads among processors are anticipated, a random allocation strategy in which every newly expanded node is assigned to a randomly chosen processor is desirable. Such a strategy is not appropriate for large graphs, however, due to the vast number of candidate paths that may be generated during the entire enumeration process.

If, on the other hand, a processor is responsible for exploring the entire space of a given sub-tree, further decomposition causes the load among the processors to become extremely unbalanced. In this case, many processors would finish exploring their search trees quickly, but a few, "unlucky" ones, would still be struggling with expanding their search trees. Since it is impossible to predict *a priori* the

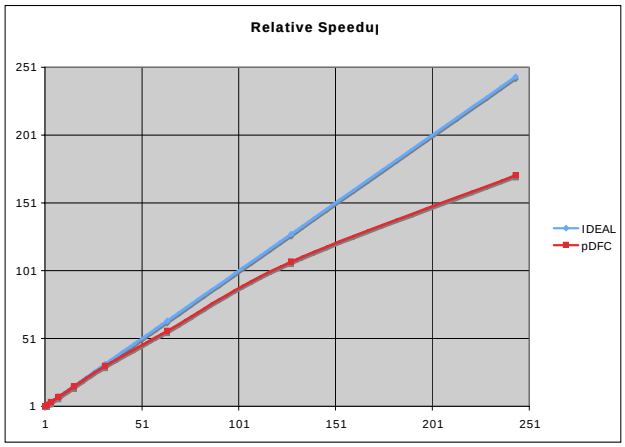

Figure 2: Relative speedup of pDFC.

size of each search tree and the time required to expand it, a dynamic load balancing mechanism with minimum communication cost is the key to a successful parallelization.

Based on these observations, we choose to adopt the random stealing [4, 5] technique with simplified stack splitting [6]. In this strategy, the search tree is split at the first level and each processor then takes charge of enumerating all maximal cliques from the allotted sub-trees. Technically, each processor is given a number of candidate paths of order one, and continues to independently enumerate maximal cliques until all candidate paths are fully expanded. When no more candidate paths are available, the processor requests a new one from a randomly chosen processor. If no candidate path is received after some trials, the processor halts. When a processor receives a request for a candidate path, it sends the one of lowest order from the queue. Since a candidate path is not yet fully explored, the one of lower order tends to spawn a larger number of higher order candidate paths. For both BFC and DFC, this scheme is easily realized by exchanging a candidate path that is retrieved from the head of the queue. By adapting this simple, stack splitting paradigm, we are able to implement parallel frameworks for both BFC (pBFC) and DFC (pDFC)

3. Empirical Study

This section presents an empirical evaluation of the pDFC framework. In particular, we discuss its scalability in terms of memory requirement and runtime on multiple processors. We begin by comparing the memory requirements of the sequential versions of BFC and DFC, as measured from a run on a small graph (~1,000 vertices). Next, the performance of pDFC over a large sparse graph (~200,000 vertices and ~2 million edges) is examined. Unfortunately, the performance of pBFC over this graph cannot be measured due to its high memory requirement. For these experiments, the initial prototypes of both pBFC and pDFC are implemented in a multithreaded environment. All measurements are conducted on the ORNL SGI Altrix 3700.

3.1. *Memory Requirements of BFC and DFC.* The memory requirements of both BFC and DFC depend primarily on the size of the candidate paths that must reside in main memory for the search process to continue. For BFC, all candidate paths of order k should be accessible in order to produce candidate paths of order $k+1$, and an order k candidate path may be safely removed from memory once it is expanded. Thus, the memory requirement of BFC is bounded by the largest candidate path set of order k and $k+1$. On the other hand, for DFC, only those candidate paths that are generated along the current path need to be stored in main memory. Therefore, the memory requirement for a DFC is bounded only by the deepest path of the search tree (a path from the root to the deepest leaf). To empirically verify this observation, we measure the memory usage of both BFC and DFC clique enumeration algorithms on a graph that has 858 vertices and 10,823 edges. The graph has the total of 12,631 cliques of size 3 or above. The largest clique size is 21. Figure 1-(a) shows the memory usages of both algorithms as measured after every 200 new cliques are found during their iterative runs. As anticipated, the memory usage of DFC is high in the beginning (probably when the backtracking tree hits a leaf) and decreases as more cliques are found. In contrast, the memory usage of BFC gradually increases and then decreases forming a bell-shaped curve. In summary, the overall memory requirement of BFC is significantly larger than that of DFC.

3.2. *Scalability Study of pDFC.* In this evaluation, we created a graph of 193,568 vertices and 2,260,872 edges based on the homology associations among genes in a set of 80 bacterial genomes. The scalability performance of pDFC is assessed with respect to both the memory required per processor and the total execution time. First, the minimum, maximum, and average memory requirements per processor are measured when pDFC is run over 2, 4, 8, 16, 32, 64, 128, and 244 processors, respectively. As shown in Figure 1-(b), the memory requirement decreases rapidly as more processors are deployed, with only a very small variance observed among minimum, maximum, and average requirements. As illustrated in Figure 2, pDFC also shows scalability across multiple processors. The number of maximal cliques found is 395,306.

3.3 *Application to Bioethanol Production.* To efficiently produce ethanol from biomass using a bacterial consortium, certain phenotypic traits (e.g the ability to metabolize different sugars and grow in oxygen-poor environments) must be present in the system. It is therefore important to link desirable microbial traits with the specific genes and biochemical pathways that control them.

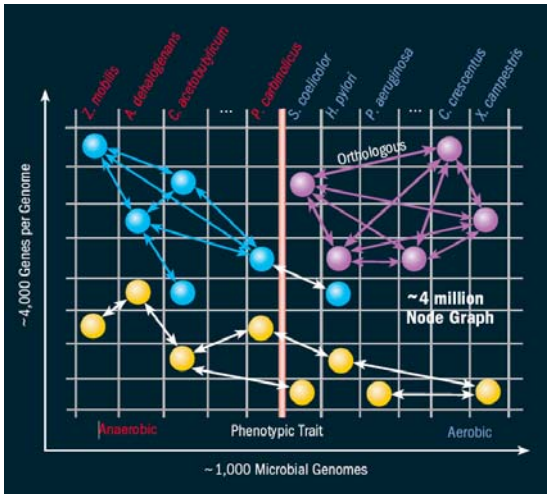

Figure 3. Graph representation of genes that are conserved among organisms that possess a beneficial trait. Illustration: A. Tovey.

To find these genes, we proposed to classify the organisms according to the presence or absence of a trait and to search for genes that are prevalent on one side of the divide but not the other. The underlying theory is that genes related to a critical trait will be conserved by evolution and, thus, will appear only in the genomes of organisms possessing this trait. This problem can be formulated in graph theoretical terms as an enumeration of the maximal cliques of genes found in organisms that express a desired trait (Figure 3). As a case study, we analyzed the genomes of 14 aerobes and 14 anaerobes in order to determine the genes responsible for anaerobic growth.

First, FASTA files containing the genomes of these microorganisms were downloaded from GenBank and were searched for similarities in order to create a graph. Vertices in this graph represent genes, and edges denote an orthologous relationship between these genes. In this study, we defined two genes to be orthologous if a comparison using the Basic Local Alignment Search Tool (BLAST) yielded an E-value below a certain threshold (0.01) in *both* directions. Intra-genomic edges were not considered. We hypothesized that the vertices representing groups of orthologous genes would form cliques.

After a maximal clique enumeration algorithm was run on this graph, several orthologous genes related to fructose and mannose metabolism were identified (Table 1). Note that while none of the aerobes have conserved genes in the mannose-specific phosphotransferase system (PTS), six out of the 14 anaerobic organisms contain genes that enrich this pathway. The mannose PTS gene cluster (man) is comprised of three genes and a putative regulator. The man operon and the regulator are constantly transcribed without being affected by culture conditions, such as the sugar supplied, growth rate, or pH. Since mannose is a common sugar found in biomass, the PTS system responsible for transporting it into the cell is important for efficient ethanol production.

The other important feature of the E IIMan complex is its involvement in the catabolite repression (inactivation) of certain sugar-metabolizing operons in favour of glucose utilization when glucose is

Table 1. An example of genes evolutionary conserved across aerobic/anaerobic organisms.

Fructose and mannose metabolism	
Specific to anaerobic bacteria	***Specific to aerobic bacteria***
Fructose-bisphosphate aldolase [EC:4.1.2.13] fbaB	Sorbitol dehydrogenase [EC:1.1.1.14]
Mannose-specific PTS component IID (ptnD)	Mannose-6-phosphate isomerase [EC:5.3.1.8]
Mannose-specific PTS component IIAB [EC:2.7.1.69] (ptnA)	Glucosyl transferase [EC:2.4.1.-]
	Mannitol 2-dehydrogenase [EC:1.1.1.67]

the predominant carbon source in the environment of the cell. Mutations leading to inactivation of this complex resulted in a loss of the preferential use of glucose over other sugars, such as lactose in *Lactobacillus casei*, and xylose in *Tetragenococcus balophila*. As a rule, microbes create enzymes for the degradation of more resistant substrates (like pentoses or lignin) only in the absence of readily available nutrients like glucose. Degradation of lignin by microbes is therefore especially delayed and slow. Considering the obtained results, we can propose that genomic modifications of the mannose PTS in anaerobic organisms may be a way to overcome catabolite repression and, therefore, increase the utilization of pentose by these organisms.

Acknowledgements
This research has been supported by the "Exploratory Data Intensive Computing for Complex Biological Systems" project from U.S. Department of Energy (Office of Advanced Scientific Computing Research, Office of Science). The work of N. F. Samatova was also sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory. Oak Ridge National Laboratory is managed by UT-Battelle for the LLC U.S. D.O.E. under contract no. DE-AC05-00OR22725.

References

1. Brassard, G. and P. Brately, *Fundamentals of Algorithmics.* Prentice Hall, 1996.
2. Bron, C. and J. Kerbosch, *Algorithm 457: finding all cliques of an undirected graph.* Commun. ACM, 1973. **16**(9): p. 575-577.
3. Zhang, Y., et al., *Genome-Scale Computational Approaches to Memory-Intensive Applications in Systems Biology*, in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. 2005, IEEE Computer Society.
4. Umut, A.A., E.B. Guy, and D.B. Robert, *The data locality of work stealing*, in *Proceedings of the twelfth annual ACM symposium on Parallel algorithms and architectures*. 2000, ACM Press: Bar Harbor, Maine, United States.
5. Joxan, J., et al., *Scalable Distributed Depth-First Search with Greedy Work Stealing*, in *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04) - Volume 00*. 2004, IEEE Computer Society.
6. Vipin, K., Y.G. Ananth, and V. Nageshwara Rao, *Scalable load balancing techniques for parallel computers.* J. Parallel Distrib. Comput., 1994. **22**(1): p. 60-79.