On the Relative Efficiency of Maximal Clique Enumeration Algorithms, with Application to High-Throughput Computational Biology[*]

Faisal N. Abu-Khzam[1], Nicole E. Baldwin[2], Michael A. Langston[3] and Nagiza F. Samatova[2]

1. Division of Computer Science and Mathematics, Lebanese American University, Chouran, Beirut, Lebanon, faisal.abukhzam@lau.edu.lb
2. Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN 37831–6367, USA, {baldwinne, samatovan}@ornl.gov
3. Department of Computer Science, University of Tennessee, Knoxville, TN 37996-3450, USA, langston@cs.utk.edu

Abstract: The efficient enumeration of maximal cliques has applications in microarray analysis and a number of other foundational problems of computational biology. In this paper, we analyze and test existing maximal clique enumeration algorithms for various classes of graphs. The classic branch and bound algorithm of Bron and Kerbosch proves to be relatively fast for sparse graphs, but slows considerably as edge density increases. Attempts to improve this algorithm are discussed. Experimental results demonstrate the difficulty of making improvements, especially when analyzing the overlap between cliques. Novel strategies for maximal clique enumeration algorithms are also described and placed in the context of ongoing research.

Keywords: Computational Biology, Graph Algorithms, High Performance Computation, Microarray Data Analysis

1. Introduction and Background

Clique is a well-known *NP*-complete problem. Its decision version is typically formulated as follows:

- Input: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
- Question: Is there a subset V' of V with $|V'| \geq k$ for which every pair of vertices in V' is joined by an edge in E?

In many applications, clique is posed as an *NP*-hard optimization problem without the parameter k. Thus the input is just a graph, and the question asked becomes what is the size of the largest clique in G? Clique may also arise as a search problem, where one seeks to find the largest clique in G. These problem variants are termed “maximum clique.” In addition, we often want to solve “maximal clique.” Here the usual goal is to enumerate all maximal cliques, each of which by definition cannot be contained as a subgraph in any larger clique. It is not hard to see that a graph with n vertices can have as many as $3^{n/3}$ maximal cliques[1], of which some (but very rarely all) are of maximum size.

[*]Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No.DE-AC05-00OR22725, by the National Science Foundation under grants CCR–0075792 and CCR--0311500, by the Office of Naval Research under grant N00014–01–1–0608, by the Tennessee Center for Information Technology Research under award E01–0178–081, and by the U.S. Department of Energy's Genomes to Life program (www.doegenomestolife.org) under the ORNL-PNNL project, “Exploratory Data Intensive Computing for Complex Biological Systems.”

Clique is well known for its utility in a wide variety of application domains [2]. Here we concentrate on its relevance to computational biology. Even a cursory search of the PubMed literature (http://www.ncbi.nlm.nih.gov/entrez/query.fcgi) reveals a myriad of uses. Applications for powerful new maximum clique tools arise, for example, in the context of *cis* regulatory motif finding [3], microarray analysis [4], and the study of quantative trait loci [5]. They have also been incorporated into the construction of ClustalXP [6], a high performance parallel version of the highly-popular ClustalW package.

In this paper we investigate, analyze, compare, and improve on existing maximal clique enumeration algorithms for various classes of graphs. Four algorithms are chosen for our study. The first two are similar recursive algorithms [7]. The third is an innovative approach that eliminates repetitive search of the same problem space [8]. The fourth is a brute force algorithm based on random set generation. Each of these algorithms is challenged with graphs derived from real microarray data, and normalized with both the MAS5.0 and RMA software packages. It is worth pointing out that each package has its supporters, yet each often produces markedly different end results. This can be seen in Figure 1, in which the X axis displays subranges of correlation coefficient values normalized to [-1,1], and the Y axis represents the number of coefficients with values in each subrange.

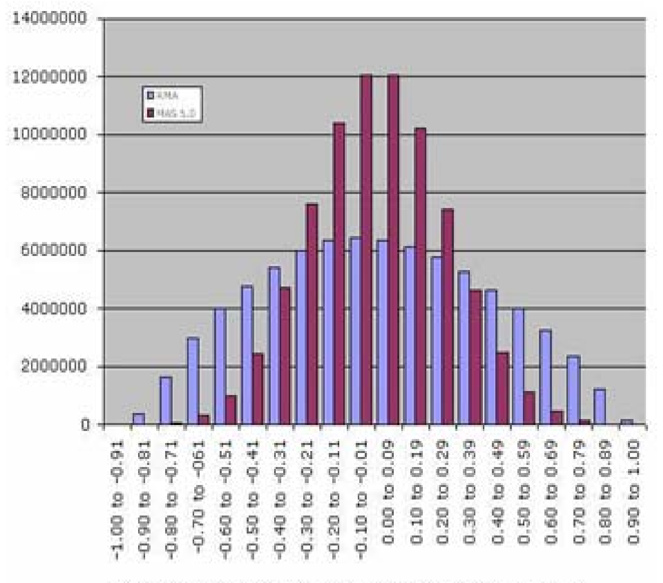

Figure 1. Sample edge weight histograms of MAS5.0 and RMA derived graphs.

2. The *Base BK* Algorithm

Published in 1973, the base maximal clique enumeration algorithm [7], called herein *Base BK*, utilizes a recursive branching strategy. It mainly consists of maintaining three dynamically changing sets:

- COMPSUB, a global set containing the current growing (or shrinking) clique,
- CANDIDATES, a local set holding all vertices that will eventually be added to the current COMPSUB, and
- NOT, a local set containing all vertices that have been previously added to COMPSUB.

The algorithm is performed by a function dubbed EXTEND, which operates as any recursive backtracking algorithm. It can be viewed as a depth-first traversal of a search tree. At each node of the search tree, EXTEND selects a vertex, *v*, from CANDIDATES. This vertex is removed from CANDIDATES and added to COMPSUB. Local sets NEW_CANDIDATES and NEW_NOT are generated, where NEW_CANDIDATES is the intersection of CANDIDATES and the neighborhood of *v*. Similarly, NEW_NOT, is the intersection of NOT and the neighborhood of *v*. If NEW_CANDIDATES and NEW_NOT are empty (i.e. no vertex remains that is completely connected to COMPSUB), then COMPSUB is a maximal clique, and EXTEND returns. Otherwise, EXTEND is called again, passing sets NEW_CANDIDATES and NEW_NOT as arguments to become the child node's CANDIDATE and NOT sets, respectively.

Returning from EXTEND causes the most recently added vertex to be removed from COMPSUB and added to NOT. To illustrate the usefulness of this action, note that for each already-examined vertex *v* of CANDIDATES, EXTEND must have found all maximal cliques containing *v*. Adding *v* to NOT means that, from the current search tree node, we do not try to find any other maximal clique containing *v*. EXTEND then selects a new *v*, iterating through the described actions, returning either when, as mentioned, it finds a maximal clique, or when set CANDIDATES is exhausted.

3. The *Improved BK* Algorithm

The second and more popular algorithm, published in the same paper as *Base BK* and called herein *Improved BK*, follows the branching blueprint laid out by the *Base BK* algorithm, but also takes some measures to limit the number of branches traversed. Its worst-case time complexity has only very recently been proven to be $O(3^{n/3})$ [9]. Its main difference from the *Base BK* algorithm lies in its choice of the selected vertex at each node of the search tree. Instead of function EXTEND choosing vertices in the order they are presented in the CANDIDATES set, this algorithm chooses a vertex with the largest number of connections to the other vertices in CANDIDATES. If there is more than one such vertex, EXTEND chooses any such vertex found. Additionally, when EXTEND chooses a new selected vertex after a return from a child node, it will only consider vertices that are not connected to the current *v*. The rationale for this is as follows. If at any point, the set NOT contains a vertex *u* that is connected to all vertices in the sets CANDIDATES and COMPSUB, then it is not possible to generate maximal clique(s) by expanding COMPSUB with the vertices in CANDIDATES because any clique so formed could only be maximal if it also contained *u* (such cliques were generated when *u* was the selected vertex). Therefore, upon return from a child node, EXTEND should ensure that the vertex chosen as the new selected vertex is disconnected from the vertex that is being added to NOT. The initial choice of a highly connected vertex minimizes the remaining vertices in CANDIDATES that are valid selections.

Such minimization clearly reduces the number of branches that would otherwise need to be traversed. This modification is, of course, only useful if the time spent finding vertices of maximum degree is less than the time that would have been spent exploring the eliminated branches of the search space. The expectation is that this algorithm would be a better choice for graphs with a large number of highly overlapping cliques. Under these settings, the algorithm should encounter the boundary condition more frequently to provide the greatest advantage. On the other hand, the *Base BK* algorithm should be faster when there is little overlap among cliques or when the number of cliques is sufficiently small.

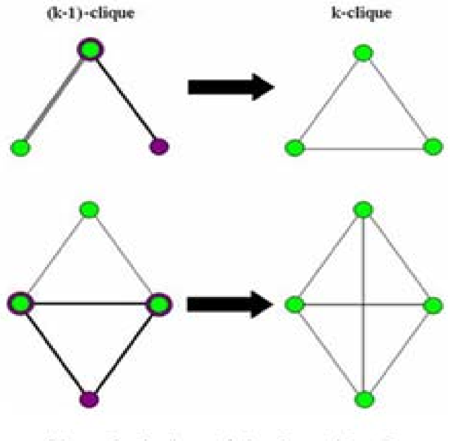

Figure 2. A clique of size *k* consists of two cliques that share *k-2* vertices.

4. The *Kose RAM/Disk* Algorithm

The algorithm of [8] takes a very different approach than the recursive branching procedures previously described. It takes advantage of the fact that every clique of size *k*, where $k \geq 2$, is comprised of two cliques of size *k-1* that share *k-2* vertices. This basic principle is illustrated in Figure 2. The algorithm takes as input an edge list with the edges (2-cliques) listed in non-repeating, canonical order and builds from it all possible 3-cliques (Top row of

Figure 2). Any 2-clique that cannot become a component of a 3-clique is declared maximal and output. When all 3-cliques are generated, the 2-clique list is deleted. The algorithm then attempts to build 4-cliques from the previously constructed set of 3-cliques using the same procedure (Bottom row of Figure 2). This procedure of enumerating maximal cliques is repeated in the increasing order of clique size until it is no longer possible to build a larger clique. This algorithm prevents repetitive generation of non-maximal clique components in the search for maximal cliques, since it once generated, such components are treated as a cohesive unit. This is in direct contrast to the other algorithms discussed in this paper. Unfortunately, the algorithm also has some less than appealing features.

- First, it is evident that building cliques in this manner requires maintaining both the (*k-1*)- and *k*-clique lists. This consumes an enormous amount of space if the in-core implementation, herein called *Kose RAM,* is used. For graphs of any realistic size and density (for the target domain of computational biology), it is not feasible for a typical workstation to keep these lists in main memory. If the lists are stored on disk as with the *Kose Disk* implementation, on the other hand, a tremendous amount of overhead would be incurred from disk I/O operations.
- Second, the algorithm has a hidden cost. Every time a *k*-clique is formed, all (*k-1*)-cliques contained within the new clique must be marked as used; else they might be mistaken for maximal cliques. This cost is not negligible, because it requires a search of the entire (*k-1*)-clique list, which can occupy on the order of 2^n memory cells in the worst case.

5. The *Brute Force* Algorithm

The last method we employ is a brute force algorithm that may seem as the most basic of clique enumeration techniques. It has some useful features, however. The main idea here is to use the primary parameter associated with the clique problem: the clique's size. Employing a user-determined maximal clique size, *k*, this algorithm can eliminate many useless attempts quickly. It applies pre-processing techniques that could reduce the graph size, then enumerates maximal cliques of size exactly *k*. Pre-processing consists of removing all low-degree vertices, that is, vertices whose degree is less than *k-1*. Such vertices are not members of any *k*-clique. Repeating this process may lead to more reductions since a vertex of degree more than *k-1* may lose some of its low degree vertices and in fact could become a low-degree vertex itself.

After preprocessing is applied, the algorithm proceeds by generating all *k*-sets and testing each to determine if it is a maximal clique. Once all *k*-sets have been tested, *k* is decremented by one, and the process is repeated. Although this algorithm was likely to perform poorly in comparison to the others employed, it is chosen for its ability to enumerate maximal cliques in descending order of the clique size, a feature that would be extremely useful in many applications.

6. Applications and Experimental Analysis

One of the holy grails of cellular biology is the elucidation of gene regulatory networks. With these, all life forms utilize cellular components and modulate interactions to carry out specific functions. One of the simplest such networks consists of the set of less than twenty genes and their products that are responsible for regulation of lactose metabolism in the bacterium *Escherichia coli* [10]. However, most networks, particularly those in advanced organisms, are more extensive and can involve hundreds of genes. Until recently, available experimental methods of investigating such networks allowed researchers to observe only a few genes at a time. With such limitations, it took decades to understand even the smallest of networks.

6.1 Clustering versus Maximal Clique Enumeration

In order to comprehend the interactions within and among larger networks, a way to observe the actions of a large number of genes in response to any experimental stimulus is needed. This is now possible with DNA microarrays, which are capable of testing an entire genome (all genes in a cell) simultaneously. Unfortunately, it is not a simple task to interpret such a mass of information, particularly considering the noise inherent in all biological experiments, and particularly in microarray experiments. A first goal in analyzing microarray data in relation to gene regulatory networks is to be able to group genes that exhibit similar responses to series of specific stimuli. This implies that the genes may be co-regulated and therefore acting within the same network.

In this case, clustering must be accomplished *a priori*, as typically there is insufficient knowledge about the system or systems being studied to permit a training phase. This lack of information also makes determining the correctness of the clustering impossible without extensive and time-consuming laboratory experiments to verify the results. Instead, clusters are used for their probative value in order either to generate new hypotheses to be tested or to evaluate those that already exist.

For this application, maximal clique enumeration has three attractive features that are lacking in other popularly used techniques. First, cliques are, by nature, one of the most stringent similarity measures. This affords the advantage that any genes that are members of a clique are highly likely to be co-regulated. This level of stringency does not effectively cope with noise, but the noise issue can be addressed by a variety of methods. Second, maximal clique enumeration permits transcript membership in multiple cliques. This is a significant advantage, because it is common for a gene to participate in multiple networks. Forcing such a gene into one cluster not only loses critical information, but also has the potential to significantly skew subsequent classifications. Finally, it is not necessary to know or be able to infer in advance the expected number of clusters, a value that is rarely available for microarray data. Supplying an incorrect value to an algorithm that required such would clearly invalidate any result.

6.2 Graphs Derived from Microarray Data

The microarray data described in this section was the courtesy of Dr. Robert W. Williams and Dr. Elissa J. Chesler from the Department of Anatomy and Neurobiology of the University of Tennessee in Memphis. The Affymetrix U74Av2 array was used to test 12,422 probe sets in samples from the brain of *Mus musculus* (mouse). Each sample consisted of tissue from three genetically identical mice. One sample was collected from each of the three related recombinant inbred strains of mice, bred such that each strain was a genetic mosaic of the parental strains (*C57BL/6J* and *DBA/2J*). In other words, a gene in one of the recombinant inbred strains has an equal chance of having been inherited from the *C57BL/6J* or *DBA/2J* parental strain. The difference in genetic background of each of the three recombinant inbred strains served as changing experimental conditions. In all other aspects, the samples were treated the same. The experiment was repeated three times and the data pooled.

The raw data from DNA microarray experiments was normalized using the MAS 5.0 (Microarray Suite) software package. Pairwise Spearman's rank coefficients were calculated, resulting in a 12,422 x 12,422 weighted adjacency matrix, where 12,422 was the number of genes measured in the microarray experiment. A threshold of 0.85 was chosen to eliminate all but the highest values. The weighted matrix was filtered using this threshold to produce an unweighted matrix where an edge (*i,j*) is present if and only if the absolute value of the Spearman rank coefficient for (*i,j*) is greater than or equal to the threshold value. A vertex degree histogram of the resulting unweighted graph is shown in Figure 3, in which the X axis lists vertex degrees and the Y axis shows the number of vertices with the corresponding degree.

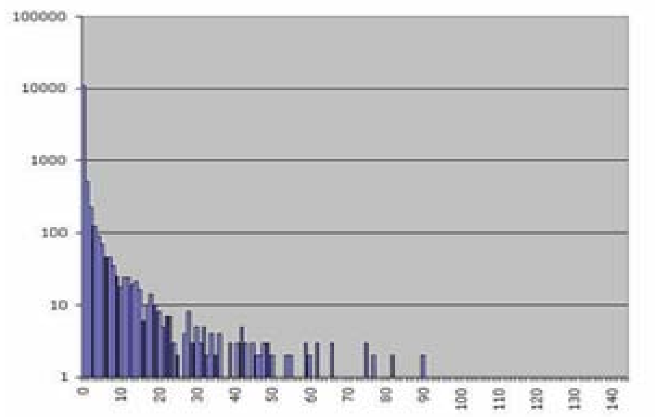

Figure 3. Vertex degree histogram of 0.85 threshold MAS5.0 graph.

Maximal clique enumeration of the unweighted graph discussed above resulted in a total of 5,227 maximal cliques. The maximum clique size was 17, with a user-determined minimum clique size of 3. The distribution of clique sizes generated is shown in Figure 4. There was a tremendous amount of overlap among these cliques, as shown in the clique intersection graph (CIG) in Figure 5. Cliques of size 15 are shown as green vertices, those of size 16 are shown in black, and those of size 17 are shown in red. Two vertices in the CIG are joined by an edge if and only if they have at least 13 vertices in common in the original graph. As indicated by a lack of isolated vertices in the CIG, every clique of size 15 or more overlaps with at least one other clique by more than 76%. Additionally, a very high density region containing the three maximum cliques of size 17, can be observed. Examination of genes occurring most frequently in the intersection of the larger cliques reveals *Veli3* (also known as *Lin7c*), a gene that is crucial to a neurological function [11]; *Sp3* and *Atf2*, members of a nuclear transcription complex active in mouse neural cells [12]; and *Strn3*, a calmodulin binding protein thought to be involved in calcium signaling pathways in mouse neural cells [13].

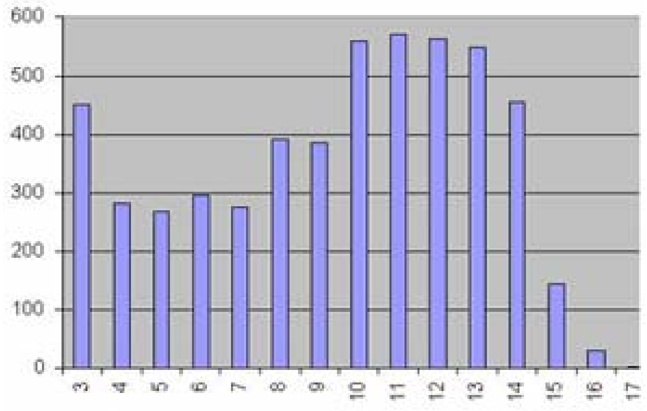

Figure 4. Histogram of clique sizes for 0.85 threshold MAS5.0 graph.

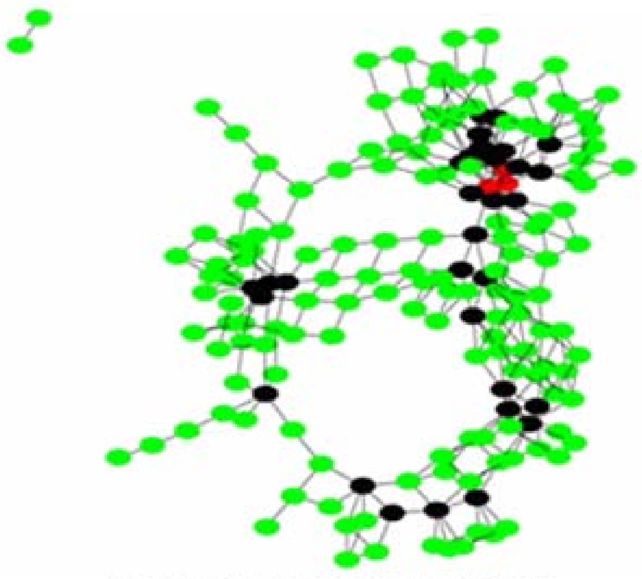
Figure 5. Clique intersection graph for 0.85-threshold MAS5.0 graph.

6.3 Performance Results

The five algorithms described above were implemented and tested. Two versions of the Kose algorithm were used to determine the overhead induced by I/O operations when accessing clique lists, *Kose RAM* and *Kose Disk*. Results are presented in Tables 1 and 2.

	Threshold			
Algorithm	0.95	0.921954446	0.9	0.87
Base BK	6 seconds	halted after 1 day	NA	NA
Improved BK	11 seconds	419 seconds	53220 seconds	halted after 1 day
Kose (RAM)	18632 seconds	halted after 1 day	NA	NA
Kose (Disk)	halted after 1 week	NA	NA	NA
Brute Force	halted after 1 day	NA	NA	NA

Table 1. Performance results for five maximal clique enumeration algorithms applied to the graphs of size 12,422 derived from RMA-processed microarray data of *Mus musculus* for different threshold values.

	Threshold			
Algorithm	0.8	0.80	0.75	0.70
Base BK	6 seconds	193 seconds	halted after 1 day	NA
Improved BK	11 seconds	13 seconds	257 seconds	53470 seconds
Kose (RAM)	17261 seconds	halted after 1 day	NA	NA
Kose (Disk)	halted after 1 week	NA	NA	NA
Brute Force	halted after 1 day	NA	NA	NA

Table 2. Performance results for five maximal clique enumeration algorithms applied to the graphs of size 12,422 derived from MAS5.0-processed microarray data of *Mus musculus* for different threshold values.

The three out of five worst performers are the *Kose RAM*, *Kose Disk*, and *Brute Force* algorithms. The *Brute Force* algorithm was halted after a day on all graphs. The fastest implementation of the *Kose RAM* algorithm that kept its clique lists in core memory finished on the two sparsest graphs, the 0.95 threshold RMA graph (0.0082 average edge density), and the 0.85 threshold MAS 5.0 graph (0.0080 average edge density) in a little over and a little under five hours, respectively. It was not capable of finishing on any other graphs in less than a day. The implementation of the Kose algorithm that stored clique lists on disk (*Kose Disk*) was still running after a week's time on both the 0.95 threshold RMA graph and the 0.85 threshold MAS 5.0 graph.

The *Base BK* algorithm, as anticipated, performed the best on the sparsest graphs. It was nearly twice as fast as the *Improved BK* algorithm. It finished in six seconds as opposed to eleven. However, when challenged with denser graphs, the branch and bound *Improved BK* algorithm was clearly superior to the others tested. It finished the 0.80 threshold MAS 5.0 graph (0.0371 edge density) in thirteen seconds as opposed to the *Base BK* algorithm's 193 seconds, and was the only algorithm capable of finishing the MAS 5.0 graphs with thresholds of 0.75 (0.1178 edge density) or 0.70 (0.2972 edge density). Similar results were seen with the RMA graphs, where only the *Improved BK* algorithm finished the 0.921954446 and 0.90 threshold graphs (edge densities of 0.0743 and 0.2093, respectively) in less than a day. Note that 0.921954446 was the threshold chosen for a recent analysis of the RMA treated data [5]. The *Improved BK* algorithm was unable to finish enumerating all maximal cliques of the 0.87 threshold RMA graph (0.5526 edge density) in less than a day.

7. Conclusions and Ongoing Research

We conclude that, of the existing maximal clique enumeration algorithms tested, the most suited to DNA microarray analysis seems to be the branch and bound *Improved BK* algorithm by Bron and Kerbosch. Although the *Base BK* algorithm by Bron and Kerbosch performed better on very sparse graphs, the branch and bound algorithm was significantly faster on the denser graphs and the loss of a few seconds on sparse graphs is not sufficient to rationalize choosing the base algorithm over the branch and bound algorithm. The Kose's algorithm (*Kose RAM/Disk*), while interesting is not useful for this application. In addition to being more than 1,000 times slower than either *Base BK* or *Improved BK* algorithm at its best, it generates cliques in increasing order. Since, for this application, the desired cliques tend to be large, this confers no advantage. Worse, the fastest implementation of the Kose algorithm has memory requirements that are not likely to be met by most workstations when running graphs of any realistic density. Running this algorithm on the sparsest graphs was only possible with all the other processes except for system software terminated, as it monopolized the available memory. This would only worsen as the graph density increased. Although the *Brute Force* algorithm (based on *k*-set enumeration) was not able to enumerate all maximal cliques within a day on any provided input, the algorithm has an ample opportunity for improvement with the introduction of boundary conditions, such as the ones used in the Bron and Kerbosch algorithm. It is possible that this algorithm could be useful in enumerating cliques when tight size boundaries are imposed.

We are currently investigating the use of better algorithms that are based on preprocessing techniques and on the use of what we call "small dominating structures." The pre-processing rule discussed in Section 5 relies on the use of the clique size as a parameter. This raises the question: what value of *k* should we choose? The maximum clique size may be 20, while we may be trying the *Brute Force* algorithm (for example) starting with $k = 1000$. This motivates us to start by computing the maximum clique size prior to enumeration. To do this, we are currently employing a few techniques to solve the maximum clique problem. The best technique so far is to take the complement of the graph and solve the minimum vertex cover problem on the complement graph. See [14] for details.

Another pre-processing that relies on the parameter *k* is the following. For each edge *uv* in the graph, we compute the common neighborhood, N_{uv} of *u* and *v*. If N_{uv} has fewer than $k-2$ elements, then *uv* is deleted since it cannot be part of a *k*-clique. Repeating this step until no more such edges are found guarantees that we get a graph where any pair of adjacent vertices has at least $k-2$ common neighbors. This is a property that increases the chance of finding maximal cliques of size *k*. Preliminary testing of this pre-processing algorithm shows that it reduces the graph size tremendously. Moreover, the process proves faster when it is interleaved with the low degree-rule mentioned in Section 5. We must admit, however, that applying this algorithm until it cannot reduce the graph further takes a long time on some large graphs. Early testing reveals that this approach is most useful and practical if we to limit it to the first few iterations.

It is easy to see that a clique of a graph *G* is an independent set (i.e., an edgeless set) in *G*'s complement. Therefore, enumeration of maximal independent sets is equivalent to enumeration of maximal cliques. In fact, it has long been known that all maximal independent sets can be enumerated in $O(3^{n/3})$ time, which is the upper bound on the number of possible maximal independent sets (or maximal cliques) in a graph. This algorithm has been modified slightly by considering maximal independent sets whose sizes are bounded above by selected parameters [15, 16]. Of course many graphs do not have these many maximal cliques (or independent sets). One way to detect this property is by looking for small subgraphs, whose complements are nearly edgeless. A vertex cover is a set of vertices whose removal yields an edgeless graph. A small vertex cover, therefore, is an excellent example of a small dominating structure. If we have a k-vertex cover, then we can enumerate all maximal cliques by simply enumerating all cliques of the graph induced by the cover. For each generated clique, only one vertex of the cover's complement can be added to the clique (then we check if it is maximal). Therefore, the maximum number of maximal cliques is bounded above by $(n-k)2^k$ (compare with $O(3^{n/3})$ when $k << n$). Moreover, enumerating all maximal cliques takes $O(2^k)$ in this case.

Vertex cover is not the only potentially small dominating structure. In some cases we do not want to pay the cost of computing a cover prior to enumeration. One alternative is maximal matching in the complement graph. Here we seek to obtain a maximal matching of small size (say k), and to enumerate all maximal independent sets of the complement graph. This takes $O(3^k)$ rather than $O(2^{2k})$ time. To illustrate, note that for each matching edge, only one vertex could be a member of a (potential) maximal independent set. Therefore we have 3 cases per edge (either add the left vertex, or add the right vertex, or none). When the matching size is small, enumeration of all maximal independent sets (or maximal cliques in the original graph) can be much faster than the one by standard algorithms. Preliminary experiments show that our use of maximal matchings is very promising. In some cases, it took only a few minutes to halt while the algorithms of Bron and Kerbosch did not terminate after a few hours. Of course, different graphs have different structures. We aim to have in hand a large suite of radically different sorts of maximal clique enumeration algorithms for future experimentation and testing.

8. References

[1] J. W. Moon and L. Moser., On cliques in graphs, *Israel J. Math*, vol. 3, 1965, 23-28.

[2] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of Combinatorial Optimization*, vol. 4, D.-Z. Du and P. M. Pardalos, Eds.: Kluwer Academic Publishers, 1999).

[3] N. E. Baldwin, R. L. Collins, M. A. Langston, M. R. Leuze, C. T. Symons, and B. H. Voy., High Performance Computational Tools for Motif Discovery, *Proceedings, IEEE International Workshop on High Performance Computational Biology (HiCOMB)*, Santa Fe, New Mexico, 2004.

[4] N. E. Baldwin, E. J. Chesler, S. Kirov, M. A. Langston, J. R. Snoddy, R. W. Williams, and B. Zhang, Computational, Integrative and Comparative Methods for the Elucidation of Genetic Co-Expression Networks, *Journal of Biomedicine and Biotechnology*, 2004, accepted for publication.

[5] E. J. Chesler, L. Lu, S. Shou, Y. Qu, J. Gu, J. Wang, H. C. Hsu, J. D. Mountz, N. E. Baldwin, M. A. Langston, J. B. Hogenesch, D. W. Threadgill, K. F. Manly, and R. W. Williams, Genetic Dissection of Gene Expression Reveals Polygenic Networks Modulating Brain Structure and Function, *Nature Genetics*, vol. accepted for publication, 2005.

[6] F. N. Abu-Khzam, F. Cheetham, F. Dehne, M. A. Langston, S. Pitre, A. Rau-Chaplin, P. Shanbhag, and P. J. Taillon, "ClustalXP," http://ClustalXP.cgmlab.org/.

[7] C. Bron and J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, *Proceedings of the ACM*, vol. 16(9), 1973, 575-577.

[8] F. Kose, W. Weckwerth, T. Linke, and O. Fiehn, Visualizing plant metabolomic correlation networks using clique–metabolite matrices, *Bioinformatics*, vol. 17, 2001, 1198-1208.

[9] E. Tomita, A. Tanaka, and H. Takahashi, The worst-case time complexity for generating all maximal cliques, *Proceedings, Computing and Combinatorics Conference*, Jeju Island, Korea, 2004.

[10] W. S. Reznikoff, The lactose operon-controlling elements: a complex paradigm, *Molecular Microbiology*, vol. 6(17), 1992, 2419-2422.

[11] G. A. C. Becamel, N. Galeotti, E. Demey, P. Jouin, C. Ullmer, A. Dumuis, J. Bockaert, and a. P. Marin., Synaptic multiprotein complexes associated with 5-HT(2C) receptors: a proteomic approach, *EMBO Journal*, vol. 21, 2002, 2332–2342.

[12] L. Cheng, Z. Jin, L. Liu, Y. Yan, T. Li, X. Zhu, and N. Jing, Characterization and promoter analysis of the mouse nestin gene, *FEBS Letters*, vol. 565(1-3), 2004, 195-202.

[13] C. Blondeau, S. Gaillard, J. P. Ternaux, A. Monneron, and A. Baude, Expression and distribution of phocein and members of the striatin family in neurones of rat peripheral ganglia, *Histochem Cell Biology*, vol. 119(2)-8, 2003, 131-138.

[14] F. N. Abu-Khzam, M. A. Langston, and P. Shanbhag, Scalable Parallel Algorithms for Difficult Combinatorial Problems: A Case Study in Optimization, *Proceedings, International Conference on Parallel and Distributed Computing and Systems*, Marina Del Rey, California, November, 2003.

[15] D. Eppstein, Small maximal independent sets and faster exact graph coloring, *Journal of Graph Algorithms and Applications*, (7(2)), 2003, 131-140.

[16] J. M. Nielsen, "On the number of maximal independent sets in a graph," Center for Basic Research in Computer Science (Denmark), 2002.