On the Performance of Polynomial-time CLIQUE Approximation Algorithms on Very Large Graphs *

Steven Homer　　Marcus Peinado

homer@cs.bu.edu　　mpe@cs.bu.edu

Department of Computer Science and
Center for Computational Science
Boston University

Abstract

The performance of a randomized version of the subgraph-exclusion algorithm (called Ramsey) for CLIQUE by Boppana and Halldórsson is studied on very large graphs. We compare the performance of this algorithm with the performance of two common heuristic algorithms, the greedy heuristic and a version of simulated annealing. These algorithms are tested on graphs with up to 10,000 vertices on a workstation and graphs as large as 70,000 vertices on a Connection Machine. Our implementations establish the ability to run clique approximation algorithms on very large graphs. We test our implementations on a variety of different graphs. Our conclusions indicate that on randomly generated graphs minor changes to the distribution can cause dramatic changes in the performance of the heuristic algorithms. The Ramsey algorithm, while not as good as the others for the most common distributions, seems more robust and provides a more even overall performance. In general, and especially on deterministically generated graphs, a combination of simulated annealing with either the Ramsey algorithm or the greedy heuristic seems to perform best. This combined algorithm works particularly well on large Keller and Hamming graphs and has a competitive overall performance on the DIMACS benchmark graphs.

1 Introduction

At the 1993 FCRC Conference Laci Babai concluded his plenary lecture by mentioning a journal editor who, reacting to the recent results on transparent proofs, has added the problem of approximating the maximum clique in a graph to other provably unachievable results like perpetual motion machines on his list of paper topics he will not accept for publication. Nonetheless approximating the maximum

*Research partially supported by the National Science Foundation under grant CCR-9103055.

size clique in a graph is what we do here, as well as we can, in practice. Our results will focus on **very** large graphs, some larger than 50,000 vertices, generated both deterministically and randomly and tested on both a Silicon Graphics workstation and on the Connection Machine CM5. Our main contributions are two-fold: (1) We exhibit the first test data on the Boppana/Halldórsson subgraph exclusion algorithm, a combinatorially interesting algorithm with a non-trivial performance guarantee. (2) We present experiments on the largest graphs yet implemented and tested for approximating maximum size cliques, including Hamming and Keller graphs of practical and theoretical interest [9], [4].

As with many NP-hard optimization problems, the clique problem, has been attacked from two sides. There are exact algorithms which solve the problem to optimality in superpolynomial time. Then there are algorithms which run in polynomial time but return only an approximate solution. In the case of CLIQUE, the range of application of these two types of algorithms depends on the size of the input graph. Currently, the exact algorithms can be expected to find the largest clique in graphs with up to 400 to 1000 vertices. For bigger graphs the running times become prohibitive. It appears natural to use the second kind of algorithms to find approximate solutions to the CLIQUE problem in graphs that have more than 1000 vertices. The approximation we achieve will, by necessity, be asymptotically weak as the recent results on transparent proofs tell us that approximating clique to within a factor of n^c, c fixed, is NP-hard.

Unlike many other NP-hard problems, the CLIQUE problem has for a long time resisted attempts by researchers to construct polynomial-time approximation algorithms with a non-trivial performance guarantee. This was changed recently by the subgraph exclusion algorithm of Boppana and Halldórsson [2]. Its performance guarantee of $O(n/\log^2 n)$ is tight. However, it is not clear if it remains tight for a randomized version of the algorithm which we study. Furthermore, it is not clear how frequently families of graphs for which the performance of the algorithm is poor appear in practice.

The first part of this paper tries to answer the question of whether the theoretical advantage the subgraph-exclusion algorithm has over all other polynomial time heuristics in terms of provable worst-case behavior is matched by an improved performance on common problem instances. In order to answer these questions, we have implemented a randomized version of the subgraph exclusion algorithm. The algorithm always finds a clique, but may find different size cliques depending on the random choices made during the runs of the program. By way of comparison we have implemented a version of simulated annealing which uses a penalty function to measure the quality of solutions and can be found in [8] . And we have implemented the simple greedy heuristic algorithm of always picking nodes of highest degree to attempt to add to our clique as it is being built. This algorithm can be found, for example, in [10] . All three algorithms are similar in terms of their running time (they can be made to run very fast) and the fact that they give no, or only very weak, guarantees for their worst case behaviors.

Since the algorithms are very fast, they allow us to solve much larger input graphs than most algorithms which are more sophisticated but have a longer running time. On a workstation, the algorithms run on 10,000 vertex input graphs in less than 4 minutes. In the second part of this paper, we take this to the limit and explore how far (in terms of graph sizes) one can go with the hardware currently available. We describe a Connection Machine implementation of the subgraph-exclusion algorithm and simulated annealing and describe experiments on input graphs with up to 70,000 vertices.

The next section presents the three algorithms we use and the five types of graphs on which they are tested. Section 3 gives the results for the experiments using an SGI workstation and Section 4 the results on the CM5. Finally we present some conclusions and further work. The appendix describes the performance of our programs on the DIMACS benchmark graphs.

2 The Algorithms and the Graphs

The Greedy Heuristic

In this work we consider three algorithms. The simplest is a greedy heuristic algorithm.

greedy(G):
  IF G is empty THEN return $\emptyset$
  ELSE choose a vertex v of highest degree
    return $\{v\} \cup$ greedy($neighborhood(v)$)

The algorithm starts by selecting a vertex (called **pivot** vertex) of highest degree in the graph for the clique and deleting it and all of its non-neighbors from the graph. This process is repeated until all of the vertices have been deleted. The algorithm dates back to Johnson[7] and has been used in several applications, for example in the work of Lecky, Murphy and Absher[10] on the PLA folding problem. It works well on random graphs and some graphs with large cliques but it is trivial to construct graphs where this algorithm performs arbitrarily badly.

The Ramsey Algorithm

The algorithm is due to Boppana and Halldórsson. Our presentation of it follows the one given in [3], [2].

The algorithm consists of a subgraph exclusion procedure and a recursive subprocedure (Ramsey) which is motivated by Ramsey theory and which, given an input graph, returns a clique and an independent set. The subgraph exclusion procedure calls Ramsey, stores the clique returned and removes the independent set from the graph. This is repeated until the graph has become empty.

The Ramsey subprocedure improves and generalizes the greedy method by making an additional call to search the non-neighborhood of the pivot vertex. Thus, each recursive call has two cliques to chose from: the clique found in the neighborhood of the pivot together with the pivot and the clique found in the non-neighborhood. Ramsey returns the larger one.

Clearly, the same idea can be used to find an independent set by interchanging the terms neighborhood and non-neighborhood. Ramsey returns both an independent set and a clique in the input graph.

```
Ramsey((V, E)):
        IF (V, E) is empty THEN return (∅, ∅)
        ELSE choose a vertex v ∈ V
            (C_1, I_1) := Ramsey(N(v))
            (C_2, I_2) := Ramsey(N̄(v))
            return (larger of (C_1 ∪ {v}, C_2), larger of(I_1, I_2 ∪ {v}))
```

Using Ramsey theory, Boppana and Halldórsson [3] show for the clique C and independent set I returned by Ramsey(G) that $|C| \cdot |I| \geq \log^2 n/4$. This bound in itself does not guarantee a minimum size of C since $|I|$ can be large.

The purpose of the subgraph exclusion algorithm is to modify the graph such that, eventually, $|I|$ will be small. This is achieved by repeatedly calling Ramsey and excluding (removing) the returned independent sets:

```
IS Removal(G):
        i := 1
        (C_i, I_i) := Ramsey(G)
        WHILE G ≠ ∅
            G := G \ I_i
            i := i + 1
            (C_i, I_i) := Ramsey(G)
        return max_{j≤i} C_j
```

A clique in G can loose at most one vertex per iteration because a clique and an independent set can share at most one vertex. If the graph has a large enough clique, a constant fraction of the graph will be left even if all independent sets of a certain minimum size k are excluded. If Ramsey is run on the resulting graph, the size of I can be at most k. This implies a lower bound on $|C| \geq \log^2 /(4k)$. If the largest clique is small, the performance of the algorithm on the graph if trivially good. The result of this analysis is a performance guarantee of $O(n/\log^2 n)$ (cf. [3] for details).

It is easy to see that the performance of the algorithm depends on the sequence of pivot nodes chosen during the procedure. In the Boppana/Halldórsson paper [3] the pivot nodes are chosen deterministically in an arbitrary manner and the worst case behavior is analyzed. The analysis establishes a performance guarantee and shows nonconstructively that for certain graphs and particular sequences of pivot nodes the algorithm performs no better than this guarantee. In the version we implement we choose the pivots randomly. By running the algorithm repeatedly using different sequences of pivot nodes we hope to achieve better performance. In fact, we have found this to be the case for certain classes of graphs. It is not clear that the performance bounds given in [2] remain tight for this version of the algorithm. Peinado [11] investigates the construction of graphs which are hard for this randomized subgraph exclusion algorithm.

Simulated Annealing

There is a large body of literature on the general principle of simulated annealing and its application to particular optimization problems (e.g. [1]). The application of simulated annealing to the CLIQUE problem is described in [1, p. 81]. We use the penalty function approach described there: The current *state* of the algorithm is given by any subset V (not necessarily a clique) of the vertex set of the graph. The quality of each state V is determined by the cost function $f(V) = |V| - \lambda|E|$ where E is the number of edges that are missing in the subgraph induced by V. Thus, the quality of a solution is the number of vertices in it reduced by a weighted penalty for the number of edges that are missing. The allowed state changes are adding to V or deleting from V a randomly chosen vertex. The penalty function approach to the GRAPH COLORING has been studied in Johnson, et al [8].

The cooling schedule is simple: During the first 25% of the annealing time, the temperature parameter T is reduced linearly from 1 to 0.5. During the remaining 75% of the annealing steps T is reduced linearly from 0.5 to 0. As opposed to the standard version of the penalty function approach, the parameter λ is not kept constant but slowly increased from initially 0.7 to 1.2. Thus, the penalty for missing edges is less severe initially than in the end. The intention is to facilitate the discovery of new solutions in the beginning and to force a clique (or almost clique) solution at the end. If the algorithm finds a solution with missing edges, a clique solution can be found by removing at most one vertex for each missing edge.

The Graphs

We have tested our algorithms on the following five types of graphs.

1. Random Graphs: The standard random graph model in which each edge is chosen independently with probability p. We have concentrated on the most common case $p = 0.5$. $G_{n,p}$ denotes the class of random graphs with n vertices and edge probability p.
2. Random Graphs with big cliques: Like random graphs. However, a subset of size l of the vertices is forced to be a clique. The parameters are the number of vertices n, the edge probability p and the clique size l, and the graphs are denoted by $G_{n,p,l}$. We have concentrated on the case $p = 0.5$ and $l = n^{\alpha}$ for various $\alpha \in (0, 1)$. This kind of graph was motivated by the fact that the maximum clique size in almost every random graph is only $O(\log n)$.
3. Random Graphs with big cliques and adjusted edge probabilities: Like the previous case. However, the probabilities of the edges between vertices in the large built-in clique and the remaining graph are adjusted so that the expected degree of the vertices of the large clique and the expected degree of the other vertices are equal. The motivation for considering this type of graph is explained below.
4. Graphs that are related to Keller's conjecture [9].
5. Hamming graphs [4]. We have concentrated on the case in which the required Hamming distance is 4.

The first three examples were chosen to demonstrate the behavior of the algorithm on graphs with certain well known properties. The last two examples are motivated by practical problems. The graph classes 1,2,4, and 5 are popular test instances. We have chosen them in order to be able to compare our results with those of other researchers. However, great caution should be used when making claims about the *general* or "in practice" performance of algorithms based on very few classes of graphs – especially, if a class of graphs has special properties which can make finding large cliques in it trivial. In this case, it should be asked if this class of graphs is a good indicator for an algorithm's general performance in applications.

One example to be considered is class 2 with $l > \sqrt{n}$ and $p = 0.5$. It is easy to prove that for almost every graph in this class, the vertices of the built-in clique have the largest degrees. Therefore, the largest clique can be found by simply determining the degree of each vertex. In particular, the greedy heuristic finds the largest clique in almost every graph in this class. It appears that simple properties, like the one just described, can distort the real performance of an algorithm. In order to show this, we have tested class 3. It should be noted that the differences between classes 2 and 3 are minimal, yet sufficient to alter the expected degrees of the clique vertices enough so that they are no longer larger than the degrees of all other vertices. The result of this minor change is a complete breakdown of the greedy heuristic (see Section 3). We emphasize that class 3 is not a theoretical construction with the purpose of showing the worst case performance of a particular algorithm, but rather a small perturbation of a standard test case. The sensitivity of heuristics like the greedy heuristic and simulated annealing to minor changes in the graph structure indicates a serious lack of robustness of those heuristics. This problem seems to be less severe with the Ramsey algorithm, and in the sense of the (weak) performance guarantee is provably so.

3 Sequential Implementations

We have experimented on very large instances of the graphs described in the previous section on a Silicon Graphics Indigo workstation. The goal was to gather data which allow a direct comparison of the three algorithms, rather than tune the parameters to achieve optimal performance. In our test runs we have varied the graph size between 1000 and 10,000 vertices. On the workstation the running time of the subgraph exclusion algorithm is less than four minutes for graphs with 10,000 vertices. The time increases with n^2 where n is the number of vertices, i.e. it is linear in the size of the input. For comparison we have chosen the running times for simulated annealing to roughly match those of the other two algorithms. Storing and reading the graphs from a file turned out to be a major bottleneck in our programs. Therefore, we used a very compressed graph format, which stores one vertex per bit.

We started our experiments with random graphs $G_{n,p}$ because this class has been well studied both theoretically and experimentally. Figure 1a shows the results of our experiments with the three algorithms on random graphs with edge probability $p = 0.5$.

In general, unless otherwise stated, if the graph had a random component, we took the average over runs on five different graphs and rounded to the next integer. For the standard random graphs, this procedure was insignificant since the clique sizes returned by the algorithms were almost constant. As expected from theoretical considerations, as the graph size increases, the clique sizes found by all three algorithms approaches $\log n$. To do significantly better would require a more com-

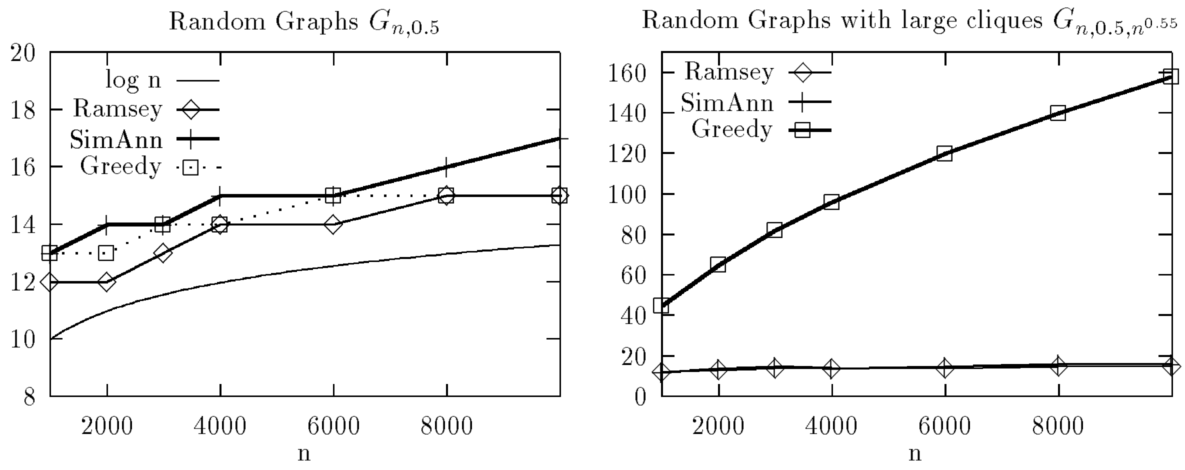

Figure 1: Purely random graphs (left) and random graphs with big cliques (right): $p = 0.5$, $l(n) = n^{0.55}$. Horizontal: graph size in vertices; Vertical: clique sizes found by the algorithms.

plicated heuristic (like GRASP[5]) and a much longer running time which would be prohibitive for the larger graphs.

One of the main reasons why experiments with random graphs $G_{n,p}$ are of limited interest is that in almost every graph the clique size is very small ($O(\log n)$). The natural next step which is usually taken is to artificially build a larger clique of size $l = l(n)$ into an otherwise random graph (the graphs $G_{n,p,l}$). We have tested the algorithms on graphs $G \in G_{n,p,l}$ for several dependencies between l and n. The easiest case is a large clique of size $l = n^c$ where c is close to 1. As the clique size l is decreased, for each of the algorithms there is a point at which the performance drops to sizes around $\log n$. For $l(n) > n^{0.8}$ the greedy heuristic and simulated annealing find the largest clique. The clique sizes found in different runs of the Ramsey algorithm on the same graph vary by large factors. This suggests that, as opposed to the case of pure random graphs, it can help to run the algorithm several times. Usually, we find a clique which comes to within a few vertices of the optimum in less than ten runs. Around $l = n^{0.7}$, the clique sizes found by `Ramsey` drop to $\log n$. Simulated annealing and the greedy heuristic exhibit the same phenomenon at about $l = n^{0.6}$ and $l = \sqrt{n}$, respectively.

Figure 1b shows the behavior of the three algorithms for $G \in G_{n,0.5,l}$ where $l = n^{0.55}$. The behavior of the greedy heuristic has the simple explanation which has already been mentioned in the previous section: the fact that for $l > \sqrt{n}$, the largest clique can be found simply by calculating the degrees of the vertices. In order to remove this artifact, we have designed the class $P_{n,p,l}$, which is similar to $G_{n,p,l}$, except that the edge probabilities between the clique and non-clique vertices are reduced so that the expected degree of all vertices is the same. Figure 2a shows the behavior of the three algorithms for $G \in P_{n,0.5,l}$ and $l = n^{0.65}$.

As a result of the minor change in the probabilities of relatively few edges, the greedy heuristic breaks down completely. At the same time, the changes in the performances of the Ramsey algorithm and simulated annealing are relatively

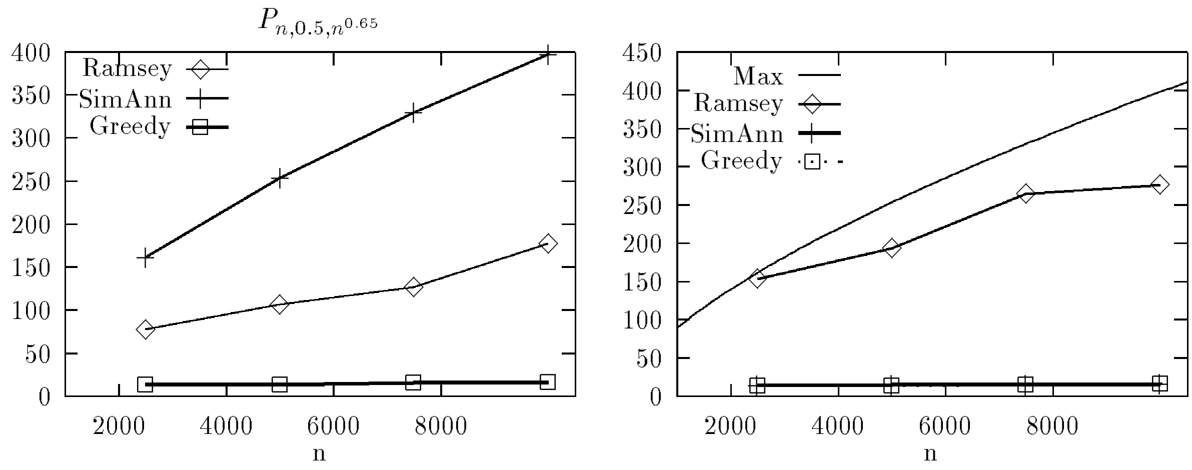

Figure 2: left: edge probabilities are adjusted so that the expected degrees of all vertices are the same. right: edge probability $p = 0.3$ for edges between clique and non-clique vertices. The values for Ramsey and simulated annealing are the maxima over ten runs averaged over four different graphs. Max indicates the size of the largest clique in the graph.

small. For the parameter choice of figure 2, the performance improves. In general, depending on the size of l, the performance can improve or deteriorate slightly. This is due to the interplay of the recursive subprocedure – whose performance improves – and the subgraph exclusion procedure – whose performance deteriorates as the clique-nonclique edge probabilities are reduced.

The question lies near. How will the behavior of the algorithms change if the probabilities of the edges between the vertices of the large clique and the rest of the graph are reduced even further? Figure 2b shows the result for the case in which this probability has been reduced to 0.3. Now, simulated annealing has broken down as well, while the performance of the Ramsey algorithm has improved significantly. It should be noted that as this probability is reduced, the large clique again becomes easier to find, because it becomes isolated from the rest of the graph. A simple analysis of the Ramsey algorithm shows that it takes advantage of this kind of clue and performs well on such graphs.

Finally, Figure 3 shows the results of our experiments with Keller and Hamming graphs. These graphs are at least partially motivated by practical considerations. The greedy heuristic appears to perform best on these graphs. Due to the structure of these graphs, there is only a relatively small number of them with an appropriate number of vertices. We will present experimental results on Keller and Hamming graphs with more than 50,000 vertices in Section 4.

There are many interesting classes of graphs. However, any experimental paper has to be confined to a small subset of them. We have presented experiments on a small selection of popular graphs and a class of graph which we have designed in order to demonstrate the different behaviors of the algorithms, in particular the Ramsey algorithm. As should have been expected, there is no single algorithm which performs best on all graphs. It appears that many graphs – at least many

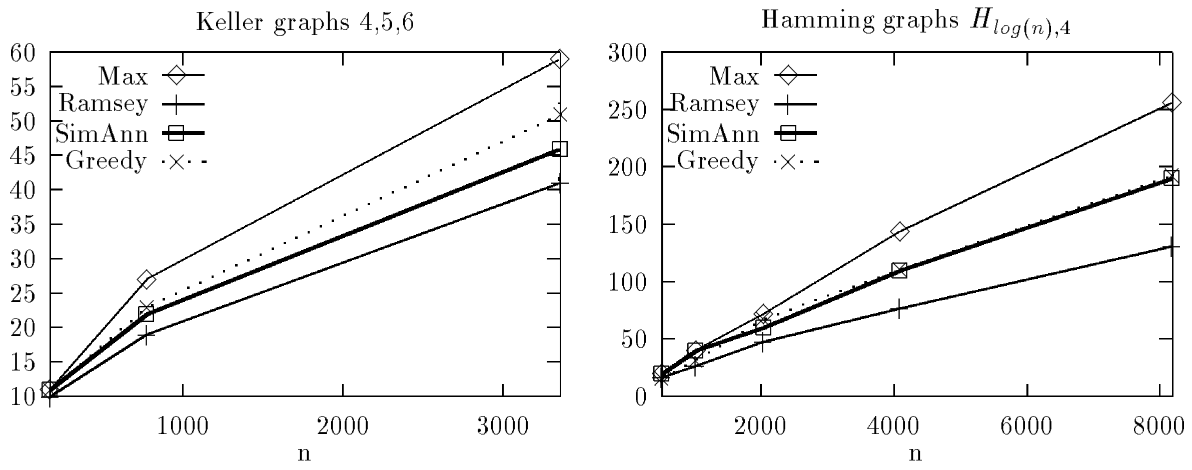

Figure 3: left: results for keller graphs of size 4,5,6. right: results for Hamming graphs of size 9,10,11,12,13. Max is the size of the largest clique in the graph.

of the popular test instances – reveal at least parts of their large cliques by the degrees of their vertices. The success, of the greedy heuristic gives evidence for this. On the other hand, it is trivial to find graphs on which the greedy heuristic breaks down completely. The Ramsey algorithm can be considered as a generalization of the greedy heuristic, if the pivot vertices are selected according to the maximum degree instead of at random. We have run tests with this combined greedy and Ramsey algorithm. Since it performs a superset of the computation done by the greedy-heuristic, the solutions it returns are at least as good as those of the greedy heuristic. In certain cases (e.g. $G_{n,0.5,l}$ for small l) a significant improvement over both the random version of the Ramsey algorithm and the greedy heuristic can be observed. However, the price to be paid is running time. Finding the vertex with the largest degree is a relatively expensive ($O(n^2)$) operation. If the Ramsey algorithm is combined with the greedy heuristic, its running time increases to $O(n^4)$ and it becomes very slow for $n > 4000$ vertices.

4 Parallel Implementations

The main obstacle to extending the results obtained on a workstation beyond graphs with 10,000 vertices is a lack of main memory. It should be noted that since the Ramsey algorithm and simulated annealing choose vertices (and thereby the positions they read in the adjacency matrix) at random, their accesses to the adjacency matrix have little or no spatial and temporal locality. A similar observation can be made about the greedy heuristic – at least when the input graphs are random. As a result, the program will thrash as soon as the adjacency matrix becomes too large to fit into main memory.

The Connection Machine (CM5), provides not only parallelism but also a large amount of random access memory, which is partitioned among the processors. More specifically, the 32 node partition we used in which each node had 32 MBytes of

RAM, provides a total of 1 Gigabyte, which is currently beyond the capacity of most workstations. This amount of memory allowed us to run simulated annealing and Ramsey on graphs with up to 70,000 vertices. Our implementations use the CMMD communication library. We did not implement the greedy heuristic because we did not expect any new insights from doing so at this stage of the experiments: After some preliminary experiments on the workstation, it became clear the that on the graph classes we wanted to test on the CM5, simulated annealing, given somewhat more time, would easily find larger cliques than the greedy heuristic. Furthermore, parallelizing the greedy heuristic does not pose any challenges not present in the parallelization of Ramsey or simulated annealing.

Since the memory is partitioned among the processors, the adjacency matrix had to be partitioned among them. We chose to divide the matrix into blocks of rows and stored each block at a single processor of the CM-5. The approach of storing entire rows at each processor was guided by the observation that for our implementation of simulated annealing and for Ramsey, the 'innermost' operations on the adjacency matrix involve looking at the edges which are incident on one particular vertex. Thus, we obtain a natural SIMD parallelization of the innermost operations: Each processor is responsible for the part of the 'innermost' operation which involves the vertices that correspond to the local rows. This task can be performed in parallel and without communication by operating on the local rows. The structure of the parallelizations thus obtained is very similar to that of the original algorithm.

In the case of simulated annealing, the 'innermost' operation involves counting how many edges are missing between the vertex which is about to be moved into or out of the current state and all other vertices in the state. This operation is performed in parallel. The processors have to be synchronized at two points: Firstly, all processors have to agree on the vertex to be removed from or inserted into the current state. This can be achieved without communication by initializing the random number generator at each processor to the same initial seed. After a vertex has been selected, each processor performs the 'innermost' operation on its local vertices, i.e. counts how many edges are missing between the selected vertex and the local vertices which are in the state. The second point of synchronization is the summation of the local edge counts and the distribution of the sum to all processors. All other parts of the algorithm are performed identically by each processor, i.e. without speedup.

In order to determine the efficiency of the parallelization, it should be noted that all parts of a single annealing step except the edge counting require constant time. The time needed for the edge counting operation is proportional to size of the current state, and thus to the size of the clique the algorithm will find. For most common classes of graphs, this size increases fairly rapidly as the graph size increases. Thus, for a constant number p of processors, the speedup should approach p as the graph sizes increase. Figure 4 shows that the speedups we achieve on the larger Hamming graphs are almost perfect.

In the case of the Ramsey algorithm, the 'innermost' operation involves determining the neighborhood or non-neighborhood of the pivot vertex. The first step in each recursive call is to determine the pivot vertex. This requires communication among all processors. A designated coordinator processor coordinates the vertex

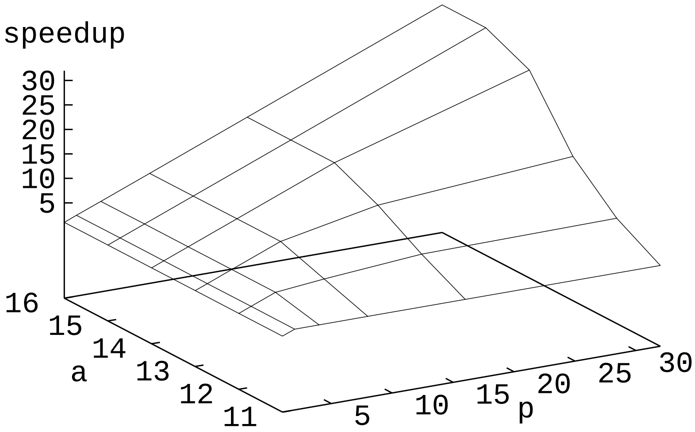

Figure 4: Speedups of our parallelization of simulated annealing on different Hamming graphs ($H_{a,4}$) as the number p of processors is varied between 1 and 32. The size parameter a is varied between 11 (2048 vertices) and 16 (65536 vertices). As the graphs become larger, the speedup becomes almost perfect (i.e. linear).

selection. After that, all processors but the coordinator compute the local neighborhood of the pivot vertex, i.e. those local vertices which are neighbors of the pivot. Then a recursive call is made. The same steps are repeated for the non-neighborhood. Finally, the coordinator computes the clique and independent set to be returned. In this implementation, communication constitutes a considerable overhead (depending on the input graph and the number of processors used, between 30% and 60% for the subgraph exclusion algorithm). Although the running times (about 3 hours for 60000 vertex graphs) are still acceptable, we are exploring ways to reduce the communication overhead.

The relatively high communication cost can be explained as follows: The speedup achievable depends on the size of the input graph to the recursive call. At the deeper levels of the recursion tree, the input graphs become increasingly small. In particular, for random graphs, the size of the input graph is approximately cut in half from one recursive level to the next. Hence, at depth d, the expected size of the input graph is $n/2^d$. At the same time, the expected number of recursive calls at depth d is 2^d. Hence, the recursive calls with small input graphs and thus small potential for speedup far outnumber the recursive calls with larger input graphs.

One approach to this problem is to execute the algorithm described above until the input graph of a recursive call becomes too small and then switch to a second kind of parallelization in which each processor has the adjacency matrix of the current (small) input graph. Each processor is responsible for the computation in one particular recursive subtree. While the computation of the subtrees yields optimal speedup, it requires the corresponding adjacency matrix to be derived from the original adjacency matrix and to be distributed (communication). The communication cost might be minimized by combining the two approaches appropriately.

The potential speedup of both of our parallel programs depends critically on the average clique size in the input graph. In this sense, random graphs are the 'worst case' input because they contain only cliques of size $O(\log n)$. The parallelization itself is a direct consequence of the partitioning of the adjacency matrix among the nodes which, in turn, is a consequence of our goal to run tests on very large graphs. Much better speedups are easily achievable by abandoning this goal and keeping a local copy of the entire adjacency matrix at each node.

It should be noted that in dealing with graphs of the sizes we are considering storing the adjacency matrix and the running time are not the only problems. A further obstacle is the generation of these graphs. The standard approach of storing the graph in a file which is then read by the program is no longer an option due to its size (hundreds of megabytes). Therefore, we decided to build the generators into our programs, thus, not only avoiding the disk as a bottleneck, but also making use of the possibility of speeding up the graph generation by doing it in parallel.

Generating graphs in parallel is straightforward and leads to linear speedups for graphs like Keller graphs or Hamming graphs, for which a boolean function can be defined that decides the existence of an edge based solely on the two input vertices. The generation of graphs with a random component, on the other hand, is more complex. Not all matrix elements can be determined at random since the condition $a_{ij} = a_{ji}$ has to hold for all matrix elements. We have solved this problem by first generating the part of the matrix above the diagonal in parallel and then sending segments from above the diagonal to the processors responsible for the corresponding segments below the diagonal. All this is done in parallel.

So far, we have implemented generators for Keller graphs, Hamming graphs, random graphs and random graphs with large cliques. The times necessary to generate these graph lie between 4 and 10 minutes for graph sizes of about 70,000 vertices. These times are small compared to the times needed by the programs that find cliques.

We have run the Ramsey algorithm and simulated annealing on the four graph classes mentioned above for graph sizes of up to 70,000 vertices. The cooling schedule used for simulated annealing differs slightly from the one used in the sequential implementation: The temperature is reduced linearly from 0.35 to zero. At the same time, the penalty parameter λ is increased linearly from 0.75 to 1.0. The running time and the performance of the simulated annealing algorithm depends on how many iterations we allow it. The Ramsey algorithm depends on no such parameter. We have tried to improve its performance by running it repeatedly on the same input and taking the largest clique found in all runs. However, this method is not successful on the larger Hamming and Keller graphs: Even large numbers of iterations of Ramsey lead only to slightly larger clique sizes. The results for the Keller and Hamming graphs are displayed in Table 4. They show a very good performance of simulated annealing on Keller graphs. The large differences in the running times of Ramsey on Keller graphs and Hamming graphs show that the running time also depends on properties of the input graph. The recursive subprocedure of Ramsey finds much larger cliques in Keller graphs than in the

Keller graphs on the CM5				
size	nr vert	max clique	Simulated Annealing	Ramsey
4	71	11	11 (in 12 sec)	10 (in .18 sec)
5	776	27	27 (in 1:28 hours)	23 (in 1.8 sec)
6	3361	59..63	59 (in 8:00 hours)	43 (in 15.1 sec)
7	14190	123..127	119 (in 20:00 hours)	83 (in 2.6 min)
8	58967	???	229 (in 2 days)	153 (in 22 min)

Hamming graphs on the CM5				
size	nr vert	max clique	Simulated Annealing	Ramsey
13	8192	256	202 (in 3 min)	123 (in 3 min)
14	16384	512	378 (in 4 hours)	220 (in 12 min)
15	32768	1024	683 (in 8 hours)	417 (in 46.8 min)
16	65536	2048	1199 (in 3:25 hours)	674 (in 3:22 hours)

Table 1: Keller and Hamming graphs on the CM5. The third column contains the size of the maximum clique (as far as it is known) in the graph. The fourth and fifth columns display the best results achieved by simulated annealing and Ramsey, respectively. Simulated annealing outperforms Ramsey and does particularly well on Keller graphs.

other graphs we have considered. Consequently, the graph is exhausted after fewer iterations of the subgraph-exclusion loop of the algorithm.

In an attempt to find larger cliques, we set the initial state of simulated annealing to be the largest clique we had found in the previous experiments in the given input graph (e.g. Hamming 14) and run it at a fixed temperature for a relatively long time (1 to 2 days). This temperature was chosen high enough to allow relatively frequent state changes and low enough to ensure that sufficiently often the state would be about as large as the largest known (initial) state. This temperature was determined by trial and error. During the experiment, simulated annealing entered a large number of largely different states of the size reported in table 4 but did not find a single larger state. This indicates that the number of cliques declines drastically as the maximum clique size is approached. Versions of this idea have been used in other contexts to prove negative results about simulated annealing [6], [12], [13]. While we cannot prove our observation in any exact sense, it seems to indicate that Hamming and Keller graphs become exceedingly hard as their size increases.

The Connection Machine has also enabled us to run our algorithms on much larger random graphs and random graphs with large embedded cliques. The results of these experiments confirm the trends we have have reported in the previous section for graphs with up to 10,000 vertices. For random graphs (each edge having probability 0.5) we have run both the Ramsey algorithm and simulated annealing on graphs from 20,000 to 50,000 vertices. The running time for these trials is three to four hours. Both algorithms find cliques of about the same size (slightly larger than $\log n$), with simulated annealing doing a bit better.

Similarly, we were able to confirm the observations made on the workstation for smaller random graphs with large embedded cliques. The largest graph tested had 70,000 vertices and an embedded clique of size 7518. As expected, simulated annealing found this large clique exactly within 3 hours.

5 Conclusions

We have conducted three groups of experiments with the three algorithms. With the first group of experiments we have explored properties of the randomized Ramsey algorithm and compared it with simulated annealing and the greedy heuristic. Ramsey performs particularly well when the large cliques are relatively isolated from the rest of the graph. This isolation corresponds to a somewhat lower edge density between the clique in question and the rest of the graph compared to the overall density of the graph. The other two algorithms tend to have a weak performance on graphs with this property. However, on graphs without this property, i.e. most popular test instances, simulated annealing and the greedy heuristic tend to perform better whereas the performance of Ramsey deteriorates. In this sense, the three algorithms complement each other.

The first group of experiments was carried out on a workstation and thus limited to graphs with at most 10000 vertices. The second group of experiments was carried out with the goal of being able to handle much larger graphs. We parallelized Ramsey and simulated annealing and implemented them on a Connection Machine CM5. This has allowed us to test graphs with as many as 70,000 vertices. In addition to continuing on the larger graphs the comparative experiments begun on the workstation, we also made finding the large cliques in the larger Keller and Hamming graphs one of our objectives. Confirming the trends observed in the first group of experiments, simulated annealing outperformed Ramsey. This was also due to the fact that our parallelization of simulated annealing achieves near perfect (linear) speedup on the larger graphs. Our implementation of Ramsey had a relatively large communication overhead due to the less uniform structure of the computation.

Simulated annealing is particularly successful on Keller graphs. Within reasonable time (considering the graph sizes), simulated annealing finds a clique of the largest known size (59) in the Keller 6 graph and comes to within few vertices of the optimum on the much larger Keller 7 and Keller 8 graphs, heretofore untested.

Finally, we have run the algorithms on a workstation on the DIMACS benchmark graphs. In order to optimize performance, we tuned the parameters and combined the algorithms in order to take advantage of their complementary strengths. The tuning of the parameters was limited to finding an appropriate range for the penalty parameter λ and annealing only at lower temperatures (cf. appendix) because we had observed that annealing at very high temperatures (> 0.4) can be omitted without jeopardizing the final result.

The strength of simulated annealing are relatively **dense graphs** with edge densities between 0.7 and 0.95. It outperforms the reference algorithm `dmclique` on most benchmark graphs whose density is in this range (table 2). The typical running time vs. clique size trade behavior of simulated annealing is displayed in figure 5.

graph	opt	SA		dmclique	
		size	time	size	adj. time
hamming10-4	??	40	25sec	40	31min
keller6	59...63	59	20 hours	55	2.6 days
p_hat500-3	??	50	17sec	49	79sec
p_hat700-3	??	62	10sec	62	18min
p_hat1000-3	??	68	47sec	65	1.2hours
p_hat1500-3	??	94	93sec	93	3.6hours

Table 2: A small selection of the benchmarks on which simulated annealing (SA) appeared to perform well are directly compared to `dmclique`. For simulated annealing, we report the largest clique size found over 500 runs on the same graph. The corresponding running time t_{MAX} is the total running time over all 500 runs on one graph divided by the number of times the maximum was found. Thus, t_{MAX} can be interpreted as the average time needed to find a clique of the reported size. The data for `dmclique` were computed in the same way based on the data provided in the file `results.dmclique` provided by DIMACS. In addition, the running times of `dmclique` were multiplied by 4.3 – the speedup factor of the DIMACS reference machine w.r.t. our machine.

Clearly, the greedy heuristic performs well if the vertices in large cliques have relatively large degrees. In most graphs, this holds only to a limited degree. However, in graphs which have this property (e.g. Steiner triple graphs `MANNxx`, random graphs with large embedded cliques), the greedy heuristic finds an optimal or near optimal solution extremely fast.

The Ramsey algorithm benefits if the overall edge density differs from the density of the edges connecting a large clique to the rest of the graph. Since its subgraph exclusion procedure does not perform hill climbing, it is less influenced by small local minima than simulated annealing. In particular, it can be combined with simulated annealing to help simulated annealing get beyond certain local minima. However, since the current DIMACS benchmark database does not contain graphs which necessitate the use of Ramsey, we did not include it in the experiments reported in the appendix.

References

[1] E.H.L. Aarts and J.H.M Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, Chichester, U.K., 1989.

[2] R. Boppana and M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. In *SWAT*, pages 11–25. Springer Verlag, 1990.

[3] R. Boppana and M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32:180–196, 1992.

[4] A.E. Brouwer, J.B. Shearer, N.J.A. Sloane and W.D. Smith. A New Table of Constant Weight Codes. *J. IEEE Trans. Information Theory*, 36: 1334–1380, 1990.

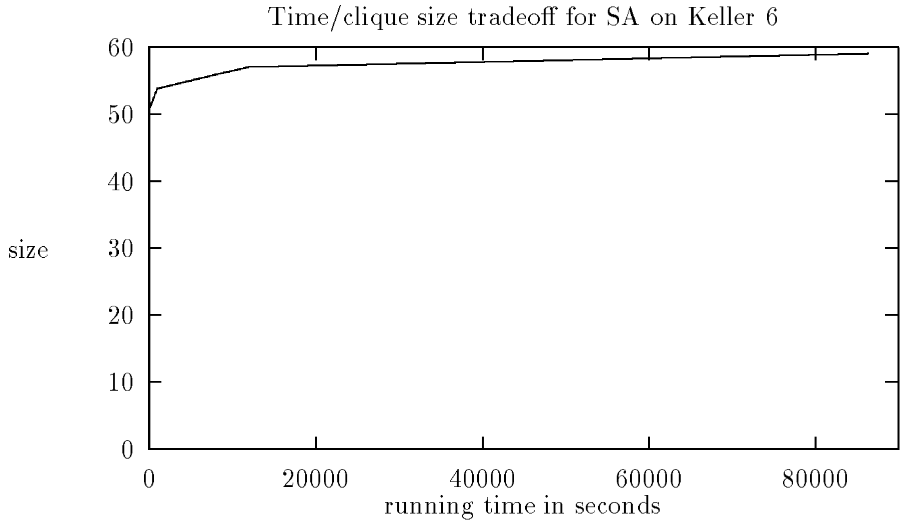

Figure 5: Clique size – running time trade off for simulated annealing on the Keller 6 graph.

[5] T. A. Feo, M.G.C. Resende and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. Manuscript, 1992.

[6] Mark Jerrum. Large cliques elude the metropolis process. *Random Structures and Algorithms*, 3(4):347–360, 1992.

[7] D. S. Johnson. Approximation algorithms for combinatorial problems. *JCSS*, no. 9, pages 256-278, 1974.

[8] D. S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon. Optimization by simulated annealing: An experimental evaluation; Part II, graph coloring and number partitioning *Operations Research*, Vol. 39, no. 1, 1991.

[9] J.C. Lagarias and P.W. Shor. Keller's cube-packing conjecture is false in high dimensions. *Bull. AMS (New Series)*, Vol. 27, no. 2, pages 279-283, 1992.

[10] J.E. Lecky, O.J. Murphy and R.G. Absher. Graph theoretic algorithms for the PLA folding problem. *IEEE Trans. on Comp.-Aided Design*, Vol. 8, no. 9, pages 1014-1021, 1989.

[11] M. Peinado. Hard graphs for randomized subgraph exclusion algorithms. Manuscript, in preparation.

[12] G. H. Sasaki and B. Hajek. The time complexity of maximum matching by simulated annealing. J. ACM, Vol. 35, pages 387–403, 1988.

[13] G. H. Sasaki. The effect of the density of states on the Metropolis algorithm. Inf. Process. Lett., 37, pages 159–163, 1991.