

## Constructing Cliques Using Restricted Backtracking

MARK K. GOLDBERG

REID D. RIVENBURGH

ABSTRACT. The restricted backtracking algorithmic paradigm is applied to the Maximum Clique Problem. The notion of backtracking coordinates is introduced. The program searches for those cliques whose backtracking coordinates are bounded by the values given in the input.

### 1. Introduction and motivation

The problem of constructing a clique of the maximum size in a given graph, *the clique problem*, was proved to be NP-hard in [10] (see also [8] and [13] for a survey on the problem). Recently, it was proved ([1]) that even constructing a clique whose size is within a factor of  $n^\epsilon$  of the maximum is NP-hard (here  $n$  is the vertex number and  $\epsilon > 0$ ). Nevertheless, we argue that these results do not necessarily imply the intractability of the problem “in practice.” The essential difference between the theoretical model and the way the problem occurs in applications is as follows:

- the size of the graph which can be stored in the memory of any computer is bounded; it cannot exceed  $10^{40}$ —the estimated number of particles in the known universe—but obviously, the “real-life” bound on  $n$  is much smaller, probably close to  $10^{10}$ .
- there is often additional information pertaining to the problem; it relates to either the structure of the input graph, or the structure of the desired solution.

One implication of the above points is that an important ingredient of an algorithm’s efficiency is its ability to be modified to effectively use any additional available information, including the size of the input. In this paper, we present the results of computational experiments with a program implementing *restricted backtracking* for finding cliques in graphs. The restrictions expressed

in terms of *backtracking coordinates* are given as a part of the input together with the input graph. Our experience shows that the language of the backtracking coordinates is sufficiently flexible to enable one to utilize additional information about the problem, even though its nature may not be known beforehand.

It is well known (see [6], for example) that a greedy algorithm is generally a poor method for constructing cliques of maximal or near-maximal size. Several modifications of the greedy idea propose the selection of the *next* available vertex in such a way that some target function (usually, polynomially computable) is minimized. Often, the target function is simply the degree of a vertex. Although these modifications improve the results somewhat, as a rule, the clique sizes are often far from optimal. The reason for such a performance of the greedy algorithm is, obviously, that for most of the inputs, the step-by-step construction of a maximum or near-maximum clique must include some *non-greedy* steps. Considering all combinations of greedy and non-greedy steps, which is what the standard backtracking algorithm does, is infeasible even for small graphs, unless they are very sparse.

Most of the algorithms for the clique problem can be viewed as executing some type of restricted backtracking. The difference between the algorithms is in the pattern<sup>1</sup> of the restrictions. Roughly, our algorithm considers all solutions that can be obtained with the use of a bounded number of “non-greedy” steps. Let  $S$  be the size of the clique that the algorithm finds and  $T$  be the maximum number of the allowed non-greedy steps. Our experiments with different types of graphs that led us to consider this type of algorithmic strategy showed that the dependence of  $S$  on  $T$  is far from linear. The corresponding function  $S = F(T)$  often seems<sup>2</sup> to be a concave function with the first derivative having large values for  $T$  near 0 and small values for large  $T$ . Thus, setting  $T$  to be *small* but positive yields the value of  $S$  which is substantially bigger than that obtained by the greedy algorithm ( $T = 0$ ) and often close to the maximum. At the same time, small values of  $T$  keep the running time of the program within acceptable limits. Furthermore, we found that most of the non-greedy steps that are “needed” to obtain good values of  $S$  occur on the first levels of the backtracking tree, while the “end” of the process can almost always be made greedy without any loss in the value of  $S$ . This led us to introduce the notion of *backtracking coordinates*. Using the language of the backtracking coordinates, we describe the search area of the algorithm.

Finally, we use reorderings to narrow the search-space of the algorithm; in particular, appropriate reorderings allow us to decrease the value of the bound  $T$  without reducing the sizes of the constructed cliques. The experiments show that for large graphs, the reduction in running time caused by decreasing  $T$  more than compensates for the time needed to perform reorderings.

---

<sup>1</sup>We believe that such a view is valid even if the algorithm’s description does not yield an explicit description of the search pattern.

<sup>2</sup>The behavior of this function depends on the type of the input graph; in particular, its size and density.

Since the restrictions of the backtracking are specified in the input to our program, changes of the search pattern can be easily implemented. Thus, it is easy to tailor the restrictions to specific classes of graphs, or even to individual graphs. Consequently, this flexibility facilitates “training” the program on classes of graphs. Thus, for most of the benchmark graphs in the appendix, the restrictions used were developed before the graphs became known to us. We took advantage of the fact that the “types” of those graphs, i.e. random, brock-graphs, etc. were known beforehand. Obviously, individual tuning up is also possible, and indeed was used for some of the benchmark graphs.

The next section contains a description of the approach and the main features of the program; the program itself, called RB-clique, can be obtained by contacting M. Goldberg at `goldberg@cs.rpi.edu`. Section 3 presents computational results. The conclusion and future work are outlined in the last section.

## 2. Algorithm

The backtracking algorithmic paradigm can be viewed as “walking” along a (virtual) backtracking tree. In our implementation, the backtracking tree of the clique problem is a directed rooted tree whose nodes are labeled by lists of vertices of the graph and whose edges are labeled by individual vertices, so that

- the label  $L_0 = \{v_0, v_1, \dots, v_{n-1}\}$  of the root comprises all vertices of the graph;
- if a node  $z$  has a non-empty label  $L' = \{u_0, \dots, u_{k-1}\}$ , then the children nodes of  $z$  are formed as follows:
  - apply a reordering procedure to obtain a new list  $L'' = \{w_0, \dots, w_{k-1}\}$ ;
  - for each  $i$  ( $0 \leq i < k - 1$ ), construct a new list  $M_i$  comprised of the vertices  $w_j$  in  $L''$  with indices  $j > i$  that are adjacent to  $w_i$ ; label the child node  $z_i$  of  $z$  by  $M_i$  and the directed edge  $(z, z_i)$  by  $w_i$ .

Thus, the leaves of the backtracking tree correspond to maximal cliques; they can be retrieved by tracing the labels of the path leading from the root to the leaf. The level of the root is set to be 0; the level of any other node is  $r + 1$ , where  $r$  is the level of the parent-node.

If the reordering procedures used by the algorithm are fixed, then every node  $z$  of the backtracking tree can be uniquely described by the integer vector  $(l_0, l_1, \dots, l_{r-1})$  defined as follows. If  $(z_0, z_1, \dots, z_r)$  is the sequence of the tree nodes of the path connecting the root  $z_0$  with  $z = z_r$  and  $s_i$  is the label of the edge  $(z_i, z_{i+1})$ , then  $l_i$  is the index of  $s_i$  in the list which serves as the label for  $z_i$  ( $i = 0, \dots, r - 1$ ). We call  $(l_0, \dots, l_{r-1})$  the *backtracking coordinates* of  $z$  and of the clique corresponding to  $z$ . Obviously, the vector  $(0, 0, \dots, 0)$  corresponds to the clique obtained by applying the greedy algorithm.

Given a non-negative integer  $T$  and two vectors  $P = (p_0, \dots, p_{m-1})$  and  $D = (d_0, \dots, d_{m-1})$ , let  $\mathcal{S} = \mathcal{S}(T; P; D)$  denote the set of all integer vectors  $(x_0, \dots, x_s)$  satisfying the inequalities below. Here  $h = \min(s, m - 1)$ .

1.  $\forall i = 0, \dots, h, p_i \leq x_i \leq p_i + d_i$ ;
2.  $\sum_{i=0}^h (x_i - p_i) \leq T$ ; and
3. if  $s > m - 1$ ,  $\forall i \geq m, x_i = 0$ .

The notation  $\mathcal{S}(T; P, D)$  will also be used to denote the subtree  $\mathcal{T}'$  of the backtracking tree comprised of the nodes whose coordinates belong to  $\mathcal{S}(T; P, D)$ .

Now, the search area of the algorithm RB-clique is the region of the backtracking tree which is the union

$$\mathcal{S} = \bigcup_{i=1}^k \mathcal{S}(T_i; P_i, D_i),$$

where triples  $(T_1; P_1, B_1), \dots, (T_k; P_k, B_k)$  ( $k \geq 1$ ) are listed in the input file. RB-clique outputs the first clique whose size equals the value of `target`, one of the parameters specified in the input; if such a clique is not found, the program outputs the first maximal size clique which is in the region specified by the backtracking coordinates. The vector  $P_i$  (resp.  $D_i$ ) is called the `position` (resp. `deviation`) vector. Often,  $k = 1$  and  $P_1$  is the zero vector; in this case,  $T_1$  is simply the upper bound for the maximal number of non-greedy steps that can be made to reach any node in  $\mathcal{S}$ .

Several improvements of the basic scheme implemented in RB-clique are described below.

*Pruning.* When processing a node  $z$  of the tree and before generating a child-node of  $z$ , the program computes the length  $q$  of the list with which the child-node will be labeled. The length is compared with  $C - r - 1$ , where  $C$  is the maximal size of a clique constructed so far (initially,  $C = 1$ ) and  $r$  is the level of  $z$ . The child-node is generated only if  $q > C - r - 1$ . The experiments show that such pruning is very effective for diverse types of graphs. Note that a more sophisticated pruning technique is described in [11]. Our experiments with that technique showed that using it actually increases the running time of the program. The reason for that is, apparently, that the reduction of work brought by successful applications of pruning does not make for additional work needed to perform the pruning.

*Reordering.* During its work, the algorithm processes many subgraphs whose vertices are used to generate other subgraphs. The order in which the nodes of the backtracking tree are generated is determined by the order of the vertices of the subgraphs and may reduce the running time of the program by making pruning efficient. Even more of a saving can be made by reducing the bounds

that determine the search area which contains as good a solution as that obtained without reordering. The fundamental question is which vertices should be considered first as candidates for inclusion in the clique under construction, or in other words, how and when to reorder vertices of the subgraphs in order to improve the performance of the algorithms. It seems to be plausible that if the vertices are ordered by their degrees, one of the few at the top should belong to one of the largest cliques. Since the vertices of a subgraph are given as an ordered list  $L$ , the list  $L'$  of the vertices of a child-node is obtained from  $L$  by removing some of the entries and preserving the order of the remaining vertices. In the subgraph induced on  $L'$ , the order inherited from  $L$  may not always be of the type which is considered the best even if the order of  $L$  were such. Thus, reordering of  $L$  may be needed. The experiments support this observation: even for the Brock- and Gen-graphs (see [4], [14]) that were designed to “hide” large cliques by composing them from the vertices of relatively small degrees, the on-line reordering substantially “pushes” the cliques close to the lexicographically smallest set.

*Short-cuts.* If a degree of a vertex  $v$  is  $t - 1$  or  $t - 2$ , where  $t$  is the vertex number of the graph, then  $v$  belongs to a maximum clique of this graph. In fact, it is a special case of a more general observation made in [11]. A vertex  $v$  is called *anti-simplicial*<sup>3</sup> if for any edge  $(u, w)$ ,  $(u \neq v, w \neq v)$ , at least one of  $u$  and  $w$  is adjacent to  $v$ . Obviously, if  $v$  is anti-simplicial, then there is a maximum clique containing  $v$ . We decided not to implement in full the search for all anti-simplicial vertices, expecting that for most of the graphs, the overall time needed for the search will not be compensated by the time reduction resulting from the successful applications of the search. Our program searches only for the vertices of degree  $t - 1$  and  $t - 2$ .

Note that the vertices of degree  $t - 3$  offer one more possibility for a useful short-cut. If  $v$  is such a vertex, then either  $v$  is anti-simplicial which implies that  $v$  can be included in the output-clique, or  $v$  must be excluded from consideration but both neighbors of  $v$  must be included. Moreover, there is a potential advantage to special consideration being given to vertices of degree  $t - 4$  and even smaller degrees, but we believe that for most of the inputs, this will not reduce the running time for the same reason as above: the reduction in the running time which would result from successful applications of the routine would not compensate for the total time needed for all applications.

### 3. Performance

**3.1. Background.** The problem of selecting parameters for running RB-clique can be viewed as a replacement for the original problem. There are two main modes of the selection process, “blind” and “informative”.

---

<sup>3</sup>In [11], the authors deal with the problem of constructing an independent set, so they define *simplicial* vertices; thus, anti-simplicial vertices are simplicial in the complement to the subgraph.

*Blind.* Given a graph, a sequence of input restrictions is developed that describes expanding search areas for RB-clique. The halting condition may be set dependent on the time available for the run, or on the rate of the increase in the clique size, etc. The selection can be interactive so that every next set of parameters is chosen based on the results of the preceding runs. There can be many ways to execute RB-clique in the blind mode. One of them, which we call *gradual parameter increase*, is as follows. For the initial run, the `deviations` and `total` are set small. For every next iteration, if  $P$  and  $Q$  are the current position and deviation vectors, then their new values are taken, respectively,  $P + D_{cut}$  and  $D$ , where  $D_{cut}$  is obtained from  $D$  by making some coordinates equal to 0. We used the strategy of gradual parameter increase when developing parameters to be used by RB-clique on random graphs.

*Informative.* In this mode, the parameters are determined based on the graph’s type which is supposed to be known before a specific graph of that type becomes available. Thus, before applying RB-clique to corresponding benchmark graphs, a series of experiments was conducted on test graphs. The objective of these experiments was to find a pattern of the distribution of the backtracking coordinates that correspond to cliques of the maximum or near-maximum size. The restriction parameters eventually used for the benchmark graphs were developed based on the discovered patterns. We used both modes of the parameter selection when we applied RB-clique to benchmark graphs (see the appendix).

Experiments with diverse types of graph show that RB-clique achieves comparatively “good” results even if the value of `total` is small and the components  $d_i$  of the deviation vector are zeros for  $i > \Delta$ , where the threshold  $\Delta$  is small. The next table illustrates this empirical<sup>4</sup> observation. For more than 88 out of 100 randomly generated graphs with  $n$  vertices and edge probability  $p = 0.5$ , using `total` and  $\Delta$  from the table yields statistically optimal clique sizes.

$n$	1000	1500	2250
$\Delta$	8	9	10
<code>total</code>	24	32	40

Table 1

**3.2. Sparse vs. dense graphs.** For dense graphs, the choice of `total` and `deviations` is mainly determined by the running time: taking large values for `total` and `deviations` may substantially increase the running time. It is partially caused by the implementation of the program<sup>5</sup> but the main reason is large cliques in dense graphs. Thus, when RB-clique was applied to `MANN_a45.clq`, (second run) the value of `total` was 4, still the running time was close to 20 hours. Table 2 contains the results of applying RB-clique to random graphs with edge probability 0.9 and vertex numbers 1000 and 2000; all entries are

<sup>4</sup>At the moment, we cannot specify the notion of a “small number” for different classes of graphs.

<sup>5</sup>The algorithm uses adjacency matrices, which is inefficient in the case of very dense graphs.

the averages over 100 inputs. The values of `total` are 3 and 5; `position` are 0-vectors, and all `deviations` are either 0, or 1.

$n$	1000		2000	
<code>total</code>	3	5	3	5
Average size	62.37	63.64	71.2	72.42
Average time	58s	766s	462s	2390s

Table 2

On the other hand, for sparse graphs, the value of the parameters can be taken quite large without making the running time prohibitive. Obviously, taking all `deviations` and `total` equal to  $n$  yields an exact algorithm. When run with such parameters on random graphs with  $n = 700$  and  $p = 0.4$ , RB-clique finishes in less than 300 seconds; the corresponding number for  $n = 1000$  and  $p = 0.4$  is 5400 seconds (see Table 3). An interesting observation related to the exhaustive search was that reordering the vertices in a non-increasing order of their degrees yields shorter execution times when compared to no-reordering or reordering in a non-decreasing order of vertex degrees.

$p =$	0.4	0.5	0.6	0.7	0.8	0.9
$n = 100$	0.04s	0.07s	0.27s	1.67s	21s	838s
200	0.64s	2.83s	23s	452s	24250s	
300	4.61s	26s	551s	19090s		
500	42s	665	31723s			
700	290s	6435s				
1000	5400s					

Table 3

**3.3. Random, brock- and gen-graphs.** The asymptotics of the expected sizes of maximum cliques in random graphs have been studied by many researchers ([3], [6], [9], [12]). Let  $Z_{n,p}$  denote the size of the largest clique in a random graph with  $n$  vertices and edge probability  $p$ . Matula proved in [12] that for every given integer  $n$  and  $p > 0$ ,

$$\sum_{j=0}^k \frac{\binom{n}{k} \binom{n}{j}}{\binom{n}{k}} p^{-\frac{j(j-1)}{2}} < Prob(Z_{n,p} > k) < \binom{n}{k} p^{\frac{k(k-1)}{2}}.$$

Using these inequalities, one can compute the expected sizes of maximum cliques in graphs with given  $n$  and  $p$  (see Table 5). Knowing the expected sizes of the largest cliques turns out to be extremely valuable for the development of the parameters used by RB-clique to construct cliques of corresponding sizes. The methodology we used is as follows.

Given an integer  $n$  and  $p$  ( $0 < p < 1$ ), we generate 100 random graphs with  $n$  vertices and edge probability  $p$  (the values considered were  $n = 1000, 1500, 2250$

and  $p = 0.5$ ). For each graph, RB-clique is applied several times, each time with `target` equal to the corresponding expected clique size. For the first application, `deviations` and `total` are taken small, and are gradually increased for every next application until RB-clique finds a clique of the expected size. The backtracking coordinates of the 100 cliques found are then lexicographically sorted, which ends the first stage of the selection process. During the second stage, the value of `total` together with a sequence of pairs (`positions`, `deviations`) are developed that (1) “dominate” all the backtracking coordinates from the data base; and (2) minimize the maximum running time of RB-clique when used with these<sup>6</sup> parameters. The resulting parameters are then used as the initial parameters for the next iteration of the whole process which is applied to a series of 100 new randomly generated graphs. Similar experiments were conducted for graphs with  $p = 0.5$ ,  $n = 1000, 1500, 2250, 3375, 5060$  (every next value is 1.5 bigger than the previous) and the value of `target` one less than the corresponding expected value.

For all cases, five or more iterations<sup>7</sup> were done before we arrived at the final set of parameters. For each combination of  $n, p$  and the corresponding value of `target`, a control test on 100 new randomly generated graphs was run. In every case, the program found cliques of the expected sizes for 97 or more graphs. The results are presented, with rounding off of large numbers, in the Tables 4 and 5 below.

number of vertices	1000	1500	2250
aver & max run-time	72:404	735:3022	4750:13330
standard deviation	79	694	3762

Table 4 (`target` = expected size)

number of vertices	1000	1500	2250	3375	5060
expected clique size	15	16	17	18	19
aver & max run-time	0.47:2.48	3.53:18.25	18:69	82:421	687:2747
standard deviation	0.53	3.92	18.29	87	613

Table 5 (`target` = expected size -1)

Graph generators designed in [4] and [14] produce graphs together with maximum cliques<sup>8</sup> of specified sizes. This property of the generators was used to develop statistics of the “winning” coordinates for the brock- and gen-graphs, that were in turn fed into RB-clique when it was applied to the benchmark graphs produced by those generators. The statistics were developed with the help of an auxiliary program, *Transform*, which given a set, computes its backtracking coordinates. Additionally, two script-files were developed that link the graph generators with *Transform*. The scripts accept inputs comprised of six parts:

<sup>6</sup>In our experiments, this part was done by trial and error.

<sup>7</sup>For  $n = 2250$  and  $p = 0.5$ , 10 iterations were performed, totaling more than 1000 graphs.

<sup>8</sup>We slightly modified the Sanchis code to have this property.

the number of graphs to be generated; the vertex number of the graphs; the edge probability for the brock-graphs and the number of edges for the Sanchis-graphs; the clique size; the hiding parameter; and the starting seed. A script calls the corresponding graph-generator, reads the contents of the clique, calls *Transform* to compute backtracking coordinates, and stores the results in a log-file. Simultaneously, for every  $i \geq 0$ , the average and the maximal  $i^{\text{th}}$  coordinates are updated. Each graph is generated by using the seed which is 1 more than the seed of the previous graph; the seed of the first graph is the starting seed. Using the maximal coordinates as the deviation vectors and 0 as the position vectors would have produced optimal results with high probability. However, this would have also taken an unacceptable amount of time. Thus, we used a compromising strategy consisting of the following four steps: (a) reorder in lexicographically increasing order the vectors of the produced backtracking coordinates; (b) split the sequence into three or four<sup>9</sup> groups of consecutive vectors; for each group, compute the maximum and minimum of the coordinates; let  $Min_i$  and  $Max_i$  ( $i = 1, 2, 3, 4$ ) be the results, and let  $P_i = Min_i$  and  $D_i = Max_i - Min_i$ ; (c) reduce each of  $D_i$  by a small amount and make 0 all components  $d_i$  with sufficiently large  $i$  (this is done to make the running time feasible); (d) use the new  $(P_i; D_i)$  ( $i = 1, 2, 3, 4$ ) as the position and the deviation vectors to run any new graph of the corresponding type.

#### 4. Conclusions and Future Work

Our experiments showed that restricted backtracking, which defines the search space by restricting the backtracking coordinates, is a viable algorithmic strategy for solving the maximum clique problem and probably other<sup>10</sup> hard combinatorial optimization problems. The program implementing restricted backtracking can be run to find not-too-poor solutions quickly and good solutions in a reasonably short time. By specifying the restriction parameters, one can narrow the search space to reflect on additional information about the class of inputs in question.

It is easily adapted for running on a parallel computer. We believe that the efficient use of parallel architecture with a good speedup can be obtained in the following way. During the first stage of the work, the program generates a sufficient number of partial solutions to load every processor (probably with some overlap). After that, processors execute the same program, each on its own partial solution. The sizes of the cliques obtained by the processors are periodically compared and the best is used by all for pruning. Note that the ability to learn about a better solution earlier will often be an additional speeding factor.

The strategy readily lends itself to learning, the property facilitated by the separation of the restriction parameters defining the search strategy and the pro-

---

<sup>9</sup>The strategy was applied separately for each of the 11 types of brock- and gen-graphs.

<sup>10</sup>Similar results were obtained in our experiments with the graph partitioning problem (see [2]).

gram itself. Future versions of the program will be supplied with a database of statistics and procedures that will mold the search area of the algorithm by dynamically imposing restrictions on the backtracking coordinates. Replacement of the restriction parameters with restriction procedures should substantially improve the efficiency of the program. Similar development—restriction procedures via restriction parameters—should be possible for many other classes of graphs, in particular, “combinatorial” graphs, such as Keller-graphs, Mann-graphs, and graphs for the packing design problem.

### References

1. S. Arora, C. Lund, B. Motvani, M. Sudan, M. Szegedy, “Proof Verification and Hardness of Approximation Problems,” in *Proceedings of the 33rd Annual Symposium on Foundation of Computer Science*, pp. 14-23, 1992.
2. J. Berry and M. Goldberg, Path optimization and Near-Greedy Analysis for Graph Partitioning: An Empirical Study, *Proceedings of the 1995 Symposium on Discrete Algorithms*.
3. B. Bollobás and P. Erdős, Cliques in Random Graphs, *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol 80, (1976) pp. 419–427.
4. M. Brockington, J. C. Culberson, Camouflaging Independent Sets in Quasi-Random Graphs, in *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, David S. Johnson and Michael A. Trick (eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1995.
5. P. Berman and A. Pelc, Distributed Fault Diagnosis for Multiprocessor Systems, *Proceedings of the 20th Annual International Symposium on Fault-Tolerant Computing*, Newcastle, UK, 340-346 (1990)
6. B. Bollobás and A. Thomason, Random Graphs of Small Order, *Annals of Discrete Mathematics* **28** (1985), pp. 47–97.
7. U. Feige, S. Goldwasser, L. Lovasz, S. Safra, M. Szegedy, Approximating the Maximum Clique is almost NP-complete, *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pp. 2-12, 1990.
8. M.R. Garey, D.S. Johnson, *Computer and Intractability— A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.
9. M. Jerrum, Large Cliques Elude the Metropolis Process, *Random Structures and Algorithms*, 3(4), pp. 347 - 360, 1992.
10. R.M. Karp, “Reducibility among combinatorial problems,” in *Complexity of Computer Computations*, R.E. Miller and J. W. Thatcher (eds), Plenum Press, New York, pp. 85 - 103.

11. C. Mannino and A. Sassano, An Exact algorithm for the maximum stable set problem, *Computational Optimization and Application*, 3, pp. 243-258, (1994).
12. D. Matula, The Largest Clique Size in a Random Graph, Southern Methodist University, Tech. Report, CS 7608 (April 1976).
13. P.M. Pardalos and Jue Xue, The Maximum Clique Problem, Department of Industrial and System Engineering, University of Florida, Gainesville FL, 1992.
14. L. Sanchis, Test case construction for the vertex cover problem, in *Computational Support for Discrete Mathematics* Nathaniel Dean and Gregory E. Shannon (eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 15, pp. 315-326, (1994).

Mark K. Goldberg

DEPARTMENT OF COMPUTER SCIENCE, RENSSELAER POLYTECHNIC INSTITUTE, TROY, NEW YORK 12180

*E-mail address:* goldberg@cs.rpi.edu

Reid D. Rivenburgh

DEPARTMENT OF COMPUTER SCIENCE, RENSSELAER POLYTECHNIC INSTITUTE, TROY, NEW YORK 12180

*E-mail address:* rivenbur@cs.rpi.edu

current address: CIC-3, COMPUTER RESEARCH AND APPLICATIONS, MS M986; LOS ALAMOS NATIONAL LABORATORY; LOS ALAMOS, NM 87545

*E-mail address:* reid@lanl.gov

**Appendix**  
**Second DIMACS Challenge**  
 Clique Benchmark Results

*GENERAL INFORMATION*

*Authors:* Mark K. Goldberg and Reid Rivenburgh

*Title:* Constructing Cliques Using Restricted Backtracking

*Name of Algorithm:* RB-clique

*Brief Description of Algorithm:* Heuristic: uses the restricted backtracking algorithmic paradigm for the maximum clique problem; the search area is described in terms of backtracking coordinates; when run as an exact program is efficient for input graphs that are not very dense and have at most a few hundred vertices.

*Type of Machine:* Sparcstation 10 Model 51, 128 Mbytes RAM.

*Compiler and flags used:* g++, the GNU C++ compiler, with the -O optimization and -g debugging flags.

*MACHINE BENCHMARKS*

*User time for instances:*

r100.5	r200.5	r300.5	r400.5	r500.5
.08	1.00	7.87	47.68	179.98
.07	1.11	9.26	55.78	214.89

*ALGORITHM BENCHMARKS*

*Authors' Comments:*

The times recorded in the tables do not include the time for reading in the input graph. When recording running times, we dropped the digits after the decimal point if the value before the decimal point was in the hundreds.

The restriction parameters used for the brock- and gen-graphs were preselected using the informative mode of the selection process; for each of these graphs, there was only one run. For each of C125.9.clq C250.9.clq C500.9.clq, C1000.9.clq, C2000.9.clq, DSCJ500.5.clq, and DSCJ1000.5.clq, there were two runs, with the same `positions` and `deviations`, but different values of `target`. For the first run, `target`'s values were 34, 44, 55, 64, 74, 16, 18, 13, and 15 respectively, and for the second run, all of them were increased by one. The restriction parameters the `p_hat` series of graphs with 1000 and 1500 vertices were selected based on the results of applying RB-clique to the graphs of this

series with 700 and 300 vertices. Each of the  $p_{\text{hat}}300$  graphs was run three times. We record the worst and the best solutions, the running time for the first run, and the total running time for the other two runs.

Name	Runs	Time		Solution	
		First	Second	First	Second
C125.9.clq	2	35	35	34	34
C250.9.clq	2	180	223	44	44
C500.9.clq	2	11	1210	55	55
C1000.9.clq	2	641	974	65	65
C2000.9.clq	2	135	2785	74	74
DSJC500.5.clq	2	0.6	110	13	13
DSJC1000.5.clq	2	20	968	15	15
C2000.5.clq	2	13	1957	16	16
C4000.5.clq	2	3653	7360	18	18
MANN_a27.clq	1	12	—	126	—
MANN_a45.clq	2	4409	63,993	343	344
MANN_a81.clq	1	325	—	1097	—
brock200_2.clq	1	0.3	—	12	—
brock200_4.clq	1	8.55	—	17	—
brock400_2.clq	1	12.3	—	25	—
brock400_4.clq	1	1065	—	33	—
brock800_2.clq	1	940	—	21	—
brock800_4.clq	1	912	—	21	—
gen200_p0.9_4	1	392	—	42	—
gen200_p0.9_55	1	162	—	55	—
gen400_p0.9_55	1	2536	—	52	—
gen400_p0.9_65	1	2660	—	60	—
gen400_p0.9_75	1	3239	—	74	—
hamming8-4.clq	2	0.9	919	16	16
hamming10-4.clq	2	0.3	105	36	40
keller4.clq	2	0.1	1.3	11	11
keller5.clq	2	58	416	26	27
keller6.clq	2	16	9649	52	54
p_hat300-1.clq	3	1	2	8	8
p_hat300-2.clq	3	1	7	25	25
p_hat300-3.clq	3	6	200	34	35
p_hat700-1.clq	2	13	15	11	11
p_hat700-2.clq	2	450	540	44	44
p_hat700-3.clq	2	4146	4398	62	62
p_hat1000-1.clq	1	44	—	10	—
p_hat1000-2.clq	1	209	—	46	—
p_hat1000-3.clq	1	1035	—	65	—
p_hat1500-1.clq	1	240	—	12	—
p_hat1500-2.clq	1	1014	—	64	—
p_hat1500-3.clq	1	3452	—	93	—