RC23562 (W0503-061) March 10, 2005
Mathematics

IBM Research Report

Accelerating Backtrack Search

James B. Shearer
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Accelerating Backtrack Search

James B. Shearer
IBM Research Division
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, N.Y. 10598
jbs at watson.ibm.com

Abstract: We describe a method to reduce the amount of computer time required to perform an exhaustive backtrack search. For the independent set problem, the simplest version of the method achieves speedups of 3-4 on the DIMACS test graph r500.5. More complicated versions can achieve even greater speedups.

AMS Subject Classification. 05B99

Backtrack search is a well known type of algorithm for solving combinatorial problems (or proving that solutions do not exist). Solutions are built up incrementally with the algorithm "backtracking" to make a different choice at an earlier stage whenever it gets stuck (ie whenever it becomes apparent that the current partial solution cannot be successfully completed). Such searches can be organized to find all solutions. However their execution time may grow very rapidly with the size of the problem.

Some combinatorial problems have symmetry conditions which mean that solutions can be partitioned into equivalence classes. In such cases backtrack search can be made more efficient by organizing the search to find only one solution in each equivalence class. For example consider Golomb rulers ([5]). A k mark Golomb ruler of length n is a set of k integers $0 = a_1 < a_2 < \cdots < a_k = n$ such that all the differences $\{(a_i - a_j)|1 \leq j < i \leq k\}$ are distinct. Clearly the reflection (ie the set $\{(n - a_{k+1-i})|i = 1, \ldots, k\}$) of a Golomb ruler is also a Golomb ruler. So if we are performing a backtrack search to build up a Golomb ruler by choosing $a_1, a_2, \ldots$ in sequence we can assume the middle mark, $a_{(k+1)/2}$, has value at most $(n+1)/2$ ([4],[7]). It is notable that imposing this symmetry condition speeds up the backtrack search by more than a factor of two. This suggests that speedups may be possible even if the problem is not symmetric.

Consider the problem of finding the independence number of a graph. The independence number of a graph is the maximum size of an independent set in the graph. An independent set in a graph is a set of vertices no pair of which are adjacent. This problem is NP-complete. It is equivalent to the problem of finding the clique number of the complement of the graph (where a clique is a set of vertices every pair

of which is connected by an edge and the clique number is the maximum size of a clique) and is sometimes stated in that form. The second DIMACS challenge asked for efficient algorithms to solve this problem. As a benchmark DIMACS included a program dfmax.c ([1]) which solved the problem by backtrack search. Dfmax is an efficient implementation of the following algorithm (which is similar to the algorithm stated in [2] and to the algorithm used for the computations in [6])

Let the graph, G, consist of n vertices $\{v_i | i = 1, \ldots, n\}$. Build up an independent set choosing vertices $v_{j_1}, v_{j_2}, \ldots$ with $j_1 < j_2 < \cdots$. This may be implemented efficiently by maintaining lists of eligible vertices at each level of the search. After choosing a vertex go through the list of the remaining eligible vertices at that level and delete the ones adjacent to the chosen vertex. This is the new eligible list at the next level. When the search reaches a point where there are not enough eligible vertices in the current list to improve the current independence number backtrack to the next higher level in the search and try the next vertex in the eligibility list for that level. Clearly we can choose how to initially order the vertices. The dfmax algorithm optionally initially orders the vertices by successively choosing the vertex of maximum degree in the graph induced by the set of vertices not yet chosen.

Suppose the map which sends vertex v_i to v_{n+1-i} is an automorphism of G. Then when using the above algorithm to search for an independent set of size k we can assume that at least $(k+1)/2$ of the vertices of the independent set are chosen from the first $(n+1)/2$ vertices of G. This will speed up the search. In fact the speedup may exceed a factor of two. This suggests trying to use the same idea to speed up the search even when G is not symmetric. We do this by partitioning the vertices of G into two sets, A and B. Suppose we are searching for an independent set of size k. First order the vertices so that the vertices in A precede the vertices in B and perform the above backtrack search assuming at least $(k+1)/2$ vertices of the independent set are chosen from A. Next order the vertices so that the vertices in B precede the vertices in A and perform another search which assumes at least $(k+1)/2$ vertices are chosen from B. Now it is easy to see that for any independent set, S, of size k (or greater) either A or B (or both) must contain at least $(k+1)/2$ vertices of S. So by performing two backtrack searches in the above manner we will find all independent sets of size k (or greater). In some cases this algorithm will be faster than the original algorithm even though two searches are being performed. We can try to improve the performance further by choosing A and B wisely. For the times reported below we start with a partition into sets of equal (or nearly equal) size and then try to minimize the number of edges between A and B by interchanging pairs of vertices until we are at a local optimum.

We compared the above algorithm to the dfmax algorithm on the DIMACS test graph $r500.5$, a random graph on 500 vertices with edge probability .5 ([3]). We implemented our own version of the dfmax algorithm (A1) and then modified it to perform the revised algorithm (A2). We timed these algorithms with and without

Algorithm	preorder	M1	M2	M3
dfmax	no	35.37	23.65	12.71
dfmax	yes	25.20	16.34	8.59
A1	no	26.84	17.41	8.20
A1	yes	21.31	12.53	5.74
A2	no	8.86	5.70	2.13
A2	yes	5.27	3.20	1.34

Table 1: Search times

preordering for r500.5 on 3 different RS/6000 workstations (M1, M2 and M3). The results appear in table 1. Times are in seconds. For this problem the method produces speedups of $3-4$. More complicated versions of the method (corresponding to larger symmetry groups) can produce even greater speedups.

References

[1] Applegate, D., Johnson, D.S., *dfmax.c*, ftp://dimacs.rutgers.edu/pub/challenge/graph/solvers/dfmax.c.

[2] Carraghan, R., Pardalos, P.M., *An Exact Algorithm for the Maximum Clique Problem*, Operations Research Letters, pp. 375-382, 9(1990).

[3] Dimacs, *r500.5*, ftp://dimacs.rutgers.edu/pub/challenge/graph/solvers/r500.5.b.

[4] Dewdney, A. K., *Computer Recreations*, Scientific American, pp. 14, March 1986.

[5] Gardner, M., *Mathematical Games*, Scientific American, pp. 116-118, June 1972.

[6] Shearer, J.B., *Lower Bounds for Small Diagonal Ramsey Numbers*, Journal of Combinatorial Theory, Series A, pp. 302-304, 42(1986).

[7] Shearer, J.B., *Some New Optimum Golomb Rulers*, IEEE Transactions on Information Theory, pp. 183-184, 36(1990).