Learning in Gradient Descent Scheduling

Jeremy Frank
frank@cs.ucdavis.edu
(916) 758-5925

Armand Prieditis
prieditis@cs.ucdavis.edu
(916) 752-6958

Division of Computer Science
University of California at Davis
Davis, CA. 95616

June 13, 1994

Abstract

Local methods are often used to find approximate optimal solutions to scheduling problems. One such method, gradient descent, has proven useful in finding approximate solutions to such problems. This paper presents a way to improve such methods by learning from local searches. Our empirical results show that learning can significantly improve the performance of gradient descent in solving scheduling problems.

1 Introduction

Local search is a popular way to find approximate solutions. For example, [JoPa] discusses local search and local algorithms to solve the Travelling Salesman problem. [MiLa] and [TaJo] present local methods of solving scheduling problems. [Br] discusses a local method of solving the Graph Coloring problem. [Se] uses gradient descent methods to solve graph coloring and 3CNF problems. Figure 1 shows an example of local search. As the figure shows, a solution is compared with other "close" solutions; that is, solutions which are "neighbors" to it. If a neighbor is found to be a "better" solution, then that solution becomes the next solution considered. If no solution is "better" than the current solution then the solution is "locally optimal" [JoPa]. Local search may lead to optimal solutions, but for the purposes of this paper it is considered as an approximation method [LaLe]. This paper

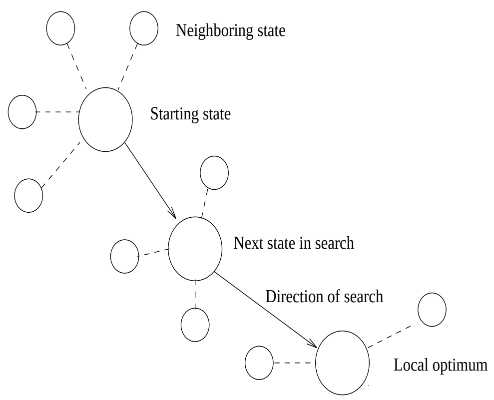

Figure 1: Local Search Algorithm

describes a method to improve a gradient descent scheduler.

Section 2 of the paper describes previous work in using learning with gradient descent algorithms. Section 3 describes the gradient descent method of local search, and Section 4 describes how learning is added to the algorithm. In Section 5 we describe the scheduling problem and how our algorithm is used to find approximate solutions. Section 6 presents the empirical results of our tests. Section 7 presents our conclusions and opportunities for future work.

2 Previous Approaches to Learning in Gradient Descent

[HeDe] discuss a learning gradient descent method to solve graph-coloring problems. Their gradient descent algorithm finds the best neighboring state to move to, even if the state is further from a solution than the current state. A *tabu* state is one that was seen in a previous iteration. A list of states the algorithm searches is kept in a FIFO list, and is used to prune searches. Each state is kept in the list for only a fixed amount of time. Thus the algorithm "learns" not move to states it has seen before, but then "forgets" those states as it learns new ones. [Zw] uses learning to choose heuristics driving a local repair algorithm. Two possible heuristics for performing local repairs may be used to create schedules, one of which is better informed than the other. The first heuristic is used to create schedules, and statistics are gathered on its performance. The learning algorithm then indicates when the better informed heuristic should be used.

3 The Scheduling Problem

The scheduling problem considered in this paper is the Constraint Satisfaction Problem [De] [JaPr] which is similar to the graph coloring problem [Br]. For the purposes of this paper jobs are assumed to take unit time, and all jobs can be performed in parallel with given resources. We also assumes that the only constraints on jobs are mutual exclusion constraints. Thus a scheduling project consists of a set of jobs to be done, and a set of mutual exclusion constraints among the jobs. The project to be scheduled is represented as a project graph. Each job corresponds to a node of the project graph, and a mutual exclusion constraint among two jobs is represented as an undirected edge between the two job nodes. A schedule for a project graph is an assignment of direction to the edges of the project graph. The duration of the schedule is the longest path through the graph, which corresponds to the Critical Path heuristic [JaPr]. The object is to generate a schedule with a duration less than some parameter d.

4 Gradient Descent

Gradient descent algorithms generate an initial schedule, then use a form of local search to find better solutions. Figure 3 shows a schedule for a particular problem. Each edge is a *component* of the state. A state is a *neighbor* of another state if each state differs in the assignment of only one component.

The *value* of a state is the Critical Path length. The *gradient* between neighboring schedules is the change in value between a schedule and its neighbor. We assume that a positive gradient indicates that a schedule is closer to a solution than it's neighbor, while a negative gradient indicates that a schedule is further away from a solution. The *maximum gradient* is therefore the maximum change in value of the schedules.

The gradient descent algorithm used in this paper is described in figure 2. The algorithm generates a schedule at random and moves to the neighboring schedule which maximizes the gradient. The algorithm can reach a schedule where the maximum gradient is either negative or zero, called a *local optimum.* If the local opti-

```
while not done
   generate schedule
   find critical path length
   while not zerogradient
      for each edge
         reverse edge
         find new critical path length
         if new length < old shortest
            save best edge
      if new length < d
         done
      else if there is a best edge
         not zerogradient
         change the schedule
      else zerogradient
```

Figure 2: Gradient Descent Algorithm

mum does not solve the problem, our algorithm must start over again, creating a new schedule at random. The procedure of randomly generating a schedule and finding a local optimum is repeated until a solution is found. It should be noted that the algorithm can fail to find an optimal solution even should one exist, and is therefore incomplete [Se] [HeDe]. Unlike the algorithms in [Se] and [HeDe], this algorithm continues to search until the gradient becomes zero, and does not move along a negative gradient.

The algorithm does most of the work in determining which schedule to move to next since it must determine the duration of all neighboring schedules. The complexity of the algorithm is thus proportional to the number of times the algorithm attempts to follow the maximum gradient. Each step of finding the the maximum gradient is called a *patch*.

5 Learning in Gradient Descent

The gradient descent method can waste work by visiting schedules it has visited before. If the algorithm moves to a schedule that it previously visited, it will follow exactly the same path to the schedule of gradient zero which it followed previously. The probability of this occurring is initially small, but grows slightly as search continues. Further, schedules may contain information that the algorithm can use to prune its searches. A search path can be pruned if the algorithm can establish that moving to a schedule will not move towards a solution to the problem, even if the algorithm has not searched the schedule previously.

To avoid searching previously seen schedules, the algorithm can store all edge assignments of every schedule it visits, and determine whether any new schedule is one that has been visited earlier. This is known as *rote learning*. The algorithm never returns to a schedule on the search path to a local optimum which does not solve the problem. Even if the number of schedules the algorithm searches is small this may be expensive in storage space and search time.

[De] presents the idea of a *minimum conflict set* of components that makes a schedule undesirable in some way. We modified this method and combined it with the gradient descent algorithm. For our purposes a minimum conflict set is a set of edges on the Critical Path if it is $< d$. The conflict sets are of size $\leq k$. The constant k is referred to as the *order* of the algorithm, and this method is called *kth order learning*. Figure 3 shows an example of a conflict set for the scheduling problem. In the figure the critical path is assumed to be the minimum conflict set. If the desired schedule duration is 4 then the duration of the critical path is too long and the solid edges on the critical path comprise the minimum conflict set.

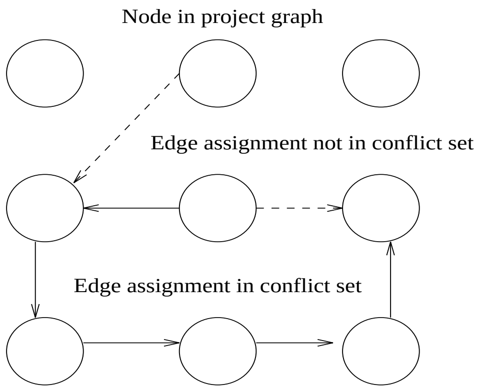

Figure 3: Minimum conflict set.

The gradient descent algorithm with learning is shown in Figure 4. We assume all schedules on the path to a local minimum which is not a solution contain potential conflict sets which might be of use to the algorithm. The algorithm searches each schedule for minimum conflict sets, which are stored in a database. The database is used to avoid visiting any schedule which contains a conflict set. Unlike [HeDe] constraints are stored through the entire run of the program.

Since the rote learning algorithm constrains the search only if it attempts to visit an identical schedule, the probability of using any constraint is very small. The conflict sets are assumed to be characteristic of schedules on paths to poor locally optimum schedules. By searching for a conflict set among the components of the schedule the kth order algorithm abstracts the most important information from a schedule. Since the number of components in the conflict sets are less than a fixed size and may be small compared to the problem size, there is a higher probability that they will appear in any schedule. Thus the kth order algorithm should outperform both the gradient descent algorithm and the gradient descent algorithm with rote learning.

```
while not done
   generate constraint-free schedule
   find critical path length
   while not zerogradient
      if length(critical path) < k
         store edges on critical path
      for each edge
         reverse edge
         if constraints not violated
            find new critical path length
            if new length < old shortest
               save best edge
      if new length < d
         done
      else if there is a best edge
         not zerogradient
         change the schedule
   else zerogradient
```

Figure 4: Gradient Descent with Learning

6 Empirical Results

The kth order algorithms were tested with respect to both the gradient descent algorithm and the gradient descent algorithm with rote learning. The results measured the number of times the algorithm performed the patch step, determining the maximum gradient, before finding a solution to the problem. We assume that the time to store conflict sets and search the database in the kth order algorithms is negligible. The experiments were designed to see if learning improves the gradient descent algorithm, and whether kth order learning improved the algorithm more than rote learning.

The gradient descent algorithm was run

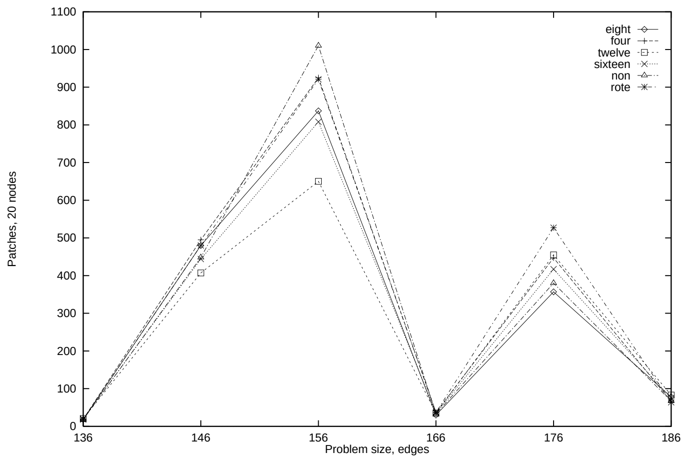

Figure 5: Average Schedules Seen, 20 nodes.

on each problem with a fixed number of patches to determine an approximate minimum, which was used as a parameter to each program to gather experimental data. Each program was run 100 times and the number of patches required to solve the problem was averaged. Two sets of projects were generated to demonstrate the performance of the algorithms on graphs with different densities of edges; one set contained 20 jobs, the other set contained 24 jobs. The schedules were randomly generated and then tested for easily exploited symmetry. Each set contained six projects with varying edge densities. To analyze the effectiveness of the kth order algorithms, we chose sufficient k to cover the range from 4 to (problem size-4) in increments of 4.

As shown in Figures 5 and 6 for some k the kth order programs perform better than both the non-learning gradient descent algorithm and the gradient descent algorithm with rote learning. In Figure 5 the twelfth-order algorithm performs best on the problems of size 146 and 156, while the eighth-order algorithm performs best on the schedule with 176 edges. Curiously, for problems with higher edge densities, rote learning starts to degrade in performance. In Figure 6 the twentieth-order algorithm performs consistently well, but the performance of rote learning is more variable. In both cases, as the problems require more and more patches to solve, the kth order algorithms are clearly superior to the other methods. Unfortunately no clear criterion for choosing k manifests itself in this data.

7 Conclusions, Shortcomings and Future Work

Our empirical results show that learning can significantly improve the performance of gradient descent algorithms designed to solve scheduling problems. While the initial results are promising, there are several shortcomings to our approach. First, there is no theoretical foundation for choosing k properly. For some problems a small k may be the most useful, while for others a larger k may improve the algorithm more. If an underlying distribution of edges is assumed then intelligent choices of k may be possible.

Second, local methods often do not have a global termination criteria. As in [HeDe] the algorithm cannot determine whether a problem has a solution or not, and thus does not know when to halt. In this paper the problem was avoided by testing

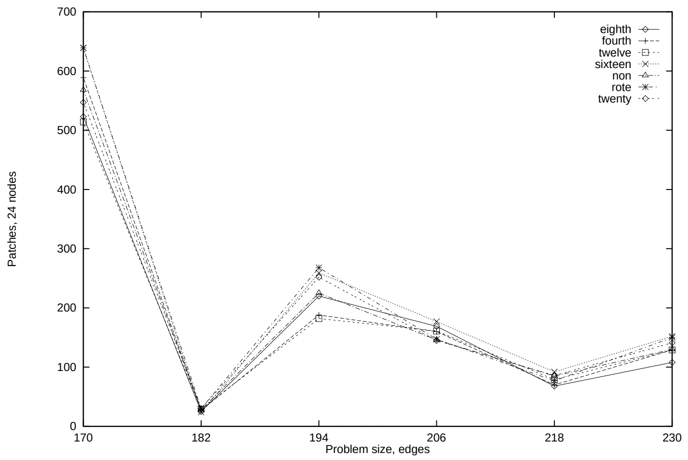

Figure 6: Average Schedules Seen, 24 nodes.

the algorithms on problems with known solutions. The information found by learning algorithms may aid in finding a global heuristic for halting the algorithm.

The learning method presented here may not be the best for improving local search. Alternative learning methods can also be explored. The definition of neighborhood corresponds to look-ahead depth in other algorithms. The kth order method can be extended by changing the definition of neighborhood and allowing the change of two or more components; this is analagous to increasing the look-ahead in other search algorithms. Minimizing the sum of product expressions or employing the use of Bayeseian statistics to constrain search are directions we are pursuing. Finally, using information to construct solutions rather than constrain search may prove more useful.

The heuristics used to build the conflict sets may be improved. For example, in the scheduling problem, it may be possible to identify other edge assignments as well as those on the Critical Path to store as part of the conflict set. Other enhancements of the heuristics are possible, as well as learning of heuristics [Pr].

Lastly, the addition of learning to local methods may prove useful in improving local algorithms to solve other problems such as TSP, 3CNF, and Binpacking. This work is currently in progress.

References

[MiLa] S. Minton, P. Laird, M. D. Johnston, A. Phillips. Solving Large Scale Constraint Satisfaction And Scheduling Problems Using A Heuristic Repair Method. Proceedings, AAAI, 1990.

[TaJo] P. Tadepalli, V. Joshi. Real-Time Scheduling Using Minimin Search. AAAI Spring Symposium on Practical Approaches to Scheduling and Planning, 1992.

[JaPr] B. Janakiraman, A. Prieditis. Generating Effective Project Scheduling Heuristics By Abstraction and Reconstitution. AAAI Spring Symposium on Practical Approaches to Scheduling and Planning, 1992.

[De] R. Dechter. Learning While Searching In Constraint Satisfaction Problems. Proceedings, AAAI, 1986.

[Br] D. Breláz. New Methods to Color the Vertices of a Graph. Proceedings, CACM, 1979, pp. 251-256

[ChKa] P. Cheeseman, Bob Kanefsky, William Taylor. Where the *Really* Hard Problems Are. Proceedings of the 12^{th} IJCAI, 1991.

[Zw] M. Zweben, E. Davis. *Learning to Improve Iterative Repair Scheduling.* Technical Report, NASA Ames, 1992.

[HeDe] A. Hertz, D. deWerra. Using Tabu Search Techniques for Graph Coloring. *Computing*, vol. 39, 1987, pp. 345-351.

[Se] B. Selman. A New Method for Solving Hard Satisfiability Problems. Proceedings, AAAI, 1992, pp. 440-446.

[JoPa] D. S. Johnson, C. Papadimitriou, M. Yannakakis. How Easy is Local Search? *Journal of Computer and System Sciences*, vol. 37, 1988, pp. 79-100.

[LaLe] E. Lawler, J. Lenstra, A. Rinnooy Kan, D. Shamoys, ed. The Travelling Salesman Problems. New York: Wiley and Sons, 1985.

[MiSe]
D. Mitchell, B. Selman, H. Levesque. Hard and Easy distribution of SAT Problems. Proceedings, AAAI, 1992, pp. 459-465

[Pr] A. Prieditis. *Machine Discovery of Effective Admissible Heuristics.* Machine Learning Journal, to appear in 1993.