Reinforcement Learning and Local Search:
A Case Study

by
Robert Moll, Andrew Barto,
Ted Perkins & Richard Sutton

CMPSCI Technical Report 97-44
September, 1997

Reinforcement Learning and Local Search: A Case Study

Robert Moll, Andrew Barto, Ted Perkins, Richard Sutton
Department of Computer Science
University of Massachusetts, Amherst MA 01003

Abstract

We describe a reinforcement learning-based variation to the combinatorial optimization technique known as local search. The hillclimbing aspect of local search uses the problem's primary cost function to guide search via local neighborhoods to high quality solutions. In complicated optimization problems, however, other problem characteristics can also help guide the search process. In this report we present an approach to constructing more general, derived, cost functions for combinatorial optimization problems using reinforcement learning. Such derived cost functions integrate a variety of problem characteristics into a single hillclimbing function. We illustrate our technique by developing several such functions for the Dial-A-Ride Problem, a variant of the well-known Traveling Salesman Problem.

1 Introduction

Combinatorial optimization problems are fundemental in many areas of computer science, engineering, and operations research. Solving such problems involves searching a discrete, finite space of feasible solutions for an optimal solution—a solution of least cost. Often, finding an optimal solution is not possible, and so a good quality near-optimal solution is acceptable.

One popular and generally effective approach to such problems is the method of local search, also known as iterative improvement or hillclimbing. The local search algorithm may be summarized as follows:

1. obtain an initial feasible solution;

2. find a neighboring, less costly solution, and repeat this step; or stop, and report the most recent best solution—a *local* optimum.

Local search suffers from two principal defects: it tends to get stuck at poor local optima; and it can be difficult to apply when a problem is particularly complex or unevenly constrained.

Healy and Moll [4,5] offered one remedy to this situation when they proposed their sacrifice principle. According to this approach, some of the time the local search algorithm moves from solution S to neighboring solution T, not because T is less expensive than S, but because T has a larger local neighborhood than S. Thus, feasible solutions are evaluated not just by their primary cost, but also by a secondary cost, namely neighborhood size. This quality sacrifice mechanism allows the algorithm to avoid getting stuck at local optima, and it also provides a natural way to apply local search to more complex problems, e.g., those problems with nonuniform neighborhood structures.

In this report we develop a more general framework for addressing the inherent defects in local search. First, we identify several cost functions that are relevant to a particular problem. We treat these as features, and we combine them to construct a single, derived, cost function that is now a suitable vehicle for hillclimbing. To make our discussion more concrete, we focus on a single traditional combinatorial optimization problem, namely the Dial-A-Ride problem, or DARP, which is a somewhat complicated variant of the well-known Traveling Salesman Problem, or TSP.

Our work is closely related to two recent studies that have appeared in the machine learning community. Boyan and Moore [2] report on a learning-based method for searching a space of feasible solutions for good initial solutions from which to begin local search. Zhang and Dietterich [12] consider instances of a NASA space shuttle mission scheduling problem. They show how reinforcement learning (RL) may be used to acquire a technique for "repairing" infeasible schedules efficiently in an iterative-improvement style.

This report is organized as follows. In Section 2 we discuss the local search technique in more detail. In section 3 we describe the DARP problem. In Section 4 we present our model for integrating local search and RL. In Section 5 we describe the algorithms we have developed that exploit this integration. In Section 6 we present our experimental results. And in Section 7 we discuss our work, present some conclusions, and outline directions for future research.

2 Local Search

Following [8], we distinguish between an optimization problem instance and an optimization problem. For example, an *instance* of TSP includes a concrete distance matrix of fixed size as part of its specification. By contrast the general TSP *problem* refers to all TSP instances of all sizes.

Definition: An optimization problem *instance* is a pair (F, c), where F is a set of feasible solutions, and c is a cost function that assigns a real number to each solution. We say $f \in F$ is a *global minimum* for the instance if $c(f) \leq c(g)$ for all $g \in F$.

Local search imposes a neighborhood structure on the set of feasible solutions of a problem instance. Informally, g is a neighbor of f for a particular instance—that is, $g \in N(f)$—if g is "close" to f in some discernible way. A solution f is a local minimum if for all $g \in N(f)$ we have that $c(f) \leq c(g)$.

The local search algorithm uses the neighborhood structure of a problem to limit search. The algorithm may be summarized as follows:

1. Find an initial feasible solution f_0
2. Given feasible solution f, search for $g \in N(f)$ for which $c(g) < c(f)$;
3. Ff there is such a g, say g^*, goto 2, replacing f with g^*;
4. Else return f (the local minimum).

We call the actions embodied in Steps 2 and 3 above, which move the algorithm from solution f to neighboring solution g, one cycle of the local search algorithm.

Traditionally local search proceeds with respect to the natural cost function of a problem, e.g., length of tour in the case of TSP. When search proceeds by moving to the first discovered element of $N(f)$ with lower cost, the resulting algorithm is called a *first-improvement* algorithm. If all members of $N(f)$ are considered, and the algorithm advances to a neighbor with lowest cost, the resulting algorithm is called a *best-improvement* algorithm.

3 DARP

DARP has the following concrete formulation. A van is parked at a terminal, and the driver receives N phone calls from customers who need rides. Each call identifies the location of the calling customer, as well as that customer's destination. After the calls have been received, the van must be routed so that it starts from the terminal, visits each site in every pick-up/drop-off pair in some order, and then returns to the terminal. There are precedence constraints on the van's tour: it is constrained to pick up a passenger before eventually dropping off that passenger.

The van tour should be of minimal length. Failing this goal—and DARP is NP-complete, so it is unlikely that optimal DARP tours will be found easily—at least a good quality tour should be constructed. We assume that the van has capacity N, where N is the number of customers, and that the distances between pick-up and drop-off locations are represented by a symmetric Euclidean distance matrix.

We use the notation

$$0\ 1\ 2\ -1\ 3\ -3\ -2$$

to denote the following tour: "start at the terminal (zero), then pick up 1, then 2, then drop off 1 (thus: -1), pick up 3, drop off 3, drop off 2 and then return to the terminal (site 0)." We will sometimes refer to single-leg tour segments, e.g., the segment from 0 to 1, 3 to -3, etc., as edges of the tour.

Given a tour T, we define $N_2(T)$ to be the set of all legal tours obtainable from T by subsequence reversal. For example, for the tour above, the new tour created by the following subsequence reversal

$$0\,1\,/\,2\ -1\,3\,/\ -3\ -2 \longrightarrow 0\,1\,3\ -1\,2\ -3\ -2$$

is an element of $N_2(T)$. We say that $N_2(T)$ is the 2-opt neighborhood of T. Notice that because of the precedence constraints not all such reversals lead to feasible tours. For instance,

$$0\,1\,2\,/\ -1\,3\ -3\,/\ -2 \longrightarrow 0\,1\,2\ -3\,3\ -1\ -2$$

is infeasible, since it asserts that passenger 3 is dropped off first, then picked up.

The neighborhood structure of DARP is highly non-uniform. Tours of the form

$$0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5 \ldots N\ -1\ -2\ -3\ -4\ -5 \ldots -N$$

have feasible N2 neighborhood size of O(N2), whereas tours of the form

$$0\,1\ -1\,2\ -2\,3\ -3\,4\ -4\,5\ -5 \ldots N\ -N$$

have feasible N_2 neighborhood size of $O(N)$.

In our discussions of the N_2 neighborhood structure, we assume throughout that there is a fixed, standard order of enumeration of the neighbors of any tour T.

Definition: Let T be a feasible DARP tour. By 2-opt(T) we mean the tour obtained by first-improvement hillclimbing using the N_2 neighborhood structure (presented in a fixed, standard enumeration), with tour length as the cost function. By slight abuse of terminology we also use the term 2-opt to refer to the local search algorithm that uses the 2-opt(T) iterative improvement step.

As with TSP, DARP admits a 3-opt algorithm, where a 3-opt neighborhood $N_3(T)$ is defined and searched in a fixed, systematic way, again in first-improvement style. This neighborhood is created by inserting three rather than two "breaks" in a tour. 3opt is much slower than 2-opt, more than 100 times as slow for $N = 50$ for each run starting from a random tour. But it is much more effective, even when 2-opt is given equal time to generate random starting tours and then complete its iterative improvement scheme. We will write 3-opt(T) to denote the tour obtained by hillclimbing on the standard cost function starting from tour T and using the N_3 neighborhood structure.

Because infeasible tours may arise when subsequences are reversed and interchanged using 2-opt and 3-opt, both algorithms employ auxiliary data structures to filter out infeasible neighbors. In particular, 2-opt uses a one-dimensional array, which records, at position j, for how much further in the array reversals are legal. A similar 2-dimensional array is used for 3-opt. Note, however, that given a tour and a legal neighbor, judging whether that neighbor has lower cost can be done in constant time since only the tour edge breaks and reattachments figure in the new cost calculation (recall that we assume a symmetric distance matrix).

Psaraftis [9] was the first to study 2-opt and 3-opt algorithms for DARP. His work applies only to tours of size up to $N = 30$, and he reports that at that size, 3-opt tours are about 30% shorter on average than 2-opt tours. Stein [13] has done theoretical studies of DARP, and has shown that for sites placed in the unit square, the globally optimal tour has a length which asymtotically approaches $1.02\sqrt{2N}$ with probability 1, where N is the number of pick-ups in the tour. This work applies to our study—although we multipy position coordinates by 100 and then truncate to get integer distance matrices—and thus a value of 1020 gives us a baseline estimate of the globally optimal tour cost for $N = 50$.

Healy and Moll [4,5] consider the problem of using a secondary cost function to extend local search on DARP. In addition to primary cost (tour length) they consider as a secondary cost the ratio of tour cost to neighborhood size, which we call *cost-hood.* They alternate between these two costing functions: starting from a random tour T, they first find 2-opt(T); then they perform a limited hill-climb using the cost-hood function, which has the effect of driving the search to a new tour with a decent cost and a large neighborhood. The cost-hood phase is bounded as follows. Each cost-hood cycle involves a subsequence reversal in a tour. The cost-hood hillclimbing phase is terminated when the sum of the lengths of the reversed subsequences exceeds the number of sites in a tour. These alternating processes are repeated until a time bound is exhausted, at which point the least cost tour seen so far is reported as the result of the search. This technique worked well, and the effectiveness of the algorithm developed for DARP fell midway between 2-opt and 3-opt.

4 Learning Derived Cost Functions

While 2-opt is a fast, effective algorithm for finding legal tours of reasonable quality, the non-uniform structure of the DARP neighborhood system—a significant characteristic of DARP—plays no role in the 2-opt algorithm. Exploiting size of neighborhood structure, as well certain other problem-specific characteristics, can be accomplished using RL.

Many significant applications of RL to date involve direct stochastic interaction with a (simulated) dynamic environment. This is the case, for example, with TD-gammon [11] and elevator dispatching [3]. Both problems are explicitly stochastic in nature. For TD-gammon, a dice roll determines the possible moves at every play; for elevator dispatching, riders request elevators from particular floors of a building with a certain probability distribution while elevator simulation is in progress. Recently efforts have been made to formulate of RL for "static" settings, that is, settings in which a complete description of an instance is available at initial execution time. Our work applies to problems in this "static" class.

Our RL-based approach to local search operates in two distinct phases. In the first phase we learn a value function V, which blends various measures of solution quality into a single function. In the second phase, we use V as a replacement for c, the cost function that is traditionally used for iterative improvement.

4.1 State-Transition Costs

Our DARP-based study of RL employs a model in which each tour is identified as a state, and the transition dynamics are determined by the N_2 neighborhood described above. The available actions are: move to a neighboring tour; or terminate the trial.

We consider two different transition cost structures for this state space, M and Z, defined below. Let T be the current tour in some trial and T' a neighbor of T. Let ϵ be a small positive value to be assessed as a penalty.

M transition costs	$c(T') - c(T) + \epsilon$	if moving from T to T'
	$c(T) - c(\text{2-opt}(T'))$	if search terminates at T

Z transition costs	ϵ	if moving from T to T'
	$c(\text{2-opt}(T))$	if terminate search at T

Roughly speaking, the M structure supplies a transition cost equal to the drop in the primary cost at each state transition, plus a small penalty factor, which is included to discourage long hillclimbing trajectories. (Contrary to the usual RL formulation, we are attempting to minimize rather than maximize returns. This formulation is more common for combinatorial optimization problems.)

The M transition cost structure leads to a convienent expression for the path-cost (cost-to-go) function. If $T_1, T_2, \ldots, T_n$ is a sequence of tours visited during an M-based hill-climb, then the path cost starting from T_1 is: $c(T_2) - c(T_1) + c(T_3) - c(T_2) + \ldots + c(T_n) - c(T_n - 1) + n\epsilon = c(T_n) - c(T_1) + n\epsilon$. So, if we ignore the penalty term, minimizing returns is equivalent to choosing a path that maximizes $c(T_1) - c(T_n)$, that is, the improvement in tour cost from start to finish.

Z is modeled after Zhang and Dietterich's [12] formulation for solving the NASA space shuttle scheduling problem. Each intermediate local search step incurs a fixed small penalty as transition cost, and the final transition cost is simply the cost of the 2-opt image of the tour at termination. This reward structure attempts to create a search space in which low-cost tours can be found with the least number of search steps; the path cost can be interpreted as the expected cost of the final tour found by local search, plus e times the number of steps to that tour. Because we have simplified our study by focusing on a single large problem size, we do not perform a problem-size normalization, as is done in [12].

For either transition cost structure, performing a 2-opt run from the final state of a derived cost function hillclimbing episode guarantees termination at a local optimum of the true cost function. With DARP, 2-opt is computationally cheap and relatively effective, and is thus a sensible endgame to the value function-based local search.

4.2 Learning a Derived Cost Function

We used both TD(0) and TD(1) to learn value functions, based on the transition cost structures described above. Our learning studies applied both to particular instances of the DARP problem, and to an averaged-across-instances case. For very small instances we were able to learn using a lookup table of states, and we observed convergence to the optimal derived cost function as expected. That is, after learning, it was possible to hill-climb from any starting point to a globally optimal tour. Of course this exhibition proves little, since it is an optimal tour we seek, but we must have found it already in order for the look-up table to be effective. We also experimented with feature-based approximators in linear, quantized feature space table, CMAC, and sigmoidal feed-forward neural net formats. With such approximators we learned on problem sizes up to $N = 50$ (50 pick-ups, 50 drop-offs).

Our most successful derived cost function was represented using a linear approximator, expressed as a linear sum of the following DARP features: neighborhood size, or *hood-size*; tour *cost*; *how-near*(k), which considers the k least expensive edges of the instance, and measures how far apart these edges are in the current tour; and the feature *cost-hood*, which is the ratio of a tour's cost to its neighborhood size.

We have diverged a bit from the action/selection procedures typical of RL systems. In our case, rather than following the traditional"greedy" rule of always selecting the state with the best value, we instead mirror the behavior of first-improvement local search. When the search is at tour T with derived cost $V(T)$, we step through a random enumeration of T's neighborhood in search of the first T' for which

$$V(T') + \text{transition-cost}(T, T') \leq V(T).$$

That is, we look for the first neighbor T' such that the transition cost to T' plus path cost from T' is less than the expected return from T. If no such neighbor exists, we terminate the search at T. In this way, our algorithm can be interpreted as performing local search as before, but substituting a new derived cost function for the original primary cost function.

Though nonstandard for RL situations, this action selection procedure suffices to produce convergence of derived cost function estimates in the case of single instance learning and a linear approximator, with suitably initialized state values, because random starting points guarantee that the full state space will be explored [10].

We used the four-feature linear approximator described above, along with a bias term, in the context of two different learning/transition cost schemes, TD(0) and the Z transition cost structure, and TD(1) or MonteCarlo learning using the M transition cost system. In the TD(0)/Z case, the function approximator, f, applied to tour T, returns the expected tour cost obtained if f-hillclimbing starts at T, and ends with a 2-opt image of the resulting f local optimum. In theTD(1)/M scheme, the function approximator estimates the 2-opt image of a tour directly. We found that our linear function approximators were quite stable across multiple instances of a fixed size.

In the algorithms developed in the next section, the effectiveness of the value function as an estimator of future quality is of considerable significance. With high certainty, we would like to believe that if $V(T_1) < V(T_2)$, then hillclimbing on V as described above will lead to tours T_1^* and T_2^* for which $c(T_1^*) < c(T_2^*)$. For Z and Monte-Carlo hillclimbing we (somewhat crudely) assessed the quality of our function approximators at size $N = 50$ by measuring the correlation between the estimates provided by the value functions and the costs of the tours arrived at after hillclimbing. In each case we examined samples of 100 randomly generated tours. With each tour, we paired the estimate supplied by the value function V with the actual tour cost arrived at after V hillclimbing. In the case of the TD(0)/Z regime, we obtained correlation coefficients for these paired value samples of between .77 and .81; In the MonteCarlo (TD(1)/M) case, similar tests yielded correlation coefficients between .3 and .45.

5 Algorithms

Throughout our study, we use 2-opt as an inexpensive "one-step" operator, which, of course, leaves us at a local optimum with respect to the primary cost function. This complicates matters, because starting local search with respect to our learned cost functions works poorly when started from pure local optima—all too often the algorithm quickly returns to the local optimum starting point. We therefore "move away" from such local optima before doing M or Z hillclimbing. In what follows, "moving away" from a local optimum means doing a fixed number of hillclimbing cycles with respect to the cost-hood (cost divided by neighborhood) cost function.

We summarize below a sample of the algorithms that we considered. We perform our experiments as follows. Given a data set at $N = 50$, we do 3 3-opt runs from random starts, reporting the total time used and the best result found. We then use this total time figure to bound the running of the other much faster algorithms. Thus when we report the best value returned by 2-opt, we have run 2-opt repeatedly from different random starts until the 3-opt time bound has been used up.

Our first two algorithms are analogous to traditional measures of local search quality. In each case we successively generate 2-opt images of random tours, and then hillclimb with respect to the M and Z secondary cost functions.

- M-hillclimbing. Generate the 2-opt image of a random tour, and move several cycles away from it. Hill-climb directly using the M transition costs. When an M-optimum t_M is found, report the cost of 2-opt(t_M). If there is time remaining, generate a new 2-opt image of a random tour and repeat the above, keeping track of the best result seen so far.

- Z-hillclimbing. Exactly like M-hillclimbing, except that the Z transition costs are used.

- Modified Healy. This algorithm is based on the algorithm reported on in [5]. In that work, the authors achieve good results by alternately hillclimbing on the standard cost function, and then hillclimbing on the metric cost-hood, the ratio of the tour cost to its neighborhood size. Here we begin with a random tour, followed by a fixed number of cost-hood cycles, then a 2-opt. If there is time remaining, the cost-hood/2-opt cycle is repeated from the last 2-opt optimum obtained. When time expires, the best score found is reported. The results reported here are superior to those reported in [5], we believe, because careful coding has made the cost-hood hillclimb run faster. Notice, however, that this algorithm differs slightly from the algorithm in [5], in that the number of cost-hood cycles is fixed.

  According to one interpretation, the value function estimates the quality of a tour as a starting point for value function-based local search. The next two algorithms exploit this characteristic of the value function. They incorporate a mechanism for maintaining a list of tours—a cache—with low value function scores (good estimates). Suppose a tour T is reached, which is a local optimum with respect to the primary cost function, and which is the best tour seen so far. Then, in the hope that T represents an entire region of high quality tours, tours near T with low value-function scores are cached for future exploration. The algorithms discussed below build and explore such caches before returning to a more conventional local search regime.

- M-Cache. This time-bounded algorithm at size $N = 50$ is competitive with 3-opt. It first does M-hillclimbing, ending with a standard 2-opt optimization. This tour is checked, and if it exceeds (is worse than) the best score seen so far by more than 1%, then the search moves several cycles away from the optimum, and M-hillclimbing is tried again from this new starting position. If, however, the new optimum is a new best, or if it is within 1% of a new best, then the algorithm moves to a tour several cycles away from the optimum, searches the entire neighborhood of that tour, and caches the k neighbors with the lowest M scores (i.e. the k best M-estimates). The algorithm then works through the cache, regarding each cache entry as an M-hillclimbing starting point. If a new best tour is found, this new best is recorded and the caching process begins again from scratch. If the cache produces no new high quality tours, then the top-level M-hillclimbing mode begins again from the last tour examined. The algorithm terminates when time runs out.

- Z-Cache. This algorithm is exactly like the M-Cache algorithm above, except that Z hillclimbing replaces M hillclimbing.

We discuss the relative effectiveness of these algorithms in the next section.

6 Experimental Results

Our experiments were implemented using MacCommonLisp on a PowerMac 7600. Comparisons were done at one representative size: $N = 50$, i.e. 50 pick-ups, 50 drop-offs, and a single origin site. We generated five data sets at this size, i.e., we generated five symmetric distance matrices. The matrices were constructed by generating random points in the plane inside the square bounded by the points (0,0), (0,100), (100,100) and 100,0). The "terminal" of the data set was placed at the point (50,50). At this size, a typical 3-opt run took 15–45 minutes. For each of the five data sets, 3-opt was run from 3 random starts, and the best of these was chosen as the 3-opt score. The time for these three runs was recorded, and each of the other algorithms was run for an equivalent amount of time, which in each case involved between 200 and 500 algorithm cycles. In the chart below, a "$*$" denotes the best result found for each dataset.

Notice that both M-Cache and Z-cache are competitive with 3-opt, and each reports the best tour found on some data sets. Modified Healy is also quite effective, and does not trail the other algorithms by very much.

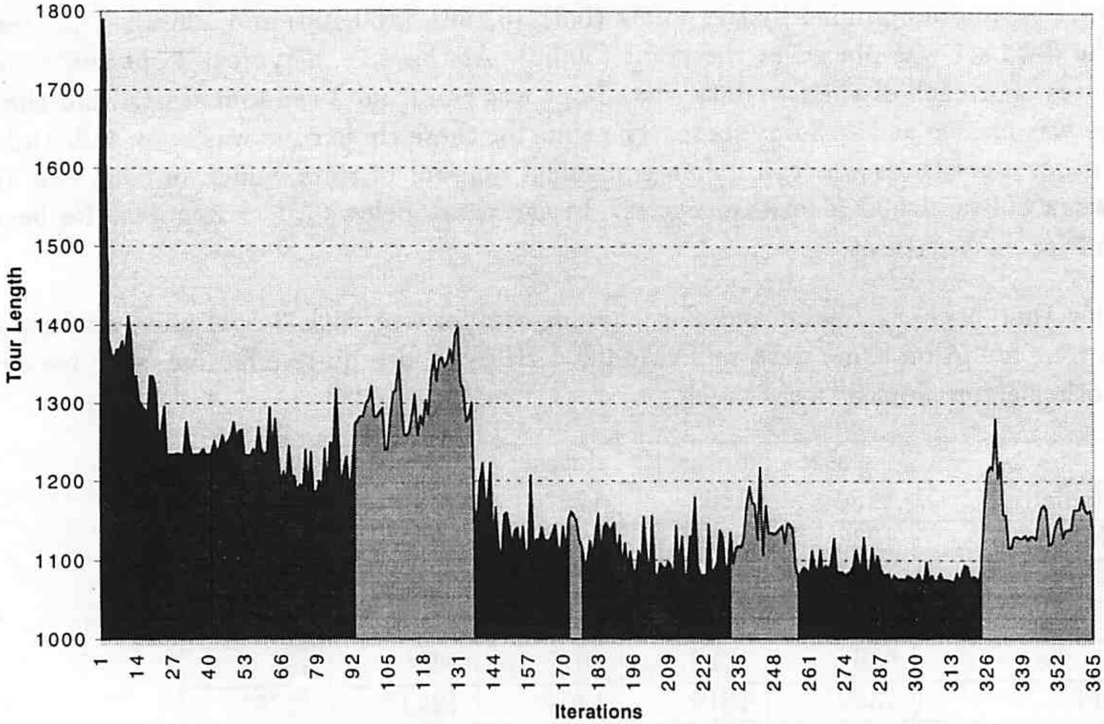
	dataset1	dataset2	dataset3	dataset4	dataset5
M-hillclimb	1250	1191	1227	1301	1275
Z-hillclimb	1238	1270	1274	1209	1320
modified Healy	1013	1008	1003	954	1021
M-cache	1027	1016	958*	995	996*
Z-cache	986	1023	1012	947*	998
2-opt	1359	1216	1304	1240	1275
3-opt	966*	974*	962	974	1010

We believe much of the success of the caching algorithms is due to the cache mechanism itself. Cached tours are tours with high quality estimates, and it makes sense to explore them as candidates before moving on to less promising tours. Indeed, this procedure makes especially good sense in the case of Z caching, since the Z value function estimates correlate so highly with outcomes. Anecdotally, we found that caching was quite effective, in the sense that the vast majority of new best tours were found in the caching phases of the algorithms. This observation is illustrated in Figure 1, which displays the costs of each of 365 starts over 45 minutes at size $N = 50$ using the M-cache algorithm. The darker portion of the graph represents starts from tours in the cache; the lighter portion represents standard starts.

7 Discussion

Our work has demonstrated a broadly applicable model that combines machine learning with local search in the context of traditional combinatorial optimization. In particular our reinforcement-learning-based value function method provides an evaluation function that blends significant secondary features with the primary cost function of the DARP problem to form a more powerful and useful combined function. While DARP has no explicit stochastic characteristics, we are able to introduce this aspect by suitably randomizing the presentation of a tour's neighborhood.

Figure 1: Hill-climbing from Cached starts



Unlike the work of Zhang and Dietterich [12] and Boyan and Moore [2], we have applied our analysis to a relatively pure optimization problem—DARP—which possesses a relatively consistent structure across problem instances. This has allowed us to construct a reliable learned cost function that apparently works well for all instances of a fixed size. Indeed we believe that with suitable normalization this cost function could be made to work for all instances of all sizes. We believe this approach could be applied to a variety of core NP-complete optimization problems with similar broad uniformity of structure, e.g. graph coloring.

Our success in developing several N_2-based learning algorithms that are competitive with 3-opt is the result of 1) the automatic construction of a value function that blends a number of features that bear on tour's potential for local improvement; 2) the high quality of learned estimates; 3) the caching mechanism, which allows us to exploit the value function as a provider of estimates of future quality; and 4) careful coding, which allowed us to "look ahead" at the evaluation of members of a tour's neighborhood without having to construct neighboring tours explicitly.

In light of these observations, the appropriateness of this approach to other optimization problems rests particularly on the identification of decisive features that indirectly bear on a problem solution's potential for local search. Moreover, in domains where speed is important, it is important to identify features that can be evaluated incrementally in a "look-ahead" fashion. We believe many traditional combinatorial optimization problems such as graph coloring, bin packing, and graph partitioning are suitable for the kind of LS-RL applications we have reported on here.

Acknowledgement

This research was supported by a grant from the Air Force Office of Scientific Research, Bolling AFB (AFOSR F49620-96-1-0254).

References

[1] Barto, A. G., Bradke, S. J., and Singh, S. P. (1995). Learning to Act Using Real-Time Dynamic Programming. *Artificial Intelligence*, 72: 81–138.

[2] Boyan, J. A., and Moore, A. W. (to appear). Using Prediction to Improve Combinatorial Optimization Search. Proceedings of AI-STATS-97.

[3] Crites, R. H., and Barto, A. G. (1996). Improving Elevator Performance Using Reinforcement Learning. In D. Touretzky, M. C. Mozer, and M. E. Hasselmo (eds.), *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pp. 1017–1023, MIT Press, Cambridge, MA.

[4] P. Healy (1991). *Sacrificing: An Augmentation of Local Search.* Ph.D. thesis, University of Massachusetts, Amherst.

[5] Healy, P., and Moll, R. (1995). A New Extension to Local Search Applied to the Dial-A-Ride Problem. *European Journal of Operations Research*, 8: 83–104.

[6] Kernigham, B. W., and Lin, S. (1970). An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49(2).

[7] Lin, S. (1965). Computer Solutions to the Traveling Salesman Problem. *The Bell System Technical Journal*, 44(10).

[8] Papadimitriou, C. H., and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ.

[9] Psaraftis, H. N. (1983). K-interchange Procedures for Local Search in a Precedence-Constrained Routing Problem. *European Journal of Operations Research*, 13:391–402.

[10] Sutton, R. and Barto, A. (to appear). *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA.

[11] Tesauro, G. J. (1992). Practical Issues in Temporal Difference Learning. *Machine Learning*, 8:257–278.

[12] Zhang, W. and Dietterich, T. G. (1995). A Reinforcement Learning Approach to Job-Shop Scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1114–1120. Morgan Kaufmann, San Francisco.

[13] Stein, D. M. (1978). An Asymptotic Probabilistic Analysis of a Routing Problem. *Math. Operations Res. J.*, 3: 89–101.