To appear in AISTATS-97

Using Prediction to Improve Combinatorial Optimization Search

Justin A. Boyan and Andrew W. Moore
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
{jab,awm}@cs.cmu.edu

Abstract

This paper describes a statistical approach to improving the performance of stochastic search algorithms for optimization. Given a search algorithm A, we learn to predict the outcome of A as a function of state features along a search trajectory. Predictions are made by a function approximator such as global or locally-weighted polynomial regression; training data is collected by Monte-Carlo simulation. Extrapolating from this data produces a new evaluation function which can bias future search trajectories toward better optima. Our implementation of this idea, STAGE, has produced very promising results on two large-scale domains.

1 Introduction

The problem of combinatorial optimization is simply stated: given a finite state space X and an objective function $f : X \to \Re$, find an optimal state $x^* = \operatorname{argmin}_{x \in X} f(x)$. Typically, X is huge, and finding an optimal x^* is intractable. However, there are many heuristic algorithms that attempt to exploit f's structure to locate good optima, e.g. hillclimbing, simulated annealing, tabu search, and genetic algorithms. All of these work by imposing a neighborhood relation on the states of X and then searching the graph that results, guided by f.

The effectiveness of these search algorithms clearly depends on the shape of f with respect to the graph structure. If f is riddled with local minima or has large regions of constant value (plateaus), then search-based algorithms are likely to fail. For this reason, the most natural f for a problem (i.e. the one which evaluates the actual quantity a human would like to minimize) is often *not* a good evaluation function to use for guiding search. In practice, a human must design a more complex evaluation functions, one which ideally shares the same global minima as f but has fewer local minima and plateaus. The coefficients of these evaluation functions are typically tweaked by hand to produce good results.

Our research has focused on machine learning methods for automatically generating high-quality evaluation functions. This paper presents a simple first step in that direction based on Monte-Carlo prediction learning.

2 Prediction for Optimization

2.1 Learning to Predict

The performance of a graph-search-based optimization algorithm depends on the state from which search starts. We express this as a mapping from starting states x to expected search result:

$$V_A(x) = \sum_{z \in X} P(x \xrightarrow{A} z) f(z) \tag{1}$$

where A refers to the search algorithm being used, and $P(x \xrightarrow{A} z)$ is the probability that a search starting from x will terminate in state z. $V_A(x)$ evaluates x's promise as a starting state for A. We seek to learn this function using a regression model such as linear regression, locally-weighted regression, or multi-layer perceptrons, where states x are represented as real-valued feature vectors. These input features may encode any relevant properties of the domain, including the original objective function $f(x)$ itself.

To illustrate, we consider the example of Figure 1. The goal is to minimize the one-dimensional function $f(x) = (|x| - 10)\cos(2\pi x)$ over the domain $X = [-10, 10]$. Assuming the natural neighborhood relation on this domain where tiny moves to the left or right are allowed, hillclimbing[1] search clearly leads to a suboptimal local minimum for all but the luckiest of starting points. However, the quality of the local minimum reached does correlate strongly with the starting position: $V_A(x) \approx |x| - 10$. Gathering data from only a few *suboptimal* trajectories, a function approximator can easily learn to predict that starting near $x = 0$ will lead to good performance.

[1] In this and other minimization problems, note that the term "hillclimbing" actually refers to stochastic greedy *descent*.

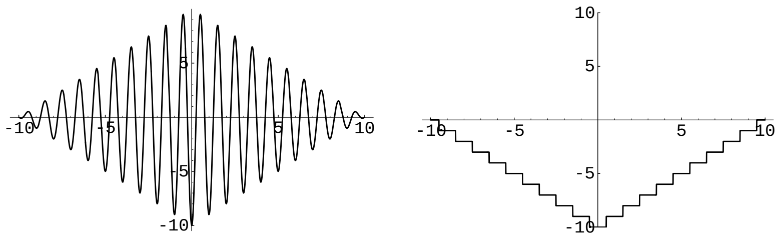

Figure 1: A one-dimensional function minimization domain (left) and the value function which predicts hillclimbing's performance on that domain (right). The value function is approximately linear in the feature $|x|$. Regression easily identifies that structure, resulting in an improved evaluation function for guiding search to a global minimum.

Training data for $V_A(x)$ may be obtained by performing Monte Carlo simulations of A from random starting states. Importantly, not only the starting states but also all other states along each search trajectory may be used as training data under the assumption that A is Markov: i.e., the future of A's search trajectory depends only on the current state. This Markov property holds trivially for stochastic hillclimbing: each state x leads stochastically to a neighboring state y for which $f(y) < f(x)$, regardless of the past history of the search; local minima are terminal states of the chain. Simulated annealing is Markovian in the expanded state space $X \times \{\text{temperature}\}$ [7]. Thus, from a search trajectory of length 100, we would obtain not one but 100 training samples for V_A.[2]

The state space X is huge, so we cannot expect our simulations to explore any significant fraction of it. Instead, we require that the function approximator predict good results for unexplored states which share many features with training states that performed well. If V_A is fairly smooth, this hope is reasonable.

[2]From a reinforcement learning perspective, A can be seen as a policy for exploring a Markov Decision Process, and V_A is the *value function* of that policy: it predicts the eventual expected outcome from every state. Since value functions satisfy the Bellman equations [1], algorithms more sophisticated than Monte-Carlo simulation with supervised learning are applicable: in particular, the TD(λ) family of temporal-difference algorithms may make better use of training data and converge faster [6]. However, our experiments reported in this paper use only supervised learning.

2.2 Using the Predictions

The learned function V_A is designed to predict which states make good starting places for search policy A. Making use of V_A requires a two-stage optimization procedure: first optimize V_A to produce a good starting place for A, and then continue by running A from there. Our algorithm, **STAGE**, integrates learning the predictions and using them to improve on the performance of A. STAGE works as follows:

1. **Initialize** the function approximator; let x be a random starting state for search.

2. **Loop** until time runs out:

   - **Optimize f using A.** From x, run search algorithm A, producing a search trajectory that ends at a local optimum z.
   - **Train $\mathbf{V_A}$.** For each point x_i on the search trajectory, use $\{[\text{features of } x_i] \mapsto f(z)\}$ as a new training pair for our function approximator.
   - **Optimize $\mathbf{V_A}$ using hillclimbing.** Continuing from z, perform a hillclimbing search on the learned objective function V_A. This results in a new state x which should be a good starting point for A. (If this search accepted no moves, so that $x = z$, then reset x to a new random starting state.)

3. **Return** the best state found.

Consider again the illustrative problem of Figure 1. We encode this problem for STAGE using a single input feature $|x|$, which makes the value function V_A approximately linear. Not surprisingly, running STAGE with a linear function approximator performs efficiently on this problem: after only a few trajectories through the space, it learns $V_A(x) \approx |x| - 10$. Hillclimbing on $V_A(x)$, then, leads directly to the global optimum.

This problem is contrived, but we think its essential property—that features of the state help to predict the performance of an optimizer—does indeed hold in many practical domains. We have obtained encouraging preliminary results in the domains of channel routing and map layout.

3 Results

3.1 Channel Routing

The problem of "Manhattan channel routing" is an important subtask of VLSI circuit design [8]. Given two rows of labelled pins across a rectangular channel, we must connect like-labelled pins to one another by placing wire segments into vertical and horizontal tracks (see Figure 2). Segments may cross but not otherwise overlap. The objective is to minimize the area of the channel's rectangular bounding box—or equivalently, to minimize the number of different horizontal tracks needed.

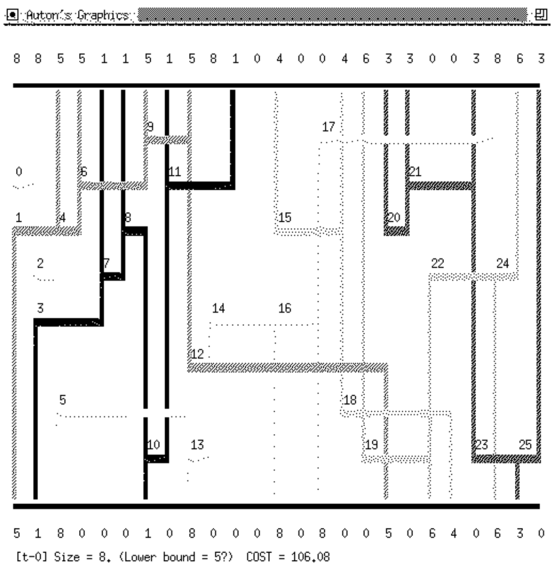

Figure 2: A small channel routing problem.

In their simulated-annealing approach to the problem, Wong et. al. say,

> Clearly, the objective function to be minimized is the channel width w. However, w is too crude a measure of the quality of intermediate solutions. Instead, for any valid partition π, the following cost function is used:

$$C = w^2 + \lambda_p \cdot p^2 + \lambda_U \cdot U \tag{2}$$

> where p is the longest path length of G_π [a graph induced by the partitioning], both λ_p and λ_U are constants, and ... $U = \sum_{i=1}^{w} u_i^2$, where u_i is the fraction of track i that is unoccupied. [8]

They hand-tuned the coefficients and set $\lambda_p = 0.5, \quad \lambda_U = 10$. To apply STAGE to this problem, we began with not the contrived function C but the natural objective function $f(x) = w$. The additional objective function terms used in Equation 2, p and U, along with w itself, were given as the three input features to STAGE's function approximator.

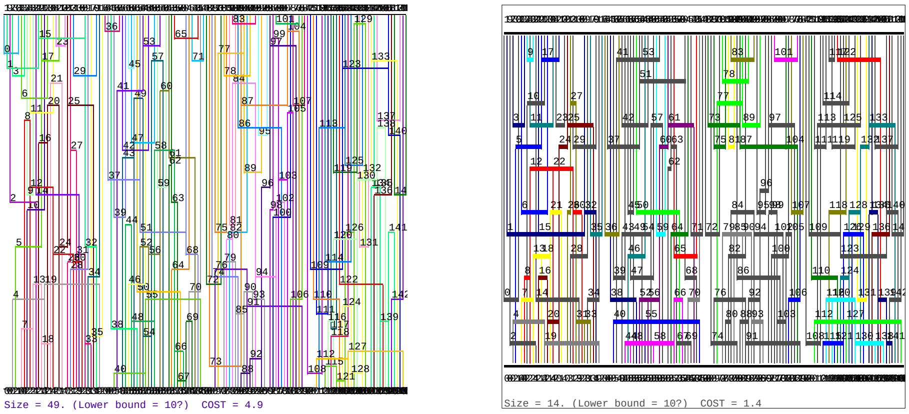

Figure 3: On a large channel routing instance, typical solutions found by hillclimbing (49 tracks, at left) and STAGE (14 tracks).

The upper half of Table 1 summarizes the results of hillclimbing, STAGE, and simulated annealing on a large channel-routing problem instance. All algorithms were run 21 times and allowed a total of $100,000$ function evaluations; the results of the best and median of the 21 runs is reported. The simulated annealing experiments made use of the successful "modified Lam" adaptive annealing schedule [5, §4.5].

Experiment (B) shows that hillclimbing gets stuck in very poor local optima, even when allowed to consider $100,000$ random moves. Experiment (E) shows that simulated annealing, as used with the objective function of [8], does considerably better. Surprisingly, the annealer of experiment (F) does better still. It seems that the "crude" evaluation function $f(x) = w$ allows a long simulated annealing run to effectively random-walk along the ridge of all solutions of equal cost w, and given enough time it will

PROBLEM	ALGORITHM	Performance best, median of 21
Channel Routing (YK4)	(A) Global optimum for problem	10
	(B) Hillclimbing	44, 47
	(C) STAGE(HC), linear regression	12, 14
	(D) STAGE(HC), quadratic regression	13, 14
	(E) Simulated annealing, $f(x) = w^2 + 0.5p^2 + 10U$	18, 20
	(F) Simulated annealing, $f(x) = w$	15, 16
	(G) Hillclimbing, accept equi-cost	14, 16
	(H) Hillclimbing, random walk	23, 26
	(I) Modified STAGE, use smoothed-$f(x)$ instead of V_A	16, 26
Map Layout (US49)	(J) Global optimum for problem	≈ 0.02
	(K) Hillclimbing	0.105, 0.163
	(L) STAGE(HC), linear regression	0.042, 0.080
	(M) STAGE(HC), quadratic regression	0.042, 0.068
	(N) Simulated annealing	0.033, 0.036

Table 1: Comparative results on two minimization problem instances. All algorithms were run 21 times and held to the same fixed number of evaluations of the objective function f. STAGE successfully learned to improve on the performance of hillclimbing in both cases. On the channel-routing problem, it outperformed simulated annealing as well.

fortuitously find a hole in the ridge. In fact, hillclimbing modified to accept equi-cost moves performed just as well (G).

The hillclimber we gave to STAGE was the fast but weakly-performing one of experiment (B). Simple linear and quadratic regression models were used for learning. The results (C,D) show that STAGE learned to optimize superbly, not only improving on the performance of (B) as it was trained to do, but also finding better solutions than the best simulated annealing runs. This seems too good to be true; did STAGE really work according to its design?

We considered and eliminated several hypotheses. (1) Since STAGE alternates between simple hillclimbing and another policy, perhaps it simply benefits from having more random exploration. This is not the case: we tried the search policy of alternating hillclimbing with 50 steps of random walk, and its performance (H) was much worse than STAGE's. (2) The function approximator may simply be smoothing $f(x)$, which helps eliminate local minima and plateaus. No, we tried a variant of STAGE which learned to smooth $f(x)$ directly instead of learning V_A (I); this also produced much less improvement than STAGE. (3) The input features happen to be excellent, providing an easy answer. This explanation turns out to be correct; we discuss how at the end of Section 3.3.

3.2 Area-Reweighted Map Layout

The second problem we considered was that of redrawing a map of the United States such that each state's area is proportional to its population. (Such maps provide a useful means of visualizing geographic data and appear in some textbooks, e.g. [4].) The goal of optimization is to best meet the new area targets for each state while minimally distorting the states' shapes and borders.

We set up the problem as follows. The map was represented as a collection of 162 points in 2-space; each state was defined as a polygon over a subset of those points. The search operator consisted of choosing one of the points and perturbing it slightly with uniform noise; perturbations that would cause two edges to cross were disallowed. The objective function was defined as

$$f(x) = \sum_{s \in \text{states}} \left(\Delta\text{area}(s) + \sum_{v \in \text{vertices(s)}} (\Delta\text{gape}(v) + \Delta\text{orient}(v) + \Delta\text{segfrac}(v)) \right)$$

where $\Delta\text{area}(s)$ penalizes a state for missing its area target and the other three terms penalize a state for being shaped differently than in the true U.S. map. Because of the complexity of the objective function and domain constraints, this problem is most conveniently handled by heuristic-search optimization techniques.

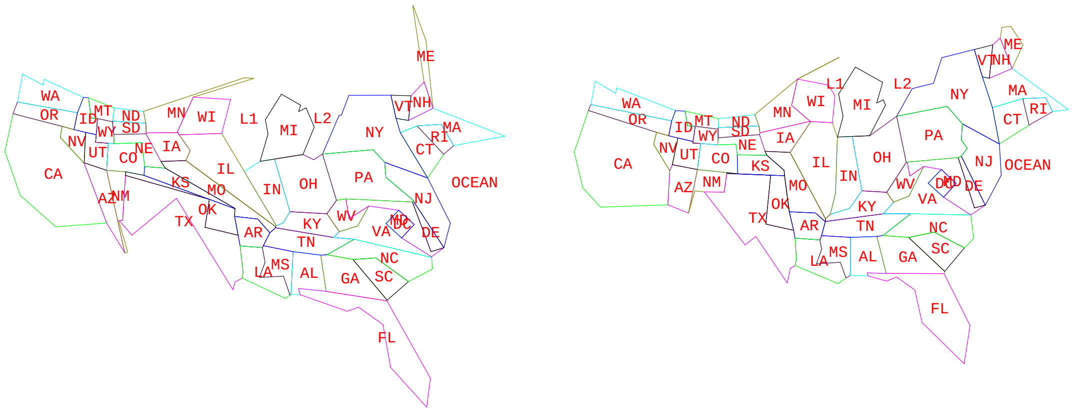

Figure 4: Each state's target area is proportional to its electoral vote for U.S. president. A good hillclimbing solution (f=0.115, at left) and a good STAGE solution (f=0.043).

Our results from this domain are shown in Table 1, experiments (K) through (N). All algorithms were held to one million function evaluations. For STAGE, we represented each state with four simple features: the sum, over all polygons, of the Δarea, Δgape, Δorient, and Δsegfrac terms. Applying STAGE to learn from the hillclimbing policy produced a significant improvement, from a median performance of 0.163 to 0.068, although in this case, STAGE did not happen to outperform simulated annealing as well. Figure 4 depicts solutions found by hillclimbing and STAGE.

3.3 Transfer

There is a computational cost to training a function approximator on V_A. For some problems this computational cost is worth it in comparison to a non-learning method, because a better or equally good solution is obtained with overall less computation. But in those cases where we use more computation, the STAGE method may nevertheless be preferable if we are then asked to solve further similar problems (e.g. a new channel routing problem with different pin assignments). Then we can hope that the computation we invested in solving the first problem will pay off in the second, and future, problems because we will will already have a V_A estimate. We call this effect *transfer* and the extent to which it occurs is largely an empirical question.

To investigate the potential for transfer, we re-ran experiment (C) on a suite of eight problems from the channel routing literature [3]. Table 2 summarizes the results and gives the coefficients of the linear evaluation function learned (independently) for each problem. To make the similarities easier to see in the table, we have normalized the coefficients so that their squares sum to one; note that the search behavior of an evaluation function is invariant under linear transformations.

Problem instance	lower bound	best-of-3 hillclimbing	best-of-3 STAGE	learned coefficients $< w, p, U >$
YK4	10	44	12	$< 0.71,\ 0.05, -0.70 >$
HYC1	8	8	8	$< 0.52,\ 0.83, -0.19 >$
HYC2	9	9	9	$< 0.71,\ 0.21, -0.67 >$
HYC3	11	15	12	$< 0.72,\ 0.30, -0.62 >$
HYC4	20	42	23	$< 0.71,\ 0.03, -0.71 >$
HYC5	35	47	38	$< 0.69,\ 0.14, -0.71 >$
HYC6	50	69	51	$< 0.70,\ 0.05, -0.71 >$
HYC7	39	73	42	$< 0.71,\ 0.13, -0.69 >$
HYC8	21	44	25	$< 0.71,\ 0.03, -0.70 >$

Table 2: STAGE results on eight problems from [3]. The coefficients have been normalized so that their squares sum to one.

The similarities among the learned evaluation functions are striking. Like the hand-tuned cost function C of [8] (Equation 2), all but one of the STAGE-learned cost functions (HYC1) assigned a relatively large positive weight to feature w and a small positive weight to feature p. Unlike the hand-tuned C, all the STAGE runs assigned a *negative* weight to feature U. The similarity of the learned functions suggests that transfer between problem instances would indeed be fruitful.

The assignment of a negative coefficient to U is surprising, because U measures the sparsity of the horizontal tracks. U correlates strongly positively with the objective function to be minimized; a term of $-U$ in the evaluation function ought to pull the search toward terrible solutions in which each subnet occupies its own track. However, the positive coefficient on w cancels out this bias, and in fact a proper balance between the two terms can be shown to bias search toward solutions with an *uneven* distribution of track sparsities. Although this characteristic is not itself the mark of a high-quality solution, it does help lead hillclimbing search to high-quality solutions. STAGE successfully discovered and exploited this predictive combination of features.

4 Discussion

Under what conditions will STAGE work? Intuitively, STAGE maps out the attracting basins of a domain's local minima. When there is a coherent structure among these attracting basins, STAGE can exploit it. Identifying such a coherent structure depends crucially on the user-selected state features, the domain's move operators, and the regression models considered. What this paper has shown is that for two large-scale problems, with very simple choices of features, operators, and models, a useful structure can be identified and exploited.

A very relevant investigation by Boese et. al.[2] gives further reasons for optimism. They studied the set of local minima reached by independent runs of hillclimbing on a traveling salesman problem and a graph bisection problem. They found a "big valley" structure to the set of minima: the better the local minimum, the closer (in terms of a natural distance metric) it tended to be to other local minima. This led them to recommend a two-phase "adaptive multi-start" hillclimbing technique very similar to STAGE. The only significant difference is that they hand-build a problem-specific routine for finding good new starting states, whereas STAGE uses machine learning to do the same.

Zhang and Dietterich have explored another way to use learning to improve combinatorial optimization: they learn a search strategy from scratch using online value iteration [9]. By contrast, STAGE begins with an already-given search strategy and uses prediction to learn to improve on it. Zhang and Dietterich reported success in transferring learned search control knowledge from simple job-shop scheduling instances to more complex ones.

Our future work includes developing the STAGE algorithm further and applying it to additional optimization domains. We also will investigate whether performance can be improved by using better state features, more sophisticated function approximators, and simulated annealing rather than hillclimbing in the inner loop. One fascinating direction for exploration is to apply STAGE recursively to itself, resulting in an automatically-generated multi-stage optimization algorithm.

Acknowledgements

We are grateful to Scott Fahlman and Shumeet Baluja for helpful advice. This work has been supported by NSF Grant IRI-9214873, a NASA Space Grant Fellowship, and the Engineering Design Research Center, an NSF Engineering Research Center.

References

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16:101–113, 1994.

[3] H-Y. Chao and M. P. Harper. An efficient lower bound algorithm for channel routing. *Integration: The VLSI Journal*, 1996.

[4] S. E. Frantzich and S. L. Percy. *American Government*. Brown & Benchmark, Madison, WI, 1994.

[5] E. Ochotta. *Synthesis of High-Performance Analog Cells in ASTRX/OBLX*. PhD thesis, Carnegie Mellon University Department of Electrical and Computer Engineering, April 1994.

[6] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 1988.

[7] P. van Laarhoven. *Simulated Annealing: Theory and Applications*. Kluwer Academic, 1987.

[8] D. F. Wong, H.W. Leong, and C.L. Liu. *Simulated Annealing for VLSI Design*. Kluwer Academic, 1988.

[9] W. Zhang. *Reinforcement Learning for Job-Shop Scheduling*. PhD thesis, Oregon State University, 1996.