Journal of Machine Learning Research 1 (????) ??–?? Submitted 4/00; published ?/??

Learning Evaluation Functions to Improve Local Search

Justin A. Boyan JBOYAN@ARC.NASA.GOV
NASA Ames Research Center
Computational Sciences Division
Moffett Field, CA 94035 USA

Andrew W. Moore AWM@CS.CMU.EDU
Carnegie Mellon University
Computer Science Department
Pittsburgh, PA 15213 USA

Abstract

This paper describes STAGE, a learning algorithm that automatically improves search performance on large-scale optimization problems. STAGE learns an evaluation function that predicts the outcome of a local search algorithm, such as hillclimbing or WALKSAT, from features of states visited during search. The learned evaluation function is used to bias future search trajectories toward better optima on the same problem. This paper presents the STAGE algorithm; an extension, X-STAGE, that transfers learned evaluation functions to new, similar optimization problems; and empirical results on seven large-scale optimization domains: bin-packing, channel routing, Bayes network structure-finding, radiotherapy treatment planning, cartogram design, Boolean satisfiability, and Boggle board setup.

1. Introduction

Global optimization problems are ubiquitous in areas such as VLSI design, drug design, job-shop scheduling, inventory management, medical treatment planning, and transportation planning. Formally, an instance of a global optimization problem consists of a *state space* X and an *objective function* $\text{Obj} : X \to \Re$; the goal is to find a state $x^* \in X$ that minimizes Obj. If X is large, then finding x^* is generally intractable unless the problem has a specialized global structure, such as a linear program. However, many general-purpose *local search* algorithms attempt to exploit Obj's local structure to locate good approximate optima; these include iterative improvement (hillclimbing), simulated annealing, tabu search, and WALKSAT. They work by imposing a neighborhood relation on the states of X and then searching the graph that results, guided by Obj.

Local search has been likened to "trying to find the top of Mount Everest in a thick fog while suffering from amnesia" (Russell & Norvig, 1995, p.111). In this metaphor, the climber considers each step by consulting an altimeter and deciding whether to take the step based on the change in altitude. But suppose the climber has access to not only an altimeter, but also additional senses and instruments—for example, the current x and y location, the slope of the ground underfoot, and whether or not the current location is on a trail. These additional "features" may enable the climber to make a more informed, more foresightful, evaluation of whether to take a step.

©???? AI Access Foundation and Morgan Kaufmann Publishers. All rights reserved.

Additional state features are generally plentiful in real optimization domains. Practitioners of local search algorithms often append such features to their objective function as extra terms; they may then spend considerable effort tweaking those terms' coefficients. This excerpt, from a book on simulated annealing for VLSI design (Wong, Leong, & Liu, 1988), is typical:

> 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, the following cost function is used:
>
> $$C = w^2 + \lambda_p \cdot p^2 + \lambda_U \cdot U \tag{1}$$

The state feature U measures the sparsity of the horizontal tracks, while p measures the longest path length in the current partition. In this application, the authors hand-tuned the coefficients of the extra state features p^2 and U, setting $\lambda_p = 0.5$ and $\lambda_U = 10$. (We will show that our algorithm learned to assign, counterintuitively, a *negative* value to λ_U, and achieved much better performance.) Similar examples of evaluation functions being manually configured and tuned for good performance can be found in, e.g., (Falkenauer & Delchambre, 1992; Szykman & Cagan, 1995).

We address the following question: can we exploit extra features of an optimization problem to generate improved evaluation functions automatically, thereby guiding search to better solutions?

This paper presents one approach to doing so, based on machine learning: the Stage algorithm. We present Stage in Section 2. Section 3 reports empirical results of Stage on a set of seven domains, including an analysis of why it helped considerably on most domains but failed to help on another. Section 4 presents a natural extension to Stage in which the learned evaluation functions are *transferred* among similar problem instances. Finally, Section 5 places Stage in the context of related work, and Section 6 outlines directions for future research in this area.

2. The STAGE Algorithm

The Stage algorithm automatically constructs predictive evaluation functions by analyzing search trajectories. It then uses these evaluation functions to guide further search on the same problem instance. In this section we present Stage, including its foundations in reinforcement learning, and illustrate its performance on a simple example.

2.1 Learning to Predict

The performance of a local search algorithm depends on the state from which the search starts. We can express this dependence in a mapping from starting state x to expected search result:

$$V^\pi(x) \stackrel{\text{def}}{=} \text{expected best Obj value seen on a trajectory that starts from state } x \text{ and follows local search method } \pi \tag{2}$$

Here, π represents a local search method such as hillclimbing or simulated annealing. $V^\pi(x)$ evaluates x's *promise* as a starting state for π.

For example, suppose we want to minimize the one-dimensional function $\text{Obj}(x) = (|x| - 10)\cos(2\pi x)$ over the domain $X = [-10, 10]$, as depicted in Figure 1. Assuming a neighborhood structure on this domain where tiny moves to the left or right are allowed, hillclimbing (greedy descent) 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: $V^\pi(x) \approx |x| - 10$. 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.

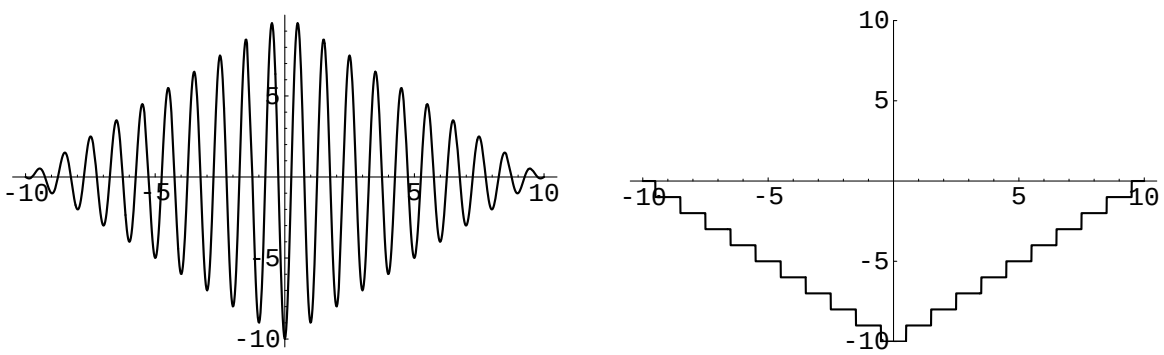

Figure 1: Left: $\text{Obj}(x)$ for a one-dimensional minimization domain. Right: the value function $V^\pi(x)$ that predicts hillclimbing's performance on that domain.

We approximate V^π using a function approximation model such as polynomial regression, where states are encoded as real-valued feature vectors. As discussed above, these input features may encode any relevant properties of the state, including the original objective function $\text{Obj}(x)$ itself. We denote the mapping from states to features by $F : X \to \Re^D$, and our approximation of $V^\pi(x)$ by $\tilde{V}^\pi(F(x))$. Our experiments reported here all use simple linear or quadratic regression models to fit $\tilde{V}^\pi$, since incremental training of these models can be made extremely efficient in time and memory (Press, Teukolsky, Vetterling, & Flannery, 1992; Boyan, 1998). These models also have the property that they aggressively extrapolate trends from their training samples. Such extrapolation benefits Stage in its search for promising, previously unvisited states.

Foundations of V^π

V^π is well-defined for any local search procedure π, assuming that the objective function is bounded below. Assuming also that π is guaranteed to terminate, training data for supervised learning of $\tilde{V}^\pi$ may readily be obtained by running π from different starting points. Assuming further that the algorithm π behaves as a Markov chain—i.e., the probability of moving from state x to x' is the same no matter when x is visited and what states were visited previously—then intermediate states of each simulated trajectory may also be considered alternate "starting points" for that search, and thus used as training data for $\tilde{V}^\pi$ as well. This insight enables us to get not one but perhaps hundreds of pieces of training data from each trajectory sampled. The extra training points collected this way may be highly

correlated, so their effect on optimization performance is an empirical question; our results show that the improvement can be substantial.

One further assumption on the local search procedure is of interest: monotonicity. The procedure π is said to be *monotonic* if the objective function never increases along a search trajectory. Under this assumption and the aforementioned Markov assumption, π is equivalent to a fixed policy for a Markov Decision Process, and V^π is the *value function* of that policy (e.g., (Puterman, 1994)). (Monotonicity allows us to rewrite Equation 2 as an expected sum of one-step rewards, as required by the MDP formalism.) Value functions satisfy local consistency equations (Bellman, 1957), so 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, converge faster, and use less memory during training (Sutton, 1988). LSTD(λ), an efficient least-squares formulation of TD(λ) (Boyan, 2000), applies specifically to the linear approximation architectures we use. However, the experiments reported in this paper use only supervised learning.

2.2 Using the Predictions

The learned evaluation function $\tilde{V}^\pi(F(x))$ evaluates how promising x is as a starting point for algorithm π. To find the best starting point, we must optimize $\tilde{V}^\pi$ over X.

Note that even if $\tilde{V}^\pi$ is smooth with respect to the feature space—as it surely will be if we represent $\tilde{V}^\pi$ with a simple model like quadratic regression—it may still give rise to a complex cost surface with respect to the neighborhood structure on X. The existence of a state with features similar to those of the current state does not imply there is a step in state-space that will take us to that state. Thus, to optimize $\tilde{V}^\pi$ we must run a second stage of stochastic local search—but with $\tilde{V}^\pi(F(x))$ instead of Obj(x) as the guiding evaluation function.

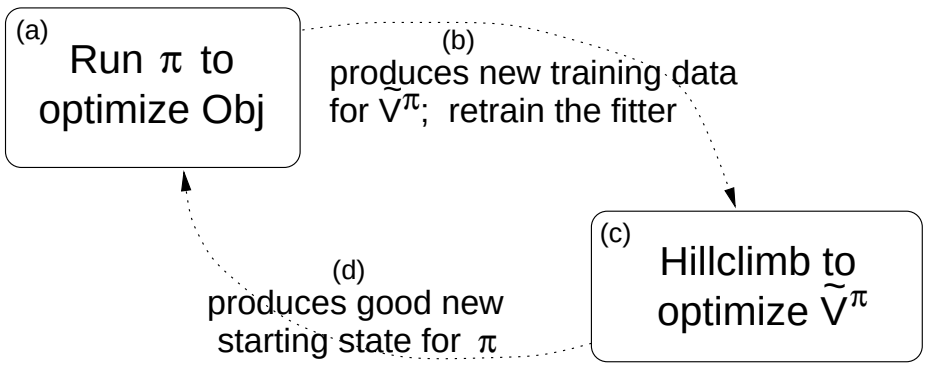

Figure 2: A diagram of the main loop of Stage

The Stage algorithm provides a framework for learning and exploiting $\tilde{V}^\pi$ on a single optimization instance. As illustrated in Figure 2, Stage repeatedly alternates between two different stages of local search: (a) running the original method π on Obj, and (c) running hillclimbing on $\tilde{V}^\pi$ to find a promising new starting state for π. Thus, Stage can be viewed as a smart multi-restart approach to local search.

Stage plots a single long trajectory through the state space, periodically switching between the original objective function Obj(x) and the newly-learned evaluation function $\tilde{V}^\pi(F(x))$. The trajectory is only broken if the $\tilde{V}^\pi$ search phase accepts no moves, indicating that x is a local minimum of *both* evaluation functions. When this occurs, Stage resets the search to a random starting state. A complete description of the algorithm is given in Table 1.

STAGE(X, π, Obj, ObjBound, F, *Fit*, Patience, TotEvals):
Given:

- a state space X (and a method for generating a random start state in X)
- a local search procedure π that is Markovian and proper (e.g., hillclimbing)
- an objective function, Obj : $X \rightarrow \Re$, to be minimized
- a lower bound on Obj, ObjBound $\in \Re$ (or $-\infty$ if no bound is known)
- a mapping F from states to feature vectors, $F : X \rightarrow \Re^D$
- a function approximator *Fit*
- a patience parameter Patience governing stochastic hillclimbing on $\tilde{V}^\pi$
- TotEvals, the number of state evaluations allotted for this run.

1. **Initialize** the function approximator; let $x_0 \in X$ be a random starting state.
2. **Loop** until the number of states evaluated exceeds TotEvals:
   (a) **Optimize Obj using π.** From x_0, run search algorithm π, producing a search trajectory $(x_0, x_1, x_2, \ldots, x_T)$.
   (b) **Train $\tilde{V}^\pi$.** For each point x_i on the search trajectory, define $y_i := \min_{j=i\ldots T} \text{Obj}(x_j)$, and add the pair $\{F(x_i) \mapsto y_i\}$ to the training set for *Fit*. Retrain *Fit*, and call the resulting learned evaluation function $\tilde{V}^\pi$.
   (c) **Optimize $\tilde{V}^\pi$ using hillclimbing.** Continuing from x_T, optimize $\tilde{V}^\pi(F(x))$ by performing a stochastic hillclimbing search. Cut off the search when either Patience consecutive moves produce no improvement, or a candidate state z_{t+1} is predicted to be impossibly good, i.e. $\tilde{V}^\pi(F(z_{t+1})) <$ ObjBound. Denote this search trajectory by $(z_0, z_1, \ldots, z_t)$.
   (d) **Set smart restart state.** Set $x_0 := z_t$. But in the event that the $\tilde{V}^\pi$ hillclimbing search accepted no moves (i.e., $z_t = x_T$), then reset x_0 to a new random starting state.
3. **Return** the best state found.

Table 1: The Stage algorithm.

2.3 Two Examples

Consider again the one-dimensional wave problem of Figure 1. We encode this problem for Stage using the single input feature x, and quadratic regression to represent $\tilde{V}^\pi$. Not sur-

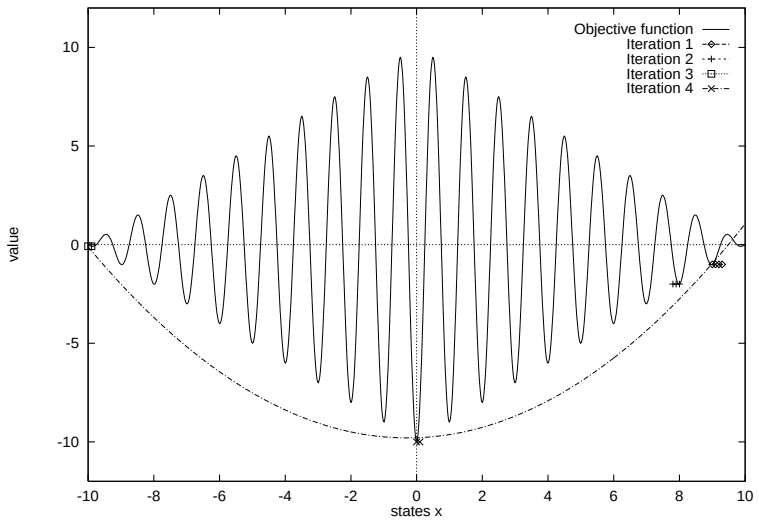

Figure 3: Stage on the 1-D minimization example

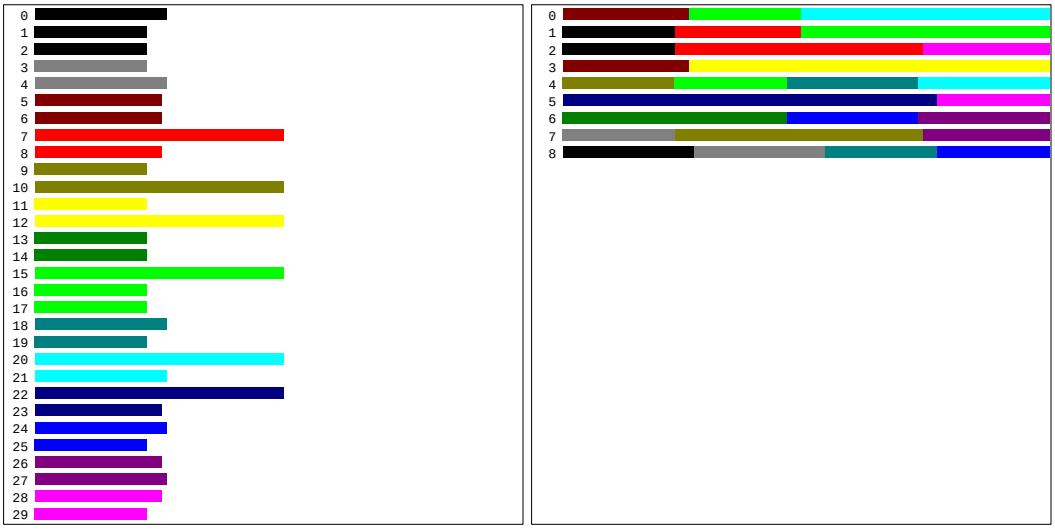

Figure 4: A small example bin-packing instance

prisingly, Stage performs efficiently on this problem: after only a few trajectories through the space, it learns a U-shaped parabolic approximation to V^π. Hillclimbing on $\tilde{V}^\pi$, then, leads directly to the basin of the global optimum.

This problem is contrived, but its essential property—that state-space features help to predict the performance of an optimizer—does hold in many practical domains. Simulated annealing does not take advantage of this property, and indeed performs poorly on this problem. This problem also illustrates that Stage does more than simply smooth out the waves in Obj(x): doing so here would produce an unhelpful flat function. Stage does smooth out the waves, but in a way that incorporates predictive knowledge about local search.

We now consider a more realistic illustration of STAGE in operation, on the NP-complete problem of bin-packing (Coffman, Garey, & Johnson, 1996). We are given a *bin capacity* C and a list $L = (a_1, a_2, ...a_n)$ of n *items*, each having a *size* $s(a_i) > 0$. The goal is to pack the items into as few bins as possible. Figure 4 depicts an example bin-packing instance with 30 items. Packed optimally, these items fill 9 bins exactly to capacity.

To apply local search, we define a neighborhood operator that moves a single random item to a random new bin having sufficient spare capacity. STAGE predicts the outcome of stochastic hillclimbing using quadratic regression over two features of the state x:

1. The actual objective function, Obj = # of bins used.
2. Var = the variance in fullness of the non-empty bins. This feature is similar to a cost function term introduced in (Falkenauer & Delchambre, 1992).

STAGE proceeds as follows:

- Iteration 1, Step (a): STAGE hillclimbs from the starting state (Obj $= 30$, Var $= 0.011$) to a local optimum (Obj $= 13$, Var $= 0.019$).

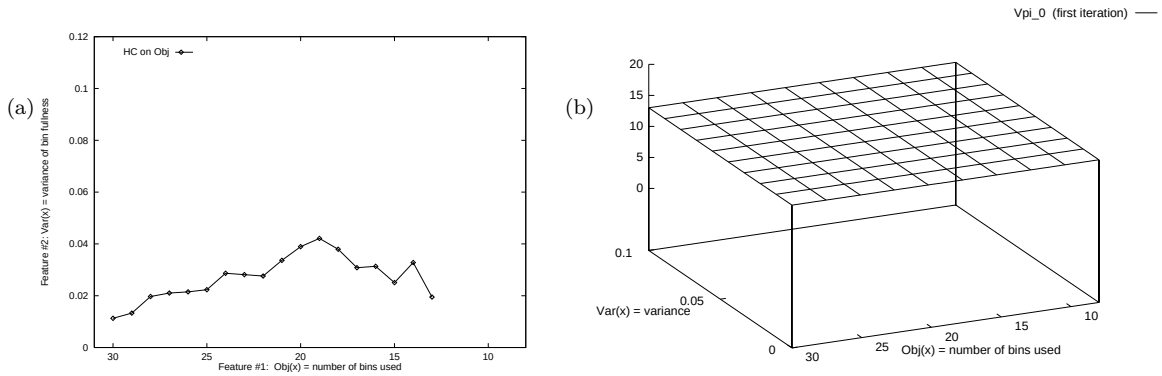

Figure 5: Iteration #1

- Iteration 1, Step (b): STAGE builds a training set that associates the feature vectors of each visited state with the observed outcome, 13. Training produces the flat $\tilde{V}^\pi$ function shown.

- Iteration 1, Steps (c) and (d): Hillclimbing on this flat $\tilde{V}^\pi$ accepts no moves, so STAGE resets to the initial state.

- Iteration 2, Step (a): Hillclimbing on Obj again, STAGE produces a new trajectory that happens to do better than the first, finishing at a local optimum (Obj $= 11$, Var $=$ 0.022).

- Iteration 2, Step (b): Our training set is augmented with target values of 11 for all states on the new trajectory. The resulting quadratic $\tilde{V}^\pi$ already has significant structure. Note how the contour lines of $\tilde{V}^\pi$, shown on the base of the surface plot,

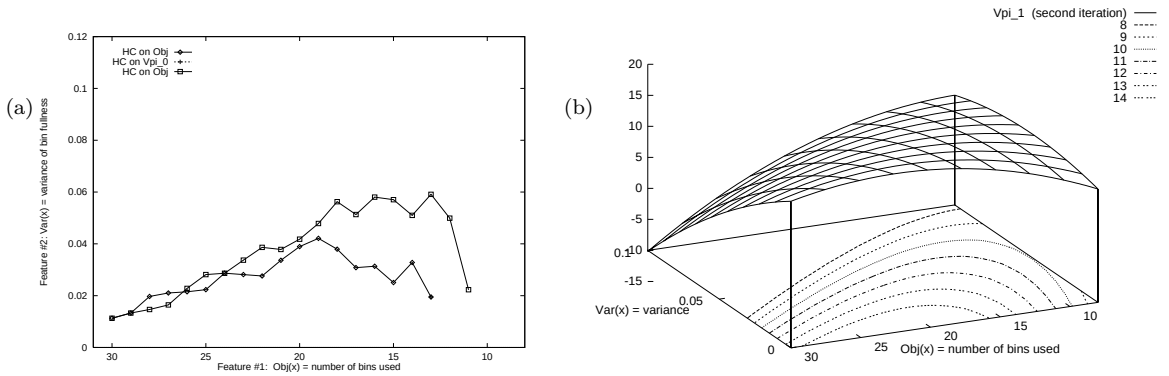

Figure 6: Iteration #2

correspond to smoothed versions of the trajectories in our training set. Extrapolating, $\tilde{V}^\pi$ predicts that the the best starting points for π are on arcs with higher Var(x).

- Iteration 2, Steps (c) and (d): STAGE hillclimbs on the learned $\tilde{V}^\pi$ to try to find a good starting point. The trajectory, shown as a dashed line in Figure 7(a), goes from $(\text{Obj} = 11, \text{Var} = 0.022)$ up to $(\text{Obj} = 12, \text{Var} = 0.105)$. Note that the search was willing to accept some harm to the true objective function during this stage.

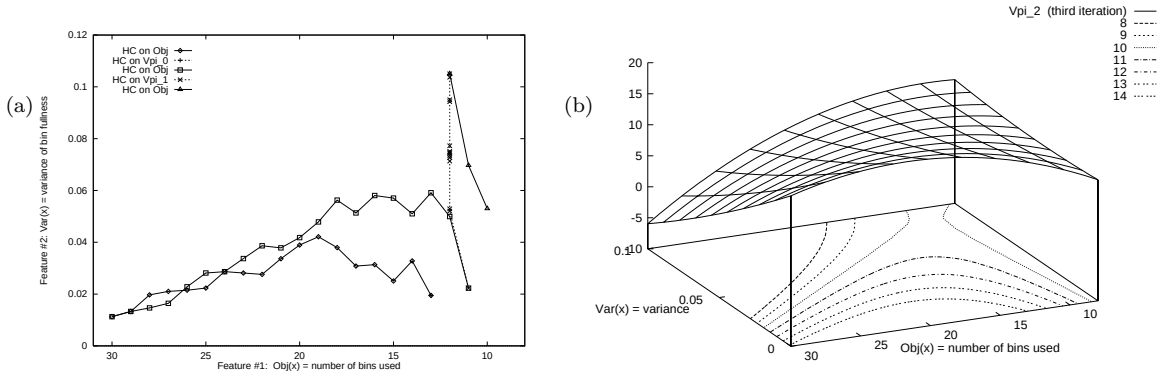

Figure 7: Iteration #3

- Iteration 3, Step (a): This time, hillclimbing on Obj does indeed lead to a yet better local optimum at $(\text{Obj} = 10, \text{Var} = 0.053)$.

- During further iterations, the approximation of $\tilde{V}^\pi$ is further refined. Continuing to alternate between hillclimbing on Obj and hillclimbing on $\tilde{V}^\pi$, STAGE manages to discover the global optimum at $(\text{Obj} = 9, \text{Var} = 0)$ on iteration seven.

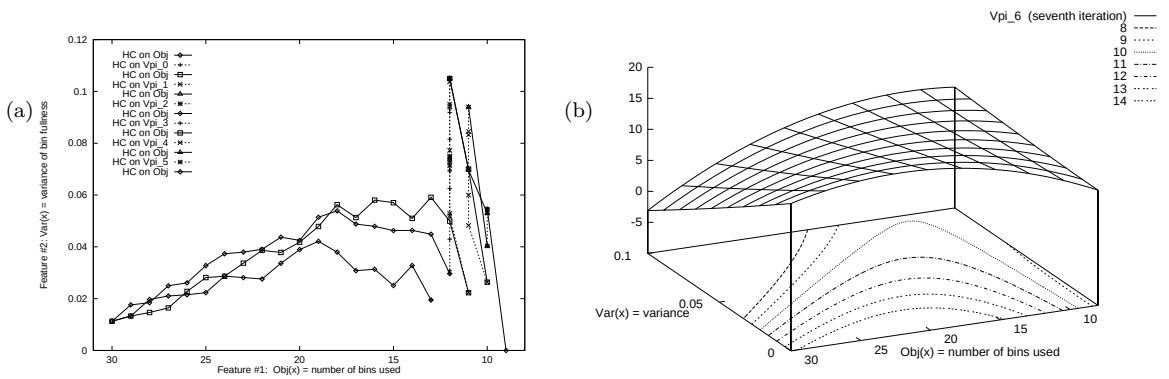

Figure 8: Iteration #7

Stage's complete trajectory is plotted in Figure 8(a). This example illustrates Stage's potential to exploit high-level state features to improve performance on combinatorial optimization problems. It also illustrates the benefit of training $\tilde{V}^\pi$ on entire trajectories, not just starting states: in this run a useful quadratic approximation was learned after only two iterations, both of which started from the same initial state. Results on larger bin-packing instances, and on many other large-scale domains, are presented in the following section.

3. Results

The results of an extensive experimental evaluation of Stage are summarized in Table 2. For seven problems with widely varying characteristics, we contrast the performance of Stage with that of multi-restart stochastic hillclimbing, simulated annealing, and domain-specific algorithms where applicable. The hillclimbing runs accepted equi-cost moves and restarted whenever *patience* consecutive moves produced no improvement. The simulated annealing runs made use of the successful "modified Lam" adaptive annealing schedule (Ochotta, 1994, §4.5); its parameters were hand-tuned to perform well across the whole range of problems but not exhaustively optimized for each individual problem instance. On each instance, all algorithms were held to the same number M of total search moves considered, and run N times.

Space limits prevent us from giving full implementation details of all seven domains. Instead, we aim to give enough detail that the domains can be understood and the results appreciated. Readers wishing to replicate the experiments shown here are asked to refer to (Boyan, 1998).

3.1 Bin-packing

The first set of results is from a 250-item benchmark bin-packing instance (u250_13, from (Falkenauer & Delchambre, 1992)). Table 2 compares Stage's performance with that of hill-climbing, simulated annealing, and *best-fit-randomized* (Coffman et al., 1996), a bin-packing

algorithm with good worst-case performance guarantees. Stage significantly outperforms all of these. We obtained similar results for all 20 instances in the u250 suite (Boyan, 1998).

How did Stage succeed? The Stage runs followed the same pattern as the runs on the small example bin-packing instance presented above. Stage's learned evaluation function, $\tilde{V}^{\pi}$(Figure 9), successfully trades off the original objective and the additional bin-variance feature to identify promising start states. As in the example instance (Figure 8), Stage learns to direct the search toward the high-variance states from which hillclimbing is predicted to excel.

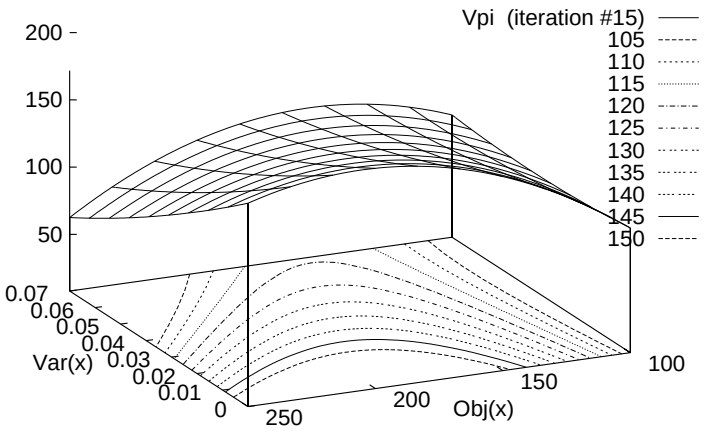

Figure 9: An evaluation function learned by Stage on bin-packing instance u250_13. Stage learns that the states with higher variance (wider arcs of the contour plot) are promising start states for hillclimbing.

3.2 Channel Routing

The problem of "Manhattan channel routing" is an important subtask of VLSI circuit design. 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. 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 10 shows example solutions for one small and one large channel routing problem.

We use the clever local search operators defined by Wong et al. for this problem (Wong et al., 1988), but replace their contrived objective function C (see Equation 1 above) with the natural objective function $\mathrm{Obj}(x)$ = the channel width w. Wong's additional objective function terms, p and U, along with w itself, were given as the three input features to Stage's function approximator.

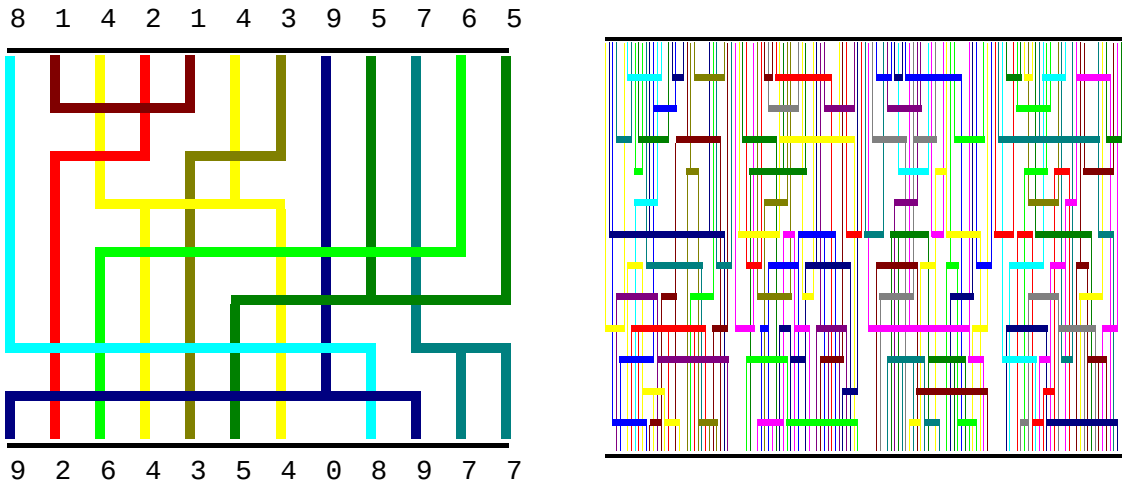

Figure 10: Left: a small channel routing instance. Right: a 12-track solution to instance YK4.

Results on YK4, an instance with 140 vertical tracks, are given in Table 2. All methods were allowed to consider 500,000 moves per run. Experiment (A) shows that multi-restart hillclimbing finds quite poor solutions. Experiment (B) shows that simulated annealing, as used with the objective function of (Wong et al., 1988), does considerably better. Experiment (C), simulated annealing with the raw objective function $\text{Obj}(x) = w$, does better still, which is surprising because Wong et al. invented the extra features of (B) to make their simulated-annealing system perform better. We believe (C) works better because it 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 fortuitously find a hole in the ridge. In fact, increasing hillclimbing's patience to ∞ (disabling restarts) worked nearly as well (D).

Stage used simple linear and quadratic regression models for learning. The results (E,F) show that Stage learned to optimize superbly, not only improving on the performance of hillclimbing as it was trained to do, but also finding better solutions on average 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 two 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 (G) was much worse than Stage's.

2. *The function approximator may simply be smoothing Obj(x), which helps eliminate local minima and plateaus?* No: we tried a variant of Stage which learned to smooth $\text{Obj}(x)$ directly instead of learning $\tilde{V}^\pi$ (H); this also produced much less improvement than Stage.

A closer look at the learned coefficients of $\tilde{V}^{\pi}$ reveals that Stage does in fact build a counterintuitive, yet successfully predictive, secondary evaluation function for search. We will return to this analysis in Section 4.1.

3.3 Bayes Network Structure-Finding

Given a data set, an important data mining task is to identify the Bayes net structure that best matches the data. We search the space of acyclic graph structures on A nodes, where A is the number of attributes in each data record. Following (Friedman & Yakhini, 1996), we evaluate a network structure by a *minimum description length score* which trades off between fit accuracy and low model complexity.

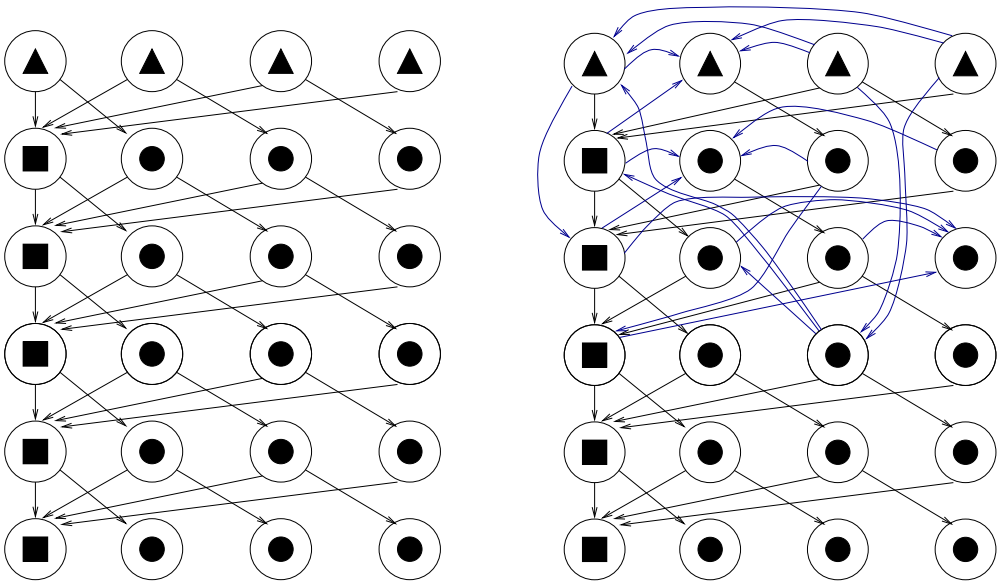

Figure 11: Left: a Bayes net on 24 binary attributes (from (Moore & Lee, 1998)) that was used to generate the SYNTH125K dataset. Its Obj score is 718641. Right: a network structure learned by a good run of Stage on the dataset. Its Obj score is 719074. Only two edges from the generator net are missing from the learned net. The learned net includes 17 edges not in the generator net (shown as curved arcs).

Stage was given the following seven extra features:

- the mean and standard deviation of the conditional entropy score at each node
- the mean and standard deviation of the number of parameters in each node's conditional probability table
- the mean and standard deviation of the number of parents of each node
- the number of nodes with no parents.

Problem Instance	Algorithm	Performance over N runs mean	best	worst
Bin-packing	Hillclimbing, patience=250	109.38± 0.10	108	110
(u250_13, opt=103)	Simulated annealing	108.19± 0.09	107	109
$M = 10^5$	Best-Fit Randomized	106.78± 0.08	106	107
	STAGE, quadratic regression	**104.77**± 0.09	**103**	**105**
Channel routing	(A) Hillclimbing, patience=250	22.35± 0.19	20	24
(YK4, opt=10)	(B) Simulated annealing, $\text{Obj} = w^2 + 0.5p^2 + 10U$	16.49± 0.16	14	19
$M = 5{\cdot}10^5$	(C) Simulated annealing, $\text{Obj} = w$	14.32± 0.10	13	15
	(D) Hillclimbing, patience=∞	14.69± 0.12	13	16
	(E) STAGE, linear regression	**12.42**± 0.11	**11**	**14**
	(F) STAGE, quadratic regression	14.01± 0.77	**11**	31
	(G) Hillclimbing + random walk	17.26± 0.14	15	19
	(H) Modified STAGE—only smooth Obj	16.88± 0.22	14	19
Bayes net	Hillclimbing, patience=200	**3563.4**± 0.3	**3561.3**	**3567.4**
(MPG)	Simulated annealing	3568.2± 0.9	**3561.3**	3595.5
$M = 10^5$	STAGE, quadratic regression	**3564.1**± 0.4	**3561.3**	3569.5
Bayes net	Hillclimbing, patience=200	440567± 52	439912	441171
(ADULT2)	Simulated annealing	440924± 134	**439551**	444094
$M = 10^5$	STAGE, quadratic regression	**440432**± 57	439773	**441052**
Bayes net	Hillclimbing, patience=200	748201± 1714	725364	766325
(SYNTH125K)	Simulated annealing	**726882**± 1405	718904	**754002**
$M = 10^5$	STAGE, quadratic regression	730399± 1852	**718804**	782531
Radiotherapy	Hillclimbing, patience=200	18.822±0.030	**18.003**	19.294
(5E)	Simulated annealing	18.817±0.043	18.376	19.395
$M = 10^4, N = 200$	STAGE, quadratic regression	**18.721**±0.029	18.294	**19.155**
Cartogram	Hillclimbing, patience=200	0.174±0.002	0.152	0.195
(US49)	Simulated annealing	**0.037**±0.003	**0.031**	0.170
$M = 10^6$	STAGE, quadratic regression	0.056±0.003	0.038	**0.132**
Satisfiability	(J) WALKSAT, noise=0, cutoff=10^6, tries=100	15.22± 0.35	9	19
(par32-1.cnf, opt=0)	(K) WALKSAT + $\delta_w = 0$ (hillclimbing)	690.52± 1.96	661	708
$M = 10^8$	(L) WALKSAT + $\delta_w = 10$	15.56± 0.33	11	19
	(M) STAGE(WALKSAT), quadratic regr.	5.36± 0.33	**1**	9
	(N) STAGE(WALKSAT/Markov), linear regr.	**4.43**± 0.28	2	**8**
Boggle setup	Hillclimbing, patience=1000	-8.413±0.066	-9.046	-7.473
5×5	Simulated annealing	-8.431±0.086	-9.272	**-7.622**
$M = 10^5$	STAGE, quadratic regression	-8.480±0.077	**-9.355**	-7.570

Table 2: Comparative results on a variety of minimization domains. Each line reports the mean, 95% confidence interval of the mean, best, and worst solutions found by 100 independent runs of one algorithm on one problem. All algorithms were limited to considering M total search moves. Best results are boldfaced.

We applied Stage to three datasets: MPG, a small dataset consisting of 392 records of 10 attributes each; ADULT2, a large real-world dataset consisting of 30,162 records of 15 attributes each; and SYNTH125K, a synthetic dataset consisting of 125,000 records of 24 attributes each. The synthetic dataset was generated by sampling from the Bayes net depicted in Figure 3.3. A perfect reconstruction of that net would receive a score of $\mathrm{Obj}(x) = 718641$.

Results are shown in Table 2. On SYNTH125K, the largest dataset, simulated annealing and Stage both improve significantly over multi-restart hillclimbing, usually attaining a score within 2% of that of the Bayes net that generated the data, and on some runs coming within 0.04%. A good solution found by Stage is drawn in Figure 3.3. Simulated annealing slightly outperforms Stage on average on this dataset. On the MPG and ADULT2 datasets, hillclimbing and Stage performed comparably, while simulated annealing did slightly less well on average. In sum, Stage's performance on the Bayes net learning task was less dominant than on the bin-packing and channel routing tasks, but it was still more consistently best or nearly best than either hillclimbing or simulated annealing on the three benchmark instances attempted.

3.4 Radiotherapy Treatment Planning

Radiation therapy is a method of treating tumors. A linear accelerator that produces a radioactive beam is mounted on a rotating gantry, and the patient is placed so that the tumor is at the center of the beam's rotation. Depending on the exact equipment being used, the beam's intensity can be modulated in various ways as it rotates around the patient. A *radiotherapy treatment plan* specifies the beam's intensity at a fixed number of source angles.

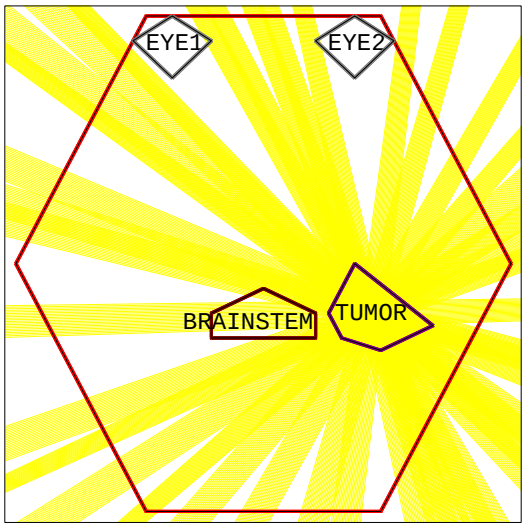

Figure 12: A simplified radiotherapy instance

A map of the relevant part of the patient's body, with the tumor and all important structures labelled, is available. Also known are good clinical forward models for calculating, from a treatment plan, the distribution of radiation that will be delivered to the patient's

tissues. The optimization problem, then, is to produce a treatment plan that meets target radiation doses for the tumor while minimizing damage to sensitive nearby structures. The current practice is to use simulated annealing and/or linear programming for this problem (Webb, 1994).

Figure 12 illustrates a simplified planar instance of the radiotherapy problem. The instance consists of an irregularly shaped tumor and four sensitive structures: the eyes, the brainstem, and the rest of the head. Given a treatment plan, the objective function is calculated by summing ten terms: an overdose penalty and an underdose penalty for each of the five structures. These ten subcomponents were the features for STAGE's learning.

Objective function evaluations are computationally expensive in this domain, so our experiments considered only 10,000 moves per run. Again, all algorithms performed comparably, but STAGE's solutions were best on average.

3.5 Cartogram Design

A "cartogram" or "Density Equalizing Map Projection" is a map whose boundaries have been deformed so that population density is uniform over the entire map (Gusein-Zade & Tikunov, 1993; Dorling, 1994). We considered redrawing the map of the United States such that each state's area is proportional to its electoral vote for U.S. President. The goal is to best meet the new area targets while minimally distorting the states' shapes and borders.

We represented the map as a collection of 162 points in $\Re^2$; each state is a polygon over a subset of those points (see Figure 13). Search begins at the original, undistorted U.S. map. The search operator consisted of perturbing a random point slightly; perturbations that would cause two edges to cross were disallowed. The objective function was defined as

$$\text{Obj}(x) = \Delta\text{area} + \Delta\text{gape} + \Delta\text{orient} + \Delta\text{segfrac}$$

where Δarea penalizes states for missing their new area targets, and the other three terms penalize states shaped differently than in the true U.S. map. For STAGE, we represented each configuration by the four subcomponents of Obj. Learning a new evaluation function with quadratic regression over these features, STAGE produced a significant improvement over hillclimbing, but was outperformed by simulated annealing.

3.6 Satisfiability

Finding a variable assignment that satisfies a large Boolean expression is a fundamental (indeed, the original) NP-complete problem. In recent years, surprisingly difficult formulas have been solved by WALKSAT (Selman, Kautz, & Cohen, 1996), a simple local search method. WALKSAT, given a formula expressed in CNF (a conjunction of disjunctive clauses), conducts a random walk in assignment space which is biased toward minimizing

$$\text{Obj}(x) = \#\text{ of clauses unsatisfied by assignment } x.$$

When $\text{Obj}(x) = 0$, all clauses are satisfied and the formula is solved.

WALKSAT searches as follows. On each step, it first selects an unsatisfied clause at random; it will satisfy that clause by flipping one variable within it. To decide which one, it first evaluates how much overall improvement to Obj would result from flipping each

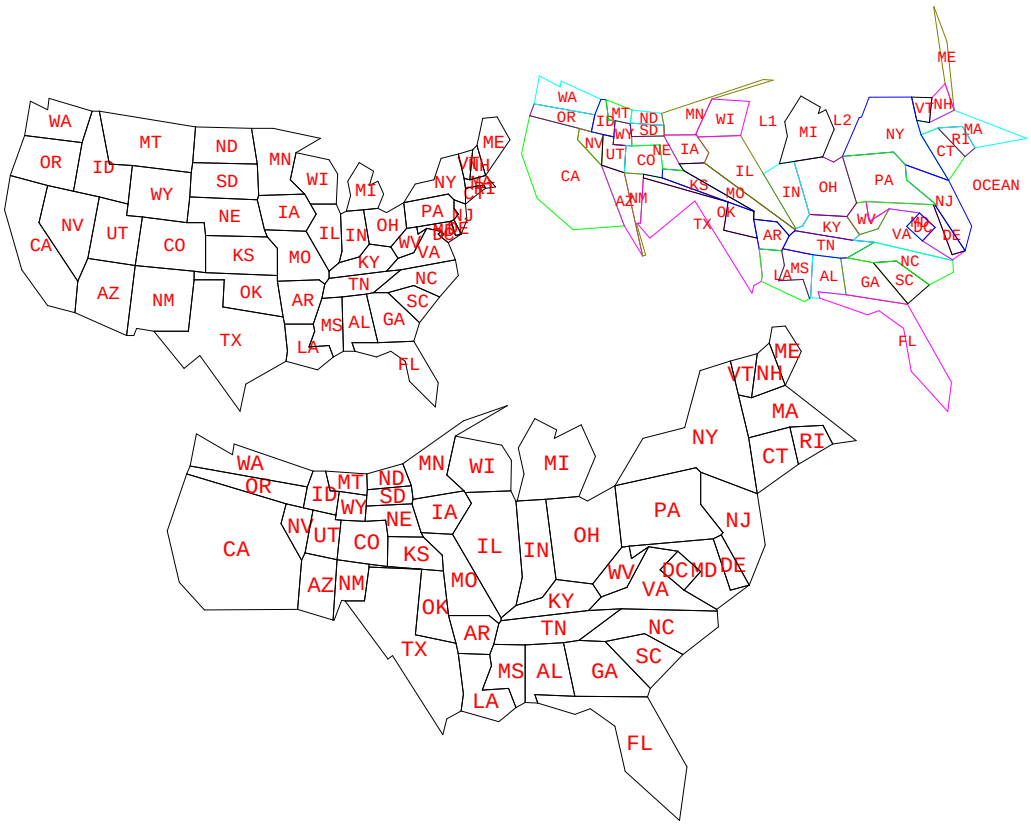

Figure 13: Cartograms of the continental U.S. Each state's target area is proportional to its electoral vote for U.S. President. The undistorted U.S. map has zero penalty for state shapes and orientations but a large penalty for state areas, so $\mathrm{Obj}(x) = 525.7$. Hillclimbing produces solutions like the one shown at top right, for which $\mathrm{Obj}(x) = 0.115$. The third cartogram, found by Stage, has $\mathrm{Obj}(x) = 0.043$.

variable. If the best such improvement is positive, it greedily flips a variable that attains that improvement. Otherwise, it flips a variable which worsens Obj: with probability (1-*noise*), a variable which harms Obj the least, and with probability *noise*, a variable at random from the clause. The best setting of *noise* is problem-dependent (McAllester, Kautz, & Selman, 1997).

Walksat is so effective that it has rendered nearly obsolete an archive of several hundred benchmark problems collected for a DIMACS Challenge on satisfiability (Selman et al., 1996). Within that archive, only the largest "parity function learning" instances (nefariously constructed by Kearns, Schapire, Hirsh and Crawford) are known to be solvable in principle, yet not solvable by Walksat. We report here results of experiments on the instance par32-1.cnf, a formula consisting of 10277 clauses on 3176 variables. Each experiment was run 100 times and allowed to consider 10^8 bit flips per run.

Experiment J (see Table 2) shows results with the best hand-tuned parameter settings for Walksat. The best such run still left 9 clauses unsatisfied. We introduced an additional

WALKSAT parameter δ_w, with the following effect: any flip that would worsen Obj by more than δ_w is rejected. Normal WALKSAT has $\delta_w = \infty$. At the other extreme, when $\delta_w = 0$, no harmful moves are accepted, resulting in an ineffective form of hillclimbing (K). However, using intermediate settings of δ_w—thereby prohibiting only the most destructive of WALKSAT's moves—seems not to harm performance (L), and in some cases improves it.

For STAGE's learning, a variety of potentially useful additional state features are available, e.g.:

- % of clauses currently unsatisfied (= Obj(x))
- % of clauses satisfied by exactly 1 variable
- % of clauses satisfied by exactly 2 variables
- % of variables that would break a clause if flipped
- % of variables set to their "naive" setting, defined to be 1 for variable x_i if x_i appears in more clauses of the formula than $\neg x_i$ does, or 0 otherwise.

Can STAGE, by observing WALKSAT trajectories, learn to combine these features usefully, as it did by observing hillclimbing trajectories in other domains?

Theoretically, STAGE can learn from any procedure π that is proper (guaranteed to terminate) and Markovian. WALKSAT's normal termination mechanism, cutting off after a pre-specified number of steps, is not Markovian: it depends on an extraneous counter variable, not just the current assignment. Despite this technicality, STAGE with quadratic regression (M) very nearly completely solved the problem, satisfying all but 1 or 2 of the 10277 clauses on several runs. With a properly Markovian cutoff criterion for WALKSAT (terminating with probability 1/10000 after each step) and linear instead of quadratic regression (N), STAGE's improvement over plain WALKSAT was about the same. Results on four other 32-bit parity benchmark instances were similar. In these experiments, WALKSAT was run with *noise*=25 and δ_w=10; full details may be found in (Boyan, 1998).

We believe that STAGE shows promise for hard satisfiability problems—perhaps for MAXSAT problems where near-miss solutions are useful.

3.7 Boggle Board Setup

In the game of Boggle, 25 cubes with letters printed on each face are shaken into a 5×5 grid (see Figure 14).[1] The object of the game is to find English words that are spelled out by connected paths through the grid. A legal path may include horizontal, vertical, and/or diagonal steps; it may not include any cube more than once. Long words are more valuable than short ones: the scoring system counts 1 point for 4-letter words, 2 points for 5-letter words, 3 points for 6-letter words, 5 points for 7-letter words, and 11 points for words of length 8 or greater.

Given a fixed board setup x, finding *all* the English words in it is a simple computational task; by representing the dictionary[2] as a prefix tree, Score(x) can be computed in about a millisecond. It is a difficult optimization task, however, to identify what fixed board x^* has the highest score. For consistency with the other domains presented, we pose the problem

1. Boggle is published by Parker Brothers, Inc. The 25-cube version is known as "Big Boggle" or "Boggle Master."
2. Our experiments make use of the 126,468-word Official Scrabble Player's Dictionary.

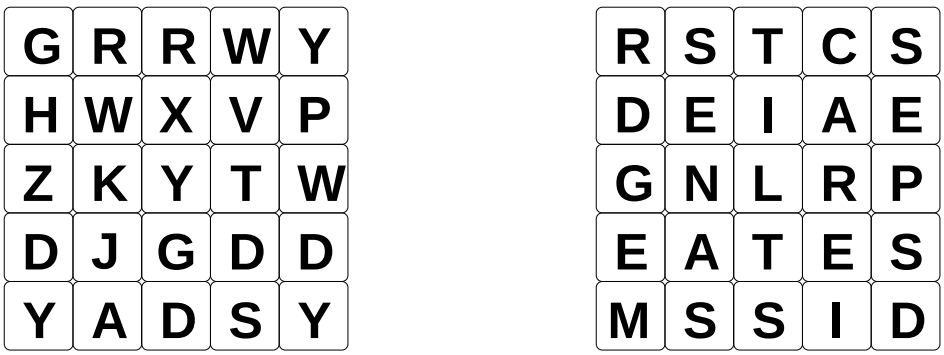

Figure 14: A random Boggle board (8 words, score=10, Obj=−0.010) and an optimized Boggle board (2034 words, score=9245, Obj=−9.245). The latter includes such high-scoring words as *depreciated*, *distracting*, *specialties*, *delicateness* and *desperateness*.

as a minimization task, where $\text{Obj}(x) = -\text{Score}(x)/1000$. Exhaustive search of 26^{25} Boggle boards is intractable, so local search is a natural approach.[3]

We set up the search space as follows. The initial state is constructed by choosing 25 letters uniformly at random. Then, to generate a neighboring state, either of the following operators is applied with probability 0.5:

- Select a grid square at random and choose a new letter for it. (The new letter is selected with probability equal to its unigram frequency in the dictionary.)
- Or, select a grid square at random, and swap the letter at that position with the letter at a random adjacent position.

The following features of each state x were provided for Stage's learning:

1. The objective function, $\text{Obj}(x) = -\text{Score}(x)/1000$.
2. The number of vowels on board x.
3. The number of distinct letters on board x.
4. The sum of the unigram frequencies of the letters of x. (These frequencies, computed directly from the dictionary, range from $\text{Freq}(e) = 0.1034$ to $\text{Freq}(q) = 0.0016$.)
5. The sum of the bigram frequencies of all adjacent pairs of x.

These features are cheap to compute incrementally after each move in state space, and intuitively should be helpful for Stage in learning to distinguish promising from unpromising boards.

3. We allow any letter to appear in any position, rather than constraining them to the faces of real 6-sided Boggle cubes.

However, Stage's results on Boggle were disappointing: average runs of hillclimbing, simulated annealing, and Stage all reach the same Boggle score of about 8400–8500 points. Boggle is the only nontrivial domain we have tried on which Stage's learned smart restarting does not improve significantly over random-restart hillclimbing. What explains this failure?

To answer this question, we return to Stage's foundations. Stage is designed to map out the attracting basins of a domain's local minima. Our hypothesis is that when there is a coherent trend among these attracting basins, Stage can exploit it. Identifying such a coherent trend depends crucially on the user-selected state features, the domain's move operators, and the regression models considered. What our positive results have shown is that for a wide variety of problems, with very simple choices of features and models, a useful structure can be identified and exploited.

However, the Boggle results illustrate the converse of our hypothesis: when the results of hillclimbing from a variety of starting states show no discernible trend, then Stage will fail. The following experiment with Boggle makes this clear. We ran 50 restarts of hillclimbing for each of six different restarting policies:

random: Reassign all 25 tiles in the grid randomly on each restart.

EEE: Start with each tile in the grid to the letter 'E'.

SSS: Start with each tile in the grid to the letter 'S'.

ZZZ: Start with each tile in the grid to the letter 'Z'.

ABC: Start with the grid set to ABCDE/FGHIJ/KLMNO/PQRST/UVWXY.

cvcvc: Assign the first, third, and fifth rows of the grid to random consonants, and the second and fourth rows of the grid to random vowels. High-scoring grids often have a pattern similar to this (or rotations of this).

The boxplot in Figure 15 compares the performance of hillclimbing from these sets of states. Apparently, the restarting policy is irrelevant to hillclimbing's mean performance: on average, each trajectory leads to a Boggle score near 7000 no matter which of the above types of starting states is chosen. Thus, Stage cannot learn useful predictions of V^π, and its failure to outperform multi-restart hillclimbing on Boggle is consistent with our understanding of how Stage works.

Similar reasoning also explains why Stage cannot bootstrap the performance of simulated annealing—i.e., use π = simulated annealing instead of π = hillclimbing in Stage's inner loop. Simulated annealing's initial period of random search (high "temperatures") makes the outcome quite unpredictable from the starting state.

3.8 Running Times

Our experiments allowed each competing algorithm to compute the same number of state evaluations per trial. However, relative to hillclimbing and simulated annealing, Stage carries the additional computational burden of training and evaluating a function approximator on $\tilde{V}^\pi$. Did this amount to an unfair advantage for Stage in the results?

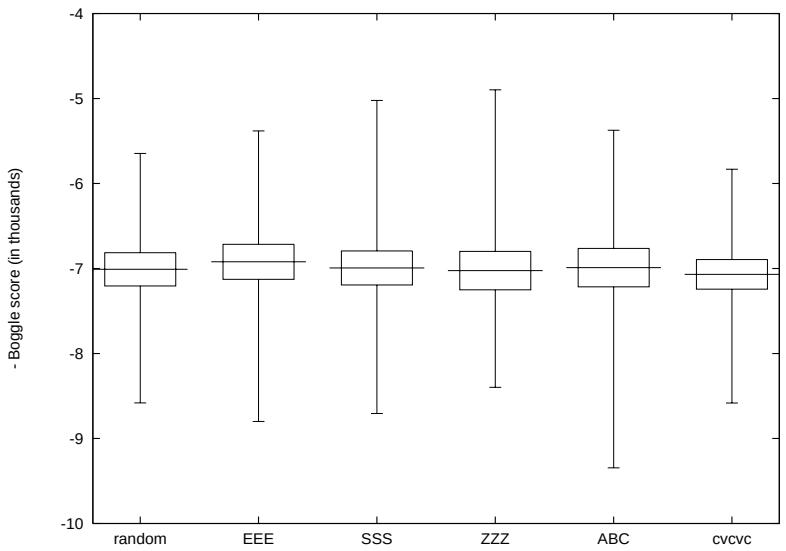

Figure 15: Boggle: average performance of 50 restarts of hillclimbing from six different sets of starting states

The answer is definitely no. We found that STAGE's running time was usually within 10% of hillclimbing's, and never more than double it. Discrepancies, when they did occur, were caused not by time spent in function approximation, but rather by such factors as (1) focusing search in a higher-quality part of the space where legal moves were more expensive to generate; or (2) accepting an overall much smaller percentage of moves, thereby incurring a penalty for undoing many moves. Moreover, in those same cases, our results would be qualitatively unchanged even if STAGE's runs had been cut off early to account for the discrepancies. For details of these and other timing results, please refer to (Boyan, 1998).

Asymptotically, the computational cost of STAGE's function approximation is $O(D^2L + D^3)$ for linear regression or $O(D^4L + D^6)$ for quadratic regression, where L is the length of a training trajectory and D is the number of features used. By using very small numbers of features, and by choosing features that were cheap to generate, we kept the overhead of learning in our experiments very low. However, STAGE's extra overhead for function approximation would become significant if many more features or more sophisticated function approximators were used. Furthermore, even if the function approximation is inexpensive, STAGE may require many trajectories to be sampled in order to obtain sufficient data to fit V^π effectively.

For some problems such costs are worth it in comparison with 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, learning may nevertheless be useful 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 already

have a $\tilde{V}^{\pi}$ estimate. This effect is called *transfer*, and we investigate it in the following section.

4. Transfer

4.1 Comparing $\tilde{V}^{\pi}$ Across Instances

To investigate the potential for transferring learned evaluation functions among similar problem instances, we first re-ran Stage on a suite of eight problems from the channel routing literature (Chao & Harper, 1996). Table 3 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 positive linear transformations.

Problem instance	lower bound	best-of-3 hillclimbing	best-of-3 Stage	learned coefficients $< \beta_w, \beta_p, \beta_U >$
YK4	10	22	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	12	12	$< 0.72, 0.30, -0.62 >$
HYC4	20	27	23	$< 0.71, 0.03, -0.71 >$
HYC5	35	39	38	$< 0.69, 0.14, -0.71 >$
HYC6	50	56	51	$< 0.70, 0.05, -0.71 >$
HYC7	39	54	42	$< 0.71, 0.13, -0.69 >$
HYC8	21	29	25	$< 0.71, 0.03, -0.70 >$

Table 3: Stage results on eight problems from (Chao & Harper, 1996). 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 (Wong et al., 1988) (Equation 1), all but one of the Stage-learned functions assign a relatively large positive weight to feature w and a small positive weight to feature p. Unlike the hand-tuned cost function, 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.

As an aside, we note that 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, sparse 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 lead search toward solutions with an *uneven* distribution of track sparsity levels. 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.

4.2 X-Stage: A Voting Algorithm for Transfer

Many sensible methods for transferring the knowledge learned by Stage from training instances to new instances can be imagined. This section presents one such method. Stage's learned knowledge, of course, is represented by the approximated value function $\tilde{V}^\pi$. We would like to take the $\tilde{V}^\pi$ information learned on a set of training instances $\{I_1, I_2, \ldots, I_N\}$ and use it to guide search on a given new instance I'. But how can we ensure that $\tilde{V}^\pi$ is meaningful across multiple problem instances simultaneously, when the various instances may differ markedly in size, shape, and attainable objective-function value?

The first step is to impose an instance-independent representation on the features $F(x)$, which comprise the input to $\tilde{V}^\pi(F(x))$. As it so happens, all the feature sets described above are naturally instance-independent, or can easily be made so by normalization. For example, in Bayes-net structure-finding problems (Section 3.3), the feature that counts the number of parentless nodes can be made instance-independent simply by changing it to the *percentage* of total nodes that are parentless.

The second step concerns normalization of the *outputs* of $\tilde{V}^\pi(F(x))$, which are predictions of objective-function values. In Table 3 above, the nine channel routing instances all have quite different solution qualities, ranging from 8 tracks in the case of instance HYC1 to more than 50 tracks in the case of instance HYC6. If we wish to train a single function approximator to make meaningful predictions about the expected solution quality on both instances HYC1 and HYC6, then we must normalize the objective function itself. For example, $\tilde{V}^\pi$ could be trained to predict not the expected reachable Obj value, but the expected reachable percentage above a known lower bound for each instance. In their algorithm for transfer among job-shop scheduling instances (described below), Zhang and Dietterich adopt this approach: they heuristically normalize each instance's final job-shop schedule length by dividing it by the difficulty level of the starting state (Zhang & Dietterich, 1998). This enables them to train a single neural network over all problem instances.

However, if the available lower bounds are not tight, such normalization can be problematic. We adopt here a different approach that eliminates the need to normalize the objective function across instances. The essential idea is to recognize that each individually learned $\tilde{V}^\pi_{I_k}$ function, *unnormalized*, is already suitable for guiding search on the new problem I': the search behavior of an evaluation function is scale and translation-invariant. The X-Stage algorithm, specified in Table 4, combines the knowledge of multiple $\tilde{V}^\pi_{I_k}$ functions not by merging them into a single new evaluation function, but by having them *vote* on move decisions for the new problem I'. Note that after the initial set of value functions has been trained, X-Stage performs no further learning when given a new optimization problem I' to solve.

Combining $\tilde{V}^\pi_{I_k}$ decisions by voting rather than, say, averaging, ensures that each training instance carries equal weight in the decision-making process, regardless of the range of that instance's objective function. Voting is also robust to "outlier" functions, such as the one learned on instance HYC1 in Table 3 above. Such a function's move recommendations will simply be outvoted. A drawback to the voting scheme is that, in theory, loops are possible in which a majority prefers x over x', x' over x'', and x'' over x. However, we have not seen such a loop in practice, and if one did occur, the patience counter Patience would at least prevent X-Stage from getting permanently stuck.

X-Stage($I_1, I_2, \ldots, I_N, I'$):

Given:

- a set of training problem instances $\{I_1, \ldots, I_N\}$ and a test instance I'.

  Each instance has its own objective function and all other Stage parameters (see Table 1). It is assumed that each instance's featurizer $F : X \rightarrow \Re^D$ maps states to the same number D of real-valued features.

1. **Run** Stage **independently on each of the** N **training instances.**

   This produces a set of learned value functions $\{\tilde{V}^\pi_{I_1}, \tilde{V}^\pi_{I_2}, \ldots, \tilde{V}^\pi_{I_N}\}$.

2. **Run** Stage **on the new instance** I'**,** but modifying Stage's Step 2c—the step that searches for a promising new starting state for π—as follows: instead of performing hillclimbing on a newly learned $\tilde{V}^\pi$, perform *voting-hillclimbing* on the set of previously learned $\tilde{V}^\pi$ functions. Voting-hillclimbing means simply:

   > Accept a proposed move from state x to state x' if and only if, for a majority of the learned value functions, $\tilde{V}^\pi_{I_k}(F(x')) \leq \tilde{V}^\pi_{I_k}(F(x))$.

   Return the best state found.

Table 4: The X-Stage algorithm for transferring learned knowledge to a new optimization instance

4.3 Experiments

We applied X-Stage to the domains of bin-packing and channel routing. For the bin-packing experiment, we gathered a set of 20 instances (the u250 suite) from the OR-Library. Using the same Stage parameters given in that section, we trained $\tilde{V}^\pi$ functions for all of the 20 except u250_13, and then applied X-Stage to test performance on the held-out instance. The performance curves of X-Stage and, for comparison, ordinary Stage are shown in Figure 16 (left). The semilog scale of the plot clearly shows that X-Stage reaches good performance levels more quickly than Stage. However, after only about 10 learning iterations and 10,000 evaluations, the average performance of Stage exceeds that of X-Stage. Stage's $\tilde{V}^\pi$ function, finely tuned for the particular instance under consideration, ultimately outperforms the voting-based restart policy generated from 19 related instances.

The channel routing experiment was conducted with the set of 9 instances shown in Table 3 above. Again, all Stage parameters were set as in the experiments of Section 3. We trained $\tilde{V}^\pi$ functions for the instances HYC1...HYC8, and applied X-Stage to test performance on instance YK4. The performance curves of X-Stage and ordinary Stage are shown in Figure 16. Again, X-Stage reaches good performance levels more quickly than does Stage. This time, the voting-based restart policy maintains its superiority over the instance-specific learned policy for the duration of the run.

These preliminary experiments indicate that the knowledge Stage learns during problem-solving can indeed be transferred profitably to novel problem instances. Future work will

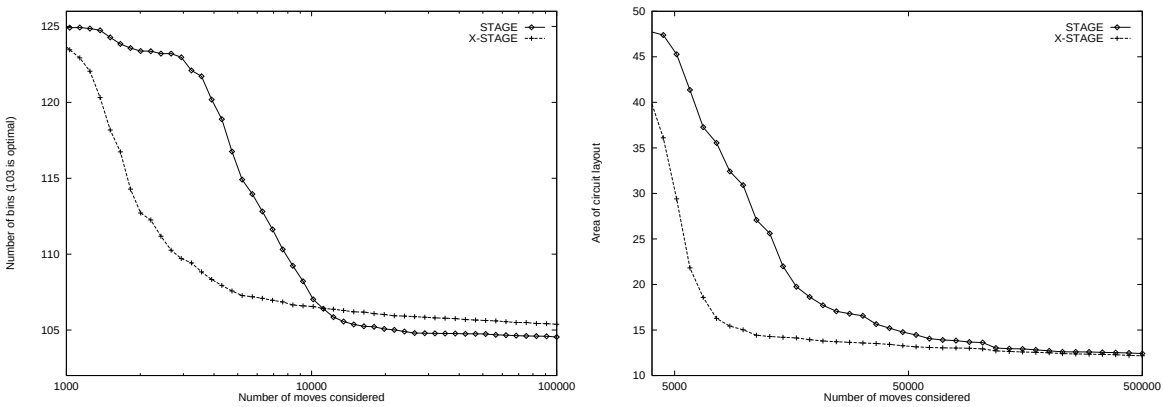

Figure 16: Left: Bin-packing performance on instance u250_13 with transfer (X-Stage) and without transfer (Stage). Right: Channel routing performance on instance YK4 with and without transfer. Note the logarithmic scale of the x-axes.

consider ways of combining previously learned knowledge with new knowledge learned during a run, so as to have the best of both worlds: exploiting general knowledge about a family of instances to reach good solutions quickly, and exploiting instance-specific knowledge to reach the best possible solutions.

5. Related Work

Stage draws on work from the communities of adaptive local search, reinforcement learning, and genetic algorithms. This section reviews the most relevant results from each of these communities in turn.

5.1 Adaptive Multi-Restart Techniques

An iteration of hillclimbing typically reaches a local optimum very quickly. Thus, in the time required to perform a single iteration of (say) simulated annealing, one can run many hillclimbing iterations from different random starting points (or even from the same starting point, if move operators are sampled stochastically) and report the best result. Empirically, random multi-start hillclimbing has produced excellent solutions on practical computer vision tasks (Beveridge, Graves, & Lesher, 1996), outperformed simulated annealing on the Traveling Salesman Problem (TSP) (Johnson & McGeoch, 1995), and outperformed genetic algorithms and genetic programming on several large-scale testbeds (Juels & Wattenberg, 1996).

Nevertheless, the effectiveness of random multi-start local search is limited in many cases by a "central limit catastrophe" (Boese, Kahng, & Muddu, 1994): random local optima in large problems tend to all have average quality, with little variance (Martin & Otto, 1994). This means the chance of finding an improved solution diminishes quickly from one iteration to the next. To improve on these chances, an *adaptive multi-start* approach—designed to select restart states with better-than-average odds of finding an improved solution—seems

appropriate. Indeed, in the theoretical model of local search proposed by Aldous and Vazirani (Aldous & Vazirani, 1994), a given performance level that takes $O(n)$ restarts to reach by a random starting policy can instead be reached with as few as $O(\log n)$ restarts when an adaptive policy, which uses successful early runs to seed later starting states, is used.

Many adaptive multi-start techniques have been proposed. One particularly relevant study has been conducted by Boese (Boese, 1995). On a fixed, well-known instance of the TSP, he ran local search 2500 times to produce 2500 locally optimal solutions. Then, for each of those solutions, he computed the average distance to the other 2499 solutions, measured by a natural distance metric on TSP tours. The results showed a stunning correlation between solution quality and average distance: high-quality local optima tended to have small average distance to the other optima—they were "centrally" located—while worse local optima tended to have greater average distance to the others; they were at the "outskirts" of the space. Similar correlations were found in a variety of other optimization domains, including circuit/graph partitioning, satisfiability, number partitioning, and job-shop scheduling. Boese concluded that many practical optimization problems exhibit a "globally convex" or so-called "big valley" structure, in which the set of local optima appears convex with one central global optimum. Boese's intuitive diagram of the big valley structure is reproduced in Figure 17.

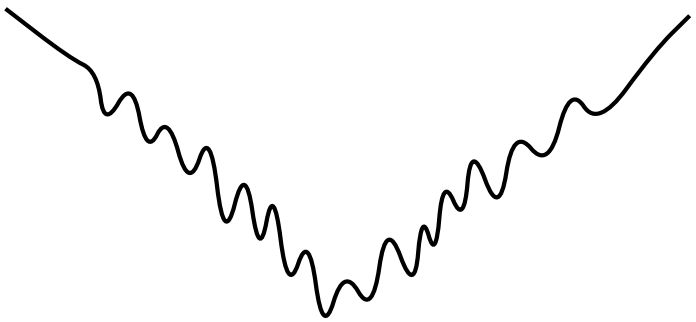

Figure 17: Intuitive picture of the "big valley" solution space structure. (Adapted from (Boese, 1995).)

The big valley structure is auspicious for a Stage-like approach. Indeed, Boese's intuitive diagram, motivated by his experiments on large-scale complex problems, bears a striking resemblance to the 1-D wave function of Figure 1, which we contrived as an example of the kind of problem at which Stage would excel. Boese went on to propose a two-phase adaptive multi-start methodology for optimization similar to Stage. The main difference is that Boese hand-builds a problem-specific routine for adaptively constructing new starting states, whereas Stage uses machine learning to do the same automatically.

A similar methodology underlies the current best heuristic for solving large Traveling Salesman problems, "Chained Local Optimization" (CLO) (Martin & Otto, 1994). CLO performs ordinary hillclimbing to reach a local optimum z, and then applies a special large-step stochastic operator designed to "kick" the search from z into a nearby but different

attracting basin. Hillclimbing from this new starting point produces a new local optimum z'; if this turns out to be much poorer than z, then CLO returns to z, undoing the kick. In effect, CLO constructs a new high-level search space: the new operators consist of large-step kick moves, and the new objective function is calculated by first applying hillclimbing in the low-level space, then evaluating the resulting local optimum. (A similar trick is often applied with genetic algorithms, as we discuss in Section 5.3 below.) In the TSP, the kick designed by Martin and Otto (Martin & Otto, 1994) is a so-called "double-bridge" operation, chosen because such moves cannot be easily found nor easily undone by Lin-Kernighan local search moves (Johnson & McGeoch, 1995). Like Boese's adaptive multi-start, CLO relies on manually designed kick steps for finding a good new starting state, as opposed to STAGE's learned restart policy. Furthermore, STAGE's "kicks" place search in not just a random nearby basin, but one specifically predicted to produce an improved local optimum.

The big valley diagram, like our 1-D wave function, conveys the notion of a global structure over the local optima. Unlike the wave function, it also conveys one potentially misleading intuition: that starting from low-cost solutions is necessarily better than starting from high-cost solutions. In his survey of local search techniques for the TSP, Johnson (Johnson & McGeoch, 1995) considered four different randomized heuristics for constructing starting tours from which to begin local search. He found significant differences in the quality of final solutions. Interestingly, the heuristic that constructed the best-quality starting tours (namely, the "Clarke-Wright" heuristic) was also the one that led search to the *worst*-quality final solutions—even worse than starting from a very poor, completely random tour. Such "deceptiveness" can cause trouble for simulated annealing and genetic algorithms. Large-step methods such as CLO may evade some such deceits by "stepping over" high-cost regions. STAGE confronts the deceit head-on: it explicitly detects when features other than the objective function are better predictors of final solution quality, and can learn to ignore the objective function altogether when searching for a good start state.

Many other sensible heuristics for adaptive restarting have been shown effective in the literature. The widely applied methodology of "tabu search" (Glover & Laguna, 1993) is fundamentally a set of adaptive heuristics for escaping local optima, like CLO's kick steps. Hagen and Kahng's "Clustered Adaptive Multi-Start" achieves excellent results on the VLSI netlist partitioning task (Hagen & Kahng, 1997); like CLO, it alternates between search with high-level operators (constructed *adaptively* by clustering elements of previous good solutions) and ordinary local search. Jagota's "Stochastic Steep Descent with Reinforcement Learning" heuristically rewards good starting states and punishes poor starting states in a multi-start hillclimbing context (Jagota, Sanchis, & Ganesan, 1996). The precise reward mechanism is heuristically determined and appears to be quite problem-specific, as opposed to STAGE's uniform mechanism of predicting search outcomes by value function approximation. As such, a direct empirical comparison would be difficult.

5.2 Reinforcement Learning for Optimization

As mentioned in Section 2.1, STAGE may be seen as approximating the value function of a policy in a Markov Decision Process that corresponds to the optimization problem. Value function approximation has previously been applied to a large-scale optimization task: the

Space Shuttle Payload Processing (SSPP) domain (Zhang & Dietterich, 1995, 1998). As this work is the closest in the literature to ours, we will discuss it here in some detail.

STAGE works by learning V^π, the predicted outcome of a prespecified optimization policy π. By contrast, the Zhang and Dietterich approach seeks to learn V^*, the predicted outcome of the *best possible* optimization policy. Before discussing how they approached this ambitious goal, we must address how the "best possible optimization policy" is defined, since optimization policies face two conflicting objectives: to produce good solutions and to finish quickly. In the SSPP domain, Zhang and Dietterich measured the total cost of a search trajectory $(x_0, x_1, \ldots, x_N, \text{END})$ by

$$\text{Obj}(x_N) + 0.001N$$

Effectively, since $\text{Obj}(x)$ is near 1.0 in this domain, this cost function means that a 1% improvement in final solution quality is worth about 10 extra search steps (Zhang & Dietterich, 1998). The goal of learning, then, was to produce a policy π^* to optimize this balance between trajectory length and final solution quality.

Zhang and Dietterich's goal was to obtain transfer from easy instances to hard instances of SSPP. Thus, they represented states by abstract instance-independent features, as we did later in X-STAGE. They also normalized their objective function so that it would span roughly the same range regardless of problem difficulty; X-STAGE's voting-based approach to transfer of X-STAGE allows this normalization to be avoided. Following Tesauro's methodology for reinforcement-learning of V^* on backgammon (Tesauro, 1992), they applied optimistic TD(λ) to the SSPP domain. The results showed that searches with the learned evaluation functions produced schedules as good as those found by the previously best SSPP optimizer in less than half the CPU time.

Getting these results required substantial tuning. One complication involves state-space cycles. Since the SSPP move operators are deterministic, a learned policy may easily enter an infinite loop, which makes its value function undefined. Loops are fairly infrequent because most operators repair constraint violations, lengthening the schedule; still, Zhang and Dietterich had to include a loop-detection and escape mechanism, clouding the interpretation of V^*. To attack other combinatorial optimization domains with their method, they suggest that "it is important to formulate problem spaces so that they are acyclic" (Zhang & Dietterich, 1998)—but such formulations are unnatural for most local search applications, in which the operators typically allow any solution to be reached from any other solution. STAGE finesses this issue by fixing π to be a proper policy such as hillclimbing.

STAGE also circumvents three other algorithmic complications that Zhang and Dietterich found it necessary to introduce: experience replay (Lin, 1993), random exploration (slowly decreasing over time), and random-sample greedy search. Experience replay, i.e., saving the best trajectories in memory and occasionally retraining on them, is unnecessary in STAGE because the regression matrices always maintain the sufficient statistics of *all* historical training data (Boyan, 2000). Adding random exploration is unnecessary because empirically, STAGE's baseline policy π (e.g., stochastic hillclimbing or WALKSAT) provides enough exploration inherently. This is in contrast to the SSPP formulation, where actions are deterministic. Finally, STAGE does not face the branching factor problem that led Zhang and Dietterich to introduce random-sample greedy search (RSGS). Briefly, the problem is that when hundreds or thousands of legal operators are available, selecting the greedy

action, as optimistic TD(λ) requires, is too costly. RSGS uses a heuristic to select an approximately greedy move from a subset of the available moves. Again, this clouds the interpretation of V^*. In STAGE, each decision is simply whether to accept or reject a single available move, and the interpretation of V^π is clear.

To summarize, STAGE avoids most of the algorithmic complexities of Zhang and Dietterich's method because it is solving a fundamentally simpler problem: estimating V^π from a fixed stochastic π, rather than discovering an optimal deterministic policy π^* and value function V^*. It also avoids many issues of normalizing problem instances and designing training architectures by virtue of the fact that it applies in the context of a single problem instance. However, an advantage of the Zhang and Dietterich approach is its potential to identify a truly optimal or near-optimal policy π^*. STAGE can only learn an improvement over the prespecified policy π.

5.3 Genetic Algorithms

Genetic algorithms (GAs), based on metaphors of biological evolution such as natural selection, mutation, and recombination, represent another heuristic approach to combinatorial optimization (Goldberg, 1989). Translated into the terminology of local search, "natural selection" means rejecting high-cost states in favor of low-cost states, as hillclimbing does; "mutation" means a small-step local search operation; and "recombination" means adaptively creating a new state from previously good solutions. GAs have much in common with the adaptive multi-start hillclimbing approaches discussed above in Section 5.1. In broad terms, the GA population carries out multiple restarts of hillclimbing in parallel, culling poor-performing runs and replacing them with new adaptively constructed starting states.

To apply GAs to an optimization problem, the configuration space X must be represented as a space of discrete feature vectors—typically fixed-length bitstrings $\{0,1\}^L$—and the mapping must be a bijection, so that a solution bitstring in the feature space can be converted back to a configuration in X. This contrasts with STAGE, where features can be any real-valued functions of the state, and the mapping need not be invertible. Typically, a GA mutation operator consists of flipping a single bit, and a recombination operator consists of merging the bits of two "parent" bitstrings into the new "child" bitstring. The effectiveness of GA search depends critically on the suitability of these operators to the particular bitstring representation chosen for the problem. Hillclimbing and simulated annealing, by contrast, allow much more sophisticated, domain-specific search operators, such as the partition-graph manipulations we used for channel routing (Wong et al., 1988). On the other hand, genetic algorithms have a built-in mechanism for combining features of previously discovered good solutions into new starting states. STAGE can be seen as providing the best of both worlds: sophisticated search operators *and* adaptive restarts based on arbitrary domain features.

Some GA implementations do manage to take advantage of local search operators more sophisticated than bit-flips, using the trick of embedding a hillclimbing search into each objective function evaluation (Hinton & Nowlan, 1987). That is, the GA's population of bitstrings actually serves as a population not of final solutions but of starting states for hillclimbing. The most successful GA approaches to the Traveling Salesman Problem all work this way so that they can exploit the sophisticated Lin-Kernighan local search moves

(Johnson & McGeoch, 1995). Here, the GA operators play a role analogous to the large-step "kick moves" of Chained Local Optimization (Martin & Otto, 1994), as described in Section 5.1 above. Depending on the particular implementation, the next generation's population may consist of not only the best starting states from the previous generation, but also the best final states found by hillclimbing runs—a kind of Lamarckian evolution in which learned traits are inheritable (Ackley & Littman, 1993; Johnson & McGeoch, 1995).

In such a GA, the population may be seen as implicitly maintaining a global predictive model of where, in bitstring-space, the best starting points are to be found. The COMIT algorithm of Baluja and Davies (Baluja & Davies, 1997) makes this viewpoint explicit: it generates adaptive starting points not by random genetic recombination, but rather by first building an explicit probabilistic model of the population and then sampling that model. COMIT's learned probability model is similar in spirit to Stage's V^π function. Differences include the following:

- COMIT is restricted to bijective bitstring-like representations, whereas Stage can use any feature mapping; and
- COMIT's model is trained from only the set of best-quality states found so far, ignoring the differences between their outcomes; whereas Stage's value function is trained from all states seen on all trajectories, good and bad, paying attention to the outcome values. Boese's experimental data and "big valley structure" hypothesis indicate that there is often useful information to be gained by modelling the weaker areas of the solution space, too (Boese et al., 1994). In particular, this gives Stage the power for directed extrapolation beyond the support of its training set.

In preliminary experiments in the Boolean satisfiability domain, on the same 32-bit parity instances described in Section 3.6, COMIT (using WALKSAT as a subroutine) did not perform as well as Stage (Davies & Baluja, 1998).

6. Conclusions and Future Work

Our primary conclusion is that learning evaluation functions can improve global optimization performance. Stage is a simple, practical technique that demonstrates this: on most tested instances, Stage robustly outperforms both multi-start hillclimbing and a good implementation of simulated annealing. Stage's simplicity enables many potentially useful extensions, such as the X-Stage algorithm for transfer presented above. Further extensions include the following:

Non-polynomial function approximators. Our results were limited to polynomial models of V^π. It would be interesting to see whether other linear architectures—such as CMACs, radial basis function networks, and random-representation neural networks—could produce better fits and better performance. A more ambitious study could investigate efficient ways to use *nonlinear architectures*, such as multi-layer perceptrons or memory-based methods, with Stage. In the context of transfer, the training speed of the function approximator is less crucial.

More aggressive optimization of $\tilde{\mathbf{V}}^{\boldsymbol{\pi}}$. On each iteration, in order to find a promising new starting state for the baseline procedure π, Stage optimizes $\tilde{V}^\pi$ by performing

first-improvement hillclimbing (Step 2c). A more aggressive optimization technique such as simulated annealing could instead be applied at that stage, and that may well improve performance.

Steepest descent. With the exception of the WALKSAT results of Section 3.6, STAGE has been trained to predict and improve upon the baseline procedure of π = first-improvement hillclimbing. However, in some optimization problems—particularly, those with relatively few moves available from each state—*steepest-descent* (best-improvement) search may be more effective. Steepest-descent is proper, Markovian, and monotonic, so STAGE applies directly; it would be interesting to compare its effectiveness with first-improvement hillclimbing's.

Continuous optimization. Our results have focused on discrete optimization problems. However, STAGE applies without modification to continuous global optimization problems (i.e., find $\mathbf{x}^* = \operatorname{argmin} \mathrm{Obj} : \Re^K \rightarrow \Re$) as well. The cartogram design problem of Section 3.5 is an example of such a problem; however, much more sophisticated neighborhood operators than the point perturbations we defined for that domain are available. For example, the downhill simplex method of Nelder and Mead (described in (Press et al., 1992, §10.4)) provides an effective set of local search moves for continuous optimization. Downhill simplex reaches a local optimum quickly, and Press et al.(Press et al., 1992) recommend embedding it within a multiple-restart or simulated-annealing framework. STAGE could provide an effective learning framework for multi-restart simplex search.

Confidence intervals. STAGE identifies good restart points by optimizing $\tilde{V}^\pi(F(x))$, the predicted expected outcome of search from x. However, in the context of a long run involving many restarts, it may be better to start search from a state with worse expected outcome but higher outcome *variance*. After all, what we really want to minimize is not the outcome of any one trajectory, but the minimum outcome over the whole collection of trajectories STAGE generates. A possible heuristic along these lines would be to exploit confidence intervals on $\tilde{V}^\pi$'s predictions to guide search, similarly to the Interval-Estimation (Kaelbling, 1993) and IEMAX (Moore & Schneider, 1996) algorithms.

Filtering refers to the early cutoff of an unpromising search trajectory—before it even reaches a local optimum—to conserve time for additional restarts and better trajectories. Heuristic methods for filtering have been investigated by, e.g., (Nakakuki & Sadeh, 1994). Perkins et al. (Perkins et al., 1997) have suggested that reinforcement-learning methods could provide a principled mechanism for deciding when to abort a trajectory. In the context of STAGE, filtering could be implemented simply as follows: cut off any π-trajectory when its predicted eventual outcome $\tilde{V}^\pi$ is worse than, say, the mean of all π-outcomes seen thus far. This technique would allow STAGE to exploit its learned predictions during both stages of search.

Sampling refers to the selection of candidate moves for evaluation during search. In our work, we have assumed that candidate moves are generated with a probability distribution that remains stationary throughout the optimization run. In optimization

practice, however, it is often more effective to modify the candidate distribution over the course of the search—for example, to generate large-step candidate moves more frequently early in the search process, and to generate small-step, fine-tuning moves more frequently later in search (Cohn, 1992, §2.4.4). In one approach (Su, Buntine, Newton, & Peters, 1998), linear regression is used to predict, over multiple simulated-annealing runs, the long-term outcome achieved by starting search at state $\mathbf{x}$ and with initial action a.[4] In reinforcement-learning terminology, their method learns to approximate the task's *state-action value function* $Q^\pi(\mathbf{x}, a)$ (Watkins, 1989). This form of value function allows the effects of various actions a to be predicted without having to actually apply the action or invoke the objective function. In optimization domains where objective function evaluations are costly, the Q^π value-function formulation offers the potential for significant speedup.

Direct Meta-Optimization. All the approaches discussed thus far have built evaluation functions by approximating a value function V^π or V^*, functions which predict the long-term outcomes of a search policy. However, an alternative approach not based on value function approximation, which we call *direct meta-optimization*, also applies. Direct meta-optimization methods assume a fixed parametric form for the evaluation function and optimize those parameters directly with respect to the ultimate objective. In symbols, given an evaluation function $\tilde{V}(x|\vec{w})$ parametrized by weights $\vec{w}$, we seek to learn $\vec{w}$ by directly optimizing the *meta-objective function*

$$M(\vec{w}) = \text{the expected performance of search using evaluation function } \tilde{V}(x|\vec{w}) \ .$$

The evaluation functions $\tilde{V}$ learned by such methods are not constrained by the Bellman equations: the values they produce for any given state have no semantic interpretation in terms of long-term predictions. The lack of such constraints means that less information for training the function can be gleaned from a simulation run; however, not having to meet the Bellman constraints may actually make learning easier. An example application of meta-optimization for combinatorial optimization may be found in (Ochotta, 1994). We believe that new memory-based stochastic optimization techniques (Moore & Schneider, 1996; Moore, Schneider, Boyan, & Lee, 1998) can significantly reduce the computational requirements of direct meta-optimization. Whether direct meta-optimization methods or Stage-like methods ultimately provide the most effective means of learning evaluation functions for global optimization remains to be seen.

Acknowledgments

We would like to acknowledge the invaluable contributions made to this research by Scott Fahlman, Tom Mitchell, and Tom Dietterich. We also acknowledge the support of a NASA Graduate Student Research Program fellowship (Boyan) and an NSF Career Award (Moore).

4. Note that the state vector $\mathbf{x}$ for simulated annealing consists of both the current configuration x and the current temperature T.

References

Ackley, D. H., & Littman, M. L. (1993). A case for distributed Lamarckian evolution. In Langton, C., Taylor, C., Farmer, J. D., & Ramussen, S. (Eds.), *Artificial Life III: Santa Fe Institute Studies in the Sciences of Complexity*, Vol. 10, pp. 487–509. Addison-Wesley, Redwood City, CA.

Aldous, D., & Vazirani, U. (1994). "Go with the winners" algorithms. In *Proceedings of the 35th Symposium on Foundations of Computer Science*, pp. 492–501.

Baluja, S., & Davies, S. (1997). Combining multiple optimization runs with optimal dependency trees. Tech. rep. CMU-CS-97-157, Carnegie Mellon University School of Computer Science.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Beveridge, J., Graves, C., & Lesher, C. E. (1996). Local search as a tool for horizon line matching. Tech. rep. CS-96-109, Colorado State University.

Boese, K. D. (1995). Cost versus distance in the traveling salesman problem. Tech. rep. CSD-950018, UCLA Computer Science Department.

Boese, K. D., Kahng, A. B., & Muddu, S. (1994). A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, *16*, 101–113.

Boyan, J. A. (1998). *Learning Evaluation Functions for Global Optimization*. Ph.D. thesis, Carnegie Mellon University.

Boyan, J. A. (2000). Technical update: Least-squares temporal difference learning. *Machine Learning*. To appear.

Chao, H.-Y., & Harper, M. P. (1996). An efficient lower bound algorithm for channel routing. *Integration: The VLSI Journal*.

Coffman, E. G., Garey, M. R., & Johnson, D. S. (1996). Approximation algorithms for bin packing: a survey. In Hochbaum, D. (Ed.), *Approximation Algorithms for NP-Hard Problems*. PWS Publishing.

Cohn, J. M. (1992). *Automatic Device Placement for Analog Cells in KOAN*. Ph.D. thesis, Carnegie Mellon University Department of Electrical and Computer Engineering.

Davies, S., & Baluja, S. (1998) Personal communication.

Dorling, D. (1994). Cartograms for visualizing human geography. In Hearnshaw, H. M., & Unwin, D. J. (Eds.), *Visualization in Geographical Information Systems*. Wiley.

Falkenauer, E., & Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. In *Proc. of the IEEE 1992 International Conference on Robotics and Automation*, pp. 1186–1192 Nice, France.

Friedman, N., & Yakhini, Z. (1996). On the sample complexity of learning Bayesian networks. In *Proc. 12th Conference on Uncertainty in Artificial Intelligence*.

Glover, F., & Laguna, M. (1993). Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems*. Scientific Publications, Oxford.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass.

Gusein-Zade, S. M., & Tikunov, V. S. (1993). A new technique for constructing continuous cartograms. *Cartography and Geographic Information Systems*, *20*(3), 167–173.

Hagen, L. W., & Kahng, A. B. (1997). Combining problem reduction and adaptive multistart: A new technique for superior iterative partitioning. *IEEE Transactions on CAD*, *16*(7), 709–717.

Hinton, G. E., & Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, *1*(1), 495–502.

Jagota, A., Sanchis, L., & Ganesan, R. (1996). Approximating maximum clique using neural network and related heuristics. In Johnson, D. S., & Trick, M. A. (Eds.), *DIMACS Series: Second DIMACS Challenge*. American Mathematical Society.

Johnson, D. S., & McGeoch, L. A. (1995). The traveling salesman problem: A case study in local optimization. In Aarts, E. H. L., & Lenstra, J. K. (Eds.), *Local Search in Combinatorial Optimization*. Wiley and Sons.

Juels, A., & Wattenberg, M. (1996). Stochastic hillclimbing as a baseline mathod for evaluating genetic algorithms. In Touretzky, D. S., Mozer, M. C., & Hasselmo, M. E. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 8, pp. 430–436.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. The MIT Press, Cambridge, MA.

Lin, L.-J. (1993). *Reinforcement Learning for Robots Using Neural Networks*. Ph.D. thesis, Carnegie Mellon University.

Martin, O. C., & Otto, S. W. (1994). Combining simulated annealing with local search heuristics. Tech. rep. CS/E 94-016, Oregon Graduate Institute.

McAllester, D., Kautz, H., & Selman, B. (1997). Evidence for invariants in local search. In *Proceedings of AAAI-97*.

Moore, A. W., & Lee, M. S. (1998). Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, *8*, 67–91.

Moore, A. W., & Schneider, J. (1996). Memory-based stochastic optimization. In Touretzky, D., Mozer, M., & Hasselmo, M. (Eds.), *Neural Information Processing Systems 8*.

Moore, A. W., Schneider, J. G., Boyan, J. A., & Lee, M. S. (1998). Q2: Memory-based active learning for optimizing noisy continuous functions. In *Proceedings of the Fifteenth Interational Conference on Machine Learning (ICML)*.

Nakakuki, Y., & Sadeh, N. M. (1994). Increasing the efficiency of simulated annealing search by learning to recognize (un)promising runs. Tech. rep. CMU-RI-TR-94-30, CMU Robotics Institute.

Ochotta, E. (1994). *Synthesis of High-Performance Analog Cells in ASTRX/OBLX*. Ph.D. thesis, CMU Electrical and Computer Engineering.

Perkins, T., Moll, R., & Zilberstein, S. (1997). Filtering to improve the performance of multi-trial optimization algorithms. Unpublished manuscript.

Press, W., Teukolsky, S., Vetterling, W., & Flannery, B. (1992). *Numerical Recipes in C: The Art of Scientific Computing* (Second edition). Cambridge University Press.

Puterman, M. L. (1994). *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Selman, B., Kautz, H., & Cohen, B. (1996). Local search strategies for satisfiability testing. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. American Mathematical Society.

Su, L., Buntine, W. L., Newton, R., & Peters, B. (1998). Learning as applied to stochastic optimization for standard cell placement. *International Conference on Computer Design*.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*.

Szykman, S., & Cagan, J. (1995). A simulated annealing-based approach to three-dimensional component packing. *ASME Journal of Mechanical Design*, *117*.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*(3/4).

Watkins, C. (1989). *Learning From Delayed Rewards*. Ph.D. thesis, Cambridge University.

Webb, S. (1994). Optimising the planning of intensity-modulated radiotherapy. *Phys. Med. Biol.*, *39*, 2229–2246.

Wong, D. F., Leong, H., & Liu, C. (1988). *Simulated Annealing for VLSI Design*. Kluwer.

Zhang, W., & Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1114–1120.

Zhang, W., & Dietterich, T. G. (1998). Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling..