Chapter 5

STAGE: Analysis

In the preceding chapter, I gave empirical evidence that STAGE is an effective optimization technique for a wide variety of domains. In this chapter, I break down the causes for that effectiveness. Does STAGE's power derive from its use of machine learning, per its design? Or are incidental aspects of the way it organizes its search just as responsible for STAGE's success? How robust is the algorithm to varying choices of feature sets, function approximators, and other user-controllable parameters?

This chapter reports the results of a series of experiments that explore these questions empirically. Most of the experiments are performed in the domains of VLSI channel routing, as described in Section 4.3, and cartogram design, as described in Section 4.6.

5.1 Explaining STAGE's Success

STAGE performs superbly on the channel routing domain, not only outperforming hillclimbing as it was trained to do, but also finding better solutions on average than the best simulated annealing runs. How can we explain its success? There are at least three possibilities:

Hypothesis A: STAGE works according to its design. Gathering data from a number of hillclimbing trajectories, it learns to predict the outcome of hillclimbing starting from various state features, and it exploits these predictions to reach improved local optima.

Hypothesis B: Since STAGE alternates between simple hillclimbing and another policy, it simply benefits from having more random exploration. STAGE uses hillclimbing on the learned $\tilde{V}^\pi$ as its secondary policy, but alternative policies would do just as well.

Hypothesis C: The function approximator may simply be smoothing the objective function, which helps eliminate local minima and plateaus.

In this section, I first describe experiments which reject the latter two hypotheses. I then describe experiments giving further evidence for Hypothesis A, that STAGE does indeed work as designed. Finally, I analyze an instance where STAGE failed.

5.1.1 $\tilde{V}^\pi$ versus Other Secondary Policies

Hypothesis B attributes STAGE's success to its unusual regime of alternating between hillclimbing and a secondary search policy. Perhaps any secondary policy which perturbed the search away from its current local optimum would be just as effective as STAGE's secondary policy of hillclimbing on the learned evaluation function $\tilde{V}^\pi$.

To test this, I ran experiments with several alternative choices for STAGE's secondary policy:

Policy B0 (normal STAGE): Hillclimb on $\tilde{V}^\pi$.

Policy B1: Perform a random walk of fixed length ω. (I tried setting ω to 1, 3, 10, and 40.)

Policy B2: Hillclimb on the *inverse* of $\tilde{V}^\pi$. In other words, move stochastically to a state which $\tilde{V}^\pi$ predicts to be the *worst* place from which to begin a search.

Policy B3: Hillclimb on a corrupted version of $\tilde{V}^\pi$, trained by replacing every target value in its training set with a random value.

Each of these policies was alternated with standard hillclimbing (patience=250), so as to imitate STAGE's normal regime. Experiments B0, B2 and B3 approximated $\tilde{V}^\pi$ using linear regression over the three channel routing features $\langle w, p, U\rangle$, each scaled to the range $[0, 1]$. I ran each resulting algorithm 50 times on channel routing instance YK4, limiting each run to TOTEVALS $= 10^5$ total moves considered. (Note that the results of Section 4.3 were tabulated over longer runs of length TOTEVALS $= 5 \cdot 10^5$.)

Instance	Algorithm	Performance (50 runs each)		
		mean	best	worst
YK4	B0 (STAGE)	**13.54**±1.13	**12**	41
	B1 ($\omega = 1$)	16.94±0.23	15	**19**
	B1 ($\omega = 3$)	17.16±0.24	16	**19**
	B1 ($\omega = 10$)	19.12±0.27	17	21
	B1 ($\omega = 40$)	22.64±0.26	20	25
	B2	37.86±3.52	14	52
	B3	15.38±0.64	13	27

TABLE 5.1. Results of different secondary policies on channel routing instance YK4. In each case, TOTEVALS $= 10^5$.

The results, given in Figure 5.1 and Table 5.1, show conclusively that the choice of secondary policy does matter. STAGE's policy significantly outperformed all others,

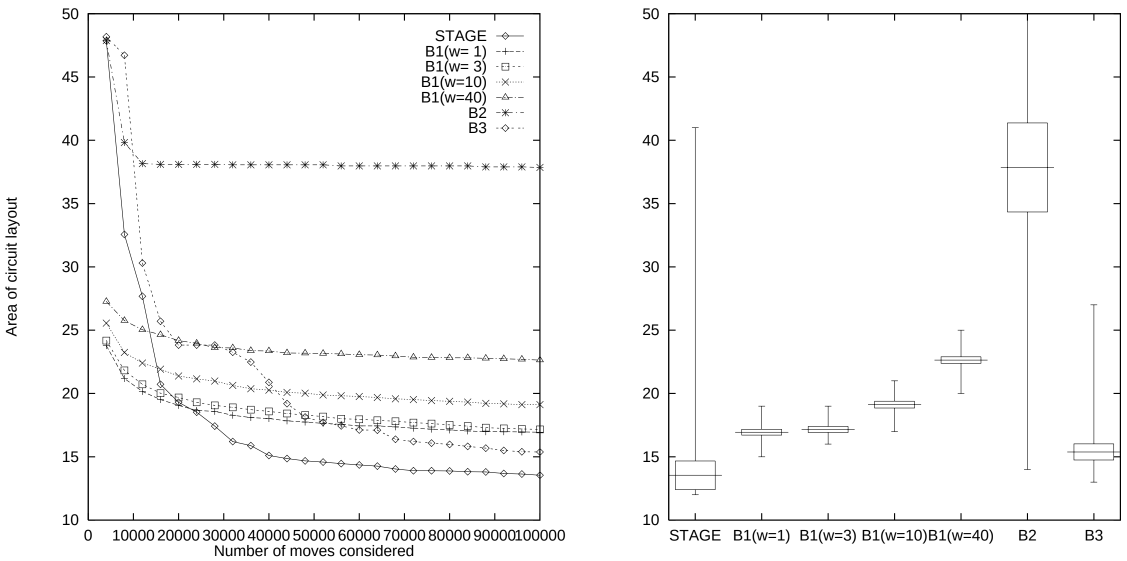

FIGURE 5.1. Performance of different secondary policies on channel routing instance YK4

with 48 of its 50 runs producing a solution circuit of quality between 12 and 14. (One outlier run did no better than 41; I analyze this in Section 5.1.3.) The random walk policies (B1) were consistently inferior. Indeed, the pattern of the B1 results shows clearly that performance degraded more and more as additional undirected exploration was allowed. Experiment B2, in which the search was purposely led against $\tilde{V}^{\pi}$ toward states judged *unpromising*, performed much worse than even the random-walk policies. This provides further evidence that $\tilde{V}^{\pi}$ has learned useful predictive information about the features.

Finally, Experiment B3 performed better than the random-walk policies, but still significantly worse than STAGE. Why did assigning random target values to the training set for $\tilde{V}^{\pi}$ result in even moderately good performance? The answer is that in this experiment, the linear regression model has only four coefficients to fit, and one of these (the coefficient of the constant term) has no effect when $\tilde{V}^{\pi}$ is used as an evaluation function to guide search. Thus, even choosing random values for the 3 meaningful coefficients would sometimes lead search in the same useful direction as the true V^{π}. This analysis is supported by the plot of Figure 5.2. This plot compares the successive local minima visited by a typical run of STAGE and a typical run of Experiment B3. Clearly, STAGE learns a policy which keeps it in a high-quality part of the space during most of its search, while B3 only occasionally "lucks into" a good solution. Still, these results highlight the potential for an alternative algorithm to

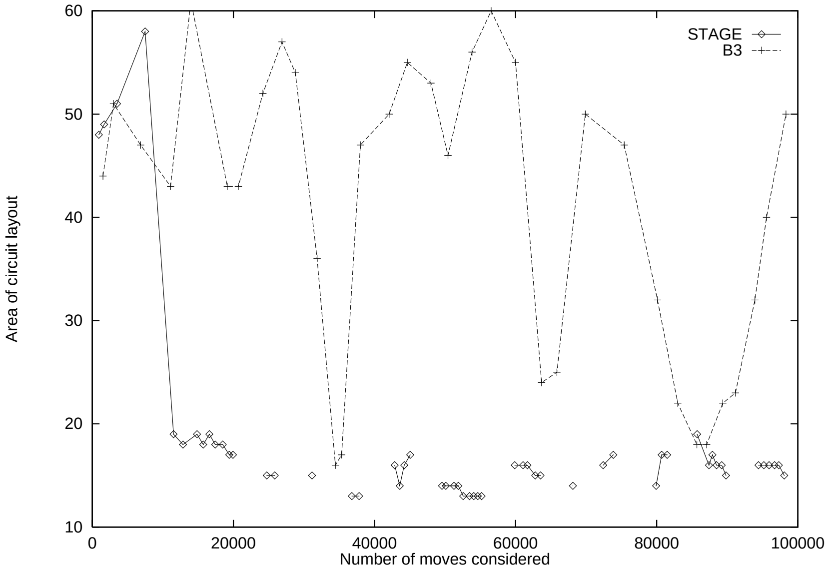

FIGURE 5.2. Quality of local minima reached on each successive iteration of a STAGE run and a B3 run. Gaps in the STAGE plot correspond to iterations on which a restart occurred.

STAGE, which works by optimizing the coefficients of a $\tilde{V}^{\pi}$-like function directly (see Section 8.2.3).

The results of this series of experiments do not prove that hillclimbing on $\tilde{V}^{\pi}$ is the only policy which can productively lead search away from a local minimum and into the attracting basin of a potentially better local minimum. However, they do at least demonstrate that not every secondary policy will be effective—and that STAGE's learned policy is more effective than most.

5.1.2 $\tilde{V}^{\pi}$ versus Simple Smoothing

STAGE performs "predictive smoothing" of the original objective function Obj: the function $\tilde{V}^{\pi}$ is a continuous mapping from the features of state x to the Obj value that policy π is predicted to eventually reach from x. But perhaps, as suggested by Hypothesis C above, the predictive aspect of STAGE's smoothing is irrelevant. Perhaps STAGE's success could be replicated by simply smoothing Obj directly over the feature space, which would eliminate the original objective function's local minima and plateaus.

One flaw of this hypothesis is immediately apparent: in channel routing as in most of the other domains described in Chapter 4, the objective function value Obj(x) is provided to STAGE as one of the features of the input feature space. Therefore, any function approximator could fit Obj perfectly over the feature space by simply copying the $\mathrm{Obj}(x)$ feature from input to output. Clearly, this perfect "smoothing" of Obj would not eliminate any local minima from Obj; STAGE would reduce to standard multi-restart hillclimbing. Experiment C0, described below, empirically demonstrates precisely this effect.

However, perhaps an imperfect smoothing of Obj would produce the hypothesized good outcome. I performed the following series of experiments. For all experiments except HC, the basic STAGE regime is unchanged; only the training set's target values for the function approximator are modified.

Experiment HC: Multi-restart hillclimbing, accepting equi-cost moves, $\mathit{patience} = 500$.

Experiment STAGE (same as Experiment B0): Normal STAGE, modelling $V^\pi(\mathrm{x})$ by linear regression over the three features $\langle w(x), p(x), U(x)\rangle$.

Experiment C0 (perfect "smoothing"): For each state in STAGE's training set, represented by the features $\langle w(x), p(x), U(x)\rangle$, train the function approximator to model not $V^\pi(\mathrm{x})$ but the objective function $\mathrm{Obj}(x) = w(x)$. Here, the function approximator can simply copy $w(x)$ from input to output, so no real smoothing occurs. Results are shown in Table 5.2 and Figure 5.3: as expected, this policy performed similarly to multi-start hillclimbing.

Experiment C1: I eliminated the w feature from the training set, so the function approximator had to model $\mathrm{Obj}(x)$ as a linear function of only $\langle p(x), U(x)\rangle$. The results were extremely poor. A closer look explained why. Recall that Wong defined

$$U(x) = \sum_{i=1}^{w(x)} u_i(x)^2$$

where $u_i(x)$ is the fraction of track i that is unoccupied [Wong *et al.* 88]. In poor-quality solutions such as those visited at the beginning of a search, $u_i(x)$ is close to 1 for each track, so $U(x) \approx \mathrm{Obj}(x)$. Thus, even a linear function approximator can model Obj quite accurately by simply copying $U(x)$ from input to output. But this smoothed objective function, $U(x)$, makes a terrible evaluation function for optimization. Starting from a local optimum of Obj, where no single move can reduce $w(x)$ further, the only way a greedy search can reduce U is to reduce the variance of the $u_i(x)$ values, that is, to distribute the wires evenly throughout the available tracks. That is exactly the opposite of the successful strategy that STAGE discovers: to distribute the wires as unevenly as possible, thereby creating some near-empty tracks which hillclimbing can more readily eliminate.

Experiment C2: Using the insight gained from the last experiment, I replaced $U(x)$ by a closely related feature:

$$V(x) \stackrel{\text{def}}{=} \sum_{i=1}^{w(x)} (1 - u_i(x))^2$$

With a bit of algebra, it can be shown that $V(x) \equiv U(x) - w(x) + C$, where C is a constant. With $w(x)$ subtracted from the feature, the regression model can no longer fit Obj well by simply copying a feature through. Indeed, after 10^5 steps of STAGE, the linearly smoothed fit of Obj over $\langle p(x), V(x)\rangle$ has an RMS error around 13, compared with an RMS error of less than 0.1 in Experiment C1. Thus, this fit does perform significant smoothing. The smoothed fit assigns a negative coefficient to $V(x)$, so searching on it tends to *increase* the variance of the u_i, resulting in improved overall performance relative to both C0 and C1. Nevertheless, C2's performance is still much worse than STAGE's. (In Section 5.2.1 below, I show that STAGE's performance with the p and V features is even better than STAGE with w, p and U.)

Instance	Algorithm	Performance (50 runs each)			RMS
		mean	best	worst	of fit
YK4	HC	21.32±0.22	19	23	—
	STAGE	**13.54**±1.13	**12**	41	2.1
	C0	21.06±0.74	15	27	0.0
	C1	40.00±0.18	38	41	0.05
	C2	19.34±1.00	15	28	13.6

TABLE 5.2. Performance comparison of STAGE with STAGE-like algorithms that simply smooth Obj. For each run, TOTEVALS = 10^5. The RMS column gives a typical root-mean-square error of the learned evaluation function over its training set at the end of the run.

In addition to the results plotted above, I also repeated experiments C1 and C2 using quadratic rather than linear regression to smooth Obj. This did decrease the RMS error of the approximations, to approximately 0.04 for C1 and 3.9 for C2.

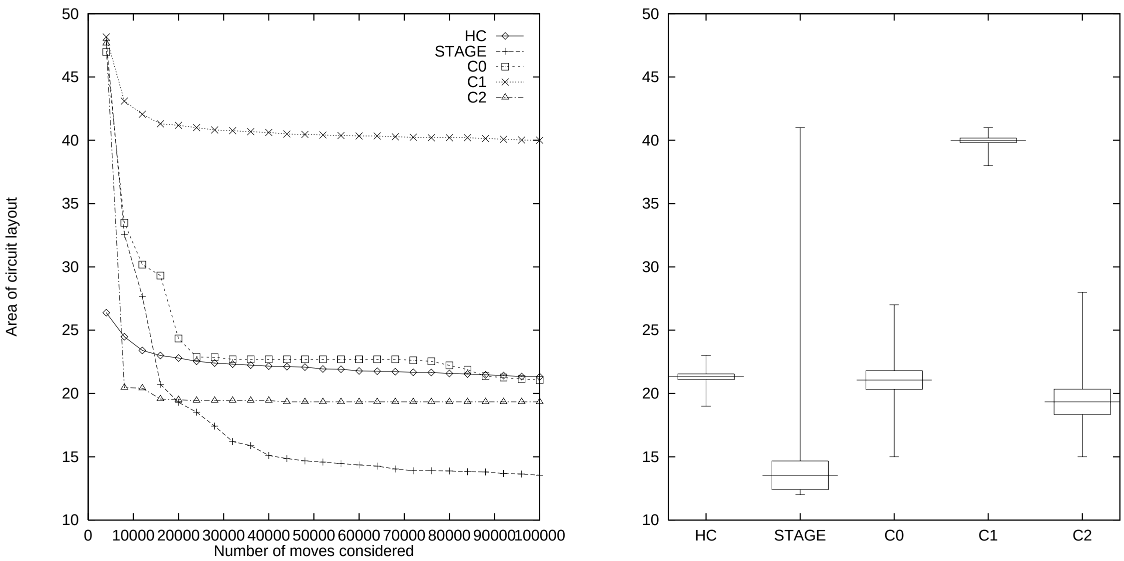

FIGURE 5.3. Performance on the experiments of Section 5.1.2

However, it did not improve optimization performance in either case. From this series of experiments, I conclude that, at least for this problem, STAGE's success cannot be attributed to simple smoothing of the objective function. STAGE's predictive smoothing works significantly better.

5.1.3 Learning Curves for Channel Routing

I have presented evidence that STAGE's secondary policy, searching on $\tilde{V}^{\pi}$, is more helpful to optimization than other reasonable policies based on randomness or smoothing. This evidence contradicts Hypotheses B and C outlined on page 107. I now examine STAGE's success on channel routing more closely, to support Hypothesis A: that STAGE's leverage is indeed due to machine learning.

Figure 5.4 illustrates three short runs of STAGE on routing instance YK4. In each row of the figure, the left-hand graph illustrates the successive local minima visited over the course of STAGE's run: a diamond symbol ($\diamondsuit$) marks a local minimum of Obj, and a plus symbol (+) marks a local minimum of the learned evaluation function $\tilde{V}^{\pi}$. There is a gap in the plot each time STAGE resets search to the poor initial state. Observe that the first two runs, (A) and (B), perform well, both attaining a best solution quality of 12; whereas run (C) performs uncommonly poorly, with a best solution quality of 34.

The right-hand graphs track the evolution of $\tilde{V}^{\pi}$ over the course of each run, plotting the coefficients β_w, β_p, and β_U on each iteration, where

$$\tilde{V}^\pi(x) = \beta_w \cdot w(x) + \beta_p \cdot p(x) + \beta_U \cdot U(x) + \beta_1$$

For clarity, the constant coefficient β_1 is omitted from the plots: its value, typically around -80, has no effect on search since it leaves the relative ordering of states unchanged.

The coefficient plots show that all runs converge rather quickly to similar $\tilde{V}^\pi$ functions. The learned $\tilde{V}^\pi$ functions have a high positive coefficient on $w(x)$, a negative coefficient of nearly equal magnitude on $U(x)$, and a coefficient near zero on $p(x)$. The assignment of a negative coefficient to U is surprising, because U measures the sparseness 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 search toward terrible solutions in which each subnet occupies its own track. Indeed, the hand-tuned evaluation function built by Wong et al. for this problem assigned $\beta_U = +10$ [Wong *et al.* 88].

However, the positive coefficient on w cancels out this bias. Recall from Experiment C2 of the previous section that the following relation holds:

$$V(x) = U(x) - w(x) + C$$

where $V(x)$ is a feature measuring the variance in track fullness levels, and C is a constant. By assigning $\beta_w \approx -\beta_U$, STAGE builds the evaluation function

$$\begin{aligned}\tilde{V}^\pi(x) &\approx -\beta_U \cdot w(x) + \beta_p \cdot p(x) + \beta_U \cdot U(x) + \beta_1 \\ &= \beta_U(U(x) - w(x)) + \beta_p \cdot p(x) + \beta_1 \\ &= \beta_U \cdot V(x) + \beta_p \cdot p(x) + \beta_1'\end{aligned}$$

Thus, in order to predict search performance, STAGE learns to extract the variance feature $V(x)$. The coefficient β_U is negative, so minimizing STAGE's learned evaluation function biases search toward increasing $V(x)$—that is, toward creating solutions with an uneven distribution of track fullness levels. Although this characteristic is not itself the mark of a high-quality solution, it does help lead hillclimbing search to high-quality solutions.

Given that all three STAGE runs of Figure 5.4 learned quickly to extract the variance feature $U - w$, why are their performance curves so different? A close look reveals that the third feature, $p(x)$, matters: performance breaks through to the "excellent" level when the coefficient β_p becomes positive. In run (A), which is most typical, β_p is positive throughout the entire run, and STAGE reaches excellent solutions on every iteration. In run (B), β_p becomes consistently positive only around

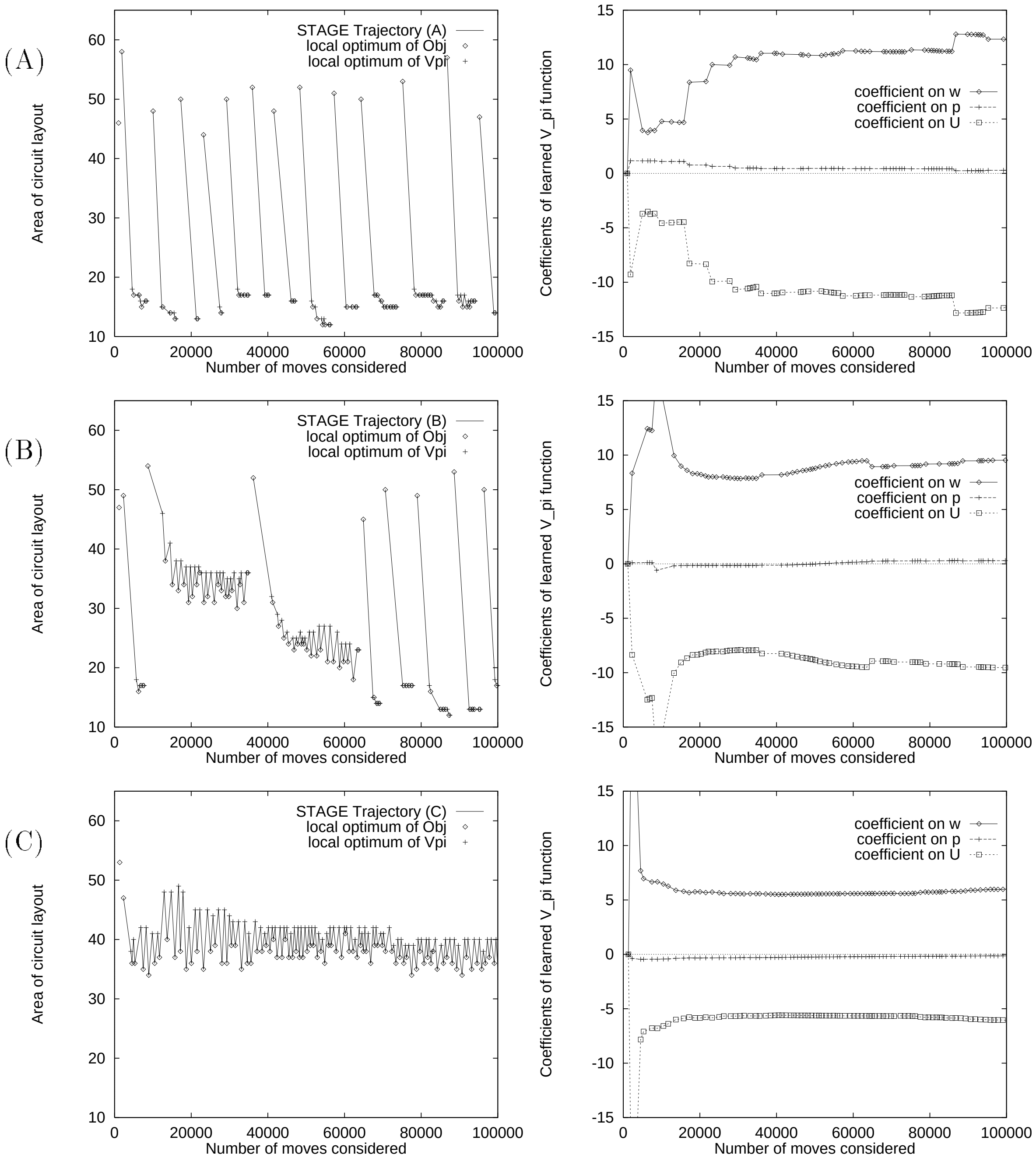

FIGURE 5.4. Three runs of STAGE on channel routing instance YK4. Left-hand graphs plot the Obj value reached at the end of each search trajectory within STAGE; right-hand graphs plot the evolution of the coefficients of $\tilde{V}^{\pi}(x)$ over the run.

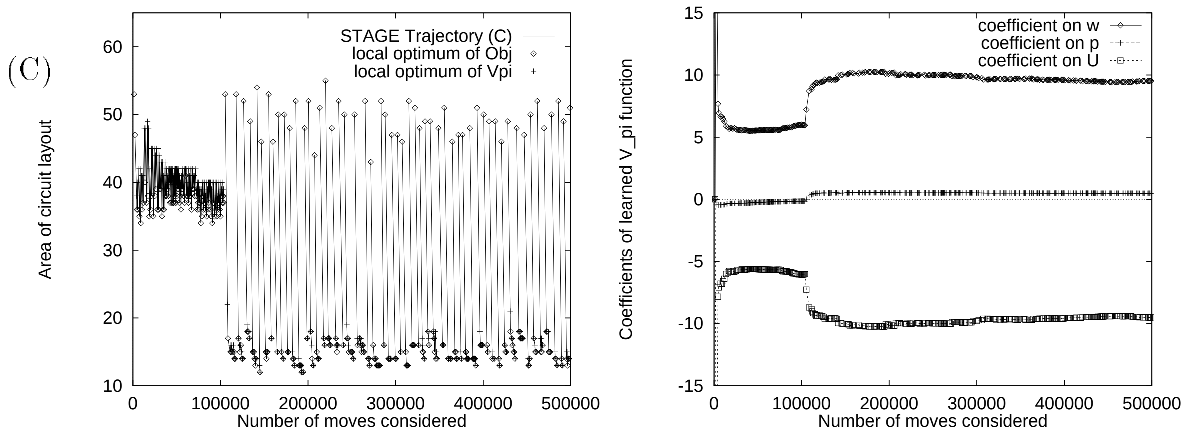

FIGURE 5.5. Run (C) of Figure 5.4, extended to TOTEVALS = 500,000.

move 60000, which corresponds to STAGE's breakthrough. Finally, in the outlier run (C), β_p remains negative throughout the first $100,000$ moves. However, it is slowly increasing, and in fact, if the same run is allowed to extend to TOTEVALS $= 500,000$, then β_p soon becomes positive and performance improves (see Figure 5.5). This run reaches a solution quality of 12 after $145,000$ moves.

These runs illustrate that STAGE's convergence rate may depend on the particular regions that STAGE happens to explore. In run (C), STAGE apparently becomes "stuck" in a region of poor local optima until the β_p coefficient becomes positive, triggering a random restart from which the learned $\tilde{V}^\pi$ can be effectively exploited. This suggests that faster convergence may be attained by more frequent random restarts; I investigate this in Section 5.2.4.

Despite the varying convergence rates, all the STAGE runs on instance YK4 do eventually converge to an approximation of V^π close to the following:

$$\tilde{V}^\pi(x) \approx 10 \cdot w(x) + 0.5 \cdot p(x) - 10 \cdot U(x) - 80 \tag{5.1}$$

The coefficients on all three features are significant. As discussed above, the component $10(w(x) - U(x))$ biases search toward solutions with an uneven distribution of track fullness levels. As for $p(x)$, recall from Section 4.3 that it provides a lower bound on all solutions derived from x by future merging of tracks. Thus, the coefficient of $+0.5$ on $p(x)$ helps keep the search from straying onto unpromising trajectories. In Section 6.2 I show that STAGE learns similar coefficients on a number of other instances of the channel routing problem.

5.1.4 STAGE's Failure on Boggle Setup

STAGE improved significantly over multi-restart hillclimbing on six of the seven large-scale optimization domains presented in Chapter 4. On the "Boggle setup" domain of Section 4.8, however, STAGE failed to produce a significant improvement. What explains this failure?

According to Hypothesis A, STAGE succeeds when it can identify and exploit trends in the mapping from starting state to hillclimbing outcome. As it turns out, the Boggle domain illustrates the converse: when the results of hillclimbing from a variety of starting states show no discernible trend, then STAGE will fail.

The following experiment makes this clear. I 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 5.6 compares the performance of hillclimbing from these sets of states. Apparently, the restarting policy is irrelevant to hillclimbing's mean performance: on average, hillclimbing 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 is consistent with our understanding of how STAGE works.

5.2 Empirical Studies of Parameter Choices

We have shown that STAGE learns to predict search performance as a function of starting state features, and exploits these predictions to improve optimization results. However, STAGE's performance clearly depends on the user's choice of features $F(x)$, the function approximator used to model $\tilde{V}^\pi$, the policy π being learned, and other parameters. Section 3.4 analyzed these choices theoretically; this section continues the analysis from an empirical perspective.

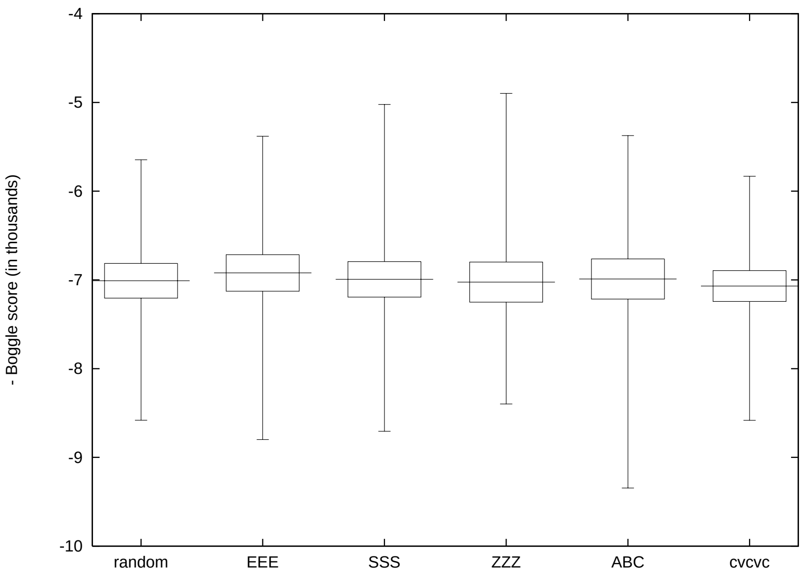

FIGURE 5.6. Boggle: average performance of 50 restarts of hillclimbing from six different sets of starting states

5.2.1 Feature Sets

Practical domains are generally abundant in potentially useful state features. But which of these should be given to STAGE for its learning? Providing many detailed state features, perhaps even a complete description of the state, gives STAGE the opportunity to model V^π very accurately; however, the extra features require more parameters to be fit, which increases the complexity of the fitter's task and may slow STAGE down intolerably. Conversely, using only a few coarse features results in efficient fitting, but limits the prediction accuracy that STAGE can achieve.

I compared STAGE's performance with a wide variety of feature sets on two domains: channel routing (instance YK4) and cartogram design (instance US49). These results are consistent with my informal experience with STAGE on many other domains: smaller feature sets learn faster and work better.

On the channel routing benchmark, I compared seven sets of features for representing any given state x. The other parameters to STAGE are detailed in Table 5.3. The results of 50 runs with each feature set are summarized in Figure 5.7 and Table 5.4. The table also gives the average final RMS error for $\tilde{V}^\pi$ on each experiment, but note that these numbers are not directly comparable, since the training set distributions can be markedly different from experiment to experiment. The feature sets tried were as follows:

WPU: $\langle w(x), p(x), U(x) \rangle$, the original features used in Section 4.3 and analyzed above.

PU: $\langle p(x), U(x) \rangle$ only, dropping the objective function $\mathrm{Obj}(x) = w(x)$ from the input space. This performs badly, for the same reasons as Experiment C1 of Section 5.1.2 above. From this and other (unreported) experiments, I conclude that $\mathrm{Obj}(x)$ should generally be included as a feature.

PV: $\langle p(x), V(x) \rangle$, where $V(x) \stackrel{\text{def}}{=} \sum_{i=1}^{w(x)} (1 - u_i(x))^2$, as suggested by Experiment C2 of Section 5.1.2 above. This performs well and, as can be seen in Figure 5.7, reaches good solutions very quickly! This shows that knowing what features are good for prediction and explicitly providing them leads to great performance. But it was through STAGE's success with the original features that this more compact feature set was discovered.

rrr: $\langle r_1(x), r_2(x), r_3(x) \rangle$, where $r_i(x)$ is a pseudorandom number in $[0, 1]$ chosen deterministically as a function of x and i. That is, these are three features of the state which are completely irrelevant for predicting V^π. As expected, this performs poorly, similar to the random-walk experiments of Section 5.1.1 above.

WPUrrr: $\langle w(x), p(x), U(x), r_1(x), r_2(x), r_3(x) \rangle$. This experiment tests whether performance degrades in the presence of irrelevant features. The result is encouraging: there is only a very slight performance degradation relative to WPU. Early on in each run, $\tilde{V}^\pi$ learns to assign near-zero coefficients to the irrelevant features.

WPU2: $\langle w(x), w(x)^2, p(x), p(x)^2, U(x) \rangle$. This includes the two quadratic terms that were used in the hand-tuned evaluation function for simulated annealing of [Wong *et al.* 88]. Apparently, these two terms are not useful to STAGE, as they are assigned relatively small coefficients and do not improve performance. (An outlier run in this series is responsible for the increased standard error of the mean.)

WPU2++: $\langle w(x), w(x)^2, p(x), p(x)^2, U(x), f_{t(1)}(x), f_{t(2)}(x), \ldots f_{t(144)}(x) \rangle$. Here, $t(i)$ denotes the current track $(1 \ldots w(x))$ on which subnet i has been placed, and $f_k(x) = 1 - u_k(x)$ denotes the fullness of track k. These 149 features provide much detail of state x, though not enough to reproduce x completely. The result: learning occurs slowly, both in terms of running time per iteration (since a 150×150 matrix must be inverted after each hillclimbing trajectory) and in terms of number of iterations to reach good performance. However, when these runs are allowed to extend ten times longer, to TotEvals $= 5{\cdot}10^6$, performance does reach the same excellent level as the other good STAGE runs.

Parameter	Setting
π	stochastic hillclimbing, rejecting equi-cost moves, patience=250
ObjBound	$-\infty$
features	between 2 and 149, depending on experiment
fitter	linear regression
Pat	250
TotEvals	500,000

Table 5.3. Summary of STAGE parameters for channel routing feature comparisons

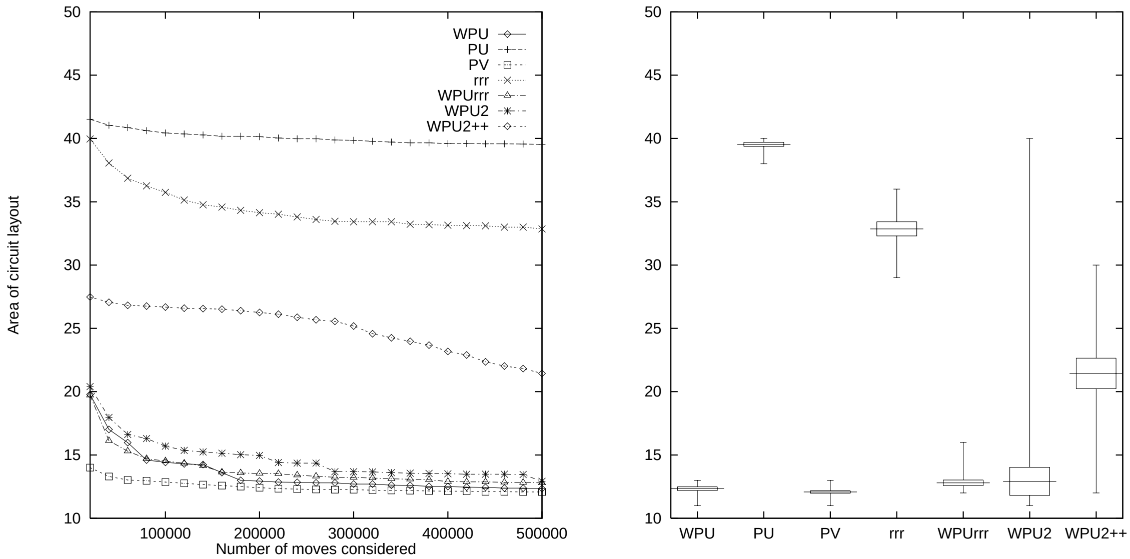

Figure 5.7. Performance of different feature sets on channel routing instance YK4

The second experiment with feature sets is from the U.S. cartogram domain. Recall from Section 4.6 that the objective is to deform the map's states so that their areas meet new prescribed targets (in this instance, proportional to electoral vote), but their shapes and connectivity remain similar to the un-deformed map. The objective function was defined as

$$\mathrm{Obj}(x) = \Delta\mathrm{Area}(x) + \Delta\mathrm{Gape}(x) + \Delta\mathrm{Orient}(x) + \Delta\mathrm{Segfrac}(x)$$

Instance	Algorithm	Performance (50 runs each)			RMS
		mean	best	worst	of fit
YK4	WPU	12.34±0.15	**11**	**13**	2.7
	PU	39.54±0.16	38	40	2.8
	PV	**12.08**±0.10	**11**	**13**	2.8
	rrr	32.86±0.56	29	36	3.7
	WPUrrr	12.80±0.22	12	16	2.7
	WPU2	12.92±1.11	**11**	40	2.7
	WPU2++	21.44±1.20	12	30	1.7

TABLE 5.4. Results with different feature sets on channel routing instance YK4

and STAGE used those four subcomponents of the objective function as its features for learning. Many other features can be imagined for this domain; I examine STAGE's performance with several of them here. These experiments used linear regression to fit $\tilde{V}^\pi$. Other STAGE parameters are detailed in Table 5.5, and the results are shown in Figure 5.8 and Table 5.6. The feature sets compared are as follows:

Asp.1: $\langle \mathrm{Asp}(x) \rangle$, a single feature describing the aspect ratio of the current map's bounding rectangle. Note that most moves leave this feature unchanged. STAGE learns a coefficient near zero for this feature, and its performance of Obj ≈ 0.16 is about the same as that of multi-restart hillclimbing.

Peri.1: $\langle \mathrm{Peri}(x) \rangle$, a single feature which sums the perimeter over all the states. STAGE works surprisingly well with this feature. It learns a positive coefficient on $\mathrm{Peri}(x)$; that is, it learns that hillclimbing performs better when it begins from a map which is "scrunched."

Cop.2: the horizontal and vertical coordinates of map x's center of population. I thought these would be a good pair of features for the US49 domain, since a good cartogram should have its population center near the geometric center of the map. However, these features were ineffective, under both linear regression (shown in graphs) and quadratic regression (not shown).

Obj.4: $\langle \mathrm{Area}(x), \mathrm{Gape}(x), \mathrm{Orient}(x), \mathrm{SegFrac}(x) \rangle$, the four subcomponents of the objective function. These features worked quite well, though not as well as Peri.1 on average. Note that the better STAGE results of Section 4.6 used quadratic regression over these four features.

ObjPeri.5: Given the good performance of Peri.1 and Obj.4, it is natural to combine them into a 5-feature representation for STAGE. This feature set performed comparably to experiment Obj.4.

St.15: This larger feature set concentrates on five "important" states of the map: New York, Texas, Florida, California, and Illinois. For each of these it provides three features: the horizontal and vertical coordinates of the state's current centroid (expressed relative to the map's current bounding rectangle) and the state's area. Note that this is the first feature set I have tailored specifically to this instance. However, STAGE's performance with these features, while better than multi-restart hillclimbing, is worse than with most of the simple instance-independent feature sets discussed above.

St.147: This experiment includes the three centroid and area features for all 49 of the map's states. With this many features, STAGE is significantly slowed down. For a fixed number of moves, however, solution quality is significantly better than multi-restart hillclimbing.

This series of experiments shows that for almost any natural choice of domain features, STAGE is able to discover a structure over those features which can be exploited to improve performance significantly over multi-restart hillclimbing.

Parameter	**Setting**
π	stochastic hillclimbing, rejecting equi-cost moves, patience=200
ObjBound	0
features	between 1 and 147, depending on experiment
fitter	linear regression
Pat	200
TotEvals	1,000,000

Table 5.5. Summary of STAGE parameters for cartogram feature comparisons.

From the experiments reported here on the channel routing and cartogram domains, and from other informal experiments not shown, I conclude that STAGE works best with small, simple sets of features. Small feature sets not only are speediest for STAGE to work with, but can also, in cases such as cartogram Experiment Peri.1, provide a bias toward good extrapolation by $\tilde{V}^{\pi}$, enabling successful exploration. Features associated with subcomponents of the objective function, or the variance of such subcomponents, were often helpful. With the function approximators tested here, STAGE is not overly sensitive to the inclusion of extra random features; however, it

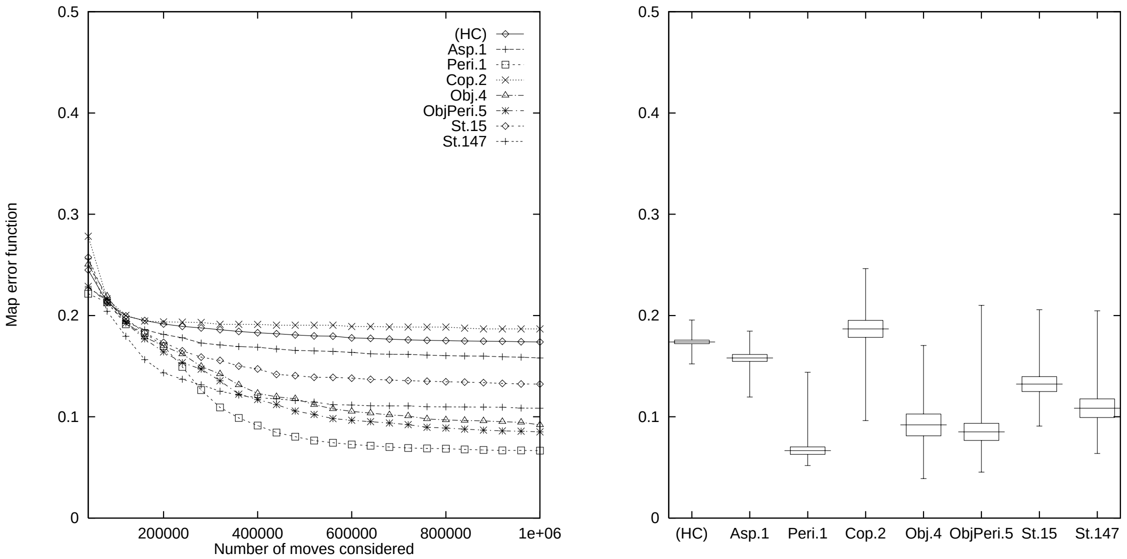

FIGURE 5.8. Cartogram performance with different feature sets

Instance	Algorithm	Performance (50 runs each)		
		mean	best	worst
US49	Asp.1	0.158±0.004	0.120	0.185
	Peri.1	**0.067**±0.004	0.052	**0.144**
	Cop.2	0.187±0.008	0.096	0.246
	Obj.4	0.092±0.011	**0.039**	0.170
	ObjPeri.5	0.085±0.008	0.045	0.210
	St.15	0.132±0.007	0.091	0.206
	St.147	0.109±0.009	0.064	0.205

TABLE 5.6. Cartogram results with different feature sets

can sometimes be drawn into abysmal performance even worse than multi-restart hill-climbing by a particularly bad set of features, as in channel routing Experiment PU.

5.2.2 Fitters

Another key parameter to STAGE is the function approximator or "fitter" used to model $V^\pi(x)$. In Section 3.4.3, I argued that fitters having a linear architecture, such as polynomial regression (of any degree), are best suited to STAGE. Here, I empirically investigate the performance of STAGE with a range of polynomial fitters, ranging from degree 1 to degree 5, on both the channel routing and cartogram domains.

The table below lists the eight fitters I tested. Let $\langle f_1, f_2, \ldots, f_D \rangle$ denote the features of each domain given to STAGE; note that $D = 3$ in the channel routing domain and $D = 4$ in the cartogram domain. For each fitter, the table also gives the total number of coefficients being fit, both as a function of arbitrary D and in the case where $D = 4$.

Key	Description	# params
F1:	linear regression	$D + 1 = 5$
F2:	quadratic regression	$\binom{D+2}{2} = 15$
F3:	cubic regression	$\binom{D+3}{3} = 35$
F4:	quartic regression	$\binom{D+4}{4} = 70$
E2:	quadratic regression *without cross-terms*. For each domain feature f_i, this model includes the terms f_i and f_i^2 but no cross-terms involving the product of more than one feature.	$2D + 1 = 9$
E3:	cubic regression without cross-terms	$3D + 1 = 13$
E4:	quartic regression without cross-terms	$4D + 1 = 17$
E5:	quintic regression without cross-terms	$5D + 1 = 21$

Results are given in the usual form in Tables 5.7 and 5.8 and Figures 5.9 and 5.9. On both domains, the results indicate that the choice of fitter has a relatively minor impact on performance. On the channel routing domain, slightly better results and fewer poor outlier runs were obtained from the most highly biased models, linear regression and crossterm-less quadratic regression. This is consistent with our earlier observation that the linear approximation $V^\pi(x) \approx 10w(x) + 0.5p(x) - 10U(x) - 80$ suffices for excellent performance here. On the cartogram domain, however, somewhat more complex models performed slightly better.

In these experiments, STAGE's computational overhead for function approximation was not a significant component of running time, even for the most complex model here, quartic regression with all cross-terms. However, since the expense of least-squares quartic regression increases as $O(D^{12})$, this overhead would certainly become prohibitive for more than $D = 4$ features. In my experience, quadratic regression (either with or without cross terms) provides sufficient model flexibility, enough bias to enable aggressive extrapolation, and efficient computational performance.

Instance	Algorithm	Performance (50 runs each)		
		mean	best	worst
YK4	F1 (linear)	**12.26**±0.17	**11**	14
	F2 (quadratic)	14.30±1.16	12	37
	F3 (cubic)	13.08±0.35	12	19
	F4 (quartic)	12.86±0.42	**11**	19
	E2 (crossterm-less quadratic)	**12.30**±0.14	**11**	**13**
	E3 (crossterm-less cubic)	**12.80**±1.07	12	39
	E4 (crossterm-less quartic)	13.54±1.15	**11**	39
	E5 (crossterm-less quintic)	15.02±0.71	12	24

TABLE 5.7. Channel routing results with different polynomial function approximators over the 3 features $\langle w, p, U \rangle$

Instance	Algorithm	Performance (50 runs each)		
		mean	best	worst
US49	(HC)	0.174±0.002	0.152	0.195
	F1 (linear)	0.092±0.011	0.039	0.170
	F2 (quadratic)	**0.057**±0.004	**0.038**	0.103
	F3 (cubic)	**0.057**±0.004	0.038	0.126
	F4 (quartic)	0.069±0.003	0.045	**0.096**
	E2 (crossterm-less quadratic)	0.078±0.010	0.041	0.174
	E3 (crossterm-less cubic)	0.067±0.005	0.038	0.135
	E4 (crossterm-less quartic)	**0.058**±0.004	0.040	0.111
	E5 (crossterm-less quintic)	0.069±0.005	0.041	0.106

TABLE 5.8. Cartogram performance with different polynomial function approximators over the 4 features $\langle \Delta\text{Area}, \Delta\text{Gape}, \Delta\text{Orient}, \Delta\text{Segfrac} \rangle$. All the STAGE runs significantly outperform multi-restart hillclimbing (HC).

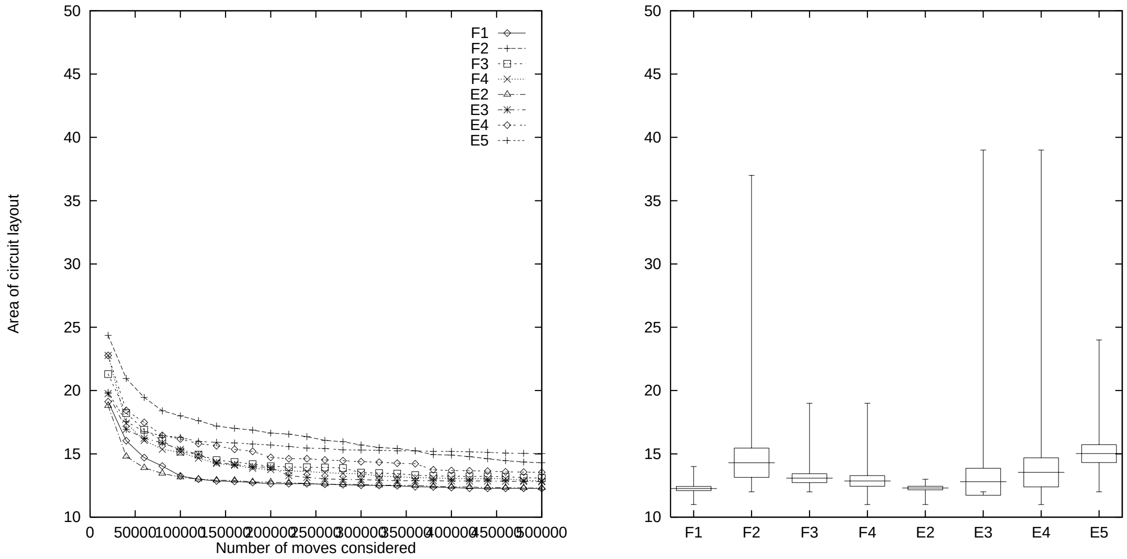

FIGURE 5.9. Performance of different function approximators on channel routing instance YK4

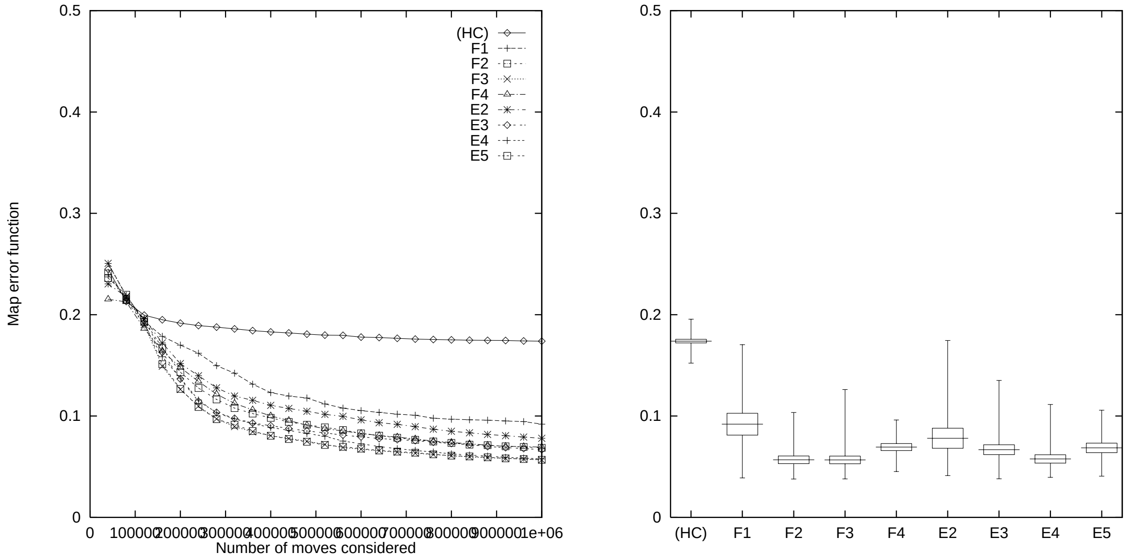

FIGURE 5.10. Cartogram performance with different polynomial function approximators

5.2.3 Policy π

Thus far, I have discussed how the user's choice of features and fitters affects STAGE's ability to learn and exploit $\tilde{V}^{\pi}(F(x))$. But what about the choice of π itself? In every domain of Chapter 4 besides Boolean satisfiability, π was chosen to be a very simple search procedure: stochastic first-improvement hillclimbing, rejecting equi-cost moves. But clearly, stronger local search procedures are available. For example, stochastic hillclimbing *accepting* equi-cost moves often performs better, and simulated annealing performs better still. Can STAGE learn to improve upon these, and achieve yet better performance?

Theoretically, even if a procedure π_1 is significantly better than another procedure π_2, it is by no means guaranteed that STAGE on π_1 will outperform STAGE on π_2. For example, π_2 may produce more diverse outcomes as a function of starting state, enabling useful extrapolation in $\tilde{V}^{\pi_2}$; whereas if π_1 is expected to reach the same solution quality no matter what state it starts from—as was the case with the Boggle example of Section 5.1.4—then $\tilde{V}^{\pi_1}$ will be flat, and STAGE will not produce improvement.

In this section, I consider the following three policies on the cartogram domain:

- π_1: regular first-improvement hillclimbing, rejecting equi-cost moves, patience=200.

- π_2: hillclimbing as above, but modified so that equi-cost and some slightly harmful moves are accepted. Specifically, a move is rejected if and only if it worsens Obj by more than δ=0.0001. (This value of δ was the most effective of a wide range tried on this domain.) On average, π_2 performs significantly better than π_1.

- π_3: simulated annealing with a shortened schedule length of 20,000 moves. The short schedule allows it to be used in the context of a multi-restart procedure.

For each of these three policies, I sought to compare the default restarting method to a "smart" restarting method learned by STAGE. In the cases of π_1 and π_2, the default restarting method is to reset to the domain's initial state (the original undeformed U.S. map). In the case of π_3, I found that a better default restarting method is to start each new annealing schedule in the same state where the previous trajectory finished. These procedures are labelled PI1, PI2 and PI3 in the results presented below.

In applying STAGE to π_2 and π_3, a complication arises: neither of these procedures is Markovian. Methods for coping with this theoretical difficulty were discussed in detail at the end of Section 3.4.1 (p. 61), and in the context of WALKSAT in Section 4.7.2. In the case of π_2, a simple fix is to train $V^{\pi_2}(x)$ on only those states which are the best-so-far on their trajectory. (Note that on the pure hillclimbing trajectories generated by π_1, every state is the best-so-far.) These procedures are labelled STAGE(PI1) and STAGE(PI2) in the results. In the case of π_3, even training only on best-so-far states does not make $V^{\pi_3}(x)$ theoretically well-defined (though results for STAGE(PI3) are given below nonetheless). Instead, giving up on the Markov property, I resort to training V^{π_3} on just the single starting state of each simulated annealing trajectory. This variant of STAGE is well-defined for any proper policy π; its results for all three of our policies are shown as STAGE0(PI1), STAGE0(PI2), and STAGE0(PI3).

The results, displayed in Table 5.9 and Figures 5.11–5.13, point to several conclusions. First, as demonstrated by Experiment STAGE(PI2), STAGE can successfully learn from a non-Markovian policy by training only on best-so-far states. Second, from Experiment STAGE(PI3) we can conclude that STAGE may not be effective at learning evaluation functions from simulated annealing; this is unsurprising, since SA's initial period of random search makes the outcome quite unpredictable from the starting state. Third, from the set of STAGE0 experiments, it is apparent that STAGE learns much more quickly when it is able to train on entire trajectories, rather than just starting states. Thus, STAGE is able to exploit the Markov property of V^{π} for efficient performance.

Instance	Algorithm	Performance (50 runs each)		
		mean	best	worst
US49	PI1 = hillclimbing	0.174±0.002	0.152	0.192
	STAGE(PI1)	**0.057**±0.004	**0.038**	**0.103**
	STAGE0(PI1)	0.099±0.013	0.042	0.174
	PI2 = hillclimbing/δ=0.0001	0.140±0.003	0.115	0.167
	STAGE(PI2)	**0.052**±0.003	**0.040**	**0.083**
	STAGE0(PI2)	0.077±0.009	0.045	0.136
	PI3 = simulated annealing	**0.049**±0.001	0.044	**0.070**
	STAGE(PI3)	**0.050**±0.002	**0.042**	0.091
	STAGE0(PI3)	0.060±0.005	0.043	0.142

TABLE 5.9. Cartogram performance with STAGE learning $\tilde{V}^{\pi}$ from a variety of different choices of policy π. All algorithms were limited to considering TOTEVALS = 10^6 moves.

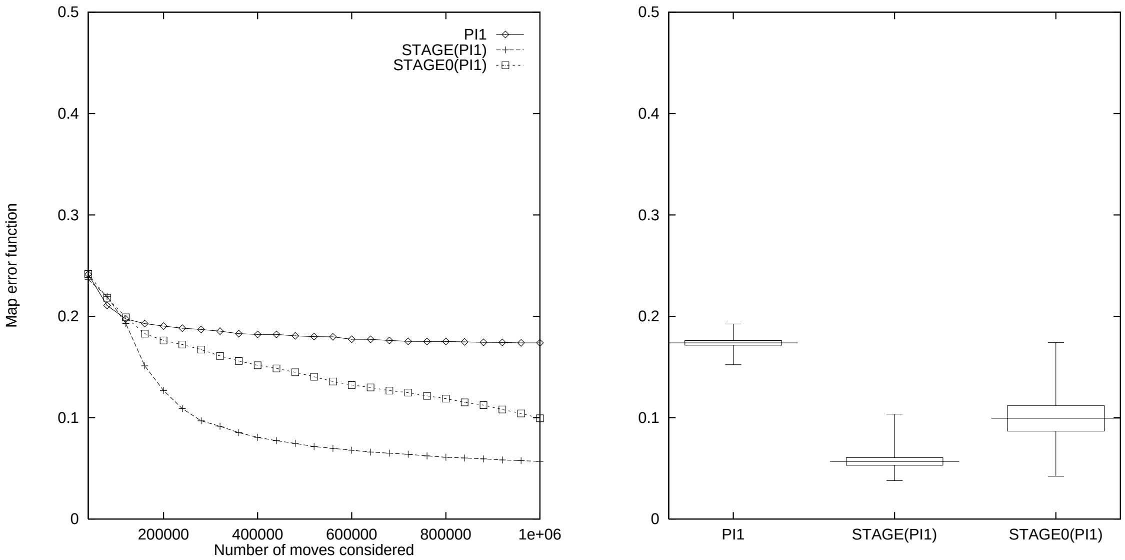

FIGURE 5.11. Cartogram performance of STAGE using π_1 = hillclimbing

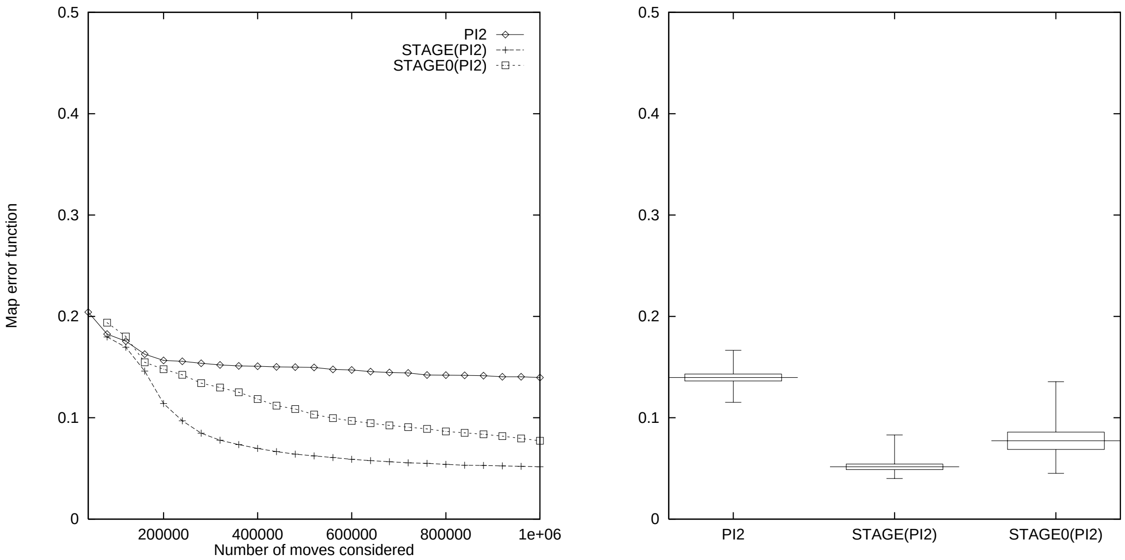

FIGURE 5.12. Cartogram performance of STAGE using π_2 = hillclimbing accepting equi-cost and slightly harmful moves

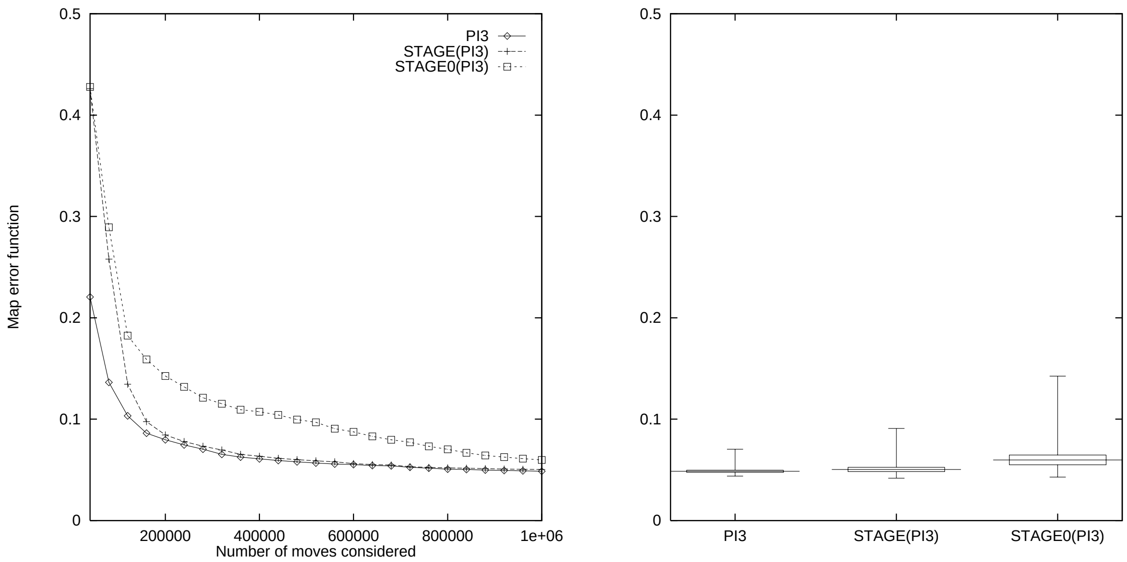

FIGURE 5.13. Cartogram performance of STAGE using π_3 = simulated annealing

5.2.4 Exploration/Exploitation

In any time-bounded search process there arises a tradeoff between *exploitation*, i.e., pursuing what appears to be the best path given the limited observations made thus far, and *exploration*, i.e., trying paths about which little is yet known. Treating this tradeoff optimally is in general intractable, and various heuristics have been proposed in the reinforcement learning literature [Thrun 92, Moore and Atkeson 93, Dearden *et al.* 98]. For its exploration, STAGE relies primarily on extrapolation by the function approximator to guide search to unvisited regions, and secondarily on random restarts when search stalls.

However, there is another aspect of STAGE's search strategy than impinges on the exploration/exploitation dilemma. On each iteration, STAGE runs π to generate a trajectory $(x_0 \ldots x_T)$, uses this trajectory to update $\tilde{V}^\pi$, and then searches for a good starting state of π by hillclimbing on the new $\tilde{V}^\pi$. This section investigates the design decision of where to begin each search of $\tilde{V}^\pi$:

Continue: This is STAGE's normal policy—namely, to begin each search of $\tilde{V}^\pi$ by simply continuing from the current local optimum x_T.

Re-init: Begin each search of $\tilde{V}^\pi$ from a random initial state. This promotes greater global *exploration* of the space, at the cost of losing the benefit of the work just done by π to reach a local optimum of Obj.

Compromises between **Continue** and **Re-init** are possible. Let **Re-init**(K) denote the policy of jumping to a random initial state if and only if K consecutive STAGE iterations have failed to improve on the best optimum yet seen. Thus, **Re-init**(0) always starts from a random state, and **Re-init**(∞) is the same as the **Continue** policy. I tested a range of settings for K.

Best-ever: Begin each search of $\tilde{V}^\pi$ from the best state seen so far on this run. This promotes greater *exploitation* of a known excellent solution, at the cost of possibly becoming over-focused on one area of search space.

Again, compromises are possible. In this case, I implemented **Best-ever**(p), which on each iteration either jumps to the best-so-far state with probability p or continues from the current state x_T with probability $1 - p$. Note that **Best-ever**(0), like **Re-init**(∞), is equivalent to **Continue**.

I compared **Continue**, **Re-init**(0), **Re-init**(5), **Re-init**(20), **Best-ever**(0.1), **Best-ever**(0.4), **Best-ever**(0.7), and **Best-ever**(1) on cartogram domain US49. The results are given in Table 5.10 and Figure 5.10. The results show that frequent jumping to a random state, as **Re-init**(K) does for small K, does negatively affect solution quality, whereas frequent jumping to the best-so-far state, as done by **Best-ever**(p), neither helps performance nor hurts it significantly. STAGE's **Continue** policy seems to strike an acceptable balance between exploration and exploitation.

Instance	Algorithm	Performance (50 runs each)		
		mean	best	worst
US49	**Continue** (normal STAGE)	**0.057**±0.003	**0.037**	0.103
	Re-init(0)	0.171±0.003	0.150	0.197
	Re-init(2)	0.072±0.004	0.045	0.112
	Re-init(5)	0.059±0.004	0.040	0.109
	Re-init(20)	**0.056**±0.005	0.037	0.114
	Best-ever(0.1)	**0.056**±0.004	0.037	**0.100**
	Best-ever(0.4)	0.062±0.006	0.038	0.147
	Best-ever(0.7)	0.060±0.005	0.041	0.129
	Best-ever(1.0)	0.062±0.004	0.039	0.113

TABLE 5.10. Cartogram performance under various modifications of STAGE's policy for initializing search of $\tilde{V}^\pi$ on each round

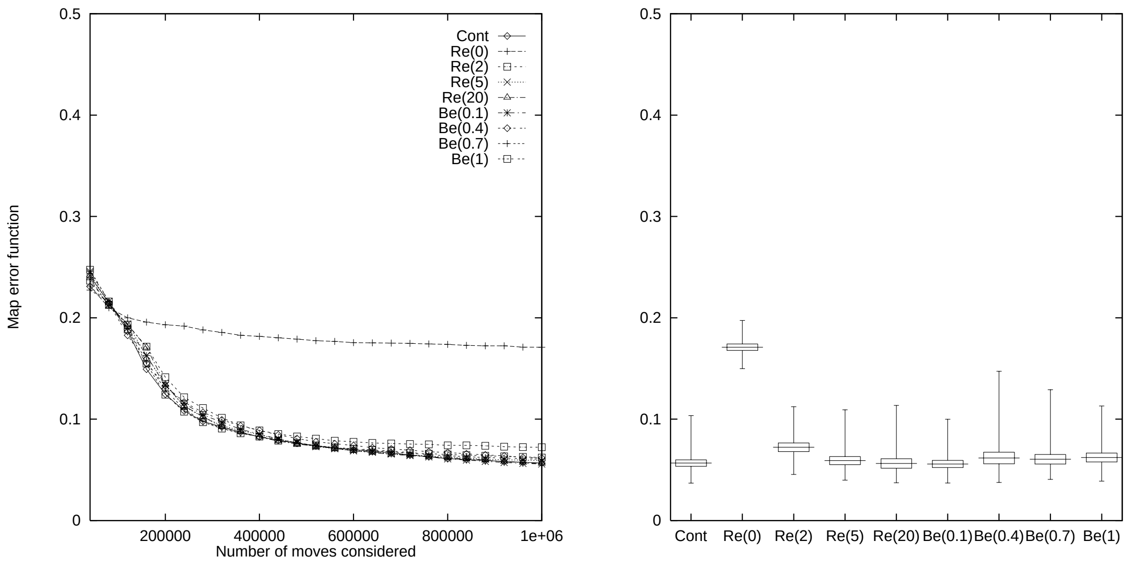

FIGURE 5.14. Cartogram performance under various modifications of STAGE's policy for initializing search of $\tilde{V}^{\pi}$ on each round

5.2.5 Patience and ObjBound

Finally, I investigated the effect of the two remaining STAGE parameters: PAT and OBJBOUND. Recall from the STAGE algorithm (p. 52) that these parameters jointly determine when STAGE's second phase of search, stochastic hillclimbing on $\tilde{V}^{\pi}$, should terminate. OBJBOUND corresponds to a known lower bound on the objective function; it cuts the search off when $\tilde{V}^{\pi}(F(x))$ predicts that x is an impossibly good starting state, i.e., $\tilde{V}^{\pi}(F(x)) <$ OBJBOUND. This cutoff may be disabled by setting OBJBOUND $= -\infty$. The other parameter, PAT, cuts the search off when too many consecutive moves have failed to improve $\tilde{V}^{\pi}$. If either of these parameters is set too aggressively (too tight a bound, too-low patience), then STAGE will fail to reach the best starting points predicted by its function approximator. If, on the other hand, these are set too loosely, then STAGE may waste valuable time that could instead have been spent on the first phase of search, namely, running π and gathering new training data. How sensitive is STAGE to the precise settings used?

Experimental results of varying these parameters on the cartogram domain are presented in Table 5.11 and Figures 5.15 and 5.16. STAGE's other parameters were set as in the cartogram experiments of Section 4.6 (p. 95). The results show that STAGE's performance suffered when PAT was very low or very high, but worked well for a wide range of settings between 64 and 1024. As for the OBJBOUND parameter, STAGE performed best when it was set to its true bound of 0, but degraded gracefully when it was set to -1 (too loose a bound) or $+0.1$ (too tight). Performance was, however, significantly worse for ObjBound=∞, demonstrating that the ObjBound parameter is useful. By cutting off search on $\tilde{V}^\pi$ when the function approximator is provably inaccurate, ObjBound saves computation time and prevents search from wandering too far astray.

Instance	Algorithm	Performance (50 runs each)		
		mean	best	worst
US49	Pat=16	0.161±0.015	0.079	0.285
	Pat=32	0.095±0.008	0.040	0.147
	Pat=64	0.065±0.006	0.040	0.151
	Pat=128	**0.057**±0.004	0.039	**0.114**
	Pat=256	**0.059**±0.007	**0.038**	0.190
	Pat=512	0.062±0.006	0.041	0.134
	Pat=1024	0.070±0.005	0.044	0.126
	Pat=2048	0.100±0.006	0.046	0.159
	Pat=4096	0.126±0.008	0.074	0.187
	Pat=8192	0.122±0.007	0.057	0.172
	Pat=16384	0.132±0.006	0.068	0.178
	Pat=32768	0.129±0.007	0.079	0.189
	Pat=65536	0.135±0.007	0.081	0.184
	ObjBound=$-\infty$	0.083±0.011	**0.036**	0.189
	ObjBound=-1	0.060±0.006	0.037	0.143
	ObjBound=0	**0.057**±0.004	0.038	**0.103**
	ObjBound=0.1	0.061±0.006	0.040	0.171

Table 5.11. Cartogram performance with varying settings of Pat and ObjBound

5.3 Discussion

This chapter gives depth to the results of the previous chapter, demonstrating that STAGE does indeed obtain its success by exploiting the power of reinforcement learning, and that it works reliably over a wide range of parameter settings. The chapter's empirical conclusions may be summarized as follows:

- Evaluation functions other than STAGE's learned $\tilde{V}^\pi$, built at random or by simply smoothing the objective function, perform much worse than $\tilde{V}^\pi$ in the context of multi-start optimization. **STAGE successfully learns and exploits the predictive power of value functions.**

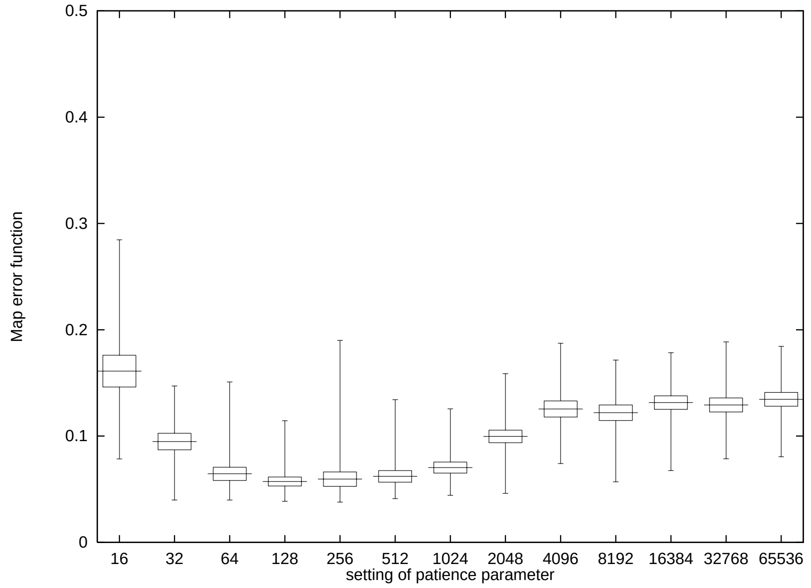

FIGURE 5.15. Cartogram performance with various patience settings

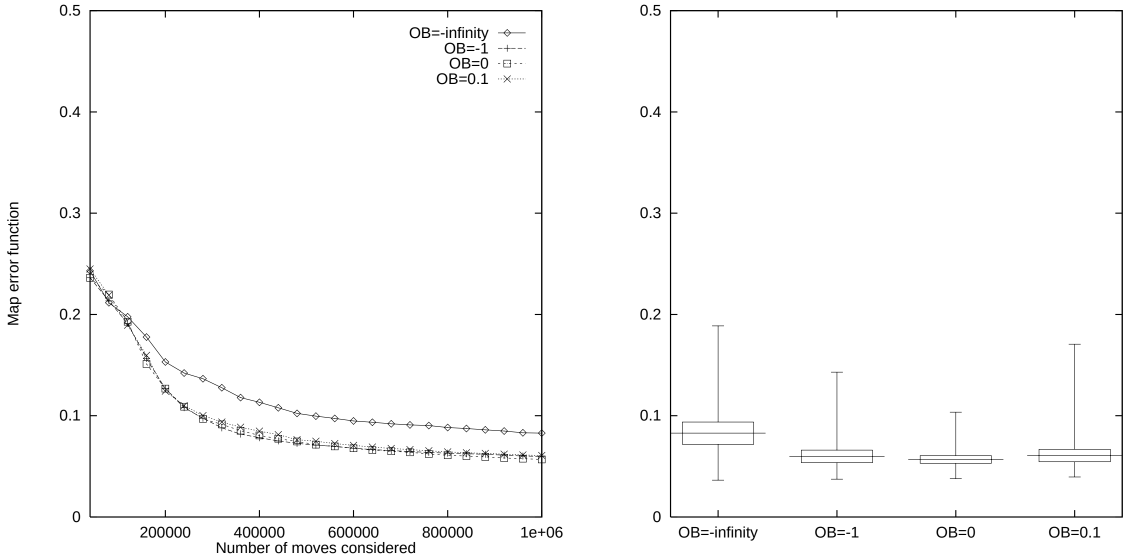

FIGURE 5.16. Cartogram performance with different OBJBOUND levels

- When the baseline policy π does not demonstrate predictable trends as a function of state features, then $\tilde{V}^\pi$ will not be able to guide STAGE to promising new starting states, so STAGE will not improve (nor hurt) performance compared to multi-start π. This was demonstrated for hillclimbing in the Boggle domain and for simulated annealing in the cartogram domain.

- For the policy of π=stochastic hillclimbing, STAGE can empirically learn a useful global structure from many different natural choices of feature sets. STAGE is robust in the presence of irrelevant features, though it works most efficiently and effectively with small feature sets. As a choice of function approximator, quadratic regression provides sufficient model flexibility, enough bias to enable aggressive extrapolation, and efficient computational performance.

- Empirically, STAGE makes reasonable tradeoffs between exploration and exploitation, and it performs robustly over a wide range of settings for the parameters Pat and ObjBound.