Chapter 7
Related Work

Heuristic methods for global optimization have assuredly *not* been overlooked in the literature. Their practical applications are important and numerous, and the literature is correspondingly immense. There is also a significant, if not quite as overwhelming, literature on learning evaluation functions for heuristic search in game-playing and problem-solving. In this chapter, I review the prior work most relevant to STAGE from both these literatures. The reviewed topics are organized as follows:

- §7.1: adaptive multi-restart techniques for local search;
- §7.2: reinforcement learning for combinatorial optimization, especially the study of Zhang and Dietterich [95];
- §7.3: simulation-based methods for improving AI search in game-playing and problem-solving domains, including techniques based on "rollouts" and on learning evaluation functions; and
- §7.4: genetic algorithms.

I defer a high-level discussion of STAGE's novel contributions to the next, concluding chapter.

7.1 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 *et al.* 96], outperformed simulated annealing on the Traveling Salesman Problem (TSP) [Johnson and McGeoch 95], and outperformed genetic algorithms and genetic programming on several large-scale testbeds [Juels and Wattenberg 96].

Nevertheless, the effectiveness of random multi-start local search is limited in many cases by a "central limit catastrophe" [Boese *et al.* 94]: random local optima in large problems tend to all have average quality, with little variance [Martin and Otto 94]. 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 [94], 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 recently been conducted by Boese [95, 96]. 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 7.1.

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 3.4 (p. 48), which I contrived as an example of the kind of problem at which STAGE would excel. Working from the assumption of the big valley structure, Boese recommended the following two-phase adaptive multi-start methodology for optimization:

Phase One: Generate R random starting solutions and run Greedy_Descent from each to determine a set of corresponding *random local minima.*

Phase Two: Based on the local minima obtained so far, construct *adaptive* starting solutions and run Greedy_Descent A times from each one to yield corresponding *adaptive local minima.*

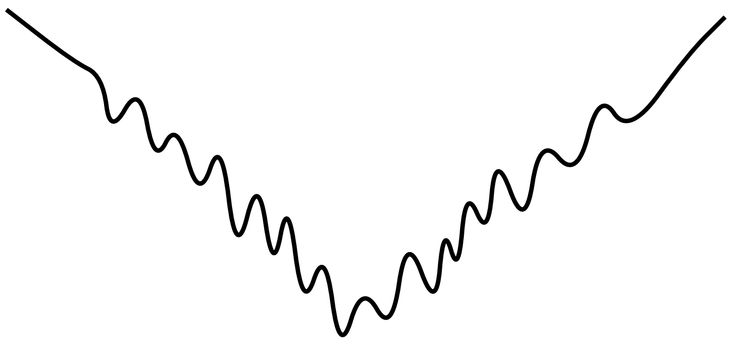

FIGURE 7.1. Intuitive picture of the "big valley" solution space structure. (Adapted from [Boese 95].)

> Intuitively, the two phases respectively develop, then exploit, a structural picture of the cost surface. [Boese *et al.* 94]

At this level of description, the two phases of Boese's recommended search regime correspond almost exactly with the two alternating phases of 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. Another difference is that STAGE's learned heuristic for constructing starting states is based on full hillclimbing trajectories, not just the local minima obtained so far.

A similar methodology underlies the current best heuristic for solving large Traveling Salesman problems, "Chained Local Optimization" (CLO) [Martin and Otto 94]. 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 I discuss in Section 7.4 below.) In the TSP, the kick designed by Martin and Otto [94] 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 and McGeoch 95]. 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 my 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 [95] 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 and Laguna 93] 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 and Kahng 97]; 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 *et al.* 96, Jagota 96]. 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.

7.2 Reinforcement Learning for Optimization

Value function approximation has previously been applied to a large-scale combinatorial optimization task: the Space Shuttle Payload Processing domain [Zhang and Dietterich 95, Zhang 96]. I will discuss this application in some detail, because both the similarities and differences to STAGE are instructive. Please refer back to Section 2.1 for details on the algorithms and notation of value function approximation.

The Space Shuttle Payload Processing (SSPP) domain is a form of *job-shop scheduling*: given a partially ordered set of jobs and the resources required by each, assign them start times so as to respect the partial order, meet constraints on simultaneous resource usage, and minimize the total execution time. Following the "repair-based scheduling" paradigm of Zweben [94], Zhang and Dietterich defined a search over the space of fully specified schedules that meet the ordering constraints but not necessarily the resource constraints. Search begins at a fixed start state, the "critical path schedule," and proceeds by the application of two types of deterministic operators: REASSIGN-POOL and MOVE. To keep the number of instantiated operators small, they considered only operations which would repair the schedule's earliest constraint violation. Search terminates as soon as a violation-free schedule is reached.

Zhang and Dietterich applied reinforcement learning to this domain in order to obtain *transfer*: their goal was to learn a value function which captured knowledge about not a single instance of a scheduling problem, but rather a large family of related scheduling instances. Thus, the input to their value function approximator (a neural network) consisted of abstract instance-independent features, such as the percentage of the schedule containing a violation and the mean and standard deviation of certain slack times. Likewise, the ultimate measure of schedule quality which the neural network was learning to predict had to be normalized so that it spanned the same range regardless of problem difficulty. (The voting-based approach to transfer introduced in Section 6.2.2 of this thesis allows such normalization to be avoided.) Trained on many scheduling instances, the neural network could then be applied to guide search on a novel scheduling instance, hopefully producing a good solution quickly.

Unlike STAGE, which seeks to learn V^π, the predicted outcome of a prespecified optimization policy π—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 even defined, since optimization policies face two conflicting objectives: "produce good solutions" and "finish quickly." In the SSPP domain, Zhang and Dietterich measured the total cost of a search trajectory $(x_0, x_1, \dots, x_N, \text{END})$ by

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

Effectively, since $\text{Obj}(x)$ is near 1.0 in this problem, this cost function means that a 1% improvement in final solution quality is worth about 10 extra search steps [Zhang and Dietterich 98]. The goal of learning, then, was to produce a policy π^* to optimize this balance between trajectory length and final solution quality.

Following Tesauro's methodology for learning V^* on backgammon, Zhang and Dietterich applied optimistic TD(λ) to the SSPP domain. Rather than training only a single neural network, they trained a pool of 8–12 networks, using different parameter settings for each. They trained the networks on a set of small problem instances. Training continued until performance stopped improving on a validation set of other problem instances. Then, the N best-performing networks were saved and used in a round-robin fashion for comparisons against Zweben's iterative-repair system [Zweben *et al.* 94], the previously best scheduler. The results showed that searches with the learned evaluation functions produced schedules as good as Zweben's in less than half the CPU time.

Getting these good results required substantial tuning. One complication involves state-space cycles. Since the move operators are deterministic, a learned policy may easily enter an infinite loop, which makes its value function undefined. Loops are fairly infrequent in the SSPP domain 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 and Dietterich 98]—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 manages to finesse three other algorithmic complications which Zhang and Dietterich found it necessary to introduce: experience replay [Lin 93], 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. 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 which 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 that it holds out the potential of identifying a truly optimal or near-optimal policy π^*. STAGE can only claim to learn an improvement over the prespecified policy π.

Another published work, the "Ant-Q" system [Dorigo and Gambardella 95], is also billed as a reinforcement learning approach to combinatorial optimization. Based on an extensive metaphor with the behavior of ant colonies, it has been applied only to the TSP; it is unclear how it would be applied to other optimization domains.

7.3 Rollouts and Learning for AI Search

Ordinary hillclimbing, simulated annealing and genetic algorithms evaluate the quality of each neighboring state x' by its objective function value $\text{Obj}(x')$. STAGE evaluates x' by how promising its features make it appear, $\tilde{V}^\pi(F(x'))$. A third possibility is to evaluate x' by performing one or more actual sample runs of hillclimbing starting from x'. This is the principle of so-called "rollout algorithms" [Bertsekas *et al.* 97, Tesauro and Galperin 97], a term borrowed from the backgammon-strategy literature [Woolsey 91]. Like STAGE, rollout methods evaluate each neighbor based upon its long-term promise as a starting state for a fixed policy π. Selecting actions in this way is tantamount to performing a single round of the policy iteration algorithm [Howard 60] and can be guaranteed to improve upon π [Bertsekas *et al.* 97].

Unlike STAGE, rollout methods do not cache the results of their lookahead searches in a function approximator $\tilde{V}^\pi$. In this way they avoid the biases that feature representations and function approximation introduce, perhaps allowing more accurate moves. However, the computational cost of replacing function approximator evaluations by full sample runs is extremely high. This cost may be tolerable in a game-playing scenario where several seconds are available for each move selection; but would probably be deadly for practical optimization algorithms, where (in the words of Buntine [97]) a general maxim applies: "speed over smarts."

In fact, an earlier study of Abramson [90] demonstrated the power of rollout methods for game-playing, although he was apparently unaware of the connection to Markov decision processes and policy iteration. In his "expected-outcome" model, moves in the game of Othello were made by computing, via multiple rollouts for each legal move, the expected probability of winning *assuming both sides play randomly from that point on.* Though this assumption is clearly inaccurate, Abramson reported empirical good play at Othello:

> Given no expert information, the ability to evaluate only leaves, and a good deal of computation time, [expected-outcome functions] were able to play better than a competent (albeit non-masterful) function that had been handcrafted by an expert. [Abramson 90]

This improvement is consistent with the experience of Tesauro and Galperin [97], who found that a rollout-based player (implemented on a parallel IBM supercomputer) dramatically improved the performance of both weak and strong evaluation functions.

Abramson [90] went on to propose and test learning the expected-outcome function with linear regression over state features, just as STAGE does. Reinterpreted in the language of reinforcement learning, Abramson's method fixed a policy π (both players play randomly), inducing a Markov chain on the game space. He collected samples of the policy value function $V^{\pi}(x)$ by running π multiple times from each of 5000 random starting states; trained a linear predictor from the samples; and finally, used the predictor as a static evaluation function $\tilde{V}^{\pi}(F(x))$ to select moves. STAGE can be seen as extending Abramson's work from game-playing into the realm of combinatorial optimization. A key difference is that STAGE uses a high-quality baseline policy π, such as hillclimbing or WALKSAT, in place of Abramson's random policy. STAGE also interleaves its training and decision-making phases, which enables it to adapt its training distribution over time to focus on high-quality states.

Abramson's work is one of many examples in the Artificial Intelligence literature of learning evaluation functions for game-playing. Samuel's pioneering work on checkers [Samuel 59, Samuel 67], Christensen's work on chess [Christensen 86], and Lee's work on Othello [Lee and Mahajan 88] fall into this category; I have already discussed these in Section 2.1.2 of this thesis. More recently, the successful application of reinforcement learning to backgammon [Tesauro 92, Boyan 92] has inspired similar investigations in chess [Thrun 95, Baxter *et al.* 97], Go [Schraudolph *et al.* 94], and other games.

Moving from game-playing to problem-solving domains, a study by Rendell [83] addressed evaluation function learning in the sliding-tiles puzzle (15-puzzle). His method, when abstracted of many details, bears significant similarities to STAGE: it learns to approximate a function which measures the promise of each state as a starting state for a given policy. In particular, the policy π is a *best-first search* (with backtracking) guided by a given evaluation function $f(x)$; and the measurement being approximated is not a value function but a so-called "penetrance" measure at each state, as defined by Doran and Michie [66]. In STAGE-like notation, the penetrance is defined as

$$P^{\pi}(x) \stackrel{\text{def}}{=} \frac{\text{length of discovered path from } x \text{ to solution}}{\text{total nodes expanded by } \pi \text{ during search from } x}$$

For example, in the 15-puzzle, given a constant evaluation function $f(x) \equiv 1$, the policy π reduces to breadth-first search, and $P^{\pi}(x)$ is on the order of 10^{-9} for random starting states x; whereas a perfect evaluation function $f(x) \equiv V^{*}(x)$ would give rise to a backtracking-free policy π with penetrance $P^{\pi}(x) = 1$ everywhere. Rendell fits a linear approximation to the penetrance, then applies the fit to improve search control using a complex bootstrapping and normalization procedure.

7.4 Genetic Algorithms

Genetic algorithms (GAs)—algorithms based on metaphors of biological evolution such as natural selection, mutation, and recombination—represent another heuristic approach to combinatorial optimization [Goldberg 89]. Translated into the terminology of local search, "natural selection" means rejecting high-cost states in favor of low-cost states, like hillclimbing; "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 7.1. 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 to STAGE, where features can be any real-valued vector function 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.

Of course, the effectiveness of *any* local search algorithm depends on the neighborhood operators available; but genetic algorithms generally allow less flexibility in designing the neighborhood, since mutations are represented as bit-flips. Hillclimbing and simulated annealing, by contrast, allow sophisticated, domain-specific search operators, such as the partition-graph manipulations used in simulated annealing applications of VLSI channel routing [Wong *et al.* 88]. 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 and Nowlan 87]. 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 and McGeoch 95]. Here, the GA operators play a role analogous to the large-step "kick moves" of Chained Local Optimization [Martin and Otto 94], as described in Section 7.1 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 and Littman 93, Johnson and McGeoch 95].

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 [97], a descendant of PBIL [Baluja and Caruana 95] and MIMIC [de Bonet *et al.* 97], 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 (see page 163) indicate that there is often useful information to be gained by modelling the weaker areas of the solution space, too [Boese *et al.* 94]. 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 4.7, COMIT (using WALKSAT as a subroutine) did not perform as well as STAGE [Davies and Baluja 98].

7.5 Discussion

The studies described in the last four sections make it clear that STAGE bears close relationships to previous work done in the optimization, reinforcement learning, and AI problem-solving communities. A concise summary of this chapter might read as follows:

> STAGE takes its main idea—learn an evaluation function to improve search performance—from the AI literature on problem-solving. It grounds that idea in the theory of value function approximation. And it applies that idea in the successful framework of adaptive multi-start approaches to global optimization.

STAGE unifies these lines of research in an algorithm which is nonetheless quite simple to explain and to implement. In the next, concluding chapter, I will summarize the novel contributions made by STAGE and suggest a number of future directions for integrating reinforcement learning into effective optimization algorithms.