Statistical Machine Learning for Large-Scale Optimization

Contributors

S. Baluja,	A.G. Barto,	K.D. Boese,	J. Boyan,	W. Buntine,	T. Carson
R. Caruana,	D.J. Cook,	S. Davies,	T. Dean,	T.G. Dietterich,	P.J. Gmytrasiewicz
S. Hazlehurst,	R. Impagliazzo,	A.K. Jagota,	K.E. Kim,	A. McGovern,	R. Moll
A.W. Moore,	E. Moss,	M. Mullin,	A.R. Newton,	B.S. Peters,	T.J. Perkins
L. Sanchis,	L. Su,	C. Tseng,	K. Tumer,	X. Wang,	D.H. Wolpert

Editors Justin Boyan, Wray Buntine, and Arun Jagota

Contents

Introduction

Large-scale global optimization problems arise in all fields of science, engineering, and business; and exact solution algorithms are available all too infrequently. Thus, there has been a great deal of work on general-purpose heuristic methods for finding approximate optima, including such iterative techniques as hillclimbing, simulated annealing, and genetic algorithms (e.g., [4]). Despite their lack of theoretical guarantees, these techniques are popular because they are simple to implement and often perform well in practice.

Recently, there has been a surge of interest in analyzing and improving these heuristic algorithms with the tools of statistical machine learning. Statistical methods, working from the data generated by heuristic search trials, can discover relationships between the search space and the objective function that the current techniques ignore, but that may be profitably exploited in future trials. Research questions include the following:

- Can one learn a pattern about local minima from which one could locate superior local minima more efficiently than by simple repeated trials?
- Can multiple heuristics be combined on the fly, or perhaps by pre-computation?
- Is the outcome of a search trajectory predictable in advance, and if so, how can such predictions be learned and exploited?
- Can effective high-level search moves be learned automatically?
- Does the problem have a natural clustering or hierarchy that enables the search space to be scaled down?
- Can the statistical models built in the course of solving one problem instance be profitably transferred to related, new instances?

These questions are starting to be answered affirmatively by researchers from a variety of communities, including reinforcement learning, decision theory, Bayesian learning, connectionism, genetic algorithms, satisfiability, response surface methodology, and computer-aided design. In this survey, we bring together short summaries of 14 recent studies that engage these questions.

The 14 studies overlap in many ways, but perhaps are best categorized according to the *goal* of their statistical learning. We consider each of the following goals of learning in turn: (1) understanding search spaces; (2) algorithm selection and tuning; (3) learning generative models of solutions; and (4) learning evaluation functions.

Understanding search spaces

Statistical analyses of the search spaces that arise in optimization problems have produced remarkable insights into the global structure of those problems. The analyses give essential guidance to those who would design algorithms to exploit such structure. Our survey includes three abstracts in this category:

- Boese defines the "central limit catastrophe" of multi-start optimization, illustrates the "big valley" cost surface that empirically describes many large-scale optimization problems, and outlines a number of promising research directions.
- Caruana and Mullin introduce a probabilistic method for counting the local optima in a large search space, with application to improving the cutoff criteria in genetic algorithms and simulated annealing.
- Carson and Impagliazzo introduce the property of "local expansion" of a search graph, show how to test for that property in large-scale domains, and use the test to predict how easy or difficult an optimization instance will be for a given heuristic.

Algorithm selection and tuning

A natural yet under-investigated approach to accelerating optimization performance is to apply machine learning to tune the optimizer's parameters automatically. Such parameters may include domain-specific terms, such as the coefficients of extra objective-function terms; generic parameters of the heuristic, such as the cooling-rate schedule in simulated annealing; and even high-level discrete parameters, such as which of a set of heuristics to apply. From sample optimization runs, a mapping from parameters to expected performance can be learned. This mapping can then itself be "meta-optimized" to generate the best set of parameters for a family of problems.

Two abstracts in our survey fall into this category:

- Cook, Gmytrasiewicz, and Tseng apply machine learning to the task of automatically selecting the best heuristic for use by EUREKA, their parallel search architecture, on a given problem instance. They compare decision-tree and Bayes-network learning methods.
- Jagota and Sanchis describe several heuristics for the NP-hard Maximum-Clique problem. The heuristics are parameterized by an initial state and/or a weight vector, which adapt from iteration to iteration depending on their effect on optimization performance.

Learning generative models of solutions

Boese's "big valley" hypothesis indicates that in practical problems, high-quality local optima tend to be "centrally located" among the local optima in the search space. This suggests an adaptive strategy of collecting the best local optima found during search and training a model of those solutions. If the model is *generative*, it can be called upon to generate new, previously untried solutions similar to the good solutions on which it was trained. This survey includes two relevant abstracts:

- Baluja and Davies point out that implicitly, genetic algorithms do precisely this sort of modeling: the "population" stores good solutions that have already been found, and the mutation and recombination operators generate new, similar solutions. Their abstract summarizes three algorithms that make the genetic algorithm's modeling function *explicit*, consequently improving optimization performance.
- Buntine, Su and Newton learn a generative model in the problem of hyper-graph partitioning, crucial in VLSI design [3]. The model is in the form of a *clustering* of the graph nodes, based on a statistical analysis of the best solutions found so far in the search. The clustering effectively scales down the size of the search space, enabling good new candidate solutions to be generated very quickly.

Learning evaluation functions

Finally, the fourth and most active category of research covers *learning evaluation functions*. An evaluation function is a mapping from domain solutions to real numbers—the same form as the objective function itself. And just as the objective function is used to guide search through the state space, so may any other evaluation function be used for that purpose. In fact, there are many ways in which a learned evaluation function might usefully supplement the domain's given objective function:

Evaluation speedup: In cases where the domain objective function is expensive to calculate, a fast approximate model of the objective function could lead search to the vicinity of the optimum with less computation (e.g., [5]).

Move selection: An appropriately built evaluation function could be used in place of the original objective function to guide search. Ideally, such a function would share its global optimum with that of the original objective, but would eliminate the local optima and plateaus that impede search from reaching that goal (e.g., [7]).

Restarting: Iterative algorithms are often run repeatedly, each time starting from an independent random "restart" state. Instead, an evaluation function may be trained to guide search to new states that are promising restart states. Such a function can effectively provide large-step "kick moves" that guide the search out of a local optimum and into a more promising region of space. Generative models may also be used this way.

Move sampling: In domains with many search moves available at each step, it is time-consuming to sample moves at random, hoping for an improvement. Instead, a "state-action" evaluation function (one that estimates the long-term effect of trying a given move in a given state) may be applied to screen out unpromising moves very quickly.

Trajectory filtering: An evaluation function that predicts the long-term outcome of a search trajectory may be employed as a criterion for cutting off an unpromising trajectory and beginning a new one.

Abstraction: Some problems naturally divide into two or more hierarchical levels; e.g., in traditional VLSI design, place-then-route. Although the true objective function is only defined over fully instantiated solutions (at the lowest level), learned evaluation functions can provide an accurate heuristic to guide search at higher levels.

Transfer: Evaluation functions defined over a small set of high-level state-space "features" may readily be *transferred*—i.e., built from a training set of instances, and then applied quickly to novel instances in any of the ways described above.

How can useful evaluation functions be learned automatically, through only trial-and-error simulations of the heuristic? In most cases, what is desired of the evaluation function is that it provide an assessment of the long-range utility of searching from a given state. Tools for exactly this problem are being developed in the reinforcement learning community under the rubric of "value function approximation" [2]. Alternatives to value function approximation include learning from "rollouts" (e.g., [1]) and treating the evaluation function weights as parameters to "meta-optimize" (e.g., [6]), as described above in the section on algorithm tuning.

Our survey includes summaries of five studies on learning evaluation functions for optimization:

- McGovern, Moss, and Barto learn an evaluation function for move selection in the domain of optimizing compiled machine code, comparing a reinforcement-learning-based scheduler with one based on rollouts.
- Boyan and Moore use reinforcement learning to build a secondary evaluation function for smart restarting. Their "STAGE" system alternately guides search with the learned evaluation function and the original objective function.
- Moll, Perkins, and Barto apply an algorithm similar to STAGE to the NP-hard "dial-a-ride" problem (DARP). The learned function is instance-independent, so it applies quickly and effectively to new DARP instances.
- Su, Buntine, Newton, and Peters learn a "state-action" evaluation function that allows efficient move sampling. They report impressive results in the domain of VLSI Standard Cell Placement.
- Wolpert and Tumer give a principled method for decomposing a global objective function into a collection of localized objective functions, for use by independent computational agents. The approach is demonstrated on the domain of packet routing. (Also see Boese's abstract for other results on multi-agent optimization.)

Finally, since the techniques of reinforcement learning are so relevant to this line of research, we include summaries of two contributions that do not deal directly with large-scale optimization, but rather advance the state of the art in large-scale reinforcement learning:

- Wang and Dietterich summarize the types of models that have been used for value function approximation, and introduce a promising new model based on regression trees.
- Dean, Kim, and Hazlehurst describe an innovative, compact representation for large-scale sparse matrix operations, with application to efficient value function approximation.

It is our hope that these 14 summaries, taken together, provide a coherent overview of some of the first steps in applying machine learning to large-scale optimization. Numerous open yet manageable research problems remain unexplored, paving the way for rapid progress in this area. Moreover, the improvements that result from the maturation of this research are not merely of academic interest, but can deliver significant gains to computer-aided design, supply-chain optimization, genomics, drug design, and many other realms of enormous economic and scientific importance.

REFERENCES

[1] D. Bertsekas, J. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. Technical Report LIDS-P 2386, MIT Laboratory for Information and Decision Systems, 1997.

[2] J. A. Boyan, A. W. Moore, and R. S. Sutton, editors. *Proceedings of the Workshop on Value Function Approximation*, Machine Learning Conference, July 1995. CMU-CS-95-206. Internet resource available at `http://www.cs.cmu.edu/~reinf/ml95/`.

[3] L. W. Hagen and A. B. Kahng. Combining problem reduction and adaptive multi-start: A new technique for superior iterative partitioning. IEEE Transactions on CAD, 16(7):709–717, 1997.

[4] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*. Wiley and Sons, 1997. Internet resource available at `http://www.research.att.com/~dsj/papers/TSPchapter.ps`.

[5] A. W. Moore and J. Schneider. Memory-based stochastic optimization. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Neural Information Processing Systems 8*, 1996.

[6] E. Ochotta. *Synthesis of High-Performance Analog Cells in ASTRX/OBLX*. PhD thesis, CMU Electrical and Computer Engineering, 1994.

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

A Review of Iterative Global Optimization

Kenneth D. Boese

Cadence Design Systems, San Jose, USA,

An instance of finite global optimization consists of a finite solution set S and a real-valued cost function $f : S \to \Re$. Global optimization seeks a solution $s^* \in S$ which (without loss of generality) minimizes f. Combinatorial optimizations of this type arise in a wide variety of computational domains such as computer architecture, operations research, computational chemistry and biology, and neural network training. Because many of these optimization problems are NP-hard [8], and probably impossible to solve optimally in polynomial time, heuristic algorithms are necessary for large instances. These heuristics often use an iterative search that is broadly be described by the iterative global optimization (IGO) template below.

Iterative Global Optimization (IGO)

```
1. for $i = 0$ to $+\infty$
2.     Given the current solution $s_i$, generate a new trial solution $s'$
3.     Decide whether to set $s_{i+1} = s_i$ or $s_{i+1} = s'$
4.     if a stopping condition is satisfied
5.         return the best solution found
```

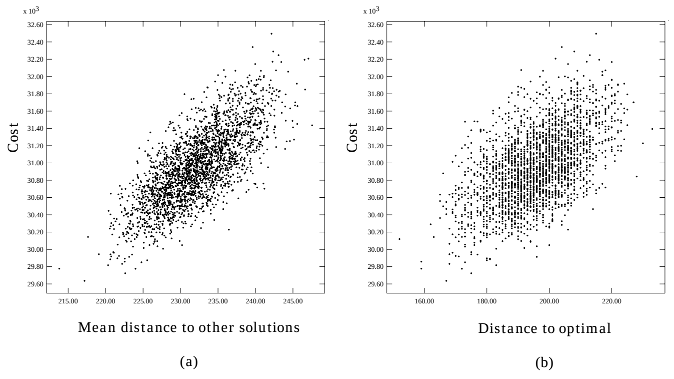
Typically, s' in Line 2 of the template is generated by a well-defined, often randomized, perturbation to s_i, i.e., $s' \in N(s_i)$ where $N(s_i)$ indicates the set of neighboring solutions or *neighborhood*, of s_i. Together with N, the cost function f defines a *cost surface* over the neighborhood topology.

Some important variations of the general IGO framework include 1) greedy descent IGO; 2) hill-climbing IGO; 3) multi-start or multi-agent IGO; and 4) IGO with problem-size reduction. In greedy descent, the candidate solution s' is chosen as s_{i+1} only if it has lower cost than s_i. Sophisticated implementations of greedy descent can quickly search a large neighborhood of s_i for an improving solution. Examples include Bentley's fast implementation of the 3-Opt neighborhood for the traveling salesman problem (TSP) [2] and other more complicated greedy algorithms, such as the Lin-Kernighan algorithm [18] for the TSP and the Kernighan-Lin algorithm [16] for graph partitioning.

The main weakness of greedy descent is that it becomes stuck at any locally minimum solution. IGO variations 2 and 3 are designed to avoid this weakness. Hill-climbing allows some disimproving or "uphill" moves, i.e., $f(s_{i+1}) > f(s_i)$. Some popular algorithms in this class are simulated annealing [17], tabu search [9], threshold acceptance [6], and simulated tempering [19]. Tabu search and simulated tempering are also *adaptive*, i.e., they modify their own parameters during the run based on earlier results of the same run. Another way to avoid getting stuck at local minima is multi-start IGO, which restarts a new greedy descent from a new starting solution s_0 whenever a local minima is encountered. Multi-start IGO is easily parallelizable by executing separate runs on different processors or "agents". Some examples of multi-start heuristics include [20] [5] [11].

The simplest hill-climbing and multi-start implementations may still have trouble finding near-optimal solutions, however. For many NP-Hard problems, as the problem size increases, the number of local minima appears to grow exponentially, and most local minima may be significantly worse than the optimal solution. We have called this phenomenon the "central limit catastrophe" [5] [3], while elsewhere it has been described as the "complexity catastrophe" [13, 14, 15] and the "error catastrophe" [7]. The problem is that if the distribution of the costs of the local minima is approximately Gaussian, and if the number of local minima increases exponentially, then an increasing percentage of the local minima costs will be clustered near the average local minima cost and far from the optimal cost.

One way to avoid the central limit catastrophe is to exploit the structure of the cost surface defined by the neighborhood operator N and cost function f. For example, Figure [3] plots solution cost versus distance to the optimal solution and versus average distance to the other local minima for 2,500 different local minima of the ATT532 TSP test case. It appears from these plots that lower-cost local minima are



2,500 Random 2-Opt local minima for ATT532. Tour cost (vertical axis) is plotted against (a) mean distance to the other local minima and (b) distance to the global minimum.

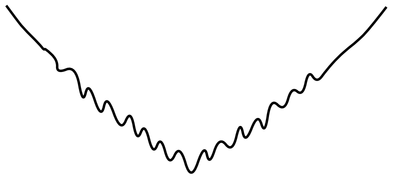


Figure 1: Intuitive picture of a "big valley" cost surface.

located closer to the global minimum and closer to the "center" of the set of local minima. This suggests a "big valley" appearance or structure, as illustrated in Figure 1. One way to exploit the big valley is to run multi-start IGO using starting points s_0 generated an "averaging" of previously found local minima [5]. In general, the goal is to generate an *interaction* between the different multi-start or multi-agent runs in order to exploit the structure of the cost surface.

The fourth IGO variation of problem-size reduction provides another way to avoid the central limit catastrophe. It uses clustering to dramatically reduce the size of the solution space. This approach, combined with multi-start, has been particularly successful for circuit partitioning in VLSI computer circuit design [1] [10] [12].

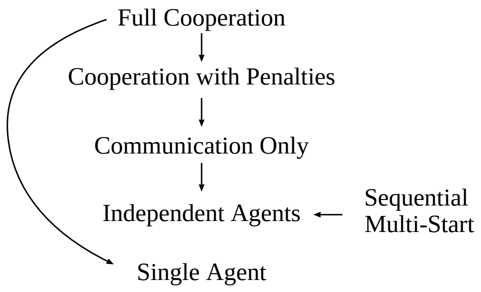


Figure 2: Hierarchy of dominance between different models of adaptive annealing. Each arrow points from a dominating to a dominated model.

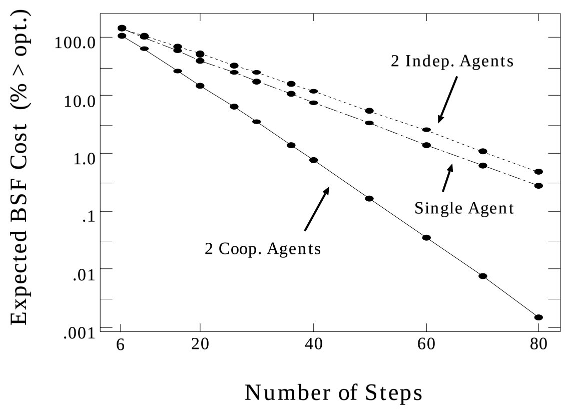


Figure 3: Expected best-so-far qualities of optimal policies on instance 3SAT6.

Finally, we suggest that modeling IGO on small optimization instance can give insights for improving IGO strategies. For example, in [4] and [3] we computed optimal temperature schedules for simulated annealing on small instances of TSP, graph partitioning, circuit placement, and 3-SAT. The schedules we found indicate that temperature schedules should not be monotone decreasing, in contrast to the conventional wisdom for simulated annealing. These results can be used to motivate the non-decreasing schedules used in simulated tempering [19]. In Chapter 9 of [3], we also studied optimal simulated annealing using adaptive schedules and multiple agents. We found that interaction among agents can be potentially very powerful, depending on the level of interaction. Using our models, we can also compared different kinds of interaction, including "communication" (sharing attained costs) and "cooperation" (sharing costs and swapping solutions). Figure 2 shows a partial hierarchy between different multi-agent models in our study. (The quality of a model equals the expected cost of the best solution for a given number of IGO steps divided among the different agents. In general, one regime will dominate another if it can simulate the other one without using any extra IGO steps.) For a small 3-SAT instance, Figure 3 shows the expected solution cost of different regimes using optimal adaptive annealing schedules. From the figure, we note that cooperating agents appear to have a large potential for producing nearly optimal solutions, compared to multiple agents working independently.

In conclusion, we have reviewed some directions for analyzing and extending the iterative global optimization strategy. We believe there are a number of fruitful areas for further research, including the further understanding of cost surfaces structures and the exploitation of interaction between multiple processors or "agents".

References

[1] C. J. Alpert, J.-H. Huang and A. B. Kahng, "Multilevel Circuit Partitioning", in *Proceedings of the 34th Design Automation Conf.*, 530-33 (1997).

[2] Bentley, J.L., "Fast Algorithms for Geometric Traveling Salesman Problems", *ORSA Journal on Computing* **4** (4), 387-411 (Fall 1992).

[3] K. D. Boese, *Models for Iterative Global Optimization*, Ph.D. Thesis, UCLA Computer Science Dept., 1996.

[4] K. D. Boese and A. B. Kahng, "Best-So-Far vs. Where-You-Are: Implications for Optimal Finite-Time Annealing", *Systems and Control Letters* **22 (1)**, 71-78 (1994).

[5] K. D. Boese, A. B. Kahng and S. Muddu, "On the Big Valley and Adaptive Multi-Start for Discrete Global Optimizations", *Operations Research Letters*, **16 (2)**, 101-113 (1994).

[6] G. Dueck and T. Scheuer, "Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing", *J. of Computational Physics*, **90**, 161-75 (1990).

[7] M. Eigen and P. Schuster, *The Hypercycle*, Springer-Verlag, 1979.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[9] F. Glover, "Tabu Search — Part II", *ORSA Journal on Computing*, **2 (1)**, 4-32 (1990).

[10] L. W. Hagen and A. B. Kahng, "Combining Problem Reduction and Adaptive Multi-Start: A New Technique for Superior Iterative Partitioning", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **16 (7)**, 709-17 (1997).

[11] D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization", in E. H. L. Aarts and J. K. Lenstra, eds., *Local Search in Combinatorial Optimization*, Wiley and Sons, 1997.

[12] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", *SIAM Journal on Scientific Computing*, **20 (1)**, 359-92 (1998).

[13] S. Kauffman and S. Levin. "Toward a General Theory of Adaptive Walks on Rugged Landscapes", *Journal of Theoretical Biology* **128**, 11-45 (1987).

[14] S. Kauffman, Adaptation on Rugged Fitness Landscapes, in D. L. Stein, ed., *Lectures in the Sciences of Complexity*, Addison-Wesley, 1989.

[15] S. A. Kauffman, *The Origins of Order : Self-Organization and Selection in Evolution*, Oxford University Press, 1993.

[16] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *The Bell System Technical Journal*, **49**, 291-307 (1970).

[17] S. Kirkpatrick, C. D. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing", *Science*, **220**, 671-680 (1983).

[18] S. Lin and B. W. Kernighan, "An Effective Heuristics Algorithm for the Traveling-Salesman Problem", *Operations Research*, **31**, 498-516 (1973).

[19] E. Marinari and G. Parisi, "Simulated Tempering: a New Monte Carlo Scheme" *Europhysics Letters*, **19 (6)**, 451-5 (1992).

[20] O. Martin, S. W. Otto and E. W. Felton, "Large-Step Markov Chains for the TSP Incorporating Local Search Heuristics", *Operations Res. Letters*, **11 (4)**, 219-24 (1992).

Estimating the Number of Local Minima in Complex Search Spaces

Rich Caruana †,‡ and Matthew Mullin‡

† UCLA; ‡ JustResearch,

Most optimization research is devoted to new or improved algorithms. Little effort is spent characterizing search spaces so that appropriate algorithms can be selected. We present an efficient method for estimating the number of local minima in big search spaces. The method is based on the statistics of the birthday problem: "How many people must be in a room before the probability is 1/2 that two people share the same birthday?" Assuming that the probability of birthdays is uniform on a 365 day year, one can estimate that the probability of a birthday being duplicated is 0.5 when there are 23 people in the room.

We reverse the birthday problem to address the question "*How long is the year if a birthday is duplicated when we put k people in the room?*" Generalizing the usual approximation for the birthday problem and solving for N, the number of days in the year, as a function of k, the number of people in the room, and P_D, the probability of duplication, yields:

$$N \approx \frac{k^2}{-2ln(1-P_D)}$$

By counting the number of local minima explored by search before some local minimum is visited twice, we can estimate the total number of local minima in the search space. One procedure for doing this is to randomly sample local minima, recording each one, until one of the minima is visited a second time. This yields an estimate of the number of minima needed for the probability to be about 0.5 that a minimum is duplicated. Unfortunately, it is difficult to efficiently sample local minima uniformly. In the absence of an efficient means of uniformly sampling local minima we can either use an inefficient uniform sampling procedure, or an efficient non-uniform sampling procedure. Here we sacrifice accuracy for a gain in efficiency. If sampling is not uniform, fewer samples will be needed for duplication to occur, so the expected total number of local minima will be *underestimated.* This is acceptable because lower bounds on the number of local minima are more useful than upper bounds. Iterated hillclimbing using steepest, nearest, or stochastic descent starting from randomly selected initial points is one way to efficiently sample local minima in many search spaces.

We used this method to estimate the number of local minima in the search spaces of multiplierless FIR digital filters. Optimal tap values for *full-precision* FIR filters can be computed analytically [1], but full-precision filters are slow and consume a lot of chip area. Restricting tap values to powers of 2 allows fast, compact shift registers to be used instead of multipliers [2]. There are no analytic techniques for selecting optimal tap values for these filters, so we use numerical optimization. Low-pass multiplierless filters with 31, 41 and 51 taps yield search spaces with 16, 21 and 26 dimensions, respectively, because tap values are symmetric about the central tap. Taps can take 15 values (zero, and positive and negative reciprocal powers of two). The search spaces are large, contain a large number of poorly performing local minima, and the proportion of good filters grows increasingly small as the number of taps gets large [3].

We performed 10,000 hillclimbs with iterated steepest descent for filters with 31, 41, and 51 taps. On average, 27 steps were required for each descent to become trapped in a local minimum. A duplicated local minimum was found in only the smallest search space (31 taps). Thus the estimates for the 41 and 51 tap search spaces are lower bounds on the estimated lower bounds. The results indicate that there are more than 4.22×10^7 local minima in the 31-tap search space. Thus the minima comprise at least 6.4×10^{-8} percent of the 31-tap search space. A simple upper bound analysis shows that there are fewer than 10^{14} local minima in the 31-tap search space, or less than 1.5×10^{-3} of the points in the space are local minima.

The birthday procedure is efficient for two reasons: 1) it needs sample sizes of roughly the square root of the total number of local minima to estimate the total number; 2) it uses optimization to find local minima. The savings can be dramatic. In the 31-tap space, each hillclimb required about 27 steps to find a local

Table 1: Estimated Number of Local Minima in the Search Spaces

# Taps	# of States	# of Local Minima until 1st Duplicate	Estimated # of Local Minima in Space
31	6.6×10^{18}	7,651	4.22×10^7
41	5.0×10^{24}	>10,000	$> 7.21 \times 10^7$
51	3.8×10^{30}	>10,000	$> 7.21 \times 10^7$

minimum, and each step required 32 function evaluations to determine the direction of steepest descent. Thus the 7651 hillclimbs required 6.4×10^6 function evaluations to find the first duplicate. A nave approach to estimating the number of local minima is to randomly sample points in the space and determine what fraction of these are minima. If minima comprise 6.4×10^{-8} percent of the space, we would have to sample 7.8×10^{10} points to have a 50% chance of finding a single minimum, and at each of these points 33 function evaluations would be required to determine if the point is a local minimum. Thus 2.6×10^{12} evaluations would be required to estimate the number of local minima using random point sampling. This is 10^5 times more function evaluations than required using birthday statistics. Moreover, if we stop sampling points after using the 10^4 samples used with the birthday method, we almost certainly will not have found even one local minimum, so all we'll know is that there probably are fewer than 6.6×10^{14} local minima in the space. That upper bound isn't very informative. One key advantage of the birthday procedure is that the information needed to estimate the number of local minima is available whenever optimization is run multiple times. It is not necessary to run a different, and possibly costly procedure to make an estimate.

It can be estimated that seeing most points in a space containing N points requires about $N \ln N$ random samples. Thus it would take at least 7×10^7 samples of local minima in the 31-tap space to have a high probability of seeing the global minimum. This is 7×10^3 times more hillclimbs than were needed to make this estimate. 10^4 hillclimbs took 1 CPU day, so it would take at least 20 years of iterated hillclimbing to reliably find the optimal filter. A more efficient search procedure (or a faster computer) probably is required. With a small amount of search, we can estimate how much more search is required for that search method to find the global optimum (and that estimate can be made without knowing the true global optimum). Estimating the number of local minima gives us a criterion for halting search long before the first duplicate minimum is found: **If it is important to find the global optimum, we can stop running a search procedure as soon as we collect enough unique local minima so that the estimated lower bound on the total number of local minima is larger than we can afford to search with that procedure**.

Another use of estimates made with the birthday method is to select what optimization procedure to use: Suppose we run simulated annealing (SA) 15 times using a slow cooling schedule and estimate that SA-SLOW *effectively* searches a space containing at least 75 local minima. Suppose we run SA again, doing 150 trials with a fast cooling schedule that is more likely to get stuck in inferior local minima, and estimate that SA-FAST *effectively* searches a space containing about 5,000 local minima. Assuming these are tight lower bounds, if SA-FAST is more than

$$\frac{5000 \log 5000}{75 \log 75} \approx 131.5$$

times faster than SA-SLOW, and we want to find the global optimum, it probably will be better to run more SA-FASTs despite the fact that each run of SA-FAST is more likely to find inferior local minima than SA-SLOW. This decision making can be automated and embedded in a tool that interleaves execution of different optimization methods, and uses the accumulating statistics to decide which optimization method to allocate future trials to.

The main difficulty when using the birthday method to estimate the number of local minima in a search space is the problem of non-uniform sampling. If some minima have larger basins of attraction than other minima, search procedures like iterated hillclimbing will fall into the larger basins more often, skewing the statistics so that duplicates occur more frequently, and degrading the tightness of the lower bound. (If sampling were uniform the method would yield an unbiased estimate instead of an estimate of a lower bound.) There are a number of ways to deal with the problem of non-uniform sampling of local minima, depending on how extreme the differences in basin sizes are and how tight the bound must be. (See http:/www.cs.cmu.edu/ caruana/pubs/ijcai99 for details.)

References

[1] L.R. Rabiner and B. Gould. *Theory and Application of Digital Signal Processing* . Prentice Hall, Englewood Cliffs, NJ, 1975.

[2] D. Koo and A. Miron. Design of Multiplierless FIR Digital Filters with two to the Nth Power Coefficients. Philips Labs Report #TR-86-036, September 20, 1986.

[3] R.A. Caruana and B.J. Coffey. Searching for Optimal FIR Multiplierless Digital Filters with Simulated Annealing. Philips Labs Report #TR-88-031, March 21, 1988.

Experimentally Determining Regions of Related Solutions for Graph Bisection Problems

Ted Carson and Russell Impagliazzo

University of California, San Diego,

Local search heuristic algorithms (e.g. Metropolis, simulated annealing [5, 7], WalkSAT [1], Go-With-the-Winners [2, 3, 4], etc.) present intuitively appealing mechanisms to mix greedy behavior with diversity, and have achieved remarkable success on some problem domains. However, it is often the case that there is little understanding, either theoretical or empirical, of why a heuristic succeeds or fails for various problem domains, or of how to optimize a method for a particular domain. Implementations of these algorithms for hard combinatorial optimization problems is therefore often by trail-and-error, and performance predictability and confidence are low.

We propose a general method for the experimental study of such heuristics. This method is intended to relate performance of a heuristic directly to the combinatorial features of the search graph upon which it operates. To do this we gather statistics with respect to the clustering and connectedness of solutions with roughly the same cost. Since for small costs these solutions are rare this cannot be done simply by random sampling. However, if the sub-graphs of low cost solutions have the a property called "local expansion" (intuitively meaning that there are a few highly connected components covering the sub-graph), they can be uniformly sampled by using a Go-With-the-Winners (GWW) algorithm [2, 3, 4]. Once we gather such statistics, they are used to give a quantitative analysis of the search space, defining key characteristics that affect local search heuristic performance. This analysis is finally used to build a *causal* model of performance for various heuristics.

The method we propose can be divided into three phases: *validation*, *mapping*, and *prediction*.

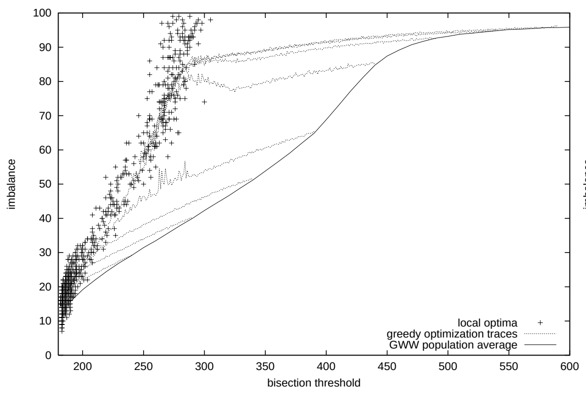
Validation: Before we can use the data from the GWW algorithm, we must establish confidence that the algorithm is sampling uniformly. This will be true if the search graph has the local expansion property, however we cannot directly test for or prove that this is the case (else we could prove effectiveness of the GWW algorithm directly!). We instead offer falsifiable tests that this is the case. These tests are intended to verify that the lowest cost solutions are being found, that known structures in the search space are reflected in the samples, that the samples and consistent across runs, and that the algorithm is behaving asymptotically with regard to its parameters.

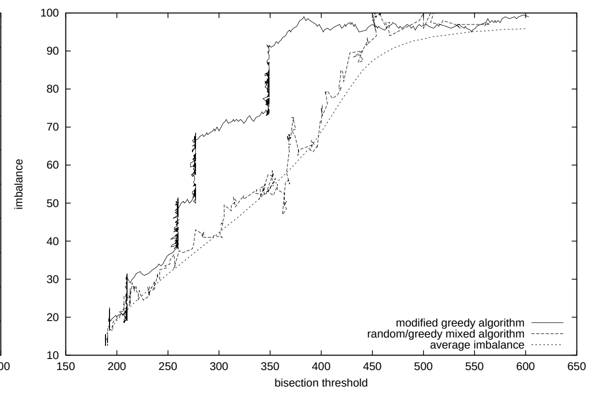
Mapping: Once it is established that the GWW algorithm is generating uniform samples, these samples can be used to build a map of the structure of the search graph. We are interested in features of the search graph that affect local search heuristics. The number of connected components of the sub-graph of some quality. and the connectedness (expansion) of these components are two such features of particular importance.

Prediction: Using this map we can model what would happen if we applied a particular local search heuristic to the problem. The entire run of the heuristic is modeled, and not just the final solution value. In this way the effects of various algorithm techniques and parameters can be examined and optimized.

Our method has the following advantages: 1) *Causal models* of heuristic performance are produced allowing us to predict the performance of an algorithm, *and* explain why it behaves as it does. In addition this provides for principled ways to select parameters of various algorithms, often a difficult problem when applying general purpose heuristics. 2) The models are *falsifiable* in the sense of the natural sciences: i.e. although proofs of performance are not obtained by the experiments, consequences and predictions can be rigorously tested. 3) Our method gives insight into which problems are susceptible to a broad class of heuristics and which are not.

As an example of this method we considered the minimum bisection problem. We examined problems instances drawn from a random graph model, $G_{n,p,q}$, which are generated by dividing a graph into a preferred bisection and adding edges with high probability p between vertices on the same side of the partition and





with lower probability q between vertices on opposite sides. Intuitively this "planted bisection" influences the search space by placing a bias that solutions closer to the planted bisection are of expected lower value. We selected a small separation between p and q where the problems are most difficult.

Once we validated the uniformity of the GWW algorithm (10000 particles and 4096 random walk steps for a 400 node and 1195 edge graph), we proceeded to map the search graph for this problem. As expected, the search space was smoothly biased with lower cuts occurring closer to the planted bisection. Cuts that were at an average distance from the planted bisection for their value were never local minima. However, those that were farther than average from the planted bisection were often local minima. Further, greedy algorithms tended to move toward these bisections with low cuts but high distances from the planted bisection (left) and consequently fail. These observations led to the creation of several simple algorithms designed to avoid this overly optimized region by mixing random walks with greedy moves. We show the traces of two such algorithms (right). Algorithm A introduces a random walk when the greedy method finds a plateau in the search space, while algorithm B mixes greedy and random moves more smoothly.

We see that the algorithms behaved as predicted by the map of the search space, avoiding the difficult region of overly optimized bisections. The map generated by our method proved accurate and useful for algorithm creation and optimization.

REFERENCES

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

[2] D. Aldous and U. Vazirani. "Go with the winners" Algorithms. In *Proc. 35th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 492–501, 1994.

[3] Dimitriou, A., and Impagliazzo, R., Towards a Rigorous Analysis of Local Optimization Algorithms", *28th ACM Symposium on the Theory of Computing* 1996.

[4] Dimitriou, A., and Impagliazzo, R., Go-with-the-winners Algorithms for Graph Bisection, SODA 98, pp. 510-520.

[5] M. R. Jerrum and G. Sorkin. Simulated annealing for graph bisection. In *Proc. 34th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 94–103, 1993.

[6] M. R. Jerrum and A. Sinclair. Conductance and the rapid mixing property of Markov chains: The approximation of the permanent resolved. In *Proc. 20th ACM Symposium on Theory of Computing (STOC)*, pages 235–244, 1988.

[7] D. S. Johnson, C. R. Aragon, L. A. McGeoch and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation, Part I (Graph Partitioning). *Operations Research* 37 (1989), pages 865–892.

Optimization of Parallel Search Using Machine Learning and Uncertainty Reasoning

Diane J. Cook, Piotr J. Gmytrasiewicz, and Chiu-Che Tseng

University of Texas at Arlington,

Because of the dependence AI techniques demonstrate upon heuristic search algorithms, researchers continually seek more efficient search methods. Advances in parallel and distributed computing offer potential performance improvement, and in response a number of approaches to parallel search have been developed. While these approaches have many contributions to offer, determining the best use of each contribution is difficult because of the diverse search algorithms, machines, and applications reported in the literature.

In response to this problem, we have developed the Eureka parallel search engine that combines many of these approaches to parallel heuristic search. Eureka is a parallel IDA* search architecture that merges multiple approaches to task distribution, load balancing, and tree ordering, and can be run on a MIMD parallel processor, a distributed network of workstations, or a single machine with multithreading. Our goal is to create a system that automatically selects an optimal parallel search strategy for a given problem space and hardware architecture.

A parallel search algorithm requires a balanced division of work among processors. One method of dividing IDA* is to give a copy of the entire search tree to each processor with a unique cost threshold [11]. Using this approach, processors search the tree simultaneously to their own threshold and terminate when a goal is found. An alternative approach distributes the tree among processors [4]. Using this approach, the root node of the search space is given to the first processor and other processors are assigned subtrees of that root node as they request work. A compromise between these approaches is to divide the set of processors into *clusters* [1]. Each cluster is given a unique cost threshold, and the search space is divided between processors within each cluster.

Because one processor may run out of work before others, load balancing is used to activate the idle processor. Approaches must be selected for deciding when to load balance, which processor to approach for more work, and how much work to share. In addition, methods for modifying the left-to-right order of the tree during search can yield substantial performance improvements for serial and parallel search algorithms.

The Eureka system merges together many parallel search strategies. Parameters can be set that control the task distribution strategy, the load balancing strategies, and the ordering techniques. To automate the selection of parallel search strategies, Eureka The system searches a sampling of the space and calculates search space features including average branching factor, average heuristic estimate error, tree imbalance, heuristic branching factor, and heuristic distance estimate of the root. Information describing the hardware is also used such as the number of processors and average communication latency.

Initially, we used C4.5 to induce a decision tree from pre-classified training examples. Training examples represent runs of sample problem spaces with varying search strategies, and the correct "classification" of each training example represents the search strategy yielding the greatest speedup. For each new problem, Eureka performs a shallow search through the space to collect features describing the new problem space and architecture. Features of the tree are calculated and used to index appropriate learned rules. Eureka then initiates a parallel search employing the selected strategies.

Two sources of uncertainty arise in this domain that can prevent traditional machine learning techniques from performing well. First, the search space feature values are estimates and thus not always accurate. Second, there does not always exist a clear strategy winner for each training case. On some problem instances two or more strategy selections perform almost equally well, and some run time variances occur on many machines.

Due to these uncertainties in the domain, we next model the problem with a belief network. We use an extension of belief networks known as influence diagrams. Apart from nodes that represent uncertain variables (oval nodes), influence diagrams also have decision nodes (rectangular nodes) and a utility node

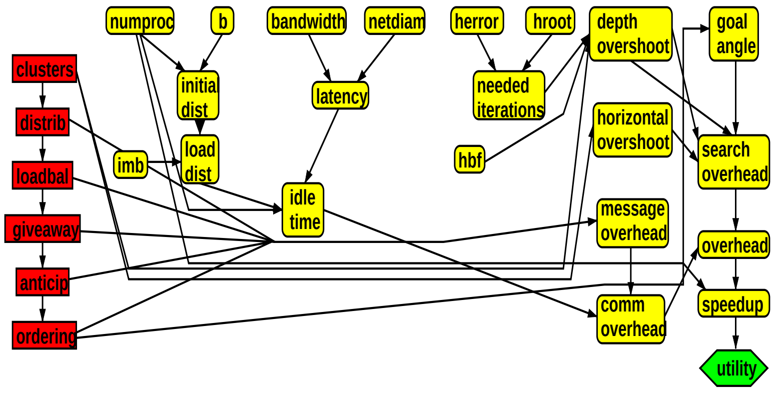


Figure 4: A model of the factors that influence speedup of parallel search algorithms.

Approach	1 Cluster	2 Clusters	4 Clusters	C4.5	C4.5Fil	BeliefNetwork
Speedup	55.30	60.98	58.79	57.14	86.76	68.66

Table 2: Clustering Speedup Results

(hexagonal node). By representing the way alternative decisions influence the state of the domain, the influence diagram can be used to arrive at optimal decisions.

A graphical representation of our influence diagram is shown in Figure 4. Decisions to be made include the task distribution strategy, number of clusters, amount of work to share, anticipatory load balancing trigger, type of load balancing, and tree ordering. For our parallel search problem, Eureka's utility node in Figure 4 directly depends on a single node of the domain – speedup. Conditional probability tables modeling the probabilistic dependencies between nodes in the network are learned from provided training data.

We tested the ability of the influence diagram to provide a basis of making parallel search strategy decisions by comparing decisions based on predicted speedup values from the influence diagram built using Netica with decisions based on the C4.5 learning system. To create test cases, we ran 100 Fifteen Puzzle problem instances multiple times on 32 processors of an nCUBE, once using each parallel search strategy in isolation. Features of the search space, and architecture are stored for each problem.

We compared the results of Netica-selected strategies on test data to C4.5-selected strategies and to each strategy used exclusively for all problem instances. Speedup results for various strategy decisions averaged over all problem instances are shown in Table 2 below.

From the results of this experiment, the belief network did outperform all of the fixed strategies as well as C4.5 using all 100 problem instances. C4.5Fil yielded the best results, but was only trained and tested on cases with clear winners. Each of the automated approaches to selected search strategies resulted in better performance than using just one fixed parallel search strategy. These results indicate that machine learning and uncertainty reasoning techniques can be effectively used to perform automatic selection of parallel search strategies, and may be effective for other optimization problems as well.

References

[1] D J Cook and R C Varnell. Maximizing the benefits of parallel search using machine learning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 559–564. AAAI Press, 1997.

[2] Diane J Cook, Larry Hall, and Willard Thomas. Parallel search using transformation-ordering iterative-deepening A*. *International Journal of Intelligent Systems*, 8(8):855–873, 1993.

[3] Martin Frank, Piyawadee Sukavirija, and James D Foley. Inference bear: designing interactive interfaces through before and after snapshots. In *Proceedings of the ACM Symposium on Designing Interactive Systems*, pages 167–175. Association for Computing Machinery, 1995.

[4] V Kumar and V N Rao. Scalable parallel formulations of depth-first search. In Kumar, Kanal, and Gopalakrishan, editors, *Parallel Algorithms for Machine Intelligence and Vision*, pages 1–41. Springer–Verlag, 1990.

[5] Henry Lieberman. Integrating user interface agents with conventional applications. In *Proceedings of the ACM Conference on Intelligent User Interfaces*. Association for Computing Machinery, 1998.

[6] A Mahanti and C Daniels. SIMD parallel heuristic search. *Artificial Intelligence*, 60(2):243–281, 1993.

[7] Nihar R. Mahapatra and Shantanu Dutt. New anticipatory load balancing strategies for parallel A* algorithms. In *Proceedings of the DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 197–232. American Mathematical Society, 1995.

[8] Steven Minton. Automatically configuring constraint satisfaction programs: a case study. *Constraints*, 1(1):7–43, 1996.

[9] Peter Norvig and D Cohn. Adaptive software. *PC AI Magazine*, 1997.

[10] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.

[11] Curt Powley and Richard E Korf. Single-agent parallel window search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):466–477, 1991.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.

[13] Ken Calvert Samrat Bhattacharjee and Ellen W Zegura. An architecture for active networking. In *High Performance Networking*, 1997.

[14] R. D. Shachter. Evaluating influence diagrams. *Operations Research*, 34:871–882, 1986.

[15] Peter Steenkiste, Allan Fisher, and Hui Zhang. Darwin: resource management for application-aware networks. Technical Report CMU-CS-97-195, Carnegie Mellon University, 1997.

[16] Scott Taylor, David Levine, Krishna Kavi, and D. J. Cook. A comparison of multithreading implementations*. In *Yale Multithreaded Programming Workshop*, 1998.

[17] Jian Xu and Kai Hwang. Heuristic methods for dynamic load balancing in a message-passing multicomputer. *Journal of Parallel and Distributed Computing*, 18(1):1–13, 1993.

[18] Songnian Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering*, 14(9):1327–1341, September 1988.

Adaptive Heuristic Methods for Maximum Clique

Arun Jagota † and Laura A. Sanchis ‡

† University of California, Santa Cruz; ‡ Colgate University,

Maximum Clique is the problem of finding a largest set of pairwise adjacent vertices in a given graph G [5]. This problem is NP-hard even to approximate well [1], and arises in several applications [2]. This problem has attracted considerable attention over the years (see the review published by Pardalos and Xue with about 260 references [15]).

For NP-hard problems such as Maximum Clique no efficient exact algorithms exist. Heuristic methods therefore abound. In recent years researchers have begun to recognize the value of incorporating adaptation into heuristic methods (as one example, see [4]). This short paper surveys our own work along these lines [11, 7, 17] which incorporates adaptation into multiple restarts methods in which each restart performs randomized local search.

We found in extensive experiments on Maximum Clique that the adaptive restarts we employed consistently worked no worse (and often better) than nonadaptive restarts, while imposing an almost negligible computational overhead. Simple forms of adaptation are also easy to add on to otherwise nonadaptive methods. Thus it seems there is no reason not to do so.

In this short paper we present our methods, then discuss some experimental results. Two types of adaptation are explored—one in which vertex weights are adapted, and the other in which the initial clique is adapted. On different types of graphs, the different types of adaptation seem to work better.

The Adaptive Methods

The algorithms are best described as evolving from a base algorithm, which we call *randomized greedy search* (RGS).

```
FinalClique RGS(Graph $G$, WeightVector $\mathbf{w}$, Clique $C_i$ )
// $\mathbf{w} = (\mathbf{w_i})$ where $w_i$ is vertex $i$'s weight


1. $C = C_i$;
2. $S = \{ v \in G \backslash C \mid v$ is adjacent to every vertex in $C\}$;
3. while $S$ is not empty do
4.         Pick vertex $i$ from $S$ with probability $w_i / \sum_{j \in S} w_j$;
5.         Add vertex $i$ to $C$;
6.         Recompute $S$ as in step 2;
7. return C;
```

RGS works by extending the initial, possibly empty, clique C_i to a final clique C by adding feasible vertices one by one in a randomized greedy way. The greediness comes in in step 4, since the chosen non-uniform distribution favors picking those vertices from S that have large weights. The idea is to add some randomization to avoid the usual traps that greedy falls into but not too much so as to totally wipe out the greediness. The randomization in RGS will also work well with multiple restarts.

Next, we describe a nonadaptive restarts method, called NA, on top of RGS. NA takes two arguments in addition to the graph: a vector $\mathbf{w}$ of weights on the vertices, and the number k of restarts.

```
FinalClique NA(Graph $G$, WeightVector $\mathbf{w}$, integer k)
```

```
1. $C = \emptyset$;
2. for r = 1 to k do
3.     $C_r$ = RGS( $G$, $\mathbf{w}$, $\emptyset$ );
4.     $C$ = larger-set-of ($C$, $C_r$);
5. return $C$;
```

Notice that NA always calls RGS from the same initial clique, the empty set. It is obvious why NA will work better than (a single call to) RGS.

In our work, we investigated two choices for $\mathbf{w} = (\mathbf{w_i})$: (i) $w_i = 1$ for all i and (ii) $w_i = d(i)$ for all i. Here $d(i)$ is the degree of vertex i in the given graph G. We will denote the first choice as NA(1) and the second as NA(d). Interestingly, despite the intuitive appeal of the second weighting (vertices in large cliques have large degree), we found via extensive experimentation that NA(d) worked better than NA(1) only occasionally (and sometimes worse). Perhaps the restarts offset any possible benefits of degree-based weighting.

The third method, which we call AW, is an extension of NA that adapts the vertex weights from restart to restart.

```
FinalClique AW(Graph $G$, WeightVector $\mathbf{w}$, integer k)

1. $C := \emptyset$;
2. $\alpha_1 := \alpha_2 := 1$;
3. for r := 1 to k do
4.     $C_r$ := RGS( $G$, $\mathbf{w}$, $\emptyset$ );
5.     if $|C_r| > |C|$ then
6.         $w_i := w_i + \alpha_1 \times (|C_r| - |C|)$ for all $i \in C_r$;
7.     else
8.         $w_i := w_i + (-1 + \alpha_2 \times (|C_r| - |C|))$ for all $i \in C_r$;
9.     $C$ := larger-set-of ($C$, $C_r$);
10.    $\alpha_1 := 1.01 \times \alpha_1$;
11.    $\alpha_2 := 0.99 \times \alpha_2$;
12. return $C$;
```

The vertex weights are adapted as follows. AW keeps track of the size of the best clique it has found so far. If in the next restart, it finds a better clique then it "rewards" the vertices in the clique by increasing their weight, otherwise it "punishes" them by decreasing their weight. The magnitude of the reward (or punishment) is made to depend on the magnitude of the improvement (or worsening) and also on the restart number. With regards to the latter, an improvement found in a late restart counts for more, as does a worsening found in an early restart.

We find that the method exhibits the following characteristics. If an improved clique is found in some restart, AW focuses subsequent search into its "neighborhood". If no further improvement is found soon enough, the search gradually gets defocused from this neighborhood.

In extensive testing, AW often found significantly larger cliques than NA in the same amount of alloted time [11, sec 4.3].

Like with NA, we investigated AW with the same two types of initial weighting, denoted AW(1) and AW(d) respectively. Once again, there was no clear winner between AW(1) and AW(d). AW(1) seemed to work a bit better overall. Perhaps the adaptive restarts offset any possible benefit of degree-based weighting.

The fourth method, which we call AIC, is an alternative extension of NA that adapts the initial clique, rather than the vertex weights, from restart to restart. Thus, unlike AW, AIC does not always start a restart from the same initial clique. For the same number of restarts, AIC runs faster than AW because it often starts from a large clique and thus has to grow it less. On the other hand, in extensive experiments we find that AIC often takes many more restarts to find the same quality solution than does AW. The reason for this is not clear.

```
FinalClique AIC(Graph $G$, WeightVector $\mathbf{w}$, integer k)

1. $C = \emptyset$;
2. $C_i = \emptyset$;
3. for r = 1 to k do
4.     $C_r$ = RGS( $\mathbf{w}$, $C_i$ );
5.     if $|C_r| > |C|$ then
6.         $C_i = C_r$;
7.     $C_i = C_i \setminus$ { one randomly-chosen vertex in $C_i$ }
8.     $C$ = larger-set-of ($C$, $C_r$);
9. return $C$;
```

AIC exhibits characteristics similar to those of AW, focusing the search into the neighborhood of a newly-found better clique, defocusing subsequent search gradually when this does not lead to an improvement.

Like with NA and AW, we investigated AIC with the same two types of weighting, denoted AIC(1) and AIC(d) respectively. Once again the results were mixed (sometimes AIC(1) worked better, sometimes AIC(d)).

The AIC principle was influenced by the work of T. Grossman [6] where the power of a simple greedy algorithm for Maximum Clique was boosted significantly and demonstrably (via extensive experiments) by adaptive initialization of a somewhat similar kind. Adaptations from restart to restart have also been used in [4] and [3].

Most of our experiments were done on slightly different versions of NA, AW, and AIC in which the number of restarts given to each method was not fixed in advance. Rather, the restart parameter k in each of these methods was replaced by a "no progress" parameter np, and the method terminated when np successive restarts led to no improvement in solution quality. Although this does not reduce the number of parameters to be set, the np parameter is clearly superior to the k parameter.

Experimental Results Summary

This section reports a summary of our computational experiments and results on NA(1), NA(d), AW(1), AW(d), AIC(1), and AIC(d). These methods were all evaluated on different kinds of graphs: (i) graphs with prespecified maximum clique size designed to be hard for the problem, (ii) random graphs of various densities, and (iii) highly compressible graphs of a certain type.

The graphs of type (i) are described in detail in [11, 17]. Experiments were conducted on a variety of graphs (parametrized by maximum clique size C, and density D) of this type on 800 vertices. The value of the np parameter was (by trial and error) fixed to 100, for all the methods. On 800-vertex graphs of density 0.70, AW(1) worked best. Interestingly, NA(1) worked better than AIC(1). On graphs of density 0.9, AW(1) still worked best. This time however AIC(1) worked better than NA(1). The differences between the performances were striking for the larger values of C. On graphs of density 0.9, AIC(1) worked best on graphs with larger C. AW(1) worked better than NA(1). From these experiments, the following clear trends emerged: *AIC(1) worked better on the denser graphs, AW(1) on the less dense graphs; for the same density, the solution quality differential was highest on graphs with the larger cliques.*

The graphs of type (ii) are parametrized by p, the probability of independently introducing an edge between a pair of vertices. The six methods were tested on ten 1000-vertex graphs each for three values of p. (The variance in the results was seen to be small, indicating that the small sample size should suffice.) For $p = 0.5$ and $p = 0.7$, there was little difference in solution quality. For $p = 0.99$, AIC(d) worked best, with AIC(1) a close second. The remaining methods were significantly poorer, with AW(1) edging out the others.

The graphs of type (iii) are highly compressible in nature, and were designed to elicit poor performance from simple methods. It is difficult to describe these graphs here, the reader is refered to [10]. (In that paper it was experimentally demonstrated that these graphs weed out the poorer methods from the better ones, while random graphs don't.) Experiments were conducted on fifty 100-vertex graphs of this type. On average, there was little difference between the solution quality found by all methods. AW(1) edged out the others. Both NA methods were a close second. Both AI methods were a bit poorer.

Here we examine how well the new algorithms perform on these graphs. To facilitate comparisons, we evaluate the new algorithms on the exact same graphs evaluated earlier. The results are presented in the full paper; here we summarize them. Both the NA algorithms perform very well. This correlates well with the previous result that NA0(1,400) and NA0'(1,400) also worked very well. Here NA0' is a variant of NA0 which begins from the initial state V rather than from $\emptyset$. Note that the only difference between NA and NA0 is in the number of restarts used. It is reassuring to see that the current performance of NA(1,150) is almost identical to that of the earlier recorded performance of NA0(1,400). The AW(1) algorithm performs the best among the new algorithms; its performance is virtually identical to that of NA0'(1,400) which worked best in the original experiments. Note that AW(1) achieves this performance in roughly three-fifths as many restarts as did NA0'(1,400). Interestingly, both AI algorithms perform somewhat poorer.

Conclusions and Future Work

No single winner emerged. Adding on adaptation to NA nonetheless never hurt (and often helped considerably).

Our methods are easily extendible to a wider class of problems, that we call *induced subgraph optimization problems*, defined as follows: *given a graph G and a property P on vertex-induced subgraphs of G, we wish to find a maximum-cardinality set $U \subseteq V(G)$ which satisfies P.* Maximum Clique is a member of this class. There is a general theorem establishing intractability of a large subclass of problems in this class [14, 13]. Our future work will involve testing how well our methods work on some other problems in this class.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *The Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[2] E. Balas and C.S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4), November 1986.

[3] S. Baluja and S. Davies. Combining Multiple Oprimization Runs with Optimal Dependency Trees. Technical Report, Department of Computer Science, Carnegie-Mellon University, 1997.

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

[5] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[6] T. Grossman. Applying the INN model to the MaxClique problem. In D.S. Johnson and M.A. Trick, editors, *DIMACS Series: Second DIMACS Challenge*, pages 125–145. American Mathematical Society, 1996. Proceedings of the Second DIMACS Challenge: Cliques, Coloring, and Satisfiability.

[7] A. Jagota. An adaptive, multiple restarts neural network algorithm for graph coloring. *European Journal of Operational Research*, 93:257–270, 1996.

[8] A. Jagota and M. Garzon. On the Mappings of Optimization Problems to Neural Networks. In *Proceedings of World Congress on Neural Networks 1994*, pages 391–398, IEEE, 1994.

[9] J.H.M. Korst and E.H.L. Aarts. Combinatorial optimization on a Boltzmann machine. *Journal of Parallel and Distributed Computing*, 6:331–357, 1989.

[10] A. Jagota and K.W. Regan. Performance of neural net heuristics for maximum clique on diverse highly compressible graphs. *Journal of Global Optimization*, 10:439–465, 1997.

[11] A. Jagota, L. Sanchis, and R. Ganesan. Approximating maximum clique using neural network and related heuristics. In D.S. Johnson and M.A. Trick, editors, *DIMACS Series: Second DIMACS Challenge*, pages 169–204. American Mathematical Society, 1996. Proceedings of the Second DIMACS Challenge: Cliques, Coloring, and Satisfiability.

[12] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation, Part II (Graph Coloring and Number Partitioning). *Operations Research*, 39:378–406, 1991.

[13] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems, In *Proceedings of 20th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Comput. Sci. 700, Springer-Verlag, 40-51.

[14] M. Lewis and M. Yannakakis; The Node-Deletion Problem for Hereditary Properties is NP-Complete, *Journal of Computer Systems and Sciences*, **20**, (1980).

[15] P.M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4:301–328, 1994.

[16] L. Sanchis. Test Case Construction for the Vertex Cover Problem. in N. Dean and G.E. Shannon, editors, *Computation Support for Discrete Mathematics*, pages 315–326. American Mathematical Society, 1994.

[17] L. Sanchis and A. Jagota. Some experimental and theoretical results on test case generators for the maximum clique problem. *INFORMS Journal on Computing*, 8:2:87–102, Spring 1996.

Probabilistic Modeling for Combinatorial Optimization

Shumeet Baluja, Scott Davies

Carnegie-Mellon University,

This work originated in an attempt to create an explicit probabilistic model of the behavior of genetic algorithms [12][17][18]. Genetic algorithms, or GAs, can be viewed as creating *implicit* probabilistic models over possible solutions by maintaining a population of previously evaluated solutions. Rather than using explicit models of how the parameters of high-quality solutions tend to relate to one another, GAs attempt to preserve these relationships by using crossover-based recombination operators on members of the population in order to generate new candidate solutions.

One attempt towards making the GA's probabilistic model more explicit was the Population-Based Incremental Learning algorithm (PBIL) [1][3]; this was further explored in [19][20][16]. PBIL uses a very simple probabilistic model that does not model inter-parameter dependencies — each parameter is handled independently. The PBIL algorithm works as follows: instead of using recombination/crossover to create a new population, a real-valued vector, **P**, is sampled. Assuming the solutions are represented as bit strings, **P** specifies the probability of generating a 1 in each bit position. Initially, all values in **P** are set to 0.5, so that solutions are generated from the uniform distribution over bit strings. A number of solution vectors are generated by stochastically sampling each bit independently according to **P**. The probability vector is then moved towards the highest-quality solution vectors thus generated, in a manner similar to the updates used in unsupervised competitive learning [15]. This cycle is then repeated. The final result of the PBIL algorithm is the best solution generated. This basic version of PBIL is similar to early work in cooperative systems of discrete learning automata [23] and to the Bit-Based Simulated Crossover algorithm [22]. Numerous extensions to this basic algorithm are possible; many are similar to those commonly used with genetic algorithms, such as variable or constant mutation rates, maintenance of the best solution found throughout the search, parallel searches, or local optimization heuristics.

One of the goals of crossover in GAs is to combine "building blocks" from two different solutions to create new sampling points. Because all parameters are modeled independently, PBIL cannot propagate building blocks in a manner similar to standard GAs. Nonetheless, in a variety of standard benchmark problems used to test GAs and Simulated Annealing approaches, PBIL performed extremely well [2]. While GAs attempt to preserve important relationships betweem solution parameters, they do not do so explicitly, and so must rely on the recombination operator to combine *random* subsets of two "parent" solutions in hopes of coming up with a child solution that maintains the important building blocks. Would algorithms that employed probabilistic models in which these relationships were accounted for explicitly perform even better?

The first extension to PBIL that captured dependencies was *Mutual Information Maximization for Input Clustering (MIMIC)* [7]. MIMIC measured the mutual information [10] between each pair of parameters across a set of high-quality solutions, and then used these statistics to greedily build a probabilistic model in the form of a Markov chain over the solution parameters. Subsequent research [4] generalized this Markov chain formalism to Bayesian networks [21]. In particular, the same type of statistics used by MIMIC were used to learn a more general class of models — namely, trees rather than chains — and the optimal model within that class was found using Chow and Liu's algorithm [9]. Figure 5 illustrates the types of models used by PBIL, MIMIC, and this tree-based algorithm on a noisy version of a two-color graph coloring problem. We use Bayesian network notation for our graphs: an arrow from a node X to a node Y indicates that Y's probability distribution is conditionally dependent upon the value of X. The chain- and tree-shaped models shown were automatically learned during the process of optimization. (Note, however, that learned networks may not typically mirror optimization problems' structures so closely — in this example, the noise actually helped the algorithms recover the problem structure.)

Researchers have conducted numerous empirical comparisons (e.g. [19],[2], [4], [13]) of genetic algorithms and algorithms based on probabilistic modeling. Surprisingly, in many of the larger, real-world problems, simple models that do not maintain dependency information (such as PBIL) outperform GAs, which attempt

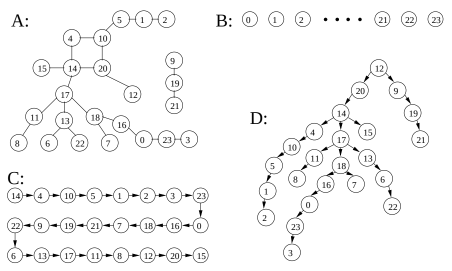


Figure 5: **A**: A two-color graph-coloring problem. **B**: the empty dependency graph effectively used by PBIL. **C**: the Markov chain learned by a MIMIC-like optimization algorithm. **D**: the Bayesian network learned by the tree-based optimization algorithm.

to capture this information implicitly. On test problems designed to exhibit large amounts of inter-parameter dependencies, PBIL's success was less predictable [11]. However, in the large majority of these problems, the dependency-tree-based algorithm consistently outperformed the other optimization techniques [6].

Perhaps the most interesting result found in this line of study is that the performance of the combinatorial optimization algorithms consistently improved as the accuracy of their statistical models increased: trees generally performed better than chains, and chains generally performed better than models with no dependencies. This suggests the possibility of using even more complex probabilistic models, although using models that are *too* complex would hurt the optimization algorithms by preventing them from performing enough exploration of the search space. Unfortunately, when we move toward network models in which variables can have more than one parent variable, the problem of finding an optimal network with which to model a given set of data becomes NP-complete [8]. Heuristics have been developed for finding good networks in such situations (e.g. [14]), and employing such methods in conjunction with combinatorial optimization is an exciting direction for future research. However, the $O(n^2)$ running time per iteration (where n is the number of solution parameters) required by the chain- and tree-based optimization algorithms is already prohibitively expensive on large-scale problems.

This computational problem can be alleviated by learning expensive probabilistic models only to generate starting points for simpler, faster optimization algorithms. *COMIT* [5] (for "Combining Optimizers with Mutual Information Trees") learned tree-based probabilistic models from the best solutions found during previous runs of hill-climbing or PBIL, and stochastically sampled these models to generate starting points for further runs of these algorithms. In the experimental results, employing the tree-based models in this manner typically significantly improved the solutions found by the faster optimization algorithms. Using more sophisticated probabilistic models for similar restarting algorithms is an interesting possible line of research, as is extending this methodology to optimization in real-valued spaces.

References

[1] S. Baluja. Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, 1994.

[2] S. Baluja. Genetic Algorithms and Explicit Search Statistics. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*. MIT Press, 1997.

[3] S. Baluja and R. Caruana. Removing the Genetics from the Standard Genetic Algorithm. In A. Prieditis and S. Russell, editors, *The International Conference on Machine Learning, 1995 (ML-95)*. Morgan Kaufmann Publishers, 1995.

[4] S. Baluja and S. Davies. Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space. In Jr. D. H. Fisher, editor, *The International Conference on Machine Learning, 1997 (ML-97)*. Morgan Kaufmann Publishers, 1997.

[5] S. Baluja and S. Davies. Fast Probabilistic Modeling for Combinatorial Optimization. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[6] S. Baluja and S. Davies. Pool-Wise Crossover in Genetic Algorithms: An Information-Theoretic Perspective. In *Proceedings of FOGA-98*, 1998.

[7] J. De Bonet, C. Isbell, and P. Viola. MIMIC: Finding Optima by Estimating Probability Densities. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, 1997.

[8] D. Chickering. Learning Bayesian networks is NP-complete. In *Learning from Data*, pages 121–130. Springer-Verlag, 1996.

[9] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. on Info. Theory*, 14:462–467, 1968.

[10] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley, 1991.

[11] L.J. Eshelman, K.E. Mathias, and J.D. Schaffer. Convergence Controlled Variation. In *Proc. Foundations of Genetic Algorithms 4*. Morgan Kaufmann Publishers, 1996.

[12] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* 1989.

[13] J. R. Greene. Population-Based Incremental Learning as a Simple Versatile Tool for Engineering Optimization. In *Proceedings of the First International Conf. on EC and Applications*, pages 258–269, 1996.

[14] D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[15] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computing.* Addison-Wesley, 1991.

[16] M. Hohfeld and G. Rudolph. Toward a Theory of Population-Based Incremental Learning. In *International Conference on Evolutionary Computation*, pages 1–5, 1997.

[17] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Addison-Wesley, 1975.

[18] K. De Jong. An analysis of the behavior of a class of genetic adaptive systems. Ph.d. thesis, 1975.

[19] A. Juels. Topics in Black-box Combinatorial Optimization. Ph.D. Thesis, University of California - Berkeley, 1996.

[20] V. Kvasnica, M. Pelikan, and J. Pospical. Hill Climbing with Learning (An Abstraction of Genetic Algorithm). In *Proceedings of the First International Conference on Genetic Algorithms (MENDEL, '95)*, pages 65–73, 1995.

[21] J. Pearl. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32:245–257, 1987.

[22] G. Syswerda. Simulated Crossover in Genetic Algorithms. In D. L. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 239–255. Morgan Kaufmann Publishers, 1993.

[23] M. Thathachar and P. S. Sastry. Learning optimal discriminant functions through a cooperative game of automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(1),

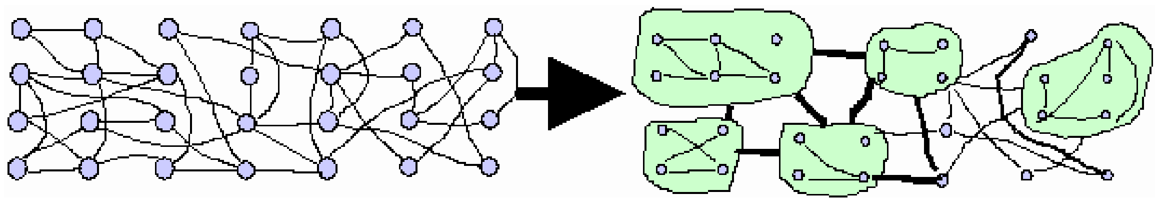


Figure 6: Diagramatic view of clustering

Adaptive Approaches to Clustering for Discrete Optimization

Wray Buntine, Lixin Su, and A. Richard Newton

University of California, Berkeley,

Many optimization algorithms in the field of computer aided design (CAD) of VLSI systems suffer from the fact that they do not scale linearly with the problem complexity. Clustering techniques have been adopted in some of these algorithms to address complexity but they are unfortunately limited to classes of problems where good intuitions about locality hold. In this paper, we demonstrate that learning good clusters is feasible for hyper-graph partitioning problem using a learning model for adaptive clustering.

Introduction

CAD tasks like place and route and hyper-graph partitioning use a technique called clustering to achieve significant performance increases. In fact, on those problems where clustering is used, it seems to be essential to achieve state-of-the-art performance on large problems. Clustering can be applied to SAT, place-and-route and hyper-graph partitioning by finding "nearby" nodes/variables and forcing their values to be identical.

Consider the hypergraph partitioning problem. In this, a hypergraph (a graph with n-ary edges instead of binary edges, e.g., VLSI circuit) is to be split into two pieces such that the number of hyper-edges split is minimum. Clustering on this works as follows: certain nodes are tied together or clustered so that, thereafter, they always occur in the same partition. Nodes so tied together form a single super-node and induce a new problem with fewer nodes and often fewer hyper-edges (these merge or are absorbed within a single super-node) For a generic graph partitioning problem (where all hyper-edges are of size 2), a representative clustering is illustrated on the right of Figure 1. Notice the induced edges onthe reduced problem.

An engineering-oriented study of several methods appears in [Hauck and Boriello, 1997].

Clustering has typically been based on *ad hoc* heuristics for a neighborhood metric together with standard clustering algorithms from the pattern recognition community, such as agglomerative clustering. Methods using clustering typically generate a sequence of solutions to the one problem. The current solution is mapped down into the reduced space, some optimization is done on this reduced problem, and then the solution mapped back up into the original space, thus affecting a non-local move guided by the clustering. Search then continues in the full space, and the process is repeated. Since the sequence of solutions is discarded, clustering is currently a static approach that cannot improve during an extensive run on a single problem or on multiple problems.

Clustering in Hypergraph partitioning

The optimization problem we consider here is standard hypergraph partitioning where the area of each partition is restricted to be less than 55% of the total area. Experiments were run on some of the larger problems in the ACM/SIGDA Layout Benchmark Suite from MCNC [1], where our copy was obtained from Charles Alpert's website at UCLA. Smaller problems are uninteresting for clustering methods.

One recent innovation in clustering is by Hagen and Kahng [3] whereby the intermediate results of local search are combined to create clusters. Note a p-way partition of a variable space induces a clustering with p super-nodes. Overlaying k binary partitions to form their finest multi-way partitioning, similarly, induces a clustering with up to 2^k super-nodes. Hagen and Kahng recommend using $k = 1.5 \log_2 C$ partitions for this construction, for C the number of nodes. Experiments on benchmarks reveal the problems here. As one increases k, the number of nodes in the induced clustering increases. For $k > \log_2 C$ the induced clustering can have up to C nodes, so nodes no longer occur in the same finest partition by chance. For k large enough, the complexity of the search on the induced problem can be just as bad as the compexity on the full problem. Note that if you are learning clusters from data "correctly," in some sense, then as k increases, the quality of the clustering should only improve as you get more data, not decrease as is the case here. For k smaller than $\log_2 C$, clustering is now introducing a random element, and thus as k decreases, the results of the search on the induced problem should degrade to produce a poor partition. Yet standard connectivity clustering methods work well, essentially with $k = 0$. Hagen and Kahng found $k = 1.5 \log_2 C$ to be a happy medium producing competitive results.

Our simple probabilistic model of local minima is as follows: we claim characteristics of the global optimum occur with frequency $(1 - q)$ in the perturbed local optimum. When we sample a local minima, it will have on average noise q on top of the global minimum. Therefore, to estimate whether a characteristic X holds in the global minimum, we estimate the frequency with which X occurs in local minima. If this frequency is greater than $(1 - q)$, then under this model with high probability, X also occurs in the global minimum. Note that q is a free parameter in our model. We claim different optimization problems with have different intrinsic noise levels in the broader solution neighborhood, and thus we leave q free to vary. Thus statistical information about the local minima are used to infer characteristics of the global minimum.

We apply this model to clustering as follows: the characteristics X we investigate take the form "node A and node B fall in the same partition." We have taken 200 FM runs to find a sample of local minima and have recorded partitions as well as the least cut-size for the entire 200 FM runs, We have taken statistics from these samples and for every pair of nodes (A, B) we then estimate the frequency with which they lie in the same partition. For a given noise level q under the simple model above, we can therefore estimate which nodes should belong in the same partition of the global minimum. Contingent on a value for q, this information is then collated for all nodes (A, B) to produce a clustering of the nodes, since "node A and node B" fall in the same partition is an equivalence relation. This approach is labelled "adaptive clustering".

EXPERIMENTS

To provide a benchmark, we have compared these results against clustering obtained using the agglomerative clustering method of Alpert *et al.* [1] excepting that recursive re-evaluation of the connectivity measure is not done. We also looked at 8 different levels of agglomeration and chose the one giving the minimum cut-size. We claim these modifications are fair since no recursive re-evaluation was done for the adaptive clustering method above, and the choice of optimum cut-size from 8 can only favor this method. Connectivity clustering and adaptive clustering are evaluated by running a high-performance non-clustering algorithm on them, ASFM of [2]. Because this is run on the clustered/reduced problems, its computation time is insignificant. We also extracted the best published results for each circuit from the literature [5, 4, 3, 2]. q is set in adaptice clustering by running the algorithm for $q = 0.8, 0.85, 0.9, 0.95, 0.97$ and picking the best, the additional computational overhead being insignificant. The geometric mean of the cut-sizes are reported in Table 1. Also note that connectivity clustering produced problems with nodes/hypergraphs having a geometric mean of 2142/2424 whereas for adaptive clustering, this is 704/1046.

[1]Being industry2, industry3, avq-small, avq-large, S9234, S13207, S15850, S35932, SS38417, S38584, 19ks and primary2, with nodes and hyper-edges of the order of 8000–20000.

best published	connectivity clustering	adaptive clustering on 200 FM	best of 200 FM
81.25	99.22	93.56	165.27

Table 3: Geometric mean cut-sizes from 12 industry benchmark

From the results we can conclude the following: (1) The adaptive clustering method generates significantly smaller hyper-graphs with significantly smaller cut-size. The difference is generally consistent across circuits. Thus adaptive clustering is significantly superior in forming clusters to methods attaining current best published. (2) Running ASFM once on a adaptive clustered hyper-graph, and no other computation, the results on the far smaller clustered hyper-graph are near best published for the problem. Typical state-of-art algorithms, considerably more sophisticated with recursive clustering and multiple iterations, score a geometric mean of about 86 on this measure so this simple approach is near state-of-the-art. (3) The clustering results provide a full 200% increase over the best cut-size resulting from the entire 200 FM runs. Thus there is clear evidence that under this model we learnt significant information from the local minima about the global minimum.

Notice the method of adaptive clustering described above uses 200 runs of FM as its sample. As a final experiment, we produced a hybrid of adaptive clustering and connectivity clustering using Bayesian statistics. We mapped the connectivity of an edge (A, B), $c(A, B)$, into a probability distribution on the probability that nodes A and B lie in the same partition at the global maximum. Lets call this probability $p(A, B)$. From calibration data for numerous problems, we modeled the probability $p(A, B)$ with a Beta distribution with mean $m(X) = 0.5 + 0.5 * (c(A, B)/0.6)^{0.7}$ and sample size 5. This means we place a density function over $p(A, B)$. Data from the FM runs is then used to update this Beta distribution using binomial sampling to obtain an estimate of $p(A, B)$ incorporating both the data from the FM runs and the connectivity $c(A, B)$. In statistical terms, the initial Beta distribution is a conjugate prior and the information from the sample (nodes A, B in same partition or not) is its complementary likelihood function. Using $k = 0.8 \log_2 C$, a half that of Hagen and Kahng, we were able to achieved results with this method comparable to the adaptive clustering on 200 FM runs. Moreover, the method yields clusters identical to connectivity clustering for $k = 0$.

Conclusion

In this paper, we proposed a simple probabilistic model of local minima and applied it to clustering for hyper-graph partitioning. The clustering metric in the model can also be improved through learning as more problems and their solutions are encountered. The experimental results on the tested benchmark circuits provided a clear evidence that under this model we learnt significant information from the local minima about the global minimum.

References

[1] C. J. Alpert and A. B. Kahng. Recent developments in netlist partitioning: A survey. *Integration: the VLSI Journal*, 19(1–2):1–81, 1995.

[2] W.L. Buntine, L. Su, and A.R. Newton. Adaptive methods for netlist partitioning. In *IEEE/ACM Int. Conference on Computer Aided Design*, 1997.

[3] S. Hauck and G. Borriello. An evaluation of bipartitioning techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16:849–866, 1997.

[4] G. Karypis, R. Aggrawal, V. Kumar, and S. Shekhar. Multilevel hyper-graph partitioning: Application in vlsi domain. In *Proc. Design Automation Conference*, pages 526–529, 1997.

[5] J. Li, J. Lillis, and C.-K. Cheng. Linear decomposition algorithm for VLSI design applications. In *IEEE International Conference on Computer-Aided Design*, pages 223–228, 1995.

Building a Basic Block Instruction Scheduler with Reinforcement Learning and Rollouts

Amy McGovern, Eliot Moss, and Andrew G. Barto

University of Massachusetts, Amherst,

Although high-level code is generally written as if it were going to be executed sequentially, most modern computers exhibit parallelism in instruction execution using techniques such as the simultaneous issue of multiple instructions. To take the best advantage of multiple pipelines, a compiler employs an instruction scheduler to reorder the machine code. Building this instruction scheduler is a large-scale optimization problem. Because schedulers are specific to the architecture of each machine, and the general problem of scheduling instructions is NP-Complete, schedulers are currently hand-crafted using heuristic algorithms. Building algorithms to select and combine heuristics automatically using machine learning techniques can save time and money. As computer architects develop new machines, new schedulers would be built automatically to test design changes rather than requiring hand-built schedulers for each change. This would allow architects to explore the design space more thoroughly and to use more accurate metrics in evaluating designs.

We formulated and tested two methods for automating the design of instruction schedulers: one uses rollouts, the other uses reinforcement learning (RL). We also investigated a combination of these two methods. Rollouts evaluate schedules online during compilation, whereas RL trains on more general programs and runs more quickly at compile time. Both types of schedulers use a greedy algorithm to build schedules sequentially with no backtracking (*list scheduling*).

We focused on scheduling *basic blocks* of instructions on the 21064 version [2] of the Compaq Alpha processor [6]. A basic block is a set of machine instructions with a single entry point and a single exit point. It does not contain any branches or loops. Our schedulers reorder the machine instructions within a basic block but cannot rewrite, add, or remove instructions. The goal of the scheduler is to find an ordering of the instructions in a block that preserves the semantically necessary ordering constraints of the original code while minimizing the execution time of the block.

The 21064 is a dual-issue machine with two execution pipelines. Compaq has made a 21064 simulator publicly available that also includes a heuristic scheduler for basic blocks, which we refer to as *DEC* (to maintain consistency with our earlier papers). The simulator gives the running time of a scheduled block assuming all memory references hit the cache and all resources are available at the beginning of the block. We also ran the schedules on a cluster of Compaq Alpha 21064 machines to obtain actual run-time results.

We tested each scheduling algorithm on the 18 SPEC95 benchmark programs (Reilly, 1995). Ten of these are FORTRAN programs, containing mostly floating point calculations, and eight are C programs, focusing more on integer, string, and pointer calculations. Each was compiled using the commercial Compaq compiler at the highest level of optimization. We call the schedules output by the compiler *COM*. A more detailed description of the problem and experimental setup can be found in [4]. As a performance measure, we used the ratio of a weighted execution time of the scheduler being assessed to a weighted execution time of DEC, where the weight of each block is the number of times that block is executed. This ratio is less than one whenever a scheduler produced a faster running time than DEC.

ROLLOUT SCHEDULING

Rollout scheduling works like this: given a set of candidate instructions to add to a partial schedule, the scheduler appends each candidate to the partial schedule and then follows a fixed policy, π, to schedule the remaining instructions. The running time of each completed schedule is determined. After rolling out each candidate (repeatedly for stochastic policies), the scheduler selects the instruction with the best estimated running time. (Rollouts were used in backgammon by Woolsey, 1991, Galperin, 1994, and Tesauro and Galperin, 1996).).

Bertsekas *et al.* (1997) proved that if we used the DEC scheduler as π, we would perform no worse than DEC, but an architect proposing a new machine would not have such a good heuristic policy available. Therefore, we considered rollouts using (1) the random policy, denoted *RANDOM-π*, in which a rollout makes all choices in a valid but uniformly random fashion (20 rollouts per instruction); (2) the the optimizing compiler COM, denoted *COM-π*; and (3) the DEC scheduler itself, denoted *DEC-π*. (For the latter two, only one rollout per instruction was needed since each is deterministic). As a baseline scheduler, we also scheduled each block with a valid but uniformly random ordering, denoted RANDOM.

The following table summarizes the performance of the rollout scheduler for each policy as compared DEC on all 18 benchmark programs for both the simulator and the actual execution times. All numbers are geometric means of the performance measure over 30 runs of each benchmark. Ratios less than one (italics) mean that the scheduler outperformed DEC.

	RANDOM		**RANDOM-π**		**COM**		**COM-π**		**DEC-π**	
	Sim	**Real**	**Sim**	**Real**	**Sim**	**Real**	**Sim**	**Real**	**Sim**	**Real**
Fortran geometric mean:	1.417	1.112	1.093	1.050	1.040	1.003	1.008	1.000	*0.987*	1.001
C geometric mean:	1.123	1.028	1.003	*0.996*	1.017	*0.991*	*0.995*	*0.994*	*0.991*	1.008
Overall geometric mean:	1.278	1.074	1.052	1.026	1.030	*0.998*	1.002	*0.998*	*0.989*	1.004

As expected, the random scheduler performed very poorly (27.8% slower than DEC for simulation mode). In contrast, RANDOM-π came within 5% of the running time of DEC. COM was only 3% slower than DEC and outperformed DEC on two applications; and COM-π outperformed DEC on 6 applications. The DEC-π scheduler was able to outperform DEC on all applications, producing schedules that ran about 1.1% faster than those produced by DEC. Although this improvement may seem small, the DEC scheduler is known to make optimal choices 99.13% of the time for blocks of size 10 or less [7]. The actual performance of the binaries built using each of our schedulers was similar to the performance in the simulator, although with smaller differences, showing that the assumptions that DEC makes for the simulator can be detrimental on the actual machine.

Although the performance of the rollout scheduler can be excellent, rollouts are inherently computationally expensive. Rollouts can be used to optimize schedules for important blocks (those with long running times or which are frequently executed) within a program but not for scheduling large programs unless computation time improves. With the performance and the timing of the rollout schedulers in mind, we looked to RL to obtain high performance at faster running times.

REINFORCEMENT LEARNING

We used a *temporal difference* (TD) algorithm to estimate the value function of the current scheduling policy. At the same time, we used this estimate to update the current policy. This results in a kind of generalized policy iteration (Sutton and Barto, 1998) that tends to improve the policy over time. Instead of learning the direct value of choosing instruction A or instruction B, our RL system learned a preference function between candidate instructions: it learned the difference of the returns resulting from choosing instruction A over instruction B. The preference function was represented as a weighted sum of a set of feature values describing the current partial schedule and two candidate instructions. Each feature was derived from knowledge of the DEC simulator. At each choice point during learning, the RL scheduler chooses the most preferred action according to the current value function with a high probability, and otherwise chooses a random but valid instruction. The reward was zero until a block was scheduled, in which case it was a measure of how well the schedule outperformed DEC, normalized by the block size. A discussion of how well other reward functions performed can be found in [4].

We trained the RL scheduler for 100 epochs on the application *compress95*, and we used the best resulting value function to schedule the other 17 benchmarks. The results are shown below. As before, we also tested the schedules on Compaq Alphas.

Performance using the DEC reward function											
Fortran programs						**C programs**					
App	**Sim**	**Real**	**App**	**Sim**	**Real**	**App**	**Sim**	**Real**	**App**	**Sim**	**Real**
applu	1.094	1.017	apsi	1.090	1.029	cc1	1.023	1.007	compress95	*0.991*	*0.967*
fpppp	1.086	1.038	hydro2d	1.036	1.002	go	1.034	*0.925*	ijpeg	1.025	1.031
mgrid	1.410	1.132	su2cor	1.035	1.012	li	1.016	1.004	m88ksim	1.012	*0.983*
swim	1.569	1.007	tomcatv	1.061	1.028	perl	1.022	*0.997*	vortex	1.035	*0.977*
turb3d	1.145	1.016	wave5	1.089	*0.994*	C geometric mean:				1.020	*0.986*
Fortran geometric mean:				1.151	1.027	Overall geometric mean:				1.090	1.009

By training the RL scheduler on compress95 for 100 epochs, we were able to outperform DEC on compress95. The RL scheduler came within 2% of the performance of DEC on all C applications and within 15% on unseen Fortran applications. Although the Fortran performance is not as good as that on the C applications, the RL scheduler has more than halved the difference between RANDOM and DEC. This demonstrates good generalization across basic blocks. Although there are benchmarks that perform much more poorly than the rest (mgrid and swim), those benchmarks perform more than 100% worse than DEC under the RANDOM scheduler. In actual execution of the learned schedules, the RL scheduler outperformed DEC on the C applications while coming within 2% of DEC on the Fortran applications. We also experimented with training the RL scheduler on the Fortran program applu. By doing so, the simulated performance on Fortran applications improved from 15% slower than DEC to only 8% slower than DEC. At the same time, performance on unseen C programs slowed by slightly less than 1%.

Combining Reinforcement Learning with Rollouts

We also experimented with a scheduler, RL-π, that used the value function learned by RL on compress95 as the rollout policy. The results are shown below.

Fortran programs						**C programs**					
App	**Sim**	**Real**	**App**	**Sim**	**Real**	**App**	**Sim**	**Real**	**App**	**Sim**	**Real**
applu	1.017	*0.999*	apsi	1.015	1.012	cc1	1.000	1.005	compress95	*0.974*	1.014
fpppp	1.021	*0.992*	hydro2d	1.006	*0.999*	go	1.001	*0.991*	ijpeg	*0.986*	*0.986*
mgrid	1.143	1.027	su2cor	1.006	1.010	li	*0.995*	1.005	m88ksim	*0.998*	*0.979*
swim	1.176	1.003	tomcatv	1.035	*0.999*	perl	*0.996*	*0.981*	vortex	1.001	*0.984*
turb3d	1.039	*0.988*	wave5	1.025	*0.992*	C geometric mean:				*0.994*	*0.993*
Fortran geometric mean:				1.047	1.002	Overall geometric mean:				1.023	*0.998*

By adding only one rollout, we were able to improve the C results to be faster than DEC overall. The Fortran results improved from 15% slower to only 4.7% slower than DEC. When executing the binaries from the RL-π schedules, a user would see slightly faster performance than DEC.

Conclusions

We have demonstrated two successful methods of building instruction schedulers for straight line code. The first method, using rollouts, was able to outperform a commercial scheduler both in simulation and in actual run-time results. The downside of using a rollout scheduler is its inherently slow running time. By using an RL scheduler, we were able to maintain good performance while significantly reducing scheduling time. Finally, we showed that a combination of RL and rollouts was able to compete with the commercial scheduler. In a system where multiple architectures are being tested, any of these methods could provide a good scheduler with minimal setup and training.

Acknowledgments

This work was supported in part by the National Physical Science Consortium, Lockheed Martin, Advanced Technology Labs, AFOSR grant F49620-96-1-0234 to Andrew G. Barto, and NSF grant IRI-9503687 to Roderic A. Grupen and Andrew G. Barto. We also thank Sun Microsystems, Hewlett-Packard, and Compaq/Digital Equipment Corporation for their support.

REFERENCES

[1] Bertsekas, D. P., Tsitsiklis, J. N. & Wu, C. (1997). Rollout algorithms for combinatorial optimization. *Journal of Heuristics.*

[2] DEC (1992). *DEC chip 21064-AA Microprocessor Hardware Reference Manual* (first edition Ed.). Maynard, MA: Digital Equipment Corporation.

[3] Galperin, G. (1994). Learning and improving backgammon strategy. In *Proceedings of the CBCL Learning Day.* Cambridge, MA.

[4] McGovern, A., Moss, E. & Barto, A. G. (1999). Building a basic block instruction scheduler with reinforcment learning and rollouts. *Machine Learning.* Accepted to appear.

[5] Reilly, J. (1995). SPEC describes SPEC95 products and benchmarks. SPEC Newsletter.

[6] Sites, R. (1992). *Alpha Architecture Reference Manual.* Maynard, MA: Digital Equipment Corporation.

[7] Stefanović, D. (1997). The character of the instruction scheduling problem. University of Massachusetts, Amherst.

[8] Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning. An Introduction.* Cambridge, MA: MIT Press.

[9] Tesauro, G. & Galperin, G. R. (1996). On-line policy improvement using Monte-Carlo search. In *Advances in Neural Information Processing: Proceedings of the Ninth Conference.* MIT Press.

[10] Woolsey, K. (1991). Rollouts. *Inside Backgammon, 1(5)*, 4–7.

"STAGE" Learning for Local Search

Justin A. Boyan † and Andrew W. Moore ‡

† NASA Ames Research Center; ‡ Carnegie-Mellon University,

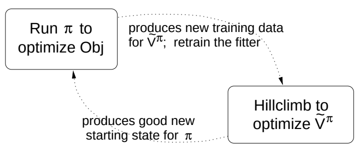
Stage is a machine-learning algorithm for accelerating the performance of local search. Collecting data from sample search trajectories, Stage builds an auxiliary evaluation function which is then used to bias future search trajectories toward better optima. The algorithm is described only briefly here; for fuller descriptions please see [3, 1]. Other algorithms related to STAGE that are also described in this survey include the contributions of Moll et al. and Su and Buntine.

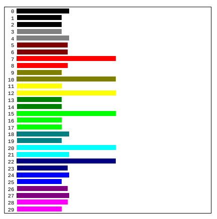
The auxiliary evaluation function that Stage builds is an approximation to this predictive function:

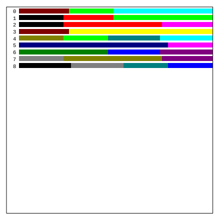
$$V^\pi(x) \overset{\text{def}}{=} \text{expected best Obj value seen on a trajectory that starts from state } x \text{ and follows local search method } \pi$$

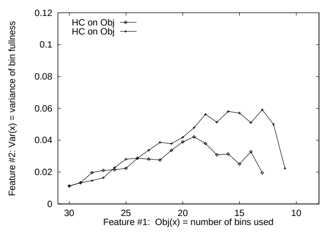
Here, π represents a local search method such as hillclimbing or simulated annealing, and $\text{Obj} : X \to \Re$ is the objective function which we would like to minimize. From a reinforcement learning perspective, π can be seen as a policy for exploring a Markov Decision Process, and V^π is the *value function* of that policy: it predicts the eventual expected outcome from every state. Thus, well-studied learning algorithms such as TD(λ) [7, 2] may be applied to approximate V^π from sampled search trajectories. Here, we approximate V^π using a regression model, where states x are encoded as real-valued feature vectors. (Such features are plentiful in real-world applications.) We denote the mapping from states to features by $F : X \to \Re^D$, and our approximation of $V^\pi(x)$ by $\tilde{V}^\pi(F(x))$.

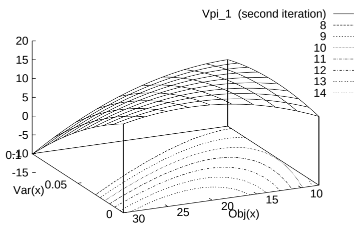
The approximate value function $\tilde{V}^\pi(F(x))$ evaluates the *promise* of state x as a starting point for algorithm π. To find the most promising starting point, then, we must optimize $\tilde{V}^\pi$ over X. Stage does this by applying hillclimbing with $\tilde{V}^\pi$ instead of Obj as the evaluation function. As illustrated in Figure 7a, Stage repeatedly interleaves two different stages of local search: running the original method π on Obj, and running hillclimbing on $\tilde{V}^\pi$ to find a promising new starting state for π. Thus, Stage can be viewed as a *learning multi-restart* approach to local search.

(7a)

Run π to optimize Obj
produces new training data for Ṽπ; retrain the fitter
Hillclimb to optimize Ṽπ
produces good new starting state for π

(7b)


(7c)


(7d)


(7e)


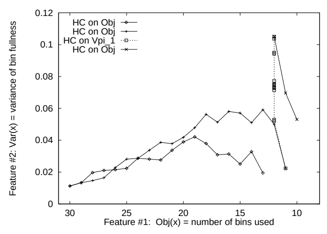
(7f)


Figure 7: Stage working on the bin-packing example

We illustrate Stage's operation with a small example from the NP-complete domain of bin-packing [4], shown in Figure 7b. Packed optimally, this set of 30 items fills 9 bins exactly to capacity (Figure 7c). We define a local-search operator which moves a single random item to a random new bin with sufficient spare capacity, and we define two state features $F(x)$ for use by Stage:

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 [5].)

Two trajectories of stochastic hillclimbing in this 2-D feature space are plotted in Figure 7d. Both trajectories start at the initial state where each item is in its own bin $(\mathrm{Obj} = 30, \mathrm{Var} = 0.011)$, and they end at different local optima: $(\mathrm{Obj} = 13, \mathrm{Var} = 0.019)$ and $(\mathrm{Obj} = 11, \mathrm{Var} = 0.022)$, respectively. STAGE, trained by quadratic regression to predict the observed outcomes 13 and 11 from those two trajectories, learns the approximate value function shown in Figure 7e. Note how the contour lines shown on the base of the surface plot correspond to smoothed versions of the training trajectories. Extrapolating, $\tilde{V}^\pi$ predicts that the the best starting points for hillclimbing are on arcs with higher Var(x).

STAGE switches to the auxiliary evaluation function $\tilde{V}^\pi$ and hillclimbs to try to find a good new starting point. The resulting trajectory, shown as a dashed line in Figure 7f, goes from $(\mathrm{Obj} = 11, \mathrm{Var} = 0.022)$ up to $(\mathrm{Obj} = 12, \mathrm{Var} = 0.105)$. Note that the search was willing to accept some harm to the true objective function during this stage. From the new starting state, hillclimbing on Obj does indeed lead to a yet better local optimum at $(\mathrm{Obj} = 10, \mathrm{Var} = 0.053)$. During further iterations, the approximation of $\tilde{V}^\pi$ is further refined, and STAGE manages to discover the global optimum at $(\mathrm{Obj} = 9, \mathrm{Var} = 0)$ on iteration seven.

RESULTS

Extensive experimental results are given in Table 4. We constrast the performance of STAGE with that of multi-start stochastic hillclimbing, simulated annealing, and domain-specific algorithms where applicable, on six domains:

Bin-packing. As in the example above, but with a 250-item benchmark instance.

Channel routing. Lay out wires so as to minimize the width of a channel in VLSI.

Bayes Net Structure-Finding. Find the graph structure that best captures the dependencies among the attributes of a data set.

Radiotherapy Treatment Planning. Produce a treatment plan that meets target radiation doses for a tumor while minimizing damage to sensitive nearby structures. (Experiments were conducted on a simplified 2-D version of the problem.)

Cartogram Design. For geographic visualization purposes, redraw a map of the USA so that the states' areas are proportional to population, while minimally deforming the overall shape.

Boolean Satisfiability. Minimize the number of unsatisfied clauses of a Boolean formula expressed in CNF. Here, WALKSAT [6] rather than hillclimbing was used as the baseline local search procedure for STAGE's learning.

On each instance, all algorithms were held to the same number M of total search moves considered, and run N times. Full details of these experiments may be found in [1]. The results, summarized in Table 4, indicate that STAGE always learned to outperform the baseline local search method on which it was trained, and usually outperformed simulated annealing as well.

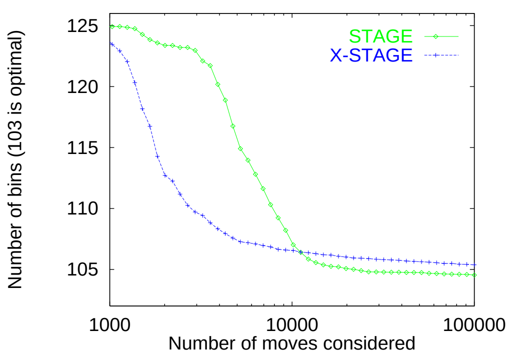
TRANSFER

In the above experiments, the computational cost of training a function approximator on V^π was minimal—typically, 0–10% of total execution time. However, STAGE's extra overhead would become significant if many

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
$M = 10^5, N = 100$	Best-Fit Randomized	106.78± 0.08	106	107
	STAGE, quadratic regression	**104.77**± 0.09	**103**	**105**
Channel routing	Hillclimbing, patience=250	22.35± 0.19	20	24
(YK4, opt=10)	Simulated annealing, $\text{Obj}(x) = w$	14.32± 0.10	13	15
$M = 5 \cdot 10^5, N = 100$	STAGE, linear regression	**12.42**± 0.11	**11**	**14**
Bayes net	Hillclimbing, patience=200	440567± 52	439912	441171
(ADULT2)	Simulated annealing	440924± 134	**439551**	444094
$M = 10^5, N = 100$	STAGE, quadratic regression	**440432**± 57	439773	**441052**
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
$M = 10^6, N = 100$	STAGE, quadratic regression	0.056±0.003	0.038	**0.132**
Satisfiability	WALKSAT + $\delta_w = 0$ (hillclimbing)	690.52± 1.96	661	708
(par32-1.cnf, opt=0)	WALKSAT, noise=0, cutoff=10^6, tries=100	15.22± 0.35	9	19
$M = 10^8, N = 100$	STAGE(WALKSAT), quadratic regression	**5.36**± 0.33	**1**	9

Table 4: Comparative results on a variety of minimization domains. For each problem, all algorithms were allowed to consider the same fixed number of moves M. Each line reports the mean, 95% confidence interval of the mean, best, and worst solutions found by N independent runs of one algorithm on one problem. STAGE was best on average (boldfaced) in five of six domains.

more features or more sophisticated function approximators were used. For some problems such cost is worth it in comparison to a non-learning method, because a better or equally good solution is obtained with overall less computation. But in those cases where we use more computation, the STAGE method may nevertheless be preferable if we are then asked to solve further similar problems (e.g., a new channel routing problem with different pin assignments). Then we can hope that the computation we invested in solving the first problem will pay off in the second, and future, problems because we will already have a $\tilde{V}^\pi$ estimate. This effect is termed *transfer*.

(a)


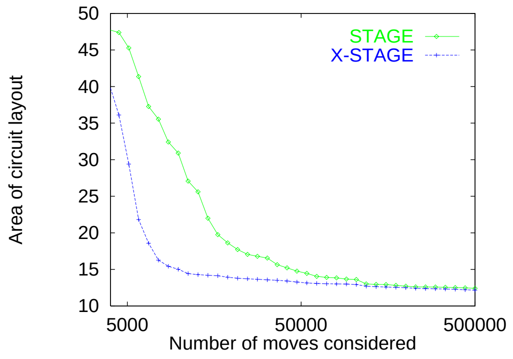
(b)


Figure 8: Optimization performance with transfer (X-STAGE) and without transfer (STAGE) on bin-packing (a) and channel routing (b). Note the logarithmic scale of the x-axis.

We investigated the potential for transfer in local search with a modified algorithm called X-STAGE. The X-STAGE algorithm uses a simple voting mechanism to combine arbitrarily many previously trained $\tilde{V}^\pi$ functions; full details of the voting mechanism are given in [1]. Figure 8 shows performance curves illustrating transfer in the domains of bin-packing and channel routing. In the bin-packing experiment (a), X-STAGE combined 19 previously trained $\tilde{V}^\pi$ functions; it 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 on the new instance. In the channel-routing experiment (b), X-STAGE combined 8 previously trained $\tilde{V}^\pi$ functions. This time, the voting-based restart policy maintained 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 profitably transferred to novel problem instances. Future work will 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.

DISCUSSION

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

REFERENCES

[1] J. A. Boyan. *Learning Evaluation Functions for Global Optimization.* PhD thesis, Carnegie Mellon University, 1998.

[2] J. A. Boyan. Least-squares temporal difference learning. In *Machine Learning: Proceedings of the Sixteenth International Conference (ICML)*, 1999. (Best Paper Award).

[3] J. A. Boyan and A. W. Moore. Learning evaluation functions for global optimization and Boolean satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*, 1998. (Outstanding Paper Award).

[4] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, 1996.

[5] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proc. of the IEEE 1992 International Conference on Robotics and Automation*, pages 1186–1192, Nice, France, May 1992.

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

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

Enhancing Discrete Optimization with Reinforcement Learning: Case Studies Using DARP

Robert Moll, Theodore J. Perkins, and Andrew G. Barto

University of Massachusetts, Amherst,

Introduction

Reinforcement learning methods can be used to improve the performance of algorithms for discrete optimization by learning evaluation functions that predict the outcome of search. In this study we use reinforcement learning (RL) for developing good solutions to a particular NP-complete logistics problem, namely the Dial-A-Ride Problem (DARP), a variant of the better-known traveling salesman's problem. DARP is a useful problem for study because the space of feasible solutions to a DARP instance has a nonuniform structure which is nevertheless coherent, in the sense that relatively simple and easily computable features behave similarly for all instances of all sizes, and can therefore be combined to form a reasonably accurate instance-independent optimal value function approximation.

We summarize our technique as follows. Using the TD(λ) algorithm in an off-line learning phase, a value function is learned for DARP which estimates performance along local search trajectories in the space of feasible solutions. Because of the general coherence of the Dial-A-Ride problem, the resulting value function is effective as a local search cost function for all DARP instances of all sizes. We believe our methodology is broadly applicable to many combinatorial optimization problems. Our research on enhancement techniques for local search combines aspects of previous work by Zhang and Dietterich [5], Boyan and Moore [1], Boyan [2], Healy and Moll [3], and Healy [4].

The Dial-A-Ride Problem

DARP has the following formulation. A van is parked at a terminal. The driver receives calls from N customers who need rides. Each call identifies the location of a customer, as well as that customer's destination. The van must be routed so that it starts from the terminal, visits each customer pick-up and drop-off site, and then returns to the terminal. For a tour to be feasible, every pick-up site must precede its paired drop-off site. The van has unlimited capacity. The objective of the problem is to minimize tour length.

We impose a neighborhood structure on the space of feasible solutions to a DARP instance. If s is a legal tour, we write $A(s)$ for the neighbors of s. Most prominent in our work is the "2-opt" neighborhood structure of [7], in which two tours are neighbors if the first can be transformed into the second by reversing a subsequence of site visits in the first tour. This neighborhood structure is highly non-uniform: neighborhood size varies between $O(N)$ and O(N^2). A "3-opt" neighborhood structure also makes sense for DARP [8], and the "3-opt" algorithm of [7], suitably modified, is extremely effective but very slow.

Following [5, 1, 3], we note that secondary characteristics of feasible solutions can provide valuable information for search algorithms. From the point of view of local seach, such characteristics can be important components of a generalized cost function for hill-climbing. A value function, constructed using RL, and associating, with every state, an estimate of performance along a local search trajectory starting from that state, is precisely such a generalized cost function. Indeed, by adjusting the parameters of a function approximation system whose inputs are feature vectors describing feasible solutions, a computationally tractable RL algorithm can produce a compact representation of such an approximate optimal value function V.

Elaborating on this scheme, our approach operates in two phases. In the *learning phase*, a value function is learned by applying the TD(λ) algorithm to several thousand suitably normalized randomly chosen instances of the problem. In the *performance phase*, the resulting value function, now held fixed, is used to guide local search for additional problem instances. This approach is in principle applicable to any suitably coherent combinatorial optimization problem.

Enhanced 2-opt for DARP

In the learning phase identifed above, we conduct training episodes until we are satisfied that the function approximator's weighting scheme has stabilized. For each episode we select a problem size N at random (from a predetermined range) and generate a random DARP instance of that size, i.e., we form a symmetric Euclidean distance matrix by generating random points in the plane inside the square bounded by the points (0,0), (0,100), (100,100) and (100,0). We set the "terminal site" to point (50,50) and the initial tour to a randomly generated feasible tour. We then conduct a modified first-improvement 2-opt local search using the negated current value function, $-V_k$, as the cost function. The modification is that termination is controlled by a nonnegative parameter ϵ as follows: the search terminates at a tour s if there is no $s' \in A(s)$ such that $V_k(s') > V_k(s) + \epsilon$. In other words, a step is taken only if it produces an improvement of at least ϵ according to the current value function. The episode returns a final tour s_f. We next run one unmodified 2-opt local search, this time using the standard DARP cost function c (tour length), from s_f to compute 2-opt(s_f). We then apply a batch version of undiscounted TD(λ) to the saved search trajectory using the following immediate rewards: $-\epsilon$ for each transition, and $-c(\text{2-opt}(s_f))/Stein_N$ as a terminal reward, where $Stein_N$ is the estimated optimal tour for size N as calculated theoretically by Stein in [9].

We can now use the learned value function V in the performance phase, which consists of applying to new instances the modified first-improvement 2-opt local search with cost function $-V$, followed by a 2-opt application to the resulting tour. The results described here were obtained using a simple linear approximator with a bias weight and three features: normalized cost; normalized neighborhood size; and a third feature we call *proximity*, which reflects the positioning of the $N/4$ least expensive pairs of sites in a DARP tour.

Comparisons among algorithms were done at five representative sizes, $N = 20, 30, 40, 50$, and 60. For the learning phase, we conducted approximately 3,000 learning episodes, each one using a randomly generated instance of a size selected randomly between 20 and 60 inclusive. The result of the learning phase was a reasonably stable value function V. Table 5 compares the tour quality found by six different local search algorithms. For the algorithms using learned value functions, the results are for the performance phase after learning using the algorithm listed. Table entries are the percent by which tour length exceeded $Stein_N$ for instance size N, averaged over 100 instances of each representative size. Thus, 2-opt exceeded $Stein_{20} = 645$ on the 100 instance sample set by an average of 42%. The last row in the table gives the results of using the five different value functions V_N, for the corresponding N. Results for TD(λ) appeared to be best with $\lambda = .8$. The learning-enhanced algorithms do well against 2-opt when running time is ignored, and indeed TD(.8), $\epsilon = 0$, is about 35% percent better (according to this measure) by size 60. Note that 3-opt clearly produces the best tours, and a non-zero ϵ for TD(.8) decreases tour quality, as expected since it causes shorter search trajectories.

Table 5: Comparison of Six Algorithms at Sizes $N = 20, 30, 40, 50, 60$. Entries are percentage above $Stein_N$ averaged over 100 random instances of size N.

Algorithm	N=20	N=30	N=40	N=50	N=60
2-opt	42	47	53	56	60
3-opt	8	8	11	10	10
TD(1)	28	31	34	39	40
TD(.8) $\epsilon = 0$	27	30	35	37	39
TD(.8) $\epsilon = .01/N$	29	35	37	41	44
TD(.8) $\epsilon = 0$, V_N	29	30	32	36	40

The algorithm TD(.8) $\epsilon = .01/N$ ran between 2 and 3 times longer than traditional 2-opt on problem instances in the range N=20-60; TD(.8) $\epsilon = 0$ ran 3 to 7 times slower. When performance is equalized for time, both algorithms still outperform traditional 2-opt, and by size N=60 these algorithms are 30-40% better. Thus a methodology that constructs a learned value function involving secondary problem characteristics, and uses the value functiona as a generalized cost function for local search, can significantly enhance local search performance for combinatorial optimization problems.

Other DARP Case Studies

We also investigated two other learning-based enhancements to combinatorial optimization algorithms, again using DARP as our test problem. We considered the rollout method [13, 12, 10], and we used it to extend a very effective constructive DARP algorithm developed by Kubo and Kasugai [14]. Although our rollout extension is extremely long-running, it significantly outperforms the best algorithm reported in [14]. Indeed even a drastically truncated rollout algorithm outperforms the Kubo-Kasugai algorithm at small problem sizes.

Finally we considered a variation of the STAGE algorithm [1, 2] called the Expected Improvement Algorithm, which uses the same control structure as STAGE, but which learns a different hill-climbing function—one that seeks to maximize the expected improvement over the best-so-far solution, rather than just the expected value of hill-climbing. Our results here are preliminary and inconclusive, but we believe that this approach shows promise as yet another learning-based technique for combinatorial optimization.

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

References

[1] J.A. Boyan, and A.W. Moore (1997). Using Prediction to Improve Combinatorial Optimization Search, Proceedings of AI-STATS-97.

[2] J.A. Boyan (1998). Learning Evaluation Functions for Global Optimization. Ph.D. Thesis, Carnegie-Mellon University.

[3] P. Healy and R. Moll (1995). A New Extension to Local Search Applied to the Dial-A-Ride Problem. *EJOR*, 8: 83-104.

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

[5] W. Zhang, and T.G. Dietterich (1995). A Reinforcement Learning Approach to Job-Shop Scheduling, Proceedings of the 14th IJCAI, pp. 1114-1120. Morgan Kaufmann, San Francisco.

[6] R. Moll, A.G. Barto, T.J. Perkins, R. S. Sutton (1998). Learning Instance-Independent Value Functions to Enhance Local Search, Proceedings of NIPS-98. Denver.

[7] S.Lin, and B.W. Kernigham and (1973). An Efficient Heuristic for the Traveling-Salesmen Proglem. *OR*, 21: 498-516. 291-307.

[8] Psaraftis, H. N. (1983). K-interchange Procedures for Local Search in a Precedence-Constrained Routing Problem.*EJOR*, 13:391–402.

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

[10] Tesauro, G., and Galperin, G. R. (1996). On-line Policy Improvement using Monte-Carlo Search. In *Advances in Neural Information Processing: Proceedings of the Ninth Conference*. MIT Press.

[11] Bertsekas, D. P., and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

[12] Bertsekas, D. P. (1997). Differential Training of Rollout Policies. In *Proc. of the 35th Allerton Conference on Communication, Control, and Computing.* Allerton Park, Ill.

[13] Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C. (1997). Rollout Algorithms for Combinatorial Optimization. *Journal of Heuristics.*

[14] Kubo, M. and Kasugai, H. (1990). Heuristic Algorithms for the Single Vehicle Dial-A-Ride Problem. *Journal of Operations Research*, 33:354–364.

Stochastic Optimization with Learning for Standard Cell Placement

Lixin Su, Wray Buntine, and A. Richard Newton

University of California at Berkeley,

Introduction

Stochastic combinatorial optimization techniques, such as simulated annealing [2] and genetic algorithms [3], have become increasingly important in design automation [4, 5] as the size of design problems have grown and the design objectives have become increasingly complex. In addition, as we move towards deep-sub-micron technologies, the cost function must often evolve over time or handle a variety of tradeoffs between area, power, and timing, for example. Design technologists can often tackle simplified versions of some of these problems, where the objective can be stated in terms of a small number of well-defined variables, using deterministic algorithms. Such algorithms may produce as good or even better results than the stochastic approaches in a shorter period of time. Unfortunately, these algorithms run into a variety of difficulties as the problems scale up and the objective functions begin to capture the real constraints imposed on the design, such as complex timing, power dissipation, or test requirements. Stochastic algorithms are naturally suited to these larger and more complex problems since they are very general, making random perturbations to designs and, in a controlled way, letting a cost function determine whether to keep the resulting change.

However, stochastic algorithms are often slow since a large number of random design perturbations are required to achieve an acceptable result. They have no built-in intelligence and no ability to adapt their performance in a particular problem domain. The goal of this research was to determine whether statistical learning techniques can improve the run-time performance of stochastic optimization for a particular solution quality, and in [1, 14] we demonstrated that for the problems we considered, the adaptive approach offers a significant improvement.

In our previous work presented in [1], we used a one-time, regression-based approach to train the stochastic algorithm. We presented results for simulated annealing as representative stochastic optimization approach. The standard-cell-based layout placement problem was selected to evaluate the utility of such a learning-based approach, since it is a very well explored problem using both deterministic [6, 7, 8, 9, 10] as well as "manually trained" stochastic approaches [11, 12, 13]. In our current work, we extended our approach to incremental learning and we reported detailed results for a regression-based empirical, incremental approach to learning in [14].

Stochastic placement algorithms have evolved significantly since their initial application in the EDA area over fifteen years ago [2]. Over that period, the qualities of results they can produce have improved significantly. For example, in the development of TimberWolf system [11, 12, 13], which is a general-purpose placement and routing package based on simulated annealing, many techniques have been tried to speed up the algorithm. They include reducing the computation time of each move, early rejection of bad moves, the use of efficient and adaptive cooling schedules combined with windowed sampling, and hierarchical or clustered annealing. In many ways, these variations and improvements can be viewed as "manually learned" approaches, based on the application of considerable experimental as well as theoretical work taking place over a long period of time. Commercial developments and other university-based work have also shown significant improvements over the early work in this area. In our work, we explore another opportunity for improving the utility of a stochastic algorithm through automatic learning of the relative importance of various criteria in the optimization strategy. We learn from previous annealing runs to distinguish potentially good moves from bad ones. The good ones will be selected with a higher probability to expand the search.

Technical Background and Approaches

We assume that the readers are already familiar with the simulated annealing algorithm. In the algorithm, any perturbation of the current solution is called a move. A move can be either accepted or rejected depending on the Boltzmann test. In Conventional Simulated Annealing (CSA), proposing of a move is totally random.

The primary goal of standard cell placement problem is to find positions on the chip for all the cells in a net-list so that the estimated total wire length is minimized. A net-list is the result of logic synthesis in the ASIC design flow. In the case of standard cell design, it represents a set of standard cells and their logical connections (nets). As a constraint, the cells cannot overlap with each other in the final placement. For the sake of simplicity, we also assume all the cells are of the same size in our experiments.

In most placement algorithms, the cost function, which is the estimate of the final total wire length, is either linear or quadratic. As a common practice with simulated annealing, we define the cost function as the sum of half perimeters of bounding boxes of all the nets. The move set is defined as the pair-wise swaps.

We defined a feature vector of a swap as a real vector of seven components. Our learning machine is a linear function of the feature vector. The parameters in the linear function were determined by linear regression, either in a batch-mode [1] or incrementally [14].

The learned regression model was then used as an evaluation function judging the "goodness" of a swap. More specifically, a randomly chosen set, which was a sub-set of the move set was formed first, and then the "best" move selected from the chosen set using the evaluation function was given to the Boltzmann test. The simulated annealing with this modification is called Trained Simulated Annealing (TSA). If the evaluation function is updated from run to run, it is called Incrementally Trained Simulated Annealing (ITSA); otherwise if the evaluation function is trained in a batch-mode, it is called Batch-Mode Simulated Annealing (BTSA).

Experimental Results

We did our placement experiments on a set of circuits taken from MCNC91 combinational benchmark set [15], NCSU benchmark set [16] and ISCA89 sequential benchmarks [15]. The net-lists were synthesized using SIS. For convenience, the MSU standard-cell library was used for generating the cell layouts.

Batch-Mode Learning

The test set was divided into two groups, Group 1 was used for the construction of regression models; while Group 2 was used for the blind test.

As an initial test of the approach and to provide a control for later comparisons, we constructed individual models for each of the circuits. The learning data were obtained by running CSA 5,000 times for each of the circuits. Each individual model was applied to the circuit on which it was developed by running TSA to see if the individual model is robust for the entire (much larger than the part each model was learned from) solution space.

In the second phase of the experiments, we used simple averaging of the parameters across the test circuits to build the overall general model. While there are more effective statistical approaches to the combination of the individual models, simple averaging can be seen as a simple and sub-optimal approach. The general (averaged) model was then applied to all the circuits in Group 1 by running TSA in order to see if the general model works as well as the circuit's own individual model. Our experiment showed that TSA with both the individual and the general model works equally well. For the same number of moves proposed to the Boltzmann test, the annealing quality returned by TSA is at least 15-43% better than that returned by CSA.

Next we tested the generality of the general model in a more significant way. The general model was applied to circuits in Group 2, none of which were used in the training of the model. For the same number of tested moves, TSA with the general trained model improved the annealing quality by 7- 41% compared to CSA. Notice that quite a few circuits in Group 2 are 1-3 times larger than the largest circuit in the Group 1 training set.

To see the CPU time versus annealing quality trade off, we changed the number of swaps proposed per temperature to control the run time. Despite the overhead introduced in TSA, for the same amount of CPU time, the annealing quality was improved from 12 to 22% for all the Group 2 circuits using TSA in contrast to using CSA. For Group 1 circuits, the best percentage improvement of final quality ranged from 14% to 28%.

Over the years, researchers have found that a windowing approach, where the maximum distance between cells in the candidate move is decreased as temperature decreases, tends to improve the overall run-time performance of the annealing at little or no cost in the quality of the final result. Our experiment also discovered that in the case of TSA, the average cell distance of the proposed swap automatically decreased with the temperature; in contrast, in the case of CSA the average cell distance did not show any change. Hence the general trained model, derived purely from the training runs on the test examples and without any a priori hints, determined that a windowing approach would actually lead to an optimal utility result and even predicted the optimal window size. Our approach indeed has automatically learned something nontrivial!

Incremental Learning

First, we experimented with learning from run to run for a particular circuit. Since incremental learning must start with an initial model, three different initial evaluation functions were investigated: non-informative (NI), batch learned (BL), and weighted non-informative (WNI).

In the case of using the NI initial model, ITSA is very effective in the sense that it achieved 10-27% reduction (compared to CSA) in the average final cost function after a single new data point was used to update the initial evaluation function. Information added to the evaluation function in the later runs did not improve the annealing quality. But if we put less weight on the initial model, namely in the case of using WNI initial model, the percentage improvement at the end of the 3rd incremental annealing run was increased by up to 9.3% compared to that achieved in the former case. Similarly, after the 3rd incremental learning run, the annealing quality converged to an almost maximum improvement. With the BL initial model, the reduction compared to CSA in average final cost function at the end of the second run was also significant i.e. 2.69-33.45%, and even more, as the incremental learning went on, the reduction almost approached the value achieved by BTSA.

Next, we experimented with learning from circuit to circuit. It was shown that the model learned from one circuit can be safely applied to another circuit, and the improvement in annealing quality at the end of the second incremental learning run was even 1-5% better than its best counterpart when a model incrementally learned from the same circuit was used. It was also observed that initial evaluation functions learned from many other circuits outperformed the ones learned from only one other circuit by up to 8%. Hence the more hybridized the model, the better the annealing quality will be for new circuits. However, learning order did make small difference too.

Conclusions and Future Work

We demonstrated that a stochastic algorithm, in this case simulated annealing can significantly improve the quality-of-results on a mainstream EDA application through effective incremental learning.

In the case of batch-mode learning, the annealing quality improvement was 15-43% for the set of examples used in training and 7-21% when the trained algorithm was applied to new examples. With the same amount of CPU time, the trained algorithm improved the annealing quality by up to 28% for some benchmark circuits we tested. In addition, the use of the response model successfully predicted the effect of the windowed sampling technique and derived the informally accepted advantages of windowing from the test set automatically.

In the case of incremental learning, for a particular circuit, even at the end of the 2nd learning run, the annealing quality was improved by 10%-27% compared to conventional simulated annealing in the examples presented. This result was further increased by up to 10% by putting less emphasis on the initial value of the evaluation function. In contrast to batch-mode learning, where a large data set must be obtained for the training of a regression model from expensive learning runs, the non-informative initial model did not cost any effort and resulted in almost identical improvement. In the case of learning from circuit to circuit, our experiments showed that information learned from one circuit could be applied safely to another, yielding a slightly (up to 5%) better result compared to the best case of using a model learned from the same circuit. Moreover, learning from more circuits yielded even better results. However, the learning order did make a difference in terms of annealing quality. Overall, we believe that this work has demonstrated that a stochastic optimization algorithm, applied to at least some EDA problems, can significantly improve its performance by learning from its optimization history automatically. We believe the application of general-purpose stochastic algorithms, with built-in general-purpose approaches to learning, could eventually form the basis of a general and adaptive approach to the solution of a variety of VLSI CAD problems.

REFERENCES

[1] L. Su, W. Buntine, A. R. Newton, and B. S. Peters. Learning as Applied to Stochastic Optimization for Standard Cell Placement. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pages 622–627. 1998.

[2] S. Kirkpatrick, C. D. Gelatt, and Jr., M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–880, 1983. 1993.

[3] David Edward Goldberg. *Genetic algorithms in search.* Addison-Wesley, 1989.

[4] D. F. Wong, H. W. Leong and C. L. Liu. *Simulated annealing for VLSI design.* Kluwer Academic Publishers, 1988.

[5] Pinaki Mazumder, Elizabeth M. Rudnick. *Genetic algorithms for VLSI design, layout & test automation.* Prentice Hall, 1999.

[6] M. Hannan, P. K. Wolff and B. Agule. Some experimental results on placement technique. In *Proc. 12th Design Automation Conference*, pages 214–244. 1976.

[7] N. Quinn and M. Breuer. A force directed component placement procedure for printed circuit boards. *IEEE Trans. CAS*, CAS-26:377–388, 1979.

[8] R-S. Tsay, E. S. Kuh, and C.-P. Hsu. PROUD: A fast sea-of-gates placement algorithm. In *ACM/IEEE Design Automation Conference*, 1988.

[9] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Trans. CAD*, CAD-10:356–365, 1991.

[10] Hans Eisenmann and Frank M. Johannes. Generic global placement and floorplanning. In *IEEE/ACM International Conference on Computer Aided Design*, 1998.

[11] Carl Sechen and Alberto Sangiovanni-Vincentelli. The TimberWolf placement and routing package. *IEEE Journal of Solid-State Circuits*, SC-20(2), 1985.

[12] Wern-Jieh Sun and Carl Sechen. Efficient and effective placement for very large circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(3), 1995.

[13] William Swartz, et al. Timing driven placement for large standard cell circuits. In *Proceedings of the 32nd Design Automation Conference*, 211–215, 1995.

[14] Lixin Su, Wray Buntine, and A. Richard Newton. Stochastic Optimization with Incremental Learning for Standard Cell Placement. Submitted to *Design Automation Conference*, 2000.

[15] Benchmark circuits released with SIS-1.2. Department of EECS, University of California at Berkeley.

[16] http://www.cbl.ncsu.edu/benchmarks.

Collective Intelligence for Optimization (Summary)

David H. Wolpert and Kagan Tumer

NASA Ames Research Center,

A "COllective INtelligence" (COIN) is a distributed set of interacting reinforcement learning (RL) algorithms designed so that their collective behavior optimizes a global utility function. One can cast a COIN as a multi agent system (MAS) where:

i) there is little to no centralized communication or control;

ii) each agent runs a 'greedy' Reinforcement Learning (RL) algorithm, in an attempt to increase its own utility;

iii) there is a well-specified global objective function that rates the full system.

Rather than use a conventional modeling approach (e.g., model the system dynamics, and hand-tune agents to cooperate), we aim to solve the COIN design problem implicitly, via the "adaptive" character of th e RL algorithms of each of the agents. This approach introduces an entirely new, profound design problem: Assuming the RL algorithms are able to achieve high rewards, what reward functions for the individual agents will, when pursued by those agents, result in high world utility? In other words, what reward functi ons will best ensure that we do not have phenomena like the tragedy of the commons , Braess's paradox, or the liquidity trap?

An example of a naturally occurring COIN is a capitalist economy. One can declare 'world utility' to be a time average of the Gross Domestic Product (GDP) of the country in question. The reward functions for the human agents can then be the achievements of their personal goals (usually involving personal wealth to some degree). To achieve high world utility in any COIN it is necessary to avoid phenomena like the Tragedy of the Commons (TOC), in which individual avarice works to lower global utility. One way to avoid such phenomena is by modifying the agents' utility functions. In the context of capitalist economies, this can be done via punitive legislation. A real world example of an attempt to make just such a modification was the creation of anti-trust regulations designed to prevent monopolistic practices.

In designing a COIN we have more freedom than anti-trust regulators though, in that there is no base-line "organic" local utility function over which we must superimpose legislation-like incentives. Rather, the entire "psychology" of the individual agents is at our disposal, when designing a COIN. This obviates the need in designed COINs for honesty-elicitation ('incentive compatible') mechanisms, like auctions, which form a central component of computational economics.

We have explored a mathematical framework for COIN design, and investigated the (successful) application of that framework in several domains, e.g., optimizing the packet throughput of a telecommunications network [10]. Here we present a summary of a different investigation of our COIN methodology involving a variant of Arthur's "El Farol" bar problem [1, 2, 4, 7, 8, 3], a problem which first arose in economics (see [11, 9] for details). In the bar problem, at each time step each agent independently predicts, based on its previous experience, whether a bar will be too crowded to be "enjoyable" at that time step. The agent then uses this prediction to decide whether attending the bar or not will maximize its local utility. The global objective in this problem is to keep the bar as close to capacity as possible. In this problem the "greedy" nature of the agents can readily thwart the optimization of the world utility.

In the problem we investigated, there are N agents, each picking one of seven nights to attend a bar the following week, a process that is then repeated. In each week, each agent's pick is determined by its predictions of the associated rewards it would receive. Each such prediction in turn is based solely upon the rewards received by the agent in those preceding weeks in which it made that pick.

The world utility is $G(\underline{\zeta}) = \sum_t R_G(\underline{\zeta}_{,t})$, where $R_G(\underline{\zeta}_{,t}) \equiv \sum_{k=1}^{7} \phi_k(x_k(\underline{\zeta}, t))$, $x_k(\underline{\zeta}, t)$ is the total attendance on night k at week t, $\phi_k(y) \equiv \alpha_k y \exp(-y/c)$; and c and the $\{\alpha_k\}$ are real-valued parameters.

Intuitively, this G is the sum of the "world rewards" for each night in each week. Our choice of $\phi_k(.)$ means that when too few agents attend some night in some week, the bar suffers from lack of activity and therefore the world reward is low. Conversely, when there are too many agents the bar is overcrowded and the reward is again low.

Each agent η has a 7-dimensional vector representing its estimate of the reward it would receive for attending each night of the week. At the end of each week, the component of this vector corresponding to the night just attended is proportionally adjusted towards the actual reward just received. At the beginning of the succeeding week, to trade off exploration and exploitation, η picks the night to attend randomly using a Boltzmann distribution with 7 energies $\epsilon_i(\eta)$ given by the components of η's estimated rewards vector, and with a temperature decaying in time. This learning algorithm is similar to Claus and Boutilier's independent learner algorithm [5].

We considered three agent reward functions, using the same learning parameters (learning rate, Boltzmann temperature, decay rates, etc.) for each. The first reward function was $\gamma_\eta = G \; \forall \eta$, i.e., agent η's reward function equals R_G. The other two reward functions are: $R_{UD;\eta} \equiv \phi_{d_\eta}(x_{d_\eta}(\underline{\zeta}, t))/x_{d_\eta}$, $R_{WL;\eta}(\underline{\zeta}_{,t}) \equiv R_G - R_G(CL_\eta)$, where d_η is the night picked by η, and CL_η is a function derived from COIN theory.

The R_{UD} reward is a "natural" reward function to use; each night's total reward is uniformly divided among the agents attending that night. R_G is the "team game" reward function that has been investigated in the MAS community [6]; every agent gets the world reward as its reward signal. R_{WL} is the reward function recommended by COIN theory.

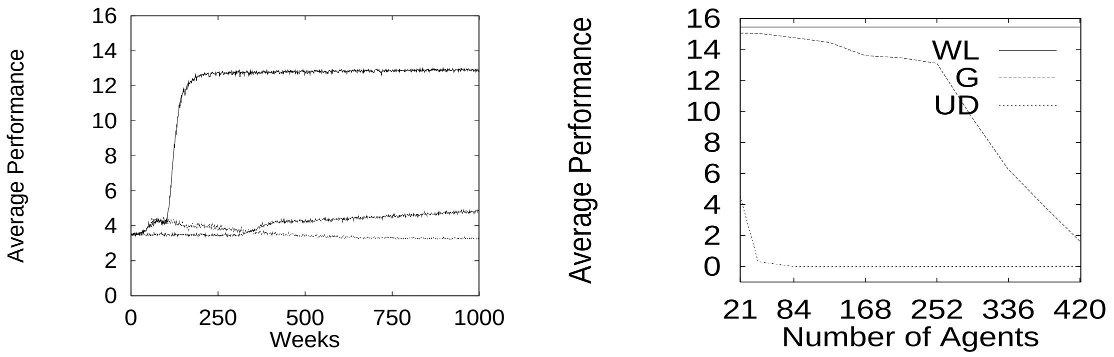


Figure 9: Average world reward convergence and scaling properties. In both plots the top curve is R_{WL}, middle is R_G, and bottom is R_{UD}.

The left-hand figure in Figure 9 graphs world reward value as a function of time, averaged over 50 runs, for all three reward functions. The naive choice of R_{UD} actually leads to *deterioration* of performance with time. Performance with R_G eventually converges to the global optimum. Systems using R_{WL} also converged to optimal performance, but far faster (30 times as quickly). This slow convergence of systems using R_G is a result of the reward signal being "diluted" by the large number of agents in the system. The right-hand figure in Figure 9 shows how performance at $t = 2000$ scales with N. Systems using R_{UD} perform poorly regardless of N. Systems using R_G perform well when N is low. As N increases however, it becomes increasingly difficult for the agents to extract the information they need from R_G.

In conclusion, the COIN framework summarized in this article addresses large distributed computational optimization tasks from a novel perspective, one that works much better than the other systems we investigated.

REFERENCES

[1] W. B. Arthur. Complexity in economic theory: Inductive reasoning and bounded rationality. *The American Economic Review*, 84(2):406–411, May 1994.

[2] G. Caldarelli, M. Marsili, and Y. C. Zhang. A prototype model of stock exchange. *Europhys. Letters*, 40:479–484, 1997.

[3] A. Cavagna, Garrahan J. P., I. Giardina, and D. Sherrington. A thermal model for adaptive competition in a market. preprint cond-mat/9903415 v.3, July 1999.

[4] D. Challet and Y. C. Zhang. On the minority game: Analytical and numerical studies. *Physica A*, 256:514, 1998.

[5] C. Claus and C. Boutilier. The dynamics of reinforcement learning cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752, June 1998.

[6] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems - 8*, pages 1017–1023. MIT Press, 1996.

[7] M. A. R. de Cara, O. Pla, and F. Guinea. Competition, efficiency and collective behavior in the "El Farol" bar model. preprint cond-mat/9811162 (to appear in European Physics Journal B), November 1998.

[8] W. A. Sethares and A. M. Bell. An adaptive solution to the El Farol problem. In *Proceedings. of the Thirty-Sixth Annual Allerton Conference on Communication, Control, and Computing*, Allerton, IL, 1998. (Invited).

[9] D. H. Wolpert and K. Tumer. An Introduction to Collective Intelligence. In J. M. Bradshaw, editor, *Handbook of Agent technology.* AAAI Press/MIT Press, 1999. to appear.

[10] D. H. Wolpert, K. Tumer, and J. Frank. Using collective intelligence to route internet traffic. In *Advances in Neural Information Processing Systems - 11.* MIT Press, 1999.

[11] D. H. Wolpert, K. Wheeler, and K. Tumer. General principles of learning-based multi-agent systems. In *Proceedings of the Third International Conference of Autonomous Agents*, pages 77–83, 1999.

Efficient Value Function Approximation Using Regression Trees

Xin Wang and Thomas G. Dietterich

Oregon State University,

Value function approximation is critical for the application of reinforcement learning in large state spaces, such as those that arise in combinatorial optimization problems. The majority of successful applications of reinforcement learning have employed neural network function approximators [5, 3], but these are slow and often require substantial parameter tuning and special training methods to obtain good performance. For example, to obtain successful results in resource-constrained scheduling problems, Zhang and Dietterich [7] had to carefully adjust learning rates, output representations, experience replay, and reverse-trajectory TD(λ) training.

Table 6: Comparison of Value Function Approximation Methods

Function Approximator	Bias	Speed	Scaling	Discontinuities?	Irrelevant & Correlated Features?
Neural Networks	Low	Slow	Good	Ok	Yes
Linear Functions	High	Fast	Good	No	No
Local Linear	Low	Slow	Poor	Ok	No
CMAC	Medium	Fast	Poor	Ok	No
Regression Trees	Low	Fast	Good	Good	Yes

Our goal is to develop a new family of function approximators that have low bias, high speed, good scaling to high-dimensional input spaces, and the ability to represent discontinuous value functions and to handle irrelevant and correlated input features. Table 6 summarizes the advantages and disadvantages of various existing function approximator families according to these criteria. As the table shows, we have chosen to focus on regression trees, because they show promise of doing well on all of these criteria.

Our regression trees are binary trees. Each internal node contains a splitting plane that divides the feature space into two half-spaces corresponding the node's left and right child nodes. Each leaf node in the tree contains a linear function defined over the feature space. Given feature vector $\mathbf{x}$, its value is predicted by "dropping" it through the tree, obeying the splitting plane at each internal node, and evaluating the linear function at the leaf node.

There are three key steps in any regression tree algorithm: (a) choosing the splitting planes at the internal nodes, (b) fitting the planes at the leaf nodes, and (c) halting tree growth.

Choosing Splitting Planes. We try to put splitting planes where there are discontinuities in the value function. To avoid searching the infinite space of splitting planes, we borrowed an idea from Hinton and Revow's [4] decision-tree algorithm in which splitting planes are defined by taking the plane that lies mid-way between two training examples. They considered all pairs of training examples belonging to different classes, and evaluated each of these candidate planes (by one-step lookahead) to choose the best one.

In reinforcement learning in deterministic (or nearly deterministic) state spaces, discontinuities in the value function are found primarily between pairs of states (s_1, s_2) where s_2 can be reached in one step from s_1. Our algorithm proceeds by randomly drawing a sample of 50 such pairs and constructing the plane that is the perpendicular bisector of each pair. Each splitting plane is evaluated by one-step lookahead. The training examples at the current node are partitioned according to the splitting plane, and the two resulting sets of points are fitted with linear surfaces as described below. The split that gives the best overall fit is chosen.

Fitting Linear Functions to the Leaf Nodes. In supervised learning, the usual practice is to find the function $\hat{V}$ that minimizes the squared error between the predicted and the actual values. We will call this the supervised error:

$$E_1 = \sum_s \left[V(s) - \hat{V}(s)\right]^2. \tag{1}$$

This method has been applied in reinforcement learning (e.g., [2]), but even if $\hat{V}$ is a good approximation to V, the action recommended by $\hat{V}$ can still be much worse than the action recommended by V. The problem is that unless the learning algorithm can reduce the supervised error to zero, $\hat{V}$ will not be a solution to the Bellman equation, and this is a requirement for producing an optimal policy.

To solve the Bellman equation, we can try to minimize the Bellman error:

$$E_2 = \sum_s \left(\hat{V}(s) - \max_a \sum_{s'} P(s'|s,a)[R(s'|s,a) + \hat{V}(s')]\right)^2. \tag{2}$$

This is the basis of $TD(\lambda)$ and related algorithms. However, if $\hat{V}$ cannot represent this solution exactly, the action recommended by $\hat{V}$ may not be optimal.

Baird [1] suggested minimizing the Advantage Error, which we can write as follows. Let $Q(s,a)$ be the return of performing action a in state s:

$$Q(s,a) = \sum_{s'} P(s'|s,a)[R(s'|s,a) + V(s')]. \tag{3}$$

Let a_{best} be the action that has the highest Q value. Then the *advantage* of performing action a in state s is defined as

$$A(s,a) = Q(s,a) - Q(s,a_{best}). \tag{4}$$

The advantages of all actions except a_{best} are negative. We can then define the squared *advantage error* to be

$$E_3 = \sum_s \sum_a \left(\left[\hat{Q}(s,a) - \hat{Q}(s,a_{best})\right]_+\right)^2 \tag{5}$$

where the notation $[x]_+$ is 0 if x is negative, and x if x is positive. Hence, if the predicted advantage of a is positive, then we have an error (since all advantages should be negative or zero), and we square that error.

To train the leaves of the regression tree, we combine all three error terms to give a weighted composite error

$$E = \omega_1 E_1 + \omega_2 E_2 + \omega_3 E_3. \tag{6}$$

Here, the $\omega_i \geq 0$ control the relative importance of the supervised, Bellman, and Advantage error terms.

Stopping Rule. If the improvement in the composite error between a parent node and its children is less than 5%, then tree growth is halted.

We tested our regression tree algorithm on the ART-1 set of resource-constrained scheduling problems developed by Wei Zhang [8]. For each of the 25 training problems, we applied a simple greedy heuristic developed by Zhang to guide a beam search (beam width 20) to find a single "best" path from the starting state to a feasible solution. We then grew a regression tree from these "best paths" (and all one-step departures from the best path) using the algorithm outlined above. The values of the ω's were chosen to optimize performance on 25 validation problems. Finally, the learned regression tree was applied to solve 50 test problems and compared to the solutions found by Wei Zhang's neural network value function. Figure 10 summarizes the results.

The regression tree gives better results for 23 problems, gives the same result for 3, and gives worse results for 24. A 2-sided t test cannot reject the null hypothesis that the two function approximators are giving the same performance ($p = .84$).

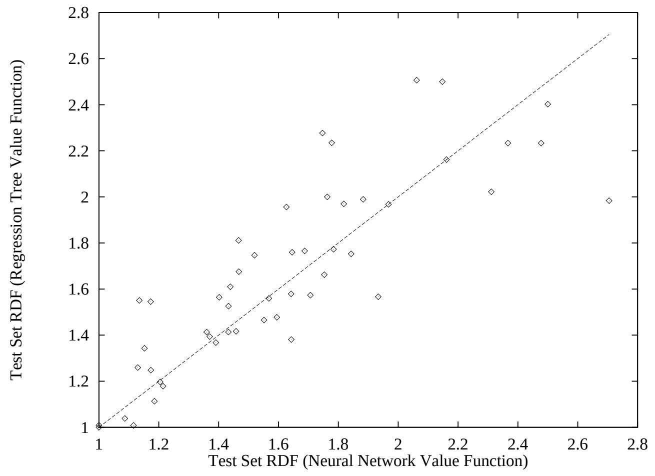


Figure 10: Regression tree (1 best path and using RDF) vs. Neural net on test set; points below the line are problems where the regression tree gave a better solution than the neural network.

Table 7: Importance of Individual Terms in the Composite Objective Function

			Regression tree vs. Neural net		
Supervised	Bellman	Advantage	win	tie	lose
yes	yes	yes	23	3	24
no	yes	yes	1	0	49
yes	no	yes	14	3	34
yes	yes	no	11	1	38
yes	no	no	4	0	46
no	yes	no	3	0	47
no	no	yes	6	0	44

Table 7 shows that each of the three terms is essential in order for the regression tree to match the performance of the neural network.

The final question we address is training time. According to our most recent measurements, the training time for the regression trees on the ART-1 problem set is about a factor of 10 faster than for the neural network. On more difficult problem sets, we expect the differences to be more dramatic.

The results of this initial study are promising. However, before we can apply regression trees to large combinatorial optimization problems, we need to address three major problems. First, our algorithm requires supervised values for the states in the training data. For ART-1, we have a good heuristic, but for other problems, a better method is needed for finding supervised values. Second, our algorithm is a batch algorithm. We need some way to interleave exploration with function fitting (e.g., by making the method more incremental). Finally, our current splitting rule assumes that all features are equally relevant (and uncorrelated). We need to improve the rule to perform some kind of feature selection during splitting so that irrelevant and correlated features can be detected and ignored.

Acknowledgements

The authors gratefully acknowledge the support of NSF Grant 9626584-IRI and AFOSR Grant F49620-98-1-0375.

REFERENCES

[1] Baird, L. C. 1993. Advantage updating. WL-TR-93-1146, Wright-Patterson Air Force Base.

[2] Boyan, J. A. and Moore, A. W. 1995. Generalization in reinforcement learning: Safely approximating the value function. In Tesauro, G., Touretzky, D. S., Leen, T. K. (Eds), *Advances in Neural Information Processing Systems*, Vol 7, 369–376. The MIT Press, Cambridge.

[3] Crites, R. H. Barto, A. G. 1996. Improving elevator performance using reinforcement learning. In Touretzky et al. [6], 1017–1023.

[4] Hinton, G. E. Revow, M. 1996. Using pairs of data-points to define splits for decision trees. In Touretzky et al. [6], 507–513.

[5] Tesauro, G. 1992. Practical issues in temporal difference learning. *Machine Learning*, *8*, 257–277.

[6] Touretzky, D. S., Mozer, M. C., Hasselmo, M. E. (Eds). 1996. *Advances in Neural Information Processing Systems*, Vol 8. The MIT Press, Cambridge.

[7] Zhang, W. Dietterich, T. G. 1995. A reinforcement learning approach to job-shop scheduling. In *1995 International Joint Conference on Artificial Intelligence*, 1114–1120. Morgan Kaufmann, San Francisco, CA.

[8] Zhang, W. 1996. *Reinforcement Learning for Job-Shop Scheduling.* Ph.D. thesis, Oregon State University, Department of Computer Science.

Numerical Methods for Very High-Dimension Vector Spaces

T. Dean, K.E. Kim, and S. Hazlehurst

Brown University,

There is a large class of numerical optimization problems that can be described in terms of equations involving vectors and matrices. One example that we use throughout this short overview is the problem of finding an optimal or near-optimal policy for a Markov decision problem (MDP). MDPs constitute a stochastic generalization of the deterministic propositional planning problems studied in artificial intelligence [3]. The objective functions and solution methods for MDPs are often characterized in terms of matrix-vector equations [5].

Consider calculating $\mathbf{v}$ defined as $\mathbf{v} = \mathbf{A}\mathbf{u}$, in which A is an $n \times n$ matrix and $\mathbf{u}$ and $\mathbf{v}$ are vectors of dimension n. The matrix A might be the adjacency matrix for a graph representing a database relation or the state-transition matrix for a planning problem and $\mathbf{u}$ might be the specification of vertices corresponding to a query or the initial-state distribution. Often enough, we are interested in solving problems where $n = k^m$ for some m and $k \geq 2$. Of course, if m is large, we probably can't represent $\mathbf{v} = \mathbf{A}\mathbf{u}$ explicitly by allocating space for each entry of A, $\mathbf{u}$, and $\mathbf{v}$. But what if A, $\mathbf{u}$, and $\mathbf{v}$ are "symmetrical" or "regular" in some sense? For example, sparse matrix representations exploit certain types of symmetries for problems in which $O(n)$ is acceptable (and hence m is relatively small) but $O(n^2)$ is not. In this work, we are interested in a different class of regularities for problems in which m is large and $O(n)$ time or space is out of the question.

In many of the optimization problems encountered in artificial intelligence, including the problem of finding optimal policies for MDPs, the matrices and vectors can be quite large. In particular, it is often the case that the number of indices (and hence the size of a matrix, say, if represented as simple table) is exponential in the size of the problem description.

For example, in the case of factored MDPs [1], if we were to allocate space for each entry, the sizes of the state-transition matrix and the reward vector would be exponential in the number of state variables used to describe the domain. In many cases, however, there are representations for these large matrices and vectors that allow us to encode these objects in a compact and tractable form. In the following, we describe one such representation based on trees that works well for MDPs.

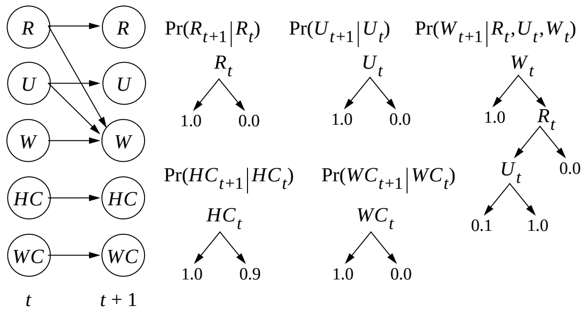


Figure 11: Compact representation for a robot action

Figure 11 depicts a compact representation for the state-transition probability distribution for an action in a simple robot domain modeled as an MDP [2]. We define R, U, W, HC, WC as (boolean) *index (or state) variables* representing, respectively, the weather outside being rainy, the robot having an umbrella, the robot being wet, the robot holding coffee, and the robot's boss wanting coffee. The network shown on the left in Figure 11 indicates the functional dependencies involving these variables. The tree structures on the right indicate the transition probability distributions for each index variable given its parents as determined by the network on the left.

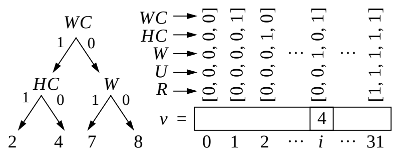


Figure 12: A vector compactly represented as a tree

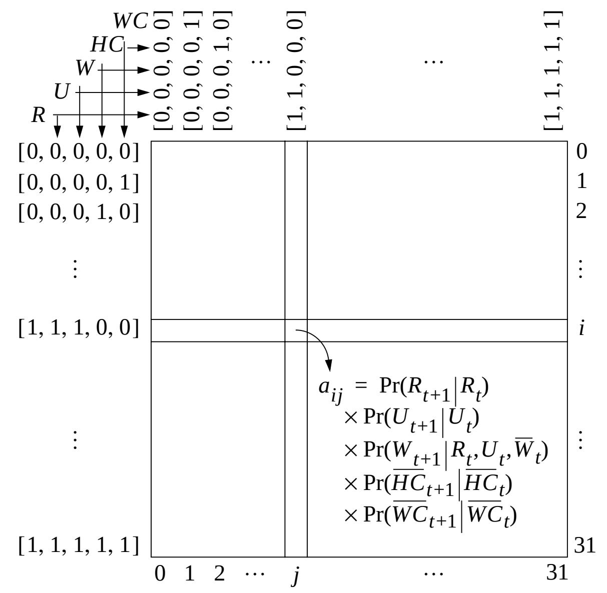

WC
HC
W
U
R
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 1]
[0, 0, 0, 1, 0]
[1, 1, 0, 0, 0]
[1, 1, 1, 1, 1]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 1]
[0, 0, 0, 1, 0]
[1, 1, 1, 0, 0]
[1, 1, 1, 1, 1]
0
1
2
i
31
0 1 2 ··· j ··· 31
$a_{ij} = \Pr(R_{t+1}|R_t)$
$\times \Pr(U_{t+1}|U_t)$
$\times \Pr(W_{t+1}|R_t, U_t, \overline{W}_t)$
$\times \Pr(\overline{HC}_{t+1}|\overline{HC}_t)$
$\times \Pr(\overline{WC}_{t+1}|\overline{WC}_t)$

Figure 13: Computing an entry in a state-transition matrix

Let $\mathbf{Z} = [\mathbf{R}, \mathbf{U}, \mathbf{W}, \mathbf{HC}, \mathbf{WC}]$ denote a vector $\mathbf{z} \in \{\mathbf{0}, \mathbf{1}\}^{\mathbf{5}}$. $\mathbf{z}$ determines the *index* into the vectors and matrices of the original problem. Two indices are said to be *equivalent with respect to value* if their corresponding entries are the same. This equivalence relation induces a partition on the set of all indices. Ideally, we would only want to allocate an amount of space polynomial in the number of index variables for each block in this partition — this amount of space would have to suffice for both the value of the entry (common to all of the entries) and for the representation of the block. Fortunately, in some cases, we can represent large blocks of indices quite compactly, *e.g.*, we might interpret the formula $WC \wedge \overline{HC}$ as representing the set of all indices in which WC is assigned 1 and HC is assigned 0. Note that in Figure 12 which depicts the reward vector (function) for the robot problem as a tree, we have achieved economy of representation by exploiting independence involving the index variables, *e.g.*, if WC is 1, then the entry of the vector is independent of the value of W.

We can represent large matrices in a similar manner. The probability of ending up in one state having started from another is determined as the product of the values found in the transition probability trees. Figure 13 shows the formula for a particular entry in the state-transition probability matrix. Note that although the dimension of the vectors and matrices is exponential in the number of index variables, retrieving a particular entry can be done in time polynomial in the size of the representation.

Once we can actually write down the equations for the underlying optimization problem in terms of matrices and vectors, there remains the problem of carrying out the basic operations, e.g., transposing matrices, computing vector-vector and matrix-vector products, and raising a matrix to a power, that are required to implement various numerical methods. In [4], we describe procedures for carrying out the basic operations on vectors and matrices represented in terms of trees; in the following, we provide an example to

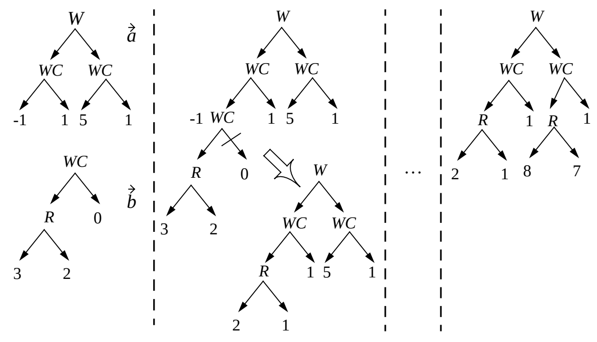


Figure 14: Adding two vectors represented as trees

illustrate the basic idea.

Generalizing on the work of Boutilier *et al.* [2], we define vector addition, inner product, and matrix-vector multiplication operators in terms of a basic tree grafting operator. In Figure 14, we show how vector addition is carried out. Note that the operation is carried out without explicitly enumerating all the entries in the vector. Hence, we use the term *structured* for linear algebra operators and numerical algorithms built on these operators. By implementing linear algebra operators using these structured operations, we can, for example, apply Richardson's method, various extrapolation methods, conjugate gradient descent, and a host of other numerical algorithms to problems involving very large matrices and vectors.

In some cases, the vectors or matrices that result from carrying out basic operations can be larger than any of the matrix or vector terms involved in the operations. In the worst case, the most compact representation for the result of a computation involving k terms can be of size exponential in k. To deal with this potential for combinatorial explosion, we apply techniques from machine learning and data compression to control the size of the intermediate and final results of performing operations on matrices and vectors. In the case of trees, these methods work by pruning the leaves of trees, thereby merging blocks in the partitions that are implicit in the tree data structures.

By keeping track of bounds on the values of the entries for indices in each block of the partitions, we can report bounds on the error in the final results. Standard pruning techniques such as those presented in [6] can be applied to elementary structured operations to produce approximate structured operations. In much the same way as finite-precision arithmetic introduces errors in computer implementations of numerical methods, approximate operations on matrices and vectors introduce errors and thus require careful analysis with respect to convergence and precision [7]. Under reasonable assumptions, we can guarantee that the resulting approximations converge and the error (the difference between result from the algorithm using the approximate structured operators and the true answer) is bounded.

The contributions of this work include data structures for representing very large matrices and vectors, a set of procedures that operate on these data structures, and a set of analytical methods that enable us to apply a wide range of numerical methods based on linear algebra directly to the solution of combinatorial optimization problems involving very large matrices and vectors.

REFERENCES

[1] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[2] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. In *Proceedings IJCAI 14*, pages 1104–1111. IJCAII, 1995.

[3] Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[4] Kee-Eung Kim, Thomas Dean, and Samuel E. Hazlehurst. Linear algebra in very high-dimension vector spaces: Algorithms and data structures for implementing exact and approximate solution methods. Technical report, Computer Science Department, Brown University, 1999. To Appear.

[5] Martin L. Puterman. *Markov Decision Processes.* John Wiley & Sons, New York, 1994.

[6] J. Ross Quinlan. *C4.5 : Programs for Machine Learning.* Morgan Kaufmann, 1992.

[7] J. H. Wilkinson. Modern error analysis. *SIAM Review*, 1971.