MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Human-Guided Tabu Search

Gunnar W. Klau, Neal Lesh, Joe Marks, Michael Mitzenmacher

TR2002-09 December 2002

Abstract

We present a human-guidable and general tabu search algorithm. Ourguiding an exhaustive algorithm.

AAAI02

Submitted January 2002. Revised and released May 2002.

Human-Guided Tabu Search

Gunnar W. Klau[1], **Neal Lesh**[2], **Joe Marks**[2], **Michael Mitzenmacher**[3]

[1] Vienna University of Technology, Austria
guwek@ads.tuwien.ac.at
[2] Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA, 02139
{lesh,marks}@merl.com
[3] Harvard University, Computer Science Department
michaelm@eecs.harvard.edu

Abstract

We present a human-guidable and general tabu search algorithm. Our work expands on previous interactive optimization techniques that provide for substantial human control over a simple, exhaustive search algorithm. User experiments in four domains confirm that human guidance can improve the performance of tabu search and that people obtain superior results by guiding a tabu algorithm than by guiding an exhaustive algorithm.

Introduction

Interactive, or human-in-the-loop, optimization systems have been developed for a variety of applications, including space-shuttle scheduling, graph drawing, graph partitioning, and vehicle routing. While automatic algorithms typically solve an oversimplified formulation of a real-world problem, users can steer an interactive algorithm based on their preferences and knowledge of real-world constraints. Interactive optimization also leverages people's skills in areas in which people currently outperform computers, such as visual perception, strategic thinking, and the ability to learn. An additional advantage of interactive systems is that people can better trust, justify, and modify solutions that they help construct than they can automatically generated solutions.

Our work expands on the Human-Guided Simple Search (HuGSS) framework (Anderson *et al.*, 2000), which provides the user a greater degree of control than previous interactive optimization approaches, but employs a relatively weak optimization algorithm. With HuGSS, users can manually modify solutions, backtrack to previous solutions, and invoke, monitor, and halt an exhaustive breadth-first search. More significantly, users can constrain and focus the exhaustive search algorithm by assigning *mobilities*, which we describe below, to elements of the current solution. Experiments have shown that human guidance can improve the performance of the exhaustive search algorithm on the capacitated-vehicle-routing-with-time-windows problem to the point where the interactive algorithm is competitive with the best previously reported algorithms (Anderson *et al.*, 2000; Scott, Lesh, & Klau, 2002).

We present a general and human-guidable tabu search algorithm that serves as a superior alternative to the exhaustive search algorithm in HuGSS. Tabu search is a powerful heuristic search algorithm that has proven effective for a wide variety of problems (for an overview see Glover & Laguna (1997)). While previous research on interactive optimization has generally addressed individual problems, we show the generality of our guidable tabu algorithm by applying it to four diverse optimization problems. The algorithm is guidable in that (1) it can be constrained and focused with the same mobilities metaphor used in HuGSS, (2) it controls its own search using mobilities, which provides a natural visualization of the tabu search, and (3) its control parameters are more easily understandable than the analogous ones used to fine-tune the performance of automatic tabu algorithms.

We describe two experiments designed to compare guided tabu search to guided exhaustive search and to unguided (i.e., fully automatic) tabu search. The first experiment included a total of seven test subjects, two domains, and 80 trials. The results indicate that 10 minutes of guided tabu search is comparable to, on average, 70 minutes of unguided tabu search. Furthermore, our experiments demonstrate that guided tabu search significantly outperforms guided exhaustive search. Our second set of experiments investigates if experienced users can improve highly optimized solutions produced by five hours of tabu-search precomputation. We show that half an hour of human-guided tabu search improves these solutions slightly more than 10 additional hours of unguided tabu search.

Background

Interactive Optimization

Interactive systems that leverage the strengths of both humans and computers must distribute the work involved in the optimization task among the human and computer participants. Existing systems have implemented this division of labor in a variety of ways.

In some interactive systems, the users can only indirectly affect the solutions to the current problem. For example, in interactive evolution, an approach primarily applied to design problems, the computer generates solutions via biologically inspired methods and the user selects which solutions will be used to generate novel solutions in the next iteration (Sims, 1991; Todd & Latham, 1992).

Other systems provide more interactivity by allowing the users to control search parameters or add constraints as the search evolves. Colgan *et al.* (Colgan, Spence, & Rankin,

1995) present a system which allows users to interactively control the parameters that are used to evaluate candidate solutions for circuit-design problems. Several constraint-based systems have been developed for drawing applications (Gleicher & Witkin, 1994; Ryall, Marks, & Shieber, 1997; Nelson, 1985). Typically, the user imposes geometric or topological constraints on an emerging drawing.

Some systems allow more direct control by allowing users to manually modify computer-generated solutions with little or no restrictions and then invoke various computer analyses on the updated solution. An early vehicle-routing system allows users to request suggestions for improvements after making schedule refinements to the initial solution (Waters, 1984). An interactive space-shuttle operations-scheduling system allows users to invoke a repair algorithm on their manually modified schedules to resolve any conflicts introduced by the user (Chien *et al.*, 1999).

The human-guided simple search (HuGSS) framework (Anderson *et al.*, 2000) also allows users to manually modify solutions, but in addition it allows them to explicitly steer the optimization process itself. In this approach, users invoke, monitor, and halt optimizations as well as specify the scope of these optimizations. Users control how much effort the computer expends on particular subproblems. Users can also backtrack to previous solutions. HuGSS was utilized in an interactive vehicle-routing system. Initial experiments with this system showed that human-guided optimization outperformed almost all reported vehicle-routing algorithms. A more focused study examined people's ability to guide search in the various ways allowed by HuGSS (Scott, Lesh, & Klau, 2002).

Following the HuGSS framework, do Nascimento and Eades developed an interactive layered graph-drawing system that provided most of the functionality of HuGSS and also allowed users to add constraints to the problem at runtime (Nascimento & Eades, 2001). Preliminary experiments have shown that people can improve automatically generated solutions using this system.

Tabu Search

Tabu search is a heuristic approach for exploring a large solution space (Glover & Laguna, 1997). Like other local search techniques, tabu search exploits a neighborhood structure defined on the solution space. In each iteration, tabu search evaluates all neighbors of the current solution and moves to the best one. The neighbors are evaluated both in terms of the problem's objective function and by other metrics designed to encourage investigation of unexplored areas of the solution space. The classic "diversification" mechanism that encourages exploration is to maintain a list of "tabu" moves that are temporarily forbidden, although others have been developed. Recent tabu algorithms often also include "intensification" methods for thoroughly exploring promising regions of the solution space (although our algorithm does not currently include such mechanisms). In practice, the general tabu approach is often customized for individual applications in myriad ways (Glover & Laguna, 1997).

Algorithm

Example applications

We applied our tabu search algorithm to the following four applications.

The *Crossing* application is a graph layout problem (Eades & Wormald, 1994). A problem consists of m levels, each with n nodes, and edges connecting nodes on adjacent levels. The goal is to rearrange nodes within their level to minimize the number of intersections between edges. A screenshot of the Crossing application is shown in Figure 1.

The *Delivery* application is a variation of the Traveling Salesman Problem in which there is no requirement to visit every location (Feillet, Dejax, & Gendreau, 2001). A problem consists of a starting point, a maximum distance, and a set of customers each at a fixed geographic location with a given number of requested packages. The goal is to deliver as many packages as possible without driving more than a given maximum distance. A screenshot of the Delivery application is shown in Figure 2.

The *Protein* application is a simplified version of the protein-folding problem, using the hydrophobic-hydrophilic model introduced by Dill (Dill, 1985). A problem consists of a sequence of amino acids, each labeled as either hydrophobic or hydrophilic. The sequence must be placed on a two-dimensional grid without overlapping, so that adjacent amino acids in the sequence remain adjacent in the grid. The goal is to maximize the number of adjacent hydrophobic pairs. A screenshot of the Protein application is shown in Figure 3.

The *Jobshop* application is a widely-studied task scheduling problem (Aarts *et al.*, 1994). In the variation we consider, a problem consists of n jobs and m machines. Each job is composed of m operations (one for each machine) which must be performed in a specified order. Operations must not overlap on a machine, and the operations assigned to a given machine can be processed in any order. The goal is to minimize the time that the last job finishes. A screenshot of the Jobshop application is shown in Figure 4.

Terminology

We introduce terminology for the abstractions in our framework. For each optimization problem, we assume there is some set of *problem instances*. For each problem instance, there is a set of candidate *solutions*. We assume that the solutions are totally ordered (with ties allowed); the function ISBETTER(s_1,s_2) returns true iff solution s_1 is strictly superior to s_2. The function INIT(p) returns an initial solution for problem p. A *move* is a transformation that can be applied to one solution to produce a new solution. Each move is defined as operating on one problem *element* and altering that element and possibly others. For example, moving a node from the 3rd to the 8th position in a list, and shifting the 4th through 8th nodes up one, would operate on the 3rd element and alter the 3rd through the 8th. The function MOVES(s,e) returns the set of transformations that operate on element e in solution s. The function ALTERED(m) returns the set of elements altered by m.

The definition of elements varies from application to application. The elements are customers, nodes, amino acids, and job operations in the Delivery, Crossing, Protein, and

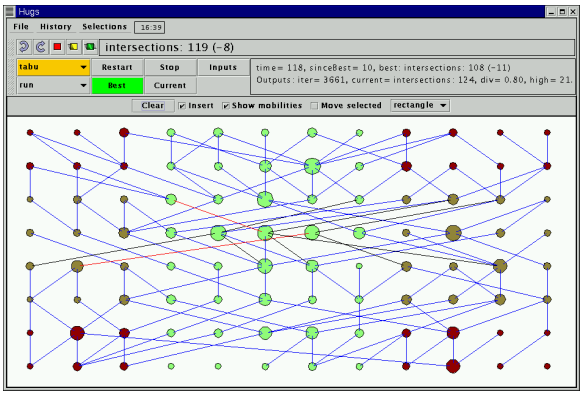

Figure 1: The Crossing Application.

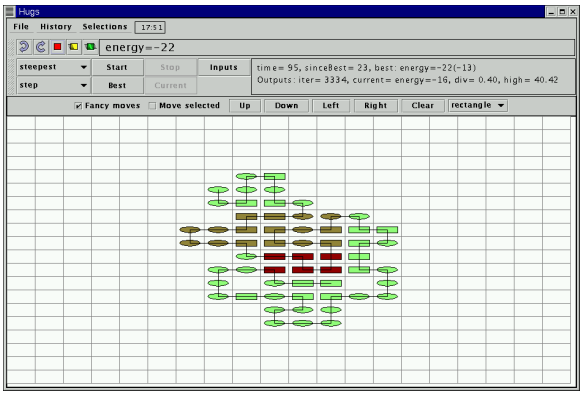

Figure 3: The Protein Application.

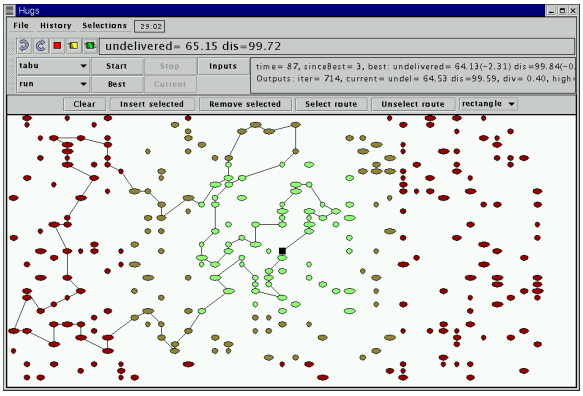

Figure 2: The Delivery Application.

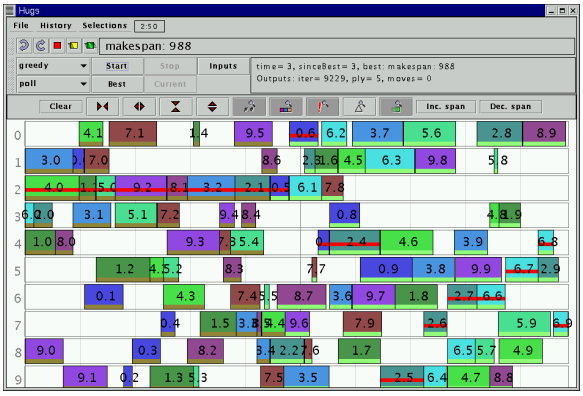

Figure 4: The Jobshop Application.

Jobshop applications, respectively. The definition of transformations also varies from application to application. In fact, as with automatic optimization, which transformations to include is an important design choice for the developer. Example transformations are: swapping adjacent nodes within a level for Crossing; swapping adjacent operations on the same machine in Jobshop; and inserting a customer into the route in Delivery (among other transformations). Our framework requires an additional decision by the developer: the elements that are altered by each tranformation. For example, in Delivery, the insertion transformation alters only the inserted customer. Alternatively, this transformation could be defined as altering nearby customers on the route. Initial experience with the application can help guide these decisions.

Mobilities

In our system, as with HuGSS, the system maintains and displays a single current solution, such as the ones shown in Figures 1, 2, 3, and 4. Mobilities are a general mechanism that allow users to visually annotate elements of a solution in order to guide a computer search to improve this solution. Each element is assigned a *mobility*: high, medium, or low. The search algorithm is only allowed to explore solutions that can be reached by applying a sequence of moves to the current solution such that each move operates on a high-mobility element and does not alter any low-mobility elements.

We demonstrate mobilities with a simple example. Suppose the problem contains seven elements and the solutions to this problem are all possible orderings of these elements. The only allowed move on an element is to swap it with an adjacent element. Suppose the current solution is as follows, and we have assigned element 3 low mobility (shown in dark gray), element 5 and 6 medium mobility (shown in medium gray), and the rest of the elements have high mobility (shown in light gray):

A search algorithm can swap a pair of adjacent elements only if at least one has high mobility and neither has low mobility. It is limited to the space of solutions reachable by a series of such swaps, including:

Note that setting element 3 to low mobility essentially divides the problem into two much smaller subproblems. Also, while medium-mobility elements can change position, their relative order cannot be changed. Mobility constraints can drastically reduce the search space; for this example, there are only 12 possible solutions, while without mobilities, there are 7!=5040 possible solutions.

We have found that this generalized version of mobilities is useful in a wide variety of applications, including the four described above.

Guidable Tabu

We now present GTABU, a guidable tabu search algorithm. The algorithm maintains a current solution and current set of mobilities. In each iteration, GTABU first evaluates all legal moves on the current solution given the current mobilities, in order to identify which one would yield the best solution. It then applies this move, which may make the current solution worse, and then updates its current mobilities so as to prevent cycling and encourage exploration of new regions of the search space. The pseudocode for GTABU is shown in Figures 5 and 6.

The algorithm updates the mobilities in two ways. First, the call to the MEMORY function prevents the tabu search from immediately backtracking, or cycling, by setting elements altered by the current move to medium mobility. For example, in Crossing, if the current move swaps two nodes, then both nodes are set to medium mobility, so that these two nodes cannot simply be reswapped to their original locations. The nodes are restored to their original mobility after a user-defined number of iterations elapse, controlled by an integer $memSize$ which is an input to GTABU. Most tabu search algorithms have a similar mechanism to prevent cycling.

A second mechanism, performed by the DIVERSIFY function in Figure 6, encourages the algorithm to choose moves that alter elements that have been altered less frequently in the past. The algorithm maintains a list of all the problem elements, sorted in descending order by the number of times they have been altered. The *diversity* of an element is its position on the list divided by the total number of elements. The *diversity* of a move is the average diversity of the elements it alters. The *diversity* of a search is the average diversity of the moves it has made since the last time it has found a best solution. The user is allowed to indicate a target minimum diversity $minDiv$ between 0 and 1 for the search. Whenever the average diversity falls below this threshold, then any element with a diversity less than $minDiv$ is set to medium for one iteration. This forces the tabu algorithm to make a move with high diversity.

Under the assumption that a system is more guidable if it is more understandable, we strove to design a tabu algorithm that was easy to comprehend. Many automatic tabu algorithms, for example, have a mechanism for encouraging diversification in which the value of a move is computed based on how it affects the cost of the current solution and some definition of how diverse the move is. The two components are combined using a control parameter which specifies a weight for the diversification factor. We originally took a similar approach, but found that users had trouble

```
GTABU (solution, mobilities, memSize, minDiv):
    best ← solution
    originalMobilities ← mobilities
    until halted by user
        m ← best move in LEGALMOVES(solution,mobilities)
        solution ← result of m applied to solution
        if ISBETTER(solution, best) then
            best ← solution
            mobilities ← originalMobilities
        else
            mobilities ← MEMORY(m,mobilities,memSize)
            mobilities ← DIVERSIFY(m,mobilities, minDiv)
    return best
```

Figure 5: Pseudo code for guidable tabu search.

understanding and using this control parameter. Our experience from the training sessions described below is that the test subjects can easily understand the $minDiv$ control parameter.

The understandability of the algorithm is also greatly enhanced by the fact that the tabu algorithm controls its search by modifying mobilities. The users of our system learn the meaning of the mobilities by using them to control and focus the search. All four applications provide a color-coded visualization of the users' current mobility settings. This same mechanism can be used to display GTABU's mobilities. We provide several different visualization modes that allow the user to step through the search one iteration at a time or to view GTABU's current solution and mobility settings briefly at each iteration. During an optimization session, these visualizations are typically turned off because they reduce the efficiency of the system. However, while learning how to use the system, these visualization modes help users understand how the algorithm works.

Experimental Results

Implementation

We implemented domain-independent middleware for interactive optimization in Java and then implemented our four applications using this middleware. All applications use the same implementation of our tabu search algorithm. The middleware also includes a GUI and functions for managing the current working solution and mobilities, the history of past solutions, file Input/Output, and logging user behavior. This software is freely available for research or educational purposes. More details are described in Klau *et al.* (2002).

Our code follows the HuGSS framework. Users can manually modify solutions, backtrack to previous solutions, assign mobilities to problem elements, and invoke, monitor, and halt a search algorithm. Unlike HuGSS, however, our system provides a choice of search algorithms and visualization modes. In addition to tabu search, we also provide (a domain-independent) steepest-descent and greedy exhaustive search, similar to those in the original HuGGS system. Both exhaustive algorithms first evaluate all legal moves, then all combinations of two legal moves, and then all combinations of three moves and so forth. The steepest-descent algorithm searches for the move that most improves the current solution. The greedy algorithm immediately makes any move which improves the current solution and then restarts

	greedy	Tabu									
		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Delivery	79.8	72.2	70.6	67.6	65.6	**64.6**	65.2	66.9	68.8	70.3	70.9
Crossing	69.0	85.0	75.9	67.7	64.3	53.9	48.1	40.0	38.4	**37.9**	43.4
Protein	-31.5	-30.0	-31.3	-34.8	-36.1	**-36.6**	-36.1	-35.7	-34.1	-33.4	-29.3
Jobshop	2050.4	2167.4	2045.3	1922.2	1820.7	1821.9	**1779.5**	1846.9	1860.4	2059.0	2164.2

Table 1: Results of unguided greedy and unguided tabu with different minimum diversities. All results averaged over 10 problems, after five minutes of search. The best result in each row is shown in bold. All the problems are minimization problems, so lower numbers are better. Note that the numbers are negative for Protein.

LEGALMOVES ($solution$, $mobilities$):
  returns the set of all moves m in MOVES($solution$,e)
  where e has high mobility in $mobilities$ and every element
  in ALTERED(m) has high or medium mobility in $mobilities$

DIVERSIFY ($move$, $mobilities$, $minDiv$):
  restore any elements to $high$ mobility that were set to
    medium mobility by previous call to DIVERSIFY
  compute average diversity of search (as defined in the paper)
if average diversity is less than $minDiv$ **then** set all
  elements with high mobility in $mobilities$ and diversity
  less than $minDiv$ to medium mobility
return $mobilities$

MEMORY ($move$, $mobilities$, $memSize$):
  restore any elements to high mobility that were set to
    medium mobility $memSize$ iterations ago by MEMORY
  set all high-mobility elements in ALTERED($move$) to
    medium mobility
  return $mobilities$

Figure 6: Support functions for guidable tabu search.

its search from the resulting solution.

Each application requires a domain-specific implementation of the problems, solutions, and moves. Essentially, all the functions described in the "Terminology" section above must be defined for each application. Each application also requires a visualization component to display the current solution and mobilities, as well as allow users to perform manual moves.

We generated problems as follows. For Delivery, we randomly distributed 300 customers on an 80×40 grid, and randomly assigned each customer between three and seven requests. The truck is allowed to drive a total of 400 units on the grid. For Crossing, we used ten 12×8 graphs with 110 edges which are publicly available from http://unix.csis.ul.ie/~grafdath/TR-testgraphs.tar.Z. We randomly generated similar graphs to train our test subjects. We also generated our own random 15×10 graphs, with between 213-223 edges, for the second set of experiments described below. For Protein, we created random sequences of amino acids of length 100 (each acid had 50% chance of being hydrophobic) and allowed them to be positioned on a 30×30 grid. For Jobshop, we used the "swv00"-"swv10" instances of size 20×10 and 20×15 (Storer, Wu, & Vaccari, 1992) and the four "yn1"-"yn4" instances of size 20×20 (Yamada & Nakano, 1992) available at http://www.ms.ic.ac.uk/info.html.

All experiments were performed with unoptimized Java code on a 1000 MHz PC. All user experiments were performed on a tabletop projected display, as was done in the original HuGSS experiments (Anderson *et al.*, 2000).

	Delivery		Crossing	
	10 min. guided tabu	10 min. guided greedy	10 min. guided tabu	10 min. guided greedy
unguided tabu	61	29	79	25
unguided greedy	>150	>150	>150	135

Table 2: Average number of minutes of unguided search required to match or beat the result produced by 10 minutes of guided search.

Experiments with unguided search

By *unguided search*, we mean running either the tabu or exhaustive algorithm without intervention and with all elements set to high mobility.

We performed experiments to evaluate our method for encouraging diversity of the tabu search. For each application, we ran the unguided tabu search with various minimum-diversity settings. We ran the search on 10 problems for five minutes with a fixed memory size of 10. The results, shown in Table 1 show that for each application, forcing the algorithm to make diverse moves improves the search, but that forcing too much diversity can hinder it.

We also compared exhaustive search to tabu search. We used the greedy variant of exhaustive search because the steepest-descent variant is ineffective when starting from poor initial solutions. As also shown in Table 1, with a reasonably well-chosen diversity setting, unguided tabu significantly outperforms unguided greedy search.

Finally, as an external comparison, we ran a suite of standard graph-layout heuristics (Gutwenger *et al.*, 2002) on the Crossing problem instances. The average best score was 36.63, which is slightly better than unguided tabu's best score of 37.9 from Table 1. (For these smaller instances, the optimal solutions have been computed (Kuusik, 2000) and average to 33.13.)

User studies

The goal of these experiments was to compare human-guided tabu search to unguided tabu search and to human-guided exhaustive search.

In our first set of experiments, we trained test subjects for 2-4 hours on how to use our system. We used the visualization modes in order to teach the subjects how the algor-

	Delivery					Crossing				
minutes	W	L	T	ave win	ave loss	W	L	T	ave win	ave loss
10	16	4	0	1.76	0.85	14	3	3	3.21	4.67
20	10	10	0	1.10	1.06	11	6	3	2.64	5.67
30	10	10	0	0.95	1.27	11	6	3	2.55	5.83
60	8	12	0	0.86	1.38	10	8	2	2.70	6.25
90	8	12	0	0.80	1.46	10	8	2	2.70	7.00
120	6	14	0	0.69	1.48	9	9	2	2.33	6.89
150	4	16	0	0.6	1.42	9	9	2	2.33	6.89

Table 3: The number of wins (W), losses (L), and ties (T) when comparing the result of 10 minutes of human-guided tabu search to 10 to 150 minutes of unguided tabu search, as well as the average difference of the wins and losses.

ithms work and how tabu uses its minimum-diversity feature. Each subject performed five 10-minute trials using our system with only our GTABU algorithm and five 10-minute trials with only exhaustive search. The test subjects were students from nearby selective universities: our goal is to show that *some* people can guide search, not that most people can.

We used the same 10 problem instances for every subject. Half the subjects did the tabu trials first, and half did the exhaustive-search trials first. For each problem instance, half the subjects used tabu and half used exhaustive. For this first experiment, we fixed the minimum diversity of tabu to be the one that produced the best results in preliminary experiments on random problems for each application.

To evaluate each result, we compared it to 2.5 hours of unguided tabu search on the same problem. Table 2 shows the number of minutes required by unguided tabu and unguided greedy, on average, to produce an equal or better solution to the one produced by 10 minutes of guided search. As shown in the table, it took, on average, more than one hour for unguided tabu search to match or beat the result of 10 minutes of guided tabu search. Furthermore, the results of guided tabu were substantially better than those of guided greedy, as can be seen by the fact that unguided tabu overtakes the results of guided greedy search much more quickly.

Table 3 shows a detailed comparison of the result of 10 minutes of guided tabu search to between 10 and 150 minutes of unguided tabu search. The win and loss columns show how often the human-guided result is better and worse, respectively. The table shows that for Crossing, 10 minutes of guided search produced better results than 2.5 hours of unguided search in nine of 20 instances and tied in two. When guided search loses, however, it does so by more, on average, than it wins by. Incidentally, some test subjects consistently performed better than others. We plan to study individual performance characteristics more fully in future work.

We ran a second experiment on the larger instances to determine if experienced users could improve on highly optimized solutions produced by unguided search. Prior to the user sessions, we ran unguided tabu search for five hours to precompute a starting solution. The test subjects then tried to improve this solution using guided tabu for one half hour. The authors of this paper were among the test subjects for this experiment. Because the users were experienced, they were allowed to modify the minimum diversity setting. As shown in Table 4, users were able to improve upon the solutions more in half an hour than unguided tabu did in 10 hours, although by small amounts. (In Crossing and Delivery, the users outperformed or matched unguided tabu in all but one case. In that case, however, tabu found a significantly better solution.) We again ran the graph-layout heuristics on these larger Crossing problem instances. The average best score was 252.13; here guided search slightly outperforms the heuristics.

	num trials	initial solution	after 30 minutes guided	after 300 minutes unguided	after 600 minutes unguided
Delivery	4	61.46	60.86	60.97	60.94
Crossing	8	253.13	251.13	251.5	250.75
Jobshop	6	958	952.33	954	954

Table 4: The initial solution was computed with 5 hours of unguided tabu search. We compare a half hour of guided search to an additional 5-10 hours of unguided search.

We also performed an initial investigation with the Protein application. As one anecdotal example, we applied our system to a hard instance with best known score of -49 (Bastolla *et al.*, 1998). Five hours of unguided tabu produced a solution with score -47; one of the authors was able to guide tabu to improve this solution yielding a score of -48 in under an hour.

Informal observations

While each application had its own unique "feel," there were several common characteristics and general strategies for guiding tabu search in the four applications. A common pattern, for example, is for the user to try to escape a deep local minimum by making several manual moves. These moves often cause the score to become temporarily worse, but reinvoking the algorithm usually improves the solution. Mobilities are sometimes used to prevent the algorithm from returning to its previous local minimum. This approach fails to produce a new best solution more often than not, but a series of attempts often yields a new best solution. An efficient approach is to plan the next attempt while the computer is working on the current attempt.

In general, the user looks for combinations of moves that the computer would not consider at once. For example, in Delivery, it is a good strategy to remove a cluster of customers that are, as a whole, far from the route, set them to low mobility, and reinvoke the tabu search. The computer would not readily explore this option because removing one or two customers at a time would not significantly reduce the distance of the route. Similarly, in Crossing, the user often looks for nearly independent clusters of nodes which can all be moved to a new location. In Jobshop, it is common to move several operations earlier (or later) on their machines in order to give the machine more flexibility.

For the 10-minute user tests, an important strategy was to let the tabu search run uninterrupted for the first minute or two, since it would most often make its biggest improvements in the early part of the search. The test subjects had a

harder time learning how to guide the search for the Crossing application than the Delivery application and uniformly spent more hours practicing before they felt comfortable to try the test cases. They seem to have reached a higher level of mastery, however.

Conclusions

We have presented GTABU, a guidable tabu search algorithm, and experiments to verify its effectiveness. To our knowledge, no previous human-guided tabu algorithm has been published previously.

GTABU represents a clear advance in interactive optimization. Because tabu search generally provides a more powerful search strategy than exhaustive search, a human-guidable tabu search can provide better solutions while still enjoying the advantages of involving people in the optimization process. Our experiments confirm that people can understand and control GTABU, and that guided tabu search outperforms guided exhaustive search.

GTABU also shows the potential for human interaction to improve on automatic optimization. Our experiments demonstrate that guided tabu outperforms unguided tabu in several domains; in particular, small amounts of human guidance can be as valuable as substantial amounts of unguided computer time.

Acknowledgements

We would like to thank all the participants in our user study. Michael Mitzenmacher's contribution was supported in part by NSF CAREER Grant CCR-9983832 and an Alfred P. Sloan Research Fellowship, and done while visiting Mitsubishi Electric Research Laboratories.

References

Aarts, E.; Laarhoven, P. v.; Lenstra, J.; and Ulder, N. 1994. A computational study of local search algorithms for job-shop scheduling. *ORSA Journal on Computing* 6(2):118–125.

Anderson, D.; Anderson, E.; Lesh, N.; Marks, J.; Mirtich, B.; Ratajczak, D.; and Ryall, K. 2000. Human-guided simple search. In *Proc. of AAAI 2000*, 209–216.

Bastolla, U.; Frauenkron, H.; Gerstner, E.; Grassberger, P.; and Nadler, W. 1998. Testing a new Monte Carlo algorithm for protein folding. *PROTEINS* 32:52–66.

Chien, S.; Rabideau, G.; Willis, J.; and Mann, T. 1999. Automating planning and scheduling of shuttle payload operations. *J. Artificial Intelligence* 114:239–255.

Colgan, L.; Spence, R.; and Rankin, P. 1995. The cockpit metaphor. *Behaviour & Information Technology* 14(4):251–263.

Dill, A. K. 1985. Theory for the folding and stability of globular proteins. *Biochemistry* 24:1501.

Eades, P., and Wormald, N. C. 1994. Edge crossings in drawings of bipartite graphs. *Algorithmica* 11:379–403.

Feillet, D.; Dejax, P.; and Gendreau, M. 2001. The selective Traveling Salesman Problem and extensions: an overview. TR CRT-2001-25, Laboratoire Productique Logistique, Ecole Centrale Paris.

Gleicher, M., and Witkin, A. 1994. Drawing with constraints. *Visual Computer* 11:39–51.

Glover, F., and Laguna, M. 1997. *Tabu Search*. Kluwer academic publishers.

Gutwenger, C.; Jünger, M.; Klau, G. W.; Leipert, S.; and Mutzel, P. 2002. Graph drawing algorithm engineering with AGD. In Diehl, S., ed., *Software Visualization: State of the Art Survey. Proc. of the International Dagstuhl Seminar on Software Visualization, Schloss Dagstuhl, May 2001*, volume 2269 of *Lecture Notes in Computer Science*. Springer. To appear.

Klau, G. W.; Lesh, N.; Marks, J.; Mitzenmacher, M.; and Schafer, G. T. 2002. The HuGS platform: A toolkit for interactive optimization. *To appear in Advanced Visual Interfaces 2002.*

Kuusik, A. 2000. *Integer Linear Programming Approaches to Hierarchical Graph Drawing*. Ph.D. Dissertation, Department of Computer Science and Information Systems, University of Limerick, Ireland.

Nascimento, H. d., and Eades, P. 2001. User hints for directed graph drawing. *To appear in Graph Drawing*.

Nelson, G. 1985. Juno, a constraint based graphics system. *Computer Graphics (Proc. of SIGGRAPH '85)* 19(3):235–243.

Ryall, K.; Marks, J.; and Shieber, S. 1997. Glide: An interactive system for graph drawing. In *Proc. of the 1997 ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST '97)*, 97–104.

Scott, S.; Lesh, N.; and Klau, G. W. 2002. Investigating human-computer optimization. *To appear in CHI 2002.*

Sims, K. 1991. Artificial evolution for computer graphics. *Comp. Graphics (Proc. of SIGGRAPH '91)* 25(3):319–328.

Storer, R.; Wu, S.; and Vaccari, R. 1992. New search spaces for sequencing instances with application to job shop scheduling. *Management Science* 38:1495–1509.

Todd, S., and Latham, W. 1992. *Evolutionary Art and Computers*. Academic Press.

Waters, C. 1984. Interactive vehicle routeing. *Journal of Operational Research Society* 35(9):821–826.

Yamada, T., and Nakano, R. 1992. A genetic algorithm applicable to large-scale job-shop instances. In *Parallel instance solving from nature 2*. North-Holland, Amsterdam: R. Manner, B. Manderick(eds). 281–290.