Human-Guided Simple Search

**David Anderson[1], Emily Anderson[1], Neal Lesh[1], Joe Marks[1],
Brian Mirtich[1], David Ratajczak[1], Kathy Ryall[1,2]**

[1]MERL — Mitsubishi Electric Research Laboratory, Cambridge, MA 02139
[2]University of Virginia, Dept. of Computer Science, Charlottesville, VA 22903
Contact: lesh@merl.com

Abstract

Scheduling, routing, and layout tasks are examples of hard operations-research problems that have broad application in industry. Typical algorithms for these problems combine some form of gradient descent to find local minima with some strategy for escaping nonoptimal local minima. Our idea is to divide these two subtasks cleanly between human and computer: in our paradigm of *human-guided simple search* the computer is responsible only for finding local minima using a simple hill-climbing search; using visualization and interaction techniques, the human user identifies promising regions of the search space for the computer to explore, and intervenes to help it escape nonoptimal local minima. We have applied our approach to the problem of capacitated vehicle routing with time windows, a commercially important problem with a rich research history. Despite its simplicity, our prototype system is competitive with the majority of previously reported systems on benchmark academic problems, and has the advantage of keeping a human tightly in the loop to handle the complexities of real-world applications.

Introduction

Most previous research on scheduling, routing, and layout problems has focused on developing fully automatic solution methods. There are, however, at least two reasons for developing cooperative, interactive systems for optimization problems like these. First, human users may have knowledge of various amorphous real-word constraints and objectives that are not represented in the objective function given to computer algorithms. In vehicle-routing problems, for example, human experts may know the flexibility or importance of certain customers, or the variability of certain routes. The second reason to involve people in the optimization process is to leverage their abilities in areas in which humans (currently) outperform computers, such as visual perception, learning from experience, and strategic assessment. Although both motivations seem equally important, we have used the second, more quantitative consideration to drive our current round of research.

In this paper, we present a new cooperative paradigm for optimization, human-guided simple search (HuGSS). In our current framework, the computer performs a very

simple, hill-climbing search. One or more people interactively "steer" the search process by repeatedly initiating focused searches, manually editing solutions, or backtracking to previous solutions. When invoking a focused search, the user determines which modifications to the current solution should be considered, how to evaluate them, and what type of hill-climbing search to use.

We have designed and implemented a prototype system that supports HuGSS for the *capacitated-vehicle-routing-with-time-windows* (CVRTW) problem. Below, we describe the CVRTW problem and our prototype, and report results from 48 hours of controlled testing with our system.

Sample Application

Problem Description and Definitions

We chose vehicle routing as our initial problem domain for three reasons: it is commercially important; it has a rich research history, which facilitates comparison with previous work; and routing problems are ones for which the human capabilities of vision, learning, and judgment should be useful. In the CVRTW problem (Solomon 1987), trucks deliver goods from a single central depot to customers at fixed geographic locations. Each customer requires a certain quantity of goods, and specifies a time window within which delivery of the goods must commence. All trucks have the same capacity, and travel one unit of distance in one unit of time. Delivery takes a constant amount of time, and each customer can receive only one delivery. All trucks must return to the depot by a fixed time. A *solution* to a CVRTW problem is an ordered list of customers assigned to each truck, and is *feasible* if it satisfies all the constraints. The optimization problem is first to minimize the number of trucks required to construct a feasible solution; and second to minimize the total distance traveled by the trucks.

As we describe below, users can force the system to consider *infeasible* solutions. Thus we needed to extend the classical objective function for CVRTW to rank infeasible as well as feasible solutions. We define the *maximum lateness* of a truck as the maximum tardiness with which it arrives at any of its customers; or if a truck has insufficient capacity to service its customers, we assign it an infinite maximum-lateness value. We optimize infeasible solutions by minimizing the sum of the maximum latenesses over all

the routes. We rank any feasible solution as better than any infeasible solution.

We define a *1-ply* move as the transfer of a customer from its current route onto another route. Such a move requires that both routes be re-optimized for distance (if feasible) or maximum lateness (if infeasible).[1] An *n*-ply move is simply a combination of *n* 1-ply moves.

HuGSS for CVRTW

In our system, the user controls the optimization process by performing the following three actions:

1. Edit the current solution by making a 1-ply move.

2. Invoke a focused local search, starting from the current solution. The user controls which *n*-ply moves are considered, how they are evaluated, and what type of search is used.

3. Revert to an earlier solution, or to an initial seed solution generated randomly prior to the session.

We now describe each type of action in the context of our implemented system, followed by a description of the visualization and interface (see Figures 1 and 2) that support these actions.

Manual edits: To edit the current solution manually, the user simply selects a customer and a route. The system transfers the customer to the route and re-optimizes both affected routes. Moving the last customer off a truck's route eliminates that truck. Also, the user can create infeasible solutions by assigning customers with conflicting constraints, or with too much total demand, to a single truck.

Focused searches: The principal feature of our system is the following set of methods for allowing users to repeatedly invoke deep, focused searches into regions of the search space they feel are promising. The user determines which moves the hill-climbing engine will evaluate by:

- *Setting a priority (high, medium, or low) for each customer.* The user controls which customers can be moved, and the routes onto which they can be moved, by assigning priorities to them. The search engine will only consider moving high-priority customers, and only consider moving them onto routes that have no low-priority customers. For example, the user can restrict the search engine to exchanging customers between a pair of routes by setting all the customers on those routes to high priority and all other customers to low priority.

- *Deciding which n-ply moves (1-ply to 5-ply) to enable.* In general, deeper searches are more likely to produce good results, but take more time.

- *Setting an upper bound on the number of moves that the computer can consider.* The search is stopped when all enabled moves have been considered, or when this user-supplied upper limit is reached.

---

[1]Computing the route for a truck once customers have been assigned to it is an instance of the Traveling Salesman Problem with Time Windows. Although an NP-hard problem, the instances that arose in our experiments are small enough that exhaustive search is practical.

Figure 1: The Optimization Table.

Focusing the search dramatically reduces the number of moves that the search engine evaluates. In one example from our experiments, we focused the search on two of 12 routes (20 of 100 customers), which decreased the number of 1-ply moves considered by a factor of 30, 2-ply moves by a factor of 222, and 3-ply moves by a factor of 18,432.

In addition to determining which moves are evaluated, the user determines how they are evaluated by selecting an objective function. We currently support two objective functions: the standard CVRTW objective function modified to assess infeasible solutions; and a function we call *minimize-routes*, which removes $2 \times len^2$ from the cost attributed to each route that contains $len < 6$ customers. The idea behind this objective function is to encourage a short route to become shorter, even if it increases the total distance traveled, in the hope of eventually eliminating that route.

Finally, the user can select between greedy or steepest-descent search mode. In greedy mode, the search engine immediately adopts any move that improves the current solution under the given objective function. It considers 1-ply moves (if enabled) first, then 2-ply moves (if enabled), and so on. Within a ply, the moves are evaluated in a random order. As soon a move is adopted, the search engine begins, again, to evaluate 1-ply moves.

In steepest-descent mode, moves are considered in the same order as in greedy mode, but only the best move is adopted. The best move is defined as the one that decreases the cost of the solution the most, under the given objective function. If no move decreases the cost of the solution, then the best move is the one that increases the cost the least.[2] Making the least-bad move provides useful information to the user, and can always be undone by reverting to the previous solution.

Switching among candidate solutions: The third type of action the user can perform is to switch candidate solutions, either to backtrack to a previous solution, or to load a precomputed, "seed" solution. The seed solutions are generated prior to the session using our hill-climbing search engine. They are intended to be used both as starting points for finding more optimal solutions and to give users a sense of how various combinations of customers can be serviced.

---

[2]However, we never adopt a move that increases the infeasibility of a solution. Finding and ranking all infeasible moves is not worth the added computational expense.
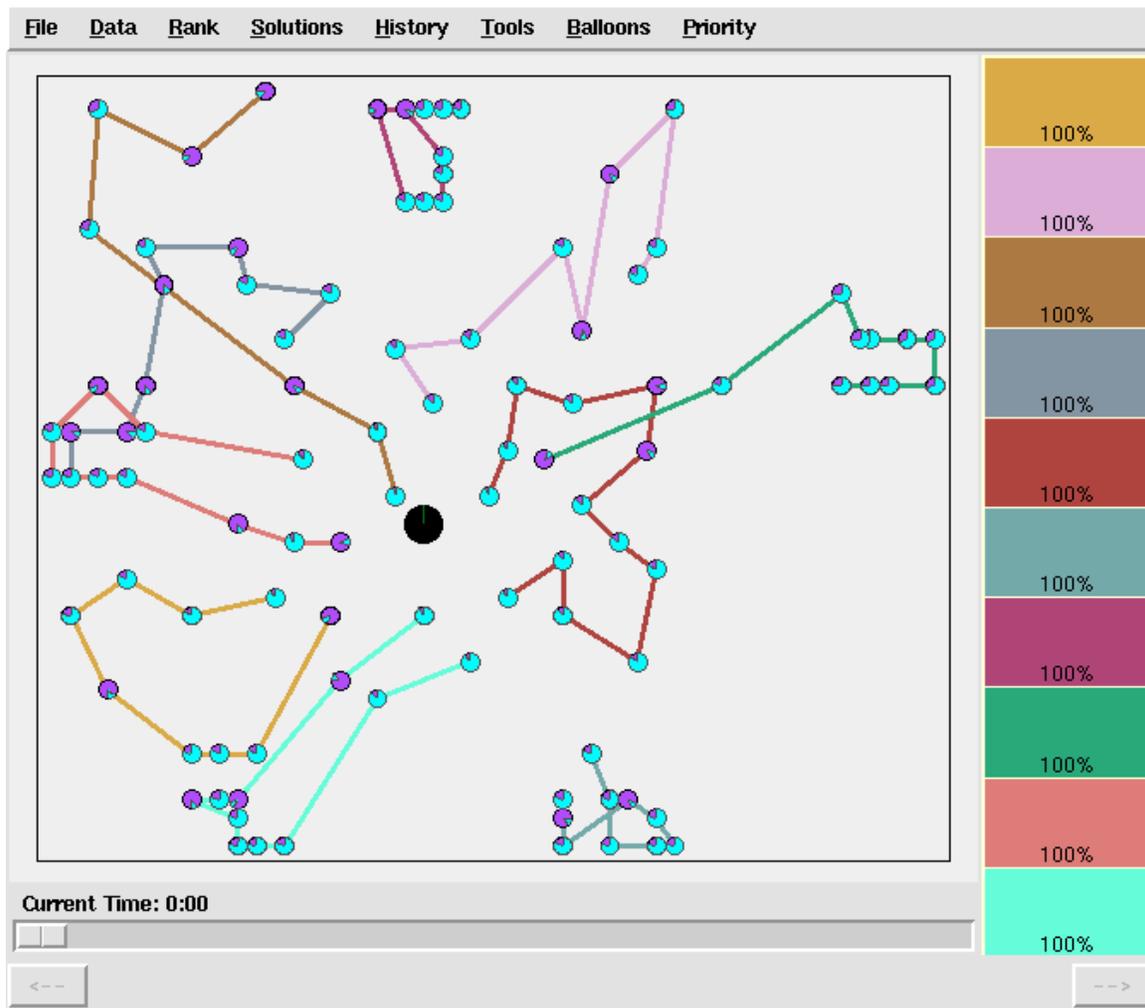
Figure 2: A snapshot of our interface.

Interface and Implementation

For our initial implementation we have used a tabletop display, which we call the *Optimization Table* (see Figure 1). We project an image down onto a whiteboard. This allows users to annotate candidate solutions by drawing or placing tokens on the board, a very useful feature. In addition, several users can comfortably use the system together.

For this kind of problem, creating an effective visualization is an intrinsic challenge in bringing the human into the loop. Figure 2 shows our attempt to convey the spatial, temporal, and capacity-related information needed for CVRTW. The central depot is the black circle at the center of the display. The other circles represent customers. The pie slices in the customer circles indicate the time windows during which they are willing to accept delivery. The truck routes are shown by polylines, each in a different color. At the user's option, the first and last segments of each route can be hidden, as they are in the figure, to avoid visual clutter around the depot. The search-control operations described in the previous subsection are supported by mouse operations and pull-down menus. Detailed information about in-

dividual customers and trucks can also be accessed through standard interface widgets.

The interface was written in Tcl, and the hill-climbing algorithm in C++. We use a branch-and-bound algorithm to optimize truck routes during move evaluation. We carefully crafted several pruning rules and caching procedures to streamline this algorithm.

Experimental Investigation

Four test subjects participated in our experiments. Three of them are authors of this paper. The fourth tester is a Ph.D. student unaffiliated with this project, who received five hours of training prior to his first test.

The Solomon datasets (Solomon 1987) were our source of benchmark CVRTW problems for our experiments. This corpus consists of 56 problem instances, each with 100 customers, divided into three categories according to the spatial distribution of customers: C-type (clustered), R-type (random), and RC-type (a mix of the two.) There are two problem sets for each category: the C1, R1, RC1 sets have a

narrow scheduling horizon, while the C2, R2, and RC2 sets have a large scheduling horizon.

As we developed and refined our system, we tested users informally on a selection of R1 and RC1 problems. In the second, more controlled, phase of experimentation, we ran two tests on each of the RC1 problems. During this phase, subjects worked only on problem instances to which they had no previous exposure. In each test, the user spent 90 minutes working on the problem without reference to the precomputed seed solutions. Then, after an arbitrarily long break, the user spent another 90 minutes working on the same problem, this time with the precomputed seed solutions available for perusal. We recorded logs for a total of 79.4 hours of test sessions, 48 hours of which were the controlled experiments.

We generated the seed solutions using the settings we found to be the most effective on a small sample of the Solomon problem instances. In particular, we used greedy search with 1-ply and 2-ply moves enabled and all customers set to high priority; we used the minimize-routes objective function, and started the search from an initial solution in which each customer is assigned its own truck, and searched until we reached a local optimum. Multiple runs produce varied results due to the random order in which moves are considered in the greedy search. We ran the algorithm repeatedly until we had generated 1000 solutions or a 10-hour time limit was reached. On average, it took 8.4 hours to generate the seed solutions for a problem. We ran all our experiments on a 500 MHz PC.

Observations

User strategies: During a session, the user repeatedly invokes the hill-climbing engine to perform focused searches. This simple mechanism supports a surprisingly broad range of optimization strategies. For example, consider the goal of truck reduction. A user might start by browsing the precomputed seed solutions for one with a "vulnerable" route, e.g., one that might be eliminated because it has a small number of loosely constrained customers, and nearby routes that have available capacity and slack in their schedules. Having identified such a solution, the user can shift customers off the vulnerable route by invoking a steepest-descent search: setting the route's customers to high priority and the customers of nearby routes to medium priority will cause the search algorithm to return the least costly feasible move of a customer off the vulnerable route and onto one of the nearby routes. An alternative strategy for shortening and eliminating routes is to set all the customers in the neighborhood of a vulnerable route to high priority, and to use the minimize-routes objective function and a high search ply: a search with these parameters would consider compound moves, involving multiple customers on different routes, that have the net effect of shortening the vulnerable route. A third alternative, which users often had to resort to, is to manually move a customer off a vulnerable route, even if the move produces an infeasible solution; fixing the resulting infeasibility then becomes a subproblem for which there is another suite of strategies.

User behaviors: During test sessions, our users spent more

User	Moves per hour	Searches per hour	Percent steep searches	Percent in infeasible space
A	53	47	30	78
B	46	53	99	52
C	107	101	87	60
D	26	72	99	76

Table 1: User styles: action and mode

User	Customer priority			Search ply used				
	high	med.	low	1	2	3	4	5
A	34	50	16	83	84	87	84	83
B	16	8	77	100	95	81	76	65
C	17	13	70	94	89	53	26	11
D	40	29	31	99	99	39	10	0

Table 2: User styles: depth and focus. The numbers indicate the fraction of customers assigned high, medium, or low priorities, and the frequency with which the various ply moves were enabled. E.g., on average, subject A assigned 34% of the customers to have high priority, and included 3-ply moves 87% of the time.

time thinking than the search algorithm spent searching. On average, the search algorithm was in use 31% of the time; the range was 11% to 61%. Solution improvements were made throughout the sessions. Averaging over all the test runs, a new best solution was found a little over five times per hour. Of course, improving the current solution was much more common than finding a new best solution. Focused searches yielded an average of 23 improvements per hour, and manual adjustment yielded an average of 20 improvements per hour.

Tables 1 and 2 show what features of the system were used, as well as how usage varied among the test subjects. (Note that some of the variation is very likely due to differences in the nature of the individual problems.) Three of the four users primarily used steepest-descent search instead of greedy search. We feel that steepest-descent mode was preferred largely because it makes the least-bad move if no good move is available, which turned out to be a very useful feature for shifting customers onto or off of specific routes. The minimize-routes objective function was almost never used. Everyone spent at least half of the time working on infeasible solutions. All four users made substantial use of 1-ply, 2-ply, and 3-ply searches, but only two users frequently used 5-ply search. There was a wide range among the users in terms of how often the different priorities were used, and in how many searches were invoked, on average, per hour.

During the controlled experiments, each user did better than some other user on at least one data set. The one user who was not an inventor of the system (User D in the tables) turned out to have the best record. He generated three of the eight best results on the RC1 problem instances, which are shown in Table 3.

Quantitative results

HuGSS vs. unguided simple search: Our results show that human guidance provided a significant boost to the simple

	Best found by simple search		Best found by human-guided simple search		Best pub. solution	
	Veh.	Dist.	Veh.	Dist.	Veh.	Dist.
RC101	15	**1631**	15	1662	14	1669
RC102	13	1499	**12**	1569	12	1555
RC103	11	1293	11	**1224**	11	1110
RC104	10	1156	10	**1136**	10	1136
RC105	14	1558	**13**	1691	13	1637
RC106	12	1407	**11**	1475	11	1432
RC107	11	1247	11	**1236**	11	1231
RC108	10	1191	10	**1185**	10	1140
Ave.	12.0	1373	11.63	1397	11.50	1364

Table 3: Best solutions found during 800 hours of simple search compared to 67.2 hours of precomputation and 79.4 hours of human-guided search. The best published solutions are shown for comparison.

search in almost all cases. Table 3 compares the best scores on the RC1 datasets found by the hill-climbing engine alone with the best scores found using the HuGSS system.[3] For the hill-climbing engine, the scores are the best found in approximately 100 hours of computation on a 500 MHz Pentium PC. The scores for the HuGSS system are the best found in at most 10 hours of precomputation and 10 hours of guided searching. (The table includes scores from all logged testing and training sessions, as well as those from the controlled experiments.) On three of the problems, the human-guided solution uses one fewer truck; on four of the five remaining problems, the human-guided solution has a lower distance value. The only dataset on which the unguided hill-climbing search prevailed was RC101, which is the most heavily constrained of all the problems. The very narrow time windows facilitate extremely fast computer searches (a new local optimum is found every six seconds), while making visualization more difficult.

The HuGSS results in Table 3 reflect the combined benefit of precomputed seed solutions and human-guided search. To tease these two factors apart, we considered the solutions produced by the first 90 minutes of each controlled experiment, during which precomputed seed solutions were not available to the user. In Table 4 we report these results in two ways: the average of the two scores available for each dataset represents what can be achieved with 1.5 hours of pure guided search (i.e., guided search without the benefit of precomputed seed solutions); the best of the two scores for each dataset represents what can be achieved in 3.0 hours of pure guided search, albeit using two people for separate 1.5-hour sessions. The table also shows the average results obtained by the hill-climbing engine without human guidance.[4] From this data we can conclude that 1.5 hours

---

[3]To interpret the scores correctly, it is important to recall that the primary objective is to minimize the number of trucks, which often works against the secondary concern of minimizing total distance traveled. Additionally, it is standard practice in the literature to report results by averaging the trucks and distances over many problem instances.

[4]We estimated the average value of computer-only search for *N* hours of computation by taking the best score found in *N* hours

of pure human-guided searching is comparable to about 5.0 hours of unguided hill climbing. However, 3.0 hours of pure guided searching is better than 20.0 hours of unguided hill climbing, which indicates that additional time is of more benefit to the guided regime than to the unguided one. The average score for 3.0 hours of guided search with precomputed seed solutions is also shown: the seed solutions impart a distinct benefit, but are not the sole factor behind the dominance of HuGSS over unguided simple search.

	Time	Veh.	Dist.
Our hill-climbing search engine alone	1 hour	12.35	1424
	2 hours	12.23	1416
	5 hours	12.15	1403
	8.4 hours	12.13	1390
	20 hours	12.06	1388
HuGSS (w/out seeds)	1.5 hours	12.13	1432
	3 hours	12.00	1413
HuGSS (with seeds)	10 hours precomputation and 3 hours guided search on 500 MHz machine	11.88	1389
HuGSS (pilot experiments with newest system)	90 min. precomputation and 90 min. guided search on 500 MHz machine	11.88	1380
Carlton'95[a]	-	13.25	1402
Rochat and Taillard'95	44 min. on 100 MHz machine	12.38	1369
Chiang and Russell'97[b]	-	11.88	1397
Taillard *et. al.*'97	3.1 hours on 50 MHz machine	11.88	1381
De Backer and Furnon'97	-	14.25	1385
Shaw'98	1 hour on 100 MIPS machine	12.00	1361
Shaw'98	2 hours on 100 MIPS machine	12.00	1360
Cordone and Wolfer-Calvo'98[c]	12.1 min on 18 Mflops Pentium	12.38	1409
Gambardella and Taillard'99	30 min on 167MHz, 70 Mflops Sun UltraSparc	11.92	1388
Kilby, Prosser and Shaw'99[c]	48.3 min. on 25 Mflops/s Digital Alpha	12.12	1388
Homberger and Gehring'99	5 hours on 200 MHz machine	11.5	1407
Best published solutions	About 15 years on multiple machines	11.5	1364

[a] As reported by (Taillard *et al.* 1997).
[b] As reported in (Homberger & Gehring 1999).
[c] As reported in (Gambardella, Taillard, & Agazzi 1999).

Table 4: Reported results. The numbers are averages over the eight instances in Solomon's RC1 problem set.

---

of computation randomly sampled from the 100 hours of unguided search we recorded for each problem instance. We repeated this 1000 times for each problem and report the average result.

HuGSS vs. state-of-the-art techniques: The Solomon datasets are a very useful benchmark for comparing all the different heuristic-search techniques that have been applied to the CVRTW problem, including tabu search and its variants, evolutionary strategies, constraint programming, and ant-colony optimization. Table 4 includes performance data for these techniques and others. The scores we obtained with the full HuGSS approach (i.e., with precomputed seed solutions) are competitive with those obtained by the state-of-the-art techniques, dominating several of them, and being clearly dominated only by the results from a recent genetic algorithm (Homberger & Gehring 1999).

However, the full HuGSS technique uses between one and two orders of magnitude more computational effort than other techniques. Other algorithms may benefit from a comparable amount of computation, but there is not enough information in the cited papers to accurately assess how much benefit to expect, if any.

To test whether the HuGSS approach for this problem can be effective with less computational effort, we ran a pilot set of experiments with the latest version of our system (its improvements over the system described above are listed in the concluding section of this paper). In these experiments, we used only 90 minutes of precomputation and 90 minutes of guided search. We ran one test per problem, with three of the test subjects from the first set of experiments. (In some cases, the subjects worked on a problem instance that they had worked on some months earlier.) As shown in Table 4, we achieved comparable results with our new system with significantly less computational and human effort, thus closing the gap with the state-of-the-art systems.

In summary, these results suggest that human guidance can replace the painstakingly crafted, problem-specific heuristics that are the essence of other approaches without significant compromise in the quality of the results.

Versatility

Because the user is directing the search, our system can be used for tasks other than the classic CVRTW optimization task. For example, it can be used to balance routes. Many of the best solutions found by state-of-the-art methods might be unsuitable for real use because they assign only one or two customers to a truck. The users of our system can direct the hill-climbing engine to find the lowest cost way of moving N customers to a particular truck, by only enabling N-ply moves and setting the priorities so that the search engine only considers moving customers onto the target truck.

Alternatively, it may be desirable to have a lightly loaded truck as a backup if other trucks encounter significant delays. This can be accomplished by the same means used in attempting to eliminate a truck. Similarly, if there simply are not enough trucks to satisfy all the customers' needs, our system can be used to explore various infeasible options. It is often easy to shift the infeasibility around the board, if in fact some customers are more flexible than others.

Of course, other algorithms might be modified to solve any of these tasks. The ability of our system to handle these tasks without any recoding (or even recompiling!) suggests that it will be more effective at handling new tasks as they arise. Furthermore, it demonstrates that our system can be used to pursue an objective function that is known by the human users but is difficult to describe to the computer algorithm. In this regard, HuGSS is distinctly more versatile than the algorithms cited in Table 4.

Related Work

The HuGSS paradigm is one way of dividing the work between human and computer in a cooperative optimization or design system. Other interface paradigms organize the cooperation differently.

In an iterative-repair paradigm, the computer detects and resolves conflicts introduced by the human user. In a system for scheduling space-shuttle operations (Chien *et al.* 1999), the computer produces an initial schedule that the user iteratively refines by hand. The user can invoke a repair algorithm to resolve any conflicts introduced.

Another way for the computer to address conflicts or constraint violations is to not let the user introduce them in the first place. Constraint-based interfaces are popular in drawing applications, e.g., (Nelson 1985; Gleicher & Witkin 1994; Ryall, Marks, & Shieber 1997). Typically the user imposes geometric or topological constraints on a nascent drawing such that subsequent user manipulation is constrained to useful areas of the design space.

The interactive-evolution paradigm offers a different type of cooperation: the computer generates successive populations of novel designs based on previous ones, and the user selects which of the new designs to accept and which to reject (Kochhar & Friedell 1990; Sims 1991; Todd & Latham 1992).

A related but very different line of inquiry takes human-human collaboration as the model for cooperative human-computer interaction, e.g., (Ferguson & Allen 1998). The emphasis in this work is on mixed-initiative interaction between the user and computer in which the computer has some representation of the user's goals and capabilities, and can engage the human in a collaborative dialogue about the problem at hand and approaches to solving it.

The HuGSS paradigm differs significantly from the iterative-repair, constraint-based, and interactive-evolution paradigms in affording the user much more control of the optimization/design process. By setting customer priorities and specifying the scope of the local search, the user decides how much effort the computer will expend on particular subproblems. And there are no dialogue or mixed-initiative elements in our system: the user is always in control, and the computer has no representation of the user's intentions or abilities.

Other researchers have also allowed a user to interact with a computer during its search for a solution to an optimization or constraint-satisfaction problem, e.g., (Choueiry & Faltings 1995; Smith, Lassila, & Becker 1996); one group has even applied this idea to a vehicle-routing problem (Bracklow *et al.* 1992). We believe, however, that HuGSS embodies a stronger notion of human guidance than previous

efforts. Furthermore, our work is the first rigorous investigation of how human guidance can improve the performance of an optimization algorithm.

Future Work And Conclusions

The contributions of this work are novel mechanisms for the interactive control of simple search, an application of these mechanisms to a vehicle-routing problem, and an empirical study of that application.

We are currently making our hill-climbing engine more efficient and our interface more interactive. The user now receives feedback from the hill-climbing engine that indicates the current depth of the search and the best move found to that point. The user can halt the search at any time, at which point the system returns the best solution found so far. This gives the user a much higher degree of control of the system and effectively removes the need to decide, in advance, the search depth, the maximum number of moves to evaluate, and blurs the distinction between greedy and steepest-descent search. Our pilot experiments (see Table 4) indicate that these changes greatly improve our system.

We had two principal motivations for investigating human-guided search: to exploit human perceptual and pattern-recognition abilities to improve the performance of search heuristics, and to create more versatile tools for solving real-world optimization problems. Our initial investigations show that human guidance improves simple hill-climbing search to world-class levels for at least one optimization task. We are also encouraged by the system's pliability and transparency: users pursued a variety of strategies, developed their own usage styles, and were highly aware of what the search engine was doing and why.

The separation made in HuGSS between the human's and the computer's roles has several pleasant consequences. The optimization engine is more generic and reusable than those used in state-of-the-art, problem-specific systems; and many of the user-interface concepts are also easily generalized to other problems. This raises the possibility of developing a general toolkit for creating a family of human-guided optimization tools.

Acknowledgments

References

Bracklow, J. W.; Graham, W. W.; Hassler, S. M.; Peck, K. E.; and Powell, W. B. 1992. Interactive optimization improves service and performance for Yellow Freight system. *INTERFACES* 22(1):147–172.

Carlton, W. 1995. *A Tabu Search Approach to the General Vehicle Routing Problem*. Ph.D. Dissertation, The University of Texas at Austin, Texas.

Chiang, W.-C., and Russell, R. 1997. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS J. on Computing* 9:417–430.

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

Choueiry, B. Y., and Faltings, B. 1995. Using abstractions for resource allocation. In *IEEE 1995 International Conference on Robotics and Automation*, 1027–1033.

Cordone, R., and Wolfler-Calvo, R. 2000. A heuristic for the vehicle routing problem with time windows. *J. of Heuristics, forthcoming*.

De Backer, B., and Furnon, V. 1997. Meta-heuristics in constraint programming experiments with tabu search on the vehicle routing problem. In *Proc. of the 2nd Int'l Conference on Metaheuristics (MIC 97)*, 1–14.

Ferguson, G., and Allen, J. 1998. Trips: An integrated intelligent problem-solving assistant. In *Proc. 15th Nat. Conf. AI*, 567–572.

Gambardella, L.-M.; Taillard, E. D.; and Agazzi, G. 1999. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. Technical Report IDSIA-06-99, IDSIA.

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

Homberger, J., and Gehring, H. 1999. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFORMS J. on Computing* 37(3):297–318.

Kilby, P.; Prosser, P.; and Shaw, P. 1999. Guided local search for the vehicle routing problem with time windows. In *Meta-heuristics - Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers. chapter 32, 473–486.

Kochhar, S., and Friedell, M. 1990. User control in cooperative computer-aided design. In *Proc. of the 1990 ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST '90)*, 143–151.

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

Rochat, Y., and Taillard, E. D. 1995. Probabilistic diversification and intensification in local search for vehicle routing. *J. of Heuristics* 1(1):147–167.

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.

Shaw, P. 1998. A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, APES group, University of Strathclyde.

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

Smith, S.; Lassila, O.; and Becker, M. 1996. Configurable, mixed-initiative systems for planning and scheduling. In Tate, A., ed., *Advanced Planning Technology*. Menlo Park, CA: AAAI Press. ISBN 0-929280-98-9.

Solomon, M. M. 1987. Algorithms for the vehicle routing

		first session				second session					pilot experiments with new system		
	user	seedless		seeded		user	seedless		seeded				Veh.	Dist.
		Veh.	Dist.	Veh.	Dist.		Veh.	Dist.	Veh.	Dist.			Veh.	Dist.
RC101	C	15	1678	15	1666	B	15	1690	15	1662	C		15	1629
RC102	D	12	1617	12	1569	A	13	1528	13	1528	B		13	1559
RC103	B	11	1323	11	1293	D	11	1420	11	1224	A		11	1281
RC104	C	10	1136	10	1136	B	10	1146	10	1140	B		10	1148
RC105	D	14	1733	13	1691	A	15	1707	14	1582	B		13	1642
RC106	A	12	1591	12	1385	C	12	1395	12	1395	C		12	1397
RC107	B	11	1284	11	1236	D	11	1359	11	1242	C		11	1232
RC108	C	11	1134	10	1218	A	11	1193	10	1246	A		10	1148

Table 5: Detailed results of the experiments

and scheduling problems with time window constraints. *Operations Research* 35(2):254–265.

Taillard, E. D.; Badeau, P.; Gendreau, M.; Guertin, F.; and Potvin, J. 1997. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science 31* 170–186.

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

Appendix

Table 5 shows the actual scores attained during our controlled experiments and the pilot experiments for our new system.