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

Human-Guided Search for Jobshop Scheduling

Neal Lesh, Leonardo B. Lopes, Joe Marks, Michael Mitzenmacher, Guy T. Schafer

TR2002-43 January 2003

Abstract

We present an interactive jobshop scheduling application developed with the Human-Guided Search (HuGS) framework and toolkit. Our system leverages people's abilities in areas in which they currently outperform computers, and allows people to steer a computer towards effective jobshop schedules based on their knowledge of real-world constraints. Furthermore, users can better understand, justify, and modify schedules if they participate in their construction. Our prototype allows users to manually modify the current schedule, backtrack to previous schedules, and invoke, monitor, and halt a variety of search algorithms to find better schedules. These search algorithms include a variant of tabu search that users can focus and constrain by visually annotating elements of the schedule.

3rd International NASA Workshop on Planning and Scheduling for Space

Submitted June 2002.
Revised September 2002.

Human-Guided Search for Jobshop Scheduling

Neal Lesh[1], **Leonardo B. Lopes**[2], **Joe Marks**[1], **Michael Mitzenmacher**[3], **Guy T. Schafer**[4]
[1] Mitsubishi Electric Research Laboratories, {lesh,marks}@merl.com
[2] Northwestern University, leo@iems.nwu.edu
[3] Harvard University, michaelm@eecs.harvard.edu*
[4] Harvard University Extension School, gschafer@fas.harvard.edu

Abstract

We present an interactive jobshop scheduling application developed with the Human-Guided Search (HuGS) framework and toolkit. Our system leverages people's abilities in areas in which they currently outperform computers, and allows people to steer a computer towards effecitive jobshop schedules based on their knowledge of real-world constraints. Furthermore, users can better understand, justify, and modify schedules if they partcipate in their construction. Our prototype allows users to manually modify the current schedule, backtrack to previous schedules, and invoke, monitor, and halt a variety of search algorithms to find better schedules. These search algorithms include a variant of tabu search that users can focus and constrain by visually annotating elements of the the schedule.

1 Introduction

The Human-Guided Search (HuGS) project is a ongoing effort to investigate and develop interactive optimization systems. We have created applications for a variety of problems, developed general exhaustive and heuristic search algorithms that are amenable to human guidance, and studied people's ability to guide these search algorithms (Anderson *et al.*, 2000; Lesh, Marks, & Patrignani, 2000; Scott, Lesh, & Klau, 2002; Klau *et al.*, 2002b). Additionally, we have developed the HuGS Toolkit, which is Java software which supports the quick development of interactive optimization systems (Klau *et al.*, 2002a).

In this paper, we present an interactive jobshop scheduling application developed with the HuGS framework and toolkit. The HuGS framework provides the user a greater degree of control than previous interactive optimization approaches. It allows users to manually modify solutions, backtrack to previous solutions, and invoke, monitor, and halt a variety of search algorithms. More significantly, users can constrain and focus the search algorithms by assigning *mobilities*, which we describe below, to elements of the current solution.

The HuGS toolkit is Java software that helps developers quickly create interactive optimization systems. To create an application, the developer must provide domain-specific definitions of the class of possible problems and solutions, a set of possible transformations on solutions, and a graphical component to visualize the solutions. The toolkit contains generic versions of several, human-guidable search algorithms as well as code to maintain the current working solution, the mobilities, and the history of solutions.

We have developed an initial prototype system for interactively solving jobshop scheduling problems. After providing the background for the HuGS framework, we describe the domain-specific components of the Jobshop application, emphasizing the challenges of the visualization component. We also describe general extensions to the HuGS toolkit that arose while developing the jobshop application. Finally, we discuss our initial experiences with our prototype.

*Supported in part by NSF CAREER Grant CCR-9983832 and an Alfred P. Sloan Research Fellowship. This work was done while visiting Mitsubishi Electric Research Laboratories.

2 Background: HuGS

2.1 Prior applications

In this section, we briefly describe three additional applications of HuGS that serve as examples to describe the HuGS framework. For more information, see (Klau *et al.*, 2002b).

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 (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 the given maximum distance. A screenshot of the Delivery application is shown in Figure 1.

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.

2.2 Terminology

We use the following abstractions to allow a uniform description of the HuGS applications: *problems*, *solutions*, *moves*, and *elements*. A *problem* is an instance of the type of problem being optimized. For example, a Protein problem consists of a sequence of amino acids.

The goal of optimization is to find the best *solution* to the given problem. A Delivery solution, for example, is a sequence of customers. We assume that for each application there is a method for comparing any two solutions and that for any two solutions, one is better than the other or they are equally good. As mentioned in the introduction, however, we assume that this total ordering may merely approximate the real-world constraints and preferences known by the users. Additionally, for most applications, it is possible to create *infeasible* solutions which violate some of the constraints of the problem. For example, a Delivery solution may exceed the distance constraint.

For each application we have designed a set of possible *moves*, or transformations on solutions. Applying a move to a solution produces a new solution. For example, in the Crossing application, one possible move is to swap two adjacent nodes. For the Delivery applications, the moves include adding or removing customers from the current route.

Finally, we assume that each problem contains a finite number of *elements*. The elements of Crossing are the nodes, the elements of Delivery are the customers, and the elements of Protein are the amino acids. Each move is defined as operating on one 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. As with automatic optimization, deciding which moves to include is an important design choice for the developer of an optimization system.

2.3 Mobilities

Our system maintains and displays a single current solution, such as the ones shown in Figures 1 and 2. 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):

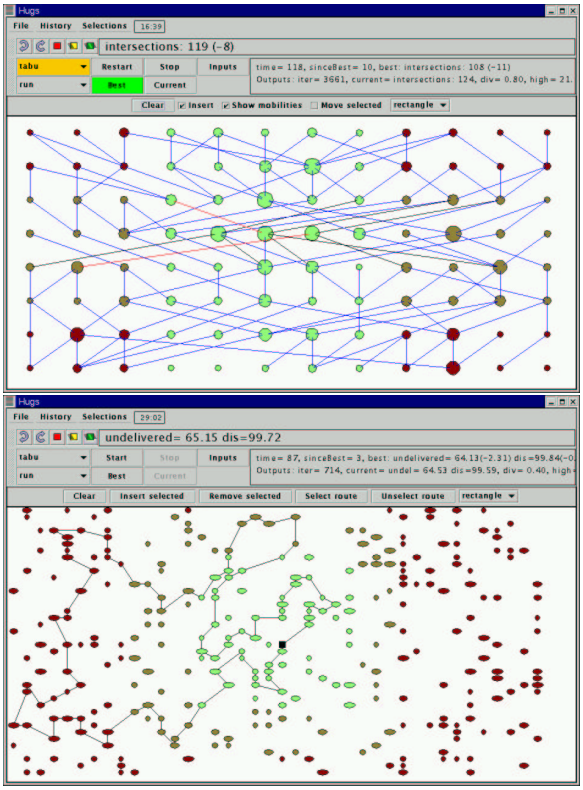
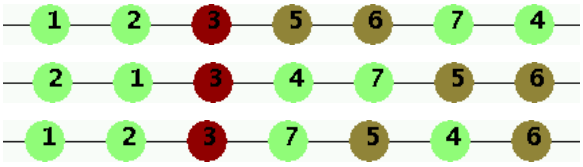


Figure 1: The Crossing and Delivery Applications.

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 useful in all of the applications described above.

2.4 Guidable Tabu

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).

GTABU a guidable tabu search algorithm developed for HuGS. 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 fact that the tabu algorithm controls its search by modifying mobilities provides several important benefits. First, applying GTABU requires no additional effort from the developer beyond what is needed to allow human guidance using mobilities. Second, users can more easily understand the progress and effects of the algorithm, by viewing the mobility settings at each iteration.

In Klau *et al.* (2002b) we describe experiments comparing guided tabu search to unguided (i.e., fully automatic) tabu search. The experiment included a total of seven test subjects, two domains, and 40 trials. The results indicate that 10 minutes of guided tabu search is comparable to, on average, 70 minutes of unguided tabu search.

2.5 Overview of User Actions

We now describe the full range of user actions in the HuGS framework. In our applications, the system always maintains a single, current working solution which is displayed to the users. The users try to improve the current solution by performing the following three actions:

1. manually choose a move to be applied to the current solution,
2. invoke, monitor, and halt a focused (via mobilities) search for a better solution,
3. revert to a previous or precomputed solution.

We now describe each type of action. The users can manually modify the current solution by performing any of the possible moves defined for the current application on the current solution. In many of our applications, a single user action on the GUI can invoke several moves. In the Delivery application, for example, the user can select multiple customers and remove them all with a single button press.

Users can also invoke a computer search for a better solution. The search algorithm starts from the current solution and explores the space of solutions that can be reached by applying moves which are allowed given the mobility assignments as described above. The users can invoke a variety of different search algorithms. In addition to GTABU described above, we currently provide steepest-descent and greedy exhaustive search algorithms. 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, up to some maximum ply. The steepest-descent algorithm keeps searching deeper and deeper for the move that most improves the current solution. The greedy algorithm immediately makes any move which improves the current solution and then restarts its search to try to improve the solution that results from applying that move. Our initial experience has been that tabu search outperforms exhaustive search but it seems useful to provide multiple search algorithms to the users.

After the users have invoked a search algorithm, they can monitor its progress to decide when to halt it. A text display shows the score of the best solution the search has found and how many seconds ago this solution was found. At any time, the user can query the search algorithm for either the best solution found so far or the current solution it is considering. This solution becomes the current visualized solution of the system. While the search is running the user can modify the current visualized solution or reassign mobility values to problem elements. The user can restart the search from these current settings, or halt the search.

Finally, the third type of user action is to revert to a previous solution. The system maintains a history of previous solutions, which can be browsed and adopted by the users. The GUI also provides menu commands to quickly undo or redo recent moves, as well as revert to the best solution seen so far. Additionally, the users can browse and adopt a set of solutions that were precomputed by the search algorithms prior to the interactive optimization session.

3 Jobshop Scheduling

3.1 Problem definition

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* which must be performed in a specified order. The ordered list of operations for each job is called an *itinerary*. Each operation must be performed by a particular machine. In our variation, every job has exactly one operation that must be performed on each machine. Each machine can process only one operation at a time.

A solution is a jobshop problem is a jobshop schedule that specifies a *roster* for each machine, which indicates the order in which the operations will be performed on that machine. Given a jobshop schedule, a simple, recursive algorithm can compute the earliest possible start time and end time for each operation: each operation starts as soon as all predecessors on its machine's roster and predecessors on its job's itinerary have completed. The end time of an operation is simply its start time plus its duration. The goal is to find a schedule which minimizes the time that the last job finishes, called the *makespan*.

In our application, we define problems, solutions, elements, and moves as follows. A Jobshop problem specifies

the number of jobs, the number of machines, an itinerary for each job, and the duration of each operation. A Jobshop solution is a roster for each machine. The Jobshop elements are the operations. We allow insertion moves which move one operation to an earlier or later position on its machine's roster and shifts the other operations accordingly. The search routines only consider shifting an operation by one position (i.e., swapping an operation with an operation it is adjacent to), but the users can manually perform any insertion move.

A useful concept in jobshop scheduling is called the *critical path*, which identifies the operations that are the bottleneck of the current schedule. The critical path can be recursively defined as follows: the operation(s) that finish last are on the critical path. Operation o_2 is on the critical path if there exists operation o_1 on the critical path such that o_2 immediately precedes o_1 on either its itinerary or roster, and o_2 ends at the same time that o_1 begins. It is impossible to improve the makespan of a schedule without changing the roster-position of at least one operation on the critical path. Therefore, our search algorithms only evaluate moves that alter operations on the critical path.

3.2 Jobshop visualization

The jobshop visualization was the most challenging we have designed because there is more information to display about each problem element than in the other applications. Our design goal was to make the following information available to the user about each operation:

- start time
- duration
- mobility
- its job
- position in its itinerary
- its machine
- position in its roster
- if it is on the critical path
- if it is currently selected (required for various user actions)
- if its current roster-position is different than on the previously displayed solution [1]

As shown in Figure 2, our visualization uses Gant charts, a standard mechanism to display schedules. Each operation is represented by a rectangle with a length that is proportional to the duration of that operation. The x-location of the rectangle indicates the start time of the operation in the current schedule. The y-location of the rectangle depends on whether the user chooses to view itineraries or rosters. If the user views rosters, then each row of the Gant chart represents the activity of one machine. If the user views itineraries, each row represents all the operations of one job. While viewing rosters is generally more useful, viewing itineraries allows users to quickly observe when the idle time for each job occurs as well as which of each job's operations lie on the critical path.

The background color of each operation displays either its mobility or its job; the user can toggle between the two. These are two of the most important pieces of information for the user and hence it is imperative that the user be able to perceive them at a glance. While it would be preferable to display both at the same time, we found that if we displayed both simultaneously then neither was easily perceivable.

A toggle switch allows the job ID and itinerary position to be drawn on top of each operation when viewing rosters. For example, we draw "4.5" for the fifth operation on the fourth job.

The rest of the information is conveyed with thin, horizontal bands drawn in each operation. Each band is 1/5 the total height of the rectangle for the operation. A white band in the middle of an operation indicates it is on the critical path. This band stands out well against the solid background colors, allowing the user to quickly perceive this important information. A purple band along the bottom indicates the operation has changed position compared to the previous solution. A band along the top is used to indicate the job id if the mobilities are being shown. These bands allow users to detect the corresponding information if the user focuses on them, but do not distract the users unnecessarily. All bands can be toggled on and off by the user.

Users can select or unselect operations individually (by clicking on an operation), by job (by double clicking on any of the job's operations), or by rectangular region in the Gant chart. Selection is used to change mobilities or mark operations (marking is described below). In our visualization, a selected operation's background color is set to black, and the top band shows its job id.

3.3 Extensions to HuGS

We now present two generic extensions to HuGS we developed while creating the Jobshop application. We also describe how these extensions were used within the Jobshop application.

The first extension was motivated by the fact that when the user manually modifies the current jobshop solution, it often increases the makespan dramatically. Quite often much of the loss in solution-quality can be recovered by making small adjustments to other parts of the schedule. However, it is typically difficult for the user to visually identify where in the schedule these small adjustments need to be made.

To address this problem, we created a *power move* option. Enabling power moves changes the semantics of manual moves made by the user. After each manual move, the HuGS application initiates a greedy exhaustive search, with the maximum ply set to one, on the solution that results from applying the manual move to the current solution. The restriction on maximum ply forces the search to complete quickly.

To prevent the system from immediately un-doing the move that the user performed, the system creates a copy of the current mobilities and sets all elements altered by the user's move to low mobility. This set of mobilities is saved. If the user holds down the shift key while making the next manual move, then the system uses the saved mobilities, again setting the elements altered by the current move to low, to constrain the greedy exhaustive search. Thus, the

[1] Our experience has been that without any indication of what has changed, it is very difficult to know what has happened when, for example, the search algorithm returns a new solution.

Figure 2: The Jobshop Application: the screenshot above shows our visualization in the mode for viewing operations by roster, where the background color of each operation indicates which job the operation is on. A white band in the middle of an operation indicates it is on the critical path, and a purple band on the bottom of an operation indicates its roster-position has changed from the previously displayed solution. Job number 5 has been selected by the user, by double-clicking on one of its operations, and so its operations are painted black

user can make a sequence of power moves without having the system undo any of the user's modifications.

The code for power moves was made entirely within the generic HuGS toolkit, and thus all of our applications were able to take immediate advantage of this extension. We discuss the use of these moves in the section below.

The second extension to HuGS is a generic mechanism and corresponding visualizations that allow the user to tailor the objective function for each invocation of the search algorithm. Developers of HuGS applications can now provide the user with a set of parameters for controlling the search. We currently support boolean, integer, and double valued parameters. The user-controlled values are fed as input to the HuGS components that compute and compare the score (or cost) of the solutions. These components are, in turn, called by the search algorithms, and thus the users to control the objective function used to evaluate solutions by all the search algorithms in HuGS.

We increased the expressiveness of objective-function control by allowing users to *mark* selected elements. Each application's visualization component must indicate which elements are marked. In Jobshop, marked operations are drawn with a diagonal line from the upper left corner to the lower right corner.

In Jobshop we found that users often could identify troublesome operations that they wanted moved either earlier or later in the schedule, but were not sure how to accomplish this. To allow users to encourage the search algorithm to shift operations in a particular direction, we designed "earlier" and "later" control parameters. If the user marks some elements and sets the "earlier" control to true, the system will try to minimize the sum of the roster-position of the marked elements. The user also has an integer control-parameter called "budget" which indicates how much the search algorithm can "spend" to achieve this goal. If the budget is 0, then the search algorithm will use the position of the marked elements only to break ties between two schedules that would otherwise appear equally good. If the budget is 10, then the algorithm can reduce the makespan by up to 10 units in order to find a schedule in which the marked elements appear earlier in their rosters.

Another control-parameter in our Jobshop application is an integer parameter called "spansize". Recall that the objective function is to reduce the makespan, i.e., the time that the last operation finishes. With spansize k, the system compares solutions based on when the last k machines complete their operations, using the lexicographic ordering. This feature is especially useful when the second to last machine completes its operations just before the last machine finishes its operations. The disadvantage of setting the spansize high is that it slows optimization, since now there are multiple critical paths. Our initial experiments suggest that a spansize of 3 produces the best results for unguided search.

4 Initial Experience with HuGS Jobshop

The authors of this paper have used this system on a variety of problems, including 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. We now report on our informal observations from this initial exploration.

On the negative side, unlike Crossing and Delivery, we rarely found it productive to focus the search algorithms on subproblems using the mobilities. Our explanation is that the operations in a given schedule are highly interconnected and so local changes are rarely sufficient to improve the schedule.

Mobilities were, however, quite useful. A typical usage pattern was to change the roster-position of one or more operations, set them to low mobility, and invoke the search algorithm. The low-mobility settings force the search algorithm to adjust other parts of the schedule. After the user halts the search, the user can reset all operations to high mobility and invoke the search algorithm again. This combination of moves often produced a new best solution by forcing the system out of its local minimum. Without the low mobilities, the first search would often return immediately to the previous local minimum.

An additional use of mobilities was to swap the roster-position of two operations and set them both to medium mobility. This imposes the constraint that the relative order of the two operations cannot be changed, but otherwise allows all moves.

Both extensions to HuGS described above proved useful. Power moves produced the desired results and give the user a better sense of the implications of a manual move. The ability to encourage the system to move operations earlier or later seems especially effective. With this feature, the users can convey a general idea for how to optimize the schedule and let the search algorithms pursue it. Prior to this feature, pursuing similar ideas often required a long series of manual moves and searches. Finally, users often adjust the spansize during the optimization session based on how many machines complete their operations near the makespan in the current schedule.

A particularly subjective impression is that strategy plays a more significant role in the Jobshop application than in the other HuGS applications. The user can often think multiple moves ahead, noticing that if some operation is moved earlier or later it will provide flexibility for some other operation which, if moved, will provide flexibility for some other operation, and so on, eventually connecting to the critical path.

5 Related Work

Interactive optimization systems have been built for a variety of applications, including space-shuttle scheduling (Chien *et al.*, 1999), graph drawing (Nascimento & Eades, 2001), graph partitioning (Lesh, Marks, & Patrignani, 2000), vehicle routing (Waters, 1984; Bracklow *et al.*, 1992; Anderson *et al.*, 2000), and constraint-based drawing (Nelson, 1985; Heydon & Nelson, 1994; Gleicher & Witkin, 1994; Ryall, Marks, & Shieber, 1997).

Other research has explored alternative methods for dividing the work between human and computer in cooperative optimization or design. In the space-shuttle scheduling application (Chien *et al.*, 1999), for example, the computer detects and resolves conflicts introduced by the user's refinements to a schedule. In interactive constraint-based drawing applications e.g., (Nelson, 1985; Heydon & Nelson, 1994; Gleicher & Witkin, 1994; Ryall, Marks, & Shieber, 1997), the user imposes geometric or topological constraints on a nascent drawing such that subsequent user manipulation is constrained to useful areas. The interactive-evolution paradigm, which has primarily been applied to design problems, offers a different type of cooperation (Sims, 1991; Todd & Latham, 1992). In this approach, 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. Thus novel designs evolve, subject to user-supplied selection criteria. This paradigm has found use in various computer-graphics applications. The HuGS paradigm differs significantly from the iterative-repair, constraint-based, and interactive-evolution paradigms in affording the user great involvement and greater control of the optimization/design process.

A preliminary version of the Jobshop application has been described briefly in our previous work (Klau *et al.*, 2002a,b). Of course, there has been extensive research on automatic optimization for the jobshop problem (e.g., Applegate & Cook. (1991); Aarts *et al.* (1994)).

6 Conclusion

We have described a new interactive jobshop scheduling application developed using the HuGS toolkit. We inherit the powerful, albeit generic, local-search algorithms from the HuGS toolkit and have not focused on improving the efficiency of these algorithms for this specific algorithm. Instead, we focused on improving the interaction between the application and the user. The visual user-interface proved a significant challenge, as the jobshop problem requires displaying a great deal of information of various types to the end user. We also developed general mechanisms for improving interaction that have become part of the toolkit: power moves and user-controlled objective functions combined with marking.

Our preliminary experience suggests that users can guide the search algorithms in our application to improve performance over purely automatic algorithms. More importantly, the interactivity of our application allows users to steer the search algorithm toward solutions that reflect their real-world knowledge of constraints and preferences, and improves user trust and understanding of solutions by allowing them to participate in their construction.

In the future, we plan to experiment with other visualizations which will provide more information about which operations are "near" the critical path, as well as those that are on it. We also plan to perform thorough user studies, along the lines of those reported in (Klau *et al.*, 2002b) for the Crossing and Delivery applications.

References

Aarts, E.; Laarhoven, P. v.; Lenstra, J.; and Ulder, N. 1994. A computational study of local search algorithms for job-shop schedul-

ing. *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.

Applegate, D., and Cook., W. 1991. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing* 3(2):149–156.

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.

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

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.

Heydon, A., and Nelson, G. 1994. The Juno-2 constraint-based drawing editor. *Digital Systems Research Center Research Report 131a*.

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

Klau, G. W.; Lesh, N.; Marks, J.; and Mitzenmacher, M. 2002b. Human-guided Tabu search. *to appear in AAAI'02*.

Lesh, N.; Marks, J.; and Patrignani, M. 2000. Interactive partitioning. *Graph Drawing* 31–36.

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.