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

Human-Guided Search: Survey and Recent Results

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

TR2003-07 July 2003

Abstract

We present a survey of techniques and results from the Human-Guided Search (HuGS) project, an ongoing effort to investigate interactive optimization. HuGS provides simple and general visual metaphors that allow users to guide the exploration of the search space. These metaphors apply to a wide variety of problems and combinatorial optimization algorithms, which we demonstrate by describing the HuGS toolkit and as seven diverse applications we developed using it. User experiments show that human guidance can improve the performance of powerful heuristic search algorithms. HuGS is also a valuable development environment for understanding and improving optimization algorithms. For two different problems, we have used HuGS to develop automatic algorithms that produce new best results on benchmark problem instances.

In submission

Submitted January 2002

Human-Guided Search: Survey and Recent Results

G.W. Klau[1], N. Lesh[2], J. Marks[2], M. Mitzenmacher[3]
[1] Konrad-Zuse-Zentrum für Informationstechnik
[2] Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA, 02139
lesh@merl.com, marks@merl.com
[3] Harvard University, Computer Science Department
michaelm@eecs.harvard.edu*

Abstract

We present a survey of techniques and results from the Human-Guided Search (HuGS) project, an ongoing effort to investigate interactive optimization. HuGS provides simple and general visual metaphors that allow users to guide the exploration of the search space. These metaphors apply to a wide variety of problems and combinatorial optimization algorithms, which we demonstrate by describing the HuGS toolkit and as seven diverse applications we developed using it. User experiments show that human guidance can improve the performance of powerful heuristic search algorithms. HuGS is also a valuable development environment for understanding and improving optimization algorithms. For two different problems, we have used HuGS to develop automatic algorithms that produce new best results on benchmark problem instances.

1 Introduction

Most previous research on optimization focuses on producing algorithms that are more efficient than previous algorithms on some class of problems. An algorithm's efficiency is judged by the value of the solutions it produces according to a given, well-defined objective function (such as the total distance traveled in vehicle-routing problems) as well as amount of computation required to produce those solutions.

This approach to evaluating optimization systems is, however, insufficient for many real-world contexts in which optimization problems arise. First, people often need to understand and trust a solution in order to be able to implement it effectively. For example, one person might need to explain or justify it to others in order to gain their cooperation. Additionally, people might have to modify a solution as unexpected events occur (e.g., a truck breaks down). Second, optimization systems are typically solving only an approximation of the problem that is really of interest to its users. People often know much more about a problem than they can specify in advance and, furthermore, cannot specify what they know in the given selection-criteria language.

One approach to addressing these often-neglected aspects of optimization is to develop systems in which people participate in constructing solutions. Interactive, or human-in-the-loop, optimization systems have been developed for a variety of applications, including space-shuttle scheduling [5], graph drawing [26], graph partitioning [20], vehicle routing [35, 2], and constraint-based drawing

*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.

[27, 12, 29]. People can better trust, justify, and modify solutions that they help construct than automatically generated solutions. 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.

In this paper, we provide an overview and present new results from our ongoing Human-Guided Search (HuGS) project.[1] In the HuGS framework, users can manually modify solutions, backtrack to previous solutions, and invoke, monitor, and halt a variety of search algorithms. More significantly, users can constrain and focus search algorithms by assigning *mobilities*, which we describe below, to elements of the current problem. During the course of this project, we have created interactive optimization systems 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 [2, 20, 30, 17]. Additionally, we have developed the HuGS Toolkit, Java software which supports the quick development of interactive optimization systems [18].

Our experiments show that human interaction can significantly improve the performance of search algorithms even when measured by only the given objective function. In particular, our experiments have shown that human guidance can improve the performance of an exhaustive search algorithm for the capacitated-vehicle-routing-with-time-windows problem to the point where the interactive algorithm is competitive with the best previously reported algorithms [2, 30]. Furthermore, our interactive system was able to achieve the best performance we know of on benchmark problems for the 2D Rectangular Strip Packing problem [21]. Additionally, an automatic system we developed using HuGS was able to find new best solutions for the three largest benchmarks in the literature for the two-dimensional hydrophobic-hydrophilic protein-folding problem [22].

Below, we describe related work and then present the current applications, techniques, toolkit, experimental results, and recent results from the HuGS project.

2 Related Work

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 [31, 34].

Other systems provide more interactivity by allowing the users to control search parameters or add constraints as the search evolves. Colgan *et al.* [7] 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 [12, 29, 27]. 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 [35]. An interactive space-shuttle operations-

[1] This paper contains material presented in previous conference publications [2, 20, 30, 17, 22, 21].

scheduling system allows users to invoke a repair algorithm on their manually modified schedules to resolve any conflicts introduced by the user [5].

The mixed-initiative approach to human-in-the-loop systems uses agents to mediate the cooperation between the computation system and the user to help the user solve an optimization problem (e.g., [33, 11]). The emphasis in this work is on not only on combining the skills of people and computers to solve problems, but in particular on having the computer play an active role in the collaboration itself. Thus, the work has focused 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 dialog about the problem at hand and approaches to solving it.

In contrast to these other approaches, HuGS allows the user to focus the search algorithms more directly through a combination of simple metaphors and visualizations and has been applied to a much wider range of problems.

3 Applications

In this section, we briefly describe seven applications we have developed using the HuGS toolkit. The first four are described in more detail in [17], the fifth in [21], and sixth and seventh are in development.

The *Crossing* application is a graph layout problem [9]. 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 [10]. 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 [8]. 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.

The *Jobshop* application is a widely-studied task scheduling problem [1]. 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.

The *Packing* application is a two-dimensional bin packing problem [14, 15]. A problem consists of n rectangles with their dimensions and a target width W. The rectangles must be placed parallel to the horizontal and vertical axes. The goal is to pack the rectangles without overlap into a single rectangle of width W and minimum height H. A screenshot of the Packing application is shown in Figure 3.

The *Heating* application is a variation of the prize-collecting Steiner tree problem (see, e.g., [16]). A problem consists of a set of potential customers, a value for each customer, a set of potential supply

stations, a cost for laying pipe, and a map of local streets along which heating pipes can be laid. The goal is to find the most profitable network of pipes.[2]

The *Labeling* application considers a classical problem in cartography (see, e.g., [6]). A problem consists of a set of sites (e.g., cities on a map) and a rectangular label for each site. Each label can be placed in any of four locations around each site. The goal is to place the maximum number of labels such that no overlaps occur.[3]

3.1 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 fully automatic optimization, deciding which moves to include is an important design choice for the developer of an optimization system. Our framework further requires the developer to determine which elements that are altered by each move.

4 Techniques

4.1 Mobilities

Our system maintains and displays a single current solution, such as the ones shown in Figure 1. 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.

[2]This problem is useful because district heating companies are faced with the problem of balancing the profit that can be obtained by providing hot water connections to potential customers and the cost to build the network of pipes. The Heating application is joint work with Andreas Moser, Vienna University of Technology.

[3]The Labeling application is joint work with Markus Chimani and Bin Hu, Vienna University of Technology.

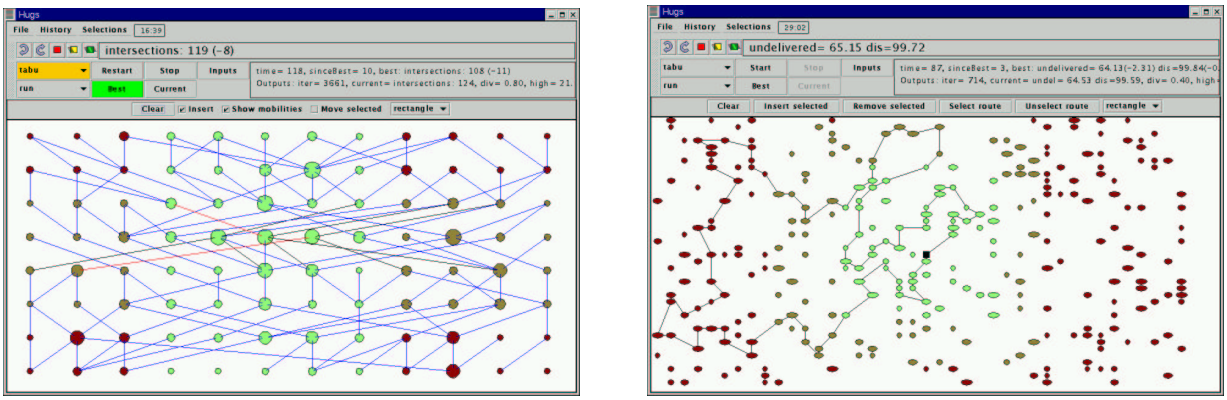

Figure 1: The Crossing and Delivery Applications.

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 useful in all of the applications described above.

4.2 Guidable Algorithms

As we describe below, our interface allows the user to choose and change which search algorithms to employ and we have found it beneficial to provide a small suite of algorithms. We now describe several algorithms that can be controlled by mobilities. First, we describe two variations of a brute-force, exhaustive search. Then we describe a guidable version of a powerful heuristic, called tabu search. Both of these algorithms are general and work on all of our applications for which a move set is defined. We then describe a domain-specific, human-guidable search algorithm we designed for the Packing application.

```
GTabu (sol, mobilities, memSize, minDiv):
    best ← sol
    originalMobilities ← mobilities
    until halted by user
        m ← best move in legalMoves(sol,mobilities)
        sol ← result of m applied to sol
        if isBetter(sol, best) then
            best ← sol
            mobilities ← originalMobilities
        else
            mobilities ← memory(m,mobilities,memSize)
            mobilities ← diversify(m,mobilities, minDiv)
    return best

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 2: Pseudo code for guidable tabu search.

4.2.1 Exhaustive search

We use two variations of exhaustive search: steepest-descent and greedy. Both algorithms first evaluate all allowed moves given a set of mobilities, then all combinations of two allowed moves, and then all combinations of three moves and so forth. 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.

4.2.2 Tabu search

While we found that human guidance of a simple search algorithm to be surprisingly effective [2], we were able to improve upon these results by providing the user with a guidable version of tabu search. Tabu search is a heuristic approach for exploring a large solution space [13]. 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 to 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 [13].

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 allowed 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 Figure 2.

The algorithm updates the mobilities in two ways. First, the call to the MEMORY function prevents GTabu 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 user-specified mobilities 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 2, 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 understanding and using this control parameter. Our experience from the training sessions described in Section 5.2 is that users 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 applications provide a color-coded visualization of the users' current mobility settings. This same mechanism can be used to display GTabu's mobilities.

4.2.3 Heuristic search for packing

We found that our local search algorithms were not effective for the packing problem. This is not surprising in that past efforts to apply standard local search techniques, such as simulated annealing or genetic algorithms have not been able to match the performance of simple heuristics (see [14] for an overview). We now describe an improved and guidable version of one of the most successful packing heuristics. Here, we assume that the orientations of the rectangles are fixed; in Section 6, we consider a variation in which the rectangles can be rotated 90 degrees.

A common method for packing rectangles is to take an ordered list of rectangles and greedily place them one by one. Perhaps the best studied and most effective heuristic in this setting is the Bottom-Left (BL) heuristic, where rectangles are sequentially placed first as close to the bottom and then as far to the left as they can fit. For some problems, BL cannot find the optimal packings [3], nor does it perform well in practice when applied to random orderings. However, a very successful

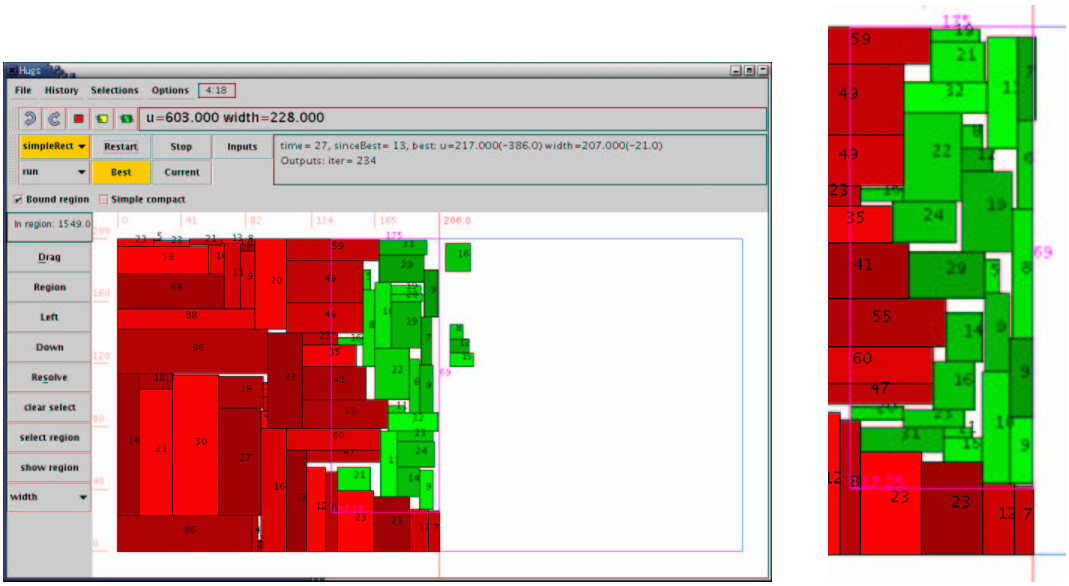

Figure 3: Interactive system: The image on the left is a screen shot of our system in use. The user has selected a region to apply BLD* to and has frozen most of the rectangles (frozen rectangles shown in red/dark gray, unfrozen in green/light gray). The image on the right shows a blowup of the selected portion on the packing, after BLD* has run for a few seconds and the user has pressed the Best button to see the best solution found. Given the aspect ratio of a computer monitor, we found it more natural to rotate the problem by 90 degrees, so that there is a fixed height and the goal is to minimize the width of the enclosing rectangle.

approach is to apply BL to the rectangles ordered by decreasing height, width, perimeter, and area and return the best of the four packings that result [14, 15]. We refer to this scheme as Bottom-Left-Decreasing (BLD).

We developed a variation of the BLD heuristic called BLD* that considers successive random perturbations of the original four decreasing orderings. Our intuition for why BLD* performs so much better than random BL is that the decreasing sorted orders save smaller rectangles for the end. BLD* chooses random permutations that are "near" the decreasing sorted orders used by BLD as they will also have this property. There are many possible ways of doing this; indeed, there is a deep theory of distance metrics for rank orderings [24]. BLD* uses the following simple approach: start with a fixed order (say decreasing height), and generate random permutations from this order as follows. Items are selected in order one at a time. For each selection, BLD* goes down the list of previously unaccepted items in order, accepting each item with probability p, until either an item is accepted or the last item is reached (in which case it is accepted). After an item is accepted, the next item is selected, starting again from the beginning of the list. This approach generates permutations that are near decreasing sorted order, preserving the intuition behind the heuristic, while allowing a large number of variations to be tried. BLD* first tries the four orders used by BLD and then permutes each of these orders in round-robin fashion.

While BLD* does not fit exactly into the framework for mobilities described above, we allowed people to guide the algorithm in a simple fashion by assigning rectangles high or low mobilities: rectangles with high mobility can be moved and those with low mobility are frozen in their current location. The BL heuristic is then used to place each high-mobility rectangle as close to the bottom

and then as far to the left as it can fit without overlapping any of the low mobility rectangles or any of the high-mobility rectangles that have already been placed. Figure 3 shows a screenshot of the interface and an example of how mobilities are used with BLD*.

4.3 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 choose which search algorithm to invoke from among the selection described above (it is also simple to add a new search algorithm to the system.)

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.

While the search algorithm is running, the users can select from a variety of search-visualization modes. The most efficient mode is to let the search algorithm run in the background without updating the current visualized solution. The users can also observe the search more directly. The users can put the search into "auto" mode, in which every solution the search considers is displayed, or "poll" mode in which the computer is polled periodically for its current solution, or "step" mode in which the computer waits for the user to press a button before moving on to the next solution it considers. These modes are useful for developing applications as well as for learning about how the system and search algorithms work.

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.

5 Results

5.1 HuGS Toolkit

We now describe our Java middleware for rapidly developing interactive optimization systems. This software was used to create the above applications and is available for research or educational purposes.[4] More details are described in [18].

Generic code in the toolkit is used to maintain the current working solution, the mobilities, and the history. The file Input/Output, including saving and loading of problems and solutions, and logging user behavior, are also performed by generic code. Furthermore, all our applications use the same implementation of the exhaustive and tabu search algorithms and the GUI's for invoking and monitoring them. Even the packing algorithm is controlled by the same generic GUI as the other search algorithms. Our implementation of the tabu search algorithm functions by modifying the mobility assignments. Thus, there is no additional burden on the developer of a new application in order to be able to use tabu search.

Of course, the developer of a new application must define what a problem is and what a solution is for that application.[5] Each problem instance needs to implement a function that returns all the elements of that problem. Each solution instance must be able to return an object which represents the *score* of that solution. An instance of a score object must be able to compare itself to another instance of a score and decide if it is better, worse, or equal to that instance. Additionally, a developer must define a set of moves which can be applied to solutions in this application. For some applications, there might be several different types of moves. Additionally, the developer must provide a function for generating all possible moves for a given element.

Each application requires a domain-specific visualization component. From the point of view of the system, the visualization component has only three responsibilities. First, it must report any manual moves made by the user. These moves will be applied to the current solution that the system maintains. Second, the visualization component must have an update function which, when called, triggers it to display the current solution and mobilities maintained by the system. Third, the system must allow users to select and unselect problem elements. The system will query the visualization component for the list of currently selected elements in order to maintain the mobilities. The users can, for example, set all the selected elements to a particular mobility, as well as reset all elements to any particular mobility.

5.2 Guided vs. Unguided Search

In [2] we describe an interactive optimization system for solving the capacitated-vehicle-routing-with-time-windows problem [32]. This system follows the HuGS framework, although it was developed prior to the HuGS Toolkit. It only provides the user with the exhaustive-search algorithm, not GTabu. Even so, we were able to achieve results on the well-known Solomon benchmarks [32] that were competitive with state-of-the-art algorithms designed specially for this problem (and these benchmarks). We also demonstrated clearly that human-guidance was essential to obtain these results. As one point of comparison: the best known solutions for the subset of benchmarks we tested require 11.5 vehicles, on average, to satisfy all the customers requests. The average result of 90 minutes of pre-computation followed by 90 minutes of interactive guidance of the exhaustive algorithms produced solutions requiring 11.88 vehicles, on average. This matched the best results by

[4] Contact lesh@merl.com for details.

[5] This involves defining classes which implement Java interfaces for a Problem class and a Solution class.

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

any one algorithm up to 1999, the year before our publication. However, running our algorithm by itself for 20 hours produces solutions that require 12.06 vehicles, on average, which is not competitive with state-of-the-art algorithms.

In [17] we describe experiments comparing guided search of the tabu and greedy-exhaustive algorithms to unguided search for the Delivery and Crossing applications (as well as some other experiments not discussed here). By *unguided search*, we mean running either the tabu or exhaustive algorithm without intervention and with all elements set to high mobility. We trained test subjects for 2-4 hours on how to use our system. Each of our four subjects 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. 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 1 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 2 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.

5.3 Recent Successes

We now describe the results we obtained from the Packing and Protein applications. In both cases, the automatic methods we developed outperform previous automatic methods. (For Packing, human-interaction further improves the results significantly.) Development of the automatic methods in HuGS benefits from what we call *researcher-in-the-loop*. As algorithm developers, we generate many ideas from solving problem instances ourselves using the HuGS system. Also, we often get ideas for how to improve our algorithm by watching it in action. Our experience has been that it is well worth the investment to build visualizations and to work within the HuGS framework, even if the

	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 2: 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.

only goal is to produce automatic methods.

For example, we developed an exhaustive search using branch-and-bound techniques for the Packing application. The key to this approach was a set of bounding techniques that reason about gaps created by the currently placed rectangles. We developed these techniques by watching the branch-and-bound algorithm in action and realizing that it was missing opportunities for bounding. The resulting algorithm is extremely effective for problems with fewer than 30 rectangles and in which the rectangles can be tightly packed with little or no unused space. For example, it solves benchmark problems containing 25 rectangles in an average of 96 seconds. To our knowledge, the best reported results were more than 5% above optimal [14].

For instances too large to be solved exhaustively, we used the BLD* algorithm described above. We ran experiments on the benchmark instances described in [14]: here we review our results on the N4, N5, N6 collections, each of which contains 5 instances. First, we established that that BLD*, our variation of the BLD heuristic, outperforms the BLD heuristic itself. For example, after just one minute, BLD* reduces the packing height from an average of 7.3% over optimal by BLD to about 5.3% over optimal on the N4-N6 data instances.

Our experiments show that human interaction improves BLD*.[6] People can identify particularly well-packed subregions of a given packing and then focus a search algorithm on improving the other parts. People can also devise multi-step repairs to a packing problem to reduce unused space, often producing packings that could not be found by the BL heuristic with any ordering of rectangles. To prepare for our user experiments, we ran BLD* for 2 hours on on each instance in N4-N6. We then performed one trial for each instance in which a user attempted to find a solution 1% closer to optimal than the best solution found by BLD* within 2 hours. As shown in Table 3, the users were able to reach these targets in about 15 minutes on average. In every case, the target was reached within 30 minutes. While this is not exactly a "head-to-head" comparison, since the users had the target scores to reach, the fact that people were able to achieve superior solutions so quickly demonstrates the value of interaction.

We also tested our interactive system on the few other benchmarks we could find in the literature, including in particular ones without known optimal solutions, referred to by Hopper as D1 and D3. [14]. The best solutions for D1 and D3 in the literature appear to have height 47 and 114. We were

[6]This is unsurprising in that people are known to outperform computers at packing irregular polygons in industrial applications [25].

dataset	number of rectangles	percent over optimal by BLD* in two hours	time for users to find packing 1% closer to optimal
N4	49	4.3%	3.3% in 14 min., 21 sec.
N5	73	4.1%	3.1% in 13 min., 52 sec.
N6	97	3.3%	2.3% in 17 min., 12 sec.

Table 3: Interaction experiment results for the Packing application: The second column shows the average percentage over optimal achieved by BLD* in two hours. These results are at least 2%-3% closer to optimal than the best previously published results. The third column shows the average time it took interactive use of BLD* to achieve a solution another 1% closer to optimal. The values are averaged over the five problem instances in the corresponding collection.

able to find a solution with height 46 in about 15 minutes. We were able to match the 114 for D3 in about 20 minutes.

The Protein application, we used the HuGS toolkit to develop a new set of local moves, which we call *pull moves* [22]. Pull moves are quite different than previous local moves applied to this problem. We developed them in the process of improving our user interface to help users manually move several elements at once. In Protein, the input sequence of amino acids must be placed on a grid in a non-overlapping path, such as those shown in Figure 4. It was quite tedious for the user to modify the path manually, and so we designed a "smart" manual move in which the user would re-position one amino acid and then the system would try to move as few other amino acids as possible to establish a valid path. Essentially, one amino acid is moved and others are pulled after it.

While users have so far found it difficult to guide the algorithms in Protein, perhaps because of the unintuitive geometric reasoning required, GTabu has proven extremely effective using pull moves. The highlight of our experimental results is that we have found new best solutions (i.e., lowest energy configurations) for the three longest benchmarks we found in the literature and matched the best results for the others. For example, a sequence of length 85 used in [19, 23] was conjectured to have a ground state energy, or minimal energy, of -52; we have found a configuration with energy -53. We have similarly found new best configurations for two sequences of length 100 used in [4, 28]. We found multiple configurations with each new best score: one sample for each is shown in Figure 4.

The length-85 example is particularly compelling. In [19], it is stated that the optimal ground state has energy -52; it appears that the authors constructed this sequence themselves with an optimal solution in mind to test their algorithm. The genetic algorithms of [19] found a ground state of -47. In [23], an evolutionary Monte Carlo algorithm found a ground state of -52, but only by specifying constraints that significantly cut down the search space. That is, the algorithm is modified to constrain specified subsequences of hydrophobic residues (covering approximately 40% of the sequence) to take one of three forms. Our algorithm finds several unstructured configurations with energy -53.

6 Latest Results

We now briefly describe recent results in the variation of the Packing in which the rectangles can be rotated by 90 degrees.

First we modified the BLD* heuristic as follows. We use a single base order, in which the

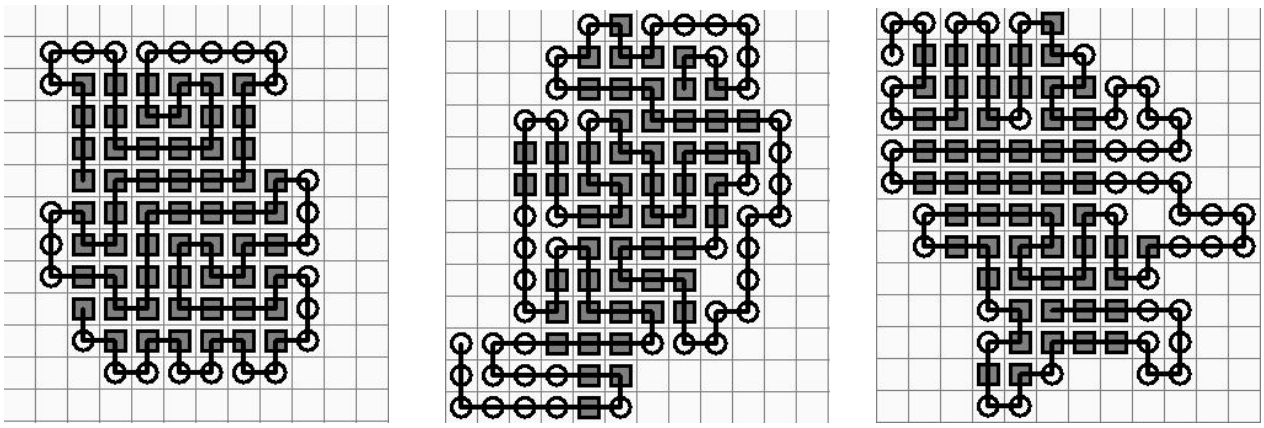

Figure 4: New best solutions for simplified protein-folding benchmarks. From left to right there is a -53 configuration for the S85 sequence, a -48 configuration for S100a and -50 for S100b. The squares represent hydrophobic amino acids, and the circles represent hydrophilic ones. Thus each image contains the problem definition as well as its solution. The score is the number of vertically or horizontally adjacent nonsequential hydrophobic pairs.

rectangles are sorted in decreasing order by their *smaller* dimension. We then permute that order as described above in Section 4.2.3. Given an ordering, we place each rectangle by calling the BL heuristic on both possible orientations of the rectangle. We chose the orientation that puts the top right corner of the rectangle closer to the bottom, or closer to the left to break ties. We prefer the tall orientation to the wide orientation if there is still a tie. This variation outperforms the other variations we tested by a substantial degree. The variations we tested included ordering the rectangles by their larger dimension, and choosing the placement based on the bottom left corner or the center of the rectangle.

Allowing BLD* to change the orientation improves the packings considerably. For the N4 and N5 benchmark sets, two hours of computation yields solutions 2.75% over optimal, compared with 4.2% over optimal using the given orientations.

We added a feature to our interactive Packing application which allows a user to manually reorient a rectangle. Then we again ran tests in which the users attempted to find solution 1% closer to optimal than the best solution found by BLD* within 2 hours. We thought this task might be too difficult since the targets were so much closer to optimal. However, for the 10 benchmarks in N4 and N5 it only took an average of 23 minutes and nine seconds to achieve these scores.

7 Conclusions and Future Directions

A major aim of our research is to better understand general principles of interactive optimization and to generate general ideas, techniques, and software. While interactive optimization systems have been built for a variety of applications, no other interactive optimization approach has previously been applied to such a varied set of optimization problems as we have described here.

All of the applications described in this paper, however, consider problems with, at most, several hundred elements. A future research direction is to explore techniques for applying interactive

optimization to larger problems, in which people cannot view all the elements at once on the computer screen.

There is also a need for performing deeper analysis and user experimentation on interactive systems. In [30], we present initial investigation into understanding of how well people can perform the various tasks in HuGS, but would like to build on this work. Additionally, we would like to investigate whether the impact of human-guidance increases or decreases as the speed of the algorithm increases. We would also like to carry out experiments to investigate how people's understanding and trust of solutions is effected by participating in the optimization process.

Finally, we see a need for a broader set of metrics for evaluating optimization systems. If two algorithms are equally efficient, but one affords more or better interaction then it is superior for many tasks. Indeed, in many contexts, interaction is more important than efficiency because the optimization algorithm is working with an impoverished objective function and the ability to successfully implement the solution depends on how well people understand and trust it.

References

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

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

[3] B. S. Baker, E. G. Coffman, Jr., and R. L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9:846-855, 1980.

[4] U. Bastolla, H. Frauenkron, E. Gerstner, P. Grassberger, and W. Nadler. Testing a new Monte Carlo algorithm for protein folding. *Proteins: Structure, Function, and Genetics*, 32:52-66, 1998.

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

[6] J. Christensen and J. Marks and S. Shieber. An Empirical Study of Algorithms for Point-Feature Label Placement. *ACM Trans. Graph.*, 14(3):203-232, 1995.

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

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

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

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

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

[12] Michael Gleicher and Andrew Witkin. Drawing with constraints. *Visual Computer*, 11:39–51, 1994.

[13] F. Glover and M. Laguna. 1997. *Tabu Search.* Kluwer Academic Publishers.

[14] E. Hopper. Two-Dimensional Packing Utilising Evolutionary Algorithms and other Meta-Heuristic Methods, PhD Thesis, Cardiff University, UK. 2000.

[15] E. Hopper and B. C. H. Turton. An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem. *European Journal of Operational Research*, 128(1):34-57,2000.

[16] D. S. Johnson and M. Minkoff and S. Phillips. The Prize Collecting Steiner Tree Problem: Theory and Practice In *Proceedings of Proceedings of 11th ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[17] G. Klau, N. Lesh, J. Marks, and M. Mitzenmacher. Human-Guided Tabu Search. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pp. 41-47, 2002.

[18] G. Klau, N. Lesh, J. Marks, M. Mitzenmacher, and G.T. Schafer. The HuGS platform: A toolkit for interactive optimization. In *Proceedings of Advanced Visual Interfaces*, pp. 324-330, 2002.

[19] R. König and T. Dandekar. Improving genetic algorithms for protein folding simulations by systematic crossover. *BioSystems*, 50:17-25, 1999.

[20] N. Lesh, J. Marks, and M. Patrignani. Interactive partitioning. *Graph Drawing*, pages 31–36, 2000.

[21] N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher. New Exhaustive, Heuristic, and Interactive Approaches to 2D Rectangular Strip Packing In submission, 2003.

[22] N. Lesh, M. Mitzenmacher, and S. Whitesides A Complete and Effective Move Set for Simplified Protein Folding To appear in *Proceedings of the 7th Intl. Conf. on Research in Computational Molecular Biology*, 2003.

[23] F. Liang and W. H. Wong. Evolutionary Monte Carlo for protein folding simulations. *Journal of Chemical Physics*, 115(7):3374-3380, 2001.

[24] J. I. Marden. *Analyzing and Modeling Rank Data* Chapman & Hall, New York, New York, 1995.

[25] V. J. Milenkovic and K. M. Daniels. Translational Polygon Containment and Minimal Enclosure using Mathematical Programming. *International Transactions in Operational Research*, 6:525-554, 1999.

[26] H.A.D. do Nascimento and P. Eades. User hints for directed graph drawing. In *Proc. of the Symposium on Graph Drawing*, 2001.

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

[28] R. Ramakrishnan, B. Ramachandran, and J. F. Pekney. A dynamic Monte Carlo algorithm for exploration of dense conformational space in heteropolymers. *Journal of Chemical Physics*, 106:2418, 1997.

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

[30] S. Scott, N. Lesh, and G. W. Klau. Investigating human-computer optimization. *Proc. of CHI 2002*, 2002.

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

[32] Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* 35(2):254–265.

[33] S.F. Smith, O. Lassila, and M. Becker. Configurable, mixed-initiative systems for planning and scheduling. In A. Tate, editor, *Advanced Planning Technology*. AAAI Press, Menlo Park, CA, May 1996. ISBN 0-929280-98-9.

[34] Stephen Todd and William Latham. *Evolutionary Art and Computers*. Academic Press, 1992.

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