A CASE STUDY IN LARGE-SCALE INTERACTIVE OPTIMIZATION

Markus Chimani[*]
Institute of Computer Graphics and Algorithms
Vienna University of Technology
email: mch@ads.tuwien.ac.at

Neal Lesh
Mitsubishi Electric Research Laboratories
201 Broadway, Cambridge, MA, 02139
email: lesh@merl.com

Michael Mitzenmacher[*†]
Computer Science Department
Harvard University
email: michaelm@eecs.harvard.edu

Candy Sidner
Mitsubishi Electric Research Laboratories
201 Broadway, Cambridge, MA, 02139
email: sidner@merl.com

Hidetoshi Tanaka
Information Tech. R&D Center
Mitsubishi Electric, Japan
email: htanaka@isl.melco.co.jp

ABSTRACT
We describe lessons learned in developing a program for interactive optimization of large airlift scheduling problems. While for small problems one can create a visualization that both shows a complete solution and is editable at the same time, with large problems, such visualizations provide too high a level of aggregation and cannot display the detail necessary for interaction. We explain how this changes the interactive process, and the implications for our design, such as the need for automatic focusing on parts on the problem to ease optimization. An additional problem requirement was that the user be enabled to change the problem specification (such as delivery deadlines). As a further contribution, we provide a specialized repair algorithm that aims at generating a valid solution after such changes, while introducing as few changes as necessary.

KEY WORDS
Planning and Scheduling, Human Computer Interfaces, Human-Guided Tabu-Search, Information Visualization, Minimize Disruption

1 Introduction

While there has been a considerable amount of work on automatic optimization systems, very little work has addressed how people can best use or interact with them. As with many technologies, the quality of the interface for optimization systems is often the bottleneck for the real-world use of the system. For example, the work we report on here originated because a potential customer is currently solving large airlift scheduling problems manually. The problem involves using a given set of airplanes to satisfy a given set of delivery requests using a given set of airports, covering the flight operations of a complete year's cycle. A previous fully automatic system was rejected because the customer wanted more control.

The experts who craft the solutions manually have knowledge which is not fully captured in the problem specification. Thus, the schedules produced by automatic algorithms may not be optimal with respect to all the constraints and preferences known to the experts. This knowledge may be difficult for the experts to articulate even in natural language, let alone reduce to a form that can be input to an automatic optimization algorithm.

Research in interactive, or human-in-the-loop, optimization systems directly addresses the question of how people can effectively interact with optimization systems. The aim is to utilize the users' knowledge to steer the search algorithm towards good solutions, as well as to make her understand and trust the solution.

In this paper, we report on lessons learned by applying the Human-Guided Search (HuGS) framework and toolkit [1, 7, 6] to the above airlift schedule problem. Most previous work on interactive optimization, including our own, has been on small, academic problems. The goals of this paper are to demonstrate that many advantages of HuGS for these problems transfer well to large, complex applications as well as to present some novel techniques and concepts we needed to introduce for this application.

We can briefly summarize our findings as follows. As with previous HuGS applications:

P1 Users decide how to invest computational resources, e.g., they can *focus* an optimization algorithm on improving a chosen subset of the current schedule.

P2 Users can employ powerful optimization algorithms while manually editing and refining the current schedule to respect their real-world knowledge.

P3 Users can temporarily introduce infeasibilites into the current schedule while finding better schedules.

However, we found that large problems require a different pattern of interaction than small ones. For both small and large problems, the user's activity can be classified into three categories: inspection, modification, and user-controlled reoptimization of the current solution. (We describe these phases in more detail below.) For small problems, the user can constantly and quickly shift between these phases. It is easy for the user to visualize and control

[*]This work was done while visiting Mitsubishi Electric Research Laboratories.

[†]Supported in part by NSF CAREER Grant CCR-9983832 and an Alfred P. Sloan Research Fellowship.

what portion of the solution the optimization is currently focused on. If the optimization algorithm changes the solution in an undesirable way, the user can easily detect this problem, backtrack to a previous solution, change the focus to prevent the undesirable change, and re-invoke the optimization. Furthermore, in academic problems the problem specification remains constant during the entire session. In contrast, for our current application we needed to develop:

N1 Heuristics to help control the focus of the optimization algorithm as the user navigates to different views of the schedule because it is difficult to control the focus when the current complete schedule can not be viewed in detail on one screen.

N2 Methods for allowing the user to change the problem specification during the session, e.g., to add delivery requests or change deadlines. These changes often make the current schedule infeasible.

N3 A special mode to support refining near-acceptable schedules. In this mode, after the user searches through the schedule for errors (with respect to her real-world knowledge), she wishes to minimize the number of changes made by the optimization algorithm in the process of fixing these errors. Minimizing changes frees the user from re-inspecting much of the schedule.

2 Background

Interactive systems that leverage the strengths of both humans and computers must distribute the work involved in the optimization task among these two participants. Existing systems have implemented this division of labor in various ways. In some interactive systems, the users can only indirectly affect the solutions to the current problem.

For example, in interactive evolution 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 [12, 13]. Other systems provide more interactivity by allowing users to control search parameters (e.g., in circuit-design optimization [3]) or adding constraints as the search evolves (e.g. in drawing applications [4, 9, 10]).

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 allowed users to request suggestions for improvements after making schedule refinements to the initial solution [14]. An interactive space-shuttle operations-scheduling system allowed users to invoke a repair algorithm on their manually modified schedules to resolve any conflicts introduced by the user [2].

Our human-guided search (HuGS) framework [1, 6] also allows users to manually modify solutions, but in addition it allows them to explicitly steer the optimization process itself. Users can invoke, monitor, and halt optimizations as well as specify the scope of these optimizations. They can also backtrack to previous solutions. HuGS includes a generic mechanism for allowing users to focus the search algorithm. Users can assign one of three *mobilities* (high, medium, or low) to each problem element (e.g., a delivery request). Informally speaking, the optimization algorithms are only allowed to apply transformations to the current solution that alter at least one high-mobility element and do not alter any low-mobility elements. (For example, the algorithm could only shift high-mobility delivery requests from one day or airplane to another.) Thus, the low-mobility elements are frozen and can be used to divide the optimization problem into smaller su b-problems. The medium-level elements in the current solution can be adjusted, but only in service of adjusting the high-mobility elements. (This is relevant if, for example, a valid transformation would be to swap two delivery requests on different airplanes, in which case one must have high mobility but one could have medium mobility.)

HuGS was utilized in an interactive vehicle-routing system; initial experiments with this system showed that human-guided optimization outperformed almost all reported vehicle-routing algorithms. A more focused study examined people's ability to guide search in the various ways allowed by HuGS [11]. Following the HuGS framework, do Nascimento and Eades developed an interactive layered graph-drawing system that provided most of the functionality of HuGS and also allowed users to add constraints to the problem at runtime [8]. Preliminary experiments have shown that people can improve automatically generated solutions using this system.

3 Problem Description

This section defines our airlift scheduling problem and the relevant operations we have to support.

There are a set of airplanes, a set of airports, and a set of requests. Each airplane is of a specified *airplane type* and is associated to its *home airport* that it starts from and returns to each day. Each airplane has a varying *availability window* for each day of the year; planes can fly only during these periods. Each airport also has its own availability window and specifications of how long landing, takeoff and loading takes. Furthermore, the flight durations between airports, which depend on the airplane type, are specified.

Each request has a given *cargo type*, *cargo quantity*, *pickup airport*, *dropoff airport*, and *deadline*; it might also be associated to a required airplane type. Any feasible solution to the request set requires that each dropoff occurs during the week before its deadline, which is the *delivery window* of the request. The type of an airplane determines how much total cargo (and in some cases how much cargo of each type) can be carried. Multiple pickups or multiple dropoffs can occur simultaneously. The pickup and the dropoff of each request must be handled on the same day.

Our aim is to find a solution, or a *schedule*, that uses

as few airplanes as possible; a secondary objective is to minimize the overall flight time of all planes. Our specific test instance involves 69 airplanes, 13 airports, and over 3600 requests, which span a complete year.

4 Algorithms

We divide a complete schedule into *airplane schedules* for each airplane. Each airplane schedule can be broken up into a set of *routes*, where each route is the portion of the airplane schedule for one specific day.

A key algorithmic subroutine used in all phases of optimization is a branch and bound algorithm, which takes the set of requests on a route and returns the time-minimal schedule for that route. For our test instance, each route can typically have about 15 requests (30 operations) before it becomes infeasible. By utilizing various carefully engineered aggressive bounding strategies, our branch and bound algorithm can solve such instances in under a second.

We define a *move* to be taking one request from one route, and adding it to another route. When considering a move, each route is independently, perfectly optimized by the branch and bound algorithm. An alternative approach would have been to have a move specify how a new request would fit into a route schedule, but this greatly increases the space of possible moves; we have found this approach much more effective.

Armed with a search space defined by the possible moves, our HuGS framework allows the user to employ several different local search algorithms. The two we have focused on in this work are a greedy and tabu search. The greedy algorithm simply examines the space of possible moves until it finds a move that improves the current schedule; this move is then implemented, and the greedy search continues. Various optimizations increase the efficiency of this search; see [1] for more details.

As an example of how these algorithms are used, we outline how the branch and bound and greedy algorithms are used to generate an initial solution schedule for the user. We attempt to generate a feasible solution by sequentially putting each request R in the best route given the current route for all previously placed requests, using the branch and bound algorithm to evaluate each possible placement of R. This algorithm already provides reasonably good initial solution, currently in under 30 seconds for our test problems. Then the greedy search is run non-interactively until no further improving moves are possible; that phase results in a locally optimal initial solution. This second step can take up to 45 minutes, but this is acceptable since initial solutions are generated infrequently.

When using interaction, we have found that tabu search, another local-search based heuristic [5], generally performs better than greedy search [7]. 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.

The HuGS tabu search makes use of the mobilities used in the visualization framework to encourage diversification. After a move, the mobilities of certain requests are modified to prevent moves involving certain requests and allowing moves of other requests. A more detailed description of how mobilities are in general used within HuGS can be found in [7].

5 Approach to the Problem

We describe our program in terms of a cycle of three user activities: inspection, modification, and reoptimization of the current solution.

5.1 Inspection

In this state, the user is the one in control. She can browse through the entire solution and check if all her soft constraints are satisfied.

The application has to assist her by providing appropriate visualizations, both for giving an overview, and to examine specific details. This is best done by providing different scales of zooming, and a carefully chosen set of possible focuses. Our visualization supports the *zoom, focus, and drill down* paradigm of information visualization, although we have to design it for the added challenge of making modifications possible.

- **Zoom:** A year- and a month-view generate overview visualizations. Due to their level of aggregation, it is not possible make modifications, but they give an overall sense of the data, and show where problems remain. A "x-days" mode presents the solution in a detailed and therefore editable manner, always showing x consecutive days at once, where x can range from 1 to 10 (which gives a fairly good overview).

- **Airplane- vs. Airport-View:** Whereas the x-axis of the view is always the date/time dimension, the user can select the y-axis to represent the requests either per airplane or by airport. While the first one is in general more useful for modifications, the latter one can give valuable information about the usage of airports.

- **Coloring Schemes:** Each request can be colored depending on the information the user wants to emphasize. E.g. the user can select "Deadline" to paint requests that are near their deadline in red, or "Capacity" to see whether a plane is full (red), empty (green), or something in between. We also support a coloring scheme that uses one default color for all requests to reduce the visual distraction.

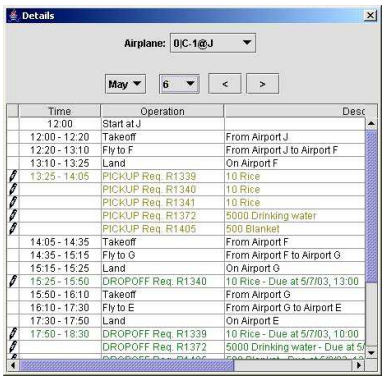

Figure 1. Tabular details

- **Filters:** The user might narrow the set of visible requests by filtering them by their cargo type, airport or airplane.

Figure 2 shows a view representing the schedule of the whole year. It shows a row for each airplane used; the x-axis denotes the dates, with vertical lines separating the months. The gray background color means that the airplane is not available at this time, whereby white regions denote availability. Colored dots represent the actual usage of a plane, with the color describing how full a plane is.

Figure 3 displays the difference between the overview views, and the detailed x-days views: for each request, there are two pentagonic boxes representing its *operations*: its pickup (with the tip to the top) and its dropoff (with the tip to the bottom). They are labeled with an abbreviation of their cargo type and their airport. If there are multiple operations happening simultaneously, they are represented by a single box with a stacked look. Such a box can be easily expanded to see the details of each of its members.

Because of the sheer size of the problem we include helpful features. We aid the user by always automatically selecting the according dropoff when a pickup is selected (and vice versa). We provide detailed information on each request (Fig. 4). If the airports are selected as the base elements, we have show the flight route for the airplane that handles the selected request (Fig. 4).

Any infeasibly scheduled operation – usually introduced by the user's modifications – is colored in black (encircled in Fig. 3).

These examples are only a summary of the most important features, provided to assist the user in her task of inspection and modification. To further assist the user, we also provide her with a detailed tabular description, which is often easier to browse (Fig. 1).

5.2 Modification

If during inspection the user is not satisfied with the current solution, she enters the next state of the cycle.

We support various types of modifications (N2), including changing the airplane and/or the day when a request is handled. Requests can be added, edited and removed; all of the request's properties are changeable. Additionally, we can change the airplanes' availability specifications. There are different reasons to allow modifications:

- Requests, deadlines and other problem aspects might change; new requests might have to be added, or old ones canceled. Being able to adopt to such changes is a real-world requirement.
- The user may have real-world knowledge that is not modelled in the problem specification and might consider a valid solution flawed.
- The user may have some soft constraints or preferences, which she might want to apply to the solution.

Modifications are challenging. Because our initial solution is quite good, manually moving a request from one day to another often introduces infeasibility. Therefore, we have to provide the user with means to simplify that task. First, the branch and bound algorithm is run on each modified route, directly after the user makes a move (P2). Second, the user is explicitly allowed to introduce infeasibilities, and the solution is automatically repaired during the next reoptimization (P3). If the user drags a pickup (or dropoff) from one route to another, its corresponding dropoff (or pickup) is moved with it. After such a manual move, the affected operations are set to low mobility; thereby we force the optimization algorithm to repair any potential infeasability without simply undoing the move.

5.3 Reoptimization

After an inspection phase and/or a modification phase, the user is likely to invoke the human-guidable tabu search on the current schedule. The user's goal may be to concentrate computational resources on optimizing an import part of the schedule (P1), to fix an infeasibility introduced during modification (P2,P3), or simply to reoptimize some portion of the schedule after making a modification that either incorporated some real-world knowledge into the schedule or was performed in order to help the optimization algorithm out of a local minimum.

We found it difficult to manage both the focus of the optimization algorithm and the user's visual focus separately. Therefore, the reoptimization is only applied on the objects the user can currently see: e.g., if she has a view which shows the first week of April, the optimization algorithm will only modify the schedule on that week (N1).

Also, for this application, we needed to extend our optimization function so that, during reoptimization, it avoids making many changes to the current schedule for small cost reductions. This extension is needed because in many cases a small gain in the objection function is often not worth disrupting of the current solution. Changes in the schedule can

Figure 2. Yearview (Zoom = Year, Base Elements = Airplanes, Coloring Scheme = Capacity, no Filters)

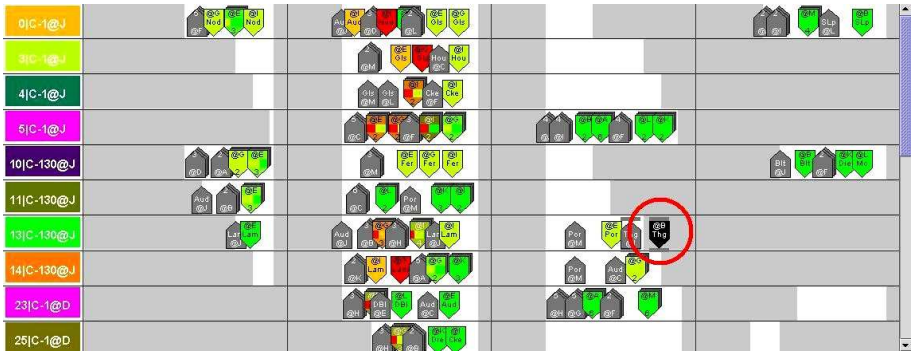

Figure 3. Detailed View (Zoom = 4-Days, Base Elements = Airplanes, Coloring Schemes = Deadline, no Filters): The black operation (here encircled) is infeasibly scheduled

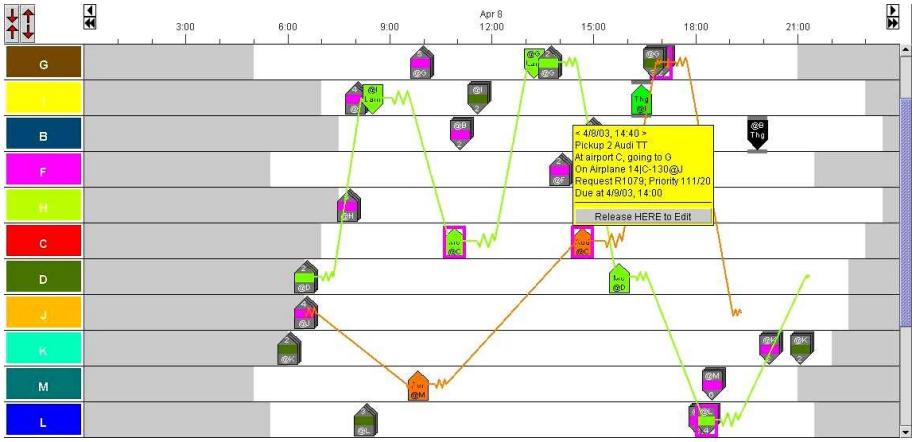

Figure 4. Detailed View (Zoom = 1-Days, Base Elements = Airports, Coloring Schemes = Airplanes, no Filters): flightroutes are shown for the marked operations; a tooltip for detailed information on a request

place a higher burden on the user, who may need to reinspect the schedule. They may also be highly undesirable in real-world applications. For example, consider a situation where the relevant schedule information for the following week has already been handed out to the pilots and airports, and there is the need to change the cargo quantity of a single request. Although a complete re-optimization might result in a better but very different solution, a solution as similar to the original one as possible will be preferred for organizational reasons (N3).

Therefore, our large-scale interactive optimization requires an (optional) shift of the objective function, to provide valid minimally-disruptive solutions. The notion of disruption itself depends heavily on the specific application, since a simple counting of the applied moves in general is not suitable. In our airlift scheduling application, it makes sense to count the disrupted routes, i.e. any route that has changed (no matter how much) from the original solution. Furthermore, we support different modes which allow, penalize, or forbid the use of routes which where empty in the original solution.

In all our optimization algorithms, we have two objective functions i and o. Objective i minimizes infeasibilities, and is 0 for all feasible solutions. Objective o – which is only considered for equal values of i – is normally the main optimization task of this domain, namely minimizing the number of planes and the overall flight time. To encode minimal disruption, we extended our system to utilize an objective s instead of o that is a weighted sum $s = wd + (1 - w)o$. The variable w is the weighting factor between 0 and 1, d the disruption. For simplicity, we currently provide the user with only two options for w:

- w = 0. This is traditional optimization in which the lowest-cost schedule is sought.
- $w = 1 - \epsilon$ (for small ϵ). The goal here is to repair infeasibilities in the schedule. Rather than finding a valid solution with minimal objective value, a valid solution that disrupts the current solution as little as possible is found. Our chosen value for ϵ is hereby 0.01 so that even a drastic change in o can never favor a more disruptive solution over a less disruptive one.

Using the second option we actually encode three objectives: We eliminate infeasibilities and minimize disruption. In addition, ϵ makes the minimization of the number of planes and flight times a third objective.

6 Conclusion

Large-scale interactive optimization systems add new contributions to the research topics of interactive optimization and information visualization. They offer significant advantages over other commonly used optimization strategies, by allowing for the user to inspect and change the optimization results. However, they also expose new challenges regarding modifiable visualizations, user-friendliness and algorithmic performance, which are met in this work through flexible changes in the visualization, and the means to minimally disrupt results during reoptimization.

References

[1] 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.

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

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

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

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

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

[7] G. W. Klau, N. Lesh, J.W. Marks, and M. Mitzenmacher. Human-guided tabu search. *In Proc. of AAAI 2002*, pages 41–47, 2002.

[8] H.A.D. do Nascimento and P. Eades. User hints for directed graph drawing. *Graph Drawing*, 2001.

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

[10] 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.

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

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

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

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