So You Want to Build an Automated Scheduling System

David Montana
BBN Technologies
10 Moulton Street, Cambridge, MA 02138
dmontana@bbn.com

Abstract

The BBN scheduling group contracts with both government and industry to perform research and development for genetic-algorithm-based scheduling. We have developed custom real-world automated scheduling systems for different clients. Here, I relate some of the insight gained from our experience organized into a few broad categories.

1 SELLING THE PROJECT

Clearly, one of the critical parts of building an automated scheduling systems is securing the commitment of sufficient money and resources to accomplish the job. Selling such a project within an organization requires a champion (or champions) inside the organization, someone who will put their influence and credibility behind the effort. As an outside contractor, one of our toughest jobs is trying to find such a champion. If you are an insider in the organization, you are at a big advantage because you can be that champion or know who else to approach.

There is likely to be a lot of resistance to the proposed project because there is a lot at stake. Scheduling is often at the heart of the business process, and changing how the scheduling is done likely means an overhaul of this process. The results can affect not only the organization's bottom line but also its people, both those people responsible for creating the schedules and (in many cases) those whose work is being scheduled. For example, when we were creating an automated scheduler for assigning computer technicians to calls for service [Montana *et al.*, 1998], the technicians were very upset at their loss of autonomy in creating their own schedules. (Their choices were often bad and were causing an inefficiency in the operations.) As another example, the Air Force was motivated to have us build an automated crew scheduler for them because their current approach was leading to unpredictability of schedules, upsetting their people's family lives and causing high turnover [Rana-Stevens *et al.*, 2000]. An additional effect of implementing an automated scheduler is that it can sometimes require an overhaul of the Information Technology (IT) systems to get the right data to and from the scheduler in a timely manner. This will likely make the IT systems managers unhappy.

It is important that the champion(s) have the ammunition required to sell the project. To start, the champion requires a cost-benefit analysis that will justify the project. This should be a realistic analysis because the project will likely be judged against it. Another very useful sales tool is a prototype system that shows the possibility of building the real system and in broad terms how the real system would work. In multiple cases, we have used such a prototype as a means to convince skeptics of the feasibility of automating a process that they previously felt could not be done by a computer. Additionally, one should remember that the selling of the project is a continuous process. Along the way, there should be demonstrations of progress and early releases to get people not doing system development involved in the process.

2 A CONCEPT OF OPERATIONS

The concept of operations is how it is envisioned that the system will be used. It is important to develop a concept of operations early in the process because it determines key requirements of the system. Such requirements include: (i) interfaces to data sources and sinks, (ii) schedule turnaround times (which in turn can affect which algorithm to use as discussed below), (iii) user interface functionality, and (iv) automated scheduler logic and functionality.

There is a wide range of possibilities for the concept of operations. For example, consider the issue of when to run the automated scheduler and how long it should take to create schedules. This can be as simple as the scheduler making each day's schedule the night before and printing out the day's schedule each morning. However, it can also involve dynamic rescheduling, i.e. having the scheduler revise the schedules as they are being executed. For example, a system we built for scheduling computer service technicians needed to create a new schedule every ten minutes. Many of the service calls required attention within a few hours after being received, so to be able to fit these calls in a reasonable manner into the schedules required a fast reaction. A system we built for aircrew scheduling for the U.S. Air Force had the automated scheduler execute only when there were changes that necessitated a new schedule. These changes were either new or modified missions arriving from headquarters or a human scheduler making manual modifications.

Another source of variation in the concept of operations is how to involve the human user. It may be as simple as the automated system having the final say with no human scheduling inputs. For our computer service scheduler, the human user needed to be able to override the scheduler in certain situations, because at times the human scheduling agent worked out with the customer certain agreements to which the computer was not privy. For our aircrew scheduler, a human user was required to take responsibility for each assignment. Therefore, we introduced the concept of approval whereby once a human signed off on a particular assignment, there was a cost in the evaluation function for changing that assignment. This cost increased as the time to execute the assigned mission drew closer.

3 SCHEDULING ALGORITHM SELECTION

In the logic of scheduling, there are two different types of constraints, hard and soft. Hard constraints cannot be violated, while soft constraints can be violated but at a cost. In a well-defined scheduling problem, it is generally easy to find a legal schedule (i.e., one that satisfies the hard constraints) but impossible to find one also satisfying all the soft constraints. The goal is to find a legal schedule that does as well as possible at satisfying the soft constraints. In a well-formed problem, the soft contraints are combined into a single evaluation function that produces a score measuring the quality of any legal schedule. Schedule optimization is the process of searching for the optimal schedule among the very large set of legal schedules.

There are a variety of different types of algorithms for performing this search. The major differentiators between different scheduling algorithms are (i) the quality of the schedules they find, (ii) the amount of computation time required to find these schedules, and (iii) the ease with which they can be adopted to a particular scheduling problem. There is no single type of scheduling algorithm that is best for all problems. Which algorithm is best depends on a variety of factors, including the nature of the problem, the required quality of the solution, and the time allowed to find a solution. For example, the airline flight scheduling problem, where a new schedule is required about once a month and where every percentage point difference in the schedule translates into millions of dollars, might best be solved by an operations research technique (such as mathematical programming) that takes a long time but finds the best schedule. However, our computer service scheduling problem (see above) that requires a new schedule every ten minutes needs a faster algorithm (at the expense of some schedule optimality), such as a genetic/evolutionary algorithm or a simple heuristic.

We have found genetic algorithms to be a highly effective all-purpose optimization method for a few reasons. First, they are easy to apply to almost any optimization problem, including those with odd constraints that may derail other algorithms. Second, compared to other general-purpose optimization techniques, including standard operations research techniques, genetic algorithms are fast at finding good (albeit often suboptimal) solutions, particularly as the problem size increases. Third, genetic algorithms allow an explicit tradeoff between the search time and the quality of the solution. Fourth, genetic algorithms, with their population-based approach, allow for easy and effective large-scale parallelization.

Once you have chosen to use a genetic or evolutionary algorithm, there are a number of algorithmic design decisions to be made (and usually some software to be written). The most critical decisions are how to represent a schedule as a chromosome and how to define the genetic operators (often crossover and mutation) that generate new chromosomes from existing ones. A direct representation has a chromosome be the schedule, or at least an abstract version of the schedule. In this case, the difficult part is defining operators that maintain the hard constraints. An indirect representation has a chromosome provide information on how to build a schedule that is passed to a schedule builder. Here, the knowledge of how to satisfy the constraints

is contained in the schedule builder. A chromosome in an indirect representation can be as simple as the order in which the schedule builder should consider the tasks, but can also contain more complex instructions.

Besides the representation and operators, there are some secondary, yet still important, considerations in designing a genetic schedule optimizer. One such consideration is how to initialize the population. Particularly when using a direct representation, it can be highly beneficial to use better-than-random individuals during initialization. Another consideration is how to select parameters of the genetic algorithm, including those for population size, selection pressure, and termination condition. These directly influence the tradeoff between schedule quality and execution time and should be selected based on the problem requirements and the available hardware.

4 KNOWLEDGE ENGINEERING

One of the big challenges of building automated scheduling systems is that most scheduling problems are unique, with a logic and a set of business rules that are different from other problems. For many problems, the logic and business rules are complex and not well documented, so it becomes a major challenge of the project to capture them in a systematic way.

Currently, for many scheduling processes, humans do the scheduling manually, and the logic for how to do it only fully exists in their brains. The scheduling system developer needs to extract this knowledge. Using the right approach can help to do this, as well as providing a chance to gain the good graces of the experts, who are often also the intended end users of the system. The experts can be resistant at first due to the feeling that an algorithm cannot capture their expertise. However, if you get them involved in the development process, they will often eventually develop a pride of ownership.

You need to ask the right questions to get the right information. One good approach is to have the experts step through the entire (current) scheduling process. Another good question is to ask what makes a good schedule, and to supply a variety of different hypothetical situations to try to capture the concept of schedule quality as thoroughly as possible. A third question is what type of hard constraints must a schedule satisfy.

Capturing the scheduling logic and business rules is an iterative redesign process. Do not expect to achieve success on the first pass. Instead, just implement the business rules as you have captured them, and execute the scheduling algorithm on sample data. When the algorithm fails in a (human-)noticeable way, it usually is not because the algorithm is reaching a suboptimal solution (at least not if you are using a genetic algorithm) but rather because of bad business rules. Work with the experts to identify the problem, fix it, and then repeat the process.

5 INTEGRATION WITH IT SYSTEMS

Since scheduling is often at the center of operations for an organization, an automated scheduling system often must interact with some of the organization's critical IT systems. Even when the scheduling system is at the periphery of operations, it must get its data from somewhere and therefore, unless the user enters the data, must interface with other system(s).

There are a variety of types of data for input and output. The scheduling system needs to read the data about the tasks and resources, usually from some existing databases (such as a personnel database or an orders database). Often, the scheduling system also needs to report the schedule as it was executed for the purposes of billing, payroll, etc., and hence needs to communicate with some other system(s). A third type of interface is for monitoring the schedules during execution. This involves the scheduling system receiving feedback so that it can update the schedule based on reported deviations, such as a service call taking longer than expected or a mission getting delayed.

It is important to define data requirements early in the development process. Then, all involved, including the IT people, know what types of data feeds are needed, as well as estimates on the data rates. If this is going to overtax the existing systems in any way, then there is time to remedy this problem. It is also important to define the formats and protocols for the interfaces early in the process. To the extent possible, the scheduling system should use existing interfaces. However, for those cases where it is not possible, it is important to identify the shortcomings and define new interfaces or extensions to old interfaces soon enough that there is enough time for their development.

Our experience has been that introducing real-time scheduling at the center of an organization's operations can identify deficiencies in the organization's IT systems. Correcting those deficiencies can improve operations as much as the automated scheduler itself. For example, for our computer service scheduling system, the company had previously handed service engineers their schedules at the beginning of the day and had only contacted them during the day in case of an emer-

gency. To perform automated scheduling, they had to create the capability to track the service engineers during their day's work, and this capability provided them large benefits beyond better schedules.

6 ALGORITHMIC TEST CASES

Test cases are a standard technique for software quality assurance. They provide scenarios to verify that the software is functioning as envisioned. For an automated scheduling system, it is important to have a set of test cases specifically designed to test the correctness of the scheduling algorithm in addition to those for testing the software. As new logic and business rules are added to improve the scheduling algorithm in certain areas, the test cases allow regression testing to verify that these changes have not caused problems in other areas. The test cases can also be very useful in convincing skeptics that the software can make good decisions.

The test cases should cover, in some sense, the full range of possibilities that can be encountered by the scheduling algorithm. For each class of business rules, there should be one or more test cases that specifically address this rule. These targeted test cases need to be manufactured so that all else is factored out and so that there is a clearly defined correct answer. A second set of test cases should address tradeoffs made in the optimization criterion. It is often tricky to tune the tradeoffs between the different objectives in a multiobjective optimization function, and these test cases allow a systematic approach to this tuning. A third set of test cases deal with scalability and the ability to handle real, rather than manufactured, data. These test cases should contain at least one or two datasets that are as large a problem as the system will be expected to handle. These test cases will have no single "correct" answer, but the ability of the scheduling algorithm to produce reasonable answers within a time limit provides a type of check that smaller, manufactured datasets cannot.

7 SYSTEM DESIGN AND IMPLEMENTATION

One big decision to make when designing a scheduling system is what the software architecture should be. For a scheduling system that has multiple users and external data sources and sinks, a client-server architecture is what makes most sense. A central sever has a database that contains the current data and schedule. The user interfaces, external data interfaces, and the automated scheduler are all clients reading from and writing to the central database. Using a web server as the central server and web browsers as the user interfaces can make maintenance and update of the system much easier at the expense of reduced interactivity in the user interface.

Another big decision is whether to use third-party software or build the system from scratch. There are many scheduling software packages available for sale, or even for free download via the Internet. Most of these scheduling packages solve a particular type of scheduling problem, e.g. crew scheduling or tournament scheduling. If such a package satisfies the needs of your problem and is available within your price range, then use it instead of developing your own.

In addition to problem-specific scheduling solutions, there exist *reconfigurable* scheduling systems that can be tailored by the user to a wide range of different types of problems. For scheduling problems that can be formulated as mathematical programming problems, there are a variety of general-purpose solvers, many based on AMPL [Fourer *et al.*, 1993], including ILOG's CPLEX product. For those scheduling problems that cannot be expressed as mathematical programming problems, there are two reconfigurable scheduling packages with much greater flexibility in expressing constraints. ILOG sells a constraint-based scheduling product based on OPL [Van Hentenryck, 1999]. We provide our Vishnu genetic-algorithm-based scheduling software for free download [Montana, 2001b]. While the ILOG product is more mature, Vishnu is free and is provided open source (hence allowing the user to customize the software).

An alternative to buying or downloading a full scheduling system is to just download a genetic algorithm (or other optimization algorithm) package. This will save some time developing the scheduling algorithm (although you will still have plenty of customization to do). However, if you are developing a full distributed system, the time savings will be a very small fraction of the overall development time because you will still need to develop the distributed infrastructure, the database, the user interface, the data interfaces, etc. These other pieces of the system are what take the majority of the development time.

One other thing to remember when designing and implementing an automated scheduling system is that it is still a software system and all the standard rules for developing such systems still apply.

8 FUTURE TRENDS

The use of evolutionary algorithms for real-world scheduling applications has been very successful, to the point where it is now almost commonplace. The goal for the future is to extend the range of applicability of evolutionary scheduling, particularly to problems that were previously not amenable to automated, optimized scheduling. There are two directions in which to proceed.

One direction is to make it cheaper and easier to develop automated scheduling systems so that the technology can be applied to more problems [Montana, 2001a]. The high costs of developing a scheduling system largely from scratch (plus the training and maintenance for such a system) mean that such development can only be justified when there is a lot of money depending on the schedules. Replacing custom development with reusable software would greatly reduce these costs and open up the range of possible applications. There is currently the beginnings of a push towards reconfigurable schedulers, i.e. automated optimizing schedulers that can be used for a wide range of scheduling problems without the need for modifying the software. It would also be beneficial to have open standards to allow easy swapping of scheduling components, but that is not on the near-term horizon.

A second direction is towards multiple schedulers interacting. This enables two important capabilities. One is the ability to solve larger-scale problems by decomposing them into smaller problems. The second, and more important, is the ability of multiple organizations to automatically coordinate their schedules [Montana *et al.*, 2000]. This is critical to fully automated flexible supply chains and other forms of automated business-to-business (B2B) transactions that rely on fitting the transaction into multiple schedules.

References

[Fourer *et al.*, 1993] R. Fourer, D. Gay, and B. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, Belmont, CA, 1993.

[Montana *et al.*, 1998] D. Montana, M. Brinn, S. Moore, and G. Bidwell. Genetic algorithms for complex, real-time scheduling. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 2213–2218, 1998.

[Montana *et al.*, 2000] D. Montana, J. Herrero, G. Vidaver, and G. Bidwell. A multiagent society for military transporation scheduling. *Journal of Scheduling*, 3(4):225–246, 2000.

[Montana, 2001a] D. Montana. Optimized scheduling for the masses. In *Genetic and Evolutionary Computation Conference Workshop Program*, pages 132–136, 2001.

[Montana, 2001b] D. Montana. A reconfigurable optimizing scheduler. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1159–1166, 2001.

[Rana-Stevens *et al.*, 2000] S. Rana-Stevens, B. Lubin, and D. Montana. The air crew scheduling system: The design of a real-world, dynamic genetic scheduler. In *Genetic and Evolutionary Computation Conference Late Breaking Papers*, 2000.

[Van Hentenryck, 1999] P. Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, Cambridge, MA, 1999.