Rapid Scheduling: A Constraint Satisfaction Approach to River Usage

Team: 16677

February 12, 2012

Contents

1 Preliminaries

1.1 Introduction

1.2 Definitions and Terminology

1.3 Assumptions and Simplifications

1.4 General Commentary

2 Details of the Model

Phrasing the problem in terms of scheduling allows us to harness the power of existing approaches and algorithms developed within Computer Science. We eventually settled on structuring the model as a constraint satisfaction problem, a group of generally NP-hard problems. The general definition for such a problem is that - given variables (X), values (D), and constraints (C) - find an assignment of value(s) in D for each X such that all constraints in C are satisfied. Good examples of this problem set are 2-SAT and coloring problems.

We view the problem of scheduling rafting trips as a progressive and dynamic CSP: each day being a constraint satisfaction problem dependent upon the solution to the previous day's problem. By limiting the values any variable may take on and applying efficient algorithms, the CSP becomes tractable and we may successfully schedule river trips without the need of unreasonable computing power.

2.1 Constraint Satisfaction

In general, our approach is to first consider that there are exponentially many combinations of travel groups to campsites during their trip. Many of these combinations, however, are invalid given the constraint of no two travel groups being allowed to stay at the same campsite. This is perhaps the greatest limiting factor in the problem as it applies to all campers passing through the same territory on a given day. This is an $n - ary$ constraint: the constraint relating n variables. Such constraints are computationally challenging for many problems, but our problem space is limited enough that the computations are relatively painless ($O(n^2)$ with n as number of travel groups) to pair all campers and detect collisions at any campsite. There were few other constraints involved in solving the problem as we were mostly interested in solutions to our schedule.

As we have been discussing, each travel group was considered to be a variable in the CSP and campsites were considered the values. The domain of these values (for each travel group) varied day to day depending upon the distance each group was capable of traveling. This travel distance was derived from a group's average speed of 4 or 8 mph, v, multiplied with a uniformly distributed random variable for desired daily time spent on the water, t. This variable is such that the travel distance per day would allow the group to complete the trip within the allotted time of 7-19 days. The following few equations explicitly describe these relationships:

$$v * t = OptimalDayDistance$$

$$v * t * 7 \geq L$$

So in generating any travel group, we ensure the group is reasonably parameterized for finishing a trip on time. Each group is also randomly assigned, uniformly distributed, a day for departure (dDay) such that

$$v * t * 7 + dDay \leq S$$

That is, a group's maximum travel speed will allow them to finish before the close of the season.

Groups are generated en masse using a varitey of uniformly distributed variables. It is possible to load in a file containing group information so that the program will work on problems of corporeal, rather than testing and verification, import. Once groups are generated and basic parameters are set (discussed in **Section 2.2**) the program begins attempting to produce a solution through our CSP solver.

2.1.1 Backtracking Search

In order to solve a CSP, one must navigate the vast multitude of combinations for variables and values. We utilize a backtracking search algorithm to assign campsites to our travel groups on a daily basis. On any given day d, the solver looks to see which groups are set to depart on d and subsequently adds them to a list, *toDepart*. All groups still on the river from the previous day are added to *toDepart*, creating a list of all travel groups requiring some movement for day d. In order to determine appropriate movements, the groups are assigned campsites (analagous to distance traveled for the day) one-by-one. Following a pairing, the full day's assignment is checked to see if this pairing is consistent with current progress. If the pairing doesn't violate any other existing pairings, it is added to the day's list of assignments. We then proceed to the next travel group.

Encountering failed pairings causes the algorithm to attempt another pairing until success is achieved or failure must be returned. If failure occurs, the algorithm returns to the most recent pairing, undoes this pairing, and then attempts a new pairing. The entire process is structured as a depth-first-search. When a day's assignment is complete, the algorithm creates a new CSP for the next day and executes again, deepening the recursing into another day. **Figure 1** highlights the major steps involved in this process and provides a simple example. In the example, the river length is only for two campsites; there are only two travel groups; and only the observed states are shown. Each level of the tree represents the possible assignments to a single travel group. The entire left side of the tree is expanded to a solution, in which case the solver would return a schedule for each group along with various statistics.

BETTER DISCUSSION OF THE FIGURE AND IT'S CONSEQUENCES

Figure 1: An example of the backtracking algorithm's attempt to schedule travel groups A and B with campsites 1 and 2.

Our solver is optimized to attempt maximal satisfaction of the travelers. That is, the possible campsites a group may travel to are presented to the solver in order of ideal travel distance. While not every group is guaranteed to always travel their desired distance, the algorithm attempts to satisfy this "fuzzy constraint". We also calculate the number of encounters between groups. After the itineraries are created, each group's campsite listing is checked to determine how many groups passed and were passed by this specific group. By combining the amount of erring from ideal travel distance and encounters with other groups, we obtain the aforementioned effect on satisfaction. This effect is calculated post-solution. Throughout the course of operation, the solver relies heavily on several parameters to relax the problem and make solutions more attainable.

2.2 Usage of High-Level Parameters

Given some of the information in the prompt and our findings, as discussed in **Sections 1.2** and **1.3**, our program is designed to adjust its operation depending on various conditions. The parameter of greatest variability, number of travel groups, effects how many groups we generate at the beginning, the load distribution on portions of the season, and some of the satisfaction scores for other groups. Varying this value was crucial in analyzing the model, discussed in **Section 3.2**.

Values that tend to be static include the river length, number of days in the season, and number of campsites. While each of these is capable of drastically effecting the outcome of any given attempt, the values have a well-defined range (see **Section 1.3**). The problem could be generalized beyond these variables such that the CSP solver attempts to find optimal values for these as well, but this would likely place our problem into the realm of truly NP-hard problems.

2.2.1 Tolerance

Built into the algorithm is a *tolerance* variable: designed to create a range for the distance a travel group may cross in a given day. This variable is straightforward at the outset but provides several neat nuances for additional complexity in the solution. The introduction of a range to the CSP allows for a larger domain of solutions and, therefore, variability. In our case, we desire such flexibility, to account for the real-world implications of directing human beings. Tolerance extends within the model as a sort of aggregate for many of the various instances where we might expect variability — such as inclement weather, cancellations, or slow travel groups—, without having to introduce variability into every aspect. This may seem like a stretch of the model's design, but any fuzziness for any aspect introduces variability into the solutions. We are satisfied that tolerance is an opportunistic variable for allowing more groups to be scheduled and thus for the rafting season to be enjoyed by more individuals.

3 Predictions and Analysis

3.1 General Expectations

3.2 Results

3.3 Addressing Carrying Capacity

3.4 Sensitivity Analysis

4 Conclusions

4.1 Advantages to the Approach

4.2 Detractors from the Approach

4.3 Future Extensions

4.4 Final Recommendations