Finding the Perfect ‘Job’

Angela Sy

Stanford University

December 4, 2015

Abstract

In this increasingly complex world, more and more industries are recognizing the importance of efficient process and task scheduling. Applications can be found in the manufacturing industry (construction of parts through batch processing), the services industry (delivery scheduling through route optimization), and even in the technology infrastructure industry (improving network speed through server time allocation). Deciding when to assign a task given a set of job requests requires optimization of multiple factors, including attributes unique to each job as well as factors dependent on other sections of the workflow. This study focuses on solving the “service-request” scheduling problem in the Kitchen Services Industry, where a “service-request” is defined as a customer request or a job that needs to be completed in a finite period of time. The goal of this study is to create a scheduling algorithm that takes domain specific inputs (set of service-requests, list of available resources, and prioritized performance objectives) and returns a schedule that maximizes for customer utility. Multiple methods were tested in this analysis and, ultimately, it was found that an optimization algorithm was the best method to solve this problem.

1 Introduction

A scheduling decision requires an allocation of resources to tasks that enable the user to reach certain objectives. Examples of these objectives are minimizing time, minimizing production cost, maximizing profit, and maximizing customer utility. The scheduling problem has three components – allocating resources to tasks, sequencing tasks, and finally timing execution of each of these tasks. We focus on the last component – determining when each task should be scheduled and, more specifically, determining on which day each task should be assigned. We simplify the problem into a static problem and assume that task requirements and job times are known beforehand. Although in reality, scheduling is a dynamic problem with new jobs arriving at unexpected times throughout the execution of already scheduled tasks.

In general, much research has been performed in finding optimization methods within the domain of Job Scheduling – the class of problems where ideal jobs are assigned to resources at particular times. The challenge with many of these methods is the heavy computation time required – the number of potential schedules increases exponentially with the number of tasks. Despite imposing constraints on jobs and on the overall schedule, the scheduling algorithm becomes overburdened with the size of the search space and the job scheduling problem has been found to be NP-hard. However, heuristic methods can be used to reduce the computation time for the scheduling problem, which will be similarly used in this study.

Relevant literature in this field includes Optimization Methods for Batch Scheduling [1] and experimentation on heuristic and genetic algorithms for Resource-Constrained Scheduling [2]. These research papers look at different types of objective functions focusing on makespan, earliness, tardiness, profit, cost, and inventory. The approaches taken include heuristic rule-based methods such as basic dispatching, improvement or search methods that use simulated annealing, and optimization methods on both discrete time models and continuous time models.

In this study, we will implement both a greedy algorithm and an optimization algorithm to create a service schedule using data on service requests from a kitchen company [5]. The paper is structured as follows: Section 2 describes the setup of the problem and the given data, Section 3 analyzes the performance of the greedy algorithm and the optimization algorithm on the data set, and finally Section 4 concludes with further extensions within job scheduling problems.

2 Setup

For a given set of service requests, we produce a schedule that maximizes customer utility based on a pre-determined objective function. This mimics the scheduling process performed by the human scheduler but creates a series of possible schedules and chooses the most ‘ideal’ schedule. We experiment with two different methods and use specific metrics to evaluate the performance of each method.

2.1 Objective Function

The objective function maximizes total utility of the customer given a request r_i and a proposed schedule date s_i.

$$U(r_i, s_i) = (s_i(date) - r_i(date)) + r_i(urgency)$$

The first component of the utility function places a premium on finishing jobs that are requested earlier to ensure that requests are completed as soon as possible. The second component of the function ensures that high priority requests are handled first. Different weights can be placed on these components to refine the objective function.

2.2 Data

Our data is taken from a kitchen servicing company that fields 10 crews who fulfill a total of 25 servicing requests each day. The data set consists of 200 service requests representing a week of service requests in the queue.

Each service request has the following attributes: service request date, urgency, and time for completion. These factors will be used to determine the appropriate time slot for each request.

2.3 Scheduling Process

The company’s current scheduling process consists of compiling a queue of service requests. These are manually sorted and entered into a week’s work schedule at the start of every week. As emergency requests come in, modifications are made to the schedule dynamically. We will focus on the static component of this scheduling problem and use our algorithm to create an optimal schedule which assigns each service request from the queue to a specific work day.

2.4 Methods and Constraints

We first implement a greedy algorithm on the data – calculating the utility of each service request, ranking each request according to its utility, and then filling each day of the schedule with the highest ranked requests. Secondly, we implement an optimization algorithm – creating a linear function with weights representing job assignments to specific days and with coefficients that represent the utilities for each job-day pairing. We solve for the optimal weighting to determine the ideal schedule.

This scheduling problem includes specific constraints, namely that each request can only be scheduled once and a limit on the number of work hours for each crew per day.

2.5 Metrics

The optimal schedules created by the two algorithms are compared to the gold schedule created by the human scheduler. Specific metrics for success are the accuracy of request-day matches, its variance over different time periods, the percentage of requests scheduled, and the total utility of the final schedule created.

3 Experiments

3.1 Greedy Algorithm

In the greedy algorithm, we calculate the utility of each service request and rank each request, slotting requests with the highest utility earliest in the schedule and proceeding down the list of ranked requests as we build the schedule.

This method had an average accuracy of 39.72% with very high variance. Initially, the algorithm performed very well as the most urgent jobs were assigned immediately. However, as more days were added to the schedule and more potential pairings became possible, this method's accuracy steadily decreased. The percentage of request scheduled, however, remained consistently high over all time periods.

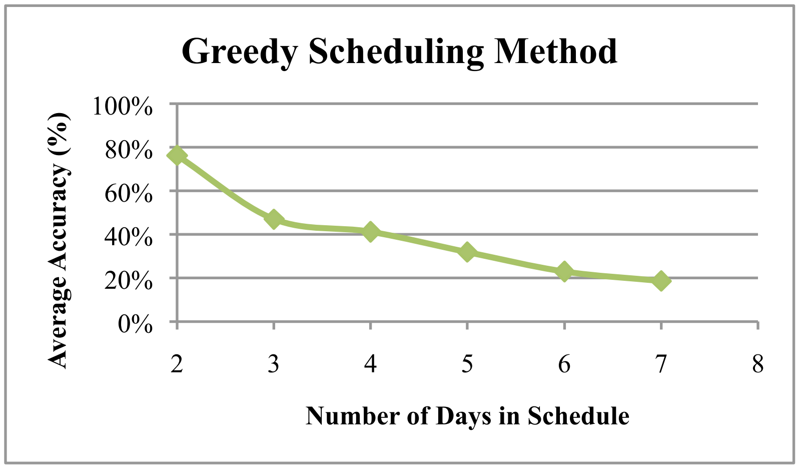

Figure 1: Accuracy of Greedy Algorithm on Schedules for Different Time Periods

In reality, there is a limitation on the number of high priority jobs that can be accomplished in the same day, both because of crew fatigue as well as limitations on the number of service vehicles and expert crews who are trusted with high priority jobs. Thus, the ‘gold’ schedule distributes urgent jobs across different days because of these factors, which the greedy algorithm is unable to account for.

3.2 Optimization Algorithm

The optimization function is given weights based on the utility of assigning each request to each particular day. The solution returns the day on which each request should be assigned to maximize the total utility of the schedule.

The optimization algorithm had an average accuracy of 21.64% with very little variance. The algorithm performs consistently over different periods of time. Furthermore, the percentage of requests scheduled is particularly high as most requests in the queue are scheduled within the week.

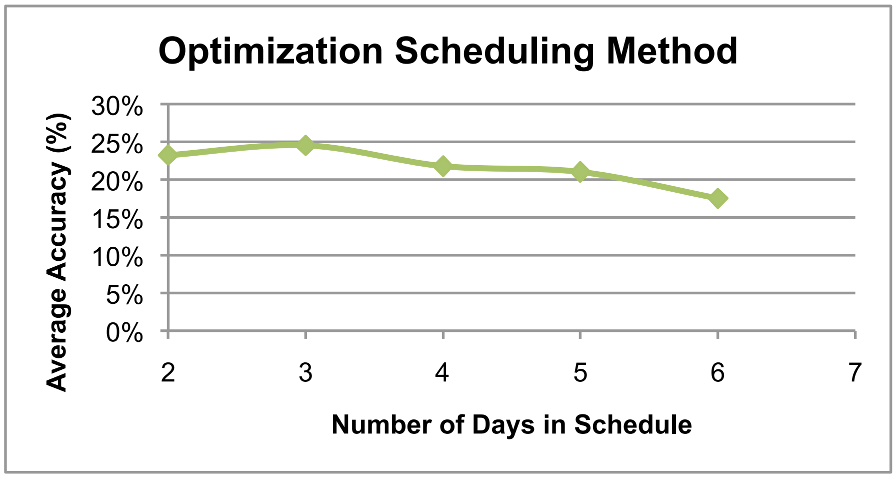

Figure 2: Accuracy of Greedy Algorithm on Schedules for Different Time Periods

The comparatively low accuracy of this solution can be attributed to the fact that many potential solutions are possible. Due to the small number of schedule constraints, the optimization algorithm returns just one of many possible optimal schedules. However, as more constraints are added to the optimization algorithm, solutions become more accurate.

One consideration against using the optimization algorithm is that as more requests are added, the optimization method becomes increasingly computationally expensive. However, we can implement heuristic methods to reduce computation and limit the search depth – for example, imposing a minimum utility bound or a utility ceiling.

3.3 Comparison of Greedy and Optimization Algorithms

The greedy algorithm had a higher average accuracy than the optimization algorithm. However, because of its high variance, the greedy algorithm will perform poorly when creating schedules for long time periods. On the other hand, the optimization algorithm is useful for long-term scheduling and can also be modified to address its shortcomings for short-period schedules. In addition, we can look at two other metrics to compare the performance of the two algorithms – total utility and percentage of jobs scheduled.

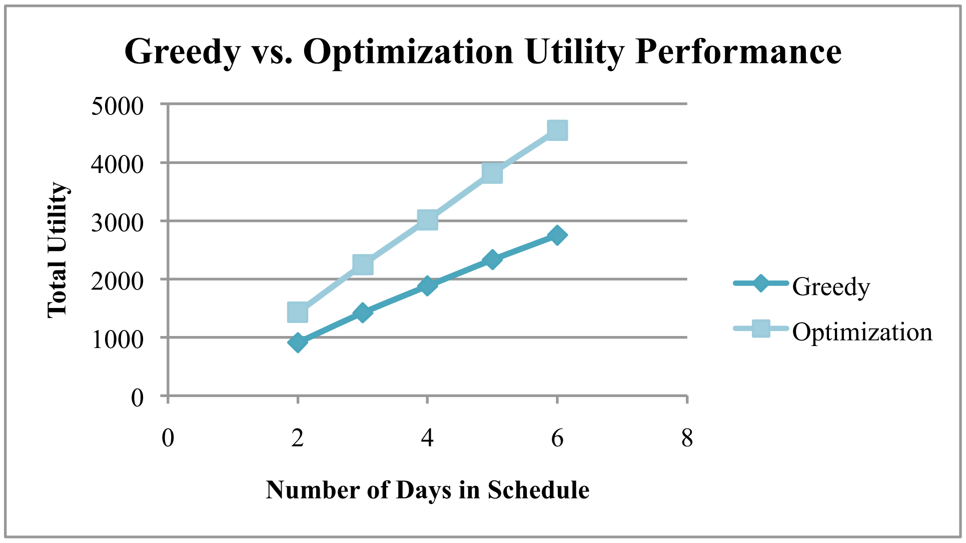

Figure 3: Comparison of Total Utility for the Two Methods

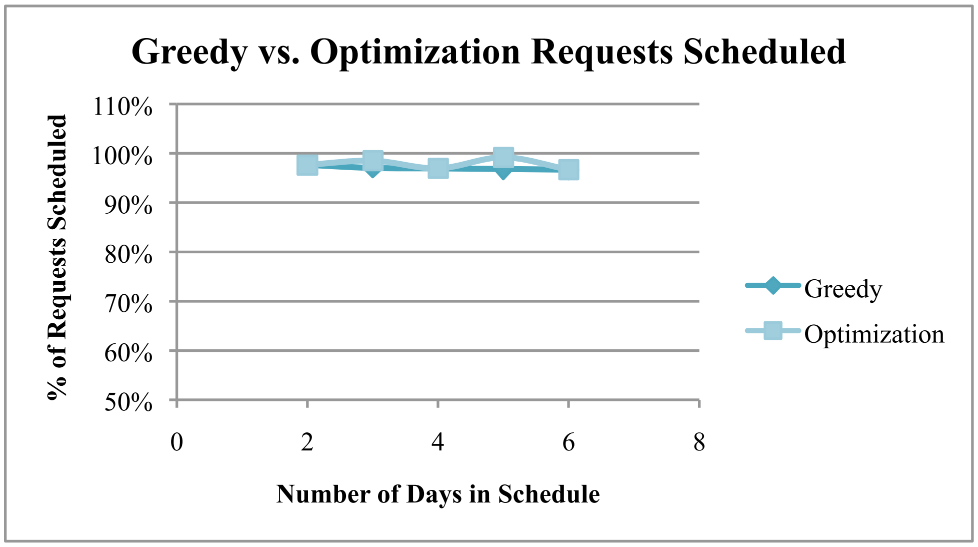

Figure 4: Comparison of Percentage Jobs Scheduled for the Two Methods

As we can see in the graphs above, the optimization method has a marked superiority in performance for total utility. While the greedy algorithm creates good short-term solutions, the optimization algorithm creates a more optimal schedule overall. In terms of the percentage of requests scheduled, both algorithms had comparable performance.

3.4 Improvements

In general, the performance of both algorithms can be improved by fine-tuning the objective function. Additional parameters can be added to the objective function such as the importance of a client and the location of service requests in order to group nearby requests and reduce travel time. These are considerations that the human scheduler takes into account which the algorithm does not yet consider.

Some constraints were not modeled such as service vehicle availability and a limit on the number of each type of service request that can be scheduled each day. This would reduce the number of high priority jobs that can be placed for each day of the week and would prevent the greedy algorithm from slotting all high priority events in one day.

Finally, additional complexities were not modeled such as specific service requests that require multiple days of work or jobs that require the same crew to work on a task.

4 Conclusion

The optimization algorithm is a better solution as it considers multiple potential schedules and allows for greater flexibility, allowing more factors and constraints to be added to the model. However, to ensure the feasibility of the optimization algorithm, steps must be taken to address the increase in time complexity with the increase of potential scheduling days.

One extension of this problem looks at adding a third component to the matching algorithm, which matches a job not only to a date but also to a specific crew. This would add an extra variable to the objective function and require new constraints, such as ensuring that the chosen crew is able to accomplish a specific service request. Additional complexity can also be added by considering different groupings of crew members, instead of just assuming that crews are pre-determined.

Another aspect for further research is the scheduling of emergency requests, specifically how to modify a set schedule when new job requests are added on-the-fly. The emergency request’s utility can be evaluated and the new request can potentially be swapped in for an existing job with a lower utility. However, the negative impact due to the disruption of the work schedule will also have to be weighted and taken into account.

As can be surmised from this discussion, job scheduling provides us with many exciting challenges and additional complexities. In all cases, pursuing a better allocation of time and resources to achieve greater efficiency is our eternal pursuit.

References

[1] Cerda, Jaime. "Optimization Methods for Batch Scheduling." Instituto De Desarrollo Tecnológico Para La Industria Química. Santa Fe. 5 Jan. 2012. Lecture.

[2] Bartschi Wall, Matthew. "A Genetic Algorithm for Resource-Constrained Scheduling." Department of Mechanical Engineering at the Massachusetts Institute of Technology (1996). Print.

[3] Crosby, Matthew, and Ronald P. A. Petrick. "Temporal Multiagent Planning with Concurrent Action Constraints." School of Informatics University of Edinburgh (2014). Print.

[4] Tompkins, Mark. "Optimization Techniques for Task Allocation and Scheduling in Distributed Multi-Agent Operations." Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology (2003). Print.

[5] Kitchen Service Request Data from Focus Global, Inc. Pioneer Corner Reliance Streets, Mandaluyong, Philippines. May-Oct. 2015.