April, 2014

Dynamic Task Scheduling Problem: Greedy Knapsack Solution

Christian Sandtveit
Darrin Winger
Jeremy Straub
Eunjin Kim

Available at: https://works.bepress.com/jeremy_straub/187/
The problem that we worked with was a dynamic scheduling problem. For this problem, we are given a set of tasks to be scheduled in an allotted time slot, so that the total value of the tasks done is maximized. Each task has a duration, value. Each task also has one or more periods in which they can be scheduled. Some tasks can have conflicting time slots that can prevent other tasks from being scheduled. As tasks are assigned time slots it is possible to prevent other tasks from being assigned a time slot. Looking for ways to minimize the possibility of a one task preventing another task from being assigned can help to improve the total value of the tasks done.

**Background**

Salman, et al. [1] proffer that “hard optimization” problems do not lend themselves to solutions by uniformly optimal solvers. Instead, they suggest that the optimization approach should be “tailored” to the particular problem instance.

The so-called “greedy knapsack” approach has been demonstrated in numerous problem domains. These include database indexing [2], credit screening [3], data transfer optimization [4], warehouse planning [5] and network traffic optimization [6]. In [7], Wrzuszczak and Borzemski utilized the approach as part of a business optimization (focusing on profit and other factors), similar to what is considered herein.

**Approach to Solving the Problem**

To solve the problem we decided to rank the tasks based on their value/duration ratio. We would take the tasks with the highest ratio and try to schedule them first. This would normally mean that the tasks with the shortest duration would be allotted a time slot first, since they would usually have the highest ratio. This way, we thought we would be able to minimize the conflicts between tasks, since the tasks done first would take up little space in the overall time period. The tasks that take a large amount of time and have the lower value would be added to the schedule last if they can add at all.

When adding the tasks to the schedule, we would add the tasks to the first open space in the schedule based on the possible periods the task can be assigned. We check each of the periods if there is an open space available in the first time slot so it can fit in. The color for task 5 in the images is red. One thing that we would have added, if we had more periods, would have been a way to shift the items in the schedule that have already been added to the schedule to the right in order to make room for more tasks before them.

**How it works**

We first read in all of the tasks to be added and order them based on value/duration ratio.

The order of tasks to be added to the schedule for the example data becomes 4, 5, 2, 1, 3.

The first task that is added to the schedule is task 4 because it has the largest value/duration ratio. We add the task that has the highest ratio because it is the task that has the most efficient use of time.

Next we add task 5 because it has the second largest value/duration ratio. It gets placed in the first slot available to it in the schedule.

Then we try to add task 1. There is not a spot available in the first time slot so it checks to see if there is another time slot that it can fit in. The second timeslot has room for the task so it is placed in the second timeslot.

**Conclusions & Future Work**

Our overall approach was a combination of a greedy algorithm and a minimal value constraint approach. Our approach finds a solution quickly but can cause some tasks to not be scheduled even if it is possible for them to be able to be scheduled. We learned that a good dynamic scheduler can be very complex. One thing that we would have added, if we had more time, would have been a way to shift the items in the schedule that have already been added to the schedule to the right in order to make room for more tasks before them.

**References**


