Feature Crew Scheduling and Resource Allocation

Meng Chao Lu and Lin Wang

March 12, 2007

CSEP 521 Project

Instructor Anna Karlin

**Introduction**

This paper first presents a feature crew scheduling problem that managers usually face in Microsoft and discuses a few algorithms that can possibly solve the problem. Then it talks about other scheduling problems that are seen today and some of the commonly used algorithms. Finally, it will discuss some of the more difficult problems that cannot be efficiently solved today.

**Problem Background and Description**

In Microsoft, the software engineering job function is broken done into three categories: Developers, Testers and Program Managers. Program Managers collect functional requirements for the product by taking customer feedback, analyzing market needs and aligning Microsoft goals with the product. Developers get the functional requirement specifications from the Program Managers, and implement them. Testers get prototypes from Developers and test them according to the functional requirements. As Testers find defects in the prototypes, they report functional defects to the Developers and specification errors to Program Managers in the form of bugs. The Developers fix the bugs and resend the prototypes to the Testers. The Program Managers fix the bugs and update the functional specifications. Developers update the prototypes and give them to the Testers. The Testers then test the prototypes and give further feedbacks to the Developers and Program Managers. The process repeats until the product is in a good quality. This process is called a development cycle.

Microsoft adopts the use of the spiral development process. Each development cycle is divided into several milestones. During each milestone, the Developers, Testers and Program Managers work on a unique set of features. At the end of the milestone, the product is at a shippable quality, but with added functionality comparing to the last milestone.

Each milestone is about 3-months long. During these 12 weeks, it’s impossible to implement all features in a sequence. It is much more efficient to implement all features at the same time & have an integration period towards the end of the milestone. If a feature is too large for a milestone, meaning that it takes more than twelve weeks to finish if done in a sequential order, then the feature will be broken down into feature chunks that can be done in parallel.

The PMs are given an initial set of features to investigate for the whole product. Then the Test, PM and DEV managers come together for an add/cut meeting to discuss and agree on the set of features for a milestone. Then the DEV manager discusses with the DEV team to break the features down into feature chunks and assign them to the developers. The developers work out a time estimation for each feature chunk. The original PM, and the Tester and Developer form a feature crew for the assigned feature, and work together for that milestone.

Sometimes this does not work out because some developers are given feature chunks that in total take more than 12 weeks to complete. So to solve this problem, developers who have less work can help out developers that have more work with this constraint: only Developers who have worked on related feature chunks have the expertise to help out the over-scheduled Developers.

**Problem Definition**

Given:

* A milestone that is 12-weeks long or 60 work days
* *n* developers that need help on feature chunks where each has *k* days of work overflow
* *m* developers that can provide help on feature chunks where each has *j* days free
* Each of the under-scheduled developers have a set of developers that they can help (total equals to *E*)

Find:

Can the work be finished if the *m* under-scheduled developers help the *n* over-scheduled developers? If so, give the assignment lists.

**Solving this problem efficiently**

This problem can be solved efficiently using the network flow algorithm. The sources are the under-scheduled developers. The sinks are the over-scheduled developers. Each under-scheduled developer is connected to a set of over-scheduled developers that they can help through the edges. Each edge is assigned a capacity equal to the number of free days that the source has. There is a super source *s* that connects to all under-scheduled developers. Each edge is assigned a capacity equal to the number of free days of the under-scheduled developer. There is a super sink *t* that is connected to all the over-scheduled developers, and each edge is assigned a capacity equal to the number of days of work overflow for that over-scheduled developer.

After setting the network flow graph, find the max flow using the Ford-Fulkerson algorithm and if the max flow is equal to the sum of days overflow for all developers, then there is a valid assignment. This is only O(C*E*) in run time.

**Alternative problem without an efficient solution**

This is another problem that was investigated, but decided against it since it is NP-hard.

 Given:

* A milestone that is 12-weeks long or 60 work days
* *n* features chunks that are planned to be implemented for the milestone
* Each feature chunk has an estimated time in work days
* A total of *d* Developers (not all Developers need to be assigned work)
* Each Developer can only work on one feature chunk at any given time

Find:

Can the managers assign these feature chunks to the Developers within the time and resource constraints? If so, what’s the list of the feature chunks?

**­Possible Algorithms for the Problem­­**

How to solve this problem? This section goes through some of the algorithms and discusses whether they can be used to solve this problem.

Greedy Algorithm

This problem can be solved using the greedy algorithm. First sort the feature chunks in descending time estimation order. Then start with the first developer, continuously assign him the biggest feature chunk from the list that can fit in the milestone. Then start with the second developer. Continue until there are no more feature chunks left to assign. This algorithm runs in O(n­2) time where *n* is the number of feature crew chunks.

However, this algorithm only works when there are more Developers than minimum required. For example, if there are three jobs 7, 4, 4. And the first developer has 8 work-days left, then he’ll select 7 instead of 4, 4. And if the second developer only has 7 work-days left, then he can only select 4. Instead, if the first developer is assigned 4, 4, then the second developer can take 8.

Network Flow Algorithm

This problem cannot be solved using the network flow algorithm. If the feature chunks are the sources and the Developers are the sinks, then every feature chunk has to be connected with every developer, which does not work, since the assignment is one on one.

We can go around this restriction by doing one developer at a time. Use all free feature chunks as sources and one developer as the sink. The capacities are the time it takes to complete the feature chunk. Then after the first is done, go to the next developer until there are no free feature chunks. However, this method does not work either, since a feature chunk cannot be split up.

Dynamic Programming

This problem is similar to the pretty printing problem, which can be solved using dynamic programming in O(n2). Each feature chunk is a word and the time it takes is the word length. The 60-workday limitation is the max number of characters on a line. The *n* Developers are the minimum number of enters required. The only difference is that the words are in a certain order & the feature chunks are not. The feature chunks can be in any order since the all can be done in parallel, which means there are *n!* (greater than exponential growth) ways to arrange the feature crews. It is definitely not a good solution.

NP Hard

This problem is NP Hard as it is a representation of the load balancing problem, which is discussed in the textbook.

**Scheduling problems that are polynomial**

Here are some other schedule and resource allocation problems that are seen in our every day lives that can be solved efficiently using an algorithm from the class.

Knapsack problem - Given a set of conflicting meetings where each meeting has a weight denoting how important it is, find the set of meetings you can attend that maximizes the weight of all the meetings. (Dynamic programming)

Interval and lateness problem - Given a set of personal tasks where each task requires a certain amount of time and has a deadline, find an order to complete the tasks so that lateness is minimized.  (Greedy algorithm)

Airline flight schedule problem - Airline flight schedule Given origin-destination schedule demands and fleet size, is it possible to serve all schedule demands?   (Network flow problem)

Crew scheduling problem - Given a set of airline flights, find a feasible assignment of crew to work all flights.  Feasible assignments must take minimum rest times, maximum monthly flying times, days off, and maximum days away from base into consideration. (Linear programming)

**Scheduling problems that are NP**

* Job Shop Scheduling - Given a set of jobs each with a set of operations that must be performed on different machines in a particular order and a finite set of m machines that can handle one operation at a time, find a schedule that finishes jobs in the shortest amount of time.
* Multiple Interval scheduling - Given a set of jobs, each with a set of time intervals, find a schedule so that no two jobs have time overlaps.
* Scheduling with release times and deadlines -Given a set of jobs that must be run on a single machine with release time (time when job becomes available), deadline time (time job must be completed by), and duration time (amount of machine time job needs to complete), find a schedule where every job is completed by it’s deadline time.
* multiprocessor scheduling - Given a set J of jobs where each job requires an amount of time to complete and a number of processors, what is the minimum possible time required to schedule all jobs on the processors such that none overlap?

**References**

Hockbaum, Dorit S. “*The Scheduling Problem”* 1999 <http://riot.ieor.berkeley.edu/~vinhun/index.html>

Barnhart, Cynthia. *“Airline Scheduling Planning”* Spring 2003 <http://ocw.mit.edu/OcwWeb/Civil-and-Environmental-Engineering/1-206JAirline-Schedule-PlanningSpring2003/CourseHome/index.htm>