Methods to Learn Abstract Scheduling Models[*]

Student: Tom Carchrae
Supervisors: J. Christopher Beck and Eugene C. Freuder

Cork Constraint Computation Center, University College Cork, Ireland
t.carchrae@4c.ucc.ie

1 Introduction

In practice, most scheduling problems are an abstraction of the real problem being solved. For example, when you plan your day, you schedule the activities which are critical; that is you schedule the activities which are essential to the success of your day. So you may plan what time to leave the house to get to work, when to have meetings, how you share your vehicle with your spouse and so on. On the other hand, you probably do not consider the activities that are easy to arrange like brushing your teeth, going to the shops, making photocopies and other such tasks that can usually be accomplished whenever you have the time available. Often, if a schedule goes wrong, it is because a missed or underestimated activity had a significant impact on the schedule. We typically learn which activities are critical by experience and create an abstract scheduling problem including only these critical activities. Instead of scheduling the non-critical activities we estimate their effects in the abstract scheduling problem.

Most industrial scheduling problems are also an abstraction of the real problem. It is common to only schedule activities on bottleneck resources and let the other activities fall into place once these are scheduled. Deciding which resources are bottlenecks is currently done by experienced human experts who determine what needs to be considered to create an accurate and high quality schedule. Unfortunately this process is both time consuming and error prone. We want to automate the process of creating a good quality abstract scheduling model.

In this paper we describe an abstraction method for scheduling problems. Abstraction reduces the complexity of optimizing the entire problem by solving it in stages. We create an abstract model of a scheduling problem using a subset of the activities and approximate the remaining activities. We then search for a good solution to the abstract model, and extend this solution to a full solution to the entire problem. While the approach reduces complexity there is a risk that the abstract solution will not produce a good full solution.

This paper is organized as follows. First we describe the abstraction process in Section 2. In Section 3 we introduce methods which select abstract models with the objective of determining a good abstract model whose solutions can

[*] This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075, Irish Research Council for Science, Engineering, and Technology under Grant SC/2003/82, and ILOG, SA.

be extended to good full solutions. We conclude in Section 4 by putting our abstraction work in the context of solving real world scheduling problems.

2 Abstraction Process

As we tackle larger problems, optimization methods suffer from scaling problems. The motivation for using an abstraction method is to reduce the complexity by solving the problem in smaller pieces. Scheduling can be seen as trying to determine a sequence of activities for each resource[2]. The complexity of scheduling using this approach is $O(A!^R)$ where A is the maximum number of activities which require a resource, and R is the number of resources.[1] Solving the problem in two stages will reduce the complexity, for example scheduling half of the activities in an abstract model, leaving the other half to be scheduled later, will give a complexity of $O(\frac{A}{2}!^R + \frac{A}{2}!^R)$. In this section we describe how we create an abstract model, solve it, and use it to produce a solution to the entire problem.

2.1 Creating the Abstract Model

To create the abstract model, we include only a subset of activities and remove all other activities. Each activity belongs to a job, which is a sequence of activities, connected by precedence constraints. When we remove an activity, we post a new precedence constraint between the preceding and succeeding activities of the removed activity. This precedence has a minimum gap equal to the duration of the removed activity, $gap = duration(a)$. That is, for an activity a, we post a precedence constraint between the previous activity in the job $prev(a)$ and the next activity in the job, $next(a)$ of the form

$$endtime(prev(a)) \leq gap + starttime(next(a)) \tag{1}$$

which requires that activity $next(a)$ starts at least gap time units after the end of activity $prev(a)$. While we do not compute the time it takes for a to be scheduled, this reserves time for a to be processed in the full solution. Note that for the approximation to be correct, it assumes that we will be able to schedule activity a immediately on the resource it requires. Otherwise, the amount of time between $prev(a)$ and $next(a)$ will be greater than our approximation.

There are however some special cases. If either $prev(a)$ or $next(a)$ have also been removed from the abstracted model then we must reserve a larger gap than $duration(a)$. To do this, we replace $gap = duration(a)$ with the sum of durations from the set I of removed activities which are in-between the pair of activities that remain in that job, $gap = \sum_{a \in I} duration(a)$.

There is one more case we must handle. If we remove the activity at the beginning or end of a job, we are unable to post the precedence constraint as we are lacking the $prev(a)$ or $next(a)$ activities. For the beginning of the job, we can simply replace the preceding activity $prev(a)$ with the earliest start time of

[1] $A!$ is the number of possible sequences on a resource

the first activity. To handle missing activities at the end of the job, we replace $next(a)$ with the makespan which represents the latest time in the schedule. This ensures that any solution to the abstract model will reserve at least *gap* time units at the beginning and end of the job to allow room for the removed activities to be scheduled.

2.2 Extending an Abstract Solution to a Full Solution

Once we have a good solution to the abstract model we can extend it into a solution to the entire problem. The approach we have used is to fix the sequence of activities on each resource in the abstract solution. We do this by adding precedence constraints to the original problem between each consecutive pair of activities in the abstract solution sequence. This effectively fixes these activities so that no search is required to schedule them. Since they are precedence constraints it allows some movement in the full solution.

The reader may be concerned that this approach could lead to insoluble problems. However, this is not the case. The addition of precedence constraints to represent the removed activities ensures that any solution to the abstract model will also be a solution to the entire problem. We have not removed any constraints, merely approximated the time some activities will take.

3 Choosing an Abstract Model

The choice of activities considered in the abstract model will determine whether good abstract solutions correspond to good full solutions. The ideal is that the optimal solution to the abstract problem yields an optimal solution to the entire problem. In practice, we may not have sufficient time to find the optimal solution, in which case we are interested in finding the best solution to the entire problem within a given time limit.

The approach we use is to select activities using probabilities which determine the likelihood of an activity being chosen for the abstract model. This provides a mechanism to learn which activities should be in the abstract model. If we set the probabilities of each activity to 1, then the abstract model will contain all of the activities. This ‘abstract’ model is guaranteed to produce the optimal solution, however, it is also the model with the worst complexity. We want to discover a model which is a subset of the activities but which produces good, if not optimal, solutions to the full problem in less time due to lower complexity. It may be the case that the best approach is to schedule all of the activities together, i.e. all probabilities are set to 1, however we believe there are many problem instances where choosing a subset of activities will yield better performance.

We now describe methods which try to learn a good abstract model. We begin by describing several ways to compute probabilities and then describe two methods for exploring different abstractions. The first exploration method runs several trials of different abstract models. After all the trials are completed, we choose the abstract model which produced the best solution. The second exploration method iteratively selects a model and performs the abstraction process. After each iteration, we adjust the probabilities for each activity. In future iterations, the activities which appear in better performing models will have a higher probability of being selected than activities which appear in poorly performing models.

3.1 Computing Probabilities

Our mechanism for choosing an abstract model is to use probabilities to select the activities. We know that if all probabilities are set to 1, we are guaranteed to find the optimal solution but it will also take the longest time to find it. We wish to determine the probabilities for each activity so that we can remove activities from the abstract model without sacrificing the quality of solutions found. In effect, we wish to search the space of abstract models. However, the number of possible abstract models is typically large and evaluation of an abstract model is time consuming. Here we describe several methods for computing the probabilities which influence our search for a good abstract model.

Problem Features One approach is to use features of the problem instance to assign probabilities. For example, we can compute the demand of each resource by summing the total amount of time required by activities of that resource. Activities which require a resource with a high demand will get a higher probability of being selected. We can also increase the probability of activities with a longer duration as these are likely to have a greater impact on solution quality.

Solution Features Another idea is to adjust probabilities based on features of solutions to the entire problem. These features may give some insight into the parts of the solution which could be improved. Examples are adjusting probabilities based on the *slack* in the start time of an activity, or the activities in the solution which cause the most delay.

Algorithm Performance Features We can also use algorithm performance as a feature to determine probabilities. By simply looking at the improvement in solution quality, we can determine if an abstraction was useful, and give activities which appear in that abstraction higher probabilities. This has the benefit of being independent of problem and solution features, making it more general in use[1].

Randomization Randomly assigning probabilities is a simple approach that may discover useful abstractions. It can be combined with other features in order to diversify the search for good abstract models.

3.2 Trials of Abstractions

In trials of abstractions, we choose a subset of activities using probabilities, search for good solutions to the abstract model, and then compute a full solution using the best abstract solution. These trials are repeated several times using different probabilities. The best abstract model found can then be allowed to run for a longer time. Alternatively, we can allocate all of the available run-time during the trials and simply take the best result found. As mentioned above, there are several choices for setting the probabilities. We can randomly assign probabilities, with the hope that we find a good abstraction. We can use problem features to bias the probabilities towards the suspected hard parts of the problem. Finally, we can include a run where all probabilities are set to 1, which is equivalent to solving the problem without abstraction as all activities will be selected. Trials allow us to try a diverse set of abstract models.

3.3 Iterative Abstractions

In contrast to trials, we can iteratively adjust probabilities over the course of several runs. We choose an abstract model using probabilities, solve it, and compute a full solution. After each iteration, we update the probabilities of the activities. The intuition is that as we try different abstractions, we can learn which activities should be included in the abstract model. Unlike the trials method, in this approach, we maintain a single set of probabilities and use our experience in solving the problem to bias future abstract model choices. As we are solving the same problem instance using different abstractions, we can use features of the algorithm performance and solutions to update the probabilities based on previous runs.

4 Conclusion

We have presented preliminary work in automating the task of modelling in order to reduce complexity when solving scheduling problems. We believe this is an important step in making optimization technology easier to use. At the moment, scheduling problems are modelled by human experts who analyze the problem and then decide how to make a tradeoff in terms of solution quality and time spent in searching for solutions. By automating this process, we not only make it simpler to apply but also allow the system to adapt to changes. For example, a factory may have different types of orders throughout the year. As the order mix changes, the bottleneck resources in the factory may also change. A system which analyzes the abstract scheduling models will be able to adapt to ensure that it finds optimal solutions to the bottleneck resources as they shift around. By removing the reliance on human experts, the system can adapt faster and with less cost. Experiments are underway on both academic and industrial problems in order determine which methods are viable.

References

1. T. Carchrae and J. C. Beck. Low knowledge algorithm control. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI04)*, 2004.
2. W. P. M. Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach.* PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.