Clustering and scheduling maintenance tasks over time
Per Kreuger
2008-04-29
SICS Technical Report T2008:09

Abstract
We report results on a maintenance scheduling problem. The problem consists of allocating maintenance task instances to and scheduling the performances of a suitable number of maintenance packages. The number of maintenance packages is not fixed, nor is, in general, the dates or durations of their performances. A constraint programming (CP) model and solver for the problem is presented together with preliminary computational results.

Keywords
Maintenance scheduling, resource allocation, task clustering

This work was funded by the Butler project run at SICS 2005–2007.

ISSN 1100-3154
ISRN: SICS-T–2008/09–SE
1 Introduction

This report describes a modelling experiment performed to investigate the feasibility of using constraint programming techniques to solve the problem of allocating maintenance task instances to and scheduling the performances of a suitable number of maintenance packages. The number of maintenance packages is not fixed, nor is, in general, the dates or durations of their performances but we may want to avoid scheduling the performance of maintenance packages so that they overlap in time.

This type of problem occurs in many industrial maintenance problems where individual components belonging to a piece of equipment has to be serviced at regular intervals and where the whole piece of equipment has to be taken out of normal service during maintenance occasions. We do not in this work address the problem of deciding the ideal separation between instances of the tasks but regard these as fixed and furthermore restrict the problem as follows:

Individual instances of tasks have a fixed duration and are performed at more or less regular intervals. Several such tasks with varying characteristic intervals exist in general but tasks from several classes can and usually are allocated to the same maintenance package. We restrict our analysis to the case where individual task instances may be performed in parallel during a the performance of a maintenance package so that its duration can be assumed to have the same duration as the longest task instance allocated to it.

The regularity of the instances particular task is not necessarily completely fixed. An individual instances may e.g. be performed before the full separation interval is complete. The instance may also be delayed with respect to the ideal separation within its task but costs are associated with both of these cases, usually a higher cost for the second type of situation. Cost are also associated with each maintenance package, and there may be (typically yearly) periods when scheduling maintenance occasions are more (or less) costly.

This means that for a particular allocation of task instances to maintenance packages and a particular schedule of package performances, there are costs associated with both the scheduled occasions and with each tasks in which the sequence of instances deviate more or less from an ideal (usually equally spaced) schedule.

We assume that in general, each task instance has to be allocated to exactly one maintenance package and that the minimum number of task instances allocated to a package is one. This means that the number of packages may vary from one to up to the number of individual task instances. There is however, a trade-off between performing the instances of a task at ideal separation intervals which may require to many individual packages and of minimising the number of package performances.
2 Problem model and parametrisation

Let the number of task instances be \( n \) and let each task be characterised by the duration of its instances and the ideal temporal separation between application of its instances. The instances themselves are explicitly represented and characterised by an unique index \( \leq n \) and earliest and latest start date.

To encode the allocation problem we associate with each instance, a package variable. When bound, this variable denotes the lowest index of any task instance allocated to the same package.

To solve the problem, we post precedence constraints on instances of each task and constrain the package variable of each instance to be less than or equal to the index of the instance itself. Then, for each pair of instances, equate the equality of their package indexes with that of their start times.

Identify each package start time with that of the instances allocated to it, let the duration of the task be their maximum and optionally post scheduling constraints on the packages.

To model the cost of a task, sum over the individual costs of the deviation of the scheduled interval between two consecutive instances of the task and that of the ideal separation.

Count furthermore the sum of the cost of scheduling each package by summing over each pair of instances allocated to it and count costs only for the instances who's index coincide with their package allocation which will be true for only one instance per package.

A redundant condition that could possibly improve propagation and speed up search would be to associate an additional variable denoting a sequence within the package. These variables could be constrained to be pairwise different which is an other way to formulate the partitioning condition.

3 Computational results

3.1 Problem instance

The basic model outlined above was assessed by using it to encode an example problem consisting of 8 tasks with ideal separations between 4 and 52 weeks and enough instances to cover a total of 5 years. The total number of task instances considered were 193.

The task duration was set to one week for tasks with separation less than 16 weeks and varying from 3 to 4 weeks for the those with longer instance separations. The start time window was set to be equal to two times the ideal separation. The effect of this type of formulation is to allow each task instance to start at the earliest at the ideal moment of its predecessor and at the latest at the ideal start of its successor. The cost function of performing an instances at non ideal separations was scaled so that delays was penalised three time more than making it early.

The cost of the package assignment is first of all determined by the number
of individual packages used and secondly by the package schedule using different weights of packages scheduled during certain times of the year. The units and relative weights of the different cost components is probably not that important at this stage of the work, but the task cost is given in weeks (counting one week delay as three) and the package cost either 10/package or 110/package depending on the time of year.

Although this is a completely artificial example it appears to correspond fairly well to some real instances we have also encountered in other projects.

3.2 Search heuristics

A number of search heuristics were investigated of which only one that appears to be most robust will be described here.

To find a feasible solution, first enumerate the package variables of the task instances. This will fix the assignment of task instances to packages and constrain the start times of task instances assigned to the same package to coincide. For each such partial solution, find the the optimal assignment of the instance separations of each individual task. Finally decide an arbitrary assignment of any start times that have not already been fixed. At this point, the cost was always determined for the particular case we considered, in more complex cases it may make sense to use an optimising search at this step as well.

Several minor modifications of this general scheme were evaluated, using e.g. step/bisect and or using only the task cost for optimising the instance separations.

To optimise the total cost, do a branch and bound search over the alternative assignments of task instances to packages. We used two different timeouts for the two optimising steps in the search, one for the inner loop, minimising the instance separations and an other for the outer loop that searches over the assignments. Varying these parameters gave slightly different results, e.g allowing a maximum of 5 seconds for each gave a solution with 65 separate packages, a task cost of 395 and a package schedule cost of 5400, while allowing a maximum 20 seconds for each iteration of the inner loop and 60 for the outer produced the same number of packages but with a slightly less costly schedule (5300) but an increased task cost (443).

4 Summary and conclusion

We have formulated a combined allocation and scheduling model for a typical industrial maintenance problem. The model has been implemented and evaluated on an artificial but fairly representative example and a number of different search heuristics has been tried out and the most promising one presented. The solver seems to be rather robust w.r.t. the number maintenance packages chosen, which was rather surprising, given the very general formulation. This may however be an an artifact of the particular example chosen and the way task instance start time windows were set.
It seems reasonable to expect that the model could be adapted to more realistic cases even though the scalability of the approach may have to be reevaluated for other cost functions and larger cases. Restricting the number of packages could be an approach to better scalability.