Enhancing predictability of schedules by task grouping

Michel Wilson\textsuperscript{a} Cees Witteveen\textsuperscript{a} Bob Huisman\textsuperscript{b}

\textsuperscript{a}Delft University of Technology, Dept of Software Technology
\textsuperscript{b}Nedtrain, Fleet Services – Maintenance Development

In a scheduling problem one has to determine time slots for a set of tasks (activities) to be completed, subject to time and resource constraints. An important problem in applying scheduling methods is the \textit{predictability} of scheduling solutions: when encountering task execution delays, one would want to minimize recomputations and changes of the currently constructed schedule as much as possible. Two possible approaches to ensure predictability can be distinguished [3]: \textit{on-line} reactive approaches and \textit{off-line} or pro-active approaches.

Our paper pursues the latter approach, by coming up with a set \(S\) of schedules instead of just one schedule, such that, during execution time, the currently chosen schedule \(\sigma \in S\) can be replaced by another one \(\sigma' \in S\) such that it meets the changed problem constraints without affecting the objectives. The method is based on a transformation of the original scheduling problem, by \textit{grouping} some tasks together into a new composite task. By grouping, the task ordering within a group is left unspecified in creating a schedule. Only at execution time a specific, most suitable, ordering of the original tasks making up the composite tasks is decided. This method increases the make span of the schedule, but as a result the predictability in the face of uncertainty during execution is increased as well.

\section{Task grouping}

The procedure of task grouping is based on the Precedence Constraint Posting approach [1] to solving the Resource Constrained Project Scheduling Problem (RCPSP). The basic precedence constraint procedure iterates over four steps: 1) compute the resource usage profile over time, using an earliest start time solution, and select a peak, i.e., a point at which the use of a resource exceeds its capacity, 2) select two (partially) concurrent tasks contributing to the selected peak, 3) decide on a sequential order for these two tasks, and add a precedence constraint to the problem enforcing this order, and finally 4) calculate updated earliest start times to create a new schedule.

Our task grouping approach involves a simple modification to the procedure above, yielding more execution-time flexibility: instead of deciding on a \textit{fixed ordering} of the two selected tasks in step two, we remove the two tasks from the problem, and replace them by a grouped task, which reserves enough resources to enable \textit{any execution order} of the two tasks. The grouped task can be treated as any other task, in particular, it is possible for such a task to participate in another grouping operation, such that a group task can represent a reservation of resources for any number of tasks, which can be executed in any (sequential) ordering.

To control task grouping, the heuristic used to choose the constraint direction in regular constraint posting is used. Normally, the constraint between two tasks is posted in the direction preserving the largest amount of slack between the two tasks. In our algorithm, the two tasks are grouped if the \textit{difference} between the slack of both orderings is below a threshold parameter \(\gamma\).

\section{Experimental results}

As experiment, simulated executions were performed of schedules for the well-known benchmark instances from PSPLIB [2]. In each execution, delays were inserted in some of the tasks, and to estimate the

\textsuperscript{1}This is an abstract of our paper accepted for the 2012 European Conference on Artificial Intelligence (ECAI)
predictability of the schedule it was determined how many of the tasks were completed at their expected time, according to the original schedule. Additionally, the effect on the makespan of the schedules was measured.

A makespan increase is expected due to the worst case envelope of a group task, for both start times and resource usage. The experiments indeed show an increase, from 7.2% for instances with 30 tasks to 16.5% for instances with 120 tasks can be seen for \( \gamma = 2 \). For \( \gamma = 8 \) the increase is bigger, from 19.8% to 43.1%.

For the predictability, two series of tests were performed: one in which the number of delayed tasks was varied, and one in which the amount of delay in the tasks was varied. The results are presented in Figures 1a and 1b. The number of tasks which complete in time drops rapidly when increasing the number of delayed tasks, which is in line with expectations. The performance of grouping increases for larger values of \( \gamma \); the cause is the larger number of groups present in the solution. If the number of delays gets large however, the gains of grouping diminish.

The amount of delay per task has a much lower effect on the predictability: longer delays do not cause additional tasks to be completed late. This shows that the schedules do not contain a lot of slack: a small delay is already enough to propagate to all tasks following the delayed tasks. Here, it can be seen that our grouping method performs well for small amounts of delay. This is in line with expectations: if a task is delayed by a large amount, a reordering of the tasks succeeding it can rarely absorb the delay.

3 Conclusions

In this paper, a novel way to create predictable schedules is presented, using grouped tasks to enable some tasks to be re-ordered during execution time, to prevent the propagation of a delay. Additionally, the executing agent gains some autonomy during execution.

Tests using simulated execution show that schedules containing grouped tasks are indeed more predictable: more tasks are completed at their scheduled time when delays are inserted. The method works best for small delays, but it still works somewhat even if a large amount of the tasks incurs a small delay.

References

