# Production and Transportation Planning Software User Evaluation Report Deliverable D27, ALCOM-FT Project

Peter Lennartz

Institute of Information and Computing Sciences
Utrecht University
Padualaan 14, 3584 CH Utrecht
The Netherlands
peterl@cs.uu.nl

**Abstract.** In this report we evaluate the software presented in deliverable D19 and present some improvements. The purpose of the software is to solve instances of the no-wait job shop problem. Right now it can solve instances of size up to 10 to 15 jobs that belong to a special class. With the current improvements it can solve all instances of the no-wait job shop problem with 10 to 15 jobs, and we expect to improve this even further.

## 1 Introduction

The no-wait job shop scheduling problem (NWJSP) can be defined as follows. Given a set of tasks (called the operations) each task is to be executed on a specific machine, which takes a given amount of time (the processing time); each machine can handle at most one operation at a time and is continuously available from time zero onwards. Furthermore we are given a set of so called no-wait constraints, where each constraint decrees that a given operation has to be started *exactly* $\Delta$ time units after another, given operation has finished; this number $\Delta$ can be negative. By the term *job* we denote a maximal set of operations connected by these so called no-wait constraints. The objective is to find the shortest schedule; i.e., an assignment of starting times to the operations such that all the constraints are fulfilled (i.e., the schedule is valid) and such that the point in time at which the last operation finishes processing (the so called makespan) is as small as possible. The NWJSP is used to model the production problem in a pharmaceutical industry (see deliverable D9, part A).

The program delivered in ALCOM-FT deliverable D19 has been designed to solve such instances. The algorithm was built on top of a TSP solver using the similarity between a special class of the NWJSP and the asymmetric traveling salesman problem (see [1]). We refer the reader to the delivery report of project D19 for a brief description of the algorithm.

## 2 User Evaluation and Improvements

### 2.1 Testing Results

When testing the algorithm, we found out that it was only well suited to solve the instances that were close to the special case; in other cases the algorithm showed bad behavior. To overcome this bad behavior we have changed the underlying model: instead of modeling the problem as an asymmetric traveling salesman problem, we now formulate it as a binary integer linear program of pseudo-polynomial size using a time indexed formulation with variables $x_{jt}$ that indicate whether job $j$ is started at time $t$ (see for example [2] for details). This time indexed formulation allows us not only to ask for the shortest possible, valid schedule but also to ask if there exists a schedule with length no more than a given upper bound. In this way we can find quickly a schedule of length ten percent above optimum by only applying pure branch and bound algorithm in which standard cuts are incorporated. Furthermore we have implemented heuristics to speed up the process of finding valid solutions, and we have implemented a new branching strategy. Instead of branching on single variables in the time indexed formulation (which were set to zero or one respectively) we now branch by splitting up the execution intervals; this is done by assuming that the starting time will be before or after one point in time in one branch respectively. This gave us a speedup of 30 to 50 percent on our test instances. By assuming that the starting time of a job lies within a certain interval we are also able to compute the consequences for the starting times of all the other jobs; i.e., do some propagation on the starting times. For example, by assuming that a job does not start before a certain point in time we can ask (and answer) the question of where other jobs cannot start because this job is being processed. This gave a speedup of another 10 to 25 percent.

### 2.2 Further improvements

As described above, we combine constraint satisfaction with integer programming by using it as a sort of preprocessing step to minimize the search space.

We are right now busy with implementing lower bounds on the makespan that follow from the single-machine relaxation (see for example [3]); we expect that this will make the constraint satisfaction part much more efficient. Besides this we are working on a refinement of the branching strategy such that we can reuse as much information as possible. For example, if a heuristic gives a negative result, it might be worth to analyze why it failed and use this information to branch deeper into the search tree.

## 3 Conclusion

The algorithm (as it is given in the software prototype in deliverable D19) could handle instances of 10 to 15 jobs of the special type mentioned above. Right now it can handle arbitrary instances of size up to 10 to 15 jobs. Both measurements were made on a Pentium IV 1700. Furthermore, the underlying time indexed formulation can be easily extended to the case that multiple machines of the same type are present, which is the type of problem that we have real-life instances of (see deliverable D28).

## 4 Software

We will publish the new, refined software on the Web when it has received enough development and testing. Development snapshots are of course available. Write an e-mail to Peter Lennartz (peterl@cs.uu.nl) and a copy of the sources will be sent to you.

## References

1. P. Lennartz, H. Hoogeveen (2004). The Correspondence Between the No-Wait Job Shop Problem and the Traveling Salesman Problem, to appear.

2. J.M. van den Akker (1994). LP-based solution methods for single-machine scheduling problems, PhD Thesis, Eindhoven University of Technology.
3. A.M.G. Vandevelde, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra (2004). Lower bounds for the head-body-tail problem on parallel machines: a computational study of the multiprocessor flow shop, to appear in INFORMS Journal on Computing.