13th International Research/Expert Conference
”Trends in the Development of Machinery and Associated Technology”
TMT 2009, Hammamet, Tunisia, 16-21 October 2009

NOVEL METHODS FOR FINDING (GENERALIZED) JOINABLE SCHEDULES

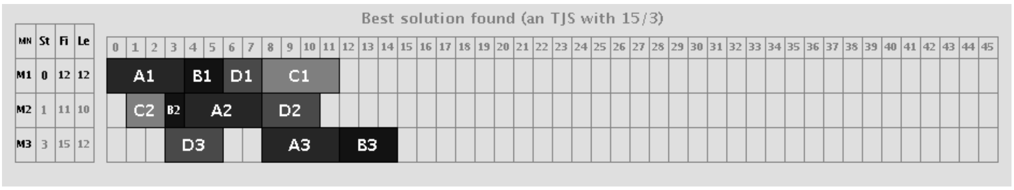
Elemér K. Nagy
István Loványi PhD
Budapest University of Technology and Economics
Magyar tudósok körútja 2., 1117 Budapest,
Hungary

ABSTRACT
This paper is about novel methods for finding Joinable Schedules (JS) as defined by J. Somló and extended by E. K. Nagy. After a short introduction to JSs and their merits,generalization of JSs are introduced. This article deals with methods aimed to improve the efficiency of finding JSs. After introducing the methods and the underlying principles, the methods are analyzed and their results on the generated example set is visualized.
Keywords: Joinable Schedules, Joinable Schedule Approach, Efficiency, Scheduling, SISONEK

1. INTRODUCTION
Flexible manufacturing systems (FMS) can be analyzed as job-shop scheduling problems (JSP). One approach to solve these problems is the joinable schedule approach (JSA), which, if a joinable schedule (JS) is found, can be evaluated in real-time to provide a solution with a known total processing time. Although this solution might not be optimal, the bottleneck machine's load/idle ratio is known and thus the goodness of the solution can be calculated in real-time.

2. JOINABLE SCHEDULES
Joinable schedules (JS) are defined and evaluated in [1], analyzed and extended in [2] and are only briefly introduced here. The JSA is a special case of lot streaming (LS) schedules for JSPs, and although the basic principle of LS is known for decades, practical considerations and the increasing complexity restricts the spread of LS. According to [1], "Many researchers studied ... Lot Streaming ... focus on the Flow Shop System (FSS) ..." - like [3]. Joinable schedules have the following criteria:
- The bottleneck machine is loaded without idle times (and processing starts immediately)
- The Gantt chart is not overlapping with itself (as evaluated later)

The usefulness of the JSA is limited by two factors: A) there are problems with no Joinable Schedule Solution (JSS), and B) finding a JS for a given problem is hard (NP or maybe even NP-hard) - although this is true to static schedulings, according to [4]: "static scheduling problems are usually NP hard." *Figure 1a* and *Figure 1b* demonstrates the basic idea of the JSA, without considering the set-up times of the tasks.


Best solution found (an TJS with 15/3)

MN	St	Fi	Le
M1	0	12	12
M2	1	11	10
M3	3	15	12

Figure 1a. A Joinable Schedule

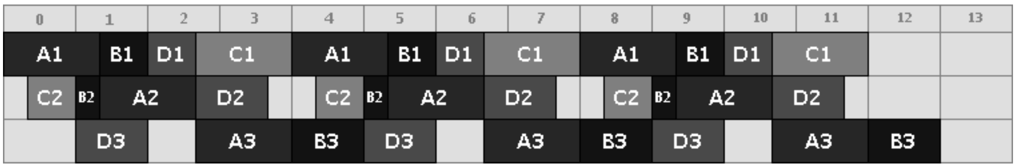
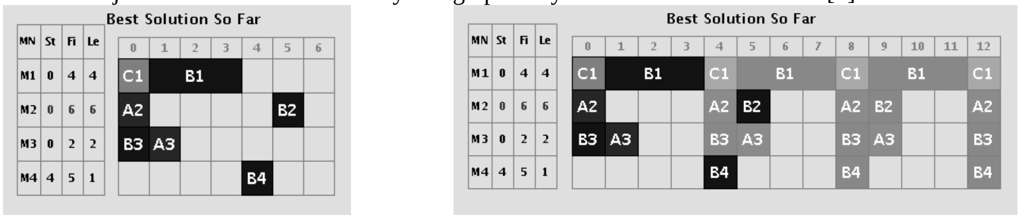

0 1 2 3 4 5 6 7 8 9 10 11 12 13
A1 B1 D1 C1 A1 B1 D1 C1 A1 B1 D1 C1
C2 B2 A2 D2 C2 B2 A2 D2 C2 B2 A2 D2
D3 A3 B3 D3 A3 B3 D3 A3 B3

Figure 1b. A Joinable Schedule joined W=3 times

With zero set-up times, W=infinity is the optimal solution, which is impossible in practice. A variable named SoSUT (Sum of Set-Up Times on the bottleneck machine) is used to calculate the proper W for the given SoSUT and load times - details are available in [1] and [2].

3. THE NON-OVERLAPPING CRITERIA AND THE JOINABILITY TEST

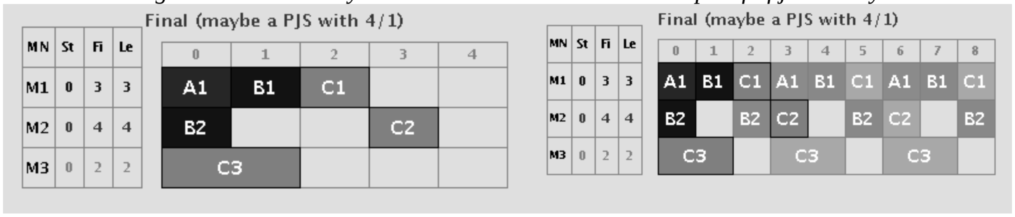
As shown in [2], there are JSs that do not pass the joinability test defined in [1]. In this article, three non-trivial joinable schedules are shortly and graphically introduced - details are in [2].



MN	St	Fi	Le
M1	0	4	4
M2	0	6	6
M3	0	2	2
M4	4	5	1

MN	St	Fi	Le
M1	0	4	4
M2	0	6	6
M3	0	2	2
M4	4	5	1

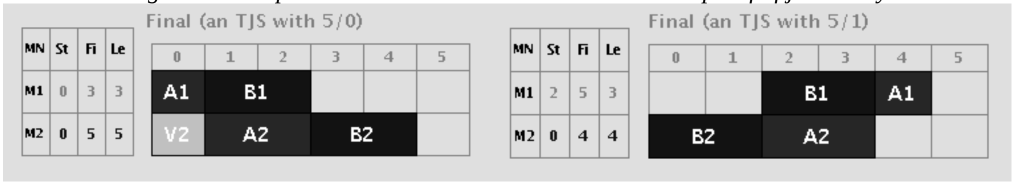
Figure 2a. a Curiously Joinable Schedule and the visual proof of joinability



MN	St	Fi	Le
M1	0	3	3
M2	0	4	4
M3	0	2	2

MN	St	Fi	Le
M1	0	3	3
M2	0	4	4
M3	0	2	2

Figure 2b. a Pipelineable Joinable Schedule with a visual proof of joinability



MN	St	Fi	Le
M1	0	3	3
M2	0	5	5

MN	St	Fi	Le
M1	2	5	3
M2	0	4	4

Figure 2c. a Virtually Joinable Schedule with a virtual task V and a JS for a similar problem

4. EFFICIENCY OF FINDING JOINABLE SCHEDULES

No matter how great is a scheduling algorithm in theory, there is always a practical question in the background: What is the maximum problem size that can be efficiently solved with the method? To have some kind of unit, a simple benchmarking program was written that deterministically generates a predefined number of problems, runs the predefined solver against all of them and then sums up the results. The solver used is the JS generation algorithm described in [2]. There were two runs, both on the same software (Debian Etch, OpenJDK 64-Bit 1.6.0_0-b11) and hardware platform (Sun X4200 with two Opteron 2220 CPUs and 8 GB RAM,). In the first run, all three JSVT tests (as described in [2]) were disabled. In the second run, all three JSVT tests were allowed. From the viewpoint of efficiency, the most important number is the number of timeouts - the number of times when the solver was not able to find a JS for the given problem and was unable to prove that the problem has no (trivial) JS solution. Clearly, the more timeouts happen, the less effective the algorithm is. Of course, with the same time limit, a faster algorithm might find better solutions as it is able to check more possible sequences. Yet, unnecessary tests (those that consume CPU or RAM but does not invalidate possible solutions) increase the time needed to check a sequence, therefore making the algorithm less effective. Of course, these results do not provide a statistical analysis, they are just an indication of performance, as the lengths of the processing times were generated by a uniform distribution and many factors define the solvability of the generated cases.

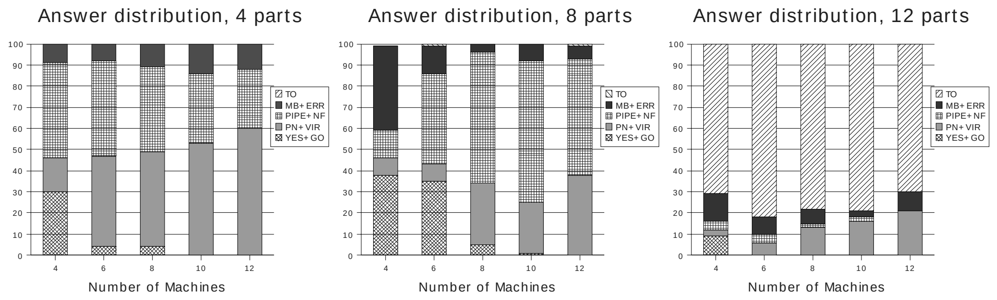


Figure 3a. Results with all JSVT tests turned off

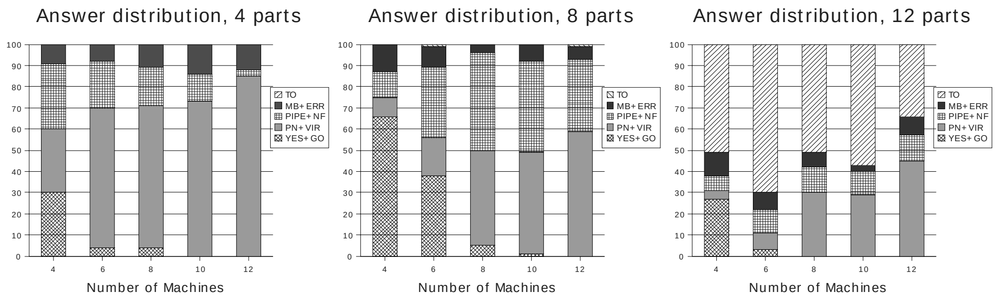


Figure 3b. Results with all JSVT tests turned on

5. ANALYSIS

There are many things we can deduce from the numbers if we decipher the legend, as described in [2]. First, the number of usable answers increased in the 12-part cases from 24 to 47 percent - so we can solve twice as many "large" problems. In the case of 8 parts, the average number of "JS found or Global Optimum found" increased from 16 to 22 percent. In the case of 4 parts, an average of 20 percent of cases were transformed from "not found or PJS" to "proved not or Virtual". The average time needed for the runs for 8 parts and 8 machines (supporting data not included) was reduced from 12.2 seconds to 7.46 seconds.

6. CONCLUSION

In the opinion of the author, the JSVT test are very practical and increase the performance of JS generation considerably, thus increasing the number and quality of JSs without major drawbacks.

7. REPRODUCTION OF THE DATA

To reproduce the data sets, download ***sisonek-0.0.3.21_docsless.jar*** from ***http://scpfw.sf.net/*** and run the Benchmarker class with the argument "***100 --no-jsvt1 --no-jsvt2 --no-jsvt3***", and then with the arguments "***100***". The results depend on the hardware and software environment.

8. REFERENCES

[1] Kodeekha, E.; Somlo, J., Optimal Lot Streaming for FMS Scheduling of Flow-Shop Systems, *International Conference on Intelligent Engineering Systems,* 2008. INES 2008., 25-29 Feb. 2008, p. 53-58, DOI: 10.1109/INES.2008.4481269

[2] Elemér K. Nagy, *Novel methods to neutralize loops and equal-effect action sequences in resource-action-constraint models used in scheduling*, PhD thesis, 2009

[3] Subhash C. Sarin, Puneet Jaiprakash, *Flow Shop Lot Streaming*, Springer, 2007

[4] Jean-Marie Proth, Scheduling: New trends in industrial environment, *Annual reviews in control*, vol. 31 (2007), pp. 157-166