INTRODUCTION TO JOINABLE SCHEDULES

Elemér Károly Nagy

1. SUMMARY

This article is an introduction to Joinable Schedules, which are a class of schedules that can be successfully applied to a number of job-shop problems. Not all job-shop problems are solvable with Joinable Schedules – there are problem instances for which the non-existence of Joinable Schedule solutions is proved. If a Joinable Schedule exists for a given problem, it might be still difficult to find. On the other hand, Joinable Schedules are both interesting and have some solid advantages. For some problem instances, Joinable Schedules are easy to find, are optimal and their optimality can be formally proved. For some other problem instances, a Joinable Schedule is easy to find and the goodness of the solution can be quickly and explicitly calculated, compared to the probably non-existing theoretically optimum solution. There are deterministic heuristic methods to find Joinable Schedules. Also, for some problem instances, where the non-existence of Joinable Schedules are proved, there are mathematically equivalent, and very similar schedules to Joinable Schedules, like the Pipelinable Joinable Schedules.

2. THE IDEA BEHIND JOINABLE SCHEDULES

Joinable Schedules are introduced in [1], for example, and are evaluated in [2], the Author's PhD thesis, which also contains many details omitted in this article – therefore Joinable Schedules are introduced only briefly and informally here.

Let us have a problem modeled as a Job Shop scheduling problem, with J machines and I parts. Amongst the J machines, there is one or more, so-called bottleneck machines. There is no machine that has more work than a bottleneck machine. Thus, we can see that the production run surely cannot take less time that the time needed by a bottleneck machine – therefore, we have a minimum time needed for the production run – but probably there is no scheduling that can do the processing in this time. However, if we can find a scheduling that, for example, takes 2% more time for the production run than this minimum time, we can be sure that this schedule is quite close to the optimum, even though we cannot be sure that any better schedule exists unless we find one.

Joinable Schedules are a clever trick probably inspired by the above observation. We narrow down the possible schedules by requesting the bottleneck machine to start immediately and to work without interruption[1]. With these restrictions, we have two phases of operation. The first one is the heavy work phase – the bottleneck machine is constantly working, and all other machines are either preparing parts for the bottleneck machine or finishing parts processed by the bottleneck machine. The second phase is the wrap-up phase – the bottleneck machine finished processing and is idle, and all other machines are finishing up parts already processed by the bottleneck machine. An example for such a schedule can be seen in Figure 1 – B3 is processed in the wrap-up phase.

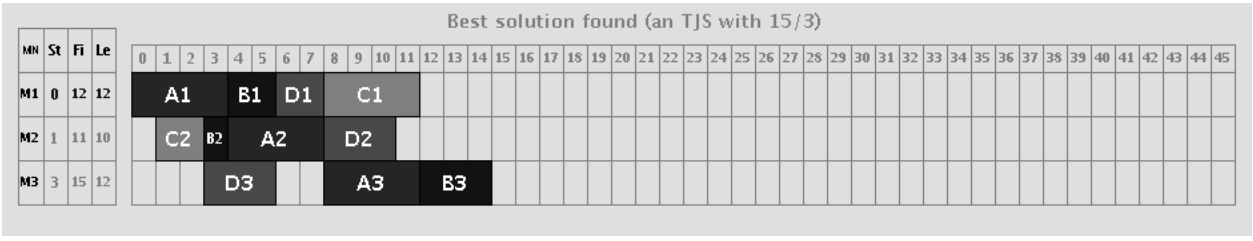


Figure 1.
A schedule for a Job Shop problem with a bottleneck machine M1

[1] We will see later that these restrictions can be relaxed.

3. NOTES ON THE FIGURES

The figures in this article are screenshots from the SISONEK package, which can be freely downloaded from http://sourceforge.net/projects/scpfw/ and is open source. Figure 2, Figure 3b, Figure 4b, Figure 5b, Figure 6b are post-processed with GIMP to present the underlying ideas more clearly. On all figures, the horizontal axis represents the time, with the time units displayed on the top. The colors represent the parts – for example, the first product, PA is always red, and is needed to precessed by machines M1, M2, M3 on Figure 1, resulting in processing phases A1, A2 and A3. The machines are displayed on the left of figures – the bottleneck machines are marked in blue, machines that fit into the given Joinable Schedule candidate are marked in green and machines that do not fit in the Joinable Schedule candidate are marked in red. The three numerical columns next to the machine names are **St**art time, **Fi**nish time and **Le**ngth of processing, in order. The capital on the top of the figure is the software's evaluation of the schedule.

4. A JOINABLE SCHEDULE

As seen on Figure 1, we have a schedule that is workable and have a wrap-up phase of three time units long. If we reduce the wrap-up phase, we will improve this schedule. There are several CPU intensive methods to find better schedules, like automated ways to swap the processing of B3 and A3 on M3, thus reducing the wrap-up phase to one time unit from three.

Another method is to lot stream – simply put, break up the product batches to smaller lots – e. g. instead of processing 1000 foobars in one batch, we process 100 foobars in each one of the 10 batches. The clever trick of Joinable Schedules is to add another constraint to the schedule so it will automatically become lot streamable. This criteria is given in [1] formally, but informally it goes as follows “If we cut out our scheduling plan from paper and make a copy of it, and we put them next to each other, we should be able to join them in a way that the bottleneck machine is working continuously and the other machines' processing do not overlap”. With that criteria, we can split or batches to any number of smaller batches with lot streaming. This is, not surprisingly, a Joinable Schedule – as seen in Figure 2. By splitting up the schedule on Figure 1 to three batches, we reduced the wrap-up phase to one time unit from three.

For zero set-up times[2], infinitely small batches are ideal as it eliminates the wrap-up phase entirely. However, with non-zero set-up times, as found in many real-world systems, increasing the number of batches above a certain limit makes the production run longer. This limit can be calculated either as in [1] or by iterative algorithms as in [2].

5. ADVANTAGES OF JOINABLE SCHEDULES

If we have a possible schedule, a few automated checks show whether the schedule is a Joinable Schedule or not, as described above. If it is a Joinable Schedule, then the ideal number of batches can be automatically calculated and in that very moment the schedule is found. This schedule has a known goodness, which can be calculated from the length of the heavy work phase and from the sum of the set-up times practically instantly. In the example presented in Figure 1, if we assume that retooling in total takes 0,1 time units, then as the optimal number of batches is six[3], then the resulting Joinable Schedule finishes after 13,1 time units, as the heavy work phase takes 12 time units, the wrap-up phase takes 3/6=0,5 time units while the 6 retoolings takes 0,6 time units. In this example, there is no schedule, that can finish sooner than 12,1 time units, so our Joinable Schedule is only 8,3% worse than the possibly non-existent ideal solution. It is possible though, that there is no solution for this problem that can finish in 12,1 time units – as long as we cannot find the solution that finishes in 12,1 time units, we cannot be sure that it exists. But we can be sure that we waste no more than 8,3% of the time by using this Joinable Schedule.

Please note that we can use this method on any schedule that meets the criteria. That schedule, can be automatically transformed to a Joinable Schedule and instantly analyzed. For example, if we swap the A3 and the B3 tasks, the wrap-up time will be 1 time unit, and with the 0,1 time unit total retooling time we will automatically have a Joinable Schedule with 3 batches and a total production time of 12,63 – which is less than 4,5% worse than the possibly unreachable 12,1 time units.

Any other scheduling found by other means (random walk, constraint programming, etc.) can be converted to a Joinable Schedule as long as it meets the criteria.

[2] or retooling times
[3] or five – both give the same numerical results

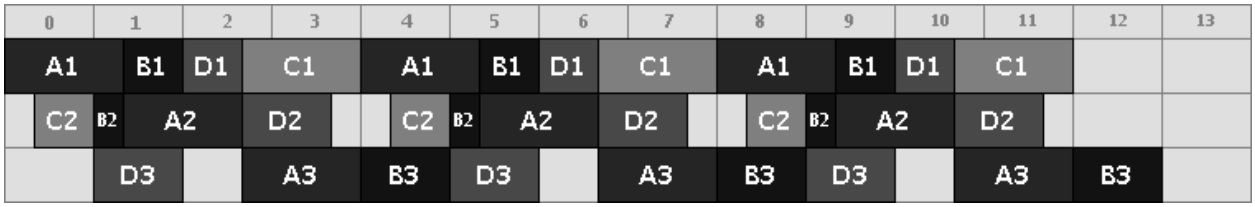


Figure 2.
A Joinable Schedule made from the schedule in Figure 1, splitting the batches to three identical smaller batches.

6. DISADVANTAGES OF JOINABLE SCHEDULES

There are problem instances where Joinable Schedules cannot exist. For example, if the bottleneck machine only has tasks that need preprocessing by other machines, the bottleneck machine will be forced to be idle at the beginning of production, therefore there will be no Joinable Schedule for the given problem. This can be easily verified with simple checks as shown in [2].

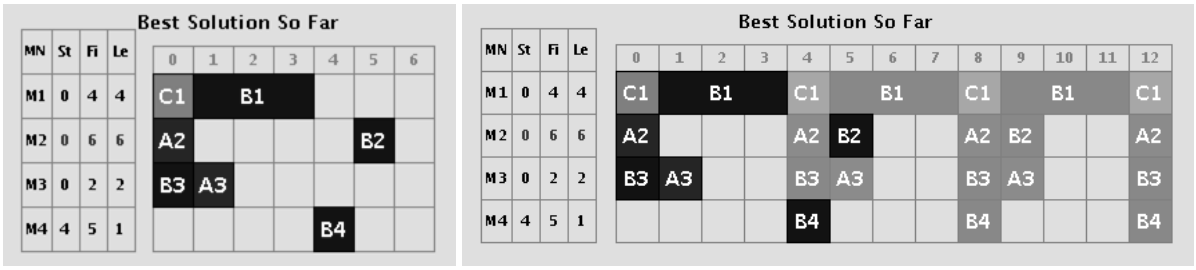
There are also problem instances that have zillions of better or worse schedules, but none of them meets the criteria of Joinable Schedules – especially if there are many machines with similar loads as the bottleneck machine. In these cases, we cannot prove that there is no Joinable Schedule, we just cannot find any while wasting CPU power trying to find one.

In both cases, the problem with Joinable Schedules is that there is no known method of instantly generating Joinable Schedules – finding Joinable Schedules is non-real time, and is most probably NP-hard.

7. NOT QUITE JOINABLE SCHEDULES

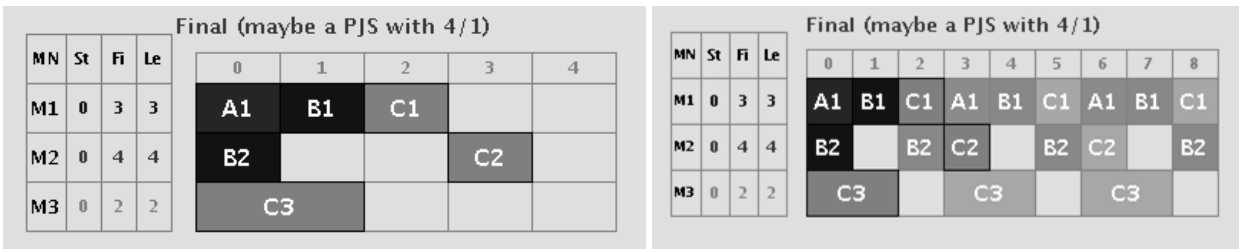
There are a few other classes of schedules that are not Joinable Schedules by the definition given in [1], yet they are equivalent with Joinable Schedules. Practically, as long as we can use the same formulas and the same lot streaming and we get the same production time, we can use these schedules as Joinable Schedules as well. One such class is the Curiously Joinable Schedules – as shown in Figure 3.

Another such class is the Pipelinable Joinable Schedules – one is shown in Figure 4. This is surely not a Joinable Schedule – it fails both the original definition and the informal definition given above. Yet, with preprocessing some tasks in the previous cycles, e.g. preprocessing another B2 in the first cycle, as shown on the right of Figure 4, we get a schedule that works exactly as good as a Joinable Schedule – as inspired by [3].



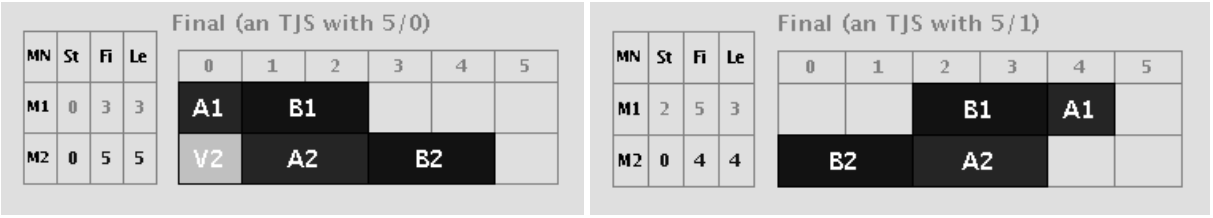
a b

Figure 3.
A schedule that, as shown on the right, is joinable, but is not Joinable Schedule as per the original definition in [1].



a b

Figure 4.
A Pipelinable Joinable Schedule that is not Joinable Schedule at all, but has the same merits, as shown on the right.



a b

Figure 5.
A Schedule that solves a problem that is proven to have no Joinable Schedule solutions, and a similar problem.

Another such class is the Virtually Joinable Schedules. For some problem instances, that are proven not to have Joinable Schedule solutions, Virtually Joinable Schedules do exist and provide a perfectly equivalent schedule. It is likely that any problem solvable with Joinable Schedules is the mirror image of a problem that can be solved with a Virtually Joinable Schedule, although this hypothesis is not proven. Please note that the two problems are very different scheduling-wise despite being symmetric – e.g. the two problems in Figure 5 have different wrap-up phases (2 and 1 time units, respectively).

To push it even further, there are the class of Virtually Continuous Joinable Schedules. These schedules are not Joinable Schedules and are solutions for problems, that are proved to not have Joinable Schedule solutions as the bottleneck machine must idle. This idle time on the bottleneck machine is filled with a virtual (dummy) task, which is later used as free space when pipelining the preprocessing tasks, similarly to Pipelinable Joinable Schedules. This results in a solution that can be handled just like a Joinable Schedule although it is absolutely not a Joinable Schedule – practically, it is two non-Joinable Schedules, one for the first cycle and another one for the other cycles.

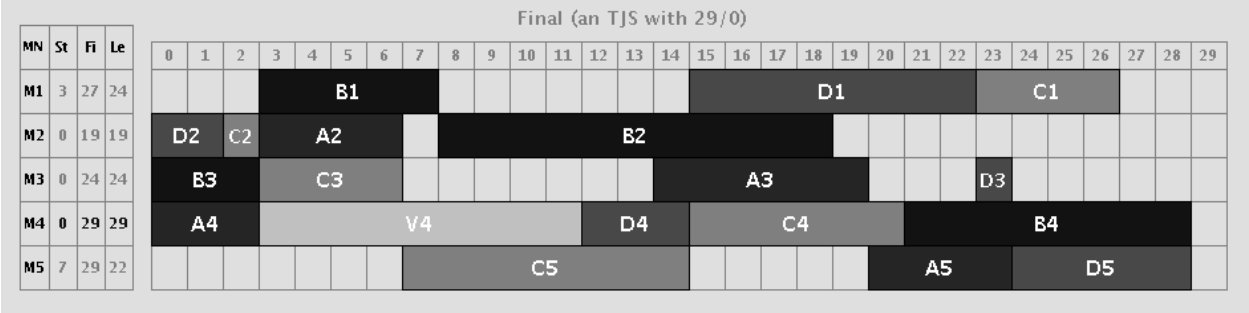


Figure 6a.
A Virtually Continuous Joinable Schedule

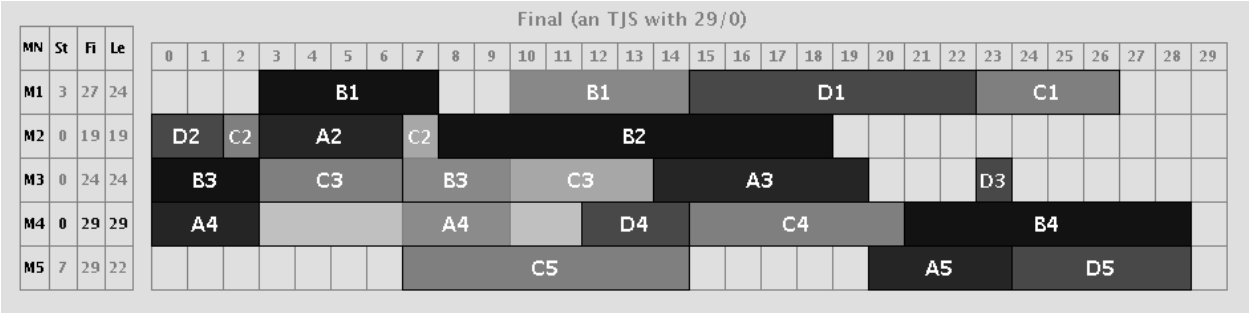


Figure 6b.
The same schedule pipelined.

8. SUMMARY

As shown, Joinable Schedules are a valuable method in solving Job Shop problems. Basically, Joinable Scheduling is a clever trick to automatically optimize lot streaming to a number of schedules for a number of problems. Although there are scenarios for which Joinable Schedules as such do not exist, there are many similar schedules that can be (automatically) converted to a schedule that can be processed with the same methodology to use the advantages of Joinable Schedules.

On the other hand, there are problem instances, for which finding Joinable Schedules is impossible with the heuristics algorithms on current computers within practical time limits. Also, there are no known methods to automatically find Pipelinable Joinable Schedules or Virtually Continuous Joinable Schedules in practical time limits for all problem instances. Also, there is no guarantee that a given problem instance is practically solvable with this methodology.

9. LIST OF REFERENCES

[1] J. Somló, E. Kodeekha, *Improving FMS scheduling by lot streaming*, Periodica Polytech. Mech. Eng., Vol. 51/1, 2007, pp. 3-14

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

[3] Péter Arató, Tamás Visegrády, István Jankovits, Szabolcs Szigeti, *High-Level Synthesis of Pipelined Datapaths*, Panem, 2000