Using structural invariants to improve quality and runtime of scheduling algorithms for re-entrant flowshops

Joost van Pinxten

July 19, 2016

1 Motivation

The heuristic scheduler has a (reported) worst-case complexity of $O(V^5)$, as it calculates for all possible choices between interleavings $O(V^2)$ the Bellman-Ford ASAP start-times $O(V \cdot E)$[1]. This document describes two heuristics that can be applied to the general case and that can be used to limit the horizon of possible choices, and reduces the Bellman-Ford worst case complexity to $O(W^3)$, where typically $W \ll V$. This should improve the overall scalability of the heuristic for larger jobs. Both techniques rely on the fact that we can detect positive cycles (meaning infeasibility) in the graph before executing the Bellman-Ford algorithm.

From the printer use case, we have seen that there is a limited number of interleavings possible due to the limited size of the buffer. Even though a job may contain thousands of pages, the interleaving options are limited by the size of the loop, which is in worst-case around 100 pages; the relation to the runtime of the algorithm and the loop size imposes a physical limit on the relation between computational power and the physical layout and properties. The expectation is that reducing from thousands of vertices to only 100 (and in Océ's case typically even less), will dramatically reduce the stress imposed by the Bellman-Ford algorithm. This should improve the real-time behaviour of the scheduling algorithm dramatically; scheduling a new page (i.e. flowshop job) can then be done in only $O(V \cdot W^4)$. This also implies that the *runtime performance* of the scheduling algorithm is directly tied to the loop length and the minimum sheet size.

This document explains the observations made on the graph of the (2-)re-entrant flowshop, and tries to make them explicit enough to use in the online algorithm. Additionally, some effort is made to make the observations extend to higher re-entrancy flowshops.

[1]The number of edges in the graph is initially $6 \cdot (r-1) + 4 \cdot (|J| - 1) + 2$, which is much less than V^2. Additionally, at most $2 \cdot V$ edges can be introduced by the ordering tuples

2 Scheduling re-entrant flowshops

The scheduling freedom in re-entrant flowshops exists in the starting times of the operations, and the relative occurrence of these operations. In a re-entrant machine in a flowshop, operations are partially ordered. As a machine executes at most one operation at a time, the operations need to be sequenced by enforcing a total ordering on them. A total order can be achieved by adding additional dependencies (and possibly more restricting setup times) between operations[2][3].

A schedule therefore consists of a total ordering of the operations that will not violate any of the deadlines. The earliest possible start times can then be calculated from such a schedule to determine the release moments of the sheets.

Observations of the scheduling decisions for re-entrant flowshops typically originate from the printer use case. A structural invariant that may not be valid in other contexts is that the first and second pass operations have identical processing times, and that only the job type matters for the setup time between two operations; i.e. it is independent of whether it was a first or second pass operation.

We distinguish the dependencies that add ordering information to the schedule versus those that don't. Non-ordering dependencies only unnecessarily restrict the starting times, as shown in Figure 1, and these decisions are therefore never considered.

A further distinction can be made by observing that we can locally determine that certain orderings can never be feasible. Take for example the sum of processing times (excluding setup times, so that it is a strict lower bound) between two first pass operations; if this time is already higher than the deadline for the second pass, then that first pass operation can never be scheduled before the second pass operation. This implies a limit on what we typically call the *scheduling window*. We will show that the scheduling window can be leveraged in order to improve the online performance of the scheduling algorithm, despite the possible influence it may have on previous jobs.

3 Properties of the (2-)re-entrant flowshop

Take as example the 3 machine flow shop $F_{3,<1,2,1>}$ where the second machine is 2-re-entrant. This flowshop is used to model the printer case. Operation x of job j is denoted with $O_{j,x}$. When there is a direct precedence relationship from $O_{j,x}$ to $O_{k,y}$, then we denote this with $O_{j,x} \rightarrow O_{k,y}$. When there is a

[2]This is enough in most cases, but not sufficient for the general case; sequence-dependent setup times may appear on some edges for partial schedules, but the next scheduling decision may remove the necessity for that setup time (by avoiding it). This does not happen in the printer case study

[3]It is assumed that the sequence-dependent setup times between two subsequent operations of the same type are a minimum setup time. I.e. there is no interleaving possible such that a smaller setup time between these operations is achieved through a combination of smaller setup times

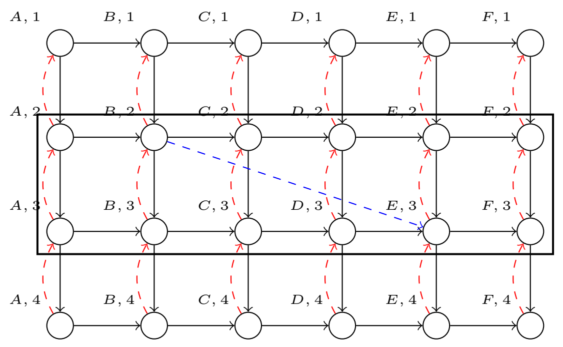

Figure 1: Example of a non-ordering dependency that might unnecessarily restrict the starting times of operation E3 and further.

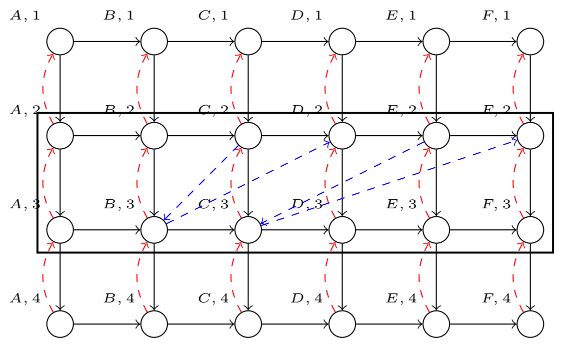

Figure 2: Second pass of job 2 can be interleaved between first pass of job 3 and 4 if the deadline between first and second pass of job allows that; dashed red edges are deadlines (and are modelled with negative weights), the ordering tuple is shown with dashed blue edges.

(transitive) precedence relationship from $O_{j,x}$ to $O_{k,y}$, then we denote this with $O_{j,x} \prec O_{k,y}$.

Firstly, the problem definition states that the sequence of jobs (i.e. sheet) is totally ordered:

$$\forall O_{i,x}, O_{j,x} \in F | i < j : O_{i,x} \prec O_{j,x} \tag{1}$$

Secondly, the operations in a job are also totally ordered:

$$\forall O_{j,x}, O_{j,x+1} \in F : O_{j,x} \to O_{j,x+1} \tag{2}$$

A valid schedule is found by creating a *feasible total ordering* of the operations on the re-entrant machine(s). Note that the operations of non-re-entrant machines do not need to be scheduled as their operations are totally ordered by definition.

Although in the generic case the deadlines can be defined between arbitrary operations, there is typically a certain structure to the deadlines and setup times in the flow shop. In the printer case, for example, the deadlines are all intra-job deadlines: i.e. in Figure 2 the source and destination of the dashed red edges are all from the same job. This enables us to use invariants to reduce the total complexity of the graph structure.

An ordering tuple is a tuple (a, b, c) which takes 3 operations and imposes a sequential constraint, eliminating possible parallelism: $a \prec b \prec c$, and $a \to c$. The analogy of the ordering tuple in the printer case is interleaving second-pass sheets with first-pass sheets.

Resolving the re-entrancy is done by selecting which *ordering tuple* is the most productive, while not sacrificing too much flexibility[4]. Such ordering tuples are generated for each combination of possible interleaving. I.e. for each job, it considers whether a higher-pass operation can be inserted in between two directly adjacent lower-pass operations.

3.1 Purging infeasible combinations of ordering tuples

The higher-pass operation is necessarily of an earlier job than the current job being scheduled (as the jobs and operations per job are totally ordered). Given two ordering tuples (a, b, c) and (d, e, f) where $c \preceq d$, the combination of the two is definitely infeasible when:

$$\forall (a, b, c), (d, e, f) \in OT : e \prec b \tag{3}$$

as a a positive cycle over $b \prec c \preceq d \prec e \prec b$ would necessarily exist[5] since:

$$\left.\begin{aligned} a \prec b \prec c \\ c \preceq d \\ d \prec e \prec f \\ e \prec b \end{aligned}\right\} b \prec c \preceq d \prec e \prec b \tag{4}$$

See for an example Figure 3.

Remark: can we generalize this also to higher re-entrancy than 2-re-entrancy?

3.2 Purging ordering tuples with infeasible deadlines

The intra-job deadlines of the printer case can be leveraged to purge the combinations. Take an ordering tuple (a, b, c) where there $\exists b' : b' \to \wedge b' \prec a$. The lower bound on the time between b' and a gives a hint to the maximal slack on the deadline. If this slack is negative, then the graph contains a positive cycle. The minimum time between b' and a is the sum of operations between

[4]leaving in the middle whether you use a ranking or trade-off scheduler

[5]under the assumption that operations always take a positive non-zero amount of time; it might not hold under 3-re-entrancy either

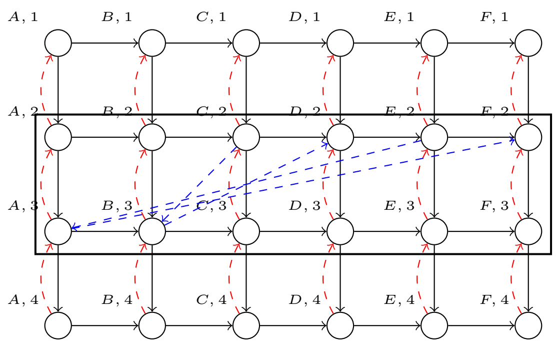

Figure 3: Infeasible combination of ordering tuples: a positive cycle exists: $B,3 \to D,2 \to E,2 \to A,3 \to B,3$

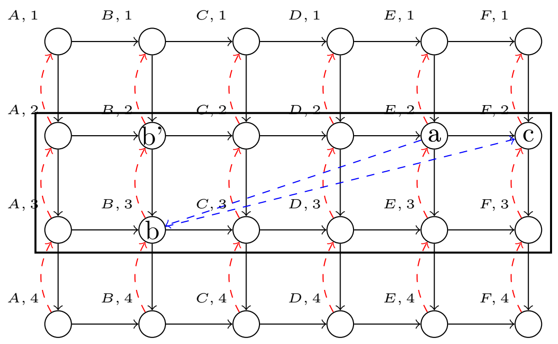

Figure 4: The ordering tuple (a,b,c) may never be able to meet the deadline between B,3 and B,2

them. This may be calculated by using a longest path algorithm from b' to a, or by using only the sum of operations in a certain path between b' and a [6]. Once an ordering tuple (a, b, c) has been deemed infeasible, any ordering tuple (d, e, f) with $a \prec d$ has necessarily less slack on the deadline, and is therefore also infeasible.

4 Windowing

From here on out, we assume that the re-entrancy is resolved per operation, from the first job to the last job, or vice-versa.

We have shown two mechanisms that can be leveraged to limit the calculation of interactions between starting-times. One provides a starting point (i.e. lower bound) for the window, and the other provides an upper bound.

[6]we may be able to prove that it's always a certain path, and the generic shortest path might be overkill to use

The lower bound is determined by the *last interleaved operation*. It determines that no operation preceding the last interleaved can be regarded as a feasible scheduling option anymore, as it would introduce a positive cycle (i.e. infeasibility) in the graph. I.e., imposing both $O_{k,x} \prec O_{l,x+1}$ and $O_{l,x} \prec O_{j,x+1}$ (when $j < k < l$).

This means that, after some point, the deadline of an interleaved higher pass is already passed. I.e. any ordering tuple after that operation, can never lead to a feasible interleaving, as operation times are non-negative.

The upper bound is determined by the deadline constraint between the current operation under consideration, and the minimal setup + operation times of the following operations.

These two heuristics lead to the extraction of a natural window, if one exists. Using said window may help greatly to avoid most of the expensive calls. Of course, this only helps if $W \ll V$ indeed holds. For small V it might not, but it will help in typical cases where the jobs are large.

5 Finite horizon Bellman-Ford

Consider forward iteration[7], where the Bellman-Ford algorithm has calculated starting times for the previous schedule. Anything scheduled before the last interleaved operation are not allowed to be re-timed anymore; their ASAP-times become fixed, and don't need to be re-calculated. We can use the first operation of the job which contains the interleaved operation as source and the rest of the algorithm can remain the same.

It is possible that a scheduling choice pushes the second pass sheet to a later starting time. As the deadline imposed on the second pass sheets may mean it needs to schedule its corresponding first pass later than planned, this may again have impact on the second pass of an even earlier sheet. If this earlier sheet has enough slack on its deadline, ánd the setup time between those operations does not exceed the sum of the processing times and setup times between the first passes of the sheets, then the schedule will be stable and feasible. As the deadlines for second pass sheets can delay their corresponding first pass transitively, the extent of a scheduling decision may encompass re-timing all operations to detect that it is a feasible decision.

Figure 5 shows an example where a set-up time between the first pass of A and second pass B may lead to a later starting time than the first pass of B. This in turn may lead to later second passes of previous jobs, which in turn can propagate to their corresponding first pass. We therefore cannot guarantee that future scheduling decisions will not influence operations that are not local to the decision. This makes it impossible to determine a window in which the influence of scheduling decisions on previously calculated starting times are all captured. However, we can take the penalty of not re-timing the operations outside the window.

[7]For reverse iteration this becomes much more convoluted, as the incoming paths may change and therefore everything may need to be re-computed

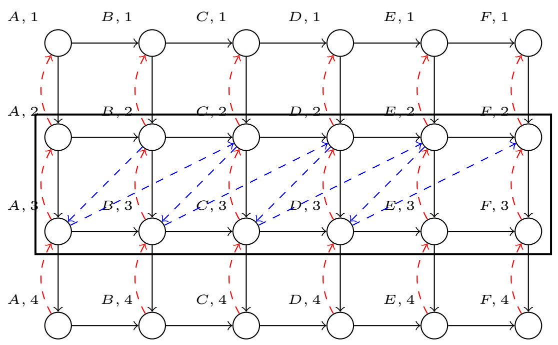

Figure 5: The 'backpressure' on the edges imposed by the deadlines and the interleaving decisions may lead to later start-times than without the choice. There seems to be no general way to determine how far back it can propagate.

In the Océ case this implies that we do not make full use of the buffer; it only works when we determine up front that the buffer will be large enough to accommodate a scheduling decision productively.

However, as we know that there is a limited window of opportunity, we can take this information into account as well; any calculation that involves starting from any of the nodes outside of that window don't need to be taken into account either, as they do not yet impose additional constraints on the starting times[8].

To tighten the worst-case complexity from $O(W \cdot E)$ to $O(W^2)$, we need to show that the edge set involving nodes from W is equal to at most $6 \cdot (r-1) + 4 \cdot (|J|-1) + 2$, and $|J|$ has been limited to $|W|$, as the jobs before the window have already been fixed.

Remark: to implement this needs some further investigation; e.g. copying around all the ASAP times all the time is not very efficient.

6 Transitive reduction of the scheduling decisions

The constraint graph encodes the requirements that e.g. $x_2 \geq x_1 + \alpha$, which is denoted by an edge with weight α from x_1 to x_2, as shown in Figure 9.

An illustration shows how this could be applied. With the system of inequalities that need to be satisfied given as:

[8]although the Bellman-Ford algorithm may not see this immediately, we just don't need to consider that information yet, and can simply skip calculating it

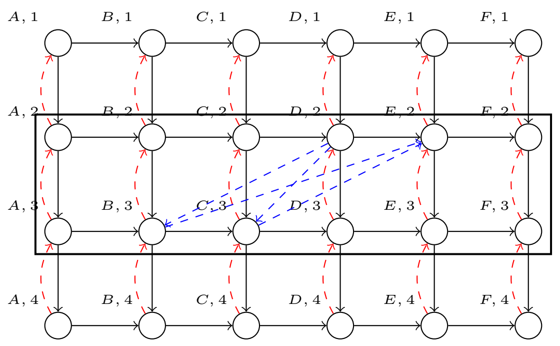

Figure 6: Example of scheduling decisions that can be reduced

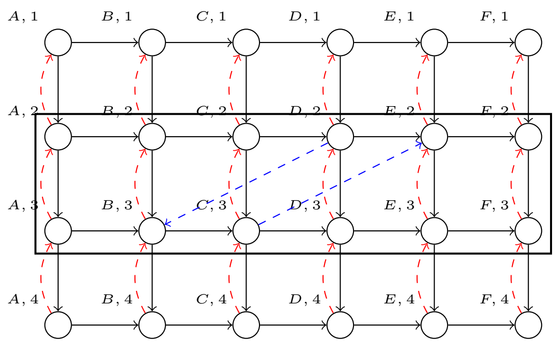

Figure 7: Example reduced set of scheduling decisions

$$\begin{pmatrix} x_3 \geq x1 + \alpha \\ x_2 \geq x1 + \beta \\ x_3 \geq x2 + \gamma \end{pmatrix} \equiv \begin{pmatrix} x_2 \geq x1 + \beta \\ x_3 \geq \max\left(x2 + \gamma, x1 + \alpha\right) \end{pmatrix}$$

$$\equiv \begin{pmatrix} x_2 \geq x1 + \beta \\ x_3 \geq x1 + \max\left(\beta + \gamma, \alpha\right) \end{pmatrix} \equiv \begin{pmatrix} x_2 \geq x1 + \beta \\ x_3 \geq x2 + \max\left(\gamma, \alpha - \beta\right) \end{pmatrix}$$

The last set of inequalities is also visualized in Figure 10. We use this transitive reduction method to reduce the scheduling decisions shown in Figure 9 into Figure 10.

Generically, the trapezoid

7 Other remarks

In definition 8 from Umar's DATE paper the requirement for feasibility is necessary, but not (by definition?) sufficient. On arbitrary graphs, the *feasibility of a schedule* may be compromised by a *feasible choice*, as it may lead to infeasibility

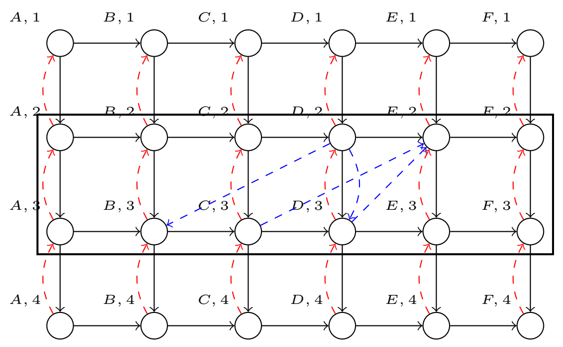

Figure 8: Example reduced set of scheduling decisions

$$\begin{tikzcd}
x_1 \arrow[rr, "\alpha"] \arrow[dr, "\beta"'] & & x_3 \\
& x_2 \arrow[ur, "\gamma"'] &
\end{tikzcd}$$

Figure 9: Simple example for the transitive closure

$$\begin{tikzcd}
x_1 \arrow[dr, "\beta"'] & & x_3 \\
& x_2 \arrow[ur, "{\max(\gamma, \alpha - \beta)}"'] &
\end{tikzcd}$$

Figure 10: Reduced version of the simple example

$$\begin{tikzcd}
& x_1 \arrow[r, dashed, "\alpha"] \arrow[dl, "\beta"] \arrow[dr, "\delta"'] & x_3 \\
x_2 \arrow[r, dashed, "\tau"'] & x_4 \arrow[ur, "\gamma"'] \arrow[r, dashed, "\epsilon"'] & x_5 \arrow[u, "\phi"']
\end{tikzcd}$$

Figure 11: 4-node example for the transitive closure

$$\begin{tikzcd}
& x_1 \arrow[r, dashed, "{max(\alpha, \gamma + (\tau + \beta))}"] \arrow[dl, "\beta"] & x_3 \\
x_2 \arrow[r, dashed, "\tau"'] & x_4 \arrow[r, dashed, "{max(\epsilon, \delta - (\tau + \beta))}"'] & x_5 \arrow[u, "\phi"']
\end{tikzcd}$$

Figure 12: Reduced version of the 4-node example

of all other choices later on. This may be caused by (strangely structured) setup times and deadlines? It is of no concern in the Océ case.

It might also be better to resolve the re-entrancy of one job first, as that may lead to fewer infeasible schedules later. The idea is that the first ordering tuple can lead to high amounts of 'stress' on the deadline of later re-entrancies, which can then not be resolved to previous scheduling decisions.