Novel methods to neutralize loops and equal-effect action sequences in resource-action-constraint models used in scheduling

Ph.D. thesis booklet

Elemér Károly Nagy

Supervisor:

István Loványi, Ph.D.

Department of Control Engineering and Information Technology

Budapest University of Technology and Economics

Budapest, 2009.
1. Preliminaries

According to [1], "Advanced representation and solution techniques in production planning and scheduling received significant attention during the past decades, both from the part of research communities and the industry. These techniques hold out a promise of increased productivity, better service level, and lower production costs, by supporting the management to make smarter decisions on various levels of the planning hierarchy."

![Figure 1 – The hierarchical scheduling model used](image)

This planning hierarchy is relevant only in hierarchical planning systems, which are divided to $3+1$ or $3\frac{1}{2}+1$ levels of production planning in [1]. Other sources either divide the planning hierarchy into three levels ([2], [3], [4]) or mention a different terminology ([3], [4], [5]). Yet, these different terminologies agree that there is a step between the tactical level and real-time control. This step will be called *plan making* in this thesis. This step is a part of the "scheduling" or the "operational scheduling" level, and turns abstract intervals into a plan, a list of exact steps, a plan that can be executed by mindless machines and unqualified workers alike.

Plans might be created by humans or by algorithms\(^1\). Humans have a lower initial cost and high levels of abstraction, and until the last decade, computing power was so

---

\(^1\) Computer softwares implementing algorithms
expensive that in many cases\(^2\) it was infeasible to generate plans by computers because of the algorithmic complexity. Then, one day, in a complex game, a computer beat a human called Kasparov.

Computing power became so cheap that it is cheaper to use a DSPs in music players than analogue components. It is cheap enough to permit Linux distributions to compile Gigabytes of source code during installation\(^3\), "wasting" more computing power on a single installation than what was available to crack the Enigma\(^4\) during World War II\(^5\).

On the other hand, engineers are becoming more and more dependent on their computers - seemingly, they are not able to do their job without a computer anymore. Thus, if current trends continue, it is reasonable to assume that more and more plans will be made by algorithms - supposedly just like compilers took the jobs of assembly programmers.

In low-volume, low-standardization production systems, as in make-to-order systems\(^6\), there are many plans to make, with the costs and time being rather limited. In such systems, ad-hoc solutions, rules of thumbs and instincts provide plans that are considered to be "good enough", even though a proper algorithm could near-instantly provide near-optimum plans, increasing productivity and/or profit.

This thesis is about planning algorithms that eliminate some specific problems from some specific models and about algorithms that yield "good enough" results faster and cheaper than humans.

This thesis was inspired by the scheduling and allocation chapters in [6], urging the Author to find and evaluate all possible scheduling combinations even in non-trivial scenarios (i. e. nested recursive loops) in general RAC graphs.

The same inspiration fruited results in another area of scheduling, namely the analysis of the Joinable Schedules presented in [7], which resulted in algorithms that can find Joinable Schedules in many cases and can prove the non-existence of Joinable Schedules in many other cases. Joinable Schedules have the advantage of having a known goodness and a real-time optimization algorithm for lot streaming.

\(^2\) Military, aviation, and space projects excluded
\(^3\) Namely, a full Gentoo system
\(^4\) The secret German crypto-typewriter
\(^5\) Author's estimation, based on http://www.tnmoc.co.uk/cipher7.htm
\(^6\) also referred to as Jumbled Flow and Job Shop systems
2. Problem statements

There are many details in scheduling which are hard, if not impossible to model in classic models. Examples include batch deadlines, individual unit deadlines, assembly, disassembly, jobs that require more than one machine, jobs that depend on other jobs, slowly reconfigurable machines, renewable resources, and so on. SIMONEK was designed with these problems in mind, yet it had to be simple enough to effectively describe and solve problems.

Another goal was to create a modeling language which can describe problems that can be solved automatically and transformed into an exact list of steps.

Chapter 8 of the dissertation contains the problems of a laptop repair shop. Although ILP is able to locate the point of maximum profit, examples are provided for cases when a) the exact steps to reach this maximum profit is not provided b) extra information, valuable for business, is provided by SIMONEK c) the maximum profit is unreachable d) by changing a few parameters, the problems becomes increasingly complex in ILP e) some changes in the parameters are hard to formulate in ILP.

Joinable Schedules are a valuable tool to solve some Job-Shop Problems with Lot Streaming, still [7] recognizes that "... we could not find an initial joinable schedule for the original example (it may exist).". Thus, the questions rise: a) when do Joinable Schedules exist? b) Can we prove if no Joinable Schedule exists for specific cases? c) Can we generalize Joinable Schedules in a way so that for cases where no Joinable Schedule exists, generalized Joinable Schedules do exist? d) Is there a schedule which is equivalent in efficiency with Joinable Schedules for cases where the non-existence of Joinable Schedules is proved?

3. Terminology

RAC models, Zero Loops, Equal-Effect Action Sequences

RAC models are composed of Resources, Actions and Constraints.

Resources include materials (Iron, Coal), mid-products (Steel) and end-products (Money) as well as abstract resources(Time).

Actions represent processes (2 Coal + 2 Iron + 1 Money + 4 Time = > 3 Steel).
Constraints represent the laws of physics (Coal > 0), the constraints of business (Money > 0) and the constraints of production (Iron + Coal + Steel < 320). States and actions that are forbidden by the constraints are referred to as invalid states and invalid actions, respectively.

RAC problems are RAC models with initial resource values, goals and value functions.

Initial resource values (Time = 100, Coal = 20, ...) define the start state of the model.

Goals (Time = 0, Money > 1000) define the possible end states of the model.

Value functions (Money = 1, Steel = 20) evaluate the goodness of the solution.

There are plenty of RAC models - we will focus on linear RAC models, where both the value functions and the actions are linear.

Action sequences or AS-es are sequences of actions. The order of the actions is important as, depending on the constraints, certain actions may enable or disable other actions.

Figure 2 shows a possible graphical representation of RAC problems: the start state is blue and is marked with “S”, the forbidden states/actions are red, the end states are green and are marked with their values, and an action sequence is marked with bold lines. For the sake of simplicity, we will refer to these graphical representations as “RAC models” in the future.

---

7 Limited storage capacity
Solutions or plans are action sequences that take the RAC model from the initial state to an end state. The solutions with the highest value, as defined by the value function, are the best solutions. The AS on Figure 2 is a solution, although not the best solution.

![Figure 3 – A RAC model with zero loops](image)

Zero loops (ZLs) are action sequences that do not change the values of resources when executed. If a zero loop can be executed once, it can be executed infinitely, thus, if there is an opportunity to execute a zero loop in a valid solution, there are infinite valid solutions for the problem. On Figure 3, two zero loops are marked with dotted lines.

Equal-effect action sequences (EEASes) are action sequences (coming in pairs, triples, etc.) which have the same effect - that is, executing them from the same initial state, they take the model to the same resulting state. Figure 4 shows a pair of EEASes marked with dotted and dashed lines.

---

8 Their linear combinations are also zero loops
The problem with ZLs and EEASes is that they increase the number of nodes in a way that solving the problem within the given timeframe becomes infeasible. Yet, as formally all these solutions are valid, they cannot be simply dropped unless we can cope with losing solution(s).

While humans inherently think in abstract notations and generalizations, computers "think" numerically. Not only are humans able to compress EEASes with abstraction, such as "these 10 steps in any order", which is 10! or 3 628 800 different plans for a computer, but they are also capable of using abstractions like "and repeat Step 4 as many times as you wish" instead of generating all the infinite solutions produced by a ZL.
SIMONEK. The latter is the JISIMONEK software package, which is distributed under GPLv3 from sourceforge.net. An example is shown in Figure 6, which is a reduced problem presented on Figure 5.

![Figure 6 – The trivial-to-solve, reduced RAC model (X denotes reduction).](image)

To eliminate ZLs and EEASes, some kind of tree traversal algorithm needs to be applied in the initial state. This algorithm will try to execute actions (travel the edges of the graph) to reach new states (model instances with different resource values), aiming to find the best solution(s) for the problem.

![Figure 7 – Reducing the RAC model - eliminating duplicate state. Numbers on edges denote discovery order, numbers in states denote resources.](image)

The graph, which is created by this traversal, is fed to the Forfex-algorithm from time-to-time, which detects ZLs and EEASes and returns them to the traversal algorithm. This is demonstrated on Figure 7 and Figure 8.

---

9 Simulation and Modeling Network
10 Java Implementation of SIMONEK
11 There are a few implemented in JISIMONEK, namely DFS, BrFS, RFS and some greedy heuristics-based BFSes.
Then, the traversal algorithm can drop all AS-es that contain any previously found ZLs or EEAS-es. After finding the best solution(s) for the problem, the algorithm needs to re-generate the solution variants with the previously found ZLs and EEAS-es, as shown in Figure 9.
Joinable Schedules

Let us model a Job Shop scheduling problem as follows:

The machines\(^{12}\) are denoted by \(M_j\) (\(j=1, 2, 3, \ldots, J\)). The machine with the highest total load is referred to as the Bottleneck Machine and is denoted by \(M_{BN}\) or BNM.

Parts are denoted by \(P_i\) (\(i=1, 2, 3, \ldots, I\)). Every part needs to be processed on an ordered sequence of machines. These processing tasks are denoted by \(Pr_{i,M} \), \(Pr_{i,M_1}, Pr_{i,M_2}, \ldots, Pr_{i,M_n}\).

The time needed for the processing \(Pr_{i,M} \) is denoted by \(T(Pr_{i,M})\).

The machine \(M_j\) starts processing at \(T_{S,j}\) and finishes processing at \(T_{F,j}\). Global processing starts at \(T_{S,G}\) or \(T_0\) and ends at \(T_{F,G}\).

The time needed to process all prerequisites of the task \(Pr_{i,M} \) is denoted by \(PrePT(Pr_{i,M})\) and is the sum of all \(T(Pr_{i,M})\) where \(Pr_{i,M} \) is before \(Pr_{i,M} \). The time needed to process all "postrequisites\(^{13}\)" of the tasks \(Pr_{i,M} \) is denoted by \(PostPT(Pr_{i,M})\) and is the sum of all \(T(Pr_{i,M})\) where \(Pr_{i,M} \) is after \(Pr_{i,M} \).

Joinable Schedules are schedules for the given machines and parts that satisfy the following 3 criteria:

a) The bottleneck machine is loaded without idle times

b) The Gantt chart is not overlapping with itself

c) Processing on the bottleneck machine starts at \(T_0\)

Joinable Schedules are tested with the Joinability Test in [1], which is defined as "if \(T_{F,G} - (T_{F,j} - T_{S,j}) \geq T_{F,G} - T_{F,M_{BN}}\) for all \(j=1, 2, 3, \ldots, J\), then the schedule is joinable".

Figure 10 visualizes the Joinable Schedule method.

---

12 homogeneous machine groups
13 This word precisely describes the relation of the tasks in question
4. **The Forfex-algorithm**

In this chapter I define a few expressions I will use in the description of the Forfex-algorithm.

**Def.:** A **Histogram** (in SIMONEK models) is a vector that contains the number of executions for each action. That is, \( H_2=4 \) means that the action denoted by "2" was executed four times.

**Def.:** The **natural difference of two scalars** is the difference of the two scalars, if it is non-negative, and 0 otherwise.

**Note:** The natural difference of two scalars is symmetric if and only if the scalars are equal (natural difference is 0).

**Def.:** The **natural difference of two equal-length vectors** is the natural difference of their corresponding elements.

**Note:** The natural difference of two vectors is symmetric if and only if the vectors are equal (natural difference is 0).

**Def:** A search is **non-shrinking** if it never finds new nodes with a lower level than any previous node. On the graph given in *Figure 11*, any non-shrinking search will result in level numbers \{1, 1, 2, 2, 3, 3, 4\}.

**Notes:** Even though in binary trees level-order searches are always non-shrinking searches, generally, non-shrinking searches are not always level-order searches.

![Figure 11 – Possible level numbers in a graph](image)

The Forfex-algorithm's input is a SIMONEK model, and the algorithm's output is the list of Zero Loops and Equal-Effect Action Sequences found, as if found by a symbolic reduction algorithm. By eliminating these redundancies, larger graphs can be traversed with the same resources. Also, by providing human readable description of the redundancies, instead of a great number of similar equivalent solution variants, only a
few, different solutions are presented, with possible redundancies indicated. The algorithm works by calculating the natural difference vectors of the histograms of EEASes and then reducing the EEASes to their base (removing common sub-AS-es), thus finding ZLs an EEASes.

A Java implementation of the Forfex-algorithm is available in JISIMONEK as SimpleProblemSolver.novelMethod() and the algorithm is given in pseudo-code in the dissertation.

Figure 12 – A flowchart-like (and dataflow-like) representation of the Forfex-algorithm
5. New Scientific Results

Thesis Group 1: SIMONEK and the Forfex-algorithm.

The SIMONEK model is a mathematical tool and software framework to model and to solve scheduling problems. The Forfex-algorithm\(^{14}\) is an algorithm which eliminates some unnecessary redundancies and displays the eliminated routes as if a symbolic reduction algorithm would. When used with a non-shrinking search, the Forfex-algorithm will find the generators of some redundancies first.

1a. I defined RAC models and I designed, defined and developed a RAC model named SIMONEK.

1b. I designed and implemented a software methodology for SIMONEK models as a Java framework, which I named JISIMONEK and for which I released the source code\(^{15}\). I tested and analyzed the framework through a series of various scenarios, which I implemented as examples.

1c. To reduce complex RAC models, I designed a general algorithm to effectively filter out zero loops and equal-effect action sequences from RAC models (specifically, from JISIMONEK models) and named it the Forfex-algorithm.

1d. I proved that any non-shrinking search used with the Forfex-algorithm will find the base zero loops with the form of \(B+O\) before finding their closures with the form of \(B+A*O\), where \(B\) is an action sequence, \(O\) is a zero loop and \(A>1\) is a positive integer.

1e. I proved that any non-shrinking search used with the Forfex-algorithm will find the base zero loops with the form of \(B+O_1\) and \(B+O_2\) before finding their closures with the form of \(B+A_1*O_1+A_2*O_2\), where \(B\) is an action sequence, \(O_1\) and \(O_2\) are zero loops and both \(A_1\) and \(A_2\) are positive integers.

1f. I proved that any non-shrinking search used with the Forfex-algorithm will find the base zero loops with the form of \(B+O_1, B+O_2, \ldots, B+O_n\) before finding their non-trivial linear combination with the form of \(B+A_1*O_1+A_2*O_2+\ldots+A_n*O_n\) where \(B\) is an action sequence, \(O_i\) are zero loops and \(A_i\) are positive integers.

\(^{14}\) Forfex is scissors in Latin
\(^{15}\) Available under the GNU public license from www.sourceforge.net
Thesis Group 2: Joinable Schedules

Joinable Schedules methodology is an efficient way to find schedules for n/m job-shop problems with a known distance from the optimal schedule even if the optimal schedule is not feasible.

2a. I proved that there are Joinable Schedules which do not pass the joinability test and I named them Curiously Joinable Schedules. I proved that there are schedules that are not Joinable Schedules but can be pipelined in a way that they are equivalent with Joinable Schedules in efficiency and I named them Pipelineable Joinable Schedules.

2b. I discovered and implemented a fast\(^1\)\(^6\) test to prove that a given ordering of the tasks on the BNM\(^1\)\(^7\) cannot yield a Joinable Schedule. I discovered and implemented a fast\(^1\)\(^8\) test to prove that a given ordering of the tasks on the BNM cannot yield a Joinable Schedule, if a part is allowed to be processed on the BNM twice or more.

2c. I discovered and implemented a fast\(^1\)\(^9\) test to prove that a given ordering of the tasks on the BNM cannot yield a better solution than an already known Joinable Schedule.

2d. I designed and implemented a simulation platform to study and visualize Joinable Schedules in Java, which I named SISONEK and for which I released the source code\(^2\)\(^0\).

2e. I discovered, implemented and analyzed an algorithm that is able to find and classify Joinable Schedules, Curiously Joinable Schedules and Pipelineable Joinable Schedules in many cases. The algorithm also proves the non-existence of Joinable Schedules in many other cases, incorporating the researched tests, although for sufficiently large problems, the algorithm will not yield any answer. However, in some cases, the resulting schedule is the global optimum, thus these cases are perfectly solved.

2f. I generalized Joinable Schedules by adding Virtual Parts, enabling the algorithm to find schedules that are similar to Joinable Schedules in their efficiency in the cases where the non-existence of Joinable Schedules is proven, and I named these schedules Virtually Joinable Schedules and Virtually Continuous Joinable Schedules.

---

\(^1\)\(^6\) \(o(n)\) if \(n\) is the number of parts on the BNM
\(^1\)\(^7\) Bottleneck Machine
\(^1\)\(^8\) \(o(m*n)\) if \(n\) is the number of parts on the BNM and \(m\) is the maximum number of processings per part
\(^1\)\(^9\) \(o(m*n)\) if \(n\) is the number of parts on the BNM and \(m\) is the maximum number of processings per part
\(^2\)\(^0\) Available under the GNU public license from www.sourceforge.net
Proof for 1d, 1e and 1f:

The $N$ zero loop is a non-trivial linear combination of the $O_1, O_2, \ldots, O_n$ zero loops, with $N$ containing the $O_i$ zero loops $A_i > 0$ times, having the same $B$ base:

$$B+N = B+A_1*O_1 + A_2*O_2 + \ldots + A_n*O_n$$

As the Forfex-algorithm detects zero-loops by identifying the nodes by their resource difference hashes, which is zero for all zero loops, so $B+A_1*O_1 + A_2*O_2 + \ldots + A_n*O_n$ and $B$ will end up with the same hash value, thus they will be compared.

Let us use the $||$ operator to denote the length of an action sequence.

As $|X+Y| = |X|+|Y|$, $|B+A_1*O_1 + A_2*O_2 + \ldots + A_n*O_n| = |B|+A_1*|O_1| + A_2*|O_2| + \ldots + A_n*|O_n|$ and

$$|B+O_i| = |B|+|O_i|$$

For non-trivial linear combinations, either $n>1$ or $A_i>1$ holds.

If $A_i>1$ holds, then by dropping all $O_z$ where $i \neq z$ (for any $0 \leq z \leq n$),

$$|B+N| = |B+A_1*O_1 + A_2*O_2 + \ldots + A_n*O_n| = |B|+A_1*|O_1|$$

As $A_i>1$, we see that $|B+N| \geq |B+O_i|$

If $A_i=1$ holds, then $n>1$, then by dropping all $O_z$ where $i \neq z$ and $j \neq z$ (for any $0 \leq z \leq n$, $i \neq j$), we reduce $|B+N|$.

$$|B+N| = |B+A_1*O_1 + A_2*O_2 + \ldots + A_n*O_n| = |B|+|O_i|+A_j*|O_j|$$

As $A_j>0$ and $|O_j|>0$, $A_j*|O_j|>0$, therefore,

$$|B+N| = |B+A_1*O_1 + A_2*O_2 + \ldots + A_n*O_n| = B+|O_i|+A_j*|O_j| > |B+O_i|$$

That is, in either case, $|B+N| > |B+O_i|$ holds for any $i$, and as the search used with the Forfex-algorithm is non-shrinking, $B+O_i$ cannot be found later than $B+N$, which is QED.
6. Publications connected to the Theses


a.5. (Thesis 1a, 1b), E. K. Nagy, “Felvonóvezérlések hangolása az utasforgalomnak megfelelően”, Felvonók (HU ISSN 1586-1228), Vol. 5 (2005), pp. 48-49


a.8. (Thesis 2a, 2b, 2c, 2d, 2e, 2f), I. Loványi, E. K. Nagy, A Method of Finding (Generalized) Joinable Schedules, being prepared for submission in Periodica Polytechnica
7. References


