Exploiting Repetitive Patterns in Practical Scheduling Problems

A. Kovács[1], J. Váncza (1)[1,2]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende u. 13-17, 1111 Budapest, Hungary

[2] Dept. of Manufacturing Science and Technology, Budapest University of Technology and Economics, Egry József u. 1, 1111 Budapest, Hungary

Abstract

Current production scheduling systems use generic models (e.g., job-shop, flow-shop) to model manufacturing processes. These models hide some structural properties of real-life scheduling problems. In this paper we propose a novel technique to re-discover a certain type of these properties, namely, repetitive task patterns. The techniques can be applied in a pre-processing step before solving the scheduling problem. The re-discovered knowledge helps finding better production schedules with less computational effort.

Keywords: Scheduling, algorithm, graph similarity.

1 INTRODUCTION

Most of the current production planning and scheduling algorithms represent manufacturing processes using generic scheduling models, like flow shop, job shop, or resource-constrained project scheduling [1]. These generic models ensure that the same scheduling model, algorithm, or even the same software product is applicable in different manufacturing environments and industrial applications.

Nevertheless, generic scheduling models hide some structural properties of specific real-life problem instances. This leads to disregarding specialized domain knowledge, and in turn, loss of efficiency when solving an actual scheduling problem. In this paper, techniques are proposed for the detection and exploitation of a certain type of those hidden structural properties within the boundaries of the generic scheduling model. We will focus on the presence of repetitive task patterns in the scheduling problem, which is inherent in discrete manufacturing, where products or components of similar/same type are produced in parallel, using the same technology and a common pool of resources. The notion of **progressive pairs** will be introduced to characterize repetitive patterns on a graph theoretical basis.

Various approaches in scheduling seek for reducing the search space by exploiting repetitive patterns. Still, well-founded theoretical results in this

field have been missing thus far. An example of this deficiency is the approach to multiprocessor scheduling proposed in [3]. Later, it has been shown that the applied sequencing of recurrent tasks can lead to losing the optimal solution [4].

The proposed approach can be considered as a specialized technique for **symmetry breaking**, a method widely used in combinatorial optimization and constraint programming [2]. Symmetry is a function that maps each solution to an equivalent solution. Breaking a symmetry means reducing the search space by eliminating all but one of the equivalent solutions. The proposed methods break symmetries by sequencing the corresponding tasks in repetitive patterns in a preprocessing step.

2 NOTATION AND DEFINITIONS

Without restricting the investigations to a specific scheduling model, the scheduling problem can be represented as a vertex- and edge-labeled directed graph, known as the activity-on-node representation. The vertices of the graph stand for the **tasks** and the edges represent **precedence relations**. Most of the common scheduling models can be encoded into this representation by embedding the parameters, e.g., processing times, resource requirements, or setup time matrix into the vertex labels. It is assumed that edge labels are non-negative real numbers that specify the required delays on the precedence relations: an edge $i \xrightarrow{d} j$ describes that task j must start at least d units of time after the start of i. Note that zero delay means a start-to-start precedence relation, while a delay equal to the processing time of i corresponds to an end-to-start precedence relation. Throughout the paper it is assumed that there are no directed circles of precedences. For the sake of simplicity, we introduce the symbol $i \xrightarrow{d}{}^{*} j$ to denote that there exists a directed path of precedences $i \xrightarrow{d_1} k_1 \xrightarrow{d_2} \ldots \xrightarrow{d_n} j$ such that $\sum_{l=1}^{n} d_l = d$.

The ensemble of all constraints in a scheduling model, except for the precedence constraints, will be called the **model-specific** constraints. The objective value of schedule σ is denoted by $f(\sigma)$, and its meaning also depends on the specific scheduling model. Without loss of generality, it is assumed that $f(\sigma)$ is to be minimized. The start time of task i will be denoted by S_i, or S_i^{σ} if it is important to refer to a specific schedule σ.

For example, in the classical job-shop scheduling problem, the label of the vertex corresponding to task i contains the pair $\langle p_i, \rho_i \rangle$, where p_j is the processing time of the task, and ρ_j is the index of the resource that it requires. The precedence relations are of the end-to-start type, therefore the delay on the edge $i \xrightarrow{p_i} j$ equals the processing time of task i. The objective $f(\sigma)$ is the makespan of the schedule.

3 PROGRESSIVE SOLUTIONS OF SCHEDULING PROBLEMS

Our notion of similarity of two task sets has two types of conditions: the first type for the similarity of the vertex labels, and the second for the precedence structure within the two sub-graphs. For the vertex labels of a pair of tasks, the condition is as follows:

Definition 1. In a given scheduling model and problem instance, two tasks i and j are called **interchangeable**, denoted by $i \prec j$, if in any feasible schedule σ such that $S_j^\sigma \le S_i^\sigma$ holds, interchanging their start times results in a schedule σ' that satisfies all model-specific constraints and $f(\sigma') \le f(\sigma)$.

Note that interchangeability is a directed relation. Clearly, the practical meaning of this definition depends on the given scheduling model. Definitions for common scheduling models are presented in Section 5.

Definition 2. Given two tasks sets P and Q, such that both P and Q are connected and $P \cap Q = \varnothing$, the bijection $\beta : P \leftrightarrow Q$ is said to define a **progressive pair** if and only if all of the following conditions hold:

- $\forall (p,q) \in \beta: \quad p \prec q$;
- $\forall (p_1,q_1),(p_2,q_2) \in \beta: \quad (p_1 \xrightarrow{d} p_2) \Leftrightarrow (q_1 \xrightarrow{d} q_2)$;
- $\forall (p,q) \in \beta, r \notin P \cup Q: \quad (r \xrightarrow{d_1} p) \Rightarrow (r \xrightarrow{d_2}{}^{*}\ q) \wedge d_1 \le d_2$;
- $\forall (p,q) \in \beta, r \notin P \cup Q: \quad (q \xrightarrow{d_1} r) \Rightarrow (p \xrightarrow{d_2}{}^{*}\ r) \wedge d_1 \le d_2$;
- There exists no $p \in P$ and $q \in Q$ such that $q \xrightarrow{d} p$.

Definition 3. Given a progressive pair $\beta : P \leftrightarrow Q$ and a pair of tasks $(p,q) \in \beta$, the precedence constraint $p \xrightarrow{0} q$ is called a **progressive constraint** induced by β.

Proposition 1. Let $\beta : P \leftrightarrow Q$ be a progressive pair in a feasible scheduling problem. Then, the scheduling problem has an optimal solution that satisfies all the progressive constraints induced by β.

Proof: Let σ be an arbitrary optimal solution of the scheduling problem, defined in the form of start times S_p^σ of the task. Let us build the schedule σ' by letting the start times $S_p^{\sigma'} = \min(S_p^\sigma, S_q^\sigma)$ and $S_q^{\sigma'} = \max(S_p^\sigma, S_q^\sigma)$ for all $(p,q) \in \beta$. For all other tasks $t \notin P \cup Q$, let $S_t^{\sigma'} = S_t^\sigma$.

Next we show that σ' is feasible and optimal. Since only interchangeable pairs of tasks $p \prec q$ are swapped during the transformation, by the definition of interchangeability relation, σ' satisfies all model-specific constraints and $f(\sigma') \le f(\sigma)$. In order to show that precedence

constraints $p_1 \xrightarrow{d} p_2$, where $p_1, p_2 \in P$, cannot be violated in σ', let us introduce q_1 and q_2 to denote the two tasks in Q for which $\beta(p_1, q_1)$ and $\beta(p_2, q_2)$ hold. Then,

- If neither the pair (p_1, q_1), nor (p_2, q_2) were swapped, then the start times of p_1 and p_2 are unchanged in σ' w.r.t. σ;
- If the pair (p_1, q_1) was swapped, but (p_2, q_2) not, then $S^{\sigma'}_{p_1} = S^{\sigma}_{q_1} < S^{\sigma}_{p_1} \leq S^{\sigma}_{p_2} - d = S^{\sigma'}_{p_2} - d$;
- If the pair (p_2, q_2) was swapped, but (p_1, q_1) not, then $S^{\sigma'}_{p_1} = S^{\sigma}_{p_1} \leq S^{\sigma}_{q_1} \leq S^{\sigma}_{q_2} - d = S^{\sigma'}_{p_2} - d$;
- If both (p_1, q_1) and (p_2, q_2) were swapped, then $S^{\sigma'}_{p_1} = S^{\sigma}_{q_1} \leq S^{\sigma}_{q_2} - d = S^{\sigma'}_{p_2} - d$.

Outgoing precedence constraints from P are satisfied because only tasks of P were moved earlier, and tasks of Q later in the schedule. Precedence constraints incoming to P from an external task r are respected because r precedes the corresponding tasks in Q as well. The proof is analogous for precedence constraints inside Q, incoming to Q, or outgoing from Q. Finally, it is trivial for the precedence constraints between tasks of $V \setminus (P \cup Q)$, because those tasks were not moved.

The consequence of this proposition is that the progressive constraints induced by a progressive pair can be added to the scheduling problem, which reduces the search space without the risk of eliminating all optimal solutions. Note that if $(p, q) \in \beta$ and p and q require a unary resource, then stronger, end-to-start progressive constraints can be added, which can boost the performance of certain solvers further.

4 EXAMPLES OF PROGRESSIVE PAIRS

Below progressive pairs are illustrated on a simplified example from a hypothetical manufacturer of railroad cars. The manufacturer receives an order for 40 identical passenger cars and 4 identical restaurant cars. Suppose that the project graphs P and Q in Figure 1.a describe the plan for manufacturing two passenger cars, and for all $i = 1,...,5$, tasks p_i and q_i are corresponding operations on the two different cars. This implies that $\beta : P \leftrightarrow Q$ is a progressive pair, and the progressive constraints induced by β can be added to the problem. The inferred progressive constraints between corresponding tasks of P and Q are displayed in red color in the figure. Obviously, all the corresponding tasks belonging to the 40 passenger cars, and separately, to the 4 restaurant cars can be sequenced in a similar way.

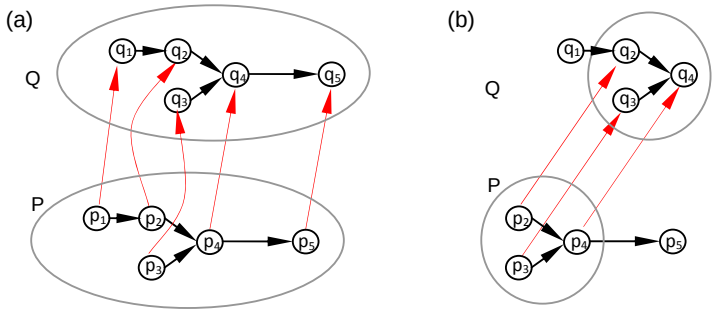

Figure 1.a and b: Examples of a progressive pair relation between identical projects (a), and between identical projects shifted in time (b).

Now, assume that rescheduling occurs after the execution of some tasks of P. Moreover, the delivery due date of some passenger cars, including Q, has been extended by some months, therefore it suffices to execute task q_5 after the end of the scheduling horizon. Again, progressive constraints can be added between corresponding tasks of P and Q, see Figure 1.b. However, some of the tasks will not have a correspondent.

In the third example shown in Figure 2.a, P represents the project plan for a passenger car, while Q describes the plan for a restaurant car. Task p_5 in P and tasks q_6, q_7, and q_8 in Q are specific for the given type of railroad car, therefore they do not have a corresponding interchangeable task in the other task set. Still, the progressive constraints displayed in the figure can be inserted between the two projects.

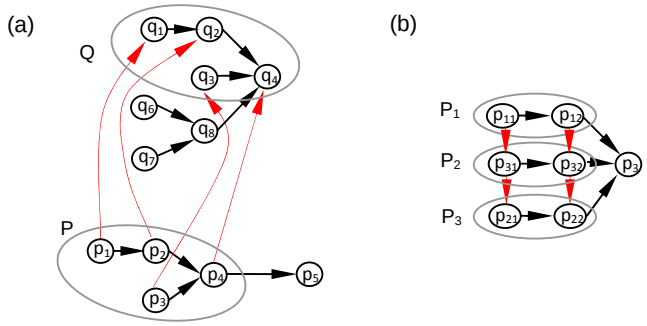

Figure 2.a and b: Progressive pair relation between two different projects (a), and between different task sets of a single project (b).

Finally, Figure 2.b shows an example of progressive pairs within a project. The different task sets P_i correspond to the assembly operations of different compartments of the same passenger car. The above results

allow one to completely order the interchangeable tasks on different compartments by progressive constraints.

5 INTERCHANGEABILITY DEFINITIONS FOR COMMON MODELS

The definition of task interchangeability differs over scheduling models. A scheduling model is composed of a set of constraints and an objective function. The interchangeability definition for a model is the conjunction of the conditions for the individual constraints and the objective function. The conditions of the interchangeability relation $i \prec j$ for common **constraints** are as follows:

- Processing times: $p_i = p_j$, where p_i is the processing time of task i;
- Resource constraints: $\forall r \in R: \rho_i^r = \rho_j^r$, where ρ_i^r is the amount of capacity that task i requires of resource $r \in R$;
- Release times: $r_i \le r_j$, where r_i is the release time of task i;
- Deadlines: $a_i \le a_j$, where a_i is the deadline of task i;
- Sequence-dependent setup times: $\forall k \quad s_{k,i} = s_{k,j} \wedge s_{i,k} = s_{j,k}$, where $s_{i,k}$ is the setup time between tasks i and k.

The conditions of the interchangeability relation $i \prec j$ for common **objective functions** are:

- Makespan, $C_{\max} = \max_i C_i$: no condition;
- Total completion time, $\sum C_i$: no condition;
- Total weighted completion time, $\sum w_i C_i$: $w_i \ge w_j$, where w_i is the weight of task i;
- Total weighted tardiness, $\sum w_i T_i$: $b_i \le b_j \wedge w_i \ge w_j$, where b_i is the due date of task i and $T_i = \max(C_i - b_i, 0)$;
- Maximum weighted tardiness, $\max w_i T_i$: $b_i \le b_j \wedge w_i \ge w_j$;
- Total weighted earliness-tardiness, $\sum_i (w_i^E E_i + w_i^T T_i)$: $b_i \le b_j \wedge w_i^E \le w_j^E \wedge w_i^T \ge w_j^T$, where w_i^E and w_i^T are the earliness and tardiness weights of task i and $E_i = \max(b_i - l_i, 0)$;
- Number of late tasks, $\sum U_i$: $b_i \le b_j$;
- Total setup cost: $\forall k \quad c_{k,i} = c_{k,j} \wedge c_{i,k} = c_{j,k}$, where $c_{i,k}$ is the cost of setup between tasks i and k.

The correctness of the above conditions can be proven easily by showing that in any schedule where task j precedes task i these two tasks can be swapped without violating any model-specific constraint or increasing

the value of the objective function. Conditions for various other constraints and objectives can be constructed in a similar fashion. Specifically, if the objective function is a monotonously increasing combination (e.g., linear combination with positive weights) of several elementary objectives from the above list, then the conjunction of the conditions for the elementary objective functions must be applied.

On the other hand, there are scheduling models where the concept of progressive pairs cannot be applied directly. These include models where some basic parameters, e.g., processing times or resource requirements are not fixed in the input of the scheduler. Some examples are batching problems (processing times are dependent on the batch sizes, which are decision variables), multi-mode scheduling problems (several execution modes exist for a task), and uniform or unrelated parallel resource scheduling problems (processing times differ by machine). In these problems, it is possible to use progressive pairs only during search, for tasks whose above parameters have been fixed.

6 APPLICATIONS OF PROGRESSIVE PAIRS

Algorithms for finding progressive pairs have been proposed first in [5]. Although that paper defined progressive pairs for one specific scheduling model only, it is straightforward to adapt the algorithms to the generic definition presented here.

Obviously, the efficiency of the proposed methods depends largely on the given application. A key factor is the degree of repetitions in the production plan, as well as the scheduling model used. In [5], the approach has been evaluated on 21 resource-constrained project scheduling problem instances for minimizing the makespan. Data originated from two sources. The first set of data derived from a manufacturer of turbine parts. Problem instances contained 585–3511 tasks, executed on ca. 100 unary and cumulative resources. The second set of problems derived from the ILOG MaScLib library [6], and contains neutralized data of an anonymous company, with 5–30 unary resources and 60–2500 tasks per instance.

The detection of progressive pairs and the insertion of progressive constraints were implemented as a preprocessing step before solving the problem by a constraint-based scheduler [7]. The performance of the scheduler has been compared on two versions of each instance, one without, and one with the progressive constraints added. The summary of the results is presented in Table 1, where each row contains combined results for one problem set. The columns display the number of tasks and resources per problem instance, the percent of instances solved to proven optimality, and the average gap between the best solution found and the best lower bound for either version of the instances. The numbers show that the usage of progressive pairs enabled us finding more optimal solutions, and better solutions for other instances, too. Namely, the gap between the solution and the lower bound was decreased by 60% on average.

Problem set	#Tasks	#Res	Without PPs		With PPs	
			Opt.	Gap	Opt.	Gap
Turbines	585–3511	95	26.67%	4.89%	33.33%	1.95%
MaScLib	60–2500	5–30	33.33%	5.79%	50.00%	2.33%

Table 1: Summary of the experimental results without and with the application of progressive pairs.

7 CONCLUSIONS

This paper introduced progressive pairs, a formal notion of similarity between repetitive task sets in a scheduling problem. Adding progressive constraints to the problem reduces the search space by ordering the corresponding tasks in the progressive pair, without losing all the optimal solutions. The proposed approach can be used for most of the common scheduling models, independently of the solution approach. In preliminary experiments on industrial data sets, the application of progressive constraints significantly reduced the gap between the solution found and the known lower bound.

8 ACKNOWLEDGEMENTS

This work has been supported by grants OMFB-01638/2009, and OTKA K76810 and T73376. A. Kovács acknowledges the support of the János Bolyai scholarship No. BO/00138/07.

9 REFERENCES

[1] Brucker, P., 2007, Scheduling Algorithms, 5th edition, Springer.

[2] Gent, I.P., Petrie, K., Puget, J-F., 2006, Symmetry in Constraint Programming, Chapter 10 in Rossi, F., Van Beek, P., Walsh, T. (eds), Handbook of Constraint Programming, Elsevier.

[3] Bender, A., 1996, Design of an Optimal Loosely Coupled Heterogeneous Multiprocessor System, European Design and Test Conference, Proceedings: 275-281.

[4] Kuchcinski, K., 1999, Synthesis of Distributed Embedded Systems, 25th Euromicro Conference, Proceedings: 1022-1028.

[5] Kovács, A., Váncza, J., 2006, Progressive Solutions: A Simple but Efficient Dominance Rule for Practical RCPSP. CPAIOR 2006, 3rd Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Proceedings (Springer LNCS 3990): 139-151.

[6] Nuijten, W., Bousonville, T., Focacci, F., Godard, D., Le Pape, C., 2004, Towards an Industrial Manufacturing Scheduling Problem and Test Bed. 9th Int. Conf. on Project Management and Scheduling, Proceedings: 162–165.

[7] Baptiste, P., Le Pape, C., Nuijten, W., 2001, Constraint-based Scheduling, Kluwer.