Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Control Engineering

Scheduling in manufacturing systems

Doctoral Thesis

Jan Kelbel

Prague, February 2012

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Control Engineering and Robotics

Supervisor: ***Doc. Dr. Ing. Zdeněk Hanzálek***

To my wife Lenka.

Acknowledgements

I would like to give my thanks to my thesis advisor Zdeněk Hanzálek for his patient leading, support and help throughout the years I spent in his research group at the Department of control Engineering. I would also like to thank to the anonymous reviewers of journals and international conferences where preliminary versions of this thesis were submitted. Their comments significantly contributed to the quality of this thesis. I would also like to express my thanks to all my colleagues for friendly discussions about the topics of combinatorial optimization. Last, but not least, many thanks belongs to all my family for the support during my work on this thesis.

This work was supported by the Ministry of Education of the Czech Republic under project 1M0567.

Goals and Objectives

This thesis is focused on combinatorial optimization problems that are present in the domain of manufacturing systems. Three different problems are studied, where with regards to computational complexity each of them belongs to the class NP-hard. The objective of each of the optimization problems is different, but the common requirement is to efficiently utilize existing manufacturing resources. For each of the three combinatorial problems, the goals of this work are set as follows:

1. Analyze and categorize the specified optimization problem.
2. Create a mathematical model of the problem.
3. Design an algorithm to solve the problem.
4. When applicable, compare the quality of the obtained solution to the existing solutions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is focused on combinatorial optimization problems that are present in the domain of manufacturing systems. Three different optimization problems are studied. The first optimization problem is an industrial case study problem of lacquer production. It is a production scheduling with earliness and tardiness penalties that reflects the scheduling part of the Just-In-Time inventory strategy. The aim of the scheduling is to create a schedule for the lacquer production, which will meet the due dates required by the customers. The storage costs for orders completed before due date and the penalty payment for not meeting the due date are considered. The second problem deals with the component reallocation problem in the Surface Mount Technology (SMT) assembly process optimization. It is aiming to improve the existing component allocation which was in this case originally created by the SMT line operators as a manual modification of the initial component allocation by the line computer optimizer. The modification was made to improve the quality of the assembly; however, it negatively affected the assembly time. This paper describes an algorithm developed as a decision support system for the SMT line operators and its purpose is to suggest changes in the existing component allocation in order to improve the production performance. The third problem deals with a scheduling problem in manufacturing of agricultural machinery. The objective is to minimize the total production time for a defined set of ordered products. The problem is categorized as a permutation flow-shop problem.

1.1 Related work

The algorithms presented in this thesis are based mainly on two approaches that will be briefly introduced in this section. The first approach is the constraint programming and the second one is the large neighborhood search. For a survey on the works related to the studied optimization problems, please refer to the introduction sections of the related chapters of the thesis (Section 2.1, Section 3.1 and Section 4.1).

The Constraint Programming (CP) approach (Dechter, 2003; Barták et al., 2009) is a method for declarative description and solving of the feasibility and optimization problems. *Constraint satisfaction* over the integer domains is a part of the constraint programming which is used to solve combinatorial problems including scheduling problems.

The constraint satisfaction problem (CSP) is defined as a triple (x, d, c), where $x = \{x_1, \ldots, x_n\}$ is a finite set of *variables*, $d = \{d_1, \ldots, d_n\}$ is a set of respective *domains* that contain possible values for each variable, $d_i = \{v_1, \ldots, v_k\}$ and $c = \{c_1, \ldots, c_t\}$ is a set of relations — called *constraints* — restricting the legal combinations of the values of the variables. Assigning the values to the variables so that all constraints are satisfied at once is one solution of the CSP.

The software systems for solving CSP (called CSP solvers) (ILOG, 2002; Gecode Team, 2010), usually employ two cooperative techniques to get a solution of CSP. *Constraint propagation*, which is the inference of the new constraints from the existing set of constraints, actually removes from the domains those values that directly violate the related constraints. Usually, constraint propagation itself is not capable of finding a solution of the CSP, so the second technique — the *search* algorithm — is used to systematically explore the search space pruned by the constraint propagation. The search consists of a search procedure (also called the labeling procedure) used to construct the search tree, and a search strategy (e.g. depth-first search) that is applied in order to explore the search tree. The search procedure typically makes decisions about the variable selection (i.e. which variable to choose) and about the value assignment (i.e. which value to assign to the selected variable). The constraint propagation and the search cooperate during the process of solving the CSP. The search decision, i.e. the assignment of the value to the chosen variable, reduces the domain of the variable, which triggers the constraint propagation of the related constraints.

The large neighborhood search (Shaw, 1998) (LNS) represents a local search method to solve combinatorial optimization problems. As any local search method, it is based on the idea of iterative improvement of the existing solution. Given the existing solution, value assignment is relaxed for a selected subset of the problem variables (i.e. the domains of the variables are restored to the initial state). The rest of the variables remain fixed at the values from the existing solution. This partially relaxed problem is then solved using a complete search method.

The difference between classical local search methods (e.g. hill climbing or tabu search) and LNS is in the construction of the neighborhood. In local search, the neighborhood is defined by local moves. On the other hand, the neighborhood in LNS is defined by all possible extensions of the fixed partial solution. The neighborhood in LNS is usually larger than the neighborhood used in local search. That is why an efficient complete search method (e.g. constraint programming, integer linear programming) is used instead of the enumeration common in local search.

1.2 Outline

This thesis is organized as follows. Chapter 2 presents the solution of the production scheduling problem with earliness and tardiness costs. The problem of reoptimization of component allocation in SMT assembly is presented in Chapter 3. Chapter 4 deals with the problem of scheduling the assembly of agricultural machinery. In the last chapter, the contributions of the thesis are concluded.

Chapter 2

Production scheduling with earliness/tardiness penalties

This chapter deals with an application of constraint programming in production scheduling with earliness and tardiness penalties that reflects the scheduling part of the Just-In-Time inventory strategy. Two scheduling problems are studied, an industrial case study problem of lacquer production scheduling, and also the job-shop scheduling problem with earliness/tardiness costs. The chapter presents two algorithms that help the constraint programming solver to find solutions of these complex problems. The first algorithm, called the cost directed initialization, performs a greedy initialization of the search tree. The second one, called the time reversing transformation and designed for lacquer production scheduling, reformulates the problem to be more easily searchable when the default search or the cost directed initialization is used. The conducted experiments, using case study instances and randomly generated problem instances, show that our algorithms outperform generic approaches, and on average give better results than other nontrivial algorithms.

2.1 Introduction

The Just-In-Time inventory strategy of supply chain management has been applied in practice since the early 1970s (Ohno, 1988). About a decade later the first formulations of the scheduling problem with earliness and tardiness cost appeared (Baker and Scudder, 1990), reflecting the scheduling part of JIT. In the problem, the earliness cost may represent the storage cost for early finished product while the tardiness cost represents the cost of a delay in the following production, for example.

This chapter describes a constraint programming approach (Barták et al., 2009) to solving scheduling problems with earliness and tardiness costs. Since the scheduling problem of minimizing the total tardiness $1 \,||\, \sum_j T_j$ is shown to be NP-hard (Du and Leung, 1990), all the earliness/tardiness problems except some special cases are NP-hard too. In the related work, we will include only those papers related to the same or similar problems to the ones we are dealing with. For more information on earliness/tardiness scheduling, we refer to the reviews of (Hoogeveen, 2005; Baker and Scudder, 1990).

In the chapter we introduce two scheduling problems with earliness/tardiness costs. The first problem, introduced in (Beck and Refalo, 2003), is a job shop scheduling problem with earliness and tardiness costs related to the completion time of each job. This problem is solved in (Beck and Refalo, 2003) using a hybrid approach based on a probe backtrack search (El Sakkout and Wallace, 2000) with the integration of constraint programming and linear programing. This hybrid approach performed significantly better than the generic (naïve) CP and MIP algorithms. With another hybrid approach, combining local search and linear programming in (Beck and Refalo, 2002), the results obtained were slightly worse than in (Beck and Refalo, 2003). The large neighborhood search (Danna and Perron, 2003) applied to the same earliness/tardiness job shop problem outperformed both hybrid approaches of Beck and Refalo.

The second problem is a lacquer production scheduling problem, an industrial case study introduced in the project Ametist (AMETIST, 2002), where it was solved using a timed automata approach (Behrmann et al., 2005). This problem can be classified as a resource-constrained project scheduling problem, more general than the job shop scheduling problem, with distinct due dates and release dates, and with job dependent earliness and tardiness costs. In comparison to the first problem, are not only dedicated resources there, but also groups of parallel identical resources. Next, the problem includes changeover times, breaks on resources and breakable tasks. The problem similar to the lacquer production scheduling is, for example, solved in (Luh P. B. et al., 1998), where a combination of Lagrangean relaxation, dynamic programming and heuristics is used.

Our motivation was to find a result for the lacquer production scheduling problem that would outperform the original timed automata solution. The second objective was to compare our method with the already existing algorithms using benchmark instances of the earliness/tardiness job-shop problem. To solve the two earliness/tardiness scheduling problems, we used the constraint programming approach, where the user declares a model of the problem, applies a generic search procedure, and obtains a result. However, for these problems, we had to develop two algorithms that helped the constraint programming solver to find solutions.

The first algorithm is a search tree initialization procedure for the earliness/tardiness scheduling problems, which initially assigns those values to the variables that leads to a solution with minimal local cost. For lacquer production scheduling, considering the structure of the problem, we also developed a time reversing transformation algorithm that transforms the problem to a formulation, which is more suitable for searching using the default search procedure of the used constraint programming system.

The proposed search tree initialization procedure is tested on a set of randomly generated instances of the job shop scheduling problem with earliness and tardiness costs. It significantly outperforms simple (default) models introduced in (Beck and Refalo, 2003), and on average it gives better results than the Unstructured Large Neighborhood Search (Danna and Perron, 2003).

For lacquer production scheduling, the search tree initialization and the time reversing transformation was applied to solve the original case study problem instance (Behrmann et al., 2005), and we were able to find a solution with more than a 60% better (lower) cost than the one computed by the timed automata. Next, our approach was tested on a set of randomly generated instances with the structure of the case study instance.

This work contains two main contributions: the search tree initialization procedure for earliness/tardiness scheduling problems, and the time reversing transformation considering the structure of the lacquer production scheduling problem.

The chapter is organized as follows: In the next section, the earliness/tardiness scheduling problems are formulated. The section "Solving techniques" describes the constraint programming approach to solving the scheduling problems. The section also includes the description of the two algorithms. The section “Experimental Results” contains the description of problem instances we used for computational experiments, and the results.

2.2 Scheduling problem formulation

2.2.1 Earliness tardiness job-shop scheduling problem

The definition of the earliness tardiness job-shop scheduling problem (ETJSSP) is based on (Beck and Refalo, 2003). We assume a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ where job J_j consists of a set of tasks $\mathcal{T}_j = \{T_{j,1}, \ldots, T_{j,n_j}\}$. Each task has a given processing time $p_{j,i}$, and required dedicated unary resource from a set $\mathcal{R} = \{R_1, \ldots, R_m\}$. The start time $S_{j,i}$ of all the tasks determine the result of the scheduling problem. The completion time of a task is defined as $C_{j,i} = S_{j,i} + p_{j,i}$. For each job J_j there are precedence relations between tasks T_i and T_{i+1} such that $C_{j,i} \leq S_{j,i+1}$ for all $i = 1, \ldots, n_j - 1$.

Concerning earliness and tardiness costs, each job has a due date d_j assigned to it, i.e. the time when the last task of the job should be finished. In general, the due dates are distinct. The cost function of job J_j is defined as $\alpha_j(d_j - C_{j,n_j})$ for an early job and $\beta_j(C_{j,n_j} - d_j)$ for a tardy job, where α_j and β_j are the earliness and tardiness costs of the job per time unit. Taking both alternatives into account, the cost function of the job can be expressed as

$$f_j = \max(\alpha_j(d_j - C_{j,n_j}), \beta_j(C_{j,n_j} - d_j)). \tag{2.1}$$

An optimal solution of the ETJSSP is the one with the minimal possible sum of the costs over all jobs

$$\min \sum_{J_j \in \mathcal{J}} f_j.$$

In this article, a specific ETJSSP will be considered in order to be consistent with the original problem instances (Beck and Refalo, 2003). All jobs have the sets of tasks with the same cardinality, which is equal to the number of resources, i.e. $n_j = m$ for all j. Each of the n_j tasks of the job is processed on a different resource. Next, the problem has a work flow structure: the set of resources $\mathcal{R}$ is partitioned into two disjunctive sets $\mathcal{R}_1$ and $\mathcal{R}_2$ of about the same cardinality, and the tasks of each job must use all resources from the first set before any resource from the second set, i.e.

task $T_{j,i}$ for all $i = 1, \dots, |\mathcal{R}_1|$ requires a resource from set $\mathcal{R}_1$, and task $T_{j,i}$ for all $i = |\mathcal{R}_1| + 1, \dots, n_j$ requires a resource from set $\mathcal{R}_2$.

2.2.2 Lacquer production scheduling problem

The lacquer production scheduling problem is an industrial case study introduced in the project AMETIST (AMETIST, 2002). Each order of the lacquer to be produced — that is a job in terms of scheduling — is specified by quantity of the produced lacquer, release date (earliest start time of the production), due date, earliness and tardiness costs, and the type of lacquer. The type of the lacquer determines the recipe used for the job. The recipe defines the production steps (called tasks) of the job. The definition includes the resources required by each task, the processing times of the tasks (related to the quantity of lacquer), and the precedence constraints. The lacquer production scheduling problem includes three recipes for metallic, bronze and uni lacquers.

The main difference between the lacquer production scheduling problem and the ETJSSP is that the resources in the case study behave in a more realistic way. The resources operate in shifts and tasks cannot be scheduled during breaks, e.g. for a resource operating in two eight-hour shifts during the night. Some of the tasks are breakable, i.e. allowed to be interrupted by a break, but not preempted by another task. In addition, there is a changeover time for the resource filling station G_4 – see below.

The resources are grouped according to their types. Some of these groups contain more than one machine, i.e. there are more machines of that type available. Inside the group, the machines are parallel identical resources in terms of the scheduling theory (Blazewicz et al., 2001), or in terms of constraint-based scheduling (Barták et al., 2009), each group is the cumulative resource with the capacity equal to the number of machines in the group.
The groups of parallel identical resources are:

- The mixing vessel $G_1 = \{R_{11}, R_{12}, R_{13}\}$ is used for the production of the metallic and bronze lacquers.
- The mixing vessel $G_2 = \{R_{21}, R_{22}\}$ is used for the production of the uni lacquers.
- The dose spinner $G_3 = \{R_{31}, R_{32}\}$ is used in all recipes.

- The filling station $G_4 = \{R_{41}, R_{42}\}$ is used in all recipes. This is the resource with sequence-dependent changeover time – cleaning is needed when two successive jobs are of a different lacquer type, and the changeover time depends on the types of the jobs. Due to the changeover time, we need to distinguish the two filling stations. Therefore, G_4 is not a cumulative resource, but a group of alternative resources.

The dedicated resources are:

- The disperser $G_5 = \{R_5\}$ is used in the uni lacquer recipe.
- The dispersing line $G_6 = \{R_6\}$ is used in the uni lacquer recipe.
- The bronze mixer $G_7 = \{R_7\}$ is used in the bronze lacquer recipe.
- The bronze dose spinner $G_8 = \{R_8\}$ is used in the bronze lacquer recipe.

Finally, there is an unrestricted resource, laboratory G_9, which is used in all recipes. The recipes for metallic, bronze and uni lacquers are depicted in Figure 2.1 with example processing times. Five production steps can be identified in the lacquer production (Loeschmann and Ludewig, 2003): In step 1 and 2, pre-dispersion and dispersion, solid and liquid input materials and solvents are prepared for actual production. This operation is only used in the production of the uni and bronze lacquers and it utilizes resources G_5, G_6, G_7 and G_8. Then, in step 3, the input materials are filled into mixing vessel G_1 or G_2 (depending on the type of the lacquer) using dose spinner G_3. Each dose spinner contains 200 valves injecting predefined quantities of the materials into the mixing vessels. After the required quantities are injected, the mixing of the materials continues for a defined period of time. When the mixing procedure is finished, the lacquer quality is checked in laboratory G_9. Step 4 is necessary if the lacquer quality requirements are not satisfied. The mixing vessel is once again moved under the dose spinner and the dosing procedure is repeated. The quality is checked again after the mixing procedure. In this case study, step 4 is always required. In step 5, the mixing vessel is emptied of the prepared lacquer at filling line G_4. When the filling operation is finished, the mixing vessel is cleaned.

The objective of the lacquer production scheduling is to minimize a combination of total weighted earliness (the cost for storage of orders that are

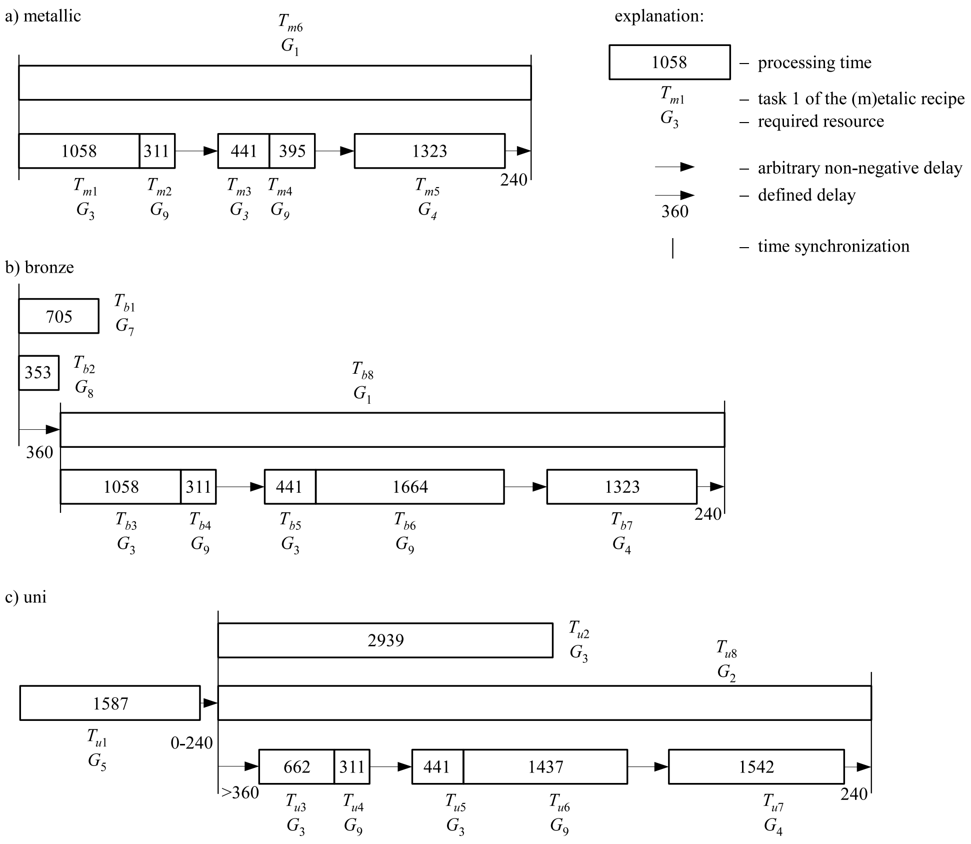


Figure 2.1: The recipes for the lacquer production.

finished too early) and total weighted tardiness (the penalty payment for delayed orders):

$$\min \sum_{j \in \mathcal{J}} \max(\alpha_j(d_j - C_j), \beta_j(C_j - d_j)), \tag{2.2}$$

where $\mathcal{J}$ is the set of jobs, d_j is the due date and C_j is the completion time of job j. α_j and β_j are the unit earliness and tardiness costs.

2.3 Solving techniques

The two earliness/tardiness scheduling problems were solved using the Constraint Programming (CP) approach.

2.3.1 Modeling earliness tardiness job-shop scheduling problem

Our approach to modeling ETJSSP is quite straightforward and is based on a usual constraint programming model for scheduling problems. The scheduling problem is modeled directly by using the formulation from the previous section, yet by using higher abstraction objects for scheduling (e.g. tasks and resources) available in the used CSP solver[1]. The precedences are declared as linear algebraic constraints between the variables representing the start times of the tasks, and also using the precedence graph constraint (Laborie, 2003). The disjunctive edge-finder (Carlier and Pinson, 1990) propagation algorithm is used for resource constraints, i.e. when we need to assure that, at most, one task is processed by one resource each time. The used CSP solver showed better performance of the computations when the cost function (2.1) was expressed as $f_j \geq \alpha_j(d_j - C_{j,n_j}) \wedge f_j \geq \beta_j(C_{j,n_j} - d_j)$.

To find a solution of the ETJSSP, we designed a search procedure called the cost directed initialization, which is described in the next subsection.

2.3.2 Cost directed initialization

Most of the constraint programming solvers have a default search procedure that builds the search tree by assigning the values from the domains to the variables in increasing order. For scheduling problems, the CSP solver used for experiments in this work employs the ranking search procedure (Baptiste et al., 1995) as a default, which is supposed for makespan minimization.

The idea of our search procedure is based on the fact that only C_{j,n_j}, the completion time of the last task of the job, influences the value of the cost function, and that the values of C_{j,n_j} inducing the lowest values of the cost functions f_j should be examined first.

[1]ILOG OPL Studio in version 3.6 (ILOG, 2002) was used as a CSP solver for the experiments in this work.

Algorithm 2.1 – CDI search procedure

```
1. Sort the last tasks of the jobs, $T_{j,n_j}$ for all $j$, according to the nondecreasing
domain size of $C_{j,n_j}$.
2. For each task from the sorted list from the domain of $C_{j,n_j}$ select a value
$v_j$ leading to the minimal $f_j$ and create two alternatives in the search tree:
    • $C_{j,n_j} = v_j$
    • $C_{j,n_j} \neq v_j$
3. Continue with the ranking search procedure for all variables.
```

The search procedure, which is denoted as the cost-directed initialization (CDI), performs as described in Algorithm 2.1: Variables representing the completion time C_{j,n_j} are selected in increasing order of the size of their domains (step 1). The value selection (step 2) is made according to the lowest possible value of the cost function f_j (2.1) and two alternatives are created in the search tree. In the first alternative, the variable C_{j,n_j} is assigned the value inducing the lowest earliness/tardiness cost for job j. As the second alternative in the search tree, this value is disabled. This is only done once for each task T_{j,n_j}, i.e. the CDI procedure creates only the first n levels of the search tree. Then, the search continues with the default search procedure, which is the ranking (step 3).
The CDI search procedure is a complete algorithm, which can be easily proven: in step 2 of the algorithm, the whole domain of each variable C_{j,n_j} is covered. All other variables are labeled in the third step by the ranking procedure.

Slice Based Search, available in (ILOG, 2002), based on (Beck and Perron, 2000), and similar to Limited Discrepancy Search (Harvey and Ginsberg, 1995) is used as a search strategy to explore the search tree created by the CDI procedure. This is necessary for obtaining good performance, as using depth first search instead, the algorithm was not capable of finding any solution for about 50% of the used instances of the ETJSSP.

2.3.3 Solving the lacquer production scheduling problem

The lacquer production scheduling requires some more complex constraints on tasks and resources, which will be described in this subsection. The variables and constraints of the lacquer production CSP model are declared

Example 2.1 – precedence and delay constraints of the metallic recipe

Job 1: production of metallic lacquer.
Variables: $S_{1,1}$, $S_{1,2}$, $S_{1,3}$, $S_{1,4}$, $S_{1,5}$, $S_{1,6}$ and $C_{1,6}$.
Precedence and delay constraints:
$S_{1,1} + p_{1,1} = S_{1,2}$
$S_{1,2} + p_{1,2} \leq S_{1,3}$
$S_{1,3} + p_{1,3} = S_{1,4}$
$S_{1,4} + p_{1,4} \leq S_{1,5}$
$S_{1,6} = S_{1,1}$
$C_{1,6} = S_{1,5} + p_{1,5} + 240$

according to the specification of the jobs in the problem instance. Example 2.1 illustrates the declaration of the precedence and delay constraints of one job using the metallic recipe from Figure 2.1.

The resource constraints are also declared according to the recipe. For the cumulative resources, i.e. a set of parallel identical resources, the cumulative edge-finder (Nuijten and Aarts, 1996; Mercier and Van Hentenryck, 2008) propagation algorithm is used.

Some of the resources operate in shifts, so it is necessary to model breaks. The used CSP solver supports breaks on disjunctive (i.e. dedicated) resources. Breakable tasks, which can be interrupted during breaks, are also supported by the solver (ILOG, 2002).

Breaks on the cumulative resources are not supported directly. One possibility to model this feature is to replace the cumulative resource by a set of alternative dedicated resources, where the breaks and the breakable tasks are supported. Then the task will be processed by one of the resources from the set, and the assignment is chosen during the solving process. However, it is more computationally efficient to use a cumulative resource with the cumulative edge-finder propagation algorithm, and to add constraints that disable the execution of the tasks during breaks. For non-breakable tasks, it is made by removing the appropriate time intervals from the domain of the start time variable. For breakable tasks, conditional constraints are added such that if the start time of the task is in the specified interval, then the processing time of that task is extended by the duration of the break. That is, the task executes during the break also, but in that case its processing time is adequately extended.

Due to the size of the lacquer production scheduling case study, it was necessary to make additional modifications in the CSP model to obtain a solution of instances of a size greater than 5 jobs in an acceptable time. The mixing vessel resource (G_1 or G_2) is needed for nearly the whole production time of each job. To use this resource, a shadow task (T_{m6}, T_{b8} or T_{u8}) with the variable processing time is created. The mixing vessel is a restricted resource, so it is necessary not only to finish the job near its due date but also to minimize the processing time of the shadow task to release the mixing vessel for the other jobs. This makes a major difference in comparison with the ETJSSP. Next, in the case study problem instance, the unit tardiness cost β_j is about 50 times larger than the earliness cost α_j for most of the jobs. This disproportion is usual in industry if the production is a part of customer's supply chain, for example if we supply lacquers to a car manufacturer.

Summing it up, we need to find a schedule, where the jobs complete close to their due dates while preferring to finish before the due date, and also where the processing time of the tasks requiring the mixing vessel resource is minimized. Instead of designing a new search procedure we chose to transform the problem and to use the cost directed initialization (Algorithm 2.1). This transformation, described in Algorithm 2.2, is based on reversing the direction of time. The constants and variables in the transformed formulation are denoted by prime notation (e.g. S'_j).

The *target start time* $\hat{r}'_j$ of job j is introduced in the transformed formulation, and it corresponds to the due date of the original formulation. To make the search space smaller, deadline $\tilde{d}_j$ is defined for each job by the maximal allowed weighted tardiness cost $\overline{f}$, that is found experimentally during the solution of the transformed problem. The introduced deadlines are more constraining for jobs with a large β_j, and possibly discard optimal solutions. However, this modification helps to find a good solution in reasonable time.

The objective of the lacquer production scheduling (2.2) is transformed using Algorithm 22 to the formulation

$$\min \sum_{j \in \mathcal{J}} \max(\beta_j(\hat{r}'_j - S'_j), \alpha_j(S'_j - \hat{r}'_j)), \tag{2.3}$$

i.e. in the transformed problem, the objective depends on the distance be-

Algorithm 2.2 – time reversing transformation

```
1. Calculate the upper bound of the schedule makespan: $UB$
2. Calculate the job deadlines:    $\tilde{d}_j = d_j + \bar{f}/\beta_j$
3. For each job $J_j$ calculate the new values of:
   a) the target start time $\hat{r}'_j = UB - d_j$
   b) the release time $r'_j = UB - \tilde{d}_j$
   c) the deadline $\tilde{d}'_j = UB - r_j$
4. Calculate the new start and completion times for all breaks on the resources (similarly to step 3.).
5. Reformulate the precedence and delay constraints by substitution:
   $S_{j,k} = UB - C'_{j,k}$    $C_{j,k} = UB - S'_{j,k}$
```

Example 2.1 (cont.) – applying time reversing transformation

Job 1: production of the metallic lacquer.
Variables: $C'_{1,1}$, $C'_{1,2}$, $C'_{1,3}$, $C'_{1,4}$, $C'_{1,5}$, $C'_{1,6}$ and $S'_{1,6}$.
Precedence and delay constraints:

$S_{1,1} + p_{1,1} = S_{1,2} \quad \longrightarrow \quad C'_{1,1} - p_{1,1} = C'_{1,2}$
$S_{1,2} + p_{1,2} \leq S_{1,3} \quad \longrightarrow \quad C'_{1,2} - p_{1,2} \geq C'_{1,3}$
$S_{1,3} + p_{1,3} = S_{1,4} \quad \longrightarrow \quad C'_{1,3} - p_{1,3} = C'_{1,4}$
$S_{1,4} + p_{1,4} \leq S_{1,5} \quad \longrightarrow \quad C'_{1,4} - p_{1,4} \geq C'_{1,5}$
$S_{1,6} = S_{1,1} \quad \longrightarrow \quad C'_{1,6} = C'_{1,1}$
$C_{1,6} = S_{1,5} + p_{1,5} + 240 \quad \longrightarrow \quad S'_{1,6} = C'_{1,5} - p_{1,5} - 240$

tween the start times and the target start times, and the earliness and tardiness costs are actually swapped.

The lacquer production scheduling problem is solved in three steps. The problem is transformed using Algorithm 2.2, then solved using the Cost Directed Initialization (Algorithm 2.1), and the resulting start and completion times are transformed back using step 5 of Algorithm 2.2.

The continuation of Example 2.1 shows the application of Algorithm 2.2 on the precedence and delay constraints of the metallic recipe.

2.4 Experimental results

2.4.1 Earliness tardiness job-shop scheduling problem

The proposed Cost Directed Initialization (CDI, Algorithm 2.1) was tested against:

MIP a mixed integer programming model with disjunctive formulation (Beck and Refalo, 2003)

ST a constraint programming model with a *SetTimes* heuristic as a search procedure and a depth-first search as a search strategy (Beck and Refalo, 2003)

uLNS Unstructured Large Neighborhood Search (Danna and Perron, 2003)

where MIP and ST are simple generic models used in (Beck and Refalo, 2003) for performance comparison. The uLNS algorithm is used in a mixed integer programming solver solving the MIP disjunctive formulation.

Benchmarks are randomly generated instances of the ETJSSP according to Section 6.1 in (Beck and Refalo, 2003). The problem instances have the work flow structure. The processing times of the tasks are uniformly drawn from the interval $[1, 99]$. Considering the lower bound tlb of the makespan of the job-shop according to (Taillard, 1993), and a parameter called the looseness factor lf, the due date of the job was uniformly drawn from the interval $[0.75 \cdot tlb \cdot lf, 1.25 \cdot tlb \cdot lf]$. The job-shops were generated for three $n \times m$ sizes, $10 \times 10, 15 \times 10$, and 20×10, and for $lf \in \{1.0, 1.3, 1.5\}$. Twenty instances were generated for each (lf,size) combination.

The experiments were executed using ILOG OPL Studio 3.6 with ILOG Solver and Scheduler for the CSP models, and ILOG Cplex 9.1 for the MIP models. The Unstructured Large Neighborhood Search (uLNS) (Danna and Perron, 2003) was used by enabling the Relaxation Induced Neighborhood Search (RINS) via a `IloCplex::MIPEmphasis=4` switch in Cplex 9.1 (Danna et al., 2005; ILOG, 2005). All algorithms were running on a PC with AMD Opteron 248 CPU at 2.2 GHz with 2 GB of RAM. The time limit for each computation was 600 s, after which the execution was stopped, and the best solution up to that time was returned.

Table 2.1 shows the average ratio of the costs of the best solutions obtained by the MIP, uLNS, and ST to the best solutions obtained by the CDI, for all types of instances.

Table 2.1: Average ratio for the best values of the cost functions of the solutions found within 600 s

size	10×10			15×10			20×10		
lf	MIP/CDI	uLNS/CDI	ST/CDI	MIP/CDI	uLNS/CDI	ST/CDI	MIP/CDI	uLNS/CDI	ST/CDI
1.0	1.8	1.2	2.6	4.7	3.1	6.2	5.3	4.9	6.7
1.3	4.8	1.8	9.2	18.4	5.3	28.3	14.0	14.3	25.8
1.5	3.8	2.1	8.1	7.9	1.9	37.9	5.5	5.7	50.6

In Table 2.1 we observe that for the generated problem instances, the CDI algorithm finds the solutions with the best cost on average. The second place belongs to the uLNS algorithm. The uLNS outperformed the generic MIP approach, which corresponds to the findings of (Danna and Perron, 2003). Table 2.1 also shows that the difference between the CDI and the other approaches is bigger for larger problem instances. Further on, in Tables 2.2 and 2.3 the ST algorithm will not be included due to its poor performance.

Table 2.2 shows the number of instances solved to optimality within the 600 s time limit, and also the number of instances, for which the algorithm proved the optimality of the solution. A solution found by the CDI model is considered as the optimal solution when the value of the objective function was equal to the one of the proven optimal solution found by the MIP models or to a lower bound found by the MIP.

Table 2.2 shows that the CDI found the most optimal solutions among the used algorithms, however the difference is small. The CDI usually needed less time than the MIP or uLNS to find a solution with the optimal cost. On the other hand, in many cases the CDI was not able to prove its optimality, which can be observed in the table.

Table 2.2: The number of optimal solutions (F)ound and (P)roven within 600 s (among 20 instances)

size	10×10						15×10						20×10					
lf	MIP		uLNS		CDI		MIP		uLNS		CDI		MIP		uLNS		CDI	
	F	P	F	P	F	P	F	P	F	P	F	P	F	P	F	P	F	P
1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1.3	0	0	0	0	0	0	1	1	2	2	5	0	0	0	0	0	0	0
1.5	7	7	9	9	10	4	5	5	9	9	12	3	3	3	4	4	5	3

Table 2.3 is inspired by (Beck and Refalo, 2002). For each problem instance, the lowest cost obtained by any of the used algorithms is selected. Then, Table 2.3 contains the number of instances for which the algorithm found the solution with the best cost, i.e. equal to the lowest cost, and the number of solutions with uniquely best cost, i.e. if no other algorithm has found a solution with the same or lower cost. Here we can see that CDI found the majority of the best and uniquely best solutions among the used

algorithms. The difference is larger for the problem instances with a smaller looseness factor. In line with the observation from Table 2.1, we can say that the power of the CDI is more visible for harder problem instances.

Table 2.3: The number of (B)est and (U)niquely best solutions found within 600 s

size	10 × 10						15 × 10						20 × 10					
lf	MIP		uLNS		CDI		MIP		uLNS		CDI		MIP		uLNS		CDI	
	B	U	B	U	B	U	B	U	B	U	B	U	B	U	B	U	B	U
1.0	0	0	5	5	15	15	0	0	0	0	20	20	0	0	0	0	20	20
1.3	0	0	2	2	18	18	1	0	2	0	20	18	1	1	1	1	18	18
1.5	8	0	12	0	20	8	5	0	14	3	16	6	3	0	7	4	16	12

2.4.2 Lacquer production scheduling case study

The case study instance originates from the AMETIST project (AMETIST, 2002), where it was denoted as the *extended case study* with *performance factors*. It contains 29 jobs. The tasks requiring an unrestricted resource laboratory can be replaced by delay constraints. Then, the problem instance includes 139 tasks in total (of which 110 are breakable). The scheduling horizon of the case study instance is 9 weeks in 1 minute resolution, but the time available for each job is about 2 weeks from the release date to the due date, which makes the search space smaller. The due dates of the jobs are almost evenly distributed from the 3^{rd} to the 9^{th} week.

Three CSP algorithms and the original timed automata result (TA) from the project AMETIST (AMETIST, 2002) are compared in the experiments. All three CSP algorithms have the same CSP model for the problem using the time reversing transformation (Algorithm 2.2) and differ only in the search algorithm. Algorithm 2.1, the cost directed initialization (CDI) is compared to the ranking (the default search procedure) with a slice based search (SBS) or depth-first search (DFS). The time limit for each experiment was 600 s, and then the best solution found up to that point was returned. The results are presented in Table 2.4, where the time needed to find the solution is located in the CPU column.

The constraint programming experiments were run on an AMD Opteron 2.2 GHz CPU with 2 GB of RAM, using OPL Studio 3.6 under Windows XP. The timed automata experiments were run on an Intel P4 Xeon 2.6GHz (Behrmann et al., 2005).

Table 2.4: Results for the case study instance

	CDI	SBS	DFS	TA
Cost	777,249	1,324,253	1,669,999	≈ 2,100,000
CPU time [s]	331.4 s	372.7 s	3.2 s	≈ 600 s

The biggest impact on the performance of the application of the CSP to the lacquer production scheduling has the time reversing transformation (Algorithm 2.1), as was already mentioned in the previous section. Comparing the results in Table 2.4, we can see that all three CSP search algorithms outperformed the original timed automata approach. CDI, which was the best performing CSP search algorithm, found a solution with a cost more than 60 % better that the original TA result.

2.4.3 Random instances based on the lacquer production case study

The purpose of this experiment is to compare the performance of the CDI (Algorithm 2.1) to the default search procedure ranking on a set of random problem instances. The time reversing transformation (Algorithm 2.2) is used with both search procedures. The random instances are based on the structure of the case study instance. Jobs have a randomly assigned recipe to one of the three recipes depicted in Figure 2.1. The release date r_j of the job J_j is uniformly drawn from the chosen time interval (which is specified later in this section), the due date is specified as $d_j = r_j + 2$ weeks. The amount of lacquer to produce is uniformly drawn from the interval [5000, 20000] units. This value determines the processing time of the tasks, and the earliness and tardiness costs.

Two sets, each of 50 random instances, were generated as follows:
1) 30 jobs executed in the time interval of 9 weeks (similar to the size of the case study instance)

2) 18 jobs executed in the time interval of 6 weeks.

The instances have been solved by the following three search algorithms:
DFS the default search procedure (ranking) with a depth-first search as a search strategy
SBS the default search procedure (ranking) with a slice-based search
CDI the cost directed initialization (Algorithm 2.1)

The results are depicted in Figure 2.2. The percentage of instances, for which the first solution was found by the given time, is shown in Figure 2.2 (a) for the set of instances 1) and in Figure 2.2 (c) for set 2). For the second comparison, the best solution obtained by the given time was selected and then the ratio of the cost function of the given solution to the one of the best solution was computed. The average of these ratios is shown in Figure 2.2 (b) for set 1) and in Figure 2.2 (d) for set 2).

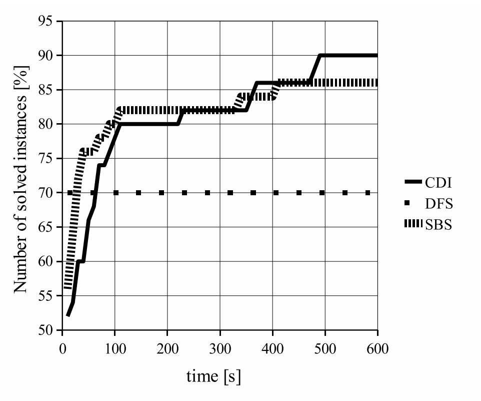
The results illustrated in Figures 2.2 (b) and (d) show that the CDI algorithm is capable of finding the solution with the best cost function among the used search algorithms. In Figures 2.2 (a) and (c) we observe that the CDI needs more time than the other two search algorithms to find the first solution. The results show that the SBS is in second place with regards to performance, while the DFS does not provide satisfactory results. The difference in performance of the CDI and SBS is more apparent for larger problem instances.

2.5 Conclusion

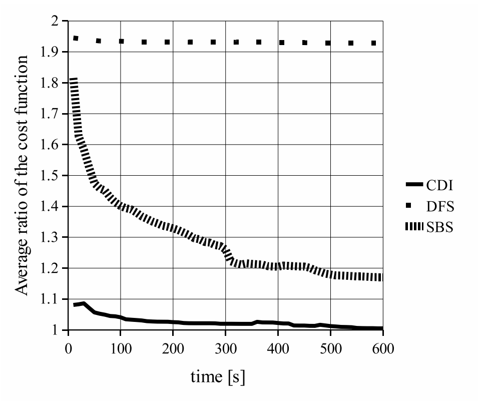
This chapter presented two algorithms for solving earliness/tardiness scheduling problems, which are designed to be applied within a constraint programming system.

For an earliness tardiness job-shop scheduling problem, a greedy search tree initialization algorithm called the cost directed initialization (CDI, Algorithm 2.1) is presented. The performance of the algorithm is compared to a mixed integer programming model (MIP), a mixed integer programming with unstructured large neighborhood search (uLNS) and to a simple constraint programming model with a SetTimes search heuristic (ST). The experiments, on randomly generated benchmark instances, show the efficiency of the cost directed initialization. Within time limit of 600 s, the CDI is capable of finding a solution that is usually better than the one found by any of the MIP, uLNS, or ST. With respect to the number of solutions with the best obtained value of the cost function, the CDI algorithm also performed better than the other algorithms. However, the weak point of the CDI is that the optimum, even if it is found, is usually not proven within the given time limit.

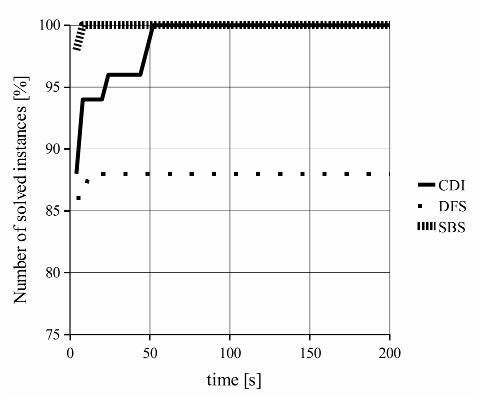
When solving the lacquer production scheduling problem, the usage of the time reversing transformation (Algorithm 2.2) had the greatest impact on the performance. Without this transformation, we were able only to solve instances of a size up to 5 jobs. The conducted experiments showed us that also adding the cost directed initialization helped to solve the majority of the randomly generated problem instances. For the lacquer production scheduling case study instance, the combination of the time reversing transformation and the CDI found a solution more than 60% better than the timed automata solution (AMETIST, 2002).



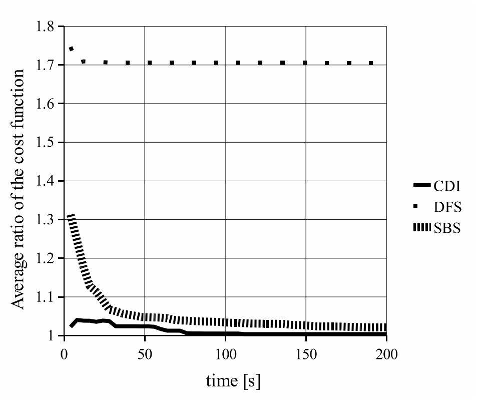
(a) The first solutions obtained for instance set 1)



(b) The average ratio of the cost functions for instance set 1).



(c) The first solutions obtained for instance set 2).



(d) The average ratio of the cost functions for instance set 2).

Figure 2.2: The results for random instances of the lacquer production scheduling

Chapter 3

Reoptimizing component allocation in surface mount technology assembly system

The component allocation problem, which is a part of the Surface Mount Technology (SMT) assembly process optimization, is a problem of distributing the components to the multiple machines at the assembly line. This chapter deals with the component reallocation problem aiming to improve the existing component allocation which was in this case originally created by the SMT line operators as a manual modification of the initial component allocation by the line computer optimizer. The modification was made to improve the quality of the assembly; however, it negatively affected the assembly time. This chapter describes an algorithm developed as a decision support system for the SMT line operators and its purpose is to suggest changes in the existing component allocation in order to improve the production performance. To minimize the impact of the optimization on the assembly quality, the number of changes in allocation is minimized and the reallocation of some components is not allowed. The SMT line considered in this work is equipped with two dual-delivery multi-station placement machines. The proposed algorithm is based on the constraint programming approach and the large neighborhood search. The problem-specific search procedure and the nozzle selection constraint are presented. Performance of the algorithm is evaluated by the optimization of one product type assembly and on a set of randomly generated problem instances following the structure of the production problem instance.

3.1 Introduction

In the Surface Mount Technology (SMT) assembly process, usually the placement machines are—because of their high purchase cost—the limiting resource of the production. Therefore, the optimization of the operation of placement machines is necessary, and placement machine manufacturers deliver software for some kind of this optimization. Yet, these tools are sometimes not able to fulfill all the needs of the SMT line operators.

The problem in scope of this work came out from cooperation with an automotive industry company producing power steering controllers. The company uses an SMT line computer optimizer developed by the manufacturer of the placement machines to compute the initial solution of the component placement optimization. This initial *setup* is implemented at the SMT line (i.e. appropriate components are loaded to component feeders, etc.) and a production trial is carried out. Then, the SMT line operators analyze the quality of the placement operations and make manual changes in the setup to reduce the number of defects in component placement. However, these modifications, which include changes in allocation of components to machines and changes in assignment of nozzle types to components, have a negative effect on the balance of the placement line. The following re-optimization does not focus solely on improvement of the workload balance; the second objective of the problem is to minimize the number of changes in allocation with respect to the manually modified setup, since each of the changes can again decrease the quality of the component placement process. The line computer optimizer is not suitable for this re-optimization task due to stochastic nature of the used algorithm. The problem described and solved in this chapter is specified as follows: for an existing setup, find a setup with lower cycle time, i.e. with workload more uniformly distributed among placement machines, while the number of differences in allocation between the setups is minimized.

The optimization of printed circuit board (PCB) assembly system is a complex optimization problem consisting of a number of related problems. These problems are categorized into three major hierarchically related subproblems (Ball and Magazine, 1988; Ammons et al., 1997). At the highest level, the *grouping* problem consists of selecting machine groups in the set of available machines, selecting printed circuit board families in the set of produced PCBs, and assigning the PCB families to machine groups. The *allocation* problem at the middle level comprises allocation of component types to placement machines, for each particular PCB family and the related machine group. At the lowest level, the *assignment* of component feeders to slots of the machine and the *sequencing* of the placement operations is performed. Due to problem complexity, the typical approach is to solve each of the three sub-problems separately.

This work focuses on the allocation problem. Therefore, the problem of re-optimizing the setup is denoted as the *component reallocation problem.* Of the two remaining subproblems, the sequencing problem is left to be solved consequently using the line computer optimizer, while the solution of the grouping problem is already determined by the initial setup. In our case, the production of PCBs for power steering controllers is a High-Volume Low-Mix production, i.e. a small range of PCB types is produced in large series. In such situations, the workload balancing of the assembly line is a more critical issue than the length of the changeover time required to reconfigure the assembly line for a production of a different PCB type (Kulak et al., 2008). Considering this, the single unique setup strategy (Ammons et al., 1997) is chosen, where a single setup is used for a card family containing only one PCB type.

The allocation problem has been solved using various approaches. In Ball and Magazine (1988) the allocation problem is formulated as a bin packing problem, assuming the time to pick and place each component is constant. Ammons et al. (1997) used a list processing heuristic similar to the best-fit-decreasing for bin packing, and also a mixed integer linear programming approach. The time to pick and place is estimated for each type of component. Kodek and Krisper (2004) applied a branch and bound algorithm to the allocation problem, balancing a heterogeneous assembly line where the assembly time is specified for each combination of component type and machine. Sun et al. (2005) used an iterative algorithm, where the component allocation is solved by a genetic algorithm, and the solution is passed on to a greedy heuristic for evaluation of the workload. A two step genetic algorithm approach is described in Kulak et al. (2008) where a set of candidate solutions to the allocation problem is generated, and these solutions are consequently used as an input for the sequencing problem algorithm determining the actual cycle time for the assembly line. Then, the best solution is selected. In the component reallocation problem, the goal is to find a modification in the existing solution of the allocation problem. Watkins and Cochran (1995) presented a line balancing heuristic for existing setups, where the existing component allocation is modified by greedy local moves.

The technology of a SMT placement machine significantly influences the optimization algorithm. Overview and classification of various machine technologies and related optimization algorithms is presented by Ayob and Kendall (2008). The SMT placement line in our problem consists of machines that can be categorized as dual-delivery collect-and-place machines. The dual-delivery machines are characterized by two placement heads that alternate in accessing one shared placement zone. The main feature of collect-and-place machines is the usage of placement heads containing multiple vacuum nozzles, that enable picking of a number of components from feeders (i.e. collecting) before the placement head moves above the PCB to perform placement operations. The placement heads can be equipped with various types of vacuum nozzles that are used depending on the physical parameters of mounted components. Typically, more than one nozzle type is needed even for production of single PCB type. For the collect-and-place machines, there are two possibilities how to deal with the nozzle requirements: either nozzles are changed during production as necessary or the heads are equipped with all the nozzles needed. The process of nozzle changes significantly decreases the production speed. Hence, if the changes are allowed, the goal is to minimize the number of nozzle changes. When we choose not to change the nozzles during production, another problem has to be solved – the selection of nozzles to be set up on the placement heads. The operation of dual-delivery collect-and-place machine is described by Tirpak et al. (2000), where the sequencing problem and the nozzle selection problem is solved. The workload balancing of the two alternating heads was not solved, since both heads were operating using the same setup. In the already mentioned work of Sun et al. (2005), which also deals with dual-delivery collect-and-place machines, the nozzle change minimization problem was solved by the workload evaluation heuristic. Raduly-Baka et al. (2008) presented an algorithm with polynomial time complexity to solve the optimal nozzle selection problem.

In this work, the component reallocation problem is formulated as a number partitioning problem, i.e. similarly to the formulation of component allocation problem in Ammons et al. (1997); Kulak et al. (2008). The formulation contains additional constraints that can be divided into two parts. The first part consists of constraints describing the reallocation property of the problem and the second one is related to the technical specifications of the placement machines. Nozzle changes are not allowed during production for two reasons: besides the already noted significant increase in the duration of assembly line cycle time, there is also a probability of breakdown caused by the unsuccessful nozzle change. The nozzle selection problem is included in the allocation problem as a nozzle selection constraint. While there are efficient algorithms for the number partitioning problem, the constraint programming (CP) (Dechter, 2003) approach is chosen due to the additional constraints. The CP model is combined with depth-first branch and bound optimization search. For better scalability and applicability to larger problem instances, the CP model was also used within the large neighborhood search framework (Shaw, 1998). The proposed algorithm can be also included as one part of the two step optimization, as described in approach of Kulak et al. (2008). The CP algorithm is tested on a production problem instance and a set of random instances generated according to the structure of the production problem instance. The results shown in this work represent a 10% improvement of the production speed, when the output of our algorithm was used for a production trial on the SMT line.

The two main contributions of the work are as follows. The nozzle selection problem is included in the reallocation problem, whereas in other approaches, either nozzle optimization is performed later as a part of the sequencing problem, or different nozzle types are not considered at all. Further, the proposed algorithm solves the component reallocation problem using complete search, whereas the related papers, e.g. Watkins and Cochran (1995), usually consider only single local search moves for the reoptimization of the component allocation.

The chapter is organized as follows. In the next section, the SMT assembly line consisting of dual-delivery machines is described. Section 3.3 presents the formulation of the component reallocation problem. In Section 3.4, the constraint programming model including the search procedure for the problem is described. Section 3.5 describes the construction of the large neighborhood search. Section 3.6 contains the description of problem instances used for the computational experiments, and the results of the performance evaluation are presented.

3.2 Description of the SMT assembly line

The SMT assembly line consists of two machines Siemens SIPLACE HS50 connected by two conveyors in parallel, as it is depicted in Figure 3.1. Each machine contains four revolver placement heads. Each placement head is equipped with 12 vacuum nozzles and a camera for visual inspection, and it has its own stationary feeder bank with a number of feeder slots. The placement head is mounted on an overhead gantry that allows simultaneous movements in X-Y directions and also movement in Z direction for the component pick and placement. The PCB does not move during the pick and place operations.

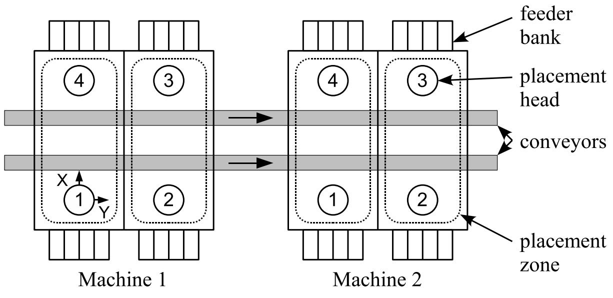


Figure 3.1: SMT assembly line with two machines HS50.

The machine can be classified as a combination of dual-delivery collect-and-place placement machine and multi-station placement machine according to Ayob and Kendall (2008); Kulak et al. (2008), and except the number of stations, it is similar to Fuji NP-132 machine in Tirpak et al. (2000). The machine is composed of two stations. Each of the stations contains one placement zone which is shared by a pair of placement heads, e.g. placement heads 1 and 4 in Figure 3.1. The pair of placement heads operates in dual-delivery mode, which means that at one time only one head of the pair can be located inside the placement zone. The dual-delivery operation of a placement heads pair is modeled by a Petri net (Petri, 1962) depicted in

Figure 3.2 and the Gantt chart illustrating the dual-delivery operation for one example assembly is presented in Figure 3.3.

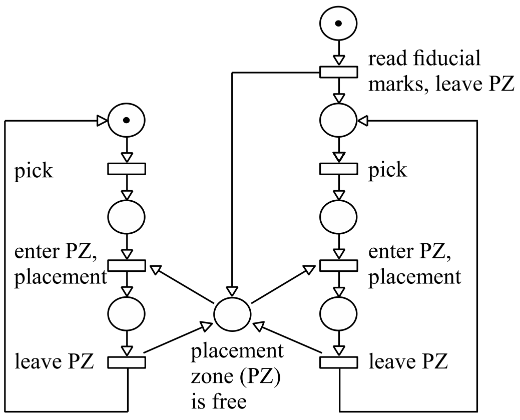


Figure 3.2: Petri net model of the dual-delivery operation of the pair of placement heads.

The sequence of operations starts when the PCB is loaded by the conveyor to the placement zone. One placement head of the pair, e.g. the head number 4, moves over the PCB to acquire the position of fiducial marks using the built-in camera, while the second placement head (the head number 1) moves over its feeder bank to perform the pick operation to collect 12 components at most (each nozzle can hold one component). Next, as soon as the placement zone is empty, the placement head 1 moves from the feeder bank over the PCB to place the components. At the same time, the placement head 4 is performing the pick operation. Then, the heads synchronize, i.e. the placement head that finishes the pick operation waits for the other head if it still blocks the placement zone with the placement operation. Then, the heads continue in alternation in the concurrent execution of pick operation and placement operation, with synchronization at the end of each pick operation before the placement head enters the placement zone.

One pick-and-place round of operations performed by the placement head is denoted as the *tour*, which is indicated in Figure 3.3. The *assembly time* is the time spent by the placement head pair on assembly of one PCB, measured from the start of the first pick operation to the end of the last movement out

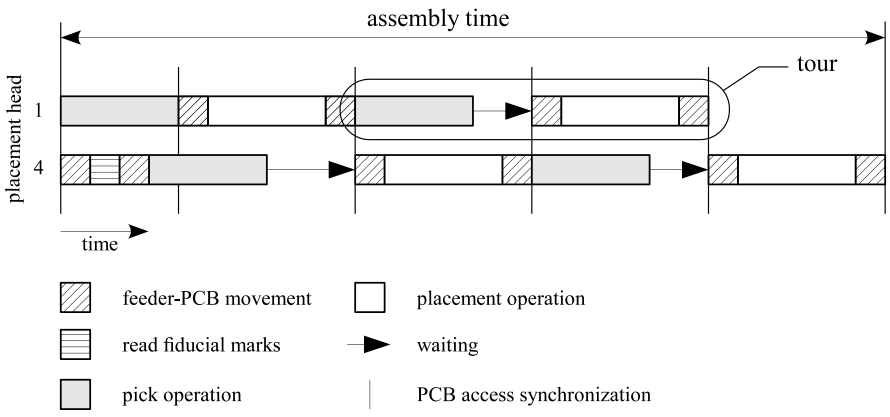


Figure 3.3: Gantt chart of an example assembly with dual-delivery operation of placement head pair.

of the placement zone. The time needed by the SMT assembly line to perform a complete assembly of one PCB is the *cycle time*. The assembly line consists of a sequence of dual-delivery placement head pairs (see Figure 3.1). For such configuration the cycle time is determined as the maximum of the assembly time for all placement head pairs.

3.3 Component reallocation problem

The aim of the component reallocation problem is to improve the *existing component allocation*, that was originally created by the SMT line operators as a manual modification of the *initial component allocation* by the line computer optimizer. The solution of the component reallocation problem is the *balanced component allocation* and is determined by the changes in component allocation with respect to the existing allocation. The component reallocation problem is a multicriteria optimization problem. The first objective of the problem is to minimize the maximal assembly time for all placement head pairs, i.e. to minimize the cycle time of the assembly line. The second objective is to minimize the number of changes in component allocation.

An instance of the component reallocation problem is specified as follows.

The PCB family in the problem is a singleton set, i.e. the re-optimization is performed for a single PCB type. The set of SMD components required for the assembly of the PCB is defined. This set is divided into mutually exclusive and collectively exhaustive subsets according to type of components, where the subset of all components of one type is denoted as the *component type*. The existing component allocation is defined. Each component type requires a specified nozzle type and is allocated to exactly one placement head. From the set of all component types, a subset is selected to be fixed at the existing allocation due to production quality requirements.

3.3.1 Assembly time approximation

The component reallocation problem is solved separately from the sequencing problem, which is a common approach to deal with the complex problem of SMT assembly optimization. Since the exact value of the assembly time is not available until the sequencing problem is solved, an approximation of the assembly time is included in the component reallocation problem.

Let $\mathcal{J}$ be the set of the placement heads. The number of tours performed by head $j \in \mathcal{J}$ is indicated by r_j. The placement zone is optimally utilized by a pair of placement heads, if the number of tours of the two placement heads is equal or different by one:

$$|r_j - r_k| \leq 1 \tag{3.1}$$

where $\{j, k\} \subset \mathcal{J}$ is the notation for the placement head pair. This requirement is easily proved. The placement zone is shared and the placement operations of the head pair are serialized. Therefore, it is not possible for the two placement heads to finish the assembly at the same time. For the absolute value of the difference (3.1) equal to zero or one, one placement head will be idle during the last placement operation of the second head, as it is depicted in Figure 3.3. For the difference of two or greater, one placement head has to be idle while the second head performs one placement operation and one or more complete tours.

Since the assembly line is equipped with one type of placement heads, the approximation of the sequencing problem is defined as follows:

A) The duration of pick and placement is equal for all components.

B) The duration of all feeder-PCB movements is equal.

Next, we assume that:
C) All placement heads perform at least one tour fully loaded.
D) Requirement (3.1) is satisfied.
Then, the assembly time of $\{j, k\} \subset \mathcal{J}$ is approximated by a number of components allocated to $\{j, k\}$, and by a number of tours performed by $\{j, k\}$. That is, the approximation replaces one criterion by two criteria.

Let us shortly explain the approximation. Due to A) and C), the duration of the first pick operation can be subtracted from the assembly time for all $\{j, k\} \subset \mathcal{J}$. Then, due to A) and D), the assembly time is a sum of durations of all placement operations and all but two feeder-PCB movements (examine in Figure 3.3). Finally, A) and B) is used to replace the durations by number of components and tours.

3.3.2 Problem specification

Let $\mathcal{I}$ be the set of all component types. Component type $i \in \mathcal{I}$ consists of w_i components. Let the integer variable l_j, the load of placement head $j \in \mathcal{J}$, represents the number of components allocated to j, which is defined by the following constraint:

$$l_j = \sum_{i \in \mathcal{I}} w_i \cdot x_{i,j} \qquad \text{for all } j \in \mathcal{J}, \tag{3.2}$$

where the binary variable $x_{i,j} = 1$ iff component type i is allocated to placement head j. A solution of the component reallocation problem is uniquely determined by values of $x_{i,j}$, i.e. $x_{i,j}$ is the main decision variable of the problem. Each component type is allocated to exactly one placement head, which is ensured by the constraint

$$\sum_{j \in \mathcal{J}} x_{i,j} = 1 \qquad \text{for all } i \in \mathcal{I}. \tag{3.3}$$

The constraints (3.2, 3.3) represent the multi-way number partitioning problem and correspond to the formulation of the allocation problem in Kulak et al. (2008).

The objective of the problem contains the requirement for minimization of changes in component allocation. Let the parameter a_i represent the placement head to which the component type i is allocated in the existing allocation. The change in allocation is then indicated by a binary variable y_i with $y_i = 1$ iff the component type i is allocated to placement head different from a_i. The relation between y_i and the allocation variable $x_{i,j}$ is expressed by the constraint

$$y_i = 0 \iff x_{i,a_i} = 1 \quad \text{for all } i \in \mathcal{I}. \tag{3.4}$$

The change of the allocation is disallowed for component types indicated by a set $\mathcal{D} \subset \mathcal{I}$, which is represented as

$$y_i = 0 \quad \text{for all } i \in \mathcal{D}. \tag{3.5}$$

Load l_j cannot exceed the number of components the head j is able to transport in r_j tours. This is expressed as

$$l_j \leq r_j \cdot c_j\,, \tag{3.6}$$

where c_j is the capacity of the revolver placement head j, i.e. the maximal number of vacuum nozzles carried by the head.

The placement heads are equipped with various types of vacuum nozzles as required by the component types. Since the nozzles are set up before the production and no changes are allowed during the production, r_j is directly related to the allocation and the nozzle type requirements of the component types. This relation is represented by the nozzle selection constraint described below. Let $\mathcal{T}$ be the set of all nozzle types used in the assembly. The binary parameter $v_{i,\tau}$ indicates whether the assembly of component type $i \in \mathcal{I}$ requires nozzle type $\tau \in \mathcal{T}$. Each component type requires exactly one nozzle type what is ensured by the parameters satisfying the constraint $\sum_{\tau\in\mathcal{T}} v_{i,\tau} = 1$ for all $i \in \mathcal{I}$. The partial load $l_{j,\tau}$ represents the load of placement head j caused by component types requiring nozzle type τ, which is expressed as

$$l_{j,\tau} = \sum_{i\in\mathcal{I}} v_{i,\tau} \cdot w_i \cdot x_{i,j} \quad \text{for all } j \in \mathcal{J}, \tau \in \mathcal{T}. \tag{3.7}$$

Load l_j is a summation of all partial loads, $l_j = \sum_{\tau\in\mathcal{T}} l_{j,\tau}$ for all $j \in \mathcal{J}$. The nozzle selection constraint then ensures that for each placement head, the total of required nozzles does not exceed the capacity:

$$\sum_{\tau \in \mathcal{T}} \left\lceil \frac{l_{j,\tau}}{r_j} \right\rceil \leq c_j \quad \text{for all } j \in \mathcal{J}. \tag{3.8}$$

This relation is more restricting than (3.6), which is demonstrated in Example 1: Given the partial loads of placement head j, the optimal (i.e. minimal feasible) value of r_j is determined.

Example 1 Nozzle selection

Determine the optimal value of r_j for head j with capacity $c_j = 12$ nozzles, which is used for placing of 16 components requiring nozzle type 1, 10 components requiring nozzle type 2 and 7 components requiring nozzle type 3.

The load of the placement head is $l_j = 33$. When the nozzles are allowed to be changed arbitrarily during the production, using Relation (3.6), the components will be placed in $r_j = \lceil 33/12 \rceil = 3$ tours. However, the nozzles have to be set up before the production and $r_j = 3$ is not sufficient since the required number of nozzles exceeds the capacity c_j:

$\lceil 16/3 \rceil + \lceil 10/3 \rceil + \lceil 7/3 \rceil > c_j$.

The optimal number of tours is $r_j = 4$:

$\lceil 16/4 \rceil + \lceil 10/4 \rceil + \lceil 7/4 \rceil \leq c_j$.

Depending on the values of partial loads $l_{j,k}$, the solution of nozzle selection constraint with smaller load l_j may require higher number of tours r_j. For illustration, compare Example 1 and another instance with load $l_j = 36$ consisting of partial loads $l_{j,1} = 15$, $l_{j,2} = 12$ and $l_{j,3} = 9$, which is placed in $r_j = 3$ tours. In the context of the component reallocation problem, the minimization of l_j may increase the value of r_j and vice versa. Since the duration of a feeder-PCB move is significantly longer than the duration of pick and placement of a component, a unit change in value of r_j has a greater impact on the assembly time than a unit change in l_j. Also, as defined by Relation (3.6), the value of r_j defines an upper limit on the value of l_j. Therefore, the minimization of the number of tours is performed prior to the minimization of the load. The objective is formulated as the lexicographic minimization Ehrgott (2005) – the lexicographic order is used for comparing the criterion vector composed of the problem criteria. The complete objective of the component reallocation problem is defined as

$$\operatorname{lexmin}\left(\max_{\{j,k\}\subset\mathcal{J}}(r_j+r_k),\max_{\{j,k\}\subset\mathcal{J}}(l_j+l_k),\max_{j\in\mathcal{J}} l_j,\sum_{i\in\mathcal{I}} y_i\right) \tag{3.9}$$

containing four criteria to be minimized. The first criterion is the maximum of the number of tours performed by the placement head pairs. The second criterion is the maximum of the load of the pairs. The purpose of the third criterion — the maximum of the load l_j for all $j \in \mathcal{J}$ — is to balance the load of the individual placement heads. Similar balancing of r_j for the individual placement heads is not necessary while the constraint (3.1) is satisfied. The last criterion represents the requirement to minimize the number of changes in the component allocation with respect to the existing allocation.

3.4 Solving the component reallocation problem

The component reallocation problem was solved using the Constraint Programming (CP) approach. Then the complete search constraint programming algorithm was used in the large neighborhood search algorithm.

3.4.1 Constraint programming model

The constraint programming model of the component reallocation problem is constructed using the constraints (3.1–3.8) and the objective (3.9) presented in the previous section. Most of these constraints are directly available in the used CSP system. The only problematic constraint is (3.8) due to the usage of the integer division rounding toward positive infinity, while the integer division available in constraint programming systems typically rounds the result toward zero. The integer division that rounds the result toward positive infinity can be formulated with the available division constraint using the relation $\lceil a/b \rceil = \lfloor a + b - 1/b \rfloor$. The nozzle selection constraint (3.8) is then implemented as

$$\sum_{\tau\in\mathcal{T}}\left\lfloor\frac{l_{j,\tau}+r_j-1}{r_j}\right\rfloor \le c_j \quad \text{for all } j \in \mathcal{J}. \tag{3.10}$$

Primarily for the purpose of the search, the CSP model contains two additional variables. The integer variable z_i represents the placement head

Algorithm 3.1 Identification of the bounds on r_j

```
1. Determine the average number of tours
      $AV = \frac{\sum_i w_i}{c \cdot |\mathcal{J}|}$
2. Determine the lower bound
if $AV - \lfloor AV \rfloor \leq 0.5$ then
      $LB = \lfloor AV \rfloor$
else
      $LB = \lceil AV \rceil$
end if
3. Determine the upper bound
      $UB = \max_j r'_j$
4. Apply the bounds
      $LB \leq r_j \leq UB$    for all $j \in \mathcal{J}$
```

the component type i is allocated to, and the relation between z_i and $x_{i,j}$ is defined by the channeling constraint

$$z_i = j \iff x_{i,j} = 1.$$

The second additional integer variable Y represents the total number of changes in allocation defined as

$$Y = \sum_{i \in \mathcal{I}} y_i.$$

To reduce the search space, upper and lower bounds on the domains of variables are identified. The upper bound is computed from the maximal number of tours r'_j in the existing solution. Algorithm 3.1 determines the lower and upper bound on variable r_j for all $j \in \mathcal{J}$. The bounds are subsequently distributed to variable l_j using constraint propagation on constraint (3.6). In the first step of Algorithm 3.1, it is assumed that the capacity of all placement heads is the same, that is $c = c_j$ for all $j \in \mathcal{J}$.

The objective (3.9), which consists of multiple lexicographically ordered criteria, is reformulated as a weighted sum of the criteria

$$\min \left(\lambda_1 \cdot \max_{\{j,k\} \subset \mathcal{J}} (r_j + r_k) + \lambda_2 \cdot \max_{\{j,k\} \subset \mathcal{J}} (l_j + l_k) + \lambda_3 \cdot \max_{j \in \mathcal{J}} l_j + Y \right)$$

where the non-negative integer parameters $\lambda_1, \lambda_2, \lambda_3$ represent the weights of the summation with sufficiently different values in order to prevent the interference of the individual criteria.

3.4.2 Search procedure

The search procedure (in CSP also denoted as the labeling procedure) designed for the component reallocation problem performs the variable selection and the value assignment as described in Algorithm 3.2. In the first step, the search procedure performs the value assignment for the variable Y in increasing order, i.e. the number of allowed changes in the allocation increases during the search. In step 2, the component type i is selected, which was originally allocated to placement head a_i with the maximal actual load. If there are several such component types, the second criterion for selecting i is the minimum of the possible alternative allocations, and further criterion is the maximum of size w_i. Then, two branches are created in the search tree: i is forced to change allocation, and alternatively allocation of i is unchanged. Step 3 performs labeling for the number of tours, selecting first j with smallest domain, i.e. using first-fail principle. Finally, in step 4, the allocation is assigned for the component sets which are supposed to change allocation.

Algorithm 3.2 Search procedure for the component reallocation problem

```
1. Select $Y$, assign the values in increasing order.
2. Labeling of $y_i \, \forall i \in \mathcal{I}$:
    Select $i$ for which $l_{a_i}$ has domain with the maximal minimum
      if not unique selection of $i$ then
        select $i$ with the minimal domain size of $z_i$
        if not unique selection of $i$ then
          select $i$ with the maximal value of $w_i$
        end if
      end if
    Create two branches in the search tree:
      – assign $y_i \leftarrow 1$
      – alternatively assign $y_i \leftarrow 0$
3. Labeling of $r_j \, \forall j \in \mathcal{J}$:
    Select $r_j$ with the smallest domain, assign values in increasing order.
4. Labeling of $z_i \, \forall i \in \mathcal{I}$:
    Select $z_i$ with the smallest domain, assign values in increasing order.
```

The used CSP system contains a generic search procedure that offers various choices for the variable selection and also for the value assignment. It was used in steps 1,3 and 4 of the search procedure described in Algorithm 3.2. The decisions made in step 2 were implemented as a special branching heuristic.

The depth-first branch and bound optimization search strategy was applied to solve the problem.

3.5 Large neighborhood search

The component reallocation problem is stated as a re-optimization of the existing solution. That is, the problem naturally fits for application of the local search. The paper of Watkins and Cochran (1995) presents a local search approach to solve a less complex version of component reallocation problem. It defines the neighborhood using simple local move composed of change in allocation of one component type. By contrast, with the application of the large neighborhood search, complex exchanges in allocation can be explored. The functionality of LNS used in this work to solve component

Algorithm 3.3 Large neighborhood search

```
$s^* \leftarrow$ existing solution of the problem
**while** *stop condition* = false **do**
    Select $R \subseteq \mathcal{I}$
    Create a new problem instance $p$ using $s^*$:
        value assignment is relaxed for $z_i$ for all $i \in R$
        $z_i = z_i^*$ for all $i \in \mathcal{I} \setminus R$
    $s \leftarrow$ solve($p$)
    **if** $s$ is better than $s^*$ **then**
        $s^* \leftarrow s$
    **end if**
**end while**
```

reallocation problem is described in Algorithm 3.3.

The algorithm is initialized with s^* containing the existing solution of the component reallocation problem. Then, the iterative loop is repeatedly executed while a defined stop condition does not hold. The main variable for the LNS approach is z_i representing the allocation of the component type $i \in \mathcal{I}$. From the set $\mathcal{I}$ a subset R is selected by neighborhood selection heuristic, as described bellow. The variables z_i of the new problem instance p are for all $i \in \mathcal{I} \setminus R$ fixed at the values of z_i^* from the current best solution s^*. The value assignment of z_i is propagated to related variables $x_{i,j}$ and y_i. Then, the partially fixed problem instance p is solved using the constraint programming model and the depth-first branch and bound optimization search described in Section 3.4. In order to explore more neighborhoods, p is not solved completely, but with a limit on number of failures during the search. In this work, the limit was set to 40k.

The important part of applying LNS is the process of selecting the set R that composes the neighborhood. In this work, three neighborhood selection heuristics are studied.

LNS1 The *randomized selection* chooses randomly the component types for which the value assignment will be relaxed. The size of R is chosen randomly between 10% and 50% of $|\mathcal{I} \setminus \mathcal{D}|$. The elements of R are randomly selected from $\mathcal{I} \setminus \mathcal{D}$. The randomized selection of the neighborhood was previously used for example in Gargani and Refalo (2007) on the steel mill slab design problem with a structure similar to the component allocation problem. The randomized selection heuristic has one drawback concerning the criterion Y of the component reallocation problem. In each iteration of LNS, a random part of the current best solution is fixed. The changes in allocation made in one iteration may be fixed for the next iteration. That way, the algorithm can accumulate a significant difference in allocation with respect to the existing solution. Then additional iterations are necessary to perform in order to reduce the number of changes represented by Y.

LNS2 The *randomized non-drifting selection* of R is designed specifically for the problem with the objective containing a criterion of minimizing the difference between the existing and the optimized solution. The idea of the heuristic is that all component types i which have changed allocation need to be included in the set R. The size of R is defined as $|\{i \in \mathcal{I} | z_i^* \neq a_i\}|$ plus 10%–50% of $|\{i \in \mathcal{I} \setminus \mathcal{D} | z_i^* = a_i\}|$. All component types in $\{i \in \mathcal{I} | z_i^* \neq a_i\}$ are selected to be part of R. Subsequently, the set R is completed by random selection from the set $\{i \in \mathcal{I} \setminus \mathcal{D} | z_i^* = a_i\}$.

LNS3 The alternation of the neighborhood selection heuristics LNS1 and LNS2. In the **while** loop of Algorithm 3.3, when the solving of the partially relaxed problem instance p fails n times in a row, the neighborhood selection heuristic is switched. The alternation is performed after 10 fails of LNS1 and after 1 fail of LNS2.

3.6 Experimental results

The proposed algorithms were implemented using Gecode 3.4, an open-source C++ library for constraint programming (`http://www.gecode.org`). The constraint programming algorithm using the depth-first branch and bound search from Section 3.4 is denoted as DFS in the following text. The neighborhood search algorithm is used with the three neighborhood selection heuristics as denoted in Section 3.5.

All algorithms were compiled as 32bit binaries and executed on a PC with AMD Opteron 248 CPU at 2.2GHz with 2GB of RAM, in the environment of Microsoft Windows Server 2003. The time limit for each computation was 600 s, after which the execution was stopped, and the best solution up to that time was returned. For the LNS algorithms, the time limit was used as the stop condition. The LNS algorithms performed 10 executions for each problem instance in order to measure the robustness of the LNS approach.

The experiments were conducted using one problem instance from production and a set of randomly generated instances following the structure of the production problem instance. The SMT assembly line for all problem instances consists of two dual station dual delivery machines Siemens SIPLACE HS50 as depicted in Figure 3.1, i.e. the number of placement heads is $|\mathcal{J}| = 8$. Each placement head j is equipped with revolver placement head containing $c_j = 12$ vacuum nozzles.

3.6.1 Production problem instance

The experimental verification of the algorithm was realized for one PCB type. The problem instance consists of $|\mathcal{I}| = 61$ component sets with 572 components in total. The placement of the components requires $|\mathcal{K}| = 7$ different types of vacuum nozzles. Change in allocation is disallowed for $|\mathcal{D}| = 11$ component sets.

All tested algorithms obtained the solution with the same objective. The computation time needed to obtain the solution is presented in Table 3.1.

	DFS	LNS1	LNS2	LNS3
CPU time [s]	103.6	25.8	9.2	21.1

Table 3.1: Computation time for the production problem instance.

The existing component allocation is compared to the balanced solution in Figure 3.4. Each column represents the load of the placement head specified at the horizontal axis label by machine-head number code following the notation used in Figure 3.1. The pairs of placement heads are indicated. The number of tours for the existing component allocation and the balanced solution is depicted in Figure 3.5.

This optimized component allocation was used for production trial at the SMT line. The conclusion of the results measured by SMT line operators during production is included in Table 3.2, with the maximum assembly time for all placement heads, and also with the balance ratio computed as mean/max assembly time in percents.

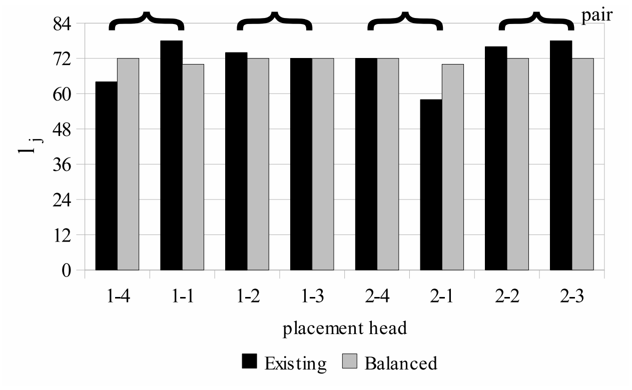


Figure 3.4: Comparison of l_j for existing and balanced component allocation for production problem instance.

Table 3.2: Cycle time comparison for existing and balanced feeder setups.

	Existing	Balanced	Difference
Cycle time [s]	34.1	30.5	-3.6
Balance ratio [%]	83.4	89.5	$+6.1$

3.6.2 Generated instances

The problem instances randomly generated for the purpose of performance evaluation share the same structure with the production problem instance. The generation of the instances is performed in two steps. First, a list of component types with assigned size and required vacuum nozzle type is created. The proportion of occurrence of different values of w_i is determined according to the production problem instance. The frequency of usage of different vacuum nozzle types also corresponds to the production problem. Next, the component list is used as an input for the constraint programming DFS algorithm from Section 3.4 with disabled optimization. It non-optimally allocates component types to placement heads, generating existing allocations which are the problem instances for the component reallocation problem. The set $\mathcal{D}$, which is selected randomly for each instance, is of size 10 for all instances. The problem instances were generated in three sizes of $|\mathcal{I}| \in \{62, 75, 100\}$,

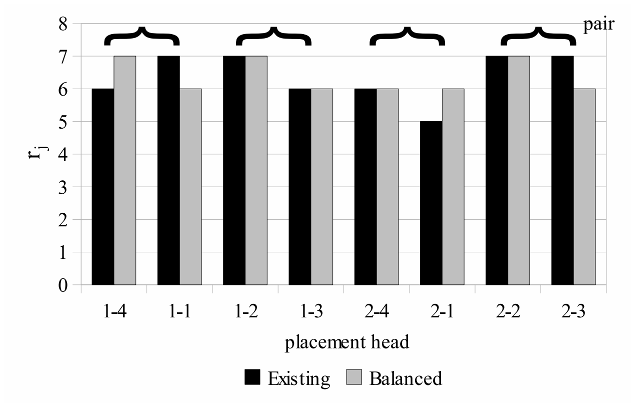


Figure 3.5: Comparison of r_j for existing and balanced component allocation.

with 20 instances in each of the three sizes.

Table 3.3 summarizes the results of the experiments. For each problem size and applied algorithm, it presents the number of *Solved* instances (i.e. found any solution) out of the 20 in each set. The best solution is selected for each problem instance from the results of all four algorithms. In row *Best*, the table presents the number of problem instances (out of 20) for which the best solution was found. The *Uniquely best* row represents the number of solutions for which the algorithm found the best solution, that was not found by the other algorithms. The CPU time is the computation time needed to obtain the best solution. The table presents the median of the CPU time for the 20 problem instances.

The conclusion from Table 3.3 is that the LNS algorithms outperform the DFS algorithm in both computation CPU time and the ability to find the best solution. The difference in performance is more apparent for larger problem instances. When comparing the LNS algorithms, the alternation of the two neighborhood selection heuristics, denoted as LNS3, outperforms both heuristics alone.

Table 3.4 relates to the experiment focusing on the robustness of the LNS algorithms. Each problem instance is solved 10 times by each LNS algorithm. A solution with the best objective is selected for each instance. The table presents, for each algorithm and problem size, the percentage of executions that obtained the best objective. In order to study the capability of the LNS

Table 3.3: Computation results for the randomly generated instances.

Size	62				75				100			
Algorithm	DFS	LNS1	LNS2	LNS3	DFS	LNS1	LNS2	LNS3	DFS	LNS1	LNS2	LNS3
Solved	20	20	20	20	20	20	20	20	12	20	20	20
Best	11	20	14	20	13	20	19	20	0	13	15	16
Unique best	0	0	0	0	0	0	0	0	0	1	1	3
Median CPU [s]	6.4	4.8	0.2	4.7	58.6	4.2	2.8	2.6	309	152	155	117

algorithms to minimize the number of changes, the results are inspected for two cases:
a) Y, the number of changes, is included in the objective. b) Y is not included in the objective.
Comparing the results for the two cases a) and b), the results by the randomized selection heuristic (LNS1) have the largest difference, which is the reason for designing the other heuristics, as discussed in Section 3.5. The heuristic LNS3 outperformed LNS1 and LNS2 also according to the results in Table 3.4. Next, the results show that with increasing size of the problem instance, the LNS algorithms are less capable to find the best solution.

Table 3.4: Robustness of the LNS algorithms: Ability to find the best solution (in percents)

Size	62			75			100		
Algorithm	LNS1	LNS2	LNS3	LNS1	LNS2	LNS3	LNS1	LNS2	LNS 3
Y in objective	67	59	66	79	85	88	13	26	22
Y not in objective	68	59	66	87	85	90	77	41	76

3.7 Conclusion

This chapter presents an algorithm for the reoptimization of the component allocation which was created by manual modifications performed in order to improve the quality of the assembly. In order to minimize the influence of the reoptimization on the quality of the assembly, the objective of the problem includes the minimization of changes in component allocation, and a list of components is specified for which the change in allocation is not allowed.

The proposed model of the component reallocation problem includes the nozzle selection problem that was in previous works solved later as a part of the sequencing problem (Sun et al., 2005; Kulak et al., 2008). The algorithm is implemented using constraint programming approach using depth-first branch and bound optimization search. More complex exchanges in the allocation can be explored than in the local search approach of Watkins and Cochran (1995). Subsequently, the CP algorithm is used within the large neighborhood search framework in order to obtain better scalability. Two heuristics for construction of the neighborhood are presented, and also an algorithm alternating the application of the two heuristics is introduced.

The algorithms are applied to one production problem instance and the conducted production trial shows an improvement of assembly line cycle time by 10 %. Next, a set of randomly generated problem instances is used to compare the constraint programming DFS algorithm and the large neighborhood search algorithms. The LNS algorithms outperform the DFS algorithm, with a more significant difference in the results for the larger problem instances. Among the neighborhood selection heuristics, the alternation of the heuristics, denoted as LNS3, shows the best performance.

Chapter 4

Permutation flow-shop with blocking

This part of the thesis deals with a problem of scheduling an assembly line producing agricultural machinery. The problem is categorized as the permutation flow-shop problem with blocking. The algorithm to solve the problem is based on the large neighborhood search framework and it uses a branch-and-bound algorithm as a complete search sub-routine. A new algorithm for computation of lower bound is presented taking into account the properties of the explored neighborhoods.

4.1 Introduction

This chapter deals with a problem of scheduling an assembly of agricultural machinery. The company manufactures a range of product types. The assembly of all these product types is performed at one assembly line consisting of a sequence of assembly stations. A job is defined by customer order where a customer specifies product type, chosen configuration and options. The order specification defines the processing time of the job at each assembly station. Each job passes through all assembly stations, and the order of visiting the stations is the same for all jobs. There are no storage buffers between the assembly sites, so a job that finishes operation at one assembly station cannot leave that station until the immediately following station is empty. The objective of the company is to minimize the total assembly time.

The problem is categorized as the permutation flow-shop problem with blocking (PFSB). There is a set of jobs that are processed on a set of machines (i.e. unary resources). All jobs visit the resources in the same order. There is no possible overtaking of the jobs during the production, that is all jobs are processed on each resource in the same order. The objective of the problem is to find such a permutation for processing of the jobs that the completion time of the last job is minimized.

Considering the computational complexity, the problem is NP-hard for $m > 3$ resources (Hall and Sriskandarajah, 1996). In the case of a flow-shop problem with two resources, denoted as $F2|block|Cmax$ according to the notation by Graham et al. (1979), the problem can be polynomial-time reduced to a special form of the Traveling Salesman Problem (Reddi and Ramamoorthy, 1972) which can be solved by a polynomial time complexity algorithm (Gilmore and Gomory, 1964).

The approaches to solve the permutation flow-shop problem with blocking can be roughly divided into two subsets – complete algorithms and heuristics. Among the complete algorithms the most common approach is to use a branch-and-bound algorithm, as for example in (Levner, 1969; Ronconi and Armentano, 2001; Ronconi, 2005; Companys and Mateo, 2007). The used branching is typically the one described by Lomnicki (1969) for solving the flow-shop problem without blocking. In brief, the branching is constructing the schedule (i.e. permutation of the jobs) from the start of the schedule. In each level of branching, a job from a set of unordered jobs is selected and inserted at the end of the current partial schedule. Concerning the recent branch-and-bound algorithms, Ronconi (2005) presents a new bounding algorithm, which utilizes a part of the algorithm (Gilmore and Gomory, 1964) for solving $F2|block|Cmax$. Companys and Mateo (2007) presents another approach (algorithm called LOMPEN), where the branch-and-bound algorithm is simultaneously executed for the original problem instance and for the time-inverse formulation of the problem. This algorithm (as the only one from the branch-and-bound algorithms) was able to verify optimal solutions for all benchmark flow-shop instances by Taillard (1993) of size $n = 20$ jobs and $m = 5$ machines.

The heuristic approaches can be further divided into constructive heuristics and meta-heuristics. The constructive heuristics are polynomial-time complexity algorithms that usually create a suboptimal solution. Such an algorithm operates first by assigning a priority to the jobs using some defined rules. In the next step, the algorithm processes the jobs according to this priority and insert them into the final schedule. In (McCormick et al., 1989), a constructive heuristic for the permutation flow-shop problem with blocking was introduced. Lestein (1990) compared heuristics developed for permutation flow-shop problem without blocking when applied to PFSB. In conclusion of that study, the best of the algorithms was the NEH heuristic (Nawaz et al., 1983). A study of the efficiency of different variants of the NEH heuristic is presented in (Ribas et al., 2011). In that work, the mentioned constructive heuristics are used as one of the steps of an iterated greedy algorithm. The presented algorithm was able to find new best solutions for most of the Taillard's benchmark instances. From the works using meta-heuristic approach, for example, a genetic algorithm is presented in (Caraffa et al., 2001). Grabowski and Pempera (2007) presented a tabu search approach to solve PFSB.

The algorithm presented in this chapter to solve the permutation flow-shop problem with blocking is based on the large neighborhood search framework (Shaw, 1998). It represents a local search method, where a local neighborhood of a current solution is explored using a complete search method. The used complete search method is based on branch-and-bound algorithm by Ronconi (2005). A new algorithm for computation of lower bound is presented which better utilizes the properties of the explored neighborhoods in comparison to the bounding algorithm in Ronconi (2005).

The chapter is organized as follows. In Section 4.2 the permutation flow-shop problem with blocking is formulated. In the next Section 4.3, the algorithm for solving the PFSB is presented. Section 4.4 shows the results of the conducted experiments. The last section then concludes the results of the work.

4.2 Problem formulation

The flow-shop problem with blocking can be formulated as described in the following text. We assume a set of n jobs $\mathcal{J} = \{J_1, \dots J_n\}$, where job J_j consists of a set of tasks $\mathcal{T} = \{T_{j,1}, \dots T_{j,m}\}$. Each task of job T_j has a given processing time $p_{j,i}$ on a unary resource R_i from a set of m resources $\mathcal{R} = \{R_1, \dots, R_m\}$. The start time $S_{j,i}$ of all task determine the result of the scheduling problem.

For each job j, there exist a precedence relation between tasks $T_{j,i}$ and $T_{j,i+1}$ such that

$$S_{j,i} + p_{j,i} \leq S_{j,i+1} \quad \text{for all } i = 1, \ldots, m-1. \tag{4.1}$$

Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ denotes a permutation of jobs, where $\pi_k = j$ iff job J_j is the k-th job of the schedule. Then, a precedence relation between two consecutive jobs is formulated as a precedence relation between the tasks sharing the same resource:

$$S_{\pi_k,i} + p_{\pi_k,i} \leq S_{\pi_{k+1},i} \quad \text{for all } k = 1, \ldots, n-1; i = 1, \ldots, m. \tag{4.2}$$

The blocking property of the schedule is expressed by relation

$$S_{\pi_k,i} + p_{\pi_k,i} \leq S_{\pi_{k+1},i-1} \quad \text{for all } k = 1, \ldots, n-1; i = 2, \ldots, m. \tag{4.3}$$

The makespan of the problem is defined as the end of processing of the last job, i.e. $C_{max} = S_{\pi_n,m} + p_{\pi_n,m}$.
The objective of the problem is to find a permutation $\pi' \in \Pi$ from the set Π of all permutations such that the makespan is minimal:

$$C_{max}(\pi') = \min_{\pi \in \Pi} C_{max}(\pi). \tag{4.4}$$

4.2.1 Computation of makespan

When a permutation π of jobs is defined, a schedule can be computed in polynomial time using a recursive computation (Pinedo, 1995). In this algorithm, variables $D_{j,i}$ are introduced denoting the departure time of job j from resource i, that is the time when the resource is released by job j and the next job can be processed. A relation $C_{j,i} \leq D_{j,i}$ applies for each task of the schedule.

$$D_{\pi_1,0} = 0, \tag{4.5}$$

$$D_{\pi_1,i} = \sum_{q=1}^{i} p_{\pi_1,q} \qquad i = 1,\ldots,m-1, \tag{4.6}$$

$$D_{\pi_j,0} = D_{\pi_{j-1},1} \qquad j = 2,\ldots,n, \tag{4.7}$$

$$D_{\pi_j,i} = \max(D_{\pi_j,i-1} + p_{\pi_j,i}, D_{\pi_{j-1},i+1}) \qquad j = 2,\ldots,n; i = 1,\ldots,m-1, \tag{4.8}$$

$$D_{\pi_j,m} = D_{\pi_j,m-1} + p_{\pi_j,m} \qquad j = 1,\ldots,n. \tag{4.9}$$

The makespan of the schedule is then represented as $C_{max}(\pi) = D_{\pi_n,m}$. This computation of makespan is used in various algorithms for PFSB, for example in (Ronconi, 2005; Grabowski and Pempera, 2007; Ribas et al., 2011), and also in the algorithm presented in this chapter.

4.3 Solving technique

4.3.1 Large neighborhood search

The functionality of large neighborhood search algorithm used in this work is described in the following Algorithm 4.1. An existing permutation of the jobs is used to initialize the LNS algorithm. This initial permutation is obtained by the same algorithm that performs the complete search in the iterations of LNS (see bellow), with the only difference in computation of lower bound – the same LB is used as in (Ronconi, 2005). The solving time for the initial permutation is limited to some reasonable value, for example 60 s. Variable f which serves as counter of failed (i.e. non-improving) iterations of LNS is initialized to value 0.

Then, in each iteration of LNS, an ordered set of jobs $OS \subset \mathcal{J}$ is selected containing jobs where precedence from the last iteration will be preserved. The size of OS is $n-5$ to $n-10$ jobs. Two types of selection are alternated: 1) random selection 2) selection of tuples of jobs $j \in \mathcal{J}$ in the non-decreasing order of values of

$$\sum_{i \in 1,\ldots,m} D_{\pi^*_{j+1,i}} - S_{\pi^*_{j,i}} - p_{\pi^*_{j+1,i}} - p_{\pi^*_{j,i}}, \tag{4.10}$$

Algorithm 4.1 Large neighborhood search

```
**Initialization:**
  $\pi^* \leftarrow$ initial permutation of jobs
  $f \leftarrow 0$
**while** *stop condition* = false **do**
  Select $OS \subset \mathcal{J}$
  Create a new problem instance $P$ using $\pi^*$ and $OS$:
    $P \leftarrow \text{precedence}(\pi^*, j \in OS)$
  $\pi \leftarrow \text{solve}(P)$
  **if** $C_{max}(\pi) < C_{max}(\pi^*)$ **then**
    $\pi^* \leftarrow \pi$
    $f \leftarrow 0$
  **else**
    $f \leftarrow f + 1$
  **end if**
  **if** $f >$ *limit* **then**
    $\pi^* \leftarrow \text{escape}(\pi)$
  **end if**
**end while**
```

that is, tuples of jobs are selected, where the idle time between the jobs is minimal.

Given the permutation π^* and set OS, a new problem instance P is created, where for jobs $j \in OS$ a precedence relation is declared in order to satisfy the order of jobs in π^*. Then, P is solved using the complete search algorithm described in the following sub-section. If the newly found permutation is better, it will be used in following iteration. If f reaches a given limit, a modified version of the complete search algorithm is executed in order to escape from the local minimum.

4.3.2 Complete search algorithm

The complete search algorithm is branch-and-bound algorithm derived from (Ronconi, 2005). The main difference is that the algorithm is implemented using constraint programming (CP) approach (Dechter, 2003), which is a method for declarative description and solving of the decision and optimization problems. *Constraint satisfaction* over the integer domains is a part of the constraint programming which is used to solve combinatorial problems.

The constraint satisfaction problem (CSP) is defined as a triple (x, d, c), where $x = \{x_1, \dots, x_n\}$ is a finite set of *variables*, $d = \{d_1, \dots, d_n\}$ is a set of respective *domains* that contain possible values for each variable $d_i = \{v_1, \dots, v_k\}$ and $c = \{c_1, \dots, c_t\}$ is a set of relations — called *constraints* — restricting the legal combinations of the values of the variables. Assigning the values to the variables so that all constraints are satisfied at once is a solution of the CSP.

The software systems for solving CSP (CSP solvers) employ two cooperative techniques to get a solution of CSP. *Constraint propagation* removes from the domains those values that directly violate the related constraints. Usually, constraint propagation itself is not capable of finding a solution of the CSP, so the second technique — the *search* algorithm — is used to systematically explore the search space pruned by the constraint propagation. The search consists of a search procedure (also called the labeling procedure) used to construct the search tree, and a search strategy (e.g. depth-first search) that is applied in order to explore the search tree. The search procedure typically makes decisions about the variable selection (i.e. which variable to choose) and about the value assignment (i.e. which value to assign to the selected variable). The constraint propagation and the search cooperate during the process of solving the CSP. The search decision, i.e. the assignment of the value to the chosen variable, reduces the domain of the variable, which triggers the constraint propagation of the related constraints.

Although CP offer specialized scheduling algorithms (Baptiste et al., 2001), their efficiency to solve PFSB was inferior when compared to (Ronconi, 2005). Therefore, CP was used only as a framework for implementation of B&B algorithm.

The problem formulation consists of the following variables:

π_k for all $k = 1, \dots, n$ represents the job at position k

h_j for all $j = 1, \dots, n$ represents the position of job j in the permutation

D_i for all $i = 1, \dots, m$ is a departure time on resource i, which represents during solving of the problem, at which time the resource is free for next job.

Algorithm 4.2 Branching algorithm

```
function $Branch(PS)$ :
  $k \leftarrow |PS| + 1$
  Select $\pi_k$
  for all $j \in d(\pi_k)$ do
    compute lower bound $LB_{j,k}$ where
    job $j$ is inserted at the end of $PS$ into position $k$
  end for all
  for all $j \in d(\pi_k)$ ordered by non-decreasing $LB_{j,k}$ do
    create $PS' = PS + j$
    update $D_i$ for all $i = 1, \ldots, m$ for created $PS'$
    $Branch(PS')$
  end for all
end function
```

The constraints in the problem formulation are:

$\pi_k = j \Leftrightarrow h_j = k$	channeling constraint connecting π_k and h_j
$h_{j1} < h_{j2}$	iff job j_1 precedes j_2

The objective of the problem is to minimize the final departure time on the last resource:

$$\min D_m.$$

The recursive branching algorithm is presented in Algorithm 4.2. The input of the algorithm is a partial sequence PS of already sequenced jobs. $d(\pi_k)$ denotes the domain of variable π_k, i.e. a set of jobs that are allowed to be placed at position k. When a new branch PS' is created where job j is inserted at the end of current PS, the departure times D_i are updated using the algorithm for computation of departure time (5–9). The computed lower bound is used for two purposes. As usually in branch-and-bound algorithms, the LB is used to discard branches of the search tree, where LB exceeds the objective value of current best solution. The second purpose is to set the search priority for the newly created branches. The computation of lower bound is presented in detail in next sub-section.

Escaping from local minimum

The algorithm for escaping from local minimum, which is used in the LNS algorithm, adds new variables and constraints to the presented complete search algorithm. The objective of the problem is different.
Using current best permutation π^*, an array of constants $next_j$ is crated for all $j \in \mathcal{J}$ representing the job which is directly following job j in permutation π^*:
$next_j = \pi^*_{h_j+1}$ for all $j \in \mathcal{J}$.
New binary variable x_j for all $j \in \mathcal{J}$ is created, which will contain value 1 iff in the current schedule job j has the same directly following job as in π^*. This is represented by the following constraint:
$x_j = 1 \Leftrightarrow (\pi_{h_j+1} = next_j)$.
Then, the problem is solved while minimizing $\sum_j x_j$ and D_m simultaneously, with higher priority on x_j, i.e. the objective is:
$\min \left(M \cdot \sum_j x_j + D_m \right)$ where $M >> \max_m D_m$.

4.3.3 Lower bound computation

The computation of lower bound utilizes the fact that when LNS approach is used, there exists a subset of jobs $j \in \mathcal{J} \setminus R$ ordered according to permutation π^* from the previously found best solution. The lower bound computation is based on the following Property 1 of the PFSB.
Property 1: *Let π is a permutation of jobs in PFSB and $J_j \notin \pi$ is a job not included in the permutation. The makespan of optimal schedule for π is denoted as $Cmax_\pi$. The makespan of optimal schedule for $\pi \cup j$, where J_j is inserted optimally between two other jobs in π is*

$$Cmax_{\pi \cup j} \geq Cmax_\pi + \min_{i=1,\dots,m} p_{j,i}$$

Proof: Given the permutation π, the grid graph (Grabowski and Pempera, 2000) representing the schedule is constructed. The critical path traversing this graph connects any two consecutive jobs on at least one of the resources from the set $\mathcal{R}$. If job j is inserted between two jobs, makespan will be increased at least by $p_{j,i}$ where i is the critical resource. The critical path starts at task $T_{\pi_1,1}$ and ends at $T_{\pi_n,m}$. Therefore, if job j is inserted in front of π, the value of makespan will be increased at least by $p_{j,1}$. Similarly, when

Algorithm 4.3 Computation of lower bound

```
initialization:
    $LB \leftarrow 0$
    Compute the departure time $D$ for the jobs in $PS$.
    Select jobs $J_U : \{j \in \mathcal{J} \mid j \in OS \setminus PS\}$
    Select jobs $J_N : \{j \in \mathcal{J} \mid j \in \mathcal{J} \setminus (OS \cup PS)\}$
    Select job $j'$ as the last job in the set $\{j \in \mathcal{J} \mid j \in OS \cup PS\}$
for all $i = 1, \ldots, m-1$ do
    Compute $D$ on resources $i, i+1$ for $j \in J_U$ when scheduled after $PS$
    $D_{U,i+1} \leftarrow D$ of last job in $J_U$ on resource $R_{i+1}$
    Update $LB$
        $LB \leftarrow \max\left(D_{u,i+1} + \sum_{j \in J_N} \min_{q=i,i+1} p_{j,q} + \sum_{q=i+2}^{m} \min_{j \in J_N \cup j'} p_{j,q}, \quad LB\right)$
end for all
```

job j is inserted after π, the value of makespan will be increased at least by $p_{j,m}$.

The computation of lower bound is based on Property 1. In level k of the branching tree, there exists a partial sequence PS of jobs $\pi_1, \ldots, \pi_k$ that create a part of the resulting permutation, beginning from the start of the resulting permutation. The subset of jobs $j \in \mathcal{J} \setminus R$ ordered according to permutation π^* will be denoted as ordered subset OS.

4.4 Experimental results

The experiments evaluating the performance of designed algorithms were conducted using the flow-shop benchmark instances by Taillard (1993). In the first part, the introduced Algorithm 4.3 for computation of lower bounds is compared to the algorithm by Ronconi (2005). In the second part, the designed LNS algorithm is compared to other algorithms for solving permutation flow-shop problem with blocking. The experiments were executed using algorithms implemented in Gecode (Gecode Team, 2010) C++ library for constraint programming. The presented algorithms were executed on a PC with AMD Opteron 248 CPU at 2.2 GHz with 2 GB of RAM. The results for the algorithms from related works were take from the relevant papers.

4.4.1 Lower bound computation

Algorithm 4.3 is designed to compute a lower bound for flow-shop problem instance where an ordered subset $OS \subset \mathcal{J}$ of jobs exists, where precedence is defined. Benchmark instances for this part of experiments were created from the benchmarks (Taillard, 1993) by following operations. For given problem size, size of OS was specified and the set OS was randomly generated as a subset of $\mathcal{J}$. The presented complete search algorithm was used with each of the two compared LB computation algorithms. The solution time was limited to 60s. We used problem instances of size $n = \{20, 50, 100\}$ jobs and $m = \{5, 10, 20\}$ resources, i.e. problem instances of 9 different problem sizes were used. The results show that although the value of lower bound computed by Algorithm 4.3 was $3 - 5\%$ higher than the lower bound computed by the algorithm (Ronconi, 2005), the output of the optimization was quite equal.

4.4.2 Minimizing the makespan

In this experiment, the proposed LNS algorithm is compared to the branch-and-bound algorithm by Ronconi (2005) (RON), to the tabu search algorithm by Grabowski and Pempera (2007) (GP) and to the greedy iterative algorithm by Ribas et al. (2011). The time limit for each computation was 1 hour, after which the execution was stopped, and the best solution up to that time was returned. We used problem instances of size $n = \{20, 50, 100\}$ jobs and $m = \{5, 10, 20\}$ resources. For each size of the problem, only the first five instances from the flow-shop benchmark set by Taillard (1993) was used. The results of the experiments are presented in Table 4.1. The LNS algorithm did not find for any problem instance a better solution than was presented in (Ribas et al., 2011). For the larger problem instances, the LNS is $5 - 7\%$ worse. In Table 4.1, the results are compared only to results obtained by algorithms RON and GP.

The presented results show that for smaller problem instances, LNS algorithm is able to find better solution than RON or GP algorithm. For problem instances of larger size, the tabu search approach GP was better performing than LNS.

4.5 Conclusion

This chapter presented an algorithm for solving permutation flow-shop problem with blocking. We were able to improve the efficiency when compared to the branch-and-bound algorithm by (Ronconi, 2005), on which the designed LNS algorithm is based. However, for the larger problem instances, the algorithm was surpassed by the tabu search approach of (Grabowski and Pempera, 2007). Only for instances of size 20×5 and 20×10 we were able to obtain solutions equal to the results from (Ribas et al., 2011). The potential for the approach presented in this work is in solving permutation flow-shop problem with additional constraints, which would make the constraint propagation more efficient.

Table 4.1: Best obtained solutions for Taillard's benchmarks.

LNS	GP	RON	LNS	GP	RON	LNS	GP	RON
20 × 5			50 × 5			100 × 5		
1374	1387	1384	3056	3163	3151	6475	6639	6455
1409	1424	1411	3280	3348	3395	6172	6481	6214
1280	1293	1294	3117	3173	3184	6096	6299	6124
1449	1451	1448	3263	3277	3303	5961	6120	5976
1345	1348	1366	3289	3338	3272	6110	6340	6173
20 × 10			50 × 10			100 × 10		
1698	1698	1736	3698	3776	3913	7354	7320	7496
1835	1836	1897	3669	3641	3798	7084	7108	7281
1672	1674	1677	3614	3588	3723	7303	7233	7400
1538	1555	1622	3787	3786	3885	7420	7413	7670
1617	1631	1658	3774	3745	3934	7172	7168	7317
20 × 20			50 × 20			100 × 20		
2448	2449	2530	4672	4627	4886	8238	8101	8347
2244	2242	2297	4418	4411	4668	8362	8105	8372
2479	2483	2560	4556	4388	4666	8127	8071	8265
2348	2348	2399	4478	4479	4650	8407	8081	8365
2436	2450	2538	4521	4359	4475	8270	8074	8304

Chapter 5

Conclusion

5.1 Summary and contributions

This thesis describes a study of three combinatorial optimization problems from the are of manufacturing systems. For each of the three problems, an analysis and categorization of the problem was performed. Then, an algorithm was developed to solve the problem. Finally, the algorithm was tested on data that was either real data obtained from our industrial partner or artificially generated problem instances. The main contributions of this thesis are enumerated in the following text.

For the production scheduling problem with earliness/tardiness penalties presented in Chapter 2, the contributions are:

- Design of the search tree initialization procedure for the earliness/tardiness scheduling problems.
- Definition of the time reversing transformation for the earliness/tardiness scheduling problems.

For the problem of reoptimization of the component allocation in surface mount technology assembly system presented in Chapter 3, the contributions are:

- Inclusion of the nozzle selection problem in the reallocation problem, whereas in other approaches, either nozzle optimization is performed later as a part of the sequencing problem, or different nozzle types are not considered at all.

- The component reallocation problem is solved using a complete search algorithm, whereas the related papers usually consider only single local search moves for the reoptimization of the component allocation.

For the permutation flow-shop problem with blocking, the contributions are:

- Design of a large neighborhood search algorithm for solving the PFSP.
- Definition of a new algorithm for computation of lower bound to be used in the LNS algorithms.

5.2 Future research

The conclusion of this work regarding the future research is that in scheduling problems from real manufacturing process, the storage space between the working stations is a scarce resource. Another finding that emerged from consultations with a colleague from the cooperating industrial partner is that in a mass production supply chain, the payment of tardiness cost usually would have a liquidation result to the supplier. Therefore, in the future research in the field of scheduling in manufacturing systems, the appropriate scheduling problem to deal with would be the earliness/tardiness scheduling problem with limited buffers, or a scheduling problem with only the earliness part of the objective.

Bibliography

AMETIST, 2002. European Community Project IST-2001-35304 (Advanced Methods for Timed Systems). `http://ametist.cs.utwente.nl/`.

Ammons, J. C., Carlyle, M., Cranmer, L., DePuy, G., Ellis, K., McGinnis, L. F., Tovey, C. A., Xu, H., 1997. Component allocation to balance workload in printed circuit board assembly systems. IIE Transactions 29, 265–275.

Ayob, M., Kendall, G., 2008. A survey of surface mount device placement machine optimisation: Machine classification. European Journal of Operational Research 186, 893–914.

Baker, K. R., Scudder, G. D., 1990. Sequencing with earliness and tardiness penalties: A review. Operations Research 38 (1), 22–36.

Ball, M. O., Magazine, M. J., 1988. Sequencing of insertions in printed circuit board assembly. Operations Research 36 (2), 192–201.

Baptiste, P., Le Pape, C., Nuijten, W., 1995. Constraint-based optimization and approximation for job-shop scheduling. In: AAAI-SIGMAN Workshop, IJCAI-95.

Baptiste, P., Le Pape, C., Nuijten, W., 2001. Constraint-Based scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer Academic Publishers.

Barták, R., Salido, M. A., Rossi, F., 2009. Constraint satisfaction techniques in planning and scheduling. Journal of Intelligent Manufacturing. doi:10.1007/s10845-008-0203-4.

Beck, J. C., Perron, L., 2000. Discrepancy-bounded depth first search. In: CP-AI-OR 2000.

Beck, J. C., Refalo, P., 2002. Combining local search and linear programming to solve earliness/tardiness scheduling problems. In: CP-AI-OR 2002.

Beck, J. C., Refalo, P., 2003. A hybrid approach to scheduling with earliness and tardiness costs. Annals of Operations Research 118 (1–4), 49–71.

Behrmann, G., Brinksma, E., Hendriks, M., Mader, A., 2005. Production scheduling by reachability analysis - a case study. In: WPDRTS 2005. IEEE Computer Society Press.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Werglarz, J., 2001. Scheduling Computer and Manufacturing Processes, 2nd Edition. Springer-Verlag, Berlin.

Caraffa, V., Ianes, S., Bagchi, T. P., Sriskandarajah, C., 2001. Minimizing makespan in a blocking flowshop using genetic algorithms. International Journal of Production Economics 70, 101–115.

Carlier, J., Pinson, E., 1990. A practical use of jackson's pre-emptive schedule for solving the job-shop problem. Annals of Operations Research 26, 269–287.

Companys, R., Mateo, M., 2007. Different behaviour of a double branch-and-bound algorithm on $fm||block||cmax$ problems. Computers and Operations Research 34 (4), 938–953.

Danna, E., Perron, L., 2003. Structured vs. unstructured large neighborhood search: A case study on job-shop scheduling problems with earliness and tardiness costs. In: CP 2003. pp. 817–821.

Danna, E., Rothberg, E., Le Pape, C., 2005. Exploring relaxation induced neighborhoods to improve MIP solution. Mathematical Programming 102 (1), 71–90.

Dechter, R., 2003. Constraint Processing. Morgan Kaufmann Publishers, San Francisco.

Du, J., Leung, J. Y.-T., 1990. Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research 15 (3), 483–495.

Ehrgott, M., 2005. Multicriteria Optimization, 2nd Edition. Springer, Berlin.

El Sakkout, H., Wallace, M., 2000. Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints 5 (4), 359–388.

Gargani, A., Refalo, P., 2007. An efficient model and strategy for the steel mill slab design problem. In: Principles and Practice of Constraint Programming – CP 2007. pp. 77–89.

Gecode Team, 2010. Gecode: Generic constraint development environment. Available from `http://www.gecode.org`.

Gilmore, P. C., Gomory, R. E., 1964. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. Operations Research 12, 655–679.

Grabowski, J., Pempera, J., 2000. Sequencing of jobs in some production system. European Journal of Operational Research 125, 535–550.

Grabowski, J., Pempera, J., 2007. The permutation flow shop problem with blocking. a tabu search approach. Omega 35, 302–311.

Graham, R. L., Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 5, 287–326.

Hall, N. G., Sriskandarajah, C., 1996. A survey of machine scheduling problems with blocking and no-wait in process. Operations Research 44, 510–525.

Harvey, W. D., Ginsberg, M. L., 1995. Limited discrepancy search. In: IJCAI-95. pp. 607–615.

Hoogeveen, H., 2005. Multicriteria scheduling. European Journal of Operations Research 167 (3), 592–623.

ILOG, 2002. Ilog OPL Studio 3.6 Language Manual.

ILOG, 2005. Ilog Cplex 9.1 User's Manual.

Kodek, D. M., Krisper, M., 2004. Optimal algorithm for minimizing production cycle time of a printed circuit board assembly line. International Journal of Production Research 42 (23), 5031–5048.

Kulak, O., Yilmaz, I. O., Günther, H.-O., 2008. A GA-based solution approach for balancing printed circuit assembly line. OR Spectrum 30 (3), 469–491.

Laborie, P., 2003. Algorithm for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. Artificial Intelligence 143 (2), 151–188.

Lestein, R., 1990. Flowshop sequencing problems with limited buffer storage. International Journal of Production Research 28 (11), 2085–2100.

Levner, E. M., 1969. Optimal planning of parts machining on a number of machines. Automation and Remote Control 12, 1972–1978.

Loeschmann, S., Ludewig, D., 2003. Case study 4: Detailed description of the problem – model of a lacquer production. AMETIST Deliverable 3.4.1.

Lomnicki, Z. A., 1969. A branch-and-bound algorithm for the exact solution of the three-machine scheduling problem. Operational Research Quarterly 16, 89–100.

Luh P. B. et al., 1998. Job shop scheduling with group-dependent setups, finite buffers, and long time horizon. Annals of Operations Research 78, 233–259.

McCormick, S. T., Pinedo, M. L., Shenker, S., Wolf, B., 1989. Sequencing in an assembly line with blocking to minimize cycle time. Operations Research 37, 925–936.

Mercier, L., Van Hentenryck, P., 2008. Edge Finding for Cumulative Scheduling. Informs Journal on Computing 20 (1), 143–153.

Nawaz, M., Enscore, E. E., Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11 (1), 91–95.

Nuijten, W., Aarts, E., 1996. A computational study of constraint satisfaction for multiple capacitated job shop scheduling. European Journal of Operational Research 90 (2), 269–284.

Ohno, T., 1988. Toyota Production System: Beyond Large-Scale Production. Productivity Press, New York.

Petri, C. A., 1962. Kommunikation mit automaten. Ph.D. thesis, University of Bonn.

Pinedo, M. L., 1995. Scheduling: Theory, Algorithms, and Systems. Prentice-Hall, New Jersey.

Raduly-Baka, C., Knuutila, T., Johnsson, M., Nevalainen, O. S., 2008. Selecting the nozzle assortment for a gantry-type placement machine. OR Spectrum 30 (3), 493–513.

Reddi, S. S., Ramamoorthy, C. V., 1972. On the flow-shop sequencing problem with no wait in process. Operations Research Quarterly 23, 323–331.

Ribas, I., Companys, R., Tort-Martorell, X., 2011. An iterated greedy algorithm for the flowshop scheduling problem with blocking. Omega 39, 293–301.

Ronconi, D. P., 2005. A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking. Annals of Operations Research 138, 53–65.

Ronconi, D. P., Armentano, V. A., 2001. Lower bounding schemes for flowshops with blocking in-process. Journal of the Operational Research Society 52, 1289–1297.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: Principles and Practice of Constraint Programming – CP 1998. pp. 417–431.

Sun, D.-S., Lee, T.-E., Kim, K.-H., 2005. Component allocation and feeder arrangement for a dual-gantry multi-head surface mounting placement tool. International Journal of Production Economics 95 (2), 245–264.

Taillard, E., 1993. Benchmarks for basic scheduling problems. European Journal of Operational Research 64 (2), 278–285.

Tirpak, T. M., Nelson, P. C., Aswani, A. J., 2000. Optimization of revolver head smt machines using adaptive simulated annealing. In: IEEE/CPMT Int'l Electronics Manufacturing Technology Symposium. pp. 214–220.

Watkins, R. E., Cochran, J. K., 1995. A line balancing heuristic case study for existing automated surface mount assembly line setups. Computers & Industrial Engineering 29 (1–4), 681–685.

List of Author's Publications

Kelbel, J., Hanzálek, Z., 2006. A case study on earliness/tardiness scheduling by constraint programming (**co-authorship 50%**). In: Proceedings of the CP 2006 Doctoral Programme. pp. 108–113.

Kelbel, J., Hanzálek, Z., 2007. Constraint programming search procedure for earliness/tardiness job shop scheduling problem (**co-authorship 50%**). In: In Proceedings of the 26th Workshop of the UK Planning and Scheduling Special Interest Group. pp. 67–70.

Kelbel, J., Hanzálek, Z., 2008. Feeder setup optimization in smt assembly (**co-authorship 50%**). In: In Proceedings of the 21st International FLAIRS Conference. pp. 575–576.

Kelbel, J., Hanzálek, Z., 2011. Solving production scheduling with earliness/tardiness penalties by constraint programming (**co-authorship 50%**). Journal of Intelligent Manufacturing 22 (4), 553–562.

Kelbel, J., Hanzálek, Z., 2012. Reoptimizing component allocation in surface mount technology assembly system (**co-authorship 50%**). Submitted to journal – in review.