

# A Hybrid Meta-Heuristic for the Batching Problem in Just-In-Time Flow Shops

MESUT YAVUZ<sup>1,★</sup>, ELIF AKCALI<sup>2</sup> and SULEYMAN TUFEKCI<sup>2</sup>

<sup>1</sup>*Research and Engineering Education Facility, University of Florida, 1350 N. Poquito Rd., Shalimar, FL 32579, USA. e-mail: yavuzm@ufl.edu*

<sup>2</sup>*Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL 32611, USA. e-mail: akcali@ise.ufl.edu, tufekci@ise.ufl.edu*

(Received 30 November 2004; in final form 4 November 2005)

**Abstract.** This paper is concerned with a batching problem encountered in the context of production smoothing in just-in-time manufacturing systems. The manufacturing system of interest is a multi-level system with a flow-shop at the final level. We develop a hybrid meta-heuristic method to solve the batching problem, which is known to be NP-hard. We hybridize strategic oscillation (SO) and path re-linking (PR) methods and compare the hybrid method's performance to two benchmark methods: a bounded dynamic programming method developed for the problem earlier and an implementation of robust tabu search (RTS) meta-heuristic. Through a computational study, we show that the proposed hybrid method is effective in solving the problem within several minutes of computer time and yielding near-optimal results.

**Mathematics Subject Classifications (2000):** 90B30, 90C27, 90C59.

**Key words:** production smoothing, just-in-time manufacturing, meta-heuristics, tabu search, strategic oscillation, path relinking.

## 1. Introduction

After decades of research on meta-heuristics, a general conclusion is that no particular method is guaranteed to perform better than all the others on all kinds of problems. A meta-heuristic method may perform better on some problems and worse on others. Enriching a certain meta-heuristic method by incorporating some components taken from another method or creating a hybrid meta-heuristic method that amalgamates components of two or more different methods has been shown to be a useful strategy in developing effective solution methods for some computationally difficult problems. In this work, we focus on two meta-heuristic methods, namely strategic oscillation (SO) and path re-linking (PR), and hybridize them in a new context. The idea of developing this hybrid method came into our attention while we were implementing the SO and PR methods separately, in an earlier study. Relative weakness of SO for the specific problem

---

★ Corresponding author.

led us to ask ourselves if we could improve its performance by replacing the local search tool of the SO with that of the PR method. Also, while engineering the algorithm of the PR method, we found out that including an infeasible solution in the population enhanced the performance of the method. Thus, we consider an SO enhanced PR method. Our method includes infeasible solutions in the population, uses SO to generate the infeasible members of the population and uses PR to evaluate current population and generate new solutions for the next population. This paper demonstrates that this hybrid approach is effective in solving a hard optimization problem. As a benchmark method, we implement robust tabu search (RTS). We show that RTS falls short in solving the problem and our hybrid approach performs significantly better.

The structure of this paper is as follows. In Section 2, we review the related work and in Section 3, introduce the problem that we consider. In Sections 4 and 5 we present the RTS and our hybrid solution method, respectively. In Section 6, we present our experimentation plan and the results obtained on the performance of the proposed hybrid method, and discuss the results obtained. Finally, in Section 7 we make some concluding remarks.

## 2. Related Work

A majority of meta-heuristic techniques (all trajectory based methods and some population based methods) are based on local search strategies, and require first finding a set of initial solutions and then obtaining a better set of solutions, using the neighborhoods of the solutions at hand. Evaluation of the neighborhood and moving to a selected neighbor solution constitute an iteration of the local search framework. In contrast to *naïve* local search approaches, meta-heuristics do not necessarily stop when no improving neighbor solution can be found, but when the termination criteria set for the technique are satisfied. Meta-heuristics allow moves to solutions with worse objective function values in order to prevent premature convergence to a local optimum solution. Meta-heuristics can employ some problem-specific heuristics in the initialization step, and use these solutions obtained as starting points for the local searches. For more on meta-heuristic techniques, we refer the reader to Blum and Roli [4], Glover and Kochenberger [12], Reeves [24] and Osman and Laporte [23]. Also, Talbi [28] gives an excellent taxonomy of hybrid meta-heuristics. In the remainder of this section, we concentrate on the RTS, SO and PR methods.

Tabu search (TS) was proposed by Glover [8] in 1980s and has been successfully implemented on numerous combinatorial optimization problems. The main principle of TS is to perform a neighborhood search and keep a history of the search process. During the search, moves to solutions with worse objective function values are allowed so that the search process does not get trapped in local optima. In order to avoid cyclic moves during the process, information about the moves performed is stored in short-term memory and defined as tabus

of the search. Moves which violate the tabus are not allowed. This property enables the method to explore different sub-regions of the feasible region. An advanced version of the TS is the so-called robust TS (RTS) [27]. In RTS, the tabu tenure is periodically changed during the search process. This increases the diversification of the search process. We discuss the design issues and present our implementation of the RTS method in Section 4. For more reading on the (R)TS, we refer the reader to [13] and the references therein.

SO is a meta-heuristic method that was introduced primarily as a diversification tool for TS. The idea behind SO is to drive the search towards and away from boundaries of feasibility [5, 15]. SO performs local search until hitting a boundary of feasibility. Then, it crosses over the boundary and proceeds into the infeasible region for a certain number of moves, that is, it continues the local search in the infeasible region. Then, a search in an opposite direction is performed to re-enter into the feasible region. Alternating between the local searches in the feasible and infeasible regions continuously during the search process creates some form of oscillation, which gives its name to the method [3, 5, 11, 15].

PR is a population based meta-heuristic. It keeps a population of solutions on hand, creates candidate solutions by combining currently known solutions at each iteration, and selects the fittest solutions to be kept in the population for the next iteration. The original form of PR consists of three stages [7]. In the first stage, a starting set of solution vectors is generated by using heuristic methods designed for the problem and a subset of the best vectors is selected to be the *reference set*. In the second stage, subsets of the reference set, each consisting of two solutions, are generated. One of the solutions is identified as the *initiating solution* and the other as the *guiding solution*. A distance measure based on the neighborhood structure is utilized, and the a path from the initiating solution to the guiding solution is created. The creation of the path is essentially an iterative neighborhood search process evaluating the neighbor solutions that decrease the distance to the guiding solution. Finally, the candidate solutions created in the second stage and the original reference solutions are evaluated and the fittest solutions are included in the reference set for the next iteration. The last two stages are repeatedly applied for a pre-determined number of iterations. Recently, PR has been successfully applied to various network design and assignment problems [1, 2, 6, 9, 10, 22, 25, 26, 29].

### 3. Problem Description

Just-in-time (JIT) manufacturing is among the most successfully applied and widely accepted manufacturing philosophies of the past several decades. The main idea behind the JIT philosophy is to manufacture products when they are required and in the required quantities. Minimization of the inventories, use of pull-systems, reduction of setup times and employment of smooth production

schedules are the best recognized aspects of the JIT. Pull systems, e.g., Kanban systems, are used to subordinate the production to customer demands. That is, when a customer buys a product, a request is placed to the final level of the production system to manufacture the product. Similarly, when the final level starts the production, the required parts are pulled from the previous levels (upstream operations), and so on. The movement of parts between consecutive levels are controlled by Kanban cards that contain the information of which product to produce and in what quantities. Zero inventory is an ultimate ideal of JIT. Ideally situation, the production is synchronized such that there is no need to keep inventories within the system. Actually, the ideal of zero-inventories is unattainable, unless instantaneous production can be established. Achieving smooth production schedules is extremely important in mixed-model manufacturing systems. For further reading on JIT, we refer the reader to [19–21].

Mixed-model manufacturing systems are widely used in industries that have a large number of products, each with low demand volumes. Automotive and electronic industries are the prominent examples. Operating a mixed-model manufacturing system under the just-in-time (JIT) philosophy brings several important challenges such as determining batch sizes, specifying the number of Kanban cards used for production authorization and sequencing the products. As far as the sequencing task is considered, the main objective is to disperse the processing of different task products over the planning horizon, as uniformly as possible [18, 19].

Using a pull system, multi-level JIT manufacturing systems are typically controlled by scheduling the final level only. As the final level (assembly or manufacturing of the end products) is scheduled, required components, parts and raw materials are withdrawn from upstream operations, defining the demand and also the schedules for these upstream operations. Therefore, the schedule of the final level defines the material movements and operations in the entire system.

The final level of a manufacturing system can be a single machine, a flow shop, a job shop or an assembly line. A majority of the existing production smoothing literature considers assembly lines. Recently, the single machine [30, 32] and flow shop [31] problems have also been studied. In this paper, we consider a flow shop system as the final level. That is, each of a number of different models follow the same route, but have arbitrary non-zero setup and processing time requirements on the machines. Our work is motivated by St. Petersburg, Florida facility of a leading electronic contract manufacturer. The volumes are so low that no distinct product can be manufactured on a line for a long period, such as a week, but the production runs require several changeovers in a day. St. Petersburg facility has two main flow lines, each consisting of the following machines: board loader, board printer, glue dots, HSP #1, HSP #2, GSM #1, GSM #2 and re-flow. Although there exists a line structure and the line performs the assembly of the final products, these lines are not assembly-lines in the traditional sense. Different machines have different setup and processing times for different

products. Significant setup times are incurred when the line switches from one product to another and there exist storage areas between some of the successive machines. In this environment, the traditional production smoothing methods are not applicable. Therefore, there exists a need for new and efficient methods.

We consider a two-phase solution methodology for the problem. We determine the batch sizes and number of batches to produce of each model in the first phase. In the second phase, we sequence these batches in order to uniformly disperse the batches over the planning horizon. We call these two phase problems the batching problem (BP) and the sequencing problem (SP), respectively. For further reading we refer the reader to [30, 32] where the problem on a single-machine type environment and the two-phase solution methodology are established, and [31] where the authors define the problem on flow-shops. It is shown in [31] that the SP is polynomially solvable and the BP is NP-hard. The authors focus on the SP and develop a bounded dynamic programming (BDP) procedure for its exact solution. In our paper, we too focus on the BP and propose a hybrid meta-heuristic to solve it.

The key to the two-phase approach adopted is the so-called *takt-time*. Each batch should be processed within the takt-time, the length of which is obtained by dividing the total available time into equal intervals, on each machine. Thus, as each unit of the takt-time elapses, each batch proceeds to its next operation. Both the number of batches and batch sizes are assumed to be integers, which reduces the possibility of satisfying the demand in exact quantities. Therefore, in all solution procedures, either excess production or underproduction should be allowed. In our work, we do not allow under production, but accept excess production in minimal amount, as stated in the following model.

### 3.1. MATHEMATICAL MODEL

Let  $N = \{1, 2, \dots, n\}$  be the set of products, indexed by  $i$ , and  $M = \{1, 2, \dots, m\}$  be the set of machines, indexed by  $j$ . Let  $s_{i,j}$  and  $p_{i,j}$  denote the setup and unit processing times of product  $i$  on machine  $j$ , respectively, and  $d_i$  the demand for product  $i$  in the planning horizon. Batch size and number of batches of product  $i$  to be manufactured in the planning horizon are denoted by  $b_i$  and  $q_i$ , respectively.  $Q = \sum_{i \in N} q_i$  is the total number of batches to be manufactured in the planning horizon and  $T$  is total available time in the planning horizon. Here, all the machines are assumed to have the same amount of available time. Also note that,  $\lceil x \rceil$  denotes the smallest integer that is greater than or equal to  $x$ . The batching problem is formulated as follows.

$$(P1) \quad \text{minimize} \quad F = \sum_{i=1}^n \frac{\left(\lceil \frac{d_i}{q_i} \rceil\right)^2 \left( \left( \sum_{h=1}^n q_h \right)^2 - q_i^2 \right)}{\sum_{h=1}^n q_h} \quad (1)$$

subject to

$$\left( s_{i,j} + p_{i,j} \left\lceil \frac{d_i}{q_i} \right\rceil \right) \sum_{h=1}^n q_h \leq T, \quad \forall i, j \quad (2)$$

$$\left\lceil \frac{d_i}{q_i} \right\rceil = b_i, \quad \forall i \quad (3)$$

$$q_i = \left\lceil \frac{d_i}{b_i} \right\rceil, \quad \forall i \quad (4)$$

$$1 \leq q_i \leq d_i, \quad q_i \text{ integer}, \quad \forall i \quad (5)$$

The objective function (1) is derived from a lower bound to the SP [32, 33]. In other words, the objective function of the first phase is estimation of the objective function of the second phase. Constraint set (2) assures that each product is batched such that the sum of setup and processing time of the batch is not longer than the takt-time. Minimization of the excess production quantities is guaranteed by constraints (3) and (4). In enforcing these rules, we define a set of *acceptable values* for the decision variable  $q_i$ .

Acceptable values of a decision variable ( $q_i$ ) are the integer values that satisfy the equation  $q_i = \lceil d_i / \lceil d_i / q_i \rceil \rceil$ . Let  $A_i = \{r_{i,1}, \dots, r_{i,a_i}\}$  be the complete set of acceptable values of  $q_i$  where  $r_{i,h_i}$  is the  $h_i^{\text{th}}$  acceptable value and  $a_i$  is the cardinality of the set. For any  $q_i \notin A_i$ , there exists an  $r_{i,j} \in A_i$  such that  $r_{i,j}$  consumes less resource time and yields smaller excess production, therefore preferred over  $q_i$ . Algorithm Find Acceptable Values finds all the acceptable value set  $A_i$  for each product  $i \in N$  (see Figure 1).

For example, if demand data for a two-product problem is given as  $d_1 = 50$  and  $d_2 = 10$ , the above algorithm yields the acceptable values presented in Table I. The first 10  $q_1$  values are found easily, each time the condition in the third step of the algorithm is evaluated to be *true*, and the next acceptable value is obtained by increasing  $q_1$  by one, as described in step 4. For the rest of the acceptable values, the condition is evaluated to be *false*, therefore obtaining them requires executing step 5, where the  $b_1$  value is first decreased by one and then the

**Algorithm Find Acceptable Values**

For each  $i \in N$

- ```

{
  1. Start with an empty list:  $A_i = \emptyset$ ,  $a_i = 0$ . Set  $q_i = 1$  and  $b_i = d_i$ .
  2. Add  $q_i$  to the list:  $a_i \leftarrow a_i + 1$ ,  $r_{i,a_i} \leftarrow q_i$ ,  $A_i \leftarrow A_i \cup \{r_{i,a_i}\}$ .
   IF ( $q_i = d_i$ ) THEN STOP.
  3. IF ( $\lceil d_i / (q_i + 1) \rceil < b_i$ )
  4. THEN Set  $q_i = q_i + 1$  and update  $b_i = \lceil d_i / q_i \rceil$ .
  5. ELSE Set  $b_i = b_i - 1$  and update  $q_i = \lceil d_i / b_i \rceil$ .
  6. Return to Step 2.
}

```

Figure 1. Pseudo-code for Algorithm Find Acceptable Values.

corresponding  $q_1$  value is calculated. The same steps are applied for the second product and the sets shown in Table I are obtained. To demonstrate the above discussion, we present why, for example,  $q_1 = 20$  is not an acceptable value. If  $q_1 = 20$ , the characteristic equation is not satisfied,  $\lceil 50/\lceil 50/20 \rceil \rceil = \lceil 50/3 \rceil = 17 \neq 20 = q_1$ . Therefore,  $q_1 = 20$  is not an acceptable value. However, for  $q_1 = 17$  instead, we see that the equation is satisfied,  $\lceil 50/\lceil 50/17 \rceil \rceil = \lceil 50/3 \rceil = 17 = q_1$ , proving that  $q_1 = 17$  is an acceptable value.

### 3.2. NEIGHBORHOOD STRUCTURE

We define a *solution*  $q = (q_1, q_2, \dots, q_n)$  as a vector of the decision variables such that all the decision variables take an acceptable value  $q_i \in A_i, \forall i$ . We further distinguish between feasible and infeasible solutions as follows. A solution is *feasible* if it satisfies the first constraint set (2), otherwise it is *infeasible*.

Now, consider the following example with  $n = 2$  products and  $m = 2$  machines. Let  $d_1 = 15$  and  $d_2 = 20$ ;  $s_{1,1} = s_{1,2} = s_{2,1} = 1$ ,  $s_{2,2} = 2$ ,  $p_{1,1} = p_{1,2} = 1$ ,  $p_{2,1} = p_{2,2} = 0.5$  and  $T = 50$  minutes. The procedure proposed in Section 3.1 to find the acceptable values implies  $q_1 \in A_1 = \{1, 2, 3, 4, 5, 8, 15\}$  and  $q_2 \in A_2 = \{1, 2, 3, 4, 5, 7, 10, 20\}$ . Any pair of these acceptable values is a solution, for example (1, 1), (5, 5) and (5, 20) are all solutions. (5, 5) is a feasible solution, since the batch sizes are 3 and 4 and these batches take 4 and 3 min on the first machine and 4 min each on the second machine. The takt time is  $50/(5 + 5) = 5$  minutes, therefore both batches can be processed within the takt time, on each machine. Similarly, (5, 20) requires 4 and 1.5 min to process the batches on the first machine and 4 and 2.5 min on the second machine. However, the takt time is too short ( $50/(5 + 20) = 2$  min), thus this solution is infeasible.

A solution  $q^1 = (q_1^1, q_2^1, \dots, q_n^1)$  is a *neighbor* of  $q^0 = (q_1^0, q_2^0, \dots, q_n^0)$  if and only if exactly one variable value is different in these vectors, and the categorical distance between the values of this decision variable is at most  $\rho$ , where  $\rho$  is a user defined integer that is greater than or equal to one. If we denote the set of neighbor solutions of a solution  $q^0$  with  $NS(q^0, \rho)$  and consider  $q^0 = (5, 5)$  and  $\rho = 2$  for example, then the neighbor solutions set of  $q^0$  is  $NS((5, 5), 2) = \{(3, 5), (4, 5), (8, 5), (15, 5), (5, 3), (5, 4), (5, 7), (5, 10)\}$ . With this definition, a solution may have at most  $2 \times \rho \times n$  neighbors.

For further use we define *the farthest corner of the solution space*. It is the solution for which every decision variable takes its largest value, that is

Table I. Acceptable values for  $d_1 = 50$  and  $d_2 = 10$ .

| $i$ | $A_i$                                           | $a_i$ |
|-----|-------------------------------------------------|-------|
| 1   | {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 17, 25, 50} | 14    |
| 2   | {1, 2, 3, 4, 5, 10}                             | 6     |

$q_i = d_i, \forall i \in N$ . If we relax integrality of batch sizes, and let  $r_i = q_i/Q$  where  $0 \leq r_i \leq 1$  such that  $\sum_{i \in N} r_i = 1$  denote the proportion of the number of batches of a certain product to the total number of batches, and assume these proportions ( $r_i$ s) are fixed, then the objective function (1) becomes  $\sum_{i \in N} (d_i/r_i)^2 (1 - r_i^2)/Q$ . This shows that larger  $Q$  values are expected to yield better solutions. We can intuitively argue that the global optimum may be located in the vicinity of the farthest corner of the solution space. Although this particular solution may be infeasible, guiding the search process towards this farthest corner might help us finding the global optimum.

#### 4. Tabu Search

We describe our implementation of RTS in two phases: initialization and improvement. In the initialization phase, the method generates an initial solution to start the search process. We employ four problem specific heuristics (adapted from [30]) to generate an initial solution. The initialization phase is completed with creating an empty tabu list. The improvement phase consists of successive iterations of neighborhood evaluations and moves. The neighborhood function (defined in Section 3.2) generates at most  $2 \times n$  solutions. We evaluate the entire neighborhood and move to the best neighbor. Note that this is the usual neighbor selection strategy for RTS.

When a move is performed, the tabu list is updated, completing an iteration. We employ two alternative tabu list structures. The first one includes the product indices ( $i \in N$ ). When a move is performed, the index of the product the number of batches for which is changed is added to the tabu list. The second structure is much less restrictive, in that it stores the number of batches as well as the product index ( $i \in N$  and  $q_i \in A_i$ ). This way not all the moves related to a product but the moves to the same number of batches of that product are blocked. For the tabu tenure, we adopt a dynamic tabu tenure management strategy and implement the RTS proposed by [27]. The minimum and maximum tabu tenure values are given by two parameters: `Minimum Tabu Tenure` and `Maximum Tabu Tenure`. After every  $2 \times \text{Maximum Tabu Tenure}$  iterations, we randomly select the tabu tenure between the minimum and maximum values. We present a pseudo-code algorithm for the RTS in Figure 2.

##### 4.1. DESIGN ISSUES

Aspiration criteria are the criteria used to enable RTS to move to some solutions that are blocked by the tabu list. We consider two aspiration criteria. The first one is based on objective function value and allows to move to the solutions that yield better objective function values than the best solution found so far. The

**Algorithm Robust TS**

Initialization  
 Generate an initial solution.  
 Initialize `iteration`  $\leftarrow 0$ .  
 Initialize `Tabu List`  $\leftarrow \emptyset$ .

Improvement  
 {  
   If  $\text{Mod}(\text{iteration}, 2 * \text{Maximum Tabu Tenure}) = 0$   
   {  
     Randomly select `Tabu Tenure` from  
     {`Minimum Tabu Tenure`, ..., `Maximum Tabu Tenure`}.  
   }  
   Evaluate neighbor solutions.  
   Select a neighbor solution and move to that solution.  
   Update `Tabu List`.  
 }  
 Iterate until `Termination Criterion` is satisfied.

Figure 2. Pseudo-code for Algorithm Robust TS.

second one is based on feasibility. Since finding feasible solutions is a difficult task in our problem, we allow moves to a solution which is in the tabu list if it is the only feasible solution in the neighborhood.

For the termination of the method, we use two simultaneous criteria. The total number of iterations should be equal or more than a designated number and the relative improvement obtained in the last pre-determined number of iterations should not exceed a pre-determined percentage.

Handling the infeasible solutions is another important aspect of an RTS implementation. Two of the most popular approaches are rejection and repair of the infeasible solutions. In rejection, the algorithm does not evaluate infeasible neighbors. In repair, on the opposite, the infeasible neighbors are converted to feasible solutions using a feasible solution search mechanism. Our preliminary results show that repair policy is outperformed by the rejection policy in our problem. Therefore, we consider rejections of the infeasible solutions only and impose the search to perform moves to feasible solutions only.

One final design issue we address is using a single or multiple runs of the RTS. In the single-start method, we select the best solution obtained using the problem specific heuristics and start the RTS from that solution. In the multi-start alternative, we perform an RTS run starting from each distinct solution obtained using the problem specific heuristics.

We have coded RTS in a flexible manner, which enables us to test different settings. The parametric representation of the method is  $\text{RTS}(\text{IterLim}, \text{TabuStructure}, \text{MinimumTabuTenure}, \text{MaximumTabuTenure}, \text{EvaluateInfeasibles}, \text{MultiStart})$ .

## 5. A Hybrid Method

In an earlier work, we considered the batching problem on a single machine, and implemented SO, scatter search and PR meta-heuristics independently [30]. Our study showed that PR yielded excellent quality solutions to the problem, within several minutes. In implementing PR, we introduced a novel idea, in that we included the farthest corner of the solution space (which is an infeasible solution) in the reference solutions set. Traditionally, the reference solutions are selected from the feasible solutions only. Presence of the farthest corner in the reference set opposes the tradition, as it uses an infeasible solution and drives the search process into the infeasible region.

The SO method, on the other hand, uses the infeasible solutions in a more systematic way as it mandates the search process to seek infeasible solutions at each iteration. SO has the ability to search non-convex or fragmented feasible solution spaces efficiently. We build our hybrid method upon the effective components of both methods. The general structure of the hybrid method is adopted from the PR procedure. We use core mechanisms of PR such as *diversification method*, *improvement method*, *reference set update method*, *subset generation method*, and *solution combination method*; and also use an *oscillation method*.

The method starts with generating initial solutions by employing simple heuristic methods devised for the problem. Without loss of generality, we can assume the initial solutions are created randomly. The initial solutions are evaluated and processed by the diversification method in an initialization phase. This initialization phase generates the initial *feasible reference set*. Then, the hybrid method starts its iterative phase, by invoking the oscillation method. The oscillation method uses all the solutions in the feasible reference set as initial solutions, searches for infeasible solutions with lower objective function values in their neighborhood, and generates the *infeasible reference set* as an outcome. Then, an initiating solution from the feasible reference set and a guiding solution from the infeasible reference set are selected by the subset generation method. These two solutions are combined and the intermediate feasible solutions lying on the combining paths are considered as candidates for the feasible reference set. An iteration ends with improving the solutions in the feasible reference set, through local search.

We define our termination criterion according to the update of the reference sets. If, at the end of an iteration, the reference sets remain unchanged (no improving solutions are found during an iteration), then the method terminates. During our preliminary tests of the method, we used a second termination criterion based on total number of iterations, but the number of iterations performed never reached large numbers. Therefore, we eliminate the second criterion and use the method with only one termination criterion based on the update of the reference sets. We provide a pseudo-code algorithm for our approach in Figure 3.

**Algorithm Hybrid SO/PR**

Initialization

Initialize the **Reference Set** with seed solutions, using problem specific heuristics.

For each seed solution

```
{
    Create all diversified solutions of the seed solution on hand.
    For each diversified solution
    {
        Find a local optimum using the Improvement Method.
        Update the Feasible Reference Set.
    }
}
```

Improvement

For each feasible reference solution

```
{
    Find an infeasible solution using the Oscillation Method.
    Update the Infeasible Reference Set.
}
```

Generate subsets using the **Subset Generation Method**.

For each subset

```
{
    Create combinations of the solutions in the subset.
    For each combination
    {
        Find a local optimum using the Improvement Method.
        Update the Reference Set.
    }
}
```

Iterate until **Termination Criterion** is satisfied.

*Figure 3.* Pseudo-code for Algorithm Hybrid SO/PR.

### 5.1. DIVERSIFICATION METHOD

The diversification method is used following the initialization of the feasible reference solutions set by implementing problem specific heuristics devised for the problem. The diversification method processes each initial feasible reference solution and creates a diversified solution by replacing each variable's value by its 100<sup>th</sup> next acceptable value. If for a certain variable the number of acceptable values to its right is less than 100, then the mod operator is used to determine the acceptable value to replace the original one. Since all the variables in an initial solution are used, at most  $n$  different diversified solutions are generated for each initial solution.

### 5.2. IMPROVEMENT METHOD

The improvement method we use is the well-known 'best improvement strategy.' It is based on the generic local search idea, the neighborhood of the current solution is scanned and the best solution found is selected as the next solution. This process iterates until a local optimum, with respect to the neighborhood

function used, is found. The improvement method is invoked at the end of each iteration. It processes each solution in the feasible reference set and tries to replace them with better solutions. When a better solution is found, it is taken as a candidate solution and inserted into the feasible reference set. Implementation of the improvement method is subject to two parameters, namely *SearchDepth* and  $\rho$ . If *SearchDepth* = 1, then only the immediate neighbors of the current solution are evaluated. If *SearchDepth* = 2, then the two-step neighbors, the neighbor's neighbors, are also evaluated. In general if *SearchDepth* =  $k$ , then all  $k$ -step neighbors are evaluated. The other parameter,  $\rho$ , defines the range of the evaluated neighbors and is the same as defined in Section 3.2.

### 5.3. REFERENCE SET UPDATE METHOD

We use two types of reference solution sets, one for feasible solutions and one for infeasible solutions. The reference set update method is based on measuring the quality of a candidate solution and comparing it to that of the solutions that are already in the reference set. A solution with an objective function value that is lower than at least one of the other solutions in the reference set, is inserted into the set. We define two parameters, namely *bFeasible* and *bInfeasible* to denote the sizes of the feasible and infeasible reference sets, respectively. We define a third parameter, *FarthestCorner*, for the inclusion of the farthest corner of the solution space in the infeasible reference set. The parameter takes *true* or *false* values, and if *FarthestCorner* = *true* then we include the farthest corner of the solution space into the infeasible reference set at the beginning of the first iteration. If *FarthestCorner* = *false*, then the farthest corner of the solution space can be inserted into the infeasible reference set, only if it is obtained by the oscillation method.

### 5.4. SUBSET GENERATION METHOD

The initiating and guiding solutions are selected from the feasible and infeasible reference sets, respectively. We adopt two modes of selecting solutions from the reference sets. The first one is a combinatorial approach that generates every possible pair of initiating and guiding solutions. The second mode is a probabilistic approach and selects the solutions from the reference sets randomly, where selection probabilities of the reference solutions depend on their objective function values. The probabilities are dynamically manipulated before every selection, so that no subset is created more than once.

### 5.5. SOLUTION COMBINATION METHOD

Our solution combination method starts from the initiating solution and moves toward the guiding solution, evaluating several intermediate solutions and

keeping some of them in temporary lists. Based on the acceptable values, we measure the distance between the two reference solutions with a categorical distance measure. If  $q^1$  and  $q^2$  are the initiating and guiding solution vectors, and we define the function  $Position(q_i)$  as an integer function which returns the position of variable  $i$ 's value in  $A_i$ , then the distance between these two vectors is defined as  $\sum_{i \in N} |Position(q_i^1) - Position(q_i^2)|$ , where  $|x|$  is the absolute value of  $x$ . Starting from the initiating solution, the neighbor solutions that decrease the distance to the guiding solution by one are considered and the best  $NTS$  solutions are stored in a list, where  $NTS$  is a parameter standing for the number of temporary solutions. In the next step, each solution in this list is considered as the initiating solution, and again the neighbor solutions that decrease the distance by one are evaluated. This is repeated until the guiding solution is reached, while keeping  $NTS$  best solutions between the steps.  $NTS = 1$  represents a single path between the origin and the destination. However,  $NTS > 1$  can be considered as  $NTS$  parallel paths that are built between the origin and the destination solutions.

#### 5.6. OSCILLATION METHOD

At the end of an iteration, all the solutions in the feasible reference set are used as starting solutions for an infeasible solution search, one by one. All the neighbors of the solution are evaluated and the one with the lowest objective function value, whether it is feasible or infeasible, is selected as the solution to move to. After a certain number of such moves, the method evaluates the final solution and considers it as a candidate solution for the infeasible reference set. Since the search process is repeated once for each solution in the feasible reference set, at most  $bFeasible$  candidate solutions for the infeasible reference set can be generated.

This oscillation method is very similar to the infeasible solution search used in the SO method and is the major component our hybrid approach takes from SO. The use of the oscillation method depends on a parameter,  $InfeasibleMoveDepth$ . The parameter defines the number of moves the oscillation method should perform into the infeasible region.

An advanced mechanism within the PR approach is known as tunneling and allows search in the infeasible region [14]. With tunneling, the neighborhood structure is altered and evaluation of the infeasible solutions, that lie on a path between the parent solutions, is enabled. The idea behind tunneling approach is to find shortcut paths between initiating and guiding solutions, which may enable the user to explore previously unknown fragments of the feasible region. Another important variant of tunneling allows parent solutions to be infeasible [17]. This way, the search process is driven into the infeasible region deliberately and exploration of new feasible solutions is expected. Our method keeps one feasible and one infeasible reference set. In the re-linking step, a path linking a feasible reference solution to an infeasible reference solution is sought. Clearly, our re-

linking step uses the tunneling tool, thus our hybrid method includes the tunneling approach. However, the main contribution of the SO in our hybrid method is maintaining the infeasible reference set. That is, the oscillation method finds good infeasible solutions which may not be found on the paths linking two feasible reference solutions by a PR with tunneling approach.

Two important properties of a meta-heuristic method are intensification and diversification [4, 13, 16]. Our hybrid method uses the solution combination mechanism as an intensification and diversification tool at the same time. Keeping the elite solutions in the reference sets represents intensification. In the probabilistic implementation of the method, the selection of the origin and destination solutions is also based on their objective values. This, clearly adds to the intensification aspect of the hybrid method. While combining the solutions, we evaluate numerous solutions lying on the paths and ideally obtain a diversified collection of solutions. The presence of the farthest corner of the solution space in the infeasible reference set contributes to the diversification aspect in that it drives the search process more into the infeasible region.

We denote the hybrid method with  $\text{HybridSOPR}(bFeasible, bInfeasible, ProbabilisticSelection, NTS, SearchDepth, \rho, InfeasibleMoveDepth, FarthestCorner)$ . Our preliminary experiments have shown that  $Search-Depth = 1$  and  $\rho = 2$  values are preferable over other possible values of these parameters, depending on either the computational time requirements or the resulting solution qualities. Therefore, we fix these three parameters to these values and simplify the method to  $\text{HybridSOPR}(bFeasible, bInfeasible, ProbabilisticSelection, NTS, InfeasibleMove-Depth, FarthestCorner)$ .

## 6. Computational Study

### 6.1. TEST INSTANCES

In our study we consider a flow shop which consists of two, five or 10 machines and produce 10 distinct products. Each product has an average demand of 750 units. This flow shop is part of a larger system which consists of multiple levels of sub-assemblies, components and parts manufacturing. In such a manufacturing system, 10 products and 10 machines at the final level represent a large instance of the problem, whereas two and five machine instances reflect small and medium problem sizes, respectively. These problems can be solved by a bounded dynamic programming procedure in reasonable times [31].

We use three experimental factors,  $s_i/p_i$  ratio  $\alpha$ ,  $T$  relaxation percentage  $\beta$  and diversification level  $\gamma$ .  $\gamma \in \{0, 1\}$  is used to create test cases in which different products are diversified in terms of demand, processing time and setup time.  $\gamma = 1$  reflects the diversified case, and  $\gamma = 0$  reflects the undiversified case where the products are very similar to each other. Demand values are randomly and uniformly generated between the minimum and maximum values, where

maximum demand is twice as large as the average demand for diversified instances and 20% over the average demand for the instances with similar products. The ratio of maximum demand to minimum demand is 50 and 1.5 for these two types of instances, respectively.

We use  $\alpha$  to denote the ratio between the expected values of  $s_i$  and  $p_i$  for the diversified instances. We first create  $p_i$  according to uniform distribution between  $(0,5]$ , and then  $s_i$  according to uniform distribution between  $[(1 - 0.1 \times \gamma) \times p_i \times \alpha, (1 + 0.1 \times \gamma) \times p_i \times \alpha]$ . We let  $\alpha \in \{100, 10, 1\}$  for our experiments.

Total available time should allow at least one setup per product, that is  $T \geq T_{LB} = \sum_{i \in N} (d_i p_i + s_i)$ . On the other hand,  $T$  should be limited with  $T$ . We create  $T$  with  $T = \beta T_{UB} + (1 - \beta) T_{LB}$  and let  $\beta \in \{0.2, 0.5, 0.8\}$ .

Allowing three different values for the parameters  $\alpha$  and  $\beta$  and two different values for  $\gamma$  results in 18 different problem sets. In order to obtain reliable results, 25 instances are created for each problem set, giving a total of 450 test instances for each  $m$  value and 1,350 test instances in total.

## 6.2. METHODS AND RESULTS

We first solve the test instances with the bounded dynamic programming (BDP) method described in [31], to obtain the optimal solutions. Then, we use one fifth of the test instances, that is five instances for each of the 18 problem sets, in order to find promising settings of the RTS and hybrid methods. We call this process fine tuning the methods.

According to the fine tuning results, we focus on eight alternative settings of the RTS method: RTS(200, 0, 5, 10, *False*, *False*), RTS(600, 1, 5, 10, *False*, *False*), RTS(1000, 0, 5, 10, *False*, *False*), RTS(3000, 1, 5, 10, *False*, *False*), RTS(200, 0, 5, 10, *False*, *True*), RTS(600, 1, 5, 10, *False*, *True*), RTS(1000, 0, 5, 10, *False*, *True*) and RTS(3000, 1, 5, 10, *False*, *True*). We refer to these settings as RTS1 through RTS8, respectively.

Our fine tuning on the hybrid method show that HybridSOPR(40, 30, *false*, 8, 5, *true*) is a promising setting. We also analyze the effect of the alternative *FarthestCorner* value, that is HybridSOPR(40, 30, *false*, 8, 5, *false*). Fine tuning on the probabilistic subset generation, on the other hand, shows that HybridSOPR(50, 40, *true*, 8, 5, *true*) is a promising combination. It also worthwhile to note that we adopt a policy of creating 20% of the possible subsets, in this probabilistic subset generation approach. Similarly, we analyze the effect of using *FarthestCorner* = *false*, that is HybridSOPR(50, 40, *true*, 8, 5, *false*). We denote these four alternative settings with H1, H2, H3 and H4, respectively, and test them over the entire set of test instances.

We test these 12 methods (eight RTS and four hybrid settings) over the entire set of test instances, and report average solution time, the relative deviation from the optimal solution and also relative improvement over the solutions obtained by the problem specific heuristic methods used in initialization phase, for each  $m$

value and each setting (see Table II). All the methods are coded in C++ 6.0 programming language and run on a personal computer with an Intel PentiumIII, 800 MHz processor and 128 Mb of memory.

Table II. Overall results.

| $m$ | Method | Hit percentage | Percent deviation |       | Time (s) |          | Percent improvement |
|-----|--------|----------------|-------------------|-------|----------|----------|---------------------|
|     |        |                | Avg.              | Max.  | Avg.     | Max.     |                     |
| 2   | BDP    | 100.00         | 0.00              | 0.00  | 289.64   | 1,409.09 | n/a                 |
|     | RTS1   | 83.33          | 0.40              | 23.70 | 6.38     | 8.56     | 41.24               |
|     | RTS2   | 84.00          | 0.40              | 23.70 | 6.45     | 8.64     | 42.91               |
|     | RTS3   | 84.00          | 0.40              | 23.70 | 6.51     | 8.72     | 42.95               |
|     | RTS4   | 84.44          | 0.39              | 23.70 | 6.85     | 9.05     | 45.58               |
|     | RTS5   | 84.22          | 0.40              | 23.70 | 6.50     | 8.71     | 44.02               |
|     | RTS6   | 84.44          | 0.39              | 23.70 | 6.57     | 8.69     | 47.26               |
|     | RTS7   | 84.89          | 0.39              | 23.70 | 6.72     | 8.81     | 48.17               |
|     | RTS8   | 85.11          | 0.39              | 23.70 | 7.46     | 9.52     | 48.50               |
|     | H1     | 96.44          | 0.09              | 13.43 | 20.42    | 224.82   | 89.87               |
|     | H2     | 92.44          | 0.21              | 21.35 | 12.72    | 174.59   | 81.42               |
|     | H3     | 96.39          | 0.08              | 6.69  | 11.57    | 114.47   | 89.35               |
|     | H4     | 95.03          | 0.18              | 21.35 | 9.81     | 156.59   | 85.15               |
| 5   | BDP    | 100.00         | 0.00              | 0.00  | 343.78   | 1,666.63 | n/a                 |
|     | RTS1   | 85.56          | 0.28              | 10.58 | 7.98     | 11.10    | 47.26               |
|     | RTS2   | 86.22          | 0.27              | 10.58 | 8.05     | 11.17    | 50.34               |
|     | RTS3   | 86.22          | 0.27              | 10.58 | 8.13     | 11.24    | 50.11               |
|     | RTS4   | 86.44          | 0.27              | 10.58 | 8.51     | 11.60    | 50.20               |
|     | RTS5   | 85.56          | 0.28              | 10.58 | 8.13     | 11.34    | 49.57               |
|     | RTS6   | 86.44          | 0.27              | 10.58 | 8.20     | 11.31    | 50.55               |
|     | RTS7   | 86.44          | 0.27              | 10.58 | 8.36     | 11.45    | 52.34               |
|     | RTS8   | 86.44          | 0.27              | 10.58 | 9.21     | 12.28    | 52.15               |
|     | H1     | 97.78          | 0.04              | 5.02  | 25.49    | 307.19   | 94.82               |
|     | H2     | 96.89          | 0.07              | 7.44  | 15.71    | 170.16   | 92.97               |
|     | H3     | 97.39          | 0.04              | 5.02  | 15.81    | 231.77   | 94.14               |
|     | H4     | 96.03          | 0.08              | 7.44  | 13.34    | 134.98   | 91.17               |
| 10  | BDP    | 100.00         | 0.00              | 0.00  | 606.71   | 3,543.31 | n/a                 |
|     | RTS1   | 86.44          | 0.39              | 25.23 | 10.70    | 15.71    | 48.45               |
|     | RTS2   | 86.89          | 0.39              | 25.23 | 10.78    | 15.79    | 50.70               |
|     | RTS3   | 87.33          | 0.39              | 25.23 | 10.87    | 15.87    | 52.23               |
|     | RTS4   | 87.33          | 0.39              | 25.23 | 11.31    | 16.27    | 52.23               |
|     | RTS5   | 86.89          | 0.38              | 25.23 | 10.90    | 15.99    | 51.66               |
|     | RTS6   | 87.78          | 0.38              | 25.23 | 10.94    | 15.91    | 54.57               |
|     | RTS7   | 87.78          | 0.38              | 25.23 | 11.13    | 16.09    | 54.50               |
|     | RTS8   | 87.78          | 0.38              | 25.23 | 12.07    | 16.90    | 54.99               |
|     | H1     | 97.78          | 0.12              | 8.74  | 35.03    | 304.91   | 86.31               |
|     | H2     | 95.56          | 0.18              | 9.18  | 22.60    | 240.65   | 75.14               |
|     | H3     | 97.43          | 0.12              | 8.74  | 20.44    | 549.83   | 86.18               |
|     | H4     | 96.34          | 0.15              | 8.74  | 17.39    | 760.46   | 80.05               |

A HYBRID META-HEURISTIC FOR THE BATCHING PROBLEM IN JUST-IN-TIME FLOW SHOPS

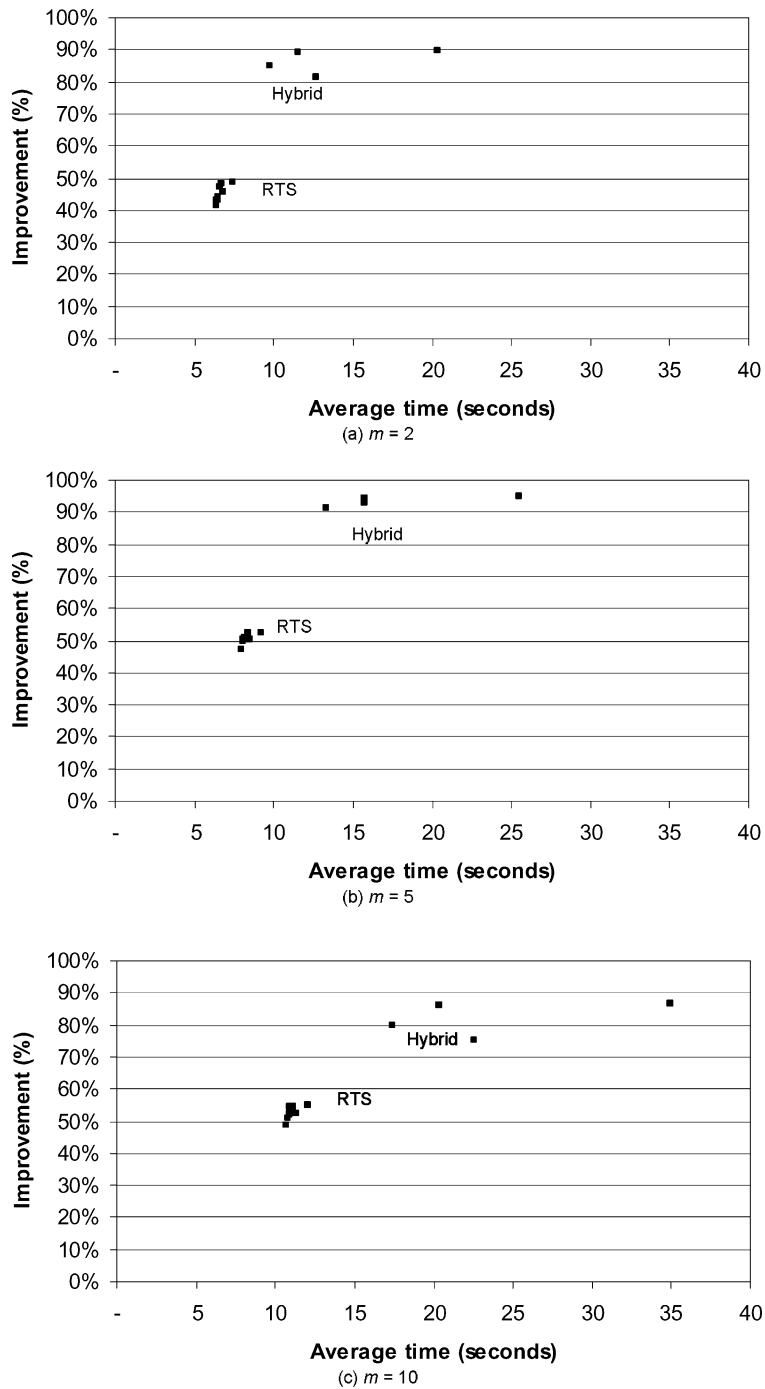


Figure 4. Improvement percentage vs. average time: comparison of the RTS and hybrid methods.

Since we use the problem specific heuristics in the initialization, the solution quality obtained by the meta-heuristics can partially be attributed to the solution quality obtained by the problem specific heuristics. In order to make a more fair comparison, we focus on the improvement percentages. The computational times and average improvement over the solution found by problem specific heuristics, for the RTS and hybrid meta-heuristic methods are depicted in Figure 4.

For the hybrid method, we also focus on the number of iterations performed (see Table III). An iteration consists of selecting the initiating and guiding solutions, linking these solutions and updating the feasible reference set, generating candidate infeasible by the oscillation method and updating the infeasible reference set. Since the termination of the method depends on the update of the reference sets, the number of iteration performed may give us useful insights to compare alternative settings of the method.

### 6.3. DISCUSSION OF THE RESULTS

Our computational study first shows that the BDP takes much longer than the heuristic methods, as expected. The results also show that the alternative settings of the RTS are slightly different from each other. That is, the multi-start settings yield better results than the single-start settings and the settings that use the advanced (and less restrictive) tabu structure yield better results than the ones that

Table III. Number of iterations results for the hybrid method.

| Design parameter | Value | Method | Number of iterations |      |
|------------------|-------|--------|----------------------|------|
|                  |       |        | Avg.                 | Max. |
| Overall          |       | 1      | 2.98                 | 6    |
|                  |       | 2      | 2.97                 | 6    |
|                  |       | 3      | 3.73                 | 9    |
|                  |       | 4      | 3.61                 | 8    |
| $m$              | 2     | 1      | 3.09                 | 6    |
|                  |       | 2      | 3.05                 | 6    |
|                  |       | 3      | 3.86                 | 8    |
|                  |       | 4      | 3.74                 | 8    |
|                  | 5     | 1      | 2.96                 | 6    |
|                  |       | 2      | 2.92                 | 6    |
|                  |       | 3      | 3.69                 | 8    |
|                  |       | 4      | 3.59                 | 8    |
|                  | 10    | 1      | 2.88                 | 5    |
|                  |       | 2      | 2.95                 | 6    |
|                  |       | 3      | 3.64                 | 9    |
|                  |       | 4      | 3.51                 | 8    |

use the simple (and more restrictive) tabu structure. An important observation arises from the comparison of the first settings  $\{1, 2, 5, 6\}$  with the settings  $\{3, 4, 7, 8\}$ . The latter group of settings perform larger number of iterations with higher computational requirements. However, they do not yield significantly better results. Based on this observation and our earlier experiments with the method, we state that the RTS is unlikely to yield a better solution quality, even if the parameters are changed and the method is given longer computation times.

Our hybrid method, using the combinatorial subset generation approach (H1 and H2), is effective in solving the problem under one minute on average and yields near optimal solutions, as the optimal solutions are found in at least 92% of the test instances.

Another important result is that the value assigned to the parameter *FarthestCorner* plays a critical role in the method's performance. Directly including the farthest corner in the infeasible reference set at the first iteration improves the solution quality. The price paid for the difference in solution quality is observed in computational time.

Our preliminary tests and efforts to adjust the method's parameters have demonstrated several relationships between values of the parameters and the performance of the method. As *bFeasible* increases, the method performs more operations on each iteration, therefore the computational time increases, as well. As a result, we expect finding better solutions with higher *bFeasible* values. The same holds for the parameter *bInfeasible*. The number of temporary solutions in solution combination, denoted by the parameter *NTS*, determines the number of operations performed during combining two selected solutions. As the value of *NTS* increases, we expect longer computational times and better solutions. Finally, parameter *InfeasibleMoveDepth* determines the computational burden of finding the infeasible solutions with the oscillation method and combining them with the feasible reference solutions. However, since there will be a larger number of solutions evaluated, it is expected that higher *InfeasibleMoveDepth* yield better solutions.

The best results are obtained by our hybrid method using the probabilistic subset generation approach. First, generating 20% of the possible subsets significantly lowers the time consumption. This result is illustrated by the shorter computation times of the probabilistic settings, despite they keep larger reference sets than the combinatorial settings. Second, the probabilistic approach yields slightly better results than the combinatorial subset generation approach. Clearly, this shows that our probabilistic hybrid method promises the best solution quality in shortest time of all the methods implemented on the batching problem of just-in-time flow-shops.

The results on the relative improvement over the starting solutions (provided by the problem specific heuristic methods) shows that the improvement obtained by the RTS ranges between 41 and 55%, while the proposed hybrid method improves the initial solutions by at least 75%. This result states that the hybrid

method is significantly more effective than the RTS, on the problem studied (see Figure 4).

A closer examination of the average number of iterations performed by alternative settings of the hybrid method yields two important results. First, the probabilistic subset generation approach performs larger number of iterations than the combinatorial approach. Since all possible subsets are generated by the combinatorial approach, a larger number of solutions are evaluated on each iteration of the combinatorial approach. The probabilistic approach, on the other hand, generates a fraction of all possible subsets and the resulting feasible reference set may not include some high quality solutions that could be obtained by the combinatorial approach. Thus, convergence of the reference sets is expected to take a larger number of iterations with the probabilistic approach. Also, including the farthest corner in the infeasible reference set affects the number of iterations for the probabilistic approach. Inclusion of the farthest corner increases the diversification of the search. As a result, we expect more diversified solutions to be stored in the reference sets and the convergence of the method to take a larger number of iterations. Second, the number of iterations performed tends to decrease with the number of machines ( $m$ ). This result can be explained by the increasing number of constraints and difficulty in finding feasible solutions, as  $m$  increases. When it is difficult to find feasible solutions, the probability of not finding any improving solutions, thus not updating the reference set on an iteration is higher. Therefore, the hybrid method tends to converge in fewer iterations for larger  $m$  values.

## 7. Concluding Remarks

This study proposes a hybrid implementation of the SO and PR methods. The hybrid method is a population based meta-heuristic method that uses both feasible and infeasible solutions. The method is tested on a batching problem arising in the context of mixed-model JIT flow-shops. The solutions obtained by the hybrid method are compared to the solutions obtained by two benchmark methods, namely BDP and RTS. The BDP yields the optimal solutions and enables us to measure the quality of the solutions found by the heuristic approaches. The RTS implementation represents a well-known and basic meta-heuristic technique and enables us compare our hybrid method with a simpler method.

The paper includes a computational study that generates and solves a large number of test instances over a broad spectrum. The computational study generates test instances in 18 problem sets, which represent a full factorial testing of three design parameters: setup to processing time ratio, time relaxation percentage and diversification level. These three parameters define the importance of setups in the system, the restrictiveness of capacity (length of the planning horizon) constraints and the similarity between different products in terms of

demands and setup and processing time requirements. The test instances, though artificially generated, represent problem instances that can be encountered in practical manufacturing systems.

Average computational time of the hybrid method is only a proportion of that of the BDP, for all three  $m$  values tested. This result proves that, with yielding only a slight deviation from the optimal solution and having a significantly less computational burden, the hybrid method proposed in this paper is effective in solving the batching problem arising in the JIT flow shops.

The proposed hybrid method benefits from the population-based nature of PR and the neighborhood structure that enables linking pairs of reference solutions to explore a significant portion of the feasible region. Moreover, its ability to work with infeasible solutions contributes to its success in solving the batching problem more effectively than RTS.

In general, the ease of finding a feasible neighbor is very important to the success of meta-heuristics. However, there are many combinatorial optimization problems for which even finding a feasible solution is difficult. Our method presents a new approach to handling infeasible solutions, and, hence, is promising in solving some hard optimization problems where other meta-heuristics may perform poorly. We believe that our hybrid approach can be successfully adapted to many other combinatorial optimization problems with fragmented feasible regions, such as resource constrained project scheduling and vehicle routing problems.

### Acknowledgments

We thank two anonymous referees who offered detailed and constructive criticisms to improve the correctness and presentation of material in the paper.

### References

1. Aiex, R. and Resende, M.: Parallel strategies for GRASP with path-relinking, in T. Ibaraki, K. Nonobe, and M. Yagiura (eds.), *Metaheuristics: Progress as Real Problem Solvers*. Kluwer, 2005.
2. Alfandari, L., Plateau, A. and Tolla, P.: A path-relinking algorithm for the generalized assignment problem, in M. Resende and J. de Souza (eds.), *Metaheuristics: Computer Decision Making*, Kluwer, Boston, 2003.
3. Amaral, A. and Wright, M.: Experiments with strategic oscillation algorithm for the pallet loading problem, *Int. J. Prod. Res.*, **39**(11) (2001), 2341–2351.
4. Blum, C. and Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison, *ACM Comput. Surv.* **35**(3) (2003), 268–308.
5. Dowsland, K.: Nurse scheduling with Tabu search and strategic oscillation, *Eur. J. Oper. Res.* **106** (1998), 393–407.
6. Ghamlouche, I., Crainic, T. and Gendreau, M.: Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design, *Ann. Oper. Res.* **131**(1–4) (2004), 109–134.

7. Glover, F.: Heuristics for integer programming using surrogate constraints, *Decis. Sci.* **8**(1) (1977), 156–166.
8. Glover, F.: Future paths for integer programming and links to artificial intelligence, *Comput. Oper. Res.* **13** (1986), 533–549.
9. Glover, F.: A template for scatter search and path relinking, in J. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers (eds.), *Selected Papers from the Third European Conference on Artificial Evolution, Vol. 1354 of Lecture Notes in Computer Science*, Springer, Berlin Heidelberg New York, 1998.
10. Glover, F.: Scatter Search and Path Relinking, in D. Corne, M. Dorigo and F. Glover (eds.), *New Ideas in Optimization*, McGraw-Hill, Chapt. 19, 1999.
11. Glover, F.: Multi-start and strategic oscillation methods – Principles to exploit adaptive memory: A tutorial on unexplored opportunities, in M. Laguna and J. Gozales Velarde (eds.), *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, Kluwer, 2000.
12. Glover, F. and Kochenberger, G.: *Handbook of Metaheuristics*, Kluwer, 2003.
13. Glover, F. and Laguna, M.: *Tabu Search*, Kluwer, 1997.
14. Glover, F., Laguna, M. and Marti, R.: Fundamentals of scatter search and path relinking, *Control Cybern.* **29**(3) (2000), 653–684.
15. Kelly, J., Golden, B. and Assad, A.: Large-scale controlled rounding using tabu search with strategic oscillation, *Ann. Oper. Res.* **41** (1993), 69–84.
16. Laguna, M., Marti, R. and Campos, V.: Intensification and diversification with elite tabu search solutions for the linear ordering problem, *Comput. Operations Research* **26** (1999), 1217–1230.
17. Marti, R., Laguna, M. and Glover, F.: Principles of Scatter Search, *Eur. J. Oper. Res.* **169**(2) (2006), 359–372.
18. Miltenburg, J.: Level schedules for mixed-model assembly lines in just-in-time production systems, *Manage. Sci.* **35**(2) (1989), 192–207.
19. Monden, Y.: *Toyota Production System: An Integrated Approach to Just-in-Time*, Engineering & Management, 3rd edn., 1998.
20. Nicholas, J.: *Competitive Manufacturing Management: Continuous Improvement, Lean Production and Customer-Focused quality*, McGraw-Hill Irwin, 1998.
21. Ohno, T.: *Toyota Production System: Beyond Large-Scale Production*, Productivity, 1988.
22. Oliveira, C., P. Pardalos, and Resende, M.: GRASP with path-relinking for the QAP, in *Proceedings of the Fifth Metaheuristics International Conference*, Japan, 2003, pp. 57.1–57.6.
23. Osman, I. and Laporte, G.: Metaheuristics: A bibliography, *Ann. Oper. Res.* **63** (1996), 513–623.
24. Reeves, C.: *Modern Heuristic Techniques for Combinatorial Problems*, Wiley, 1993.
25. Scaparra, M. and Church, R.: A GRASP and path relinking heuristic for rural road network development, *Journal of Heuristics* **11** (2005), 89–108.
26. Souza, M., Duhamel, C. and Ribeiro, C.: A GRASP with path-relinking heuristic for the capacitated minimum spanning tree problem, in M. Resende and J. de Souza (eds.), *Metaheuristics: Computer Decision Making*, Kluwer, Boston, 2003.
27. Taillard, E.: Robust taboo search for the quadratic assignment problem, *Parallel Comput.* **17** (1991), 443–455.
28. Talbi, E.: A taxonomy of hybrid metaheuristics, *Journal of Heuristics* **8** (2002), 541–564.
29. Yagiura, M., Ibaraki, T. and Glover, F.: A path relinking approach for the generalized assignment problem, in *Proceedings of the International Symposium on Scheduling*, Japan, pp. 105–108, 2002.

A HYBRID META-HEURISTIC FOR THE BATCHING PROBLEM IN JUST-IN-TIME FLOW SHOPS

30. Yavuz, M., Akcali, E. and Tufekci, S.: A Comparative Analysis of Meta-Heuristic Methods for the Single-Level Batch Production Smoothing Problem, Technical Report 06, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL, 2004.
31. Yavuz, M. and Tufekci, S.: Dynamic programming solution to the batching problem in just-in-time flow-shops, in *Proceedings of the 34th International Conference On Computers And Industrial Engineering*, 2004, pp. 205–210.
32. Yavuz, M. and Tufekci, S.: The single-level batch production smoothing problem: An analysis and a heuristic solution, Technical Report 05, Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida, 2004.
33. Yavuz, M. and Tufekci, S.: Some Lower Bounds on the Mixed-Model Level-Scheduling Problems, in *Proceedings of the 10th International Conference on Industry, Engineering and Management Systems*, 2004, pp. 385–395.