Expert Systems with Applications 40 (2013) 5839–5853

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A multi-objective approach to the application of real-world production scheduling

Peter Korošec [a], Uroš Bole [b], Gregor Papa [a,b,*]

[a] Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia

ARTICLE INFO

Keywords:
Combinatorial optimization
Decision support systems
Flexibility
Job shop scheduling
Multicriteria

ABSTRACT

An initiative was introduced in one of the production facilities of Germany's E.G.O. Group in order to enhance its SAP information system with a custom-made application for production-scheduling optimization. The goal of the optimization is to find a production schedule that satisfies different, contradictory production and business constraints. We show the challenges faced in the application of the multi-objective optimization approach, which is gaining influence in the management of production scheduling. We implement a memetic version of the Indicator-Based Evolutionary Algorithm with customized reproduction operators and local search procedures to find a set of feasible, non-dominated solutions. Such a memetic algorithm was applied to two real order lists from the production company. Additionally, we also lay out an efficient presentation of the multi-objective results for an expert's support in decision making. This provides the management with the possibility to gain additional insights into how the production schedule dynamically reacts to changes in the decision criteria. We show that the multi-objective approach is able to find high-quality solutions, which enables flexibility when it comes to quickly adapting to specific business conditions.

1. Introduction

Germany's E.G.O. Group has decided to supplement its SAP Enterprise Resource Planning (ERP) system with a custom-made application for production-scheduling optimization at one of its production facilities. In the first stage of this initiative, we implemented a solution that runs independently of the ERP system. The production-scheduling problem was optimized with a single-objective approach as reported in (Papa, Vukašinović, & Korošec, 2012). The goal was to find a production schedule that satisfies the production time constraints and minimizes the production costs. This involved many specific constraints that needed to be considered. The solution provided a valuable improvement in the company's operations; however, it resulted in the evolution of a problem, as is common in an emerging knowledge process (Bole, Jaklič, Žabkar, & Papa, 2011; Markus, Majchrzak, & Gasser, 2002). The emergence of additional requirements demanded a second stage to this initiative. Its challenges and solutions are presented in this paper.

The evolution in the complexity of the problem may be reduced to the need for the production-scheduling application to provide greater flexibility, which the production-planning process requires due to changing business conditions. As the company gained a better understanding of the potential benefits of heuristic optimization through the use of our single-objective approach new requirements became evident. Shifting business conditions call for a quick response in the production plan, i.e. a change in the order of priority/importance in the decision criteria. This new insight rendered the single-objective approach inefficient because it is based on a pre-determined and fixed order of priorities in the decision criteria. Hence we applied a multi-objective approach that uses specific local search procedures.

The Indicator-Based Evolutionary Algorithm (IBEA) (Zitzler & Künzli, 2004) was used because it is designed for multi-objective problems. With its quality-indicator-based selection process, the IBEA can substantially outperform the results generated by other multi-objective algorithms in terms of different performance measures (Zitzler & Künzli, 2004). It also builds on our previous work with an evolutionary algorithm when solving the single-objective scheduling problem (Papa et al., 2012). The IBEA, in combination with local search procedures, is called the Memetic Indicator-Based Evolutionary Algorithm (M-IBEA). This represents a synergy of the multi-objective evolutionary approach with separate, individual, learning or local improvement procedures (local searches) for the problem search.

However, as a consequence of applying the multi-objective approach we faced a non-trivial usability problem: how to represent

* Corresponding author. Tel.: +386 1 4773514.
E-mail addresses: peter.korosec@ijs.si (P. Korošec), uros.bole@gmail.com (U. Bole), gregor.papa@ijs.si (G. Papa).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.05.035

a large set of solutions in an actionable manner. In contrast to the single-objective approach, which offers only one solution, the implemented multi-objective approach provides a large set of solutions, all of which are of the same quality with regard to its many objectives. From this set a planner must choose the final solution that adapts best to the specific business conditions. However, this presented an additional challenge. To facilitate seamless decision-making it was necessary to design an easy-to-understand and intuitive visualization. We accomplish this through an innovative modification of the applications graphical user interface.

The rest of the paper is organized as follows: Section 2 covers the related works on multi-objective scheduling and memetic approaches to multi-objective scheduling; in Section 3 we briefly describe the production scheduling problem; in Section 4 we describe the IBEA; in Section 5 we introduce the used local search procedures; in Section 6 we show the integration of the IBEA with the local search procedures; in Section 7 we present the experimental environment and the results of the evaluation with the real-world data; in Section 8 we present the usability study; in Section 9 we show some managerial implications of the presented multi-objective approach; and in Section 10 we draw conclusions and propose possible future work.

2. Related work

Scheduling problems exist in many real-world industrial production settings. Due to the complexity of these scheduling problems, significant work has been devoted to the automation of scheduling processes using a variety of different methods. Furthermore, as a result of the increasing deployment of planning and scheduling systems, we often have to deal with very large search spaces, real-time performance demands, and dynamic environments (Nareyek, 2000). Effective production scheduling can result in a reduction of manpower and production costs by minimizing a machine's idle time and increasing the number of on-time job deliveries (Chan, Au, & Chan, 2006). A review of the research on integrated process planning and scheduling, along with a discussion about the extent of the applicability of various approaches, is presented in Li, Gao, Zhang, and Shao (2010); Shaotan, A, Liang, and Yi (2010).

Multi-objective optimization (Deb, 2001) is very common within the world of engineering problems. These problems have multiple objectives, which instead of only one optimum solution, call for a set of optimum solutions, known as effective solutions. In practice, it was shown that in applications of optimization, it is common to have several conflicting objective functions (Custódio, Madeira, Vaz, & Vicente, 2011). It has been found (Deb, 2001) that using evolutionary algorithms is a highly effective way of finding multiple effective solutions in a single simulation run. The set of all the effective, non-dominated solutions is called the pareto front (Deb, 2001). In technical applications, this front is usually bounded. In (Mueller-Gritschneder, Graeb, & Schlichtmann, 2009) it is shown that the boundary of the pareto front consists of the so-called trade-off limits if certain criteria are fulfilled. There were already some attempts to solve different planning and scheduling, multi-objective problems (Bozorgirad & Logendran, 2012; Li, Burke, Curtois, Petrovic, & Qu, 2012, 2010; Chryssolouris & Subramaniam, 2001). Furthermore, (Mansouri, Gallear, & Askariazad, 2012) reviews the literature available on modeling build-to-order supply chains with the focus on adopting multi-objective techniques as decision-support tools.

Various Memetic Algorithms (MAs) were developed (Caumond, Lacomme, & Tchernev, 2008; Garca-Martnez & Lozano, 2008; Kamrul Hasan, Sarker, Essam, & Cornforth, 2009; Siarry & Michalewicz, 2008) to obtain even better results than the genetic algorithm (GA) for various scheduling applications, and with the use of local search techniques the results were further improved. Such an approach not only improves the quality of the solutions, it also reduces the overall computational time (Kamrul Hasan et al., 2009). Special attention should also be given to the dynamic nature of the production. Local search is well suited for anytime requirements because the optimization goal is improved iteratively, as described in Nareyek (2000). The authors of Ishibuchi, Yoshida, and Murata (2003) show how the performance of evolutionary, multi-objective, optimization algorithms can be improved by hybridization with local search. One positive effect of the hybridization is the improvement in the convergence speed to the pareto front; however, the main negative effect is an increase in the computation time per generation. In Chica, Cordón, Damas, and Bautista (2012) different variants of multi-objective MAs were developed and compared in order to determine the most suitable exploration/exploitation trade-off for the memetic search process, while performing time and space assembly-line balancing. Several important considerations for the design of multi-objective MAs are summarized in Knowles and Corne (2005).

In our previous work (Papa et al., 2012) a guided local search algorithm was tested on real-world test cases of a production-scheduling problem. This problem is a member of the family of job shop scheduling problems, which are known to be NP-hard. Due to the problem's complexity (e.g., many constraints) specialized local searches were used. They were guided with the genetic algorithm (Goldberg, 1989), parameter-less evolutionary search (Papa, 2008), and random selection. It was shown that the use of stochastic approaches greatly improved the quality of the production schedules with respect to the expert's manual solution.

In this paper we show that the need for an increased flexibility in production-scheduling optimization was resolved with a multi-objective approach and intuitive visualization. Every scheduling problem has its own specifics, which need to be dealt with appropriately. On the basis of all the previously acquired problem-specific knowledge, we have decided to use one of the most promising multi-objective approaches, i.e., the IBEA (Zitzler & Künzli, 2004), mixed with advanced local search approaches (Papa et al., 2012). The IBEA, with its proven performance for more than three objectives (Wagner, Beume, & Naujoks, 2007), was selected since we need to handle four contradictory objectives, for which many classic multi-objective approaches are inappropriate (Ishibuchi, Tsukamoto, & Nojima, 2008). While in Papa et al. (2012) we proved the suitability of the evolutionary approach to finding an optimum solution within a broad range of possible solutions, this paper presents some additional local search procedures, as well as introducing the approach with multiple objectives. In addition, we introduce an efficient presentation of the multi-objective approach's results. It enables the decision-maker to swiftly adopt the priority in decision criteria to the specific business conditions and select the most suitable optimal solution.

3. Production scheduling

The scheduling problem was introduced while designing the application for scheduling and monitoring production in the company Eta Cerkno d.o.o. The company produces components for domestic appliances, including hot plates, thermostats and heating elements.

Among the various production stages, the most demanding was the production of cooking hot plates. Here, the fabrication process for components used in different types of plates is similar, but due to clients' demands the models differ in size (height, diameter), connector type, and power characteristics (wattage). For logistic reasons the clients group different models of plates within the same order, implying the same due-dates for different products. Therefore, their production must be scheduled very carefully to fulfill all the demands (quantities and due-dates), to maintain the specified amounts of different models in stock, to optimally occupy their workers, and to make efficient use of all the production lines. Although the assignment of due-dates is usually performed separately, and before the production scheduling, there are strong interactions between the two tasks. Each order placed by the customer somehow defines a batch of jobs, and their completion times should be as close as possible in order to reduce the waiting time and cost (Zhang & Wu, 2012). Furthermore, not all the production lines are equal, since each of them can produce only a few different models. A detailed formulation of the production-scheduling problem is presented in Papa et al. (2012).

4. Indicator-based evolutionary algorithm

Based on our previous work (Papa et al., 2012) and the evolved production-schedule requirements we implemented the Indicator-Based Evolutionary Algorithm (IBEA) (Zitzler & Künzli, 2004) with local search techniques to search for solutions.

The IBEA is a multi-objective version of a GA, where the selection process is based on quality indicators. An indicator function assigns each pareto-set approximation a real value that reflects its quality. The optimization goal becomes the identification of a pareto-set approximation that minimizes an indicator function. The main advantage of the indicator concept is that no additional diversity-preservation mechanisms are required (Basseur & Burke, 2007). The authors of Zitzler and Künzli (2004) demonstrated that an indicator-based search can yield results that are superior to popular algorithms such as the improved Strength Pareto Evolutionary Algorithm (Zitzler, Laumanns, & Thiele, 2001) and NSGA-II (Deb, Agrawal, Pratap, & Meyarivan, 2000).

4.1. Algorithm

A general representation of a basic version of the IBEA is presented in Algorithm 1. It performs binary tournaments for the selection of individuals to undergo recombination. Next, it iteratively removes the worst individual from the population and updates the fitness values of the remaining individuals. We adapted the basic implementation of the IBEA with our implementations of crossover and mutation operators in order to fully adapt to the specific problem of production scheduling. The details of the operators are presented and described in the following sections.

Algorithm 1. IBEA

```
1: SetInitialPopulation ($P$)
2: Evaluate ($P$)
3: **while** not EndingCondition () **do**
4:     $P'$ = MatingSelection ($P$)
5:     Crossover ($P', p_c$)
6:     Mutation ($P', p_m$)
7:     Evaluate ($P'$)
8:     $P$ = CalculateFitness ($P \cup P'$)
9:     $P$ = RemoveWorse ($P$)
10:    VaryControlParameters ($p_c$, $p_m$)
11: **end while**
```

4.1.1. Production-schedule encoding

The production schedule is encoded into a chromosome with tuples of values, where each tuple (gene) consists of the index of the enumerated order and its assigned production line. A general representation of a chromosome, which includes the encoded production schedule of n orders, is presented in Eq. (1).

$$C = g_{11}g_{12}, g_{21}g_{22} \cdots g_{n1}g_{n2}, \tag{1}$$

where n is the number of product orders, g_{k1} is an index on some operation $o_k \in O$ and g_{k2} is the production line used to produce the order o_k, for every $k \in \{1, 2, \ldots, n\}$.

4.1.2. Population initialization

The input orders that need to be processed are firstly sorted according to their due-dates into the initial order list O. Next, different chromosomes are constructed as variations of the initial list. Each variation of the indexes of orders from the initial list is encoded as a chromosome. The initial population P consists of N_P chromosomes. In each chromosome the orders are randomly distributed, and also the assigned production line is chosen randomly from among the possible lines for each order.

Since the numbers that are encoded in the chromosome represent the indexes of orders, their values cannot be duplicated and also no number can be missed. The assigned values for the production line always depend on the possible production lines for that order. These conditions must be considered during the initialization and during all the subsequent phases.

4.1.3. Selection operator

In each iteration mates are selected to undergo the reproduction operators. The selection model is a binary tournament (Michalewicz & Fogel, 2004; Bäck, Fogel, & Michalewicz, 2000), and the selecting criterion is based on the quality indicator (Zitzler & Künzli, 2004). The indicator is used to compare two single solutions. The solution to be selected from the comparison pair is the one with the better indicator value according to the current population.

Similarly, the solutions to be deleted from the merged population of parents and offspring are those that have the worst indicator value according to the current population. Here, the goal is to delete the solution with the smallest degradation of the overall quality of the population.

4.1.4. Reproduction operators

The order-based Crossover (P, p_c) operator is performed with the interchange of positions that store the ordered numbers within the range. An order-based crossover takes the random part of two parents, swaps the genes of the parents in this part and orders the remaining genes in the first parent in accordance with its order in the second parent. We implemented four types of order-based crossover operators: order (OX), cycle (CX), partially-mapped (PMX) and PTL crossover. During the optimization process they are switched every 10 generations. OX, CX and PMX are more precisely explained in (Michalewicz & Fogel, 2004), while PTL is explained in Czogalla and Fink (2008). In OX, PMX, and PTL the two-point crossover scheme is used, where chromosome mates are chosen randomly.

During the Mutation (P, p_m) process each value of the chromosome mutates with a mutation probability p_m. Five different types of mutation are applied:

- changing of the production line – a randomly chosen order is moved from one production line to another one (according to the possible production lines for that order);
- switching of two genes in the chromosome – two randomly chosen orders switch their positions, i.e., one is moved backward (it is produced later), and the other one is moved forward (it is produced earlier);
- shifting of a gene into some new position – a randomly chosen order is moved to some new position. All the orders between the old and new positions of a moved order are shifted. Note

that the shifting of a gene into some new position has an effect on a larger part of the chromosome. If an order is moved forward then all the orders between the new and the old position are moved backward, but if an order is moved backward then all the orders between the old and the new position are moved forward.
- replacing similar products – if any consecutive orders of similar products on the production line have descending due-dates (the earlier produced order has a later production due-date than the comparison one) then they are switched.
- merging of similar products – on the randomly chosen production line part of the orders (randomly chosen) are merged according to the different properties of the products (dimensions and power characteristics).

The first mutation type influences the second part of the gene; the second mutation type influences the whole gene; the remaining three mutation types influence only the first part of the gene.

4.2. Fitness evaluation

After the reproduction operators and local search procedures modify the solutions $p \in P$ the solutions are evaluated, so the selection process in the next iteration can be performed. Each solution p defines its set of objective values. A set of n_{obj} objective values is defined with: the number of delayed orders (n_{orders}); the required number of workers (n_{workers}); the sum of the change-over downtime in minutes (t_{change}); and the sum of delayed days of all the delayed orders (n_{days}). The objective values are calculated by the objective functions f_k, $k \in \{1, \ldots, n_{\text{obj}}\}$.

According to Zitzler, Thiele, Laumanns, Fonseca, and Grunert da Fonseca (2003), a binary quality indicator can be regarded as a continuous extension of the concept of pareto dominance on sets of objective vectors. The indicator value I quantifies the difference in quality between two objective vectors. In our implementation we used the additive $I_{\epsilon+}$-indicator (Zitzler et al., 2003) (see Eq. (2)), which gives the minimum distance by which a pareto-set approximation needs to be, or can be, translated in each dimension of the objective space such that another approximation is weakly dominated.

$$I_{\epsilon+}(p_i, p_j) = min_{\epsilon}\{f_k(p_i) - \epsilon \leqslant f_k(p_j) \text{for} k \in \{1, \ldots, n_{\text{obj}}\}\} \quad (2)$$

The population P represents a sample of the decision space, and the fitness assignment tries to rank the population members according to their usefulness regarding the optimization goal. Among the different ways of exploiting the information given by P and $I_{\epsilon+}$, we sum up the indicator values for each population member with respect to the rest of the population. This fitness value F (Eq. (3)) that needs to be maximized is a measure of the "loss in quality" if a solution p_i is removed from the population.

$$F(p_i) = \sum_{p_j \in P \setminus \{p_i\}} - e^{-I_{\epsilon+}(p_j, p_i)/\kappa} \quad (3)$$

The parameter κ is a scaling factor and in our case it was set to 0.05, as suggested in Zitzler and Künzli (2004).

4.3. Varying control parameters

To limit a possible disruptive effect of mutation during the later stages of the optimization and to speed up the convergence to the optimum solution, the crossover and mutation probability are changed within the VaryControlParameters () function. After every predefined number of generations the crossover and mutation probability are decreased by some factor. The smaller number of mutated positions can also be interpreted as a kind of local search with minor movements around the current solution (i.e., exploitation).

4.4. Ending condition

In general the algorithm is run until the user stops the optimization process. So the EndingCondition () function checks to see whether a user pressed the stop button. To mimic overnight running we decided to limit the number of evaluations to 300 million instead of checking for the user to press the button.

5. Local search

Local search (LS) procedures are optimization methods that improve the solution by maintaining a currently best one, and exploring the search space step by step within its neighborhood. Usually, the current solution is replaced by a better one in the neighborhood, which is used as a new current solution in the next iteration. This process is repeated until a stopping condition is met, i.e., if there is no better solution within the neighborhood. The interesting point of LS is the fact that it may effectively and quickly explore the basin of attraction of the optimum solutions, finding an optimum with a high degree of accuracy and within a small number of iterations. In fact, these methods are a key component of the metaheuristics that are the state-of-the-art for many optimization problems (Garca-Martnez & Lozano, 2008).

Compared to Papa et al. (2012), where only four LS procedures were used, the approach in this paper uses four additional LS procedures to implement the additional requirements and constraints, and also one of the previous procedures was upgraded. In the following paragraphs we give a description of all the implemented LS procedures: the previous three, the modified one and the new four.

Algorithm 2. LocalSearch (P)

```
1: for each solution $p \in P$ do
2:     StockReplacing ($p$)
3:     DueDateSorting ($p$)
4:     JoiningForMinimumQuantity ($p$)
5:     LineConflictResolve ($p$)
6:     ProductionLineGroupChanging ($p$)
7:     TypeChanging ($p$)
8:     MoveDelayedForward ($p$)
9:     SimilarProductMerging ($p$)
10: end for
```

LS is used to improve new solutions on the pareto front and is implemented with eight different procedures, which run sequentially (see Algorithm 2). The reason for such an order of procedures is that the influences of the changes made by, e.g., "stock replacing", are much more degrading to the current solution than the changes made by "similar product merging". This means that at the beginning we allow for major changes, which are then more refined by smaller changes. This enables us to find new solutions, which can be located in some other parts of the solution space and still be better than the current solution. The influence of such a LS, due to the very complex nature of our problem, should be a much improved convergence and quality of the obtained solutions compared to the basic optimization technique alone.

The examples shown in the following subsections are invented and their only purpose is to give the reader a visualization of the idea behind the presented type of LS. A, B, C and D represent the production of orders of different products and their indexes represent their due-date order. Empty spaces show the unused production capacity due to the change-over downtime between the different types of products. In reality, the change-over downtime can vary from a few minutes up to a few hours.

In the implemented local search procedures each new solution is tested to see whether it qualifies to replace the previous solution (its predecessor). The testing procedure is based on the idea that the new solution should not be dominated in any objective by the old solution. So, if the new solution is not dominated by the old one, then the new solution is accepted and the local search proceeds. On the other hand, the new solution is abandoned and the search continues with the old one.

5.1. Stock replacing

For each order filled from stock there is a check to see whether the order for the same product is scheduled for production (see Algorithm 3). If the order to be produced is delayed, then this order is shifted to be in front of the order from stock that was checked. In this case some orders of the same product may be moved out of stock and placed into production. If the number of delayed orders is increased, then the previously shifted order is moved back to its previous position.

ALgorithm 3. StockReplacing (p)

```
1: for each production line do
2:   for all orders i in the stock do
3:     if due-date of order after current date then
4:       for all orders j in the production do
5:         if due-date of order j after due-date of order i then
6:           shift order j before order i
7:           Evaluate (p)
8:           if solution is worse then
9:             move order j back to its position
10:          end if
11:        end if
12:      end for
13:    end if
14:  end for
15: end for
```

The stock-replacing example (see Fig. 1) is from a one-production-line process. In the upper row an initial schedule is represented with two delayed orders, B_3 and B_4. In the lower row the products of the orders B_3 and B_4 are filled from stock and the number of delayed orders is reduced (now only B_1 is delayed, which has to be produced now, since the stock quantity is used for B_3 and B_4).

5.2. Due-date sorting

For each production line, all the orders with delayed due-dates are checked to see whether they can be moved in front of some other order (see Algorithm 4). The later order is moved before each of the precedent order, so that their due-dates are sorted, while ensuring that the total number of delayed orders is not increased (see Fig. 2).

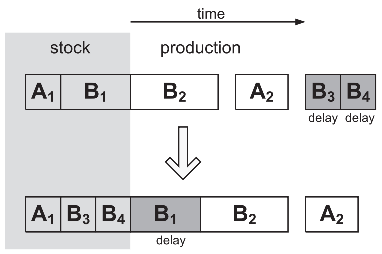


Fig. 1. Stock replacing: initial schedule (above), reduced number of delayed orders (below).

Algorithm 4. DueDateSorting (p)

```
1: for each production line do
2:   for all orders i do
3:     if order i is delayed then
4:       for all orders j before order i do
5:         if due-date of order i after (start of order j + length
   of order i) then
6:           shift order i before order j
7:           Evaluate (p)
8:           if solution is worse then
9:             move order i back to its position
10:          end if
11:        end if
12:      end for
13:    end if
14:  end for
15: end for
```

In the upper row of the due-date sorting (Fig. 2) the orders are not sorted according to their due-dates (B_2 is before B_1, and B_4 is before B_3). In the lower row the orders B_1 and B_2 are switched, as are the orders B_3 and B_4. Now, consecutive orders of the same type are sorted according to their due-dates.

5.3. Joining for minimum quantity

For each production line there is a check to see whether several orders of the same type can be joined to form a virtually larger order of a preset minimum quantity (see Fig. 3 and Algorithm 5). The idea of joining is to ensure a large enough production without an interruption for tool change-over.

Algorithm 5. JoiningForMinimumQuantity (p)

```
1: for each production line do
2:   for all orders i do
3:     while type of order is the same do
4:       summarize the quantities of consecutive orders of
   the same type
5:     end while
6:     if sum < minimumQuantity then
7:       for all orders j after i do
8:         if type of order is the same as summarized orders
9:           shift order j onto position i + 1 then
10:            add quantity of order j to sum
11:        end if
12:      end for
13:      if sum < minimumQuantity then
14:        move all orders of that type to some other
   production line
15:      end if
16:    end if
17:  end for
18: end for
```

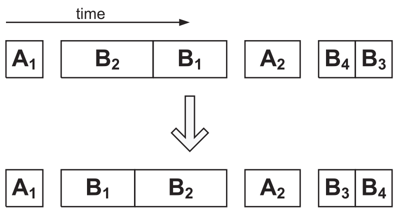


Fig. 2. Due-date sorting: initial schedule (above), sorted due-dates of similar orders (below).

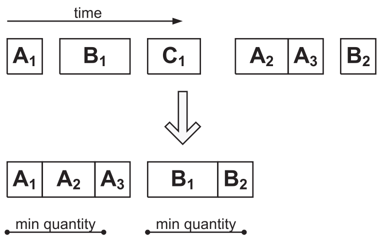


Fig. 3. Joining for minimum quantity: initial schedule (above), merged identical product orders to form minimum production quantities (below).

In the upper row of the minimum-quantity, joining-procedure example (Fig. 3) an initial schedule is represented by three different types of products (orders A, B and C) in mixed order. In the lower row the products of the orders A and B are merged together to form a minimum quantity for each product type, while order C is moved to some other production line, since its quantity was too small to reach the minimum-quantity limit.

5.4. Resolving line conflicts

For each production line there is a check to see whether some orders are marked as conflicted, i.e., if the same type of product is produced simultaneously on some other production line (see Algorithm 6). Conflicts should be resolved since there might be problems with several specific product materials' availability on the production line if more than one production line requires the same specific materials/tools at the same time. All consecutive orders that are marked as conflicted are moved to some other randomly chosen position on the same production line.

Algorithm 6. ResolvingLineConflicts (p)

```
1: for each production line do
2:   for all orders i do
3:     if order i marked as conflicted with some order on another line then
4:       find all consecutive orders of the same type
5:       move orders onto some other random position on this line
6:     end if
7:   end for
8: end for
```

In the upper row of the line-conflict, resolving-procedure example (Fig. 4) an initial schedule is represented by four different types of products (orders A, B, C and D) in mixed order on two production lines. Two orders (A_2 and B_3) on line 2 are indicated as being in conflict, since their production is scheduled concurrently with the same type of product on line 1. In the lower row A_2 and B_3 are moved to some other positions on production line 2 to avoid the conflict.

5.5. Production-line group changing

For each production line, all the orders are checked to see whether they can be placed on any other production line (see Algorithm 7). If they can be placed on some other production line, then there is another check to see if they can be merged with some other order on that new production line (see Fig. 5). Simultaneously, there is a check to see if the consecutive orders can be moved, i.e., the group of consecutive (similar) orders is moved to some other production line. However, the movement to another

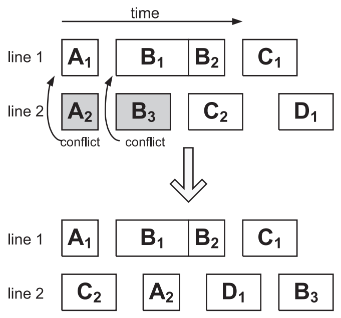


Fig. 4. Resolving line conflicts: initial schedule with marked conflicted orders (above), orders moved within the same production line (below).

production line should not increase the change-over downtime on the new production line.

Algorithm 7. ProductionLineGroupChanging (p)

```
1: for all orders i do
2:   if order i type can be placed on more production lines
3:     for all feasible lines l do
4:       for all orders j on line l do
5:         if type of i equals to type of j then
6:           move order i after order j on line l
7:           Evaluate (p)
8:           if solution is worse then
9:             move order i back to its position
10:          end if
11:        end if
12:      end for
13:    end for
14:  end if
15: end for
```

In the upper row of the production-line changing (Fig. 5) the initial schedule is represented by two production lines. Below, order B_3 is successfully moved from line 2 to line 1 and merged with two other, similar orders with product B. Similarly, order C_1 is moved from line 1 to line 2 and is merged with C_2.

5.6. Type changing

For each production line, all the orders are checked to see whether their operation type can be modified (see Algorithm 8). Namely, for each product type different operation types exist, i.e., the product can be produced with different numbers of workers and, consequently, with different production times for that product

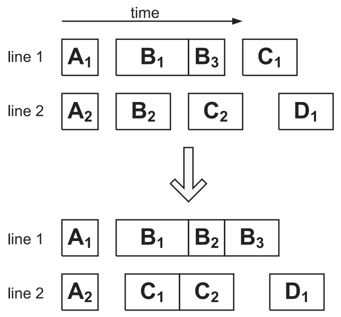


Fig. 5. Production line changing: initial schedule (above), moved product merged with similar products on another production line (below).

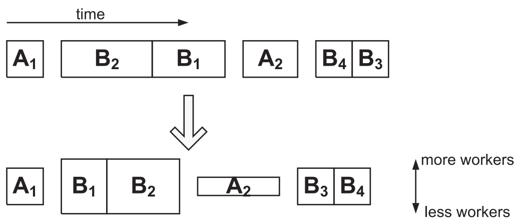


Fig. 6. Type changing: initial schedule (above), changed type of operation for some orders (below).

(see Fig. 6). The modification of the operation type should not adversely affect the schedule.

Algorithm 8. TypeChanging (p)

```
1: for all orders $i$ do
2:   for all operation types $t$ do
3:     change operation type of order $i$ to $t$
4:     Evaluate ($p$)
5:     if solution is worse then
6:       revert to initial type
7:     end if
8:   end for
9: end for
```

In the upper row of the type-changing procedure example (Fig. 6) the initial schedule is represented by two different types of orders (A and B) in mixed order. In the lower row the type of orders B_1 and B_2 is changed to be used with more workers (and consequently faster), while the type of order A_2 is changed to be produced with fewer workers.

5.7. Move delayed orders

For each order on the production line there is a check to see whether the order is delayed (see Algorithm 9). If the order is delayed, then some other earlier order of a different type is checked to see whether the first order can be placed before it.

Algorithm 9. MoveDelayedOrders (p)

```
1: for each production line do
2:   for all orders $i$ do
3:     if order $i$ delayed then
4:       for earlier orders $j$ on the same line do
5:         if orders $j$ and $j-1$ are not of the same type then
6:           move order $i$ to position $j$
7:         end if
8:         Evaluate ($p$)
9:         if solution is worse then
10:          move order $j$ back to its position
11:        end if
12:      end for
13:    end if
14:  end for
15: end for
```

In the upper row of the move-delayed procedure example (Fig. 7) an initial schedule is represented by two different types of products in the orders (A and B) in mixed order. In the lower row the order B_3 is moved forward, just after the order B_2.

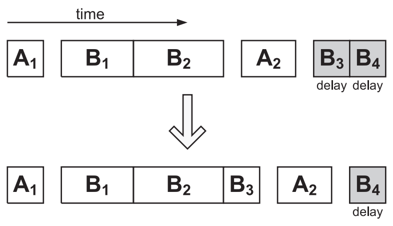


Fig. 7. Moving delayed orders: initial schedule with marked delayed orders (above), rearranged orders with fewer delayed orders (below).

5.8. Similar product merging

For each production line there is a check to see whether several orders can be merged together (see Fig. 8 and Algorithm 10). This merging is performed in four steps, according to the different properties of the products. First, the orders for the products with the same height are merged, then those with the same size are merged, after that it is the orders with the same connectors, and finally those with the same power characteristics. The idea of merging is to decrease the production time on each line, as a result of a decrease in the change-over downtime on the production line.

Algorithm 10 SimilarProductMerging (p)

```
1: for each production line do
2:   for four properties $c$ of products do
3:     for all orders $i$ do
4:       for randomly determined number of orders $j$ after order $i$ do
5:         if property $c$ of $j$ equals to property $p$ of $i$ then
6:           move order $j$ after order $i$
7:         end if
8:       end for
9:       Evaluate ($p$)
10:      if solution is worse then
11:        revert last sequence of moving
12:      end if
13:    end for
14:  end for
15: end for
```

In the upper row of the product-merging procedure example (Fig. 8) the initial schedule is represented by three different types of products (orders A,B and C) in mixed order. In the lower row the products of the orders A and B are merged to reduce the total length of the change-over downtime (arrows below the spaces between orders).

6. Memetic IBEA

There are several approaches to implementing the LS procedures. One very simple metaheuristic is the iterated greedy algorithm, which iteratively applies constructive heuristics to a solution and uses an acceptance criterion to decide whether the newly constructed solution should replace the current one (Ruiz & Stützle, 2006). In our case we merged the basic version of the IBEA to guide the LS procedures. The basic IBEA is implemented with the use of appropriate Java classes of the jMetal framework (Durillo & Nebro, 2011). Furthermore, since we are dealing with a combinatorial problem, we implemented our problem-specific versions of the crossover and mutation operators. Next, we added the local search procedures to enhance the efficiency of

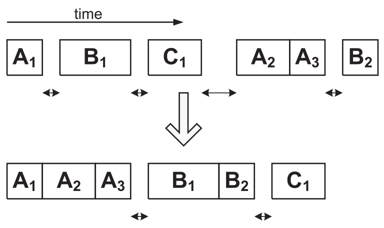


Fig. 8. Similar product merging: initial schedule (above), efficient merging of similar products with minimum change-over downtime (below).

the algorithm. The pseudo code of this Memetic IBEA (M-IBEA) is presented in Algorithm 11.

Algorithm 11. Memetic IBEA

```
1: SetInitialPopulation ($P$)
2: Evaluate ($P$)
3: while not EndingCondition () do
4:    $P'$ = MatingSelection ($P$)
5:    Crossover ($P'$,$p_c$)
6:    Mutation ($P'$,$p_m$)
7:    Evaluate ($P'$)
8:    LocalSearch ($P'$)
9:    $P$ = CalculateFitness ($P \cup P'$)
10:     $P$ = RemoveWorse ($P$)
11:     VaryControlParameters ($p_c$, $p_m$)
12: end while
```

Compared to the basic version of the algorithm (presented in Algorithm 11) the difference is in the procedure LocalSearch (P'). Here, the local search procedures are used, as described in Section 5.

7. Case study

7.1. Experimental environment

The computer platform used to perform the experiments is based on an AMD Opteron™ 2.6-GHz processor, 16 GB of RAM, and the Microsoft® Windows® operating system. The algorithm is implemented in Sun Java 1.7.

7.2. Test cases

The algorithm was tested on two real order lists from the production company. Task 1 consisted of $n = 470$ orders for 189 different products and Task 2 consisted of $n = 393$ orders for 175 different products. The number of orders n represents the problem dimension, with $m = 5$ representing the number of available production lines. Each product can only be put on certain production lines (depending on the product characteristics).

Since in its current state this is more of a prototype, we decided to mimic real-world usage of the application. We took a task, ran it over one night and looked at the results. In this time the algorithm made about 300 million evaluations, so this was set as our stopping criterion for future tests. For comparison reasons we also ran the same task using our previous algorithm (Papa et al., 2012) with one change, which was made to the evaluation function. Instead of a weighted linear equation we decided to use a more comparable approach to the pareto, multi-objective approach in the form of a lexicographic evaluation (Rentmeesters, Tsai, & Lin, 1996). We used the evaluation function, where the number of delayed orders (n_{orders}) was set as the most important objective, followed by the required number of workers (n_{workers}), the sum of delayed days for all the delayed orders (n_{days}), and the sum of the change-over downtime in minutes (t_{change}). This order was set according to the most common objective hierarchy.

7.3. Parameter settings

To acquire the M-IBEA's control-parameter values we executed a control tuning using Sobol sequence sampling (Sobol' & Levitan, 1999). Using a Sobol sequence generator we created 1000 l-tuple samples ($l = 5$, where $p_{m_{\text{change}}}$, $p_{m_{\text{switch}}}$ and $p_{m_{\text{shift}}}$ were set to use the same value), which determined the possible control parameter values. The population size ranged from 10 to 1000, while the other control parameters ranged in the $[0, 1]$ interval. All 1000 control parameter l-tuples were then tested on several test cases. Among them the best representative (values were rounded to two decimal places) l-tuple was chosen as our selected control parameter setting. Using this technique the following control parameters were set:

- the population size $N_P = 500$;
- the crossover probability $p_c = 0.5$;
- the mutation probabilities $p_{m_{\text{change}}} = 0.01$, $p_{m_{\text{switch}}} = 0.01$, $p_{m_{\text{shift}}} = 0.01$, $p_{m_{\text{randomize}}} = 0.05$ and $p_{m_{\text{merge}}} = 0.5$.

To make sure that our proposed M-IBEA was working well, we ran a brute-force (BF) approach where all the possible solutions were evaluated for $n < 10$ orders and the optimum pareto front was constructed for each of them. Table 1 shows a comparison of the number of problem evaluations, the execution time, and the matching of the pareto front obtained for $n = 7,8,9$. We did not include smaller n values, since in all cases a sub-one-second time was needed with perfect pareto matching. From the obtained results it is clear that with more than nine orders, the complexity increases well beyond an acceptable time (approximately two months) to calculate all the solutions. Also, in all cases we were able to acquire the same pareto front using the BF and M-IBEA approaches. When considering times, one must take into consideration that the BF was ran multithreaded with 12 threads fully utilized, while the M-IBEA approach was single threaded. The perfect pareto-front matching is unsurprising, since the IBEA already proved to be one of the best algorithms for solving multi-objective problems with more than three objectives (Ishibuchi et al., 2008), which was also the main reason that we selected the IBEA as our base.

Because of this initial positive feedback from the M-IBEA and the complexity of the implementation (local searches and evaluation procedure), we decided not to compare our approach to other MO methods, since we would not gain any real advantage.

7.4. Influence of the control parameters' variability

To improve the convergence and the solution quality we introduced variability of the control parameters. This was done by lowering the control parameters' current values by 50% every 500 generations. To measure the effect of our variable control parame-

Table 1
Comparison of BF (12 threads) and M-IBEA approach (1 thread).

n	Evaluations		Time		Pareto matching
	BF	M-IBEA	BF	M-IBEA	
7	3.94×10^8	3.5×10^4	22 s	17 s	4/4
8	1.58×10^{10}	5×10^5	15 min	33 s	5/5
9	7.09×10^{11}	5×10^6	11 h	5 min	15/15

ters on the algorithm's performance we conducted an experiment, where we ran both versions (with fixed and variable control parameters) on the same task, with the same random generator seed. With this we ensured the same environment and a fair comparison between the versions. The results of this comparison are visible in Figs. 9–12, where the convergence of the minimum and maximum values of the pareto fronts are shown for all the objectives. As can be seen, in all the objectives the variable version returns higher-quality results with a faster convergence for minimum values (which are represented by lower two lines and their values can be read from the left y-axis), while for maximum values (which are represented by the upper two lines and their values can be read from the right y-axis) the results are mixed. The reason for separated y-axes is the easier readability of the figure. Since we are looking for a minimum, the maximum values are of less importance. The results show that the use of variable control parameters has a positive effect on the algorithm's performance.

7.5. Results

Examples of the results for both tasks are presented in Figs. 13–20, and Tables 2 and 3. In all the figures the X axis represents the n_{orders} objective, the Y axis represents the n_{workers} objective, the Z axis represents the t_{change} objective and the color scheme represents the n_{days} objective. The solution obtained with the single-objective approach is marked with a pointing arrow.

Figs. 13–16 show the pareto front for Task 1 in 4D space from different perspectives. Here we can see that the pareto front is divided into two parts, while the single-objective solution does not belong to either of them. This is to some degree a surprise, but on the other hand it shows that the evaluation function of the single-objective approach forces the algorithm in a certain direction, which in the end does not mean that this is the best according to all the objectives. While for the n_{orders} objective it returns the same quality of solution as the best solutions according to the same objective from the pareto front and among those best solutions an average solution regarding the n_{workers} and n_{days} objectives, it is really poor regarding the t_{change} objective. Though we used the same number of evaluations, this single-objective solution does not stand out with respect to any objective – quite the opposite is the case. This can also be observed from Table 2, where the single-objective solution returns an average quality solution on all the objectives except n_{orders}.

Figs. 17–20 also show the pareto front in 4D space from different perspectives, but for Task 2. In contrast to Task 1, here we can see that the pareto front is more uniformly distributed across the

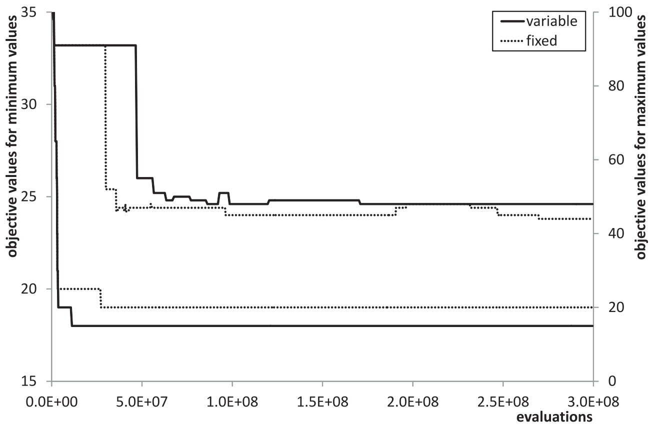


Fig. 9. Comparison of the minimum and maximum values of n_{orders} in the pareto fronts for the variable and fixed version of the control parameters.

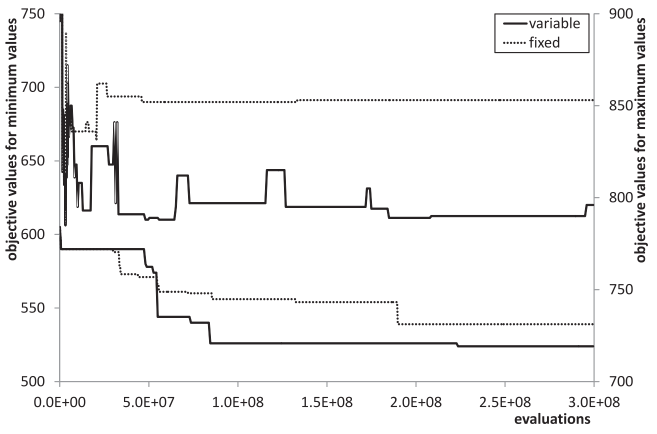


Fig. 10. Comparison of the minimum and maximum values of n_{workers} in the pareto fronts for the variable and fixed version of the control parameters.

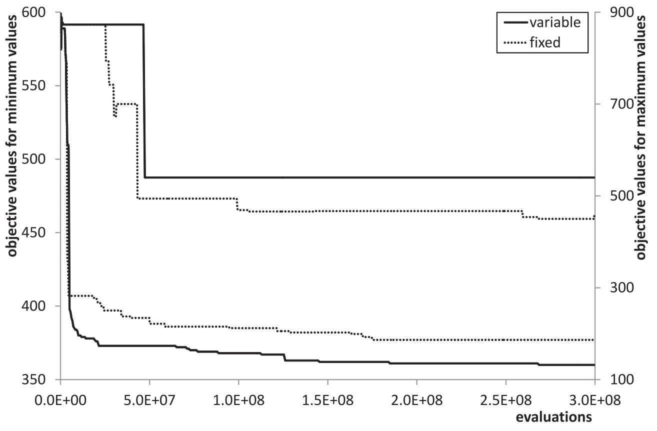


Fig. 11. Comparison of the minimum and maximum values of t_{change} in the pareto fronts for the variable and fixed version of the control parameters.

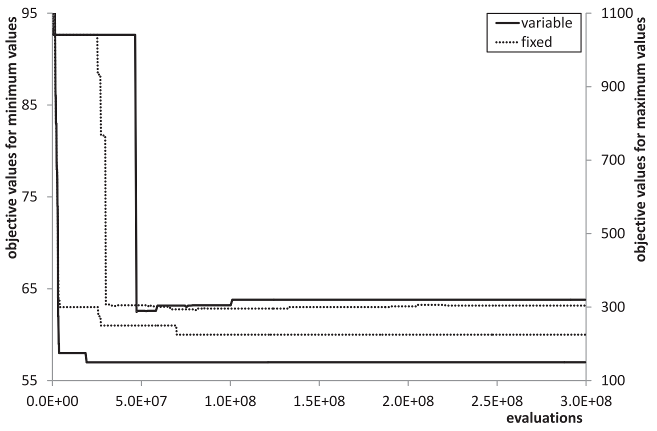


Fig. 12. Comparison of the minimum and maximum values of n_{days} in the pareto fronts for the variable and fixed version of the control parameters.

objective space and also the single-objective solution could be considered as part of that pareto front. This kind of distribution of solutions across the objective space is something one could anticipate. Here we see that the single-objective approach found the best solution regarding the n_{orders} objective, the average regarding the n_{workers} and n_{days} objectives and is actually the worst regarding the t_{change} objective according to the whole pareto front. For this task, the single-objective solution did stand out with regards to

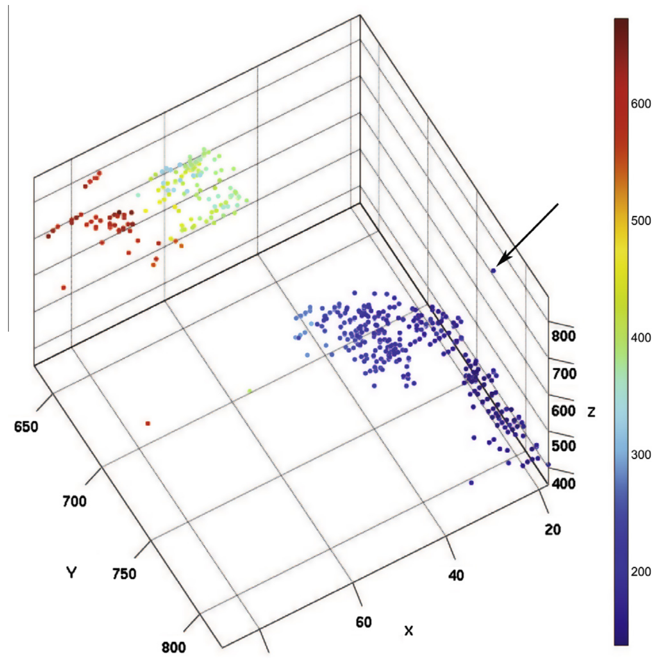


Fig. 13. Pareto front for Task 1 in the 4D space.

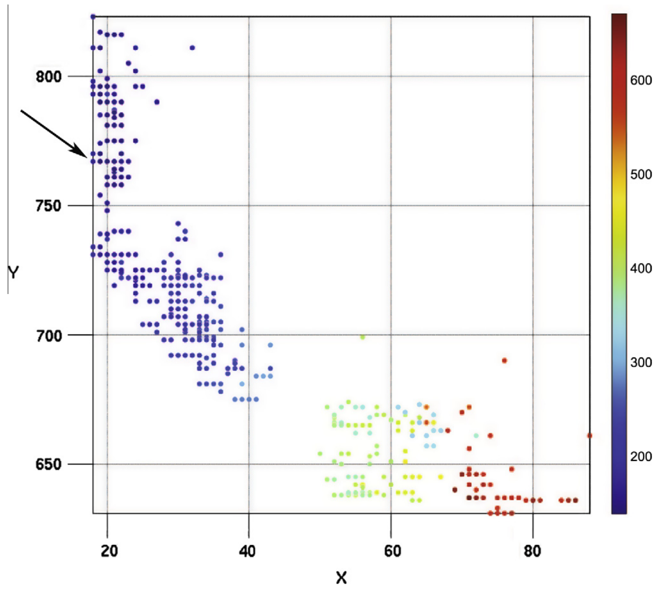


Fig. 14. Pareto front for Task 1 in the n_{orders}, n_{workers} plane projection.

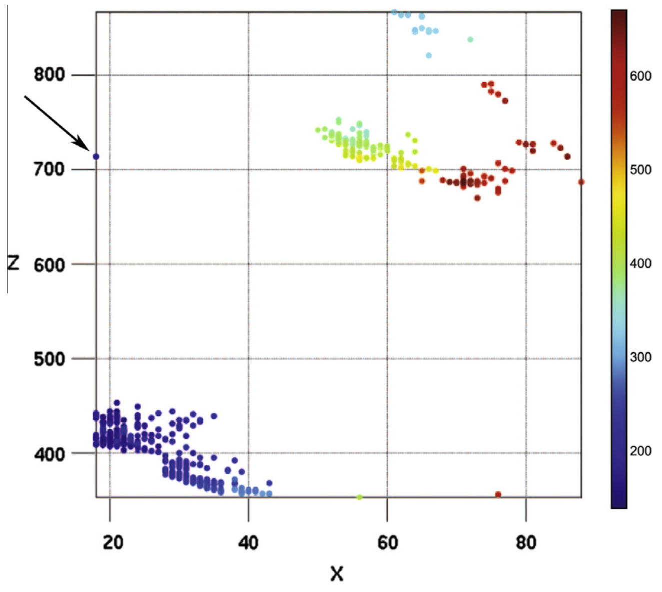


Fig. 15. Pareto front for Task 1 in the n_{orders}, t_{change} plane projection.

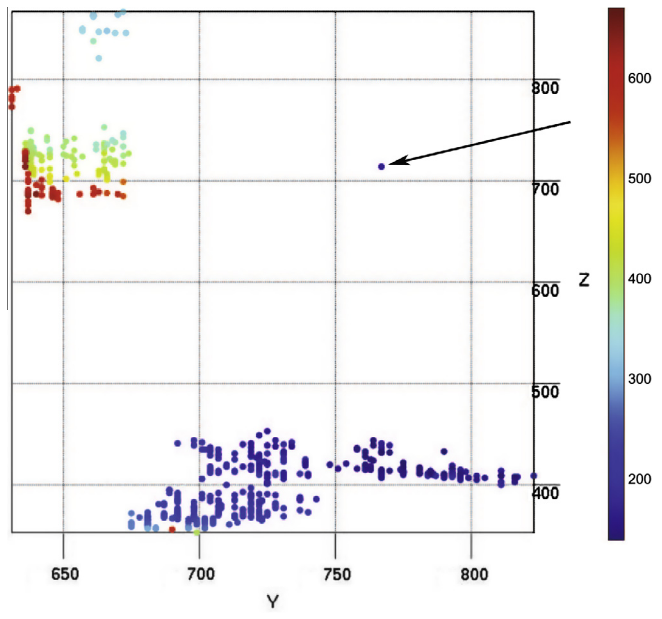


Fig. 16. Pareto front for Task 1 in the n_{workers}, t_{change} plane projection.

one objective, but decreased in terms of the quality of the remaining objectives, most probably due to lexicographic progress in the evaluation function. This can also be observed from Table 3.

From the results obtained for both tasks, we can assume that the quality of the single-objective approach is greatly influenced by the objective space. When we are dealing with a pareto front that is divided into multiple parts, the single-objective approach can be misled into objective-space areas that are somewhere in between those divided parts and not belonging to any of the expected group of solutions. On the other hand, if we have more uniformly distributed solutions of the pareto front, the single-objective lexicographic approach can return a solution that is the best regarding the most important objective and is "expectedly" worse in others.

From the results we can conclude that using the pareto-front approach gives us an expected greater versatility in choosing a good solution, while at the same time we are not sacrificing one, likely the most important, objective. The only important drawback is that multi-objective approaches need many more evaluations than single-objective approaches. So, if we do not have time to carry out enough evaluations then the single-objective approach is the only way. At this point we could ask how many is enough evaluations? Unfortunately, there is no simple answer and any discussion is beyond the scope of this paper, since it greatly depends on the complexity of the solution space and the number of objectives.

8. Usability of multi-objective solutions

The single-objective approach only offers one solution. Conversely, the implemented multi-objective approach intrinsically provides a set of feasible solutions, offering the possibility to select

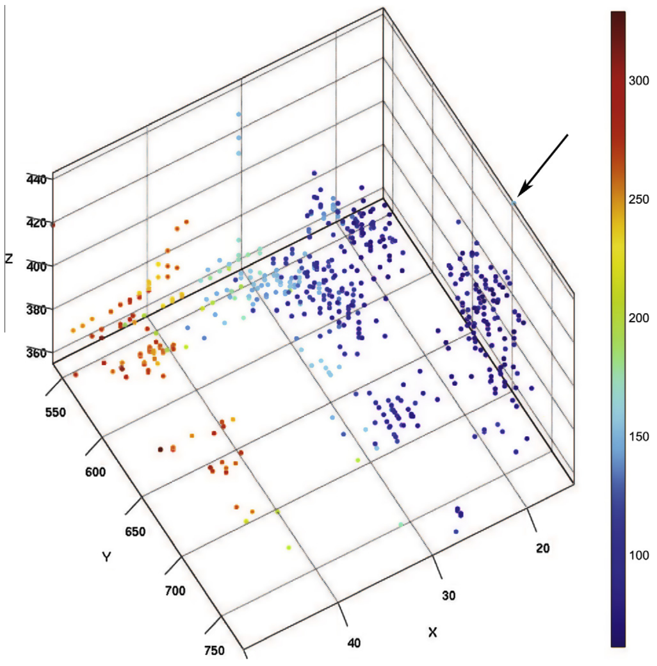


Fig. 17. Pareto front for Task 2 in the 4D space.

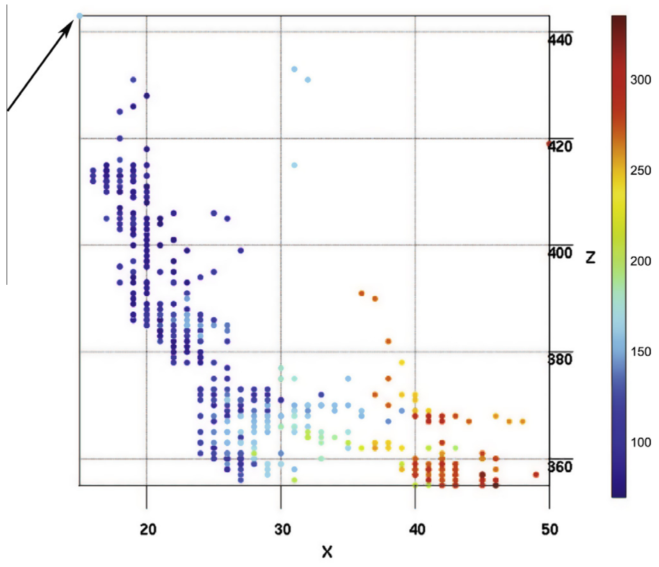


Fig. 19. Pareto front for Task 2 in the n_{workers}, t_{change} plane projection.

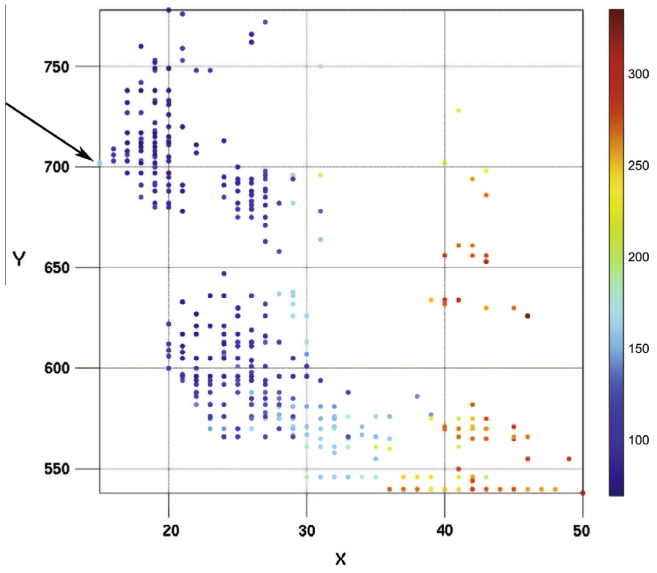


Fig. 18. Pareto front for Task 2 in the n_{orders}, n_{workers} plane projection.

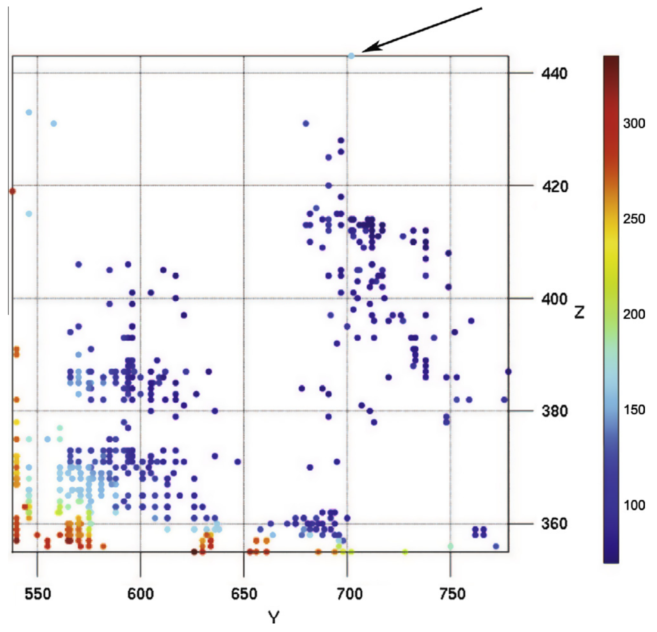


Fig. 20. Pareto front for Task 2 in the n_{workers}, t_{change} plane projection.

the final schedule based on the specific business conditions. Since none of the given solutions dominates over the other solutions, all of them are acceptable. Based on the current (daily, weekly, etc.) business conditions, and according to the proposed set of solutions, a planner can give more weight to some of the decision criteria. This can be easily achieved with the use of an intuitive representation of the resulting solutions inside the GUI of the Planer application, which is presented in Fig. 21.

After the M-IBEA algorithm found the set of non-dominated solutions, they are presented in the Planer application. In the upper-right section there is a list of all the non-dominated solutions. Depending on the inputs - the number of orders, the due-dates of the orders, free production capacity, number of available workers, and the scheduled number of shifts on each production line – there might be up to several hundred possible solutions.

Table 2
Results of optimization for Task 1.

Statistics	n_{orders}	n_{workers}	t_{change}	n_{days}
Pareto min	18	631	353	127
Pareto max	88	823	867	681
Single-objective	18	767	714	156

Table 3
Results of optimization for Task 2.

Statistics	n_{orders}	n_{workers}	t_{change}	n_{days}
Pareto min	16	538	355	59
Pareto max	50	778	433	330
Single-objective	15	702	443	155

However, some of the criteria can be set tighter according to the resulting range of each criterion, and according to the current business conditions. In the specific example shown in Fig. 21 the initial set consisted of 518 solutions. As presented in the upper-left part of Fig. 21, the first objective that represents n_{orders} (O1) was in the range from 16 to 50, the second objective that represents n_{workers} (O2) was in the range from 538 to 778, the third objective that represents t_{change} (O3) was in the range from 355 to 433, while the fourth objective that represents n_{days} (O4) was in the range from 59 to 330. Then, the decision-maker tightened the first objective into the range from 16 to 17 and manually moved the right limiter of the first slider from 50 to 17. This means that only solutions where the first objective was in the mentioned range are interesting for further investigation. The ranges of the other objectives are subsequently automatically adjusted, e.g., O2 from 697 to 738, O3 from 405 to 415, and O4 from 60 to 111. Any of the sliders can be further moved to set some other ranges of the objective values. Simultaneously, the list of possible solutions is updated to reflect the current setting of the objectives' ranges. As presented in the upper-right section of Fig. 21, the list is narrowed down to 14 candidate solutions. Here, the decision-maker selects one solution (labeled with the value of four objectives), which is then drawn on the lower section of Fig. 21. The visual representation consists of all the relevant data, i.e., the production lines' load, the order types' distribution, and change-over downtime lengths, which are necessary to make the final decision. If the visual representation of the solution is also accepted, it becomes the production schedule.

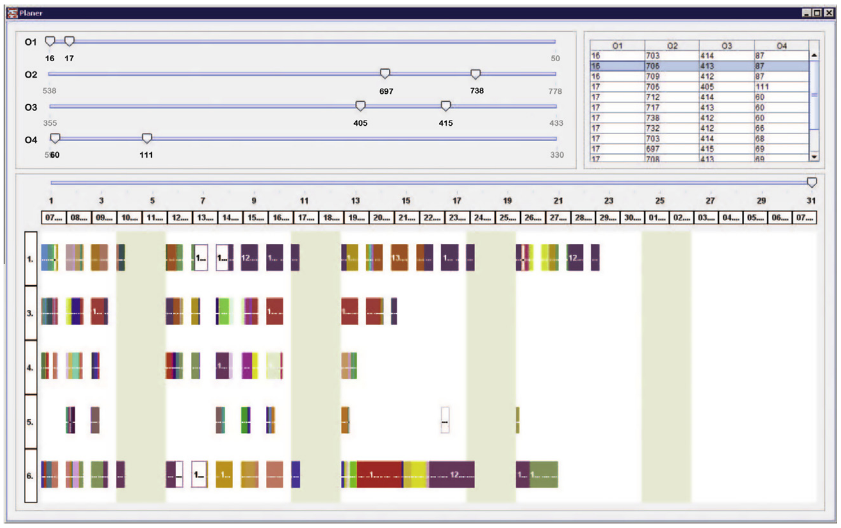
To give better understanding of what is happening in regards to selection of pareto solutions during movement of the sliders, we present Fig. 22. In Fig. 22a we present a 3D presentation of pareto solutions where dimension X presents O1, dimension Y presents O2, dimension Z presents O3, and the colors of the solutions presents O4. Here we can see that all the equivalent solutions are spread across the solution space. In such a situation an end-user would have a lot of troubles selecting the most suitable in regards to current conditions. But if we would use our slider to allow only the two lowest values of O1, then two lowest values of O2, and so on till O4, we would get only three solutions as presented in Fig. 22b. With such selection of slider movements we achieve the similar effect, as if we would use a single-objective approach where the highest priority would have O1, followed by O2, etc. As shown, using the multiobjective approach, we have not lost any advantages of using single-objective approach. On contrary, we have gained a freedom to give priority to only objective O1 (as seen in Fig. 22c) or 02 (as seen in Fig. 22d) or 03 (as seen in Fig. 22e) or 04 (as seen in Fig. 22f), where in each case, the two lowest values were selected by the slider. By determining (using sliders), which objective is the most important to us in current situation and to what extent, we automatically determine which part of pareto front is important to us and at the same time disregard all the solutions from pareto front, which do not fulfill selected conditions. This way we are able to freely move the useful part of pareto front by moving sliders and at the same time be totally unaware of it.

9. Managerial insights

This research resulted in several implications for the business: the further optimization of production, a better understanding of the problem-definition process and its management, and several practical improvements to the planner.

In Papa et al. (2012) the number of delayed orders and the scheduling expert's time were significantly reduced. However, it often required manual changes to the schedule, due to the decision criteria, which had not been made explicit up to that point, e.g., the same type of product cannot be simultaneously produced on more than one production line, the number of workers (cost) may take higher priority than late deliveries under certain circumstances, etc. These manual changes in most cases resulted in sub-optimum production schedules. This is precisely what we avoid by introducing the multi-objective approach. The planner may adjust the importance of the individual objectives and immediately obtain the corresponding solution, which belongs to the pareto front. We foresee that this approach will further increase the management's comprehension and systemic understanding of how the changes in key variables impact on the production schedule. This should lead to increased efficiency of the production process.

For example, during the years before the current economic recession, a common challenge was to keep up with demand by minimizing late deliveries and the tool change-over downtime. Costs associated with the production were important, but secondary to speed. Our single-objective approach reflects this behavior, but presented difficulties when in 2008 the demand fell sharply and the organization had to become more cost conscious, paying closer attention to minimizing the number of workers. Conversely, the multi-objective approach enables the planner to tackle the fluctuations in business conditions on a daily basis. If the circumstances demand the optimization of costs one day and the minimization of delays the next, the planner can swiftly change the production schedule using our solution.



O1	O2	O3	O4
16	703	414	87
16	706	413	87
16	709	412	87
17	706	405	111
17	712	414	60
17	717	413	60
17	738	412	60
17	732	412	66
17	703	414	68
17	697	415	69
17	708	413	69

Fig. 21. The graphical user interface.

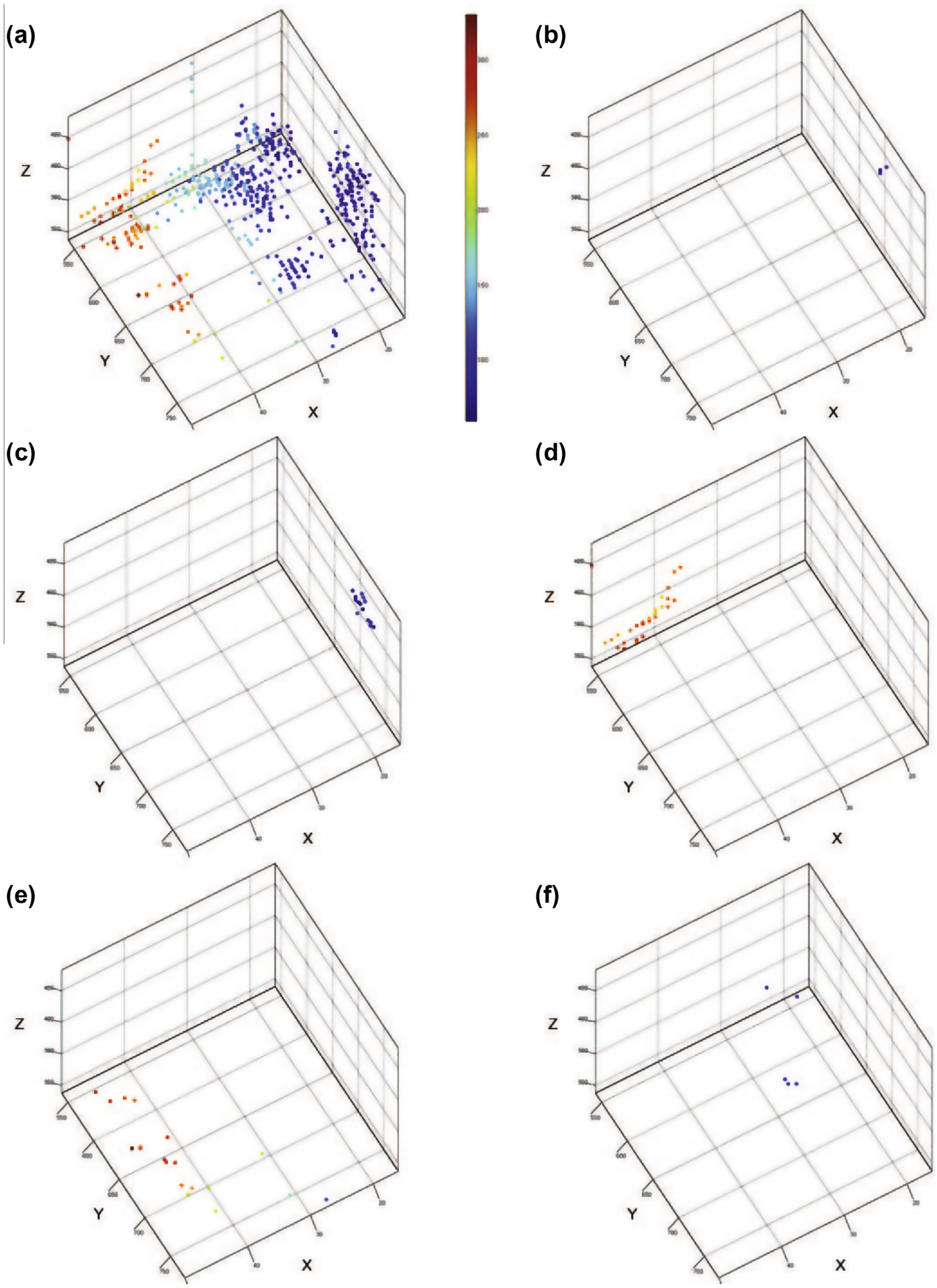


Fig. 22. Comparison of solutions distribution under different limiter settings.

When judging the economic impact of our intervention on the company, we are faced with a problem: how do we value learning? Is it a cost or an investment in capacity for future results? Reflecting on our work, we conclude that it is unlikely to come up with a good problem definition early in the problem-solving process, because it must integrate both business and optimization knowledge.

Yet this integration seems impossible until the participating parties (optimization experts and business people) develop a sufficient degree of common knowledge or shared cognition, as suggested in Markus et al. (2002). The mere communication of ideas proves insufficient for the development of such common reasoning. Concrete solutions are necessary, because many ideas may only be grasped once seen in action. It should therefore be expected that the problem definition will evolve. Over time the capacity to reason together grows by iteratively defining and solving problems together.

For example, our initial work (Papa et al., 2012) was geared towards solving a scheduling problem with what then appeared to be a clearly defined single-objective fitness function. It was not until the planners used our solutions in production that they could volunteer additional insights about the criteria that had not been considered previously. Reflecting on subsequent events we can affirm that given their level of understanding of stochastic optimization, they had not been able to tell us more than they had. We therefore conclude that to avoid any conflict and sometimes failure in optimization projects it is useful to bear in mind that it is a discovery process for both the sides involved.

From the standpoint of the planner, our proposed multi-objective approach (tool) offers an improvement over the single-objective approach in several aspects:

- Intuitive visualization of all the pareto-efficient solutions (see Fig. 21). Previously, the user was able to view only one solution.
- Flexibility to adapt the production schedule to a specific business context. Before, the planner was tied to a pre-determined set of criteria that frequently failed to reflect current business conditions. Manual changes resulted in a sub-optimum schedule.
- Changes to the candidate solutions are instantaneous and may be carried out by moving the sliders (upper-left section of Fig. 21). Previously, it was necessary to re-set the criteria and run the program again in order to calculate the optimum schedule.
- The use of this tool provides the planner with the possibility to gain additional insights into how the production schedule reacts to changes in the weight of each criterion. Among other things, they can instantaneously appreciate the trade-offs when tightening or loosening one or more criteria.

10. Conclusion and future work

A multi-objective approach combined with an actionable presentation of results was applied to meet the demands for increased flexibility of a production-scheduling optimization problem. Our solution facilitates agile decision-making to optimally adapt a production schedule to changes in the business conditions. Hence, this paper provides useful advice to many organizations and practitioners who are likely to tackle similar problems in their attempt to improve production efficiency.

We have tested a memetic, multi-objective approach on some real-world test cases of a production-scheduling problem. Such a problem is a member of the family of job shop scheduling problems, which are known to be NP-hard. Due to the problem's complexity with many constraints, we have used some specialized local searches, which were guided by the multi-objective evolutionary algorithm called the Indicator-Based Evolutionary Algorithm.

We have shown that the use of the memetic, multi-objective approach does not greatly reduce the quality of any objective with regard to the lexicographic evaluation of a single-objective approach. The only major downside of such an approach is in the increased time that is needed for a good pareto front of solutions to be constructed. However, in our case this is acceptable, given that the production-scheduling optimization may run overnight.

We lay out an efficient presentation of the multi-objective results for decision support. Management can flexibly view all the possible solutions in the light of any given business conditions. The production schedule may be instantaneously adapted by adjusting the decision criteria. This provides the management with the possibility to gain additional insights into how the production schedule reacts to changes in the weight of each criterion. This facilitates management's decision-making. Now they can see its full effects prior to making a decision effective.

For future work, we are planning to add a recommendation system, which will suggest a reduced selection of solutions. The recommendations will be generated with data-mining algorithms, based on data about the planner's past selections.

Acknowledgments

Operation part financed by the European Union, European Social Fund.

References

Bäck, T., Fogel, D., & Michalewicz, Z. (2000). *Evolutionary Computation 1: Basic Algorithms and Operators* (2nd ed.). Heidelberg: Taylor & Francis Group.

Basseur, M., & Burke, E. (2007). Indicator-based multi-objective local search. In *IEEE congress on evolutionary computation, CEC 2007* (pp. 3100–3107).

Bole, U., Jaklič, J., Žabkar, J., & Papa, G. (2011). Identification of important factors to success of organizational data mining. In *Proc. 15th Portuguese conference on artificial intelligence, EPIA 2011* (pp. 535–549).

Bozorgirad, M. A., & Logendran, R. (2012). Sequence-dependent group scheduling problem on unrelated-parallel machines. *Expert Systems with Applications, 39*, 9021–9030.

Caumond, A., Lacomme, P., & Tchernev, N. (2008). A memetic algorithm for the job-shop with time-lags. *Computers & Operations Research, 35*, 2331–2356.

Chan, F. T., Au, K., & Chan, P. (2006). A decision support system for production scheduling in an ion plating cell. *Expert Systems with Applications, 30*, 727–738.

Chica, M., Cordón, O., Damas, S., & Bautista, J. (2012). Multiobjective memetic algorithms for time and space assembly line balancing. *Engineering Applications of Artificial Intelligence 25*, 254–273 (Special Section: Local Search Algorithms for Real-World Scheduling and Planning).

Chryssolouris, G., & Subramaniam, V. (2001). Dynamic scheduling of manufacturing job shops using genetic algorithms. *Journal of Intelligent Manufacturing, 12*, 281–293.

Custódio, A., Madeira, J., Vaz, A., & Vicente, L. (2011). Direct multisearch for multiobjective optimization. *SIAM Journal on Optimization, 21*, 1109–1140.

Czogalla, J., & Fink, A. (2008). On the effectiveness of particle swarm optimization and variable neighborhood descent for the continuous flow-shop scheduling problem. In F. Xhafa & A. Abraham (Eds.), *Metaheuristics for scheduling in industrial and manufacturing applications. Studies in computational intelligence* (Vol. 128, pp. 61–89). Berlin, Heidelberg: Springer.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms. Wiley-Interscience series in systems and optimization*. Chichester: John Wiley & Sons.

Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, & J. Merelo, et al. (Eds.), *Parallel problem solving from nature PPSN VI. Lecture notes in computer science* (Vol. 1917, pp. 849–858). Berlin, Heidelberg: Springer.

Durillo, J. J., & Nebro, A. J. (2011). jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software, 42*, 760–771.

Garca-Martnez, C., & Lozano, M. (2008). Local search based on genetic algorithms. In G. Rozenberg, T. Bck, J. N. Kok, H. P. Spaink, A. E. Eiben, & P. Siarry, et al. (Eds.), *Advances in metaheuristics for hard optimization. Natural computing series* (pp. 199–221). Berlin, Heidelberg: Springer.

Goldberg, D. E. (1989). *Genetic algorithms in search. Optimization and machine learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc..

Gong, J. (2010). Multi-objective planning and scheduling. In *Proc. 2nd international conference on mechanical and electrical technology (ICMET 2010)* (pp. 384–388).

Ishibuchi, H., Tsukamoto, N., & Nojima, Y. (2008). Evolutionary many-objective optimization: A short review. In *IEEE congress on evolutionary computation, CEC 2008* (pp. 2424–2431).

Ishibuchi, H., Yoshida, T., & Murata, T. (2003). Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation, 7*, 204–223.

Kamrul Hasan, S. M., Sarker, R., Essam, D., & Cornforth, D. (2009). Memetic algorithms for solving job-shop scheduling problems. *Memetic Computing, 1*, 69–83.

Knowles, J., & Corne, D. (2005). Memetic algorithms for multiobjective optimization: Issues, methods and prospects. In W. Hart, J. Smith, & N.

Krasnogor (Eds.), *Recent advances in memetic algorithms. Studies in fuzziness and soft computing* (Vol. 166, pp. 313–352). Berlin, Heidelberg: Springer.

Li, J., Burke, E. K., Curtois, T., Petrovic, S., & Qu, R. (2012). The falling tide algorithm: A new multi-objective approach for complex workforce scheduling. *Omega, 40*, 283–293.

Li, X., Gao, L., Zhang, C., & Shao, X. (2010). A review on integrated process planning and scheduling. *International Journal of Manufacturing Research, 5*, 161–180.

Mansouri, S. A., Gallear, D., & Askariazad, M. H. (2012). Decision support for build-to-order supply chain management through multiobjective optimization. *International Journal of Production Economics, 135*, 24–36. Advances in Optimization and Design of Supply Chains.

Markus, M. L., Majchrzak, A., & Gasser, L. (2002). A design theory for systems that support emergent knowledge processes. *MIS Quarterly, 26*, 179–212.

Michalewicz, Z., & Fogel, D. (2004). *How to solve it: Modern heuristics* (2nd ed.). Berlin, Heidelberg: Springer-Verlag.

Mueller-Gritschneder, D., Graeb, H., & Schlichtmann, U. (2009). A successive approach to compute the bounded pareto front of practical multiobjective optimization problems. *SIAM Journal on Optimization, 20*, 915–934.

Nareyek, A. (Ed.), (2000). Local search for planning and scheduling. In *ECAI 2000 workshop*, Berlin, Germany, August 21, 2000. Revised Papers. volume 2148 of Lecture Notes in Computer Science, Springer.

Papa, G. (2008). Parameter-less evolutionary search. In *Proc. genetic and evolutionary computation conference (GECCO 2008)* (pp. 1133–1134).

Papa, G., Vukašinović, V., & Korošec, P. (2012). Guided restarting local search for production planning. *Engineering Applications of Artificial Intelligence, 25*, 242–253.

Rentmeesters, M., Tsai, W., & Lin, K.J. (1996). A theory of lexicographic multi-criteria optimization. In *Proc. second IEEE international conference on engineering of complex computer systems* (pp. 76–79).

Ruiz, R., & Stützle, T. (2006). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*, 2033–2049.

Shaotan, X., A, L. X., Liang, G., & Yi, S. (2010). General particle swarm optimization algorithm for integration of process planning and scheduling. *Advanced Materials Research*, 409–413.

Siarry, P., & Michalewicz, Z. (Eds.). (2008). *Advances in metaheuristics for hard optimization. Natural computing series*. Springer.

Sobol', I., & Levitan, Y. (1999). A pseudo-random number generator for personal computers. *Computers & Mathematics with Applications, 37*, 33–40.

Wagner, T., Beume, N., & Naujoks, B. (2007). Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, & T. Murata (Eds.), *Evolutionary multi-criterion optimization. Lecture notes in computer science* (Vol. 4403, pp. 742–756). Berlin, Heidelberg: Springer.

Zhang, R., & Wu, C. (2012). A hybrid local search algorithm for scheduling real-world job shops with batch-wise pending due dates. *Engineering Applications of Artificial Intelligence, 25*, 209–221. Special Section: Local Search Algorithms for Real-World Scheduling and Planning.

Zitzler, E., & Künzli, S. (2004). Indicator-based selection in multiobjective search. In *Proc. 8th international conference on parallel problem solving from nature (PPSN VIII)* (pp. 832–842). Springer.

Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In Giannakoglou, K. C., Tsahalis, D. T., Périaux, J., Papailiou, K. D., & Fogarty, T. (Eds.), *Evolutionary methods for design optimization and control with applications to industrial problems, international center for numerical methods in engineering*, Athens, Greece (pp. 95–100).

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., & Grunert da Fonseca, V. (2003). Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation, 7*, 117–132.