MULTIPLE NEIGHBORHOODS IN TABU SEARCH: SUCCESSFUL APPLICATIONS FOR OPERATIONS MANAGEMENT PROBLEMS

UDK 519.8:005 / JEL C44 ; M11 / PRELIMINARY COMMUNICATION

NICOLAS ZUFFEREY
PROFESSOR
GENEVA SCHOOL OF ECONOMICS AND MANAGEMENT
GSEM – UNIVERSITY OF GENEVA
SWITZERLAND
n.zufferey@unige.ch

ABSTRACT

A *metaheuristic* is a refined solution method able to find a satisfying solution to a difficult problem in a reasonable amount of time. A *local search* metaheuristic works on a single solution and tries to improve it iteratively. *Tabu search* is one of the most famous local search, where at each iteration, a neighbor solution is generated from the current solution by performing a specific modification (called a move) on the latter. In contrast with most of the existing literature, the goal of this paper is to present tabu search approaches where different neighborhood structures (i.e., different types of moves) are jointly used. The discussion is illustrated for various operations management problems: truck loading, job scheduling, inventory management, and dimensioning of assembly lines.

KEY WORDS: optimization, operations management, tabu search, metaheuristics.

1. INTRODUCTION

Let f be an objective function which has to be minimized (e.g., a cost function). A solution s is *optimal* for f if there is no better solution than it, that is, there is no solution s' such that $f(s') < f(s)$. As mentioned in (Zufferey & Vasquez, 2015), an *exact* method guarantees the optimality of the provided solution. However, for a large number of applications and most real-life optimization problems, such methods need a prohibitive amount of time to find an optimal solution, because such problems are NP-hard (Garey & Johnson, 1979). For these difficult problems, one should prefer to quickly find a satisfying solution, which is the goal of *heuristic* and *metaheuristic* solution methods. There mainly exist three families of (meta)heuristics: *constructive* algorithms (a solution is built step by step from scratch, like the *greedy* algorithm where at each step, the best element is added to the solution under construction), *local search* methods (a solution is iteratively modified: this will be discussed below), and *evolutionary* metaheuristics (a population of solutions is managed, like genetic algorithms and ant algorithms). The reader is referred to (Gendreau & Potvin, 2010; Zufferey, 2012b) for more information on metaheuristics and general guidelines to adapt them.

Only the context of local search methods is considered in this work. A local search algorithm starts with an initial solution and tries to improve it iteratively. At each iteration, a modification, called a *move*, of the current solution s is performed in order to generate a neighbor solution s'. Let $N(s)$ denote the set of all neighbor solutions of s. The definition of a move, that is the definition of the *neighborhood structure* N, depends on the considered problem. Popular local search methods are the descent local search, simulated annealing, tabu search and variable neighborhood search.

In a *descent local search*, the best move is performed at each iteration and the process stops when a local optimum is found. *Tabu search* was proposed by Fred Glover in the 80's and is nowadays still considered as one of the most efficient method for exploring the search space. Tabu search has a good balance between *exploitation* (i.e., the ability to guide the search in the solution space and to take advantage of the problem structure) and *exploration* (i.e., the ability to visit various zones of the solution space). Indeed, to prevent tabu search from being stuck in a local optimum, when a move is performed, the reverse move is forbidden (i.e., set as *tabu*) for *tab* (parameter) iterations. In most tabu search algorithms, only one neighborhood structure N is used. The goal of this paper is to present tabu search approaches where at each iteration, different neighborhood structures N_1, N_2, ..., N_q are used. The resulting method, denoted MNTS (for *Multiple Neighborhoods in a Tabu Search*), is presented in Algorithm 1, where s^* denotes the best visited solution (returned at the end to the user). The motivation of using several neighborhood structures is the following.

A *local optimum* according to neighborhood structure N is a solution s such that there is no solution in $N(s)$ which is better than s. Let N_1 and N_2 be two different neighborhood structures. Obviously, if s is a local optimum according to N_1, it may not be a local optimum according to N_2. In other words, if a N_1-move is not able to improve s anymore, then a N_2-move may do it.

Algorithm 1: MNTS (*Multiple Neighborhoods in a Tabu Search*)

Generate an initial solution s and set $s^* = s$.

While no stopping criterion is met, do

- from the current solution s, generate the best non-tabu neighbor $s' \in N_1(s) \cup \ldots \cup N_q(s)$;
- forbid the reverse move for *tab* (parameter) iterations;
- set $s = s'$;
- if $f(s) < f(s^*)$, set $s^* = s$;

In order to design a MNTS for a specific problem (P), the following ingredients have to be defined: a way to encode a solution s, an objective function f, the various neighborhood structures $N_1, \ldots, N_q$, the tabu list structures (i.e., the nature of the forbidden moves), and the stopping condition (e.g., a time limit, a specific number of iterations). Strongly relying on (Respen & Zufferey, 2013; Thevenin et al., 2013; Zufferey, 2014; Zufferey, 2012a), the discussion is illustrated for various operations management problems, namely truck loading (Section 2), job scheduling (Section 3), inventory management (Section 4), and dimensioning of assembly lines (Section 5).

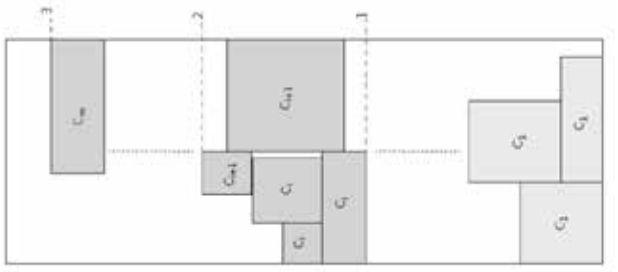
2. MNTS FOR TRUCK LOADING

2.1. Presentation of the considered problem (P)

The French car manufacturer *Renault* daily faces a complex truck loading problem (P), where items need to be placed in a truck while satisfying different constraints. More than a thousand trucks are daily considered to deliver components to the car plants. As a single truck can deliver goods to different delivery points, *classes* of items are defined, where a class is associated with a delivery point. Each problem instance contains the size of the truck and the various sizes of all the items that must fit in. The heights of the items can be ignored as they rely on complex factory constraints which are supposed to be already satisfied. At first sight, (P) seems related to a strip-packing 2D problem with rotation, which has been already covered by many research papers (e.g., Hopper & Turton, 2001; Lodi et al., 2002; Ntene & van Vuuren, 2009; Riff et al., 2009). (P) is NP-hard, and *Renault* even showed in (Nguyen & Brenaut, 2009) that no exact method can be competitive to tackle their real instances. Therefore, (meta)heuristics are more than relevant.

Problem (P) can be formally described as follows: a number n of items, each one belonging to a specific class C_i (with $i \in \{1, \ldots, m\}$ such that $m \leq n$), need to be placed in a truck such that all the items belonging to the same class are adjacent. In addition, the classes must be placed in an increasing fashion from the front to the rear of the truck. More formally, the ordinate of the origin item which belongs to class C_i (label 1 on Figure 1) must be strictly smaller than the ordinate of the extremity of any item of class C_{i+1} (label 2 on Figure 1), and such that the ordinate of the extremity item (the closest one to the rear) of class C_m (label 3 on Figure 1), denoted as f, is minimized. The truck size is a hard constraint to fulfill, as it is not allowed to exceed neither its length nor its width.

Figure 1. A possible solution (view from the top of the truck, with rear on the left)



Because of the specificity of (P) (e.g., different classes of items, a significant number of items per truck in conjunction with a large standard deviation of the sizes of the items), it is not possible to take advantage of the existing exact algorithms (e.g., (Martello & Vigo, 1998; Martello et al., 2003; Lesh et al., 2004; Pisinger & Sigurd, 2005; Puchinger & Raidl, 2007)) to tackle it. Additional references on the topic can be found in (Lodi et al., 1999; Respen & Zufferey, 2013).

To solve (P), *Renault* proposes a simple but efficient greedy heuristic (denoted *SG*), and an advanced greedy heuristic called *look-ahead greedy* (denoted *LAG*). An important aspect is the tradeoff between the computing time and the solution quality. As *LAG* is fast, another metaheuristic is only relevant if it is fast and leads to improvements. *SG* builds a solution from scratch, and at each iteration, selects an item (following different possible rules) from a list L of non-already inserted items, and adds it to the solution at minimum cost (i.e., which minimizes the augmentation of f, label 3 of Figure 1). This process stops when L is empty. In *LAG*, at each iteration, the algorithm tries each item j of L, and for each j, tries the next p (parameter) insertions following this possible insertion of j (look-ahead process). At the end of the iteration, the item j that would involve the lowest cost in the next p iterations is selected and inserted at the best position. As before, this process stops when L becomes empty. Both *SG* and *LAG* algorithms are fast (a few seconds per run), and therefore relevant to *Renault*. The two methods perform restarts as long as a given time limit T is not reached. When T is reached, the best generated solution is returned.

2.2. Tabu search for (P)

Working with *encoded* solutions is relevant to tackle (P) within the framework of local search methods. An encoded solution *s* is actually a list of elements. To build a solution and compute its quality, a *decoding greedy algorithm* (*DGA*) is performed on the encoded solution. It decodes the solution *s* into a real solution s^{real}, and then returns the total length of the truck load *f*. To drive *DGA*, information (some are mandatory and others are optional) are carried in each element of *s*, and can contain the item identifier (*ID*, mandatory), the class identifier (*C*, mandatory), the item orientation (*O*, optional), and the item side (*S*, optional). Thus, component *i* of the solution *s* takes the form $s_i = (ID_i, C_i, O_i, S_i)$, where $ID \in \{1, \ldots, n\}$, $C \in \{1, \ldots, m\}$, $O \in$ {not rotated, 90-degrees rotated}, and $S \in$ {left-sided, right-sided}. *DGA* thus decodes the vector *s* into a real solution s^{real} by inserting in s^{real} the items from *s* in a FIFO order, and using the *O* and *S* information (if provided) while respecting the class constraint. At each step, *DGA* pops the next item *i* of *s*, greedily loads it in the truck, while respecting C_i, O_i and S_i. If O_i (or S_i) is not provided, *DGA* can decide by itself its value (minimizing the augmentation of *f*), and thus owns more freedom.

The decoding process is illustrated on an example. Five items, initially oriented as presented in Figure 2, have to be placed in a truck. A possible encoded solution is the following: *s = ((ID$_1$, 1, not rotated, right-sided), (ID$_3$, 1, 90-degrees rotated, right-sided), (ID$_2$, 1, 90-degrees rotated, left-sided), (ID$_5$, 2, ?, ?), (ID$_4$, 2, ?, right-sided))*. The corresponding decoded solution s^{real} is illustrated in Figure 3. To generate this solution, *DGA* performs the following steps: it pops the first element ID_1 and loads the corresponding item on the right side, without rotation. At that time, the next item, namely ID_3, is loaded on the right with a 90-degrees rotation. Then item ID_2 is loaded on the left with rotation whereas ID_5 is inserted at the best possible position tried by *DGA* while respecting the class constraint. Finally item ID_4 is inserted on the right side but *DGA* decided its orientation. When *DGA* is over, it returns the value *f*, which is in this example the extremity of item ID_4.

Figure 2: Items (with initial orientations)

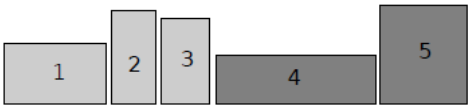
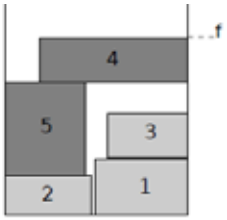


Figure 3: Decoded solution s^{real} (top view)



To generate a neighbor solution *s'* from the current solution *s*, the seven following neighborhood structures are possible:

- N_1: move item *j* from position *x* to position *y*;
- N_2 (resp. N_3): move item *j* from position *x* to *y*, and switch to the opposite value its orientation *O* (resp. side *S*);
- N_4: move item *j* from position *x* to *y*, and switch *S* and *O* to the opposite values;
- N_5 (resp. N_6): switch *O* (resp. *S*) to the opposite value;
- N_7: switch *O* and *S* to the opposite values.

All the moves are performed while respecting the class constraint, and ties are broken randomly. Four different tabu search approaches are developed, denoted TS_1, TS_2, TS_3 and TS_4. The notation *TS* is simply used if it refers to common features of the four tabu search algorithms. *TS* starts from an initial encoded solution where items are ordered by decreasing areas. TS_1, TS_2, TS_3 and TS_4 differ in the sense that for TS_1, only the information *ID* and *C* is contained in each element of the encoded solution. TS_2 contains *ID*, *C* and *O*. TS_3 contains *ID*, *C* and *S*. TS_4 contains *ID*, *C*, *O* and *S*. Thus, for TS_1, *DGA* can decide on its own the orientations and the sides (while focusing on the smallest augmentation of *f*). In TS_2 and TS_3, *DGA* has the order in which the insertion must be made, and the orientation or the side of each item (but not both). Finally TS_4 constraints *DGA* at the maximum level due to the complete information set contained in each component s_i of the vector *s*. In *TS*, only a fraction *v* (set to 50%) of the possible moves are generated (it allows to performing more iterations for a same time limit, and helps in bringing more diversification). After each move, the tabu tenure *tab* is set to a uniformly distributed value between 25 and 55.

2.3. Results

The above methods are compared on a set of 30 real benchmark instances provided by *Renault*. Tests were performed on an Intel Quad-core i7 @ 3.4 GHz with 8 GB DDR3 of RAM memory. The time limit *T* is 1800 seconds (as validated by practitioners). The results are summarized in Table 1. The first two columns indicate the values of *n* (number of items) and *m* (number of classes) of each instance. Column *f** indicates to the objective function value of the best solution ever found by any of the algorithms. The following column reports the percentage gap between the solution of *SG* and *f**. The next columns provide the same information for the other methods. The last rows give the average gap and computing time for each method. As *SG* and *LAG* are greedy constructive methods with restarts, it is not relevant to indicate their computing times.

Table 1. Computational results (averaged over 5 runs for tabu search)

n	*m*	*f**	*SG*	*LAG*	TS_1	TS_2	TS_3	TS_4
23	1	12840	1.01	0.86	4.6	0.06	0.21	0.92
25	1	13000	3.85	0.31	3.85	0	0.05	0.98
24	1	12920	1.32	0.08	1.39	0	0.06	0.38
25	1	13040	3.76	0.31	3.76	0	0	0.6
26	1	13460	0.15	0.07	0.15	0	0	0.01
20	2	13610	6.83	1.25	14.11	0	1.25	0.53
23	1	12980	3.08	0.08	6.63	0	0	0.26
25	1	14120	4.02	4.01	3.75	0.07	0	0.26
18	4	13295	0.56	0.15	24.69	0	0.05	0.18
23	3	12852	5.23	0.5	18.67	0	0.03	0.16
20	2	13330	9.08	0.6	12.08	0	0.08	0.08
17	3	13070	0.23	0.15	14.77	0	0	0
25	1	13390	1.12	0.97	1.05	0	0.16	0.42
20	2	13150	7.53	1.52	12.09	0	0.61	0.67
20	4	13325	4.68	2.42	25.77	0	2.65	0.11
24	1	13010	3.69	0.46	3.61	0	0	0.52
23	4	12902	3.36	2.19	6.17	0.02	1.02	0.39
24	1	13380	1.12	0.6	4.11	0	0	0.04
24	1	13380	1.12	0.6	4.11	0	0	0.04
23	1	13040	3.68	0.23	6.75	0	0.05	0.26
25	1	13020	3.76	0.23	3.76	0	0	0.53
25	1	13380	0.82	0.6	0.75	0	0	0.2
24	1	13380	1.12	0.6	4.04	0	0	0.01
18	2	11530	10.41	0.95	11.88	0	0	0
23	1	12550	21.12	0	9.96	0	0	0
19	2	12170	6.98	0.33	9.2	0	0	0
23	1	13070	3.83	0.23	6.96	0	0.05	0.51
25	1	13380	0.75	0.6	0.75	0	0	0.12
20	1	13300	1.5	1.13	17.29	0	0.18	1.01
25	1	13080	3.59	0.08	3.59	0	0	0.28
Average gaps			**3.98**	**0.74**	**8.01**	**0**	**0.21**	**0.32**
Average times [s]					**176**	**232**	**240**	**645**

The superiority of *LAG* over *SG* is significant. This is mainly due to the fact that *LAG*, with the parameter *p*, explores the impact of future possible insertions. Tabu search does not show amazing improvements on the greedy heuristics. One can conjecture that it is due to the fact that *f** is not far from the optimum. TS_1 is a method to avoid as it obtains poor results. This is probably due to the full freedom given to *DGA* to build a solution, when compared to the other algorithms. TS_4 is less competitive than TS_2 and TS_3. This can be easily explained by the fact that TS_4 does not let *DGA* any choice on the solution building. It is thus not surprising that TS_4 is the fastest method (regarding the time needed per iteration) of the *TS* family, as *DGA* does not have to perform many tests to insert an item at the best position. However, even if TS_4 it is the fastest method per iteration, it is the slowest method to find its best solutions. Experiments clearly show that some freedom should be given to *DGA*, as TS_2 and TS_3 are the most powerful tabu search methods. Remember that the difference between TS_2 and TS_3 is the carried information *O* or *S*. As TS_2 has the minimum gap on 29 instances over 30, it shows that *O* is the most important feature to carry in the encoded solution *s*. This can be explained in the sense that the orientation *O* has an important impact, as it clearly drives s^{real} by forcing the orientation of each item.

3. MNTS FOR JOB SCHEDULING

3.1. Presentation of the considered problem (P)

On the one hand, the range of problems consisting in selecting a subset of given jobs, and then in scheduling them in order to minimize rejections and some other costs, are called *order acceptance and scheduling* problems (OAP). It has been studied in various scheduling environments and a review is given in (Slotnick, 2011). Such problems are particularly relevant in make-to-order production systems (Zorzini et al., 2008). On the other hand, earliness and tardiness penalties have captured a lot of attention due to their correspondence with the just-in-time paradigm. In (Valente & Gonçalves, 2009), it is mentioned that the use of quadratic tardiness functions is appropriate to model customers' dissatisfaction. In (Valente et al., 2011), the authors emphasize that quadratic penalties avoid situations in which only a few jobs contribute to the objective function.

When the production capacity of a company is overloaded, all received orders cannot be performed on time. It then makes sense to reject some of them. In the considered problem (P), following the customers' requirements, a *due date* corresponds to the date at which an order has to be delivered. Late deliveries lead to customers' dissatisfaction, which is modeled by a quadratic tardiness penalty depending on the completion time of the job. The *deadline* corresponds to the point in time where the dissatisfaction associated with the rejection of the order, modeled by a rejection penalty, is equal to the dissatisfaction of delivering late. In other words, it is preferable to reject the order to allow the client to get its goods by another supplier. Usually, no job can be scheduled before its *release date*. It often corresponds to the date at which all the necessary raw material is ready to be used. In contrast, release dates can be reduced in (P) if a quadratic earliness penalty (depending on the starting time of the job) is paid. Obviously, there is a lower bound and no job can start before its *available date*. The use of controllable release dates is relevant in practice. As explained in (Shakhlevich & Strusevich, 2006), it may be profitable for the manufacturer and its suppliers to cooperate. In some cases, a supplier can allow to deliver raw material earlier, which reduces the release dates at the manufacturer's level. In counterpart, the manufacturer will pay a higher price to the involved supplier, which creates a win-win situation.

Between two consecutive jobs j and j' of different families F and F', a setup time $s_{FF'}$ must be performed and a setup cost $c_{FF'}$ is incurred. The considered problem (P) can be formally stated as follows. A set of n jobs is given, a subset of these jobs has to be selected and scheduled (without preemptions but idle times are allowed) on a single machine which can handle only one job at a time. For each job j are given: a processing time p_j, an available date R_j, a release date r_j, a due date d_j, a deadline D_j, and a rejection penalty u_j. Let C_j and B_j respectively denote the completion time and the starting time of job j. In a feasible solution, each accepted (i.e., not rejected) job j satisfies $C_j \leq D_j$ and $B_j \geq R_j$. The earliness and tardiness penalties are respectively defined as follows:

- $E_j(B_j) = w_j \cdot (r_j - B_j)^2$ if $B_j < r_j$, and 0 otherwise;
- $T_j(C_j) = w_j' \cdot (C_j - d_j)^2$ if $C_j > d_j$, and 0 otherwise.

The objective function to minimize is the sum of the three following components: (1) the setup costs $c_{jj'}$ between every successively performed jobs j and j'; (2) the rejection penalties u_j associated with each rejected job j; (3) the earliness and tardiness penalties $E_j + T_j$ for all accepted jobs j. In the context of (P), the reader interested in additional references is referred to (Yalçin et al., 2007; Oguz et al., 2010; Cesaret et al., 2012; Thevenin et al., 2013).

In (Thevenin et al., 2012), a greedy algorithm and a tabu search are proposed for a similar problem but with *regular* (i.e., non-decreasing) cost functions instead of earliness and tardiness penalties. In contrast with most scheduling objective functions, the one considered in (P) is not regular since earliness penalties are decreasing functions of the completion times. When objective functions are regular, most algorithms solving a single machine scheduling problem consist in finding an ordered sequence of jobs. From such a sequence, a schedule is easily built by starting each job as early as possible. In case of non-regular cost functions, the insertion of idle times may decrease the costs. Therefore, building an optimal schedule when a production sequence is given is not as easy, and can be time-consuming. The timing procedure proposed in (Hendel & Sourd, 2007) is adapted for (P), which is particularly efficient within the framework of local search algorithms.

3.2. Solution methods for (P)

To solve (P), a solution s is modeled by an ordered sequence $\sigma(s)$ of jobs, and a set $\Omega(s)$ of rejected jobs. Given such a solution representation, the timing procedure computes the starting and ending times of each job of $\sigma(s)$, such that the objective function is minimized. A greedy algorithm and a tabu search approach are now presented for (P). The first phase of the greedy method consists in sorting the jobs by increasing *slack time* $(D_j - R_j - p_j)$, where ties are broken by decreasing rejection penalties u_j. In a second phase, jobs are taken one by one in the previously defined order, and inserted in the schedule at the position minimizing the costs. Note that a job is rejected if it is better than inserting it. The insertions are enforced, that is, other jobs can be deleted to maintain feasibility. The associated repairing procedure is explained below.

Four types of straightforward moves can be used for tabu search. All moves are enforced by using one of the repairing procedures described later.

- N_1: *Add* takes a rejected job and inserts it in the schedule.
- N_2: *Drop* takes an accepted job and removes it from the schedule.

- N_3: *Reinsert* takes an accepted job and inserts it elsewhere in the schedule.
- N_4: *Swap*, exchanges the position of two jobs in $\sigma(s)$.

Five different tabu structures are designed. The first (resp. second) forbids to add (resp. removing) a dropped (resp. added) job during t_1 (resp. t_2) iterations. The third forbids to move a job which has been added/reinserted/swapped during t_3 iterations. The fourth forbids to move a job j between its two previous neighbors of $\sigma(s)$ during t_4 iterations, if j has been reinserted/swapped. As the cost function associated with each job is constant over the interval $[r_j, d_j]$, it induces plateaus in the search space. To escape from them, a tabu status is associated with the cost of the most recently visited solutions during t_5 iterations: it is forbidden to visit a solution whose cost is tabu.

As mentioned above, inserting a job may lead to an unfeasible solution due to available dates and deadlines constraints. To maintain feasibility, a repairing procedure must delete some jobs, and the choice of those jobs is a crucial point in local search methods for OAP. Note that a *Reinsert* move can be performed by a *Drop* move followed by an *Add* move, and a *Swap* move consists of two *Drops* followed by two *Adds*. As dropping a job cannot lead to unfeasible solutions, a repairing procedure is only needed for the move *Add*. Assuming that job j is inserted at position p, three repairing procedures can be compared.

- $REPAIR_1$. Remove randomly a job adjacent to position p until the insertion of j is possible. Deleting jobs which are adjacent to the insertion position reduces the shifting of other jobs, which is expensive with quadratic penalties.
- $REPAIR_2$. Let j' and j'' be two jobs such that j' is at the left of p, and j'' at its right. Jobs j' and j'' are *blocking* if by shifting j' (resp. j'') as most as possible towards the left (resp. right), the insertion of j is still not possible. $REPAIR_2$ deletes one of the closest blocking jobs to p until the insertion of j is possible. These blocking jobs are likely to be associated with large earliness/tardiness penalties, and dropping them should not be expensive.
- $REPAIR_3$. While the solution is not feasible, greedily remove a job (based on the costs).

3.3. Results

To generate a set of instances for (P), two critical values are used: the number n of jobs, and a parameter α which controls the interval of time in which release dates and due dates are generated. More precisely, a value *Start* is chosen large enough, and *End* is equal to $Start + \alpha \cdot \sum_j p_j$. Then, r_j is chosen in the interval [*Start*, *End*], and d_j in $[r_j + p_j, End]$. Basically, methods are likely to reject more jobs for small values of α. n belongs to {25, 50, 100, 200} and α to {0.5, 1, 2}. One instance for each pair (n, α) is generated. The weights w_j and w_j' are randomly chosen in {1, 2, 3, 4, 5}. D_j and R_j are chosen such that $T_j(D_j) = E_j(R_j) = u_j$. p_j is an integer randomly chosen in [50, 100]. As observed in realistic situations, the rejection penalty u_j is related to the processing time: $u_j = \beta \cdot p_j$, where β is an integer randomly picked in [50, 200]. The number of job families belongs to [10, 20]. Setup times and costs are likely to be related in realistic situations, therefore $s_{FF'}$ is chosen in [50, 200] and $c_{FF'} = \gamma \cdot s_{FF'}$, where γ is chosen in the interval [0.5, 2].

Four methods are compared. *Greedy* refers to the greedy method combined with $REPAIR_2$ (the other repairing procedures do not provide good results when combined with *Greedy*). TS_i is the tabu search using repairing procedure $REPAIR_i$ (with $i \in \{1, 2, 3\}$). Parameters $(t_1, t_2, t_3, t_4, t_5)$ are set to (80, 60, 90, 180, 30) for $n \in \{50, 100, 200\}$, and to (20, 20, 15, 25, 10) for $n = 25$. Average results (over 5 runs) are presented in Table 2, where the column f^* reports the best result found by any of the presented methods for the considered instance. In each cell is indicated the percentage gap between the average result obtained by the concerned method and f^*. The results clearly show the superiority of tabu search over *Greedy*, as the gap obtained by the best tabu search is 9.22%, versus 28.70% for *Greedy*. Tabu search with repairing procedure $REPAIR_3$ obtains the best results for 8 instances over 12, however the results obtained for large instances are bad. This is not surprising as $REPAIR_3$ is efficient but very slow. $REPAIR_2$ is slightly better than $REPAIR_1$: their respective average gaps are 9.22% and 10.22%. One would thus advise the use of repairing procedure $REPAIR_3$ for small instances, and $REPAIR_2$ for larger ones.

Table 2. Computational results

n	α	f^*	*Greedy*	TS_1	TS_2	TS_3
25	0.5	115361	0	0.13	0	0
25	1	28602	6.7	0	7.39	0
25	2	149134	0	0	0	0
50	0.5	237414	0.89	2.03	3.62	0
50	1	148237	12.88	13.12	11.39	4.22
50	2	38899	32.4	2.72	2.55	0
100	0.5	550950	5.36	3.33	4.69	1.11
100	1	339100	23.9	12.77	10.95	3.54
100	2	31706	176.18	57.88	42.38	204.47
200	0.5	934898	22.17	0.94	1.43	10.66
200	1	473244	68.62	3.03	4.62	84.63
200	2	42397	302.82	11.44	12.45	1586.88
Average			**28.7**	**10.22**	**9.22**	**23.7**

4. MNTS FOR INVENTORY MANAGEMENT

4.1. Presentation of the considered problem (P)

In most inventory management problems, two types of decision have to be made at the manufacturer level: *when* and *how much* to order to suppliers. It is assumed that setup, carrying and shortage costs are encountered during the year. Usually, inventory management models are characterized by stochastic demand and constant lead times. In contrast, the approach proposed in (Silver & Zufferey, 2005) and generalized below deals with the situation where there is a constant known demand rate, but probabilistic lead times whose probability distributions change seasonally. The lead times for different orders are assumed to be independent, thus crossovers can occur. Therefore, the interactive effects between different cycles (a *cycle* is defined as the time between two consecutive orders) due to the occurrence of shortages are difficult to model. Consequently, even if the annual *approximated* costs can be analytically computed with a mathematical function *f*, simulation (of the lead times) is the only way to compute the annual *actual* costs *F* of a solution.

The considered problem (P) was motivated by the management of raw material at a sawmill in North America. Without loss of generality, consider a 52-weeks planning horizon (a time period is a week). A solution *(P,S)* can be modeled by two vectors *P* and *S* defined as follows: P_t = 1 if an order occurs at the beginning of week *t*, and P_t = 0 otherwise; S_t is the order-up-to-level of available inventory at the beginning of week *t* if P_t = 1, and S_t = 0 if P_t = 0. The following reasonable assumptions are made. (1) It is possible to analytically *approximate* the annual costs with a function *f(P,S)* relying only on *P*, *S* and the probability distributions of the lead times. (2) It is possible to compute *F(P,S)* (i.e., the annual *actual* costs) with a simulation tool. (3) Based on *f*, it is possible to analytically compute *S* from *P* with a so-called *Compute(S|P)* procedure. As a consequence, anytime *P* is modified, its associated *S* vector can be immediately updated with *Compute(S|P)*.

4.2. Design of a solution method

Due to the non-stationarity in the lead time distribution, the problem is combinatorial in nature (choice of the P_t's and S_t's). Moreover, simulation is required to compute the actual cost of a solution. Thus, it makes sense to use (meta)heuristics. The solution space *X(N)* is defined as the set of all the solutions *(P,S)* with *N* orders. The approach consists in providing good solutions for different solutions spaces, starting with *U(N)* orders and ending with *L(N)* orders, where $U(N) \leq 52$ (resp. $L(N) \geq 1$) is an upper (resp. a lower) bound on *N*. At the end, the overall best solution is returned to the user. For a fixed solution space *X(N)*, the following steps are performed.

(S1) Generate an initial solution *(P,S)* with *N* orders as equi-spaced as possible.

(S2) Based on *f*, try to reduce the approximate costs of *(P,S)* with a tabu search $TS_f(P,S)$ working on *P* (using neighborhood N_1, as described below).

(S3) Based on *F* and without changing *P*, apply a descent local search *DLS(S)* working on *S* (a move in this neighborhood N_2 consists in augmenting or reducing an S_t by one unit).

In $TS_f(P,S)$, a move in the neighborhood structure N_1 consists in putting an order earlier or later, but without changing the global sequence of orders. When an order is moved, then it is forbidden (tabu) to move it again for *tab* (parameter depending on *N*) iterations. The stopping condition is a maximum number *Iter* (parameter) of iterations without improvement of the best visited solution.

An extension of $TS_f(P,S)$, denoted $TS_f^M(P,S)$, is now presented for step (S2). Instead of only providing a single solution, a set *M* containing *m* (parameter) promising local optima is provided (promising according to the quality function *f* and a diversity function *Div(M)*). To achieve this, additional ingredients are now defined. The distance between *P* and *P'* is $Dist(P,P') = \sum_t |P_t - P_t'|$. The distance between *P* and a set *M* of solutions is defined as $Dist(P,M) = |M|^{-1} \cdot \sum_{P' \in M} Dist(P,P')$. The *diversity* of a set *M* of solution is computed as $Div(M) = |M|^{-1} \cdot \sum_{P \in M} Dist(P,M-\{P\})$. *M* is initialized with solutions randomly generated. Let *P* be a solution found by tabu search at the end of an iteration. The key idea is the following: *P* should replace a bad (according to *f*) solution of *M* which poorly contributes to its diversity *Div(M)*. More precisely, let *M'* be the subset of *M* containing the *m'* (parameter) worst solutions of *M*, for which the worst value is f**. Let $P^{(div)}$ be the solution of *M'* minimizing $Dist(P',M-\{P'\})$. Then, if $f(P) > f^{**}$, *M* is not updated. Otherwise, *P* replaces $P^{(div)}$. The resulting metaheuristic is summarized in Algorithm 2, relying on neighborhoods N_1 (modify *P*) and N_2 (modify *S*). The returned solution is *(P*,S*)* with an actual cost of *F**, which is the best solution visited in all the considered solution spaces.

Algorithm 2: General approach for (P)

Initialization: set $F^* = \infty$ and $N = UB(N)$.

While $N \geq LB(N)$, do

- generate an initial solution *P* with *N* orders as equi-spaced as possible;
- apply $TS_f(P,S)$ or $TS_f^M(P,S)$, and let $M = \{P^{(1)}, \ldots, P^{(m)}\}$ be the resulting set of local optima according to *f* (*m* = 1 if $TS_f(P,S)$ is used);
- for $i = 1$ to *m*, do: apply *DLS(S)* on $(P^{(i)},S^{(i)})$;
- set $(P,S) = arg \min_{i \in \{1,\ldots,m\}} F(P^{(i)},S^{(i)})$;
- if $F(P,S) < F^*$, set $(P^*,S^*) = (P,S)$, and $F^* = F(P,S)$;
- reduce *N* by one unit;

4.3. Results

The experiments were performed on a PC Pentium 4 (1.6 GHz/1 Go RAM). The parameters *Iter*, *m* and *m'* were respectively set to 1000, 10, 3. As the method has to plan the orders for a whole year, the computing time is not an issue (but an hour of computation is never exceeded). Each instance is characterized by its cost parameters (the fixed setup cost *A* per order, the inventory cost *h* per unit per period, the shortage cost *B* per missing unit). For each period *t* is known the minimum (resp. most likely and maximum) lead time a_t (resp. m_t and b_t). From these three values, discrete triangular distributions can be easily constructed. Two types T_1 and T_2 of instances were generated according to two sets of lead time distributions, with 24 instances per type (which differ according to *A*, *h* and *B*). Set T_1 is based on realistic data from the sawmill context, and is characterized by $a_t \in \{2, 5\}$, $m_t \in \{3, 7\}$ and $b_t \in \{6, 13\}$. Set T_2, which represents a form of sensitivity analysis (the variation of the lead times is larger), is characterized by $a_t \in \{1, 8\}$, $m_t \in \{2, 10\}$ and $b_t \in \{5, 16\}$. In Table 3 is provided a summary of the average percentage improvements (over a basic constructive heuristic based on an EOQ analysis) provided by the general presented approach relying on *DLS(P)* (where a descent local search is performed at step (S2) instead of tabu search), $TS_f(P,S)$ and $TS_f^M(P,S)$, respectively. The results are shown for three levels of *B* and for the two sets T_1 and T_2. Unsurprisingly, the potential benefit of the three methods augments as the seasonality is increased. One can observe that $TS_f^M(P,S)$ outperforms $TS_f(P,S)$, and both methods are better than *DLS(P)*.

Table 3. Compact comparative results

	Set T_1			Set T_2		
Method	Small *B*	Average *B*	Large *B*	Small *B*	Average *B*	Large *B*
$DLS_f(P,S)$	1.39	1.52	1.61	3.75	3.58	3.47
$TS_f(P,S)$	1.72	1.82	2.01	4.15	4.05	3.74
$TS_f^M(P,S)$	1.82	1.86	2.16	4.18	4.06	3.79

5. MNTS FOR DIMENSIONING ASSEMBLY LINES

In this section, a tabu search based on various moves with different amplitudes is first presented, and then adapted to the dimensioning of assembly lines.

5.1. Tabu search within a simulation context

Let $X = (X^1, X^2, \ldots, X^u)$ be a solution of problem (P) which consists in maximizing an objective function *f*. Each X^i is a vector of size *s(i)* and can be denoted $X^i = (x^i_1, x^i_2, \ldots, x^i_{s(i)})$, where the x^i_j's are real number. The following limitation constraint has to be satisfied for each *i*: $\sum_j x^i_j = c^i$. As random events can occur, it is assumed that *f* can only be evaluated with a simulation tool. In such a context, within a local search framework, it is straightforward to define a move in three steps:

(A) select a decision variable type *i*;

(B) augment (resp. reduce) an x^i_k by an amount of *w*;

(C) reduce (resp. augment) some other x^i_j's (with $j \neq k$) by a total amount of *w* (in order to satisfy the limitation constraint).

Within a tabu search framework, if a decision variable x^i_k is augmented (resp. reduced), it is then forbidden to reduce (resp. augment) it during tab^i (parameter) iterations. The three key issues are presented below, and the various neighborhood structures N_1, N_2, … result from these issues.

(I1) Which type of decision variable should be selected in (A)?

(I2) What is the amplitude *w* of the move in (B)?

(I3) How should the solution be adjusted in (C)?

According to issue (I1), *u* types of phase are used in the solution method: each phase of type *i* works on X^i without modifying the other X^l's ($l \neq i$). Each phase of type *i* can be performed during I^i_{max} (parameter) uses of the simulator. Working with phases (i.e., on one decision variable type at a time) allows to having a better control on the search.

To tackle issue (I2), and in contrast with classical tabu search approaches, the move amplitude *w* is dynamically updated during each phase of the search, within interval $[w^i_{min}, w^i_{max}]$ (parameters). Each phase starts with $w = w^i_{max}$, and anytime I^i (parameter) iterations without improvement of the best encountered solution *X** have been performed, *w* is reduced by δ^i (parameter), but never under w^i_{min}. If a move leads to a solution better than *X**, the process restarts with $w = w^i_{max}$. This strategy allows to progressively focus on a promising region of the solution space.

Issue (I3) depends on the two other issues: if x^i_k has been selected for a variation of *w*, the search process should focus on that decision and not modify as much the other decision variables (of the same type) in order to adjust the solution with respect to the associated constraint c^i. Thus, if x^i_k was augmented (resp. reduced) by *w*, the process should then equally reduce (resp. augment) the *s(i)* − 1 other variables (of the same type) by a total amount of *w*, which means that each decision variable is in average reduced (resp. augmented) by *[s(i)−1]/w*.

The resulting tabu search method is summarized in Algorithm 3, which returns the best encountered solution

X^* with value f^*. At each iteration, the neighbor solution can be the best among a set of N (parameter) candidate neighbor solutions.

Algorithm 3: Tabu search with various amplitudes

Initialization

- generate an initial solution $X = (X^1, X^2, \dots, X^u)$;
- initialize the best encountered solution: set $X^* = X$ and $f^* = f(X)$;
- set $i = 1$ and $w = w^i_{max}$;

While the involved simulation software has not been used Q (parameter) times, do

- generate a non-tabu neighbor solution X_i' of X_i by modifying a variable x^i_k of X_i by w;
- update the current solution: set $X_i = X_i'$;
- update the move amplitude w:

(a) if l^i iterations without improving X^* have been performed, set $w = w - \delta^i$;

(b) if $w < w^i_{min}$, set $w = w^i_{min}$;

(c) if $f(X) > f^*$, set $w = w^i_{max}$;

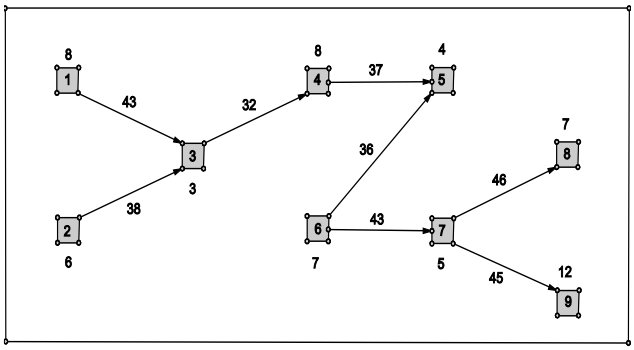
- update the best encountered solution: if $f(X) > f^*$, set $X^* = X$ and $f^* = f(X)$;
- update the tabu tenures: it is forbidden to modify x^i_k in the reverse way for tab^i iterations;
- next phase: if l^i_{max} runs of the simulator were performed, set $i = (i \bmod u) + 1$ and $w = w^i_{max}$;

5.2. Application to the dimensioning of assembly lines

The above tabu search is relevant for dimensioning assembly/disassembly production systems. By dimensioning, one can refer to maximizing the production rate of a machine without successors, with respect to limited resources (e.g., buffer capacity between the machines, total cycle time of the machines). Papers in the field are (Dolgui et al., 2007; Shi & Gershwin, 2009). As random failures might occur on the machines, the software *Arena* is appropriate to evaluate a solution.

Consider a production system with m machines and n buffer zones, modeled by a graph $G = (V,A)$ with vertex set V and arc set A. Vertex v represents machine v and there is an arc (v, v') from v to v' if a piece processed on machine v has then to be processed on machine v'. Moreover, each arc (v, v') also represents a *buffer* (i.e., a limited zone where are stored the pieces between the associated machines). Two types of decision variables (i.e., resource types) are considered: the designed cycle time t_v for machine v, and the buffer capacity $b_{vv'}$ allocated to arc (v, v'). In Figure 4, a solution for a production network. The cycle time associated with machine 1 is $t_1 = 8$, and the buffer capacity between machines 1 and 3 is $b_{13} = 43$. The considered limitations are 60 for the total cycle time (i.e., $\sum_{v \in V} t_v = 60$) and 320 for the buffer capacity (i.e, $\sum_{(v,v') \in A} b_{vv'} = 320$).

Figure 4. Graph representation of a production system with $m = 9$ and $n = 8$



The above presented MNTS approach showed a very good performance on such a production system (Zufferey & Cheikhrouhou, 2012). Indeed, it was tested on the production network associated with Figure 4, for which each machine has its own role: machines 1, 2, 4, 8 and 9 are classical processing machines, machines 3 and 5 are assembling machines, and machines 6 and 7 are disassembling machines. The objective consists in maximizing the production rate of machine 5. The breakdown probability is 5% (associated with each time step) and its length is generated with a uniform distribution in interval [100, 800]. MNTS was compared with a descent local search DLS (the same algorithm as MNTS, but without considering tabu tenures), and a classical tabu search TS for which at each iteration, a move consists in augmenting/reducing any decision variable by any possible amount (followed by the adjustment of the other variables of the same type in order to meet the upper bounds). In TS, a sample of all possible amplitudes is considered at each iteration to modify the decision variable. A pool of 50 initial solutions were generated, which have an average production rate of 1.31 (pieces/minute). DLS was able to reach an average production rate of 1.39, TS obtained 1.37, and MNTS reached 1.41. It was also observed that the quality of the resulting solution negligibly depends on the initial solution, which indicates that MNTS is a robust approach.

6. CONCLUSIONS

In contrast with most of the existing literature on tabu search where only a single type of move is used at each iteration, this paper proposes MNTS (*Multiple Neighborhoods in a Tabu Search*). With a unified terminology, the success of some existing solutions methods is presented, which can

be considered as belonging to the MNTS methodology. Problems in four domains are successively discussed: truck loading, job scheduling, inventory management, and assembly line dimensioning. It is important to mention that MNTS approaches were also successfully adapted in other fields (e.g., (Hertz et al., 2009; Hertz et al., 2005; Schindl & Zufferey, 2013).

The performance of a metaheuristic can be evaluated according to several criteria (Zufferey, 2012b): (1) *quality*: value of the obtained results; (2) *speed*: time needed to get good results; (3) *robustness*: sensitivity to variations in problem characteristics and data quality; (4) *ease of adaptation*: the ability to organize the method so that it can appropriately apply to different specific classes of problems; (5) *ability to take advantage of problem structure* (considering that efficiency often depends on making effective use of properties that differentiate a given class of problems from other classes). MNTS has a good behavior according to these criteria. Indeed, the following ingredients make it possible: (1) the best non-tabu move is performed at each iteration, which contributes to quality; (2) speed is mainly allowed by the incremental computation used to compute the value of any neighbor solution; (3) the use of various neighborhood structures enhances the robustness as well as the quality; (4) only one solution is handled during the search, which makes the method fairly easy to adapt and quick; (5) a dedicated (in contrast with generic) solution encoding allows to accounting for the structure of the problem, and it also contributes to quality.

REFERENCES

1. Cesaret, B., Oguz, C., & Salman, F. S. (2012.) A tabu search algorithm for order acceptance and scheduling. *Computers & Operations Research*, 39(6), pp. 1197-1205. Special Issue on Scheduling in Manufacturing Systems.
2. Dolgui, A., Eremeev, A., & Sigaev, V. (2007.) HBBA: Hybrid algorithm for buffer allocation in tandem production lines. *Journal of Intelligent Manufacturing*, 18(3), pp. 411-420.
3. Garey, M., & Johnson, D.S. (1979.) *Computer and Intractability: a Guide to the Theory of NP-Completeness*. San Francisco: Freeman.
4. Gendreau, M., & Potvin, J.-Y. (2010.) *Handbook of Metaheuristics*. International Series in Operations Research & Management Science, vol. 146. Springer.
5. Hendel, Y., & Sourd, F. (2007.) An improved earliness-tardiness timing algorithm. *Computers & Operations Research*, 34(10), pp. 2931-2938.
6. Hertz, A., Schindl, D., & Zufferey, N. (2005.) Lower bounding and tabu search procedures for the frequency assignment problem with polarization constraints. *4OR*, 3(2), pp. 139-161.
7. Hertz, A., Schindl, D., & Zufferey, N. (2009.) A Solution Method for a Car Fleet Management Problem with Maintenance Constraints. *Journal of Heuristics*, 15(5), pp. 425-450.
8. Hopper, E., & Turton, B.C.H. (2001.) An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, 128(1), pp. 34-57.
9. Lesh, N., Marks, J., McMahon, A., & Mitzenmacher, M. (2004.) Exhaustive approaches to 2D rectangular perfect packings. *Information Processing Letters*, 90(1), pp. 7-14.
10. Lodi, A., Martello, S., & Vigo, D. (1999.) Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems. *INFORMS Journal on Computing*, 11(4), pp. 345-357.
11. Lodi, A., Martello, S., & Monaci, M. (2002.) Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2), pp. 241-252.
12. Martello, S., & Vigo, D. (1998.) Exact Solution of the Two-Dimensional Finite Bin Packing Problem. *Management Science*, 44(3), pp. 388-399.
13. Martello, S., Monaci, M., & Vigo, D. (2003.). An Exact Approach to the Strip-Packing Problem. *INFORMS Journal on Computing*, 15, pp. 310-319.
14. Nguyen, A., & Brenaut, J.-Ph. (2009.) A truck loading algorithm for Renault's supply chain system. In: 23rd EURO Conference.
15. Ntene, N., & van Vuuren, J.H. (2009.) A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem. *Discrete Optimization*, 6(2), pp. 174-188.
16. Oguz, C., Salman, S. F., & Yalcin, Z. B. (2010.) Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics*, 125(1), pp. 200-211.
17. Pisinger, D., & Sigurd, M. (2005.) The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2(2), pp. 154-167.
18. Puchinger, J., & Raidl, G. R. (2007.) Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183(3), pp. 1304-1327.
19. Respen, J., & Zufferey, N. (2013.) A Renault Truck Loading Problem: from Benchmarking to Improvements. In: Proceedings of the 14th EU/ME Workshop (EU/ME 2013).
20. Riff, M.C., Bonnaire, X., & Neveu, B. (2009.) A revision of recent approaches for two-dimensional strip-packing problems. *Engineering Applications of Artificial Intelligence*, 22(45), pp. 823-827.
21. Schindl, D., & Zufferey, N. (2013.) Solution Methods for Fuel Supply of Trains. Information Systems and Operational Research, 51(1), pp. 22-29.
22. Shakhlevich, N. V., & Strusevich, V. A. (2006.) Single machine scheduling with controllable release and processing parameters. *Discrete Applied Mathematics*, 154(15), pp. 2178-2199.
23. Shi, C., & Gershwin, S. B. (2009.) An efficient buffer design algorithm for production line profit maximization. *International Journal of Production Economics*, 122(2), pp. 725-740.
24. Silver, E. A., & Zufferey, N. (2005.) Inventory control of raw materials under stochastic and seasonal lead times. *International Journal of Production Research*, 43, pp. 5161-5179.
25. Slotnick, S. A. (2011.) Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, 212(1), pp. 1-11.
26. Thevenin, S., Zufferey, N., & Widmer, M. (2012.) Tabu search to minimize regular objective functions for a single machine scheduling problem with rejected jobs, setups and time windows. In: Proceedings of the 9th International Conference on Modeling, Optimization & Simulation (MOSIM 2012).
27. Thevenin, S., Zufferey, N., & Widmer, M. (2013.) Tabu Search for a Single Machine Scheduling Problem with Discretely Controllable Release Dates. Pages 1590 – 1595 of: Proceedings of the 12th International Symposium on Operations Research in Slovenia (SOR 2013).
28. Valente, J. M. S., & Gonçalves, J. F. (2009.) A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties. *Computers & Operations Research*, 36, pp. 2707-2715.
29. Valente, J. M. S., Moreira, M., Singh, A., & Alves, R. (2011.) Genetic algorithms for single machine scheduling with quadratic earliness and tardiness costs. *The International Journal of Advanced Manufacturing Technology*, 54, pp. 251-265.
30. Yalçin, Z. B., Oguz, C., & Salman, S. F. (2007.) Order Acceptance and Scheduling Decisions in Make-to-Order Systems. Pages 80–87 of: Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Application (MISTA 2007).
31. Zorzini, M., Corti, D., & Pozzetti, A. (2008.) Due date (DD) quotation and capacity planning in make-to-order companies: Results from an empirical analysis. *International Journal of Production Economics*, 112(2), pp. 919-933.
32. Zufferey, N. (2012a.) Dynamic Tabu Search with Simulation for a Resource Allocation Problem within a Production Environment. In: Proceedings of 4th International Conference on Metaheuristics and Nature Inspired Computing (META 2012).
33. Zufferey, N. (2012b.) Metaheuristics: some Principles for an Efficient Design. *Computer Technology and Applications*, 3(6), pp. 446-462.
34. Zufferey, N. (2014.) Tabu Search with Diversity Control and Simulation for an Inventory Management Problem. In: Proceedings of the 5th International Conference on Metaheuristics and Nature Inspired Computing (META 2014).
35. Zufferey, N., & Cheikhrouhou, N. (2012.) Tabu search using variable amplitudes for dimensioning an assembly/disassembly production system. In: Proceedings of the 13th International Workshop on Project Management and Scheduling (PMS 2012).
36. Zufferey, N., & Vasquez, M. (2015.) A Generalized Consistent Neighborhood Search for Satellite Scheduling Problems. *RAIRO Operations Research*, 41(1), pp. 99-121.