Hakim Mitiche*, Dalila Boughaci and Maria Gini

Iterated Local Search for Time-extended Multi-robot Task Allocation with Spatio-temporal and Capacity Constraints

https://doi.org/10.1515/jisys-2018-0267
Received June 19, 2018; previously published online December 21, 2018.

Abstract: We propose a method for task allocation to multiple physical agents that works when tasks have temporal and spatial constraints and agents have different capacities. Assuming that the problem is over-constrained, we need to find allocations that maximize the number of tasks that can be done without violating any of the constraints. The contribution of this work is the study of a new multi-robot task allocation problem and the design and the experimental evaluation of our approach, an iterated local search that is suitable for time critical applications. We created test instances on which we experimentally show that our approach outperforms a state-of-the-art approach to a related problem. Our approach improves the baseline's score on average by 2.35% and up to 10.53%, while responding in times shorter than the baseline's, on average, 1.6 s and up to 5.5 s shorter. Furthermore, our approach is robust to run replication and is not very sensitive to parameters tuning.

Keywords: Multi-robot task allocation, iterated local search, routing.

2010 Mathematics Subject Classification: 68T20, 68T40, 68U01.

1 Introduction

Task allocation to physical agents addresses the problem of which agent should execute which tasks, so that the total cost is minimized and/or the total reward is maximized. Applications include robotic missions for surveillance or exploration (e.g. NASA's rovers sent to Mars); multiple vehicles routing and scheduling, such as for delivery of goods [20]; and an emergency response, which deals with important social issues, such as natural disaster mitigation [8] or law enforcement at the city scale [1].

Task allocation to multiple physical agents has been formulated in many ways, depending on the application, and solved using different methods. Our work is motivated by emergency response, but the formulation we give is relevant to other applications. Considering the intractability of the problem and the need of rapid response, we are interested in a fast and competitive heuristic for the problem. The main contributions of the paper are the following:

(i) a formalization of a practical multi-robot task allocation problem, where there is a limited number of agents that have heterogeneous capacity to execute tasks, and a relatively large number of tasks that are geographically scattered with deadlines;
(ii) the development of an efficient and effective metaheuristic, *iterated local search* (ILS) [12], based on a preliminary study of the baseline approach [21] when applied to our problem [15];
(iii) the generation of a test set and the experimental evaluation of the proposed method on it.

***Corresponding author: Hakim Mitiche,** Department of Computer Science, USTHB, BP 32 El-Alia, Babezzouar, Algiers 16111, Algeria, e-mail: hmitiche@usthb.dz. https://orcid.org/0000-0001-9571-7079
Dalila Boughaci: Department of Computer Science, USTHB, BP 32 El-Alia, Babezzouar, Algiers 16111, Algeria
Maria Gini: Computer Science and Engineering Department, University of Minnesota, 4-192 Keller Hall, 200 Union St SE, Minneapolis, MN 55455, USA

The task allocation problem we deal with falls into the *ST-SR-TA:TW* (Single Task robot, Single Robot task, Time-extended Assignment, Time Windows) *deterministic* category according to Nunes et al.'s taxonomy [17]. Our proposed approach is easy to implement; though not optimal, it computes good solutions fast, as shown in our experimental results. Hence, it is suitable for time critical applications, where solution quality is traded off for prompt response time. The experimental results show that our approach has limited sensitivity to parameter tuning and is robust to run replication.

The paper is organized as follows. In Section 2, we briefly review the literature on task allocation to multiple physical agents. We define our task allocation problem in Section 3. In Section 4, we detail a fast and effective metaheuristic approach that we propose for the problem. Section 5 describes the experimental setup, the test instances, and the experimental results we obtained. Section 6 concludes and suggests future research directions.

2 Related Work

Task allocation to multiple physical agents is mainly studied by the artificial intelligence and robotics communities, where it is usually referred to as *multi-robot task allocation* (MRTA). Major applications of MRTA are, but not limited to, emergency response, such as urban search and rescue; exploration of hostile environments, such as the Mars planet by NASA's twin geologist rovers; and surveillance of hazardous environments, such as surveillance of an enemy's border territory with unmanned air vehicles. Typically, the objective in a MRTA problem is to minimize the total distance traveled by robots to accomplish the tasks (because of limited battery power) or to minimize the mission's time (if it maximizes the reward). The allocation of tasks to robots is constrained spatially and temporally, as the robots need to travel to tasks which are only available during a certain, potentially short, period of time. Advanced applications may require further constraints, such as precedence among tasks or synchronization of their execution [17].

According to the taxonomy of Gerkey and Matarić [5], MRTA problems can be broadly categorized according to three axes: (i) how many tasks can a robot perform simultaneously, (ii) does a task require (or permit) multiple robots working on it simultaneously, and (iii) can all tasks be allocated upfront or as they arrive.

Nunes et al. [17] recently extended the taxonomy third axis by distinguishing between allocation models where tasks have time windows and allocation models where tasks have precedence or synchronization constraints. Their taxonomy is further narrowed by subcategorizing task allocation according to hard vs. soft temporal constraints, deterministic vs. stochastic task arrival, task reward, robot travel time, etc.

With respect to decision making, we distinguish *centralized* approaches (e.g. [9]) from *decentralized* approaches (e.g. [18, 19]). Centralized approaches assume a (single) central decision maker that assigns tasks to agents, while in decentralized approaches there are many independent decision makers, usually with limited awareness of tasks and peer agents. In swarm intelligence, (e.g. [3, 4]), task allocation can emerge from simple and independent behaviors of single robots. When using decentralized constraint optimization, tasks are allocated through a distributed algorithm run by multiple robots that communicate with each other [19]. Decentralized task allocation is useful in dynamic environments, to provide fault tolerance and adaptability, or when communication is limited. Centralized approaches have a single point of failure; however, they are more effective in observable environments and more suitable when optimality is desirable.

Lagoudakis et al. [11] provide auction-based multi-robot algorithms for robots to visit tasks. The algorithms have performance guarantees, but they do not include temporal constraints. Melvin et al. [14] developed auction-based methods that run fast and perform well on MRTA problems with reward for task completion. Their problem does not require the visit of all targets, and the approach is restricted to tasks with disjoint time windows. Task allocation in emergency response is modeled as an extended generalized assignment problem by Scerri et al. [19]. The agents diverge in efficiency and availability of resources. The tasks require different amounts of resources and may have interdependencies during execution. The problem formulation captures many aspects of task allocation to robots; however, it does not consider spatial or temporal constraints on tasks, and the agent capacity is considered as a reward rather than a constraint. Ramchurn et al. [18] approached the emergency response issue with teams of agents, of different joint capabilities, which band

(and disband) to work simultaneously together on tasks that have workload, time, and location constraints. While such a formulation may be effective, it comes at the cost of increased computational complexity.

Other formulations relevant to the problem we address are routing and scheduling problems, such as the vehicle routing problem with time windows [20], the team orienteering problem with time windows (TOPTW) [2], and the multiple repairman problem with time windows [13].

These problems fall within the time-extended task allocation class (see [17] for details on further problems that fall in this class).

3 Problem Statement

We assume a number of cooperative physical agents that have to travel to execute dispersed tasks. The agents and the tasks are known upfront and do not change during the allocation. We assume the ST-SR-TA allocation model [5], where an agent does at most one task at a time, each task requires one agent only, and task assignments are planned over time spans. Our MRTA model is deterministic with hard temporal constraints [17]. We assume a single type of task and a single type of agent.

3.1 Basic Definitions

Let $A = \{a_1, \ldots, a_n\}$ be the set of agents. Each agent $a \in A$ has an initial location $l_a^0 \in L_A$ at time $t = 0$, where L_A is the set of initial locations of the agents on the Euclidean plane. Let $K = \{k_1, \ldots, k_m\}$ be the set of tasks. A task $k \in K$ has a fixed location $l_k \in L_K$, a deadline d_k by which it should be finished, and a workload w_k which is an estimate of the amount of work necessary to complete k. Each agent a has a capacity p_a, which is the number of units of work that a can perform in one unit of time. Differences in capacity among agents may be due to skill, experience, or specific tools. We assume that time is discrete and agents can travel and do tasks in measurable units of time (e.g., seconds or minutes) starting from a reference time $t = 0$. We define $\rho : (L_A \cup L_K) \times L_K \to [0 \ldots \infty]$ to be the agent travel time function. For simplicity, we assume that paths are free of blockages and traffic jams, the agents have the same constant velocity and will always take the shortest path between two consecutive tasks.

3.2 Problem Constraints

The tasks have to be allocated without violating any temporal or capacity constraint. Temporal constraints restrict the time by which tasks have to be completed, while capacity constraints affect the number of tasks that can be processed and their processing time. An agent-task assignment that respects the capacity and temporal constraints is said to be *feasible*. Formally, the assignment of agent a to task k at time t is feasible, if and only if

$$t + \rho(l_a^t, l_k) + \lceil w_k / p_a \rceil \leq d_k \tag{1}$$

where l_a^t is the location of agent a at time t. The constraint above ensures that an agent will only travel to a task it can reach and finish before the deadline, given the agent's location, its capacity, and the task's workload.

3.3 Search Space

We denote by $\tau_{t_1,t_2}^{a \to k}$ the assignment of agent a to task k, during the period of time $[t_1, t_2]$. We define F, the set of all feasible assignments, as

$$F = \{\tau_{t_1,t_2}^{a \to k} \mid t_1 = t + \rho(l_a^t, l_k),\ t_2 = t_1 + \lceil w_k / p_a \rceil, \\ t_2 \leq d_k\}_{a \in A, k \in K, t \in \{0,\ldots,d_k\}} \tag{2}$$

Feasible assignments of agent a to task k are within the time window $[0, d_k]$ (Eq. (2)). An assignment start time, t_1, corresponds to the time of arrival of a to task k. It depends on a's location at time t, which is 0 at the beginning, or the time by which a finishes the previous assignment. The assignment completion time, t_2, should meet k's deadline (Eq. (2)). A candidate solution, denoted by Γ, is a subset of F that we define as

$$\Gamma = \{\{\tau_{t',t''}^{a \to k}\} \subseteq F | \forall \tau_{t_1,t_2}^{a_1 \to k_1}, \tau_{t_1',t_2'}^{a_2 \to k_2} \in \Gamma : k_1 \neq k_2,$$
$$a_1 = a_2 \Rightarrow \{t_1 \dots t_2\} \cap \{t_1' \dots t_2'\} = \emptyset\} \tag{3}$$

The first condition (Eq. (3)) ensures that every task is at most assigned to one agent and once. The second condition guarantees that each agent is assigned to no more than one task simultaneously. The search space, $\mathbf{\Gamma}$, is defined as: $\mathbf{\Gamma} = \cup\Gamma$.

3.4 Objective Function

The goal of our task allocation problem is to maximize the number of tasks that are completed (by their deadlines). It can be expressed as follows:

$$\arg\max_{\Gamma \in \mathbf{\Gamma}} \sum_{k \in K} \delta(k, \Gamma) \tag{4}$$

where $\delta(.)$ is the task selection function defined over the set of tasks and candidate solutions. For $k \in K$ and $\Gamma \in \mathbf{\Gamma}$, we have

$$\delta(k, \Gamma) = \begin{cases} 1 & \text{if } \exists a \in A, t, t' : \tau_{t,t'}^{a \to k} \in \Gamma \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The function $\delta(.)$ returns 1 if and only if task k is present in one of the assignments of the candidate solution (Γ).

To the best of our knowledge, the problem we have defined has not been studied. It bears similarity to TOPTW [2] and particularly to the MRTA problem in [18]; however, we assume single-robot tasks and capacity for agents rather than coalitions. We refer to our problem as *multi-robot task allocation with temporal and capacity constraints* (MRTA/TC).

4 A Metaheuristic Approach

We found that the MRTA/TC problem is NP-hard and relates to the TOPTW. To tackle the problem complexity, we propose an anytime approach so that it returns quickly a (suboptimal) solution, yet can find a better solution if run longer. The approach is an *enhanced* ILS that builds upon ILS [21], based on the preliminary study we did in [15]. The metaheuristic runs in two phases: *construction*, which generates local optima, and *perturbation*, which tries to escape them to find a global optimum or at least a local optimum of better quality. We will designate an agent's assignment by *visit* and an agent's set of assignments by *tour*.

4.1 Construction Step: Insertion-Based Heuristic

This phase builds a candidate solution, from some partial solution, based on a heuristic that inserts visits into all tours simultaneously. Let s_i, f_i, and T_i respectively denote the start time, the finish time, and the service time of visit i. Based of Eq. (1), the service time of visit i can be computed as $T_i = \lceil w_{a_i}/p_{a_i} \rceil$, where a_i is the agent corresponding to the tour in which visit i is considered for insertion. We denote – with some abuse of notation – by $\rho(h, i)$, the travel time between locations of visits h and i and by $\rho(-1, h)$ the travel time between

the tour start location and (h), the location of the first visit in the tour. The insertion of visit i between h and j delays j and the visits that follow j, in the tour, by $shift_i$ as follows:

$$shift_i = \rho(h, i) + T_i + \rho(i, j) - \rho(h, j). \tag{6}$$

Given that the tasks have deadlines, before considering to delay a visit, we ensure that the visit remains feasible after the insertion under consideration. For that, we compute $maxshift_j$, the maximum allowed delay of visit j, as follows:

$$maxshift_j = \max\left\{0, \begin{cases} d_j - (s_j + T_j) & \text{if } j \text{ is the last} \\ \min\{d_j - (s_j + T_j), maxshift_{j+1}\} & \text{otherwise} \end{cases}\right\} \tag{7}$$

where d_j is the deadline of visit j (with some abuse of notation) and $maxshift_{j+1}$ is the maximum delay that can be applied on the visit that follows j. A visit can be delayed as much as its task's deadline permits, when it is the last in the tour. Otherwise, the visit can be delayed no later than its task's deadline and no more than the maximum shift of the visit that follows (Eq. (7)).

The insertion of visit i between h and j is feasible if and only if i is feasible (Eq. (1)) and the resulting delay of j and, consequently, the delays of the visits that follow j are less than or equal to the maximum allowable:

$$shift_i \leq maxshift_j \tag{8}$$

A visit may be inserted in many tours and in many positions in a tour. The desirability of the insertion of visit i, noted as $desirability_i$, is inversely proportional to the resulting delay (Eq. (9)). Let F denote – with a slight change of notation – the set of possible visits to currently not allocated tasks, $K_{pending}$. Let F_k denote the set of feasible visits to task $k \in K_{pending}$. Then F^* denotes the set composed of the most desirable insertion for every pending task ($F^* \subseteq F$). We look for i_k^*, the most desirable visit to task k in the current tours (Eq. (10)). Then we select for insertion i^*, the visit with the highest desirability among the best visits to the pending tasks (Eq. (11)).

$$desirability_i = 1/shift_i \tag{9}$$

$$i_k^* = \arg\max_{i \in F_k}\{desirability_i\} \tag{10}$$

$$i^* = \arg\max_{i \in F^*}\{desirability_i\} \tag{11}$$

Algorithm 1: Construction Step.

Input: partial solution S, set of tasks $K_{pending}$
Output: new solution S'
$S' \leftarrow S$
while $K_{pending} \neq \emptyset$ **do**
  forall $k \in K_{pending}$ **do**
    Compute all possible visits to task k (Eqs. (1) and (8))
    Find the best visit to task k (Eq. (10))
  Find the best visit to insert, i^* (Eq. (11))
  if $i^* = null$ **then Return** S'
  else Insert i^* into S' and update $K_{pending}$
  Compute the start time s_{i^*} and the finish time f_{i^*}
  forall *the visits j after* i^* **do** Update s_j, f_j and $maxshift_j$ (Eqs. (12) and (7))
  Compute $maxshift_{i^*}$
  forall *the visits h before* i^* **do** Update $maxshift_h$

After an insertion, we update the maximum shift for all the visits in the tour. For the visits that come after the insertion, we further update their start times and finish times, by adding the insertion delay (Eq. (12)).

$$s_j = s_j + shift_i \qquad f_j = f_j + shift_i \tag{12}$$

The construction step iterates until all tasks are visited, or no further insertion is possible (Algorithm 1). Each time, Algorithm 1 inserts the best visit among the possible visits to the pending tasks. Then, it updates the set of pending tasks and the tour where the insertion took place by recomputing the visits' start times, finish times, and maximum shifts. The maximum shifts are updated in the inverse order of visits, as required by Eq. (7).

4.2 Perturbation Step: Alteration of the Current Solution

This phase modifies the current solution so that the construction phase can find another solution. The *perturbation strength* and the *perturbation start* define the modification to make. The adjustment of these perturbation parameters affects the search tendency (exploration or exploitation) [12]. To perturb the solution, we remove from each tour r sequential visits, starting from some visit x, in a circular way. That is, we remove visits from index x to $x + r - 1$. When we reach the end of the tour, we continue from the beginning (i.e. from 0 to $y = (x + r)\ \%\ tl - 1$, where % is the remainder of division operator and tl is the length of the tour). The perturbation strength is then nr. Removing visits requires a backward shift, $backshift_{x,r}$, of the visits that follow the removal, unless the visits removed are at the tour end:

$$backshift_{x,r} = \begin{cases} 0 & \text{if } x + r = tl \\ \sum_{i=0}^{y} [\rho(i-1, i) + T_i] + \rho(y, y+1) - \rho(-1, y+1) & \text{if } x + r > tl \\ \sum_{i=x}^{x+r-1} [T_i + \rho(i, i+1)] + \rho(x-1, x) - \rho(x-1, x+r) & \text{otherwise} \end{cases} \tag{13}$$

When visits are removed from the beginning or the middle of the tour, the visits that follow are shifted toward the beginning. The start and finish times of the shifted visits are updated as follows (where $i \in \{x + r \ldots tl - 1\}$ when $x + r < tl$ and $i \in \{y + 1 \ldots x - 1\}$ when $x + r > tl$),

$$s_i = s_i - backshift_{x,r} \tag{14}$$

$$f_i = f_i - backshift_{x,r} \tag{15}$$

The removal of visits requires to update the maximum shifts of the visits that remain. When visits are removed from the middle of the tour (i.e. when $x + r < tl$), the maximum shifts of the visits i that follow the removal ($i \in \{x + r, \ldots, tl - 1\}$) are computed as follows:

$$maxshift_i = maxshift_i + backshift_{x,r} \tag{16}$$

Algorithm 2: Perturbation Step.

Input: current solution S, free parameter *perturbationStrengthCoef*
Output: partial solution S'

1 $r \leftarrow (score(S) \times perturbationStrengthCoef)/n$
2 $S' \leftarrow S$
3 **forall** $tour \in S'$ **do**
4 $x \leftarrow$ random number $\in [0, length(tour))$
5 Remove the r first visits starting at index x in *tour* and update $K_{pending}$
6 Compute $backshift_{x,r}$ in *tour* (Eq. (13))
7 **forall** *the visits* $j \geq x + r$ **do** Update s_j, f_j and $maxshift_j$ (Eqs. (14)–(16))
8 **forall** the visits $i < x$ **do** Update $maxshift_i$ (Eq. (7))

Algorithm 3: Enhanced Iterated Local Search.

Input: *maxIter*, *perturbationStrengthCoef*
Output: best-found solution S^*
1 $S' \leftarrow \emptyset$
2 $S^* \leftarrow S'$
3 $iter \leftarrow 1$
4 **while** $iter \leq maxIter$ **do**
5 $S \leftarrow$ Construction(S') (Algorithm 1)
6 **if** $score(S) > score(S^*)$ **then** $S^* \leftarrow S$
7 $S' \leftarrow$ Perturbation(S) (Algorithm 2)

Our perturbation (Algorithm 2) removes from every tour a number of visits that is proportional to the number of visited tasks, *score*(S), the perturbation strength coefficient, *perturbationStrengthCoef*, and the number of agents n. Thus, the perturbation strength is evenly divided among the tours and is adapted to the search progression and the difficulty of the problem instance (number of agents, task/agent locations, and deadlines/capacities distribution). Namely, the perturbation strength at the tour's level increases as more tasks get visited (hence, the tours get longer). The perturbation start index, x, is selected randomly (Algorithm 2). By doing so, the perturbation is very likely to output a different solution S', if it happens to have again solution S as input. Consequently, our ILS should avoid cycles and better explore the search space compared to ILS [21].

4.3 The Metaheuristic: Enhanced ILS

Enhanced ILS (Algorithm 3) is configured with two parameters: (1) the maximum number of iterations (*maxIter*) which, contrary to *maxTrials* of ILS [21], enables the control of the runtime and (2) a coefficient that controls the perturbation strength at the tour level (*perturbationStrengthCoef*). Enhanced ILS iterates until all tasks are visited or the maximum number of iterations is reached. Each time, it constructs a new solution, S, from a partial solution, S' (initially empty), that is based on the current solution, S. The best solution found, S*, is updated with new solution S if necessary. Then, S is altered so that the next iteration may escape the local optimum towards a possibly better one.

5 Experimental Evaluation

We experimentally evaluated our approach to the MRTA/TC problem on instances we generated based on instances of the related problem (TOPTW) and against a competitive approach to the related problem.

The comparison of the state-of-the-art approaches to TOPTW was conducted by Hu and Lim [7] and recently by Gunawan et al. [6]. The authors adapted the computation times of the competitors methods according to their CPU speeds. The comparisons were carried out on two TOPTW benchmarks with up to four tours. We only consider the benchmark based on Solomon's instances as our test set. We compare the performance averaged per group of instances similarly to [6, 7]. The iterative three-component heuristic [7] tends to get the lowest gap to the best-known solutions; however, it has a high computation time. The hybridization of Simulated Annealing with ILS [6] is comparable to the iterative three-component heuristic but only for long computation times. ILS [21] is appropriate for real time, since it can in the order of seconds return solutions with a low score gap to the best-known – the average score gap was only 1.76% and 2.34%, respectively, for the second group and the first group of Solomon's instances (Gunawan et al. [6], though more recently, report ILS score gaps that are even lower than Hu and Lim [7]), while the average response time is *tens* to *hundreds* times faster than the ant colony system [16] and the iterative three-component heuristic. The hybrid metaheuristic of Labadi et al. [10] is a *few* times slower than ILS but has a slightly better average score gap (about 1.2% better on average in each instance group [7]).

The metaheuristics of [10] and [21] are suitable baselines for time critical applications, since both have prompt response times and low score gaps compared to the best-known. Nonetheless, ILS [21] has the best response time and will be the baseline for evaluating our approach.

5.1 Experimental Instances and Setup

We generated two groups of problem instances. In group 1, instances admit solutions with tours of small to moderate length; whereas in group 2, instances can have solutions with tours that are relatively long. There is a fixed number of tasks, 100, while the number of agents varies as follows: 2, 3, 5, and 7. The map is a 100×100 grid. The agents' initial locations and the tasks' locations are drawn uniformly at random. The travel velocity is commonly set at a unitary value. Agent a's capacity $p_a \in \{1, 2\}$, such that about one third of the agents are twice more efficient at work than the rest.

The task workload is set at 20. Task k's deadline, d_k, is normally drawn from $[d_k^{low}, d^{high}]$ for 25%, 50%, 75%, or 100% of the tasks. The remaining tasks take d^{high} (the slackest deadline). The tightest deadline, d_k^{low}, is set such that task k would be feasible as first task for any agent. The value d^{high} is set such that, assuming 10 agents and a travel distance between any pair of tasks of at most 25 units (1/4 of the longest distance), if d^{high} was the deadline for all tasks, they could all be visited. We experimentally found that this setting of d^{high} enables long tours, whereas setting it at half its original value enables relatively short tours.

By combining the number of agents with the percentage of normally sampled deadlines, we get 16 different *problem configurations* per group. We sample three instances per problem (configuration) to get 96 instances (the test instances are available on the Web at http://tinyurl.com/taptc15in). An instance named r23*a*501, for example, is in group 2, has 3/4 tasks with normally distributed deadlines, five agents, and is the second instance of the problem configuration.

We coded the enhanced ILS and the baseline in Java. We ran the experiments on a laptop equipped with an Intel(R) Core(TM) i7 CPU clocked at 2.20 GHz, 8 GB of volatile memory, and running under Linux Ubuntu 14.04. We replicated enhanced ILS run on a problem instance 10 times.

5.2 Parameter Tuning

We manually tuned the algorithms in a trial-and-error fashion. We stopped tuning when the quality of the solution no longer improved without significantly increasing the computation cost. We found the best parameters setting for ILS (*maxTrials*, *maxPerturbationStrengthCoef*) to be (600, 1/2) and (600, 2/5), respectively, for group 1's and group 2's instances. Based on the run replication with the median score, we found the best parameters setting for enhanced ILS (*maxIter*, *perturbationStrengthCoef*) to be (3000, 1/3) and (5000, 1/8), respectively, for group 1's and group 2's instances. We found marginal improvements beyond 15 s of execution.

5.3 Results

Table 1 summarizes the results of the experimental comparison of enhanced ILS and ILS. The score gap is reported in percentage (%) and the computation time gap in milliseconds (ms) per problem (Eq. (17)).

$$ScoreGap = (score_{enhILS} - score_{ILS}) \div score_{ILS} \times 100\%$$

$$TimeGap = \begin{cases} time_{ILS} - time_{enhILS} & \text{if } ScoreGap = 0 \\ (runtime_{runner-up} - time_{winner}) & \text{otherwise} \end{cases} \tag{17}$$

where $score_{enhILS}$ and $score_{ILS}$ are the solution qualities of the proposed metaheuristic and the baseline metaheuristic, respectively. The computation time that is *actually* necessary to find the final solution is noted $time_{ILS}$ and $time_{enhILS}$. As to $runtime_{runner-up}$, it is the runtime of the metaheuristic that underperformed, while $time_{winner}$ is the computation time of the metaheuristic that performed best on the problem at hand.

Table 1: Performance of Enhanced ILS vs. Baseline ILS, Average Per Problem.

Problem	Score gap (%)			Time gap (ms)		
	Median	Best	Worst	Median	Best	Worst
r11a2	0.0	0.0	0.0	9	15	15
r12a2	1.38	2.76	0.0	1558	1464	4
r13a2	1.43	1.43	1.43	1174	1366	1364
r14a2	5.18	5.18	3.63	1080	1141	1202
r11a3	3.1	3.1	3.1	2760	2783	2717
r12a3	4.23	4.23	3.26	2480	2418	2497
r13a3	5.21	6.51	1.95	1418	1607	2368
r14a3	3.8	3.8	1.52	2133	2122	2083
r11a5	2.68	3.25	1.34	4688	3532	4917
r12a5	0.0	1.9	−0.76	−313	3883	5389
r13a5	2.56	2.56	0.0	3358	4143	259
r14a5	3.31	4.02	1.65	2097	2968	3969
r11a7	1.3	1.3	−0.52	3443	3878	11,234
r12a7	0.0	0.95	−0.41	−207	5221	8287
r13a7	0.97	1.94	0.55	4973	3401	2927
r14a7	2.64	2.64	1.09	5547	7041	6591
Average	2.36	2.85	1.11	2262	2936	2378
Minimum	0.0	0.0	0.0	−313	15	4
Maximum	5.21	6.51	3.63	5547	7041	6591
r21a2	1.56	1.56	−0.67	1303	1901	3815
r22a2	4.02	4.73	1.65	1290	1539	2508
r23a2	1.74	1.74	0.99	1428	1315	2232
r24a2	7.21	7.21	5.11	1143	1024	1330
r21a3	1.11	1.58	0.0	−215	1678	125
r22a3	2.59	3.73	0.97	14	−657	1892
r23a3	1.72	2.4	0.69	2183	1751	2890
r24a3	0.8	1.99	−1.19	2526	3236	2893
r21a5	2.4	2.4	1.67	−102	−699	1294
r22a5	2.84	3.6	2.18	879	−975	552
r23a5	3.11	4.3	2.39	1930	2118	2996
r24a5	2.23	3.15	0.52	1544	1437	2972
r21a7	0.0	0.0	0.0	15	15	15
r22a7	0.0	0.0	0.0	17	10	18
r23a7	0.0	0.0	−0.3	−12	−476	2417
r24a7	2.49	3.12	2.08	487	1013	1577
Average	2.11	2.59	1.01	902	889	1569
Minimum	0.0	0.0	−0.67	−215	−975	15
Maximum	7.21	7.21	5.11	2526	3236	2996

Then the time gap, *TimeGap*, measures how much extra time the runner-up approach was given to catch up with the superior approach, if any (Eq. (17)).

In Table 1, column 1 lists the problems, and columns 2–4 report the score gaps obtained from the median, best, and worst run replications of enhanced ILS, respectively. Columns 5 through 7, respectively, report the time gap for enhanced ILS run replications with the median, best, and worst score gaps. The results are averaged over the problems instances. Positive score gaps account for our approach having a performance better than the baseline's. When the time gaps are positive as well, our approach could find better solutions in shorter response times (Eq. (17)).

Compared to ILS, enhanced ILS generally finds solutions of better quality, within shorter times (Table 1). Even in the worst run replication, enhanced ILS is generally better (it is *at least* as effective as ILS on 13 out of the 16 problems, in each group). In the median case, enhanced ILS is on average 2.24% and up to 7.21% more effective in allocating tasks than ILS. When we consider all run replications, enhanced ILS almost always improves the solution quality (it did not only on two problems in group 2). The total average score gap is slightly improved in the best run replication; nonetheless, the score is improved on half of the problems.

Enhanced ILS tends to have a response time that is shorter than ILS's (Table 1). In the median run replication, ILS is given, on average, 1 s and up to 5.5 s to catch up with enhanced ILS.

Adapting the perturbation strength to the current solution and problem instance, and including randomness into the perturbation phase, is better than doing cyclic incremental deterministic perturbations (the case of ILS). We think, likewise, that search recycling is very unlikely. Interestingly, enhanced ILS's

Table 2: Detailed Performance of Enhanced ILS (Median Run) Versus ILS.

Instance	Score	Gap (%)	Time (ms)		Instance	Score	Gap (%)	Time (ms)	
			Computation	Run				Computation	Run
r11a200	23.0	0.0	42	1732	r21a200	46.0	0.0	2764	5019
r11a201	23.0	0.0	13	1581	r21a201	46.0	**2.22**	**9**	**4117**
r11a202	23.0	0.0	2	1832	r21a202	45.0	**2.27**	**927**	**3854**
r11a300	31.0	**3.33**	**119**	**2481**	r21a300	59.0	0.0	170	7068
r11a301	38.0	**2.7**	**185**	**4243**	r21a301	72.0	**1.41**	**1762**	**7638**
r11a302	31.0	**3.33**	**4**	**2594**	r21a302	61.0	**1.67**	**103**	**6865**
r11a500	52.0	**1.96**	**1210**	**5962**	r21a500	97.0	**2.11**	**918**	**7448**
r11a501	52.0	**1.96**	**701**	**6297**	r21a501	97.0	**4.3**	**1993**	**8056**
r11a502	57.0	**3.64**	**1200**	**9261**	r21a502	100.0	**1.01**	**214**	**441**
r11a700	78.0	**2.63**	**8069**	**13,987**	r21a700	100.0	0.0	14	15
r11a701	79.0	**1.28**	**3952**	**14,358**	r21a701	100.0	0.0	15	22
r11a702	76.0	0.0	643	10,956	r21a702	100.0	0.0	15	17
r12a200	22.0	**4.76**	**10**	**1114**	r22a200	44.0	**7.32**	**615**	**3172**
r12a201	21.0	0.0	525	1077	r22a201	43.0	0.0	2026	3298
r12a202	23.0	0.0	9	1954	r22a202	45.0	**4.65**	**119**	**3392**
r12a300	30.0	**3.45**	**22**	**2129**	r22a300	60.0	**5.26**	**281**	**5428**
r12a301	36.0	**2.86**	**16**	**3790**	r22a301	71.0	**2.9**	**1558**	**6769**
r12a302	30.0	**7.14**	**1513**	**2177**	r22a302	59.0	0.0	551	5695
r12a500	50.0	−1.96	532	5683	r22a500	93.0	**4.49**	**596**	**6889**
r12a501	52.0	**1.96**	**764**	**5244**	r22a501	94.0	**3.3**	**634**	**7439**
r12a502	56.0	**1.82**	**3270**	**8873**	r22a502	96.0	**1.05**	**1801**	**7904**
r12a700	73.0	**2.82**	**4845**	**9382**	r22a700	100.0	0.0	15	16
r12a701	76.0	0.0	1090	9440	r22a701	100.0	0.0	40	20
r12a702	73.0	−1.35	2809	9412	r22a702	100.0	0.0	15	17
r13a200	22.0	0.0	2	934	r23a200	41.0	0.0	1614	2767
r13a201	22.0	**4.76**	**7**	**1063**	r23a201	41.0	**2.5**	**642**	**2774**
r13a202	20.0	0.0	261	851	r23a202	41.0	**2.5**	**770**	**2738**
r13a300	31.0	**3.33**	**76**	**1815**	r23a300	55.0	**1.85**	**1708**	**3859**
r13a301	35.0	**6.06**	**240**	**2836**	r23a301	67.0	**1.52**	**2417**	**5254**
r13a302	31.0	**6.9**	**430**	**1871**	r23a302	56.0	**1.82**	**1019**	**3862**
r13a500	50.0	**4.17**	**464**	**4621**	r23a500	81.0	**3.85**	**3570**	**6004**
r13a501	52.0	**4.0**	**1397**	**5472**	r23a501	89.0	**4.71**	**1082**	**6227**
r13a502	54.0	**1.89**	**1056**	**5433**	r23a502	89.0	**1.14**	**2973**	**5886**
r13a700	73.0	−1.35	1759	8799	r23a700	100.0	0.0	23	24
r13a701	75.0	**2.74**	**7782**	**8854**	r23a701	100.0	0.0	1757	61
r13a702	71.0	**1.43**	**6043**	**8527**	r23a702	100.0	0.0	15	17
r14a200	21.0	**10.53**	**397**	**864**	r24a200	36.0	**9.09**	**149**	**1702**
r14a201	20.0	0.0	4	755	r24a201	35.0	**2.94**	**123**	**1609**
r14a202	20.0	**5.26**	**10**	**1064**	r24a202	36.0	**9.09**	**233**	**1754**
r14a300	25.0	**8.7**	**32**	**1998**	r24a300	46.0	0.0	2111	3036
r14a301	31.0	**3.33**	**73**	**3036**	r24a301	56.0	0.0	3255	3982
r14a302	26.0	0.0	22	1928	r24a302	50.0	**2.04**	**7**	**3309**
r14a500	42.0	**2.44**	**85**	**3879**	r24a500	76.0	0.0	1453	6973
r14a501	43.0	**2.38**	**2069**	**5034**	r24a501	77.0	**4.05**	**1478**	**6595**
r14a502	46.0	**4.55**	**698**	**5761**	r24a502	81.0	**2.53**	**1317**	**6322**
r14a700	67.0	**3.08**	**829**	**8711**	r24a700	99.0	**2.06**	**234**	**8220**
r14a701	64.0	**1.59**	**3477**	**9961**	r24a701	98.0	**3.16**	**631**	**7608**
r14a702	67.0	**3.08**	**5681**	**9556**	r24a702	99.0	**2.06**	**2151**	**8337**
Average		2.52	1342	4983	Average		2.1	998	4157
Maximum		10.53	8069	14,358	Maximum		9.09	3570	8337
Minimum		−1.96	2	755	Minimum		0.0	7	15

perturbation proved particularly helpful on instances with short tours. Enhanced ILS achieved at least 2.5% score improvement in nine problems of group 1, compared to five problems of group 2 (Table 1).

Table 2 details the median performance results of enhanced ILS. The columns from left to right, respectively, lists the problem instances names, the solution quality, the score gap (enhanced ILS vs. ILS), enhanced ILS's computation time, and runtime. Strictly positive score gap values, which account for the superiority of our metaheuristic, are noted in bold. Enhanced ILS finds solutions of better quality than ILS solutions, about 68% of the time (or on 65 out of the 96 instances). Enhanced ILS fell shorter than the baseline only on three instances: r12a500, r12a702, and r13a700 (Table 2). The score improvement is up to about 10%. On average, enhanced ILS performs slightly better on instances with tight deadlines (group 1). Since both ILS algorithms find the optimal solution of at least 10 instances of group 2 (when 100 tasks are allocated), such instances may be easy; thus, we cannot compare the approaches on them. Enhanced ILS runs, on average, in less than 5 s and no more than 15 s. However, the response times are lesser than the runtimes (Table 2).

To further evaluate our approach, we plot the score gap evolution over time. For that, we consider enhanced ILS's median run and ILS run on the 96 problem instances. Figure 1 shows four representative patterns of score gap evolution over time – we omit a fifth pattern, with 26% of occurrences, since therein the score gap is null and does not evolve over time. Enhanced ILS is superior to ILS all over the runtime more than half of the time (Figure 1A). Specifically, in 56.3% of the instances, not only does enhanced ILS return a better solution, but it can also do that any time. Less frequently (11.5% of the time), enhanced ILS achieves a better score by the end, while being generally superior during the runtime (Figure 1B). Enhanced ILS ties with ILS by the end of the run in 29% cases (Figure 1C). Interestingly, when enhanced ILS underachieves by the end (3% of the cases), it is still competitive with ILS along the runtime (Figure 1D). We can conclude that enhanced ILS is likely to achieve a score better than ILS's, even if interrupted at different cutoff times.

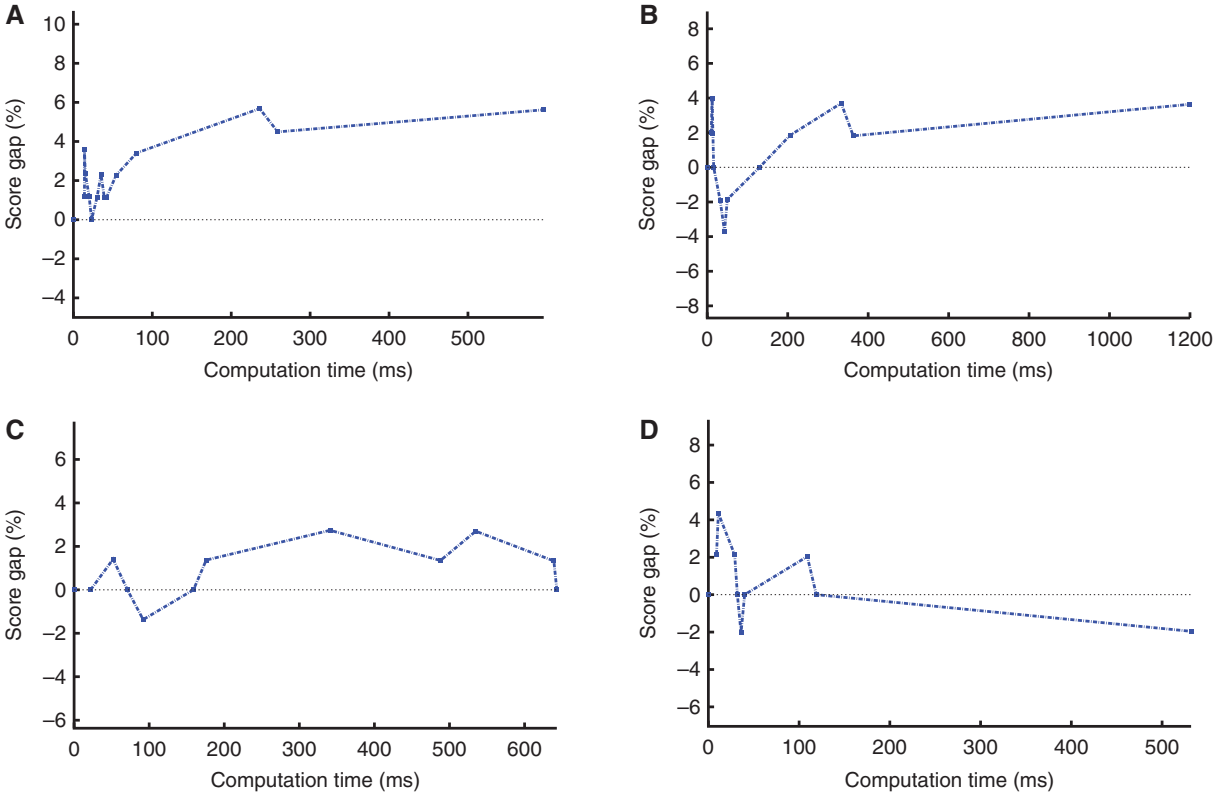

Figure 1: Patterns of Score Gap Evolution Over Time (Enhanced ILS vs. ILS). (A) enhILS superior all over the runtime; (B) enhILS superior over most of the runtime; (C) enhILS similar, better along the runtime; and (D) enhILS inferior, competitive along the runtime.

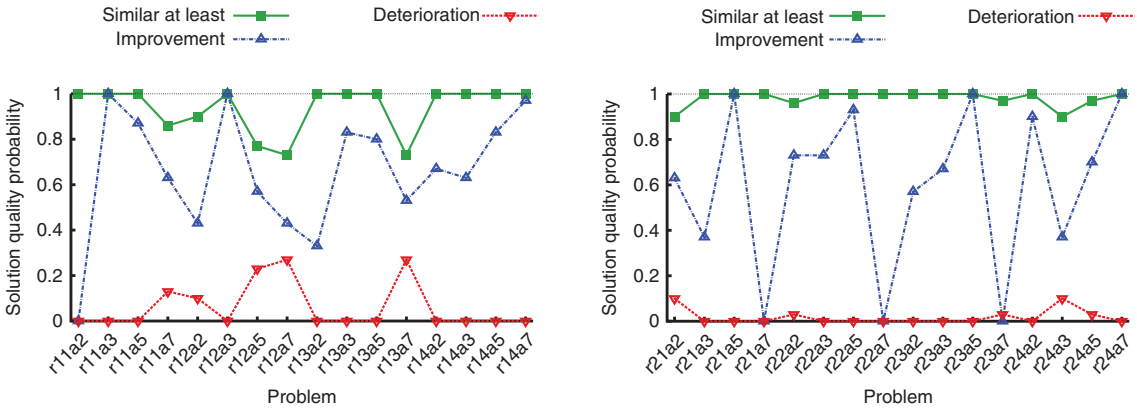

Figure 2: Probability that Enhanced ILS Improves Solution Quality of ILS.

For our statistical study on the performance of our approach, we consider 30 observations per problem configuration, obtained from running enhanced ILS on the problem instances. Specifically, we compute the probability that enhanced ILS will improve, not improve, or deteriorate, respectively, the solution quality achieved by ILS (Figure 2). Enhanced ILS is at least 50% likely to improve ILS's solution in most problems in each group. Enhanced ILS is overall 95% likely to perform at least as well as the baseline; we practically never get a score worse than ILS's, particularly for group 2 instances (Figure 2). Therefore, though stochastic, enhanced ILS is practically better than ILS when run once. It is further better if we do multiple runs.

5.4 Sensitivity to Parameters Setting

A metaheuristic's performance partly depends on how its free parameters are set. So to evaluate the impact of free parameters setting our approach's performance, we measure the sensitivity of the score gap to the free parameters setting (Figure 3). The free parameters settings we considered to configure enhanced ILS, noted as *config* in Figure 3, are given by Table 3. We configured ILS the best we could (Section 5.2). Enhanced ILS generally performs better than ILS, independently of how its free parameters are set. Specifically, it performs worse than ILS only in four problems and with a few free parameters settings in each group (Figure 3). On average, parameters tuning resulted in a score gain of 2.23% only. Therefore, enhanced ILS's performance is barely

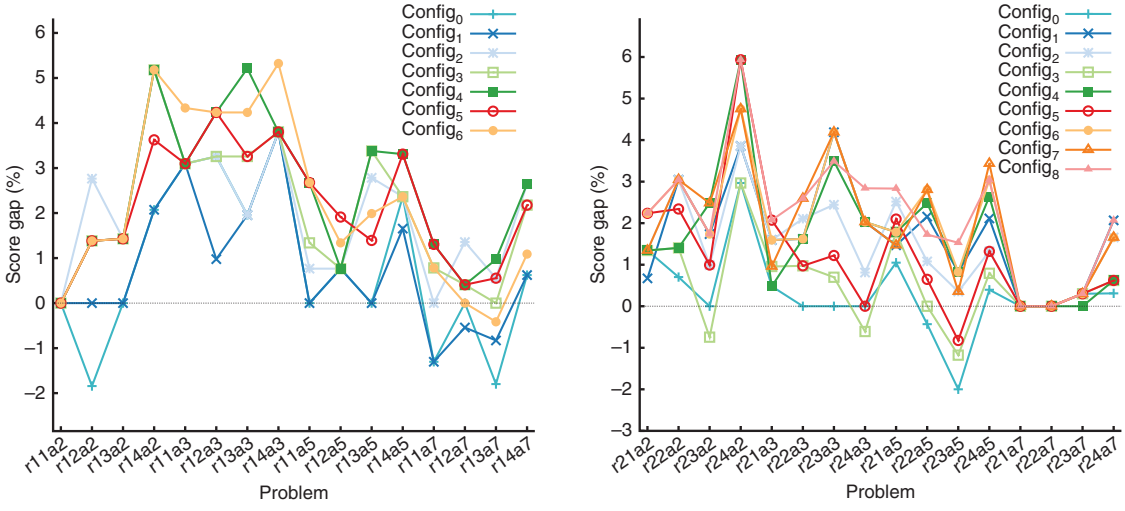

Figure 3: Enhanced ILS vs. ILS (Tuned as in §5.2) Score Gap Sensitivity to Free Parameters Setting (Config), with the Median Runs of Enhanced ILS.

Table 3: Some Free Parameters Settings Tried During Enhanced ILS Tuning.

Configuration	Group 1's instances		Group 2's instances	
	maxIter	maxPerturbationStrengthCoef	*maxIter*	maxPerturbationStrengthCoef
0	6000	0.1	1000	0.5
1	5000	0.125	3000	0.125
2	4000	0.2	2000	0.25
3	3000	0.25	1500	0.4
4	3000	0.33	4500	0.1
5	2100	0.4	1700	0.33
6	1500	0.5	4000	0.125
7	–	–	5000	0.125
8	–	–	4000	0.2

affected by the free parameters setting, and, generally, its superiority to the baseline's *best* performance does not result from parameters tuning.

6 Conclusions and Future Work

We have introduced MRTA/TC, a task allocation problem where tasks have temporal constraints and agents have capacity constraints. We presented an enhanced ILS which is suitable for time critical applications. We experimentally evaluated our approach on test instances that we generated. Our approach produces better quality solutions than the baseline while showing a more prompt response time. Our solution is barely sensitive to parameter tuning, and it is fairly robust to run replications. The proposed approach performs well when it is interrupted at different cutoff times. Future directions will investigate the use of sophisticated acceptance criterion, multiple construction heuristics, and/or adaptive parameter selection.

Acknowledgments: The first author is thankful to Prof. Maria Gini for her help, as well as to her research team at the University of Minnesota, in particular, to Dr. William Groves and Dr. Julio Godoy.

Research supported: This work was supported by the Algerian Ministry of Higher Education and Scientific Research, scholarship number 187/PNE/USA/2013–2014.

Bibliography

[1] S. Amador, S. Okamoto and R. Zivan, Dynamic multi-agent task allocation with spatial and temporal constraints, in: *Proc. International Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1495–1503, 2014.

[2] I.-M. Chao, B. L. Golden and E. A. Wasil, The team orienteering problem, *Eur. J. Oper. Res.* **88** (1996), 464–474.

[3] E. Ferrante, A. E. Turgut, E. Duéñez-Guzmán, M. Dorigo and T. Wenseleers, Evolution of self-organized task specialization in robot swarms, *PLoS Comput. Biol.* **11** (2015), e1004273.

[4] P. Ferreira, F. Boffo and A. Bazzan, Using swarm-GAP for distributed task allocation in complex scenarios, in: N. Jamali, P. Scerri and T. Sugawara, eds., *Massively Multi-Agent Technology*, Lecture Notes in Computer Science 5043, pp. 107–121, Springer, Berlin/Heidelberg, 2008.

[5] B. P. Gerkey and M. J. Matarić, A formal analysis and taxonomy of task allocation in multi-robot systems, *Int. J. Robot. Res.* **23** (2004), 939–954.

[6] A. Gunawan, H. C. Lau, P. Vansteenwegen and K. Lu, Well-tuned algorithms for the team orienteering problem with time windows, *J. Oper. Res. Soc.* **68** (2017), 861–876.

[7] Q. Hu and A. Lim, An iterative three-component heuristic for the team orienteering problem with time windows, *Eur. J. Oper. Res.* **232** (2014), 276–286.

[8] H. Kitano and T. Satoshi, Robocup rescue: a grand challenge for multiagent and intelligent systems, *AI Mag.* **22** (2001), 39–52.

[9] M. Koes, I. Nourbakhsh and K. Sycara, Heterogeneous multirobot coordination with spatial and temporal constraints, in: *Proc. 20th National Conf. on Artificial Intelligence (AAAI)*, pp. 1292–1297, June 2005.

[10] N. Labadi, R. Mansini, J. Melechovský and R. W. Calvo, The team orienteering problem with time windows: an LP-based granular variable neighborhood search, *Eur. J. Oper. Res.* **220** (2012), 15–27.
[11] M. G. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson and S. Jain, Auction-based multi-robot routing, in: *Robotics: Science and Systems*, pp. 343–350, 2005.
[12] H. R. Lourenço, O. C. Martin and T. Stützle, Iterated local search, in: M. Gendreau and J-Y. Potvin, eds., *Handbook of Metaheuristics, 2nd Edition* International Series in Operations Research and Management Science 146, pp. 363–398, Springer, New York, USA, 2010.
[13] A. Lucena, Time-dependent traveling salesman problem – the deliveryman case, *Networks* **20** (1990), 753–763.
[14] J. Melvin, P. Keskinocak, S. Koenig, C. Tovey and B. Y. Ozkaya, Multi-robot routing with rewards and disjoint time windows, in: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2332–2337, San Diego, CA, USA, 2007.
[15] H. Mitiche, J. Godoy and M. Gini, On the tuning and evaluation of iterated local search, in: *Proc. of Metaheuristics International Conference (MIC)*, Agadir, Morocco, 2015.
[16] R. Montemanni and L. M. Gambardella, An ant colony system for team orienteering problems with time windows, *Found. Comput. Decis. Sci.* **34** (2009), 287–306.
[17] E. Nunes, M. Manner, H. Mitiche and M. Gini, A taxonomy for task allocation problems with temporal and ordering constraints, *Robot. Auton. Syst.* **90** (2017), 55–70.
[18] S. D. Ramchurn, A. Farinelli, K. S. Macarthur and N. R. Jennings, Decentralized coordination in RoboCup Rescue, *Comput. J.* **53** (2010), 1447–1461.
[19] P. Scerri, A. Farinelli, S. Okamoto and M. Tambe, Allocating tasks in extreme teams, in: *Proc. International Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 727–734, 2005.
[20] M. M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints, *Oper. Res.* **35** (1987), 254–265.
[21] P. Vansteenwegen, W. Souffriau, G. V. Berghe and D. V. Oudheusden, Iterated local search for the team orienteering problem with time windows, *Comput. Oper. Res.* **36** (2009), 3281–3290.