SKATE : Successive Rank-based Task Assignment for Proactive Online Planning

Déborah Conforto Nedelmann, Jérôme Lacan, Caroline P. C. Chanel
Fédération ENAC ISAE-SUPAERO ONERA
Université de Toulouse, France
{deborah.conforto-nedelmann, jerome.lacan, caroline.chanel}@isae-supaero.fr

Abstract

The development of online applications for services such as package delivery, crowdsourcing, or taxi dispatching has caught the attention of the research community to the domain of online multi-agent multi-task allocation. In online service applications, tasks (or requests) to be performed arrive over time and need to be dynamically assigned to agents. Such planning problems are challenging because: (i) little or almost no information about future tasks is available for long-term reasoning; (ii) agent number, as well as, task number can be impressively high; and (iii) an efficient solution has to be reached in a limited amount of time. In this paper, we propose SKATE, a successive rank-based task assignment algorithm for online multi-agent planning. SKATE can be seen as a meta-heuristic approach that successively assigns a task to the best-ranked agent until all tasks have been assigned. We assessed the complexity of SKATE and showed it is cubic in the number of agents and tasks. To investigate how multi-agent multi-task assignment algorithms perform under a high number of agents and tasks, we compare three multi-task assignment methods in synthetic and real data benchmark environments: Integer Linear Programming (ILP), Genetic Algorithm (GA), and SKATE. In addition, a proactive approach is nested to all methods to determine near-future available agents (resources) using a receding-horizon. Based on the results obtained, we can argue that the classical ILP offers the better quality solutions when treating a low number of agents and tasks, i.e. low load despite the receding-horizon size, while it struggles to respect the time constraint for high load. SKATE performs better than the other methods in high load conditions, and even better when a variable receding-horizon is used.

Introduction

In the last couple of years, new applications have emerged where users make requests and a platform has to manage its resources to satisfy users' requests. Examples of these application domains are taxi dispatching (Dickerson et al. 2018), ridesharing (Herbawi and Weber 2012), crowdsourcing (Wang, Zhao, and Xu 2020) or package delivery (Cheikhrouhou and Khoufi 2021). All these application domains have led to advances in the field of online multi-agent multi-task assignment. In online service applications, tasks (or requests) arrive over time and need to be dynamically assigned to agents (Dickerson et al. 2018). Such planning problems are challenging because: (i) little or almost no information about future tasks is available for long-term reasoning; (ii) agent number, as well as, task number can be impressively high; and (iii) an efficient solution has to be reached in a limited amount of time. Regarding the first aspect, the requests arrival model is usually characterized with probability distributions (Mehta 2013) that are then used for planning (Alaei, Hajiaghayi, and Liaghat 2012), (Gong et al. 2022), (Hikima et al. 2022). The issue with this assumption is that beforehand data on the requests is needed to learn the model parameters and it can end up being very specific to a given application. In the present work, no assumption is made about the arrival requests model.

Another aspect is the nature of resources that can be disposable or reusable. Disposable resources can only be used once, while reusable resources can receive new assignments after completing their previous assigned tasks(Sumita et al. 2022). The present work assumes reusable resources since this setting is commonly found in applications, in particular ridesharing platforms. Linked to it, time management for requests assignments is also important. In online settings, planning is triggered for short time intervals. And, at each decision-making step, assignments will consider the requests that have arrived within the last time interval (Alonso-Mora et al. 2017), (Lesmana, Zhang, and Bei 2019). Differently, (Dickerson et al. 2018) assigns requests to agents as soon as they arrive (i.e. continuous planning). Both approaches have qualities and drawbacks: for the first one,it is more likely that (reusable) agents become available allowing for efficient solutions (e.g. path cost) but the waiting time of the requests to be assigned can be important; whereas for the second one, the waiting time decreases as the request is assigned to whatever available agent, however, the solution is generally worse (Sumita et al. 2022). Interestingly, (Conforto Nedelmann, Lacan, and Chanel 2023) proposed a proactive approach that can be seen as a compromise between the approaches cited above. It uses a receding-horizon to determine which agents will be available in a near-future.

Additionally to the time management aspect, online solving can be particularly challenging because one may design a solving method that has to be fast and efficient (e.g. to find efficient solutions in a limited amount of time). It can be particularly challenging when the number of agents (e.g. resources), as well as, the number of tasks (e.g. requests) is impressively high, a hard combinatorial problem present in several real-life settings.

To address these issues, we propose SKATE, a successive rank-based task assignment algorithm. SKATE is inspired by *MinPos*, a method proposed in (Bautin, Simonin, and Charpillet 2012) and later improved in (Bautin 2013). This meta-heuristic method was developed for multi-robot exploration in unknown environments. *MinPos* assigns a single exploration location for each robot by applying a ranking method based on the distance task-robot while also taking into account the position of the other robots. In its domain, *MinPos* achieved good performance while being lightweight. SKATE extends the *MinPos* algorithm to address online multi-agent multi-task assignments. SKATE fits the online planning requirements: it assigns a sequence of tasks to agents by adapting the metrics and the ranking system, it does not need to assume a task arrival model and can achieve good solutions even for large problems. Moreover, the proactive approach proposed by (Conforto Nedelmann, Lacan, and Chanel 2023) is nested to SKATE to determine near-future available agents using a receding-horizon. In this way, SKATE can be seen as a flexible approach that evaluates different sets of (available) agents to define the best assignments.

In the following section, related works are reviewed. Then the planning problem is formally described. Afterwards, the proactive approach and SKATE are presented followed by experiments and results discussion. Future research directions conclude the paper.

Related Work

The present work addresses the online multi-task assignment with reusable resources problem. In this class of problem, one may manage the resources (i.e. the agents of a fleet) to accomplish some tasks efficiently (Khamis, Hussein, and Elmogy 2015), (Hussein and Khamis 2013). In detail, one may minimize the overall traveled distance of the fleet while also minimizing the time between the registration of a request and its execution. This problem is close to the Multiple Traveling Salesmen Problem (MTSP), a classical optimization problem for which several solving methods have been proposed (Cheikhrouhou and Khoufi 2021). According to the taxonomy, our problem is a sub-variant called open-path multiple depots since the agents do not go back to their initial location after completing the assigned tasks, and the initial positions of agents are different.

(Dickerson et al. 2018) introduced the concept of reusable resources, which has been widely used in (Sumita et al. 2022), (Gong et al. 2022) and (Hikima et al. 2022). For time management, both immediate assignment ((Hikima et al. 2022), (Sumita et al. 2022)) and assignment in time intervals ((Alonso-Mora et al. 2017), (Lesmana, Zhang, and Bei 2019)) are common. Some works combine the two aspects such as (Wang and Bei 2022), where the goal is to find a balance between wanting more agents to be available while avoiding the withdrawal of the requests when the waiting time becomes too long.

In general, works assume a tasks arrival model describing the time of arrival and/or the position of future requests which allows them to reason in long-term. For instance, works focusing on online TSP, such as (De Filippo, Lombardi, and Milano 2019) or (Bampis et al. 2023) both use a priori information of requests to build a distribution of the more likely scenarios or to plan trajectories. In (Nanda et al. 2020), agents can reject specific requests based on the model. (Hikima et al. 2022) has developed a strategy when the requests arrival model is linked to the rewards for assigning a request to a resource. In (Burns et al. 2012), they determine the likely characteristics of requests a couple of time steps ahead of the assignment and then place their agents accordingly. Contrary to these works we do not assume tasks arrival model.

Several methods are used in the literature to solve the open-path multi-depot MTSP optimization problem. The most used method is Linear Programming (LP) and its variants. This approach is preferred because it finds the optimal solution (Dickerson et al. 2018), (Sumita et al. 2022). But it is mainly used for immediate assignment since LP has a potential high complexity (Basu et al. 2022). To respect the online time constraint, it handles only a small number of agents and requests. Genetic Algorithm (GA) is a popular meta-heuristic approach since it can provide solutions for large problems in a shorter amount of time than LP. They also provide good performance for multi-criteria objectives (Rangriz, Davoodi, and Saberian 2019). In the context of ridesharing, (Herbawi and Weber 2011) has compared the solution efficiency of a GA and several variants of Ant Colony Optimization (ACO). At its best the ACO gave similar performance to the GA and the GA was significantly faster than the ACO. (Wang, Zhao, and Xu 2020) used a variant of GA with some elements of ACO in the context of crowdsourcing. Their approach produced good solutions but was slightly slower than a classical GA.

Finally, in terms of experimental settings, papers focus on a small number of agents and requests. (Sumita et al. 2022) and (Dickerson et al. 2018) apply their methods to a real-life scenario of taxi dispatching in New York, where they limited their evaluation to 30 taxis and 100 or 550 requests, respectively. This is far from reality where hundreds of requests in New York can be received in 5 minutes. Regarding GA use, we found that (Herbawi and Weber 2012) has considered 250 drivers and 448 requests for their experiments in the context of ridesharing, which is higher but still much less than what exists in real-life settings. In the present work, we will show that SKATE can handle more agents and requests, being a promising and competitive approach for online multi-agent multi-task assignment planning problems.

Problem Statement

The multi-agent multi-task assignment is a combinatorial optimization problem. Over the entire time horizon, we aim to minimize the distance traveled by the fleet of agents (i.e. resources) while also minimizing the amount of time between the registration of a request and its execution. In the following, we call the latter criterion the request's waiting time. Generally speaking, we optimize both: user satisfaction while also economizing on the use of resources.

We consider a set of n agents denoted by $A = \{a_1, a_2, \dots, a_n\}$. The location of the agents is denoted by $\boldsymbol{p_a}$ for all $a \in A$. We assume requests arrive throughout the planning horizon T. The planning horizon T is divided into small intervals called time steps or time windows. The time windows are indexed by τ and their duration is a constant equal to δ. At time step $\tau \in T$, we assume the set of requests R_τ. A request $r \in R_\tau$ can be described by its location $\boldsymbol{p_r}$ and the time it was registered t^b_r. It is assumed a request can be assigned to only one agent $a \in A$. We note the set of requests assigned to an agent $a \in A$ as $R_a = \{r^1_a, r^2_a, ..., r^m_a\}$, where m is the number of requests assigned to this agent a at a given time. To describe if $r \in R_\tau$ has been assigned to $a \in A$, we use the binary variable $x_{a,r}$ with $x_{a,r} = 1$ and $x_{a,r} = 0$ if that's not the case. The set of requests assigned to $a \in A$ can also be described as $R_a = \{\bigcup_{r \in R} r | x_{a,r} = 1\}$. Assuming an ordered set R_a, the distance between the expected location of agent a and the location of request r, and the fact that agents move with a constant speed v_a, one can compute the expected execution time of a request $r \in R_a$, noted as t^{ex}_r.

Therefore, the general optimization problem we address can be formalized as:

$$\min \Bigg[\sum_{\tau=0}^{T-1} \Big(\alpha \sum_{a \in A} d_{R_a}\Big(\bigcup_{r \in R} r | x_{a,r} = 1\Big) + (1-\alpha)\Big(\sum_{a \in A} w_{R_a}\Big(\bigcup_{r \in R} r | x_{a,r} = 1\Big)\Big)\Big)\Bigg] \quad (1)$$

subject to:

$$\sum_{a \in A} x_{a,r} \leq 1, \forall r \in R$$

with :

$$d_{R_a} = \|\boldsymbol{p_a} - \boldsymbol{p_{r_1}}\| + \sum_{i=1}^{|R_a|-1} \|\boldsymbol{p_{r_i}} - \boldsymbol{p_{r_{i+1}}}\|, \text{ and}$$

$$w_{R_a} = \sum_{r \in R_a} (t^{ex}_r - t^b_r)$$

$\alpha \in [0, 1]$ and $(1 - \alpha)$ are weights used to balance the importance between the two criteria. The formulation of d_{R_a} was inspired by (Cheikhrouhou and Khoufi 2021) that uses it for an open-path multi-depot MTSP, and denotes the path distance from the initial position of $a \in A$ until the position of its last assigned request r^m_a. Then, we define the waiting time w_{r_a} as the duration between the time of registration of the request $r \in R_\tau$ and its execution by agent a.

The problem with such a formulation is that one may know which requests will be received at each time step τ in the given horizon T. Here, and in several application domains (e.g. taxi dispatching, package delivery) little information is available about the requests' arrival. As a result, it is extremely hard to solve this general optimization problem over the entire time horizon T.

To treat this online problem with no assumption regarding the requests arrival model, we were inspired by the proactive approach presented in (Conforto Nedelmann, Lacan, and Chanel 2023). In this paper, instead of solving the general optimization problem, they broke it into a small problem at each time step τ to consider the (new) set of tasks R_τ with a proactive perspective. For that, they proposed the concept of *available* agents. In classical reactive approaches, *available* agents are the ones that have already finished executing their previous requests at τ. Whereas in the article cited above, a receding-horizon approach was used for determining the agents to be used at τ: the said *available* agents are the ones that will complete their previously assigned tasks within an immediate horizon $H(k) = k\delta$ with $k \geq 0$ (e.g. $H(5) = 5\delta$ or $H(5) = \tau + 5$). This lets us anticipate the availability of resources and not have to wait for agents to finish their requests before assigning them new ones. We call $A_\tau(H)$ the available agents at τ for a certain time horizon $H(k)$.

With this in mind, and adopting a similarly proactive approach, we will assign requests at each time step τ accounting with the agents considered as *available* within the horizon H. Thus, we search for a solution for:

$$\min \Bigg[\alpha \sum_{a \in A_\tau(H)} d_{R_a}\Big(\bigcup_{r \in R_\tau} r | x_{a,r} = 1\Big) + (1-\alpha)\Big(\sum_{a \in A_\tau(H)} w_{R_a}\Big(\bigcup_{r \in R_\tau} r | x_{a,r} = 1\Big)\Big)\Bigg] \quad (2)$$

subject to:

$$\sum_{a \in A_\tau(H)} x_{a,r} \leq 1, \forall r \in R_\tau$$

Proactive Online Task Assignment Approach

In the following, we will first explain the general proactive online process for task assignment, and then SKATE.

General Process Algorithm 1 illustrates the general online process. At each time step τ, we get the new requests R_τ that have arrived since $\tau - 1$ and were stored in the buffer B (line 5). Then we check the agents' availability at horizon H (lines 7-8): we use a variable t^{occ}_a indicating the time at which the agent a will finish its last assigned request. Finally, we use an algorithm to assign a sequence of tasks of R_τ to the available agents $A_\tau(H)$ (line 9).

Note that this process can implement any task assignment algorithm that solves the optimization problem presented in Eq. 2. However, this planning problem can be challenging because an efficient solution has to be reached in a limited amount of time regardless of the number of agents or tasks. In this context, classical approaches such as Integer Linear Programming, or meta-heuristic approaches (e.g. Genetic Algorithms) may be blocked or only find mediocre solutions. In the following, we propose a method we call Successive Rank-based Task Assignment (SKATE), a simple yet efficient online method for a multi-agent multi-task assignment that can be eventually used in line 9 of Alg. 1. SKATE can be seen as a meta-heuristic solving process.

Algorithm 1: Proactive online task assignment process

```
1: Agents $A$ at position $\boldsymbol{p_a}$ and $t_{a_{occ}} = 0, \forall a \in A$
2: Horizon $H = k\delta$ and $R_{\tau=0} \leftarrow \emptyset$
3: for each time step $\tau$ do
4:     $A_\tau(H) \leftarrow \emptyset$
5:     $R_\tau \leftarrow$ GetTasksFrom$(B, R_{\tau-1})$
6:     for $a \in A$ do
7:         if $t_{a_{occ}} < \tau + H$ then
8:             $A_\tau(H) \leftarrow a$
9:     Assign $R_\tau$ tasks to $A_\tau(H)$ agents (SKATE or LP or GA)
```

Successive Rank-Based Task Assignment

The main principle of SKATE is the ranking of the requests for each agent while taking into account the expected location of the other agents. SKATE, presented in Alg. 2, assigns all the registered requests R_τ to the set of available agents $A_\tau(H)$ of size n. This assignment is implemented into several rounds where at each round, a maximum of n requests are assigned. This way, by the successive rounds, a sequence of tasks is assigned to each agent. The process is repeated until no request is left (line 2).

To proceed with the assignment we fill a cost matrix M^C of size $|A_\tau(H)| \times |R_\tau|$ where we calculate the cost to assign any request $r_j \in R_\tau$ to any agent $a_i \in A_\tau(H)$ (line 4). Since we are interested in minimizing both the traveled distance and the waiting time, the cost function will be a combination of these two criteria such as:

$$cost(a_i, r_j) = \alpha \frac{d(a_i, r_j)}{v_{a_i}} + (1-\alpha) w_{a_i}(r_j) \tag{3}$$

Algorithm 2: Successive Rank Based Task Assignment

```
1: procedure SKATE($A_\tau(H)$, $R_\tau$)
2:     while $R_\tau \neq []$ do
3:         $A = A_\tau(H)$
4:         Compute the cost matrix $M^C$ such as $M^C_{i,j} = cost(a_i, r_j), \forall a_i \in A$ and $\forall r_j \in R_\tau$
5:         Compute the rank matrix $M^R$ such as $M^R_{i,j} = Card(\bar{A})$ with $\bar{A} = \{\forall a_k \in A | M^C_{k,j} < M^C_{i,j}\}$
6:         Definition of the variable of ranking S=0
7:         while $R_\tau \neq \emptyset$ and $A \neq \emptyset$ do
8:             for $a \in A$ do
9:                 $rank_{min} = min(M^R_{a,r} \forall r \in R_\tau)$
10:                if $rank_{min} = S$ then
11:                    assign $r$ to $a$
12:                    $t^{ex}_r = w_a(r)$ and $t^{occ}_a = t^{occ}_a + \frac{d(a,r)}{v_a}$
13:                    $A = A \setminus a$ and $R_\tau = R_\tau \setminus r$
14:            $S = S + 1$
15:        for $a \in A$ do
16:            Get last request $r$ assigned to $a$
17:            $\boldsymbol{p_a} = \boldsymbol{p_r}$
18: end procedure
```

where, $d(a_i, r_j) = \|\boldsymbol{p_{a_i}} - \boldsymbol{p_{r_j}}\|$ is the Euclidean distance between the anticipated position of a_i and the position of the request r_j, v_{a_i} is the velocity of the agent assumed constant (it enables to compare the two criteria, the distance term is converted into time), and $w_{a_i}(r_j) = t^{occ}_{a_i} + \frac{d(a_i,r_j)}{v_{a_i}} - t^b_{r_j}$ is the request waiting time. It corresponds to the time between $t^b_{r_j}$ when the request was generated, and $t^{ex}_{r_j}$, when the agent arrives from his expected location to that of the request. The term $t^{occ}_{a_i}$ refers to the time the agent is considered busy (e.g. executing previous requests) until it can execute r_j. To rank the requests for an agent $a_i \in A$, we will not simply consider the value of the cost for that agent, but also take into account the costs of the other agents for each specific request $r \in R_\tau$ (line 5). For that we build the rank matrix M^R using the cost matrix M^C: for the value $M^R_{i,j}$, each agent a_i will determine how many other agents a_k have a lower cost for that request r_j. If no other agent has a lower cost, then a_i has the lower ranking with $M^R_{i,j} = 0$. Otherwise, the ranking is equal to the number of agents with a lower cost such as $M^C_{k,j} < M^C_{i,j}$.

Note, some agents can have multiple requests ranked 0 (they are the better-placed agent for more than one request) and some can have none. If two requests have the same minimal ranking value, then the request with the lower cost is assigned to the agent. The assigned request and the agent then both become occupied (line 13) and we update the waiting time of the assigned request and the time until which the agent is occupied (line 12). When all of the agents have become occupied, we consider that this round of assignment is over and we update the position of the agents using the position of the last request assigned to each of them (line 17) before starting a new round(line 3).

Complexity of SKATE To analyze the complexity of SKATE, as we can see in Alg. 2, the two necessary inputs are the n available agents and the m requests to assign. After the first assignment round, there are $m - n$ requests left. This means that to assign all the requests, the actions inside the loop of line 2 must be done $\boldsymbol{\frac{m}{n} - 1}$ times. The filling of the cost matrix in line 4 has a cost of $\boldsymbol{mn}$. And, ranking these requests for each agent while taking into account the other agents' positions is $\boldsymbol{n^2m}$ expensive. In the following, we detail the calculation of the complexity of SKATE:

$$\begin{aligned}
&\overbrace{\sum_{i=0}^{\frac{m}{n}-1} n(m-in)}^{\text{Cost matrix filling}} + \overbrace{\sum_{i=0}^{\frac{m}{n}-1} n^2(m-in)}^{\text{Ranking}} + \overbrace{\sum_{i=0}^{\frac{m}{n}-1} n^2(m-in)}^{\text{Assignment}} \\
&= \sum_{i=0}^{\frac{m}{n}-1} mn - n^2 \sum_{i=0}^{\frac{m}{n}-1} i + 2(\sum_{i=0}^{\frac{m}{n}-1} mn^2 - n^3 \sum_{i=0}^{\frac{m}{n}-1} i) + \\
&\quad + \frac{m}{n}mn - n^2 \sum_{i=0}^{\frac{m}{n}-1} i + 2(\frac{m}{n}n^2m - n^3\frac{m}{2n}(\frac{m}{n}-1)) \\
&= m^2 - \frac{1}{2}n^2\frac{m}{n}(\frac{m}{n}-1) + 2(m^2 - n\frac{m^2}{2} + \frac{n^2m}{2}) \\
&= m^2 + \frac{mn}{2} + m^2n + n^2m = \boldsymbol{m^2n + n^2m}
\end{aligned} \tag{4}$$

In the next section, we present the methodology to evaluate the proactive online assignment approach using SKATE.

Experiments & Results

To evaluate the impact of the proactive approach, we first compute the assignments with a reactive approach (no anticipation of agents availability), noted $H(0)$, then use the receding-horizon to anticipate resources from one to five time windows, denoted as $H(1)$ to $H(5)$. We also use a variable receding-horizon $H(v)$ as proposed by (Conforto Nedelmann, Lacan, and Chanel 2023), where the assignment using different receding-horizons (from $H(0)$ to $H(5)$) is computed in parallel and the best solution is kept.

To evaluate the solutions proposed by SKATE, assignments are also computed using baselines from the literature. More specifically, we compare SKATE with two baselines, one applying Integer Linear Programming (i.e. branch-and-bound algorithm), and another one applying a Genetic Algorithm. We did not include a comparison with a (simple) greedy assignment strategy given its partial novelty[1]. Indeed, the authors of *MinPos*, on which SKATE is based, have already shown the advantages of their method compared to a greedy strategy in (Bautin, Simonin, and Charpillet 2012).

Finally, assignment methods are evaluated on two benchmarks to analyze their solution efficiency and scalability. The first benchmark is a synthetic one for which we have a constant number of requests arriving every $\tau \in T$. The second benchmark, already used in the literature (Sumita et al. 2022), is based on registered requests of taxis in New York in January 2013[2]. This realistic scenario is particularly interesting for mimicking a high-load problem, where we need to find, online assignments for a high number of requests and agents in a limited amount of time. Additionally, in this scenario the number of request vary within the planning horizon. Regarding the α value, we carried out empirical tests for values of 0, 0.25, 0.5, 0.75, and 1. As, the best balance between distance and the waiting time criteria for all methods was achieved with $\alpha = 0.75$, we kept this value.

Metrics The metrics used to quantify the impact of the use of a receding-horizon and to compare the efficiency of the three solving methods are: (i) the overall distance traveled by the (fleet of) agents; (ii) the percentage of assigned requests; (iii) the average waiting time for requests; and (iv) the time necessary for computing the assignments.

Baselines

Integer Linear Programming To build the Integer Linear Programming (ILP) model we were inspired by (Kara and Bektas 2006). This work details the objective function and constraints of the classical MTSP and variants, including the open-path multi-depot MTSP. To assign a sequence of requests R_a to an agent $a \in A$, we use the set $L = A_\tau(H) \cup R_\tau$ and the cost matrix M^C of size $|L| \times |L|$ where $M^C_{i,j} = c_{i,j}, \forall (i,j) \in L^2, i \neq j$. Each element $c_{i,j}$ in this matrix refers to the cost between agents and requests, such as:

$$c_{i,j} = \alpha \frac{d(i,j)}{v_a} + (1-\alpha) w_a(i,j),\ \forall i,j \in L$$

The objective function and its constraints are defined as:

$$\text{minimize } \Big(\sum_{i \in L} \sum_{j \in L} c_{i,j} x_{i,j} \Big) \tag{5}$$

subject to:

$$\sum_{i \in L} \sum_{j \in A_\tau(H)} x_{i,j} = 0 \text{ and } \sum_{j \in R_\tau} x_{i,j} \leq 1,\ \forall i \in L$$

$$\sum_{i \in L} x_{i,j} = 1,\ \forall j \in R_\tau \text{ and } x_{i,j} + x_{j,i} \leq 1,\ \forall (i,j) \in L^2$$

with $x_{i,j}$ being a boolean variable, *i.e.* $x_{i,j} = \{0,1\}$. As we want to assign a sequence of requests to each agent, we need the agent-to-task and the task-to-task costs to determine the cost of a given sequence, and then choose the sequences that minimize the objective function for all agents. The first constraint clarifies that we can not assign an agent to another agent or an agent to a request. The second models the open-path condition of MTSP. The third imposes that all the requests must be assigned. The penultimate avoids an agent going back and forth between two positions and the last one imposes that the assignment variable is a boolean. We use the Gurobi Optimizer, which employs the branch-and-bound algorithm to solve ILP problems. The solver explores possible solutions and tries to find the optimal one. Note that the branch-and-bound can be time-consuming: in the worst-case scenario we have a complexity of O(2^n) (Basu et al. 2022), which is significantly more than SKATE (O($m^2n + n^2m$)).

Genetic Algorithm The Genetic Algorithm (GA) is an evolutionary algorithm inspired by the natural selection process. The principle is that we start with a random set of chromosomes (e.g. initial assignments). The chromosomes are evaluated and the ones with the better scores (e.g. cost) are then used to create a new population (e.g. new set of solutions). This way, the population can hopefully improve generation after generation. The operations used to build a new generation are the crossover (where two parent chromosomes are fused) and mutations (where we swap the order of the requests or the requests between different agents). The algorithm stops when a condition is reached, in general, either the solution is no longer improving or a time limit is reached. For the problem in this work, each chromosome represents a specific assignment of the requests to the available agents. We have almost used the same approach structure and parameters as (Conforto Nedelmann, Lacan, and Chanel 2023). However, in our optimization problem we focus on minimizing both the overall traveled distance of the agents and the waiting time of the request so the fitness function we use is expressed as follows:

$$\alpha \Big(\sum_{a \in A_\tau(H)} d_a \frac{1}{D_{max}} \Big) + (1-\alpha) \Big(\sum_{a \in A_\tau(H)} w_a \frac{1}{W_{max}} \Big)$$

[1]However this comparison is included in the extended version available at https://openscience.isae-supaero.fr/Default/doc/SYRACUSE/15709/skate-successive-rank-based-task-assignment-for-proactive-online-planning

[2]available at: http://www.andresmh.com/nyctaxitrips/

where in the proposed chromosome, d_a is the distance traveled by $a \in A$ to execute all its R_a requests and $w_a = \sum_{r \in R_a} w_r$ as the sum of the waiting time of the requests assigned to a. To compare it to the first generation, we normalize each term of the fitness by D_{max} and W_{max}, which are the maximum values obtained in the first generation for the total traveled distance and waiting time respectively. We consider these values as the worst case since the first generation is randomly filled.

Synthetic Benchmark Experiments & Results

Setup We consider two different set-ups: for the first, at each time step 20 requests are randomly placed in the workspace defined by a square of 10m x 10m; the second with 50 requests being generated at each time step. For both of them, the fleet will be composed of 10 agents moving at a constant speed of $1m/s$. The duration of a time window is 5 seconds. The total simulation horizon is composed of 30 time steps. The simulation process is executed 10 times. In this synthetic benchmark, we consider that the requests are executed when the agents reach their locations.

20 requests every time step The results in this set-up are illustrated in Fig. 1. Regarding the percentage of assigned requests (Fig. 1b) almost all of them are assigned for all three methods, confirming they all can handle such a scenario. In terms of traveled distance (Fig.1a), for all receding-horizons, the ILP method gives the best results. This is expected since the ILP looks for (sub-)optimal solutions. Between the GA and SKATE, the GA is more interesting when using the reactive approach ($H(0)$), however, SKATE produces better values when using the proactive approach ($H(k)$ with $k > 0$).

Note that the proactive approach provides improvements for all solving methods, helping to reduce the traveled distance. For all three methods, the better results happen when using the variable receding-horizon $H(v)$, and there, SKATE achieves values close to the ILP ones. Table 1 shows the mean of the requests' waiting time for ILP, GA and SKATE for $H(v)$. Once again, ILP gives the better solution however SKATE solutions are competitive. GA has a larger mean than the others, which is coherent with its higher traveled distance. Fig.1c compares the time necessary for computing assignments for the three methods. Central line dots correspond to the mean time for a given time step across the 10 executions. The shadow area corresponds to the min-max values. We notice the ILP method is already reaching its time limit at 5 seconds, which corresponds to our time limit for computations and the duration of the time window δ. The GA method needs, on average, 3 seconds and SKATE less than a second. This highlights that SKATE is much more lightweight than the other methods and suggests it could be interesting in higher-load setups. As a result, ILP gives the best performances (followed by SKATE and then GA) but is almost reaching the time budget in this setup.

50 requests every time step The results for this heavier set-up are given in Fig. 2. The first observation is about the disparity in the percentage of assigned requests shown in Fig. 2b. SKATE manages to assign close to 100% of the requests in general, though we notice a slight improvement using the proactive approach. For GA, the percentage of the assigned requests is lower than for SKATE but still manages to assign almost 90% or more of the requests both for reactive and proactive approaches, and for $H(v)$, the value achieved is close to the SKATE one. Surprisingly, the outlier results concern ILP which is far from assigning all the requests. We investigated why these percentages are so low and discovered that if all agents are unavailable in two subsequent time steps (which means there would be 150 requests to assign at the next time step), the ILP approach can not handle such a load and reaches the time limit before managing to find any solution. In the same situation, GA and SKATE can handle this additional load. We notice improvements in solution efficiency for ILP when the proactive approach is used: anticipating the availability of the agents allows to delay or even not encounter the problem of getting stuck. [3]. In terms of traveled distance (see Fig. 2a), SKATE gives better results than the GA regardless of the size of the receding-horizon while ILP shows fluctuations but we speculate it is linked to the low number of assigned requests. As the size of the receding-horizon grows, the traveled distance for the ILP also grows since more requests have been assigned. But this does not give us the optimal: when a solution is obtained, it is in majority a sub-optimal solution. Regarding the average request waiting time (see Table 1), ILP still gives the lowest value. However, it only gives partial information since this average only takes into account the assigned requests. SKATE has an average waiting time which is 4 seconds higher than ILP however it assigns more requests than ILP. As a result, we judge that ILP is not able to handle a higher-load set-up since it could not produce solutions within the defined computation time limit. Between the GA and SKATE, SKATE gives consistently better results whether in terms of distance, number of assigned requests, or average waiting time.

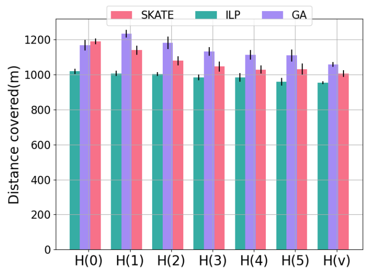
	Assignment method		
	ILP	GA	SKATE
$R_\tau = 20$	22.52	33.6	23.5
$R_\tau = 50$	51.03	263.96	55.87

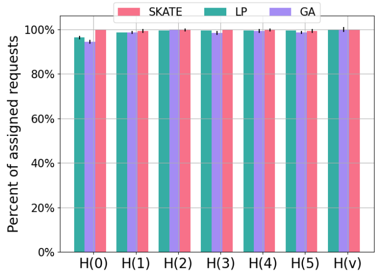
Table 1: Average waiting time of assigned requests (in seconds) for $A = 10$ and $H(v)$.

Real-life data Experiments & Results

Setup To confront solving methods to realistic settings, we exploit an open data set listing taxi requests in New York City. In this data set, a request characterizes a real-life taxi call with a starting point located at $\boldsymbol{p_{r_s}}$, a final point at $\boldsymbol{p_{r_f}}$, and a request registration time. In this benchmark, we consider the pickup time as the request registration time t_{r_b}, and the request is executed once the agent has reached the destination location. The distance cost is then adapted to account as: $d_{a,r} = \|\boldsymbol{p_a} - \boldsymbol{p_{r_s}}\| + \|\boldsymbol{p_{r_s}} - \boldsymbol{p_{r_f}}\|$. This additional distance is also taken into account for the waiting time calculation. The total distance for an agent to execute all its

[3]These results are detailed in the extended version.


(a) Overall traveled distance


(b) Percentage of assigned requests

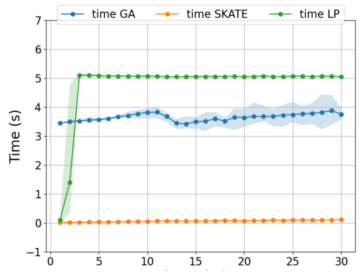
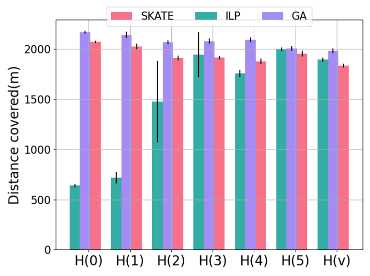
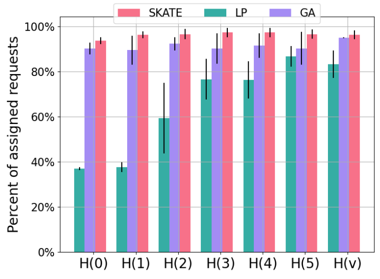

(c) Average computation time per time step.

Figure 1: Comparison of reactive and proactive approaches for ILP, GA and SKATE and for $R_\tau = 20$.


(a) Overall traveled distance


(b) Percentage of assigned requests

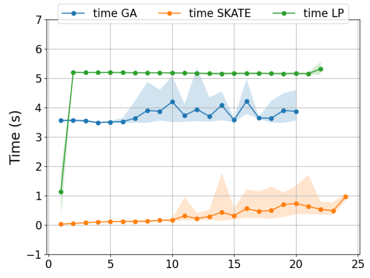

(c) Average computation time per time step.

Figure 2: Comparison of reactive and proactive approaches for ILP, GA and SKATE for $R_\tau = 50$.

assigned requests is defined as:

$$d_a = \|\boldsymbol{p_a} - \boldsymbol{p_{r_{1_s}}}\| + \sum_{i=1}^{m-1} \|\boldsymbol{p_{r_{i_s}}} - \boldsymbol{p_{r_{i_f}}}\| + \sum_{i=1}^{m-1} \|\boldsymbol{p_{r_{i_f}}} - \boldsymbol{p_{r_{i+1_s}}}\|$$

We evaluate our methods with a reactive and proactive approach over 3 nights, from January 7th to January 9th 2013 from 12AM to 7AM. Each night corresponds to an independent simulation. The time window δ (i.e. difference between two time steps) has a duration of 5 minutes. We consider a fleet of 1000 agents traveling at a constant velocity of 30 mph (the speed limit in New York). Contrary to the previous benchmark, here we address a variable number of requests[4] arriving throughout the simulation. Moreover, such a simulation has a heavier load than the previous benchmark, since we have more requests and agents (1000 agents and 200-1200 requests) and still a limited amount of time for calculations. Since ILP was already struggling in the previous benchmark, we do not include it in this part. Instead, we will be focusing only on the GA and SKATE results.

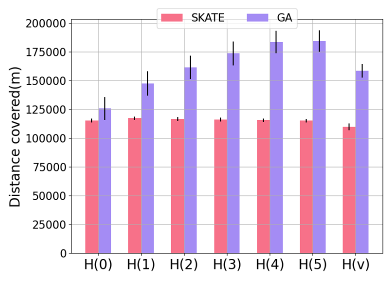
Comparison for 1000 Agents The results of experiments using real-life data are presented in Fig. 3. Looking at the percentage of assigned requests in Fig. 3a, SKATE is more efficient than GA. However, SKATE is not able to assign more than 80% of the requests due to the significant amount of requests arriving between 6 and 7 AM. With that amount of requests, the agents stay busy for more time than the receding-horizon size, being not available until the end of the simulation. GA is further impacted because of less efficient assignments: the agents cover a greater distance and are busy for an even longer time. In the time limit of 5 minutes, GA only computes a couple of generations, preventing GA from improving solutions. It is confirmed by the solving time (see Fig. 3c). GA struggles to find an efficient assignment within the time budget when compared to SKATE. Poor solutions also have an impact on the waiting time which is bigger for GA than for SKATE[5]. Interestingly, the proactive approach increases the percentage of assigned requests for both approaches but the progression is more important for GA. Additionally, we compare the time needed for assignment calculations and the theoretical complexity for SKATE shown in Figure 5 for $H(v)$. The similarity of curves shows that the complexity we calculated is correct.

SKATE with Different Fleet Sizes The experiments above showed that a fleet of 1000 agents is not sufficient to handle all of the requests. To estimate how many agents would be necessary, we studied the impact of the fleet size on the percentage of assigned requests and waiting time. To see how occupied the agents are, we have recorded how much time the agents stay idle waiting for a new assignment. We tested a fleet of 500, 750, 1000, 1250 and 1500 agents.

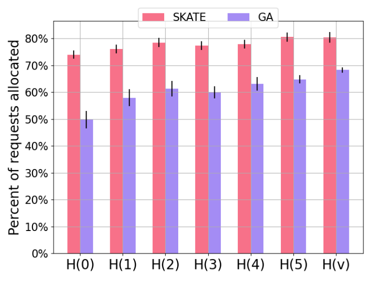
Results are compiled in Fig. 4 and Table 2. Only the results for the variable receding-horizon $H(v)$ is presented since it consistently gives the best results in previous experiments. First, Fig. 4a shows only a small variation of the overall distance covered by agents with SKATE. Consider-

[4]This varying number is represented in the extended version.

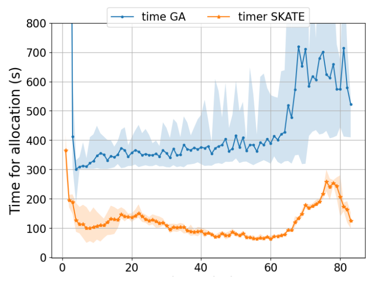
[5]These results are also detailed in the extended version.



(a) Overall traveled distance.

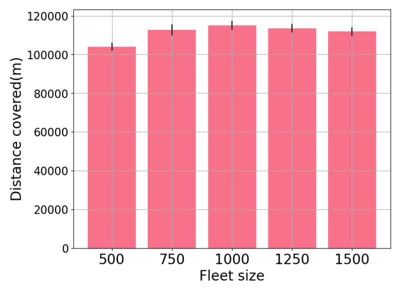


(b) Percentage of assigned requests.

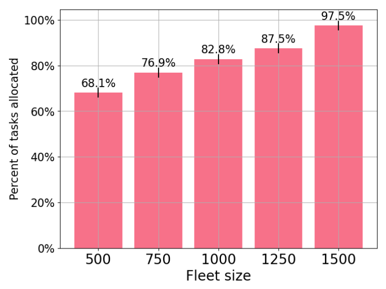


(c) Average computation time per time step.

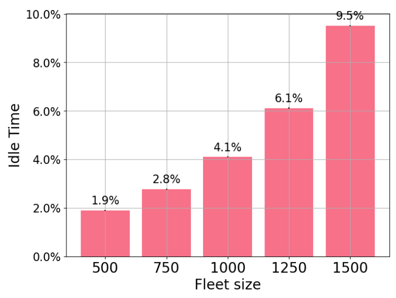
Figure 3: Comparison of reactive and proactive approaches for GA and SKATE in real-life settings.



(a) Comparison of traveled distance.



(b) Percentage of assigned requests.



(c) Idle time.

Figure 4: Comparison of the overall distance, number of assigned requests and idle time regarding the fleet size.

ing there are more agents, we notice that the individual distance agents travel is lower. It also suggests a better distribution of the requests across the agents, as we see in Fig. 4b that the number of assigned requests increases with the number of agents and gets close to 100%. In terms of waiting time (see Tab. 2), it decreases when the number of agents increases because of their better distribution. We note a sort of threshold around 15 minutes, where the waiting time decreases marginally for more than 1000 agents. We speculate it is linked to the physical aspects of the problem (distance between the starting and final point). Looking at idle time (i.e. when agents wait for their new assignments) in Fig. 4c, it increases alongside the increase of the number of agents. It is mainly due to the agents having to wait one or several rounds for their first assignment at the start of the night, and a smaller part is due to the lower amount of requests between 3:30AM and 5:00AM.

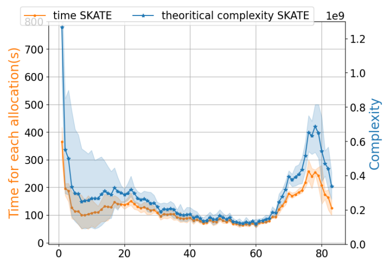


Figure 5: Time and complexity for SKATE per time step.

	Size of the fleet				
	500	750	1000	1250	1500
$\bar{w_r}$	51.52	29.44	16.3	14.64	13.36

Table 2: Average waiting time $\bar{w_r}$ (in minutes).

Conclusion and Future Work

One of the main challenges of the online multi-agent multi-task assignment field is to design solving methods able to find efficient solutions while scaling well (lightweight) given the combinatorial aspect of such planning problems. We proposed SKATE, a successive rank-based task assignment algorithm for online multi-agent planning. Built upon a ranking method considering agent-to-task costs, SKATE successively assigns a task to the best-ranked agent until all tasks have been assigned. We compared SKATE with two baseline methods already used for online planning: GA and ILP. We found from a theoretical and experimental approach that SKATE is faster than the other methods. For light-load, ILP gives better cost-wise results, however in high-load, ILP struggles whereas SKATE scales well giving efficient solutions. SKATE and GA were also compared in real-life settings (e.g. taxi dispatching). SKATE is the best method between the two. Regarding the results achieved with SKATE and different fleet sizes, we plan to further improve our approach to also determine, online, the correct fleet size to met current demands. For instance, by reducing the number of agents when requests are low or by increasing it when the number of requests increases.

References

Alaei, S.; Hajiaghayi, M.; and Liaghat, V. 2012. Online Prophet-Inequality Matching with Applications to Ad Allocation. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, 18–35. New York,USA: Association for Computing Machinery. ISBN 9781450314152.

Alonso-Mora, J.; Samaranayake, S.; Wallar, A.; Frazzoli, E.; and Rus, D. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3): 462–467.

Bampis, E.; Escoffier, B.; Hahn, N.; and Xefteris, M. 2023. Online TSP with Known Locations. In *Algorithms and Data Structures: 18th International Symposium, WADS 2023, Montreal, QC, Canada, July 31 – August 2, 2023, Proceedings*, 65–78. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-031-38905-4.

Basu, A.; Conforti, M.; Di Summa, M.; and Jiang, H. 2022. Complexity of Branch-and-Bound and Cutting Planes in Mixed-Integer Optimization. *Math. Program.*, 198(1): 787–810.

Bautin, A. 2013. *Stratégie d'exploration multirobot fondées sur le calcul de champs de potentiels*. Theses, Université de Lorraine.

Bautin, A.; Simonin, O.; and Charpillet, F. 2012. MinPos : A Novel Frontier Allocation Algorithm for Multi-robot Exploration. In Su, C.-Y.; Rakheja, S.; and Liu, H., eds., *Intelligent Robotics and Applications*, 496–508. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-33515-0.

Burns, E.; Benton, J.; Ruml, W.; Yoon, S.; and Do, M. 2012. Anticipatory On-Line Planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 22(1): 333–337.

Cheikhrouhou, O.; and Khoufi, I. 2021. A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy. *Computer Science Review*, 40: 100369.

Conforto Nedelmann, D.; Lacan, J.; and Chanel, C. 2023. Online Proactive Multi-Task Assignment with Resource Availability Anticipation. In Ferrando, A.; and Cardoso, R., eds., Proceedings of the Third Workshop on *Agents and Robots for reliable Engineered Autonomy,* Krakow, Poland, 1st October 2023, volume 391 of *Electronic Proceedings in Theoretical Computer Science*, 3–17. Open Publishing Association.

De Filippo, A.; Lombardi, M.; and Milano, M. 2019. How to Tame Your Anticipatory Algorithm. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 1071–1077. AAAI Press. ISBN 9780999241141.

Dickerson, J.; Sankararaman, K.; Srinivasan, A.; and Xu, P. 2018. Allocation Problems in Ride-Sharing Platforms: Online Matching With Offline Reusable Resources. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Gong, X.-Y.; Goyal, V.; Iyengar, G. N.; Simchi-Levi, D.; Udwani, R.; and Wang, S. 2022. Online Assortment Optimization with Reusable Resources. *Management Science*, 68(7): 4772–4785.

Herbawi, W.; and Weber, M. 2011. Ant Colony vs. Genetic Multiobjective Route Planning in Dynamic Multi-Hop Ridesharing. In *Proceedings of the 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, ICTAI '11, 282–288. USA: IEEE Computer Society. ISBN 9780769545967.

Herbawi, W.; and Weber, M. 2012. The ridematching problem with time windows in dynamic ridesharing: A model and a genetic algorithm. 1–8. ISBN 978-1-4673-1510-4.

Hikima, Y.; Akagi, Y.; Marumo, N.; and Kim, H. 2022. Online Matching with Controllable Rewards and Arrival Probabilities. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 1825–1833. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Hussein, A.; and Khamis, A. 2013. Market-based approach to Multi-robot Task Allocation. 69–74.

Kara, I.; and Bektas, T. 2006. Integer linear programming formulations of multiple salesman problems and its variations. *European Journal of Operational Research*, 174(3): 1449–1458.

Khamis, A.; Hussein, A.; and Elmogy, A. 2015. *Multi-robot Task Allocation: A Review of the State-of-the-Art*, volume 604, 31–51. ISBN 978-3-319-18299-5.

Lesmana, N. S.; Zhang, X.; and Bei, X. 2019. *Balancing Efficiency and Fairness in On-Demand Ridesourcing*, volume 32. Curran Associates, Inc.

Mehta, A. 2013. Online Matching and Ad Allocation. *Found. Trends Theor. Comput. Sci.*, 8(4): 265–368.

Nanda, V.; Xu, P.; Sankararaman, K. A.; Dickerson, J.; and Srinivasan, A. 2020. Balancing the Tradeoff between Profit and Fairness in Rideshare Platforms during High-Demand Hours. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02): 2210–2217.

Rangriz, S.; Davoodi, M.; and Saberian, J. 2019. A novel approach to optimize the ridesharing problem using genetic algorithm. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-4/W18: 875–878.

Sumita, H.; Ito, S.; Takemura, K.; Hatano, D.; Fukunaga, T.; Kakimura, N.; and Kawarabayashi, K.-i. 2022. Online Task Assignment Problems with Reusable Resources. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5): 5199–5207.

Wang, H.; and Bei, X. 2022. Real-Time Driver-Request Assignment in Ridesourcing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4): 3840–3849.

Wang, Y.; Zhao, C.; and Xu, S. 2020. Method for Spatial Crowdsourcing Task Assignment Based on Integrating of Genetic Algorithm and Ant Colony Optimization. *IEEE Access*, 8: 68311–68319.