A GA-guided Trial-based Heuristic Tree Search Approach for Multi-Agent Package Delivery Planning

Bernardo Sata, Jérôme Lacan, Caroline Ponzoni Carvalho Chanel

A GA-guided Trial-based Heuristic Tree Search Approach for Multi-Agent Package Delivery Planning

Bernardo Sata, Jérôme Lacan, Caroline P. C. Chanel
ISAE-SUPAERO, Université de Toulouse, France
bernardo.sata@student.isae-supaero.fr, jerome.lacan@isae-suparo.fr, caroline.chanel@isae-supaero.fr

Abstract

A multitude of planning and scheduling applications have to face constrained time deadlines while proposing appropriate policy solutions under uncertainty. An example of that, is the last mile delivery problem, in which a large fleet of drones needs to be managed in a broad urban area to efficiently deliver packages in response of immediate known requests and future likely requests. This application case can be seen as a sequential decision-making problem under uncertainty asking for a good solution in a constrained time deadline. In this context, this work proposes to approach a delivery policy using a combination of the Trial-based Heuristic Tree Search (THTS) and a Genetic Algorithm (GA). Specifically, during policy search trials, the GA is used as a meta-heuristic function inside the THTS paradigm to suggest the most cost-promising actions concerning drone-request immediate allocation given the current set of requests. Then, the THTS algorithm exploits the GA suggested actions and likely requests arrivals to generate only relevant branches in the tree. It enables to concentrate the search around those actions, breaking-out the inherent combinatorial nature of this planning problem. To evaluate the proposed approach a full size implementation of the aforementioned structure was built for different problem sizes, and compared to a non GA-guided THTS algorithm in order to assess its execution time and expected value performance. The results suggest this simple yet effective approach is a promising venue to fast achieve sub-optimal but reasonable cost solutions.

Introduction

In several planning and scheduling applications the time to produce solutions is constrained. In the particular case of the management of a network of small companies working in the last mile delivery such time-constraint applies. They would share their resources (such as the their vehicles) in order to increase their efficiency and serve a wider range of customers. The particular problem of managing a swarm of vehicles for last mile delivery can be described as a multi-agent planning problem (Choudhury et al. 2020). In this problem, we are looking for the most promising way to allocate each received delivery request to a specific vehicle and perform these deliveries within a certain order.

On the other hand, the planning paradigm changes when one needs to take into account future and uncertain requests in a given time horizon and to choose the most efficient way to allocate and deliver the packets with the available resources. This can be seen as a sequential decision making under uncertainty problem. This planning problem can be modeled as a Markov Decision Process (MDP) (Kolobov 2012). In a such sequential decision-making problem, efficient allocation choices should be done with the aim of minimizing the sum of the time needed to deliver actual requests and the expected sum of the time to deliver likely new future requests. The main difficulty lies in the need of finding a solution with limited time budget for a problem whose complexity grows factorial with number of likely requests and available vehicles.

In order to target such a challenge, we propose an algorithm to approach a delivery policy using a combination of the Trial-based Heuristic Tree Search (THTS) (Keller and Helmert 2013) and a Genetic Algorithm (GA). The GA generates candidate actions reflecting drone-request immediate allocations for the THTS multi-agent planning, which in turn reasons under uncertainty regarding future requests arrivals. The GA can be seen as a meta-heuristic function that proposes only cost-promising actions to the THTS algorithm. This way, THTS only explores (greedy) relevant branches in the tree. Interestingly, the MDP branching factor, inherent to the considered application case, is dramatically limited. It reduces both memory and the amount of trials needed to compute a relevant solution despite an increase on execution time observed in particular on small size problems.

The paper is organized as follows. In next section, related work is reviewed. After, background section presents fundamental tools and models on which this work is based. Next, the Multi-Agent Package Delivery problem is presented. The algorithm architecture section follows presenting the proposed planning approach. Experimental results are then presented and discussed. Finally, the paper concludes recalling contributions and proposing future work directions.

Related work

In the literature, some works proposed to solve the Vehicle Routing Problem (VRP) (Peng et al. 2019) and Traveling Salesman Problem (TSP) using Genetic Algorithms. Interestingly, (Peng et al. 2019) proposed an Hybrid Genetic Al-

gorithm for routing and scheduling vehicle-assisted multi-drone parcel delivery. Their algorithm is able to coordinate the complexity and the performance by jumping out of local optima with the use of Low Visit Cost Crossover algorithm (LVC). GA was shown in (Alaia et al. 2013) to be efficient for solution search in a multi-depot and multi-vehicle pickup and delivery problem even when precedence constraints are needed (e.g. trucks can load multiple packets before needing to deliver one). Even if those approaches propose reasonable solutions, they are not able to deal with uncertainty concerning new requests arrivals.

After the successful experience of AlphaGo, researchers have started investigating the use of Monte-Carlo Tree Search (MCTS) outside of the game sector, in order to solve real-life sequential decision-making problems (Edelkamp et al. 2016) under uncertainty. In the context of allocation, (Luo et al. 2019) proposed a MCTS-based pilot allocation scheme for massive MIMO networked systems where the pilot and power allocation problem was treated as a Markov Decision Process (MDP). In turn, (Li, Fu, and Xu 2019) presented a new Optimal Computing Budget Allocation (OCBA) tree policy for MCTS, where, unlike bandit-based tree policies (e.g. Upper-Confident bounds applied to Trees (UCT) (Kocsis and Szepesvári 2006)), the new policy maximizes Probability of Correct Selection (PCS) at the root.

(Runarsson, Schoenauer, and Sebag 2012) demonstrate the feasibility of using MCTS to address job-shop scheduling problems, demonstrating the great scalability of this algorithmic solution. On the other hand, (Trunda and Barták 2013) suggested an ad-hoc MCTS planner for transportation planning domains using templates of three typical operations (loading, moving, unloading). Such modelling choice is particularly useful for cargo planes in a real-life transportation planning problem. In the same year, (Keller and Helmert 2013) proposed THTS, a tree search algorithmic framework for solving Finite Horizon MDPs. They were able to extend Upper-Confident bounds applied to Trees (UCT, a MCTS-based algorithm) (Kocsis and Szepesvári 2006) to handle Full Bellman and Monte-Carlo backup functions to Partial Bellman backups. It was shown that such improvements enable THTS to achieve fast good policy solutions.

As far as the authors notice, only (Kim and Ahn 2018) suggested to join Genetic Algorithm and MCTS-based algorithm. They proposed a hierarchical GA-MCTS solution to tackle real-time video game problem. However, from their paper, it is not clear whether GA is used to propose actions to MCTS, or just to select some initial promising states. Thus, different from previous approaches, the present work proposes a hybrid THTS-like algorithm for very large branching factor MDPs, where the action space is restricted by a GA. More specifically, GA can be seen as a meta-heuristic function that proposes few promising actions to the THTS-based planner. THTS reasons under uncertainty concerning new requests arrivals. As far as the authors know, this kind of hybrid approach has never been proposed for a THTS-based algorithm to handle Multi-Agent Package Delivery Planning.

Background

Genetic Algorithms

Genetic Algorithm (GA) (Holland and others 1992) is a family of meta-heuristics based on natural genetic optimization mechanisms. GA starts by generating a set of random solutions, called chromosomes, forming a population or generation. Each chromosome in each generation is evaluated and ranked using a fitness function. As a consequence, the best solutions have higher chance to be selected to form the new population, by the process of crossover. In fact, crossover is a recombination procedure in which segments are exchanged between two pairs of chromosomes called parents, so generating a new individual. Some chromosomes also receive random mutations in order to avoid premature convergence and better explore the space of solutions. This process continues until the optimal solution is found or until a termination criteria set by the user is met.

Markov Decision Process

A finite-horizon MDP is a n-uplet $\{S, A, \mathcal{T}, R, \mathcal{N}\}$ where:

- S is a finite space of states,
- A is a finite space of actions,
- $\mathcal{N}$ is a finite sequence of integers $(0, 1, ..., N)$, representing the set of decision steps,
- $\mathcal{T} : S \times A \times S \to [0, 1]$ is a transition function specifying the probability value of the transition to the state $s' \in S$ from the state $s \in S$ and by using the action $a \in A$, such that $\mathcal{T}(s', a, s) = P(s'|s, a)$,
- $C : S \times A \to \mathbb{R}$ is a stationary cost function $r(s, a)$ that defines the costs when the agent uses the action $a \in A$ in the state $s \in S$

A Markov policy $\pi : S \times \mathcal{N} \to A$ is a function that maps states and decision steps to actions in the finite horizon case. The utility of a policy is defined as the expectation of the sum of the future rewards:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{N} c(s_t, \pi(s_t)) \mid s = s_0\right] \quad (1)$$

Solving a finite-horizon MDP stands for searching the policy $\pi^* \in \Pi$ that minimizes (resp. maximizes) the expected sum of costs (resp. rewards) for a given planning horizon (Kolobov 2012).

$$\pi^*(s) = \arg\min_{\pi \in \Pi} \mathbb{E}\left[\sum_{t=0}^{N} c(s_t, \pi(s_t)) \mid s = s_0\right] \quad (2)$$

Recent MDP planning algorithms perform Trial-based Heuristic Tree Search (THTS) in order to evaluate only reachable states following some action selection strategy, that allows to achieve relevant policy solutions in constrained time. (Keller and Helmert 2013) extends UCT to handle Full Bellman and Monte-Carlo backup functions to Partial Bellman backups. Therefore, they propose DP-UCT and UCT*. DP-UCT combines attributes and theoretical properties from RTDP (Barto, Bradtke, and Singh 1995) and UCT, whereas UCT* adds a limited trial length to DP-UCT that ensures that parts of the state space that are closer to the root are investigated more thoroughly.

MDP Model for Multi-Agent Package Delivery

In the Multi-Agent Package Delivery allocation problem each $s \in S$ is characterized by the list of current packet requests $k \in K$, and, of available vehicles $d \in D$, ($s = s(D, K)$). In particular each request k is described by its source's k_S and destination's position k_D on the map and by its weight k_W and volume k_V. Similarly each vehicle d is fully determined by its current position d_{POS}, by the maximum payload d_{MP} and maximum volume d_{MV} it can carry, by its nominal speed d_S, its average needed time to set up the package d_{SUT} and to drop it off d_{DOT}.

From each state $s \in S$ a set of possible actions $a \in A$ is available. Each $a \in A$ corresponds to a possible allocation, where each request is assigned to a certain vehicle which performs the tasks in a defined order. Moreover, from each action a, various futures states with their probability can be considered. In fact, while the vehicles perform their tasks, different possible sets of requests K can be received, so identifying different possible future states $s' \in S$. Therefore the successor state s' will be characterized by vehicles with different positions and battery level and, by the new set of received requests K'. As a consequence, for each action a, multiple successor states are present, which can happen with a specific probability $P(s'|a, s)$. For instance, such probabilities can be learnt from a history, being non stationary, and strongly dependent on time (e.g. holiday/weekday, lunchtime/nighttime). In this work, it is assumed that such a request arrival probability function is given (interested readers can find useful learning methods addressing this topic in (Swanson 2019)). In addition to this assumption, it is also assumed that the appearance of future packet requests forming K' is completely independent from the current allocation (action). Thus, future states probabilities depend only on the probability of the set K' that is a function of the previous set K, as:

$$P(s'|s, a) = P(D'|D, a) \cdot P(K'|K, a) = \\ = P(K'|K, a) = P(K'|K),$$

as $P(D'|D, a) = 1$ (it is assumed as deterministic).

The cost function is related to the overall cost of the allocation $a \in A$ at state $s \in S$. Specifically, it is set as the cost of the slowest vehicle to complete the task assigned to it:

$$c(s, a) = \max_{d \in D} c(d, k^d) \tag{3}$$

with,

$$c(d, k^d) = \sum_{k^d \in K^d} \frac{L(d,k^d)}{d_s} + d_{SUT} + d_{DOT}$$
$$\frac{L(d, k^d)}{d_s} = ||\overrightarrow{d_{POS}k_S}|| + ||\overrightarrow{k_D k_S}||$$

Note, $K^d \subseteq K$ is the subset of requests assigned to vehicle $d \in D$ and $L(d, k^d)$ is the total travel distance that vehicle d takes to complete the delivery of k^d.

Solving such a planning problem refers to finding the allocation policy that minimizes the expected sum of costs in a given planning horizon (see Eq. 2).

Discussion on the cardinality of the action set A

The amount of possible actions a for every state s can incredibly grow with the amount of vehicles available $|D|$ and requests to be fulfilled $|K|$. Although, the proposed allocation must satisfy the following property:

$$A = \left\{(d, K^d) : d \in D \wedge \bigcup_{d \in D} K^d = K\right\}$$

As a matter of fact, the problem of assigning a certain number of tasks to a set of vehicles leads to a total of:

$$\prod_{l=1}^{|D|-1} \frac{|K| + l}{l}$$

possible combinations. However, in the application case at hands, also the order of allocation plays a fundamental role. To take it into account, all possible permutations of the requests present in the set K must be considered for every assignment:

$$|K|! \cdot \prod_{l=1}^{|D|-1} \frac{|K| + l}{l} = \\ = \frac{(|K| + |D| - 1)!}{(|D| - 1)!}$$

As shown in the Tables 1 and 2, this leads to an incredibly large cardinality of A. As a consequence the MDP will have a huge branching factor, so an enormous amount of leaf nodes even if only few epochs are considered when planning. Such a tree requires a lot of memory and time in order to be explored, therefore an hierarchical approach has been proposed.

		$\|D\|$		
		3	**4**	**5**
$\|K\|$	**2**	12	20	30
	3	60	120	210
	4	360	840	1680
	5	2520	6720	15120
	6	20160	60480	151200
	7	181440	604800	1663200
	8	1814400	6652800	19958400

Table 1: The number $|A|$ in function of the number of vehicles $|D|$ and requests $|K|$.

In this sense, a meta-heuristic which is able to return a certain number of greedy actions (e.g. promising allocations) can be used to reduce the complexity of the problem while maintaining a certain quality of solution. In the following, the proposed GA-guided THTS-like algorithm is presented.

Algorithm Architecture

The GA-guided THTS-like algorithm can be seen as a cascade approach, where a global **N-step** planning algorithm explores the MDP tree by calling a **one-step** allocation algorithm to get the most promising actions (e.g. greedy actions) in every reachable state. The Figure 1 schematizes the algorithm architecture.

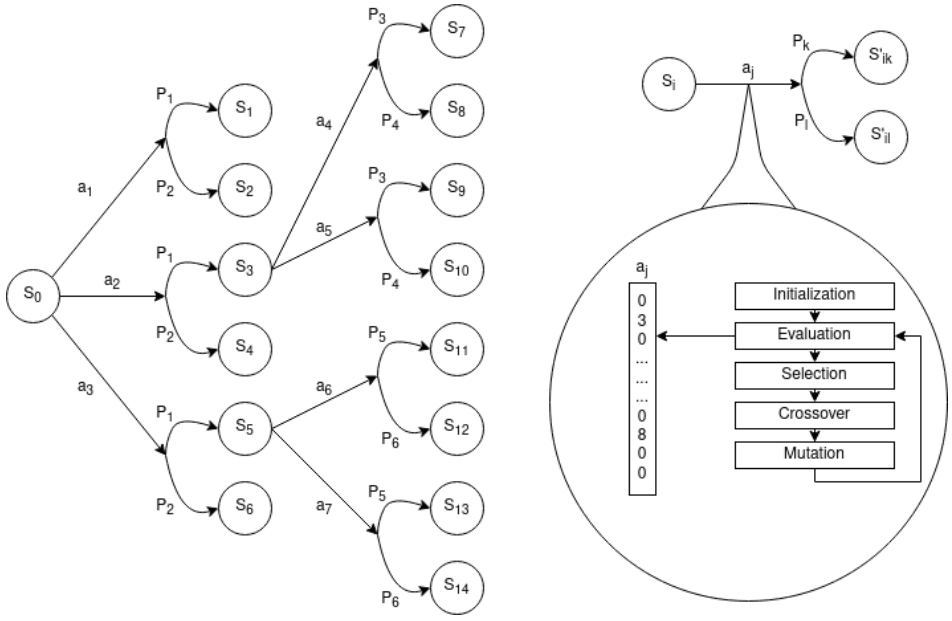

Figure 1: GA-guided THTS-like algorithm architecture. The GA (on the right) is used to generate *greedy* actions regarding allocation costs. Based on those actions, DP-UCT (on the left) minimizes the long-term (N-steps) cost expectation to define the best allocation policy.

		$\|D\|$		
		3	**4**	**5**
$\|K\|$	**3**	1.12E+07	1.15E+08	7.74E+08
	4	3.11E+09	4.95E+10	4.86E+11
	5	1.34E+12	3.11E+13	4.27E+14
	6	8.40E+14	2.73E+16	5.06E+17
	7	7.37E+17	3.24E+19	7.88E+20

Table 2: Number of leaf nodes considering 3 steps and 3 possible future sets. $|K|$ decreases by 1, remains the same, and increases by 1.

One-Step Allocation

In order to compute the most promising allocations a Genetic Algorithm (GA) is used. At this level, no uncertainty is considered as the current requests and available vehicles are given as inputs from the current state s. The GA will build a list of *chromosomes*, called *population*, which encode the allocations.

In particular, during **initialisation**, each *chromosome* is a vector of zeros of size $|D| \times |K|$. Each request $k \in K$ is linked to a number (from 1 to $|K|$), and is randomly written instead of a zero in a chromosome. Then each chromosome can be processed by the GA as $|D|$ consecutive sub-vectors of size $|K|$, representing each vehicle $d \in D$. This allows to read the non zero values as specific requests assigned to a specific vehicle. For example, a possible chromosome for $|D| = 2$ and $|K| = 3$ is $[2, 0, 1, 0, 3, 0]$ representing the case where the first vehicle should first perform the second request and then the first one, while the second vehicle just needs to complete the third request. This kind of representation allows to compact all information in a single array, but at the same time leads to the possibility of having different chromosomes for the same actual allocation.

In the **evaluation** phase, the chromosomes are then ranked following the *fitness* function that is the one defined in Eq. 3 (e.g. lower cost solutions are better). The best performing chromosomes are the ones that present low allocation costs. During evaluation, also the specific constraints of weight and volume are taken into account. Specifically it is checked that for every packet k assigned to a drone d: its weight is smaller than the drone's maximum payload ($d_{MP} \geq k_W$), and that its dimensions do not exceed the drone's maximum volume ($d_{MV} \geq k_V$). If these conditions are not met, the gene is evaluated with a very high cost. As explained later, this allows to reduce the probability of further exploring that part of the solution space and to discard these high cost solutions at the algorithm's termination.

During **crossover**, the whole population is regenerated, while conserving a part of the best performing chromosomes of the previous one. In particular, the crossover consists in copying in a new chromosome the elements of a parent chromosome whose index is lower than a random sampled number and appending in order the unrepeated elements from the other parent. This last step ensures that all requests are present within the new chromosome and preserves its correct size.

The **mutation** step is randomly chosen among two different processes. The first simply consists in swapping two elements in the chromosome. The second inverts the order of the elements contained between two random indexes.

Finally the GA is stopped when the best chromosome has had the same values for more than a given number of cycles or if the time limit expires. At this point a certain amount of the most performing allocations (e.g. promising actions),

which are the chromosomes which are characterized by the lowest cost, are returned by the algorithm.

These allocation solutions (best chromosomes) will form the set of applicable actions $App(s) \subseteq A$ in a given state $s \in S$, corresponding to the set of actions to consider in the N-step planning algorithm at state s. Therefore, within a certain time limit, a predefined number of allocations is generated. Note that selecting only the best obtained result would constrain the exploration in the global MDP solver to a single possible action per explored state. Thus selecting promising actions does not constrain the N-step planning algorithm while limiting the MDP branching factor. For simplicity, the GA implementation does not allow subsequent packet set up or drop off, which is a reasonable choice in case of vehicles such small drones. Note that a similar GA was used also for truck scheduling as shown in (Alaia et al. 2013).

N-step planning

The N-step planning allows to explore the MDP model through a Trial-based Heuristic Tree Search algorithm (THTS) framework. This framework is an online planner that enhances performances as the environment considered strictly depends on time. The proposed algorithm receives the most promising actions from the "One-step allocation" algorithm and the probability distributions for the expected future packet requests. For each allocation and for the most likely future requests it will build a new successor state s' with respectively a specific list of vehicles D' and of current requests K'. In other words, the amount of possible future states is huge (possibly infinite), therefore only the most likely ones are considered, defining the set of successors states such as $Succ(s)$. This property exploits the independence between the allocation and the packet requests' appearance ($P(s'|s,a) = P(K'|K,a) = P(K'|K)$ as previously discussed). As a consequence, a partial Bellman buckup function (Keller and Helmert 2013) that does not rely on all successor nodes in the MDP is a sound solution to be applied in order to well estimate action and state values. This can be achieved by weighting the outcomes that have been taken into account for the tree proportionally to their (normalized) probability. The immediate cost of each action is given by Eq. 3.

By the way, the THTS-based framework here proposed consists in three phases executed repeatedly until the time limit is reached or until a convergence criterion is met. As displayed in Figure 2, the three main steps are:

Selection: in this phase, a selection strategy is recursively called until a leaf node is reached. In the current implementation, a greedy-action selection strategy is used, like in RTDP (Barto, Bradtke, and Singh 1995) or (L)RTDP(Bonet and Geffner 2003) algorithms. Using the partial backup function, it selects the action presenting the best Q-value approximation, such as:

$$\hat{a} \leftarrow \underset{a \in App(s)}{\arg\min}\, \hat{Q}(s,a)$$

where,

$$\hat{Q}(s,a) = c(s,a) + \frac{\sum_{s' \in Succ(s)} P(s'|s,a)\hat{V}(s')}{\sum_{s' \in Succ(s)} P(s'|s,a)}$$

Expansion: in this phase, the actions proposed by the One-step allocation algorithm (e.g GA) are integrated into the tree followed by their successors nodes. In the current implementation an heuristic function is used to first approximate the initial state value $\hat{V}(s)$ of those new nodes.

The heuristic used in this case exploits one specific property of any multi-agent delivery problem. In fact, among all $k \in K$, if the longest one (in terms of distance between source and destination) is taken into consideration, there is no possible allocation solution where the maximum time cost for the global system is lower than the time needed by the fastest vehicle to perform the aforementioned task. Thus, this admissible heuristic *h(s)* for a given state s is given as:

$$h(s) = \min_{d \in D} \frac{L(d,\bar{k})}{d_s} + d_{SUT} + d_{DOT}$$

where, $\bar{k} = \arg\max_{k \in K} ||\overrightarrow{k_D k_S}||$.

Backpropagation or Backup: in this phase the value estimated of the new state is backpropagated through the tree up to the root state node. It enables to update all parent's values since the leaf, and by consequence also update the policy. The update strategy selected in this algorithm is the same applied by the DP-UCT algorithm (e.g. partial Bellman backup function).

The state value function $V(s)$ is then updated as:

$$\hat{V}(s) = \begin{cases} \min_{a \in App(s)} c(s,a) & \text{if } s \text{ is a terminal state} \\ \min_{a \in App(s)} \hat{Q}(s,a) & \text{otherwise} \end{cases}$$

where the Q-value is:

$$\hat{Q}(s,a) = c(s,a) + \frac{\sum_{s' \in Succ(s)} P(s'|s,a)\hat{V}(s')}{\sum_{s' \in Succ(s)} P(s'|s,a)}$$

It is interesting to point out an interesting characteristic of such algorithm. As a matter of fact this algorithm uses Partial Bellman backups rather than Monte-Carlo Backups. This leads to a combination of properties of Dynamic Programming and MCTS (see (Keller and Helmert 2013)).

One may note that this mechanism enables to implement state labeling (like in (L)RTDP (Bonet and Geffner 2003) to inform when the value has converged in a given state) and thus to set a termination condition when the root node is solved. This is possible thanks to the fact that the backup function considers the probabilities of the future events during the computation. This is specially important when comparing the quality of the solutions of the sub-action set case (by restricting the action space using GA one-step solutions) and the full action space case. As a matter of fact, simply limiting the computation time might cause one of the aforementioned approaches to have a lower cost just because the MDP was not sufficiently explored. Similarly, checking if

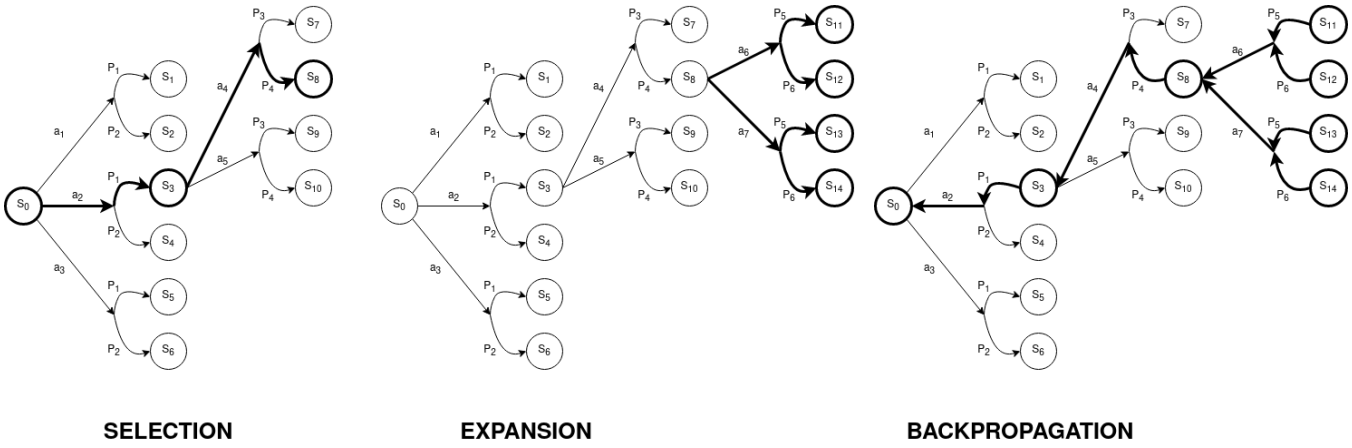

Figure 2: GA-guided THTS algorithm steps illustration. In the Selection phase the algorithm greedily selects an action among a set of promising actions given by the GA until it reaches a leaf state node. In the Expansion phase new GA solutions are computed for this leaf node generating the set of applicable actions, and new leaf state nodes are added to the tree along with their initial heuristic value $h(s)$. Finally, in the backpropagation phase, partial Bellman backup is used - updating states values and the best current policy - followed by the labeling procedure.

the root state value has not changed too much in the last trials is also not feasible as its variation can be very slow and directly depend on the input characteristics, obliging to tune it for each solving launch. In this sense, the use a labeling function (just like (L)RTDP (Bonet and Geffner 2003)) in order to stop the planning once the root state value has converged seems a reasonable option. The current implementation of the GA-guided THTS implements such labeling mechanism.

As a consequence two algorithmic approaches, one considering the full action space, and the other one considering only the most promising solutions proposed by the GA - both following the (L)RTDP structured and performing the value update using the partial Bellman backup are compared in the following section.

Experiments

Experiments were performed in order to assess the performances of the proposed GA-guided THTS-based algorithm when compared to a classical THTS-based MDP solving approach in our application case. As previously stated, the main idea of the proposed algorithm is to restrict the action space A of an MDP using the GA one-step allocation solutions ($App(s) \subseteq A$). Therefore a first experiment shows how costly this GA meta-heuristic is, when restricting the action space since the dimension of the problem increases (in this particular case for large vehicles' and requests' set cardinality); whereas the second and third experiments compares the full action space tree exploration against the proposed algorithmic approach concerning initial state value convergence and policy quality, computation time and memory consumption, respectively.

Setup

The first experiment goal is to show that looking for the whole action space is impracticable for any true system. For this, ten simulations were performed with $|D| = 2$ and $2 \leq |K| \leq 8$ and with different initial parameters (e.g. vehicles location, requests drop locations, etc). It is clearly not representative of a real life scenario of a crowded city. However, this experiment aims to illustrate how costly it can be to just list all the possible actions in a given state.

The second experiment aims to compare the proposed approach with an MDP classical THTS solution when the full action space is investigated in terms of value convergence. As no real convergence criterion exists in the THTS framework, it led us to develop a labeled version of the aforementioned algorithm following the same labeling procedure of (L)RTDP. Note it still uses partial Bellman backup (as DP-UCT) and still defines a fixed horizon (so resembling to UCT*). These mechanisms allows to assess whether the algorithm has converged by just considering the label assigned to the root state node. Therefore the tree selection-exploration-backup strategies are common. The only difference would stand in the applicable actions set generation: considering all possible actions or selecting the most promising action sets with GA (with different sizes $|App(s)| = \{2, 4, 8, 16\}$ and $App(s) \subseteq A$). Nevertheless, such an experiment is run once with a very small case scenario. This choice is mainly due to memory constraints while building the full tree. The experiment used $|D| = 2$ and $|K| = 3$ while generating three possible future states for each action with respectively $|K| = 2$, $|K| = 3$ and $|K| = 4$. The planning horizon considered is $N = 3$.

The third experiment aims to analyse the average solution quality, average execution time and average memory consumption given different problems. Thus, ten different problems with $|D| = 2$ and $|K| = 3$, three possible futures with respectively $|K| = 2$, $|K| = 3$ and $|K| = 4$, a

considered planning horizon of $\mathcal{N} = 3$, although with different initial parameters are considered. A run was launched for each of them to compute: a solution policy considering the full action space and solution policies using the proposed GA-guided THTS approach with a different number of GA promising actions set size in each state ($|App(s)| = \{2, 4, 8, 16\}$ and $App(s) \subseteq A$). Then, the average ratio between the expected value of the root node achieved with the GA-guided approach over the expected value of the root node achieved with the full action space was calculated. In the same vein, the average ratio of execution time and the average ratio of memory consumption were saved.

Results

The plot presented in Figure 3 concerns the results obtained in the first experiment. It depicts how the time needed to list all actions is much lower when the action space is little. This result means that the GA meta-heuristic would require much more time to complete its execution than the full action listing, so affecting the convergence time. In particular the GA would be respectively around 925, 252 and 52 times slower than only listing actions for the three possible futures. However, the GA meta-heuristic scales up much better once the problem dimension increases. Moreover, the GA algorithm was always able to find a set of promising actions in all the runs for the given inputs.

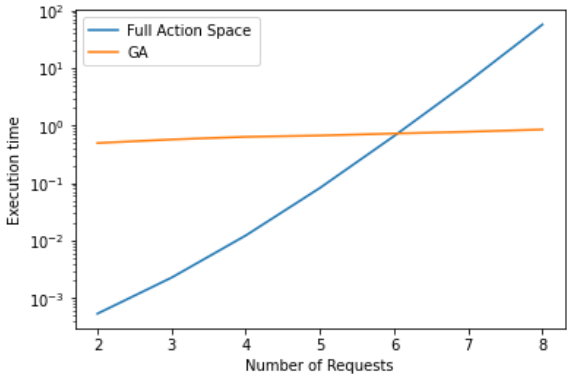

Figure 3: GA run execution time against and full action space listing time analysis.

The results obtained in the second experiment are showed in Figure 4 which depicts the root state value node in every exploration trial (e.g. initial state value convergence evolution). This result shows how the algorithm using GA one-step allocation solutions has a much rapid convergence in value with respect to trials performed. However, it converges towards a sub-optimal solution as expected. Anyway, the solution cost error is on average less than 3% higher than when the full action space is considered. More specifically, the value error is 3.008%, 2.956% 2.605% and 3.289% for $|App(s)|$ equal to 2, 4, 8 and 16, respectively.

The Table 3 resumes the results obtained in the third experiment. The average execution time ratio, the average initial state value ratio and the average memory consumption

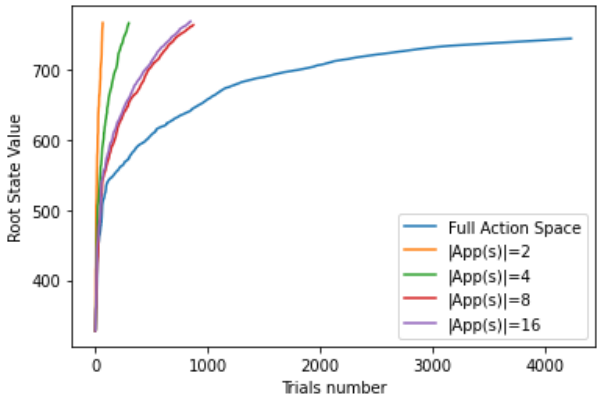

Figure 4: Initial state value convergence for the GA-guided THTS approaches compared to a full action space THTS over trials.

ratio are presented. Recall that the full action space is the common baseline. It can be seen that execution time increases linearly in the proposed approach with the number of GA solutions being considered at each state ($|App(s)| = 2, 4, 8, 16$). For $|App(s)| = 8$ or $|App(s)| = 16$, the execution time ratio is 12 and 14 times bigger, respectively. It is mainly due to the fact that GA takes 52, 252 or 925 times more time (see Fig. 3) depending if one is considering 4, 3 or 2 requests. In other words, the tree expansion phase that, in turn depends on actions generation, is time costly in the particular case of small problems. That said, the previous experiments showed that the GA-guided THTS approach scales up to treat more vehicles and requests while taking fewer trials to reach a reasonable solution. Whereas, the proposed algorithm is only 14 times slower, while getting 50 times less memory. Concerning the solution quality measure, the average initial state value ratio is almost one ($\approx 3\%$ error). It shows that the expected sum of costs and policy achieved gives a near-optimal solution. The average initial state value ratio seems to increase when 16 GA solutions are considered. At this moment, the authors are not able to explain this result, speculating that it may be related to the the number of runs (i.e. 10) used to compute averages that should not be enough to erase the bias relative to these sampling-based technique. As long as memory is concerned, the full tree exploration required approximately 0.5 Gbytes in average, while the GA one used 50 times less memory in the worst case ($|App(s)| = 16$) in average (see Table 3). This amount of memory is increasing factorially with the cardinality of the requests and vehicles sets. In any case, it would be interesting to find the best amount of promising actions from the GA to maximize policy quality while sacrificing the least possible memory and execution time.

Conclusion and Future Work

This work proposes a novel approach to compute a Multi-Agent Package Delivery policy using a combination of the Trial-based Heuristic Tree Search (THTS) and a Genetic Algorithm (GA). Specifically, during policy search trials, the

$\|App(s)\|$	Avg. Execution Time Ratio	Avg. Initial State Value Ratio	Avg. Memory Consumption Ratio
$\|App(s)\| = 2$	0.91667	1.0386	0.00012917
$\|App(s)\| = 4$	4.0885	1.0313	0.0027340
$\|App(s)\| = 8$	12.206	1.0257	0.016762
$\|App(s)\| = 16$	14.126	1.0330	0.021463

Table 3: Ten runs average execution time ratio, initial state value ratio and memory consumption ratio. The common basis is the result obtained when the full action space is evaluated.

GA is used as a meta-heuristic function inside the THTS paradigm to suggest the most cost-promising actions concerning drone-request immediate allocation given the current set of requests. It restricts the actions to a subset of the full action space. Then, THTS algorithm exploits the GA proposed actions and likely requests arrivals to generate only relevant branches in the tree. It enables to concentrate the search around those actions breaking-out the inherent combinatorial of this planning problem. Empirical results demonstrated the algorithm using GA-guided THTS convergences faster in value, however, towards a sub-optimal solution. Anyway, given the average value error observed in experiments ($\approx 3\%$), the proposed approach seems to be a reasonable solution to deal with Multi-Agent Package Delivery planning problems with regards to execution time and memory consumption.

An immediate future work would be to check if the fluctuations observed in the error on the average of the expected values decreases if the number of runs increases. A mid-term work would be to compared the GA-guided THTS approach with others state-of-the-art techniques to check its scalability and solution quality on real-life Multi-Agent Package Delivery problems. And finally, a long-term future work could address an automated technique that finds the good amount of GA solutions to be kept in the THTS algorithm during policy optimization.

References

Alaia, E. B.; Dridi, I. H.; Bouchriha, H.; and Borne, P. 2013. Optimization of the multi-depot & multi-vehicle pickup and delivery problem with time windows using genetic algorithm. In *2013 International Conference on Control, Decision and Information Technologies (CoDIT)*, 343–348. IEEE.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial intelligence* 72(1-2):81–138.

Bonet, B., and Geffner, H. 2003. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *ICAPS*, volume 3, 12–21.

Choudhury, S.; Solovey, K.; Kochenderfer, M. J.; and Pavone, M. 2020. Efficient large-scale multi-drone delivery using transit networks. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 4543–4550.

Edelkamp, S.; Gath, M.; Greulich, C.; Humann, M.; Herzog, O.; and Lawo, M. 2016. Monte-carlo tree search for logistics. In *Commercial Transport*. Springer. 427–440.

Holland, J. H., et al. 1992. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon mdps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 23.

Kim, M.-J., and Ahn, C. W. 2018. Hybrid fighting game ai using a genetic algorithm and monte carlo tree search. In *Proceedings of the genetic and evolutionary computation conference companion*, 129–130.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293. Springer.

Kolobov, A. 2012. Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6(1):1–210.

Li, Y.; Fu, M.; and Xu, J. 2019. Monte carlo tree search with optimal computing budget allocation. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, 6332–6337. IEEE.

Luo, S.; Zhang, C.; Duan, Y.; and Chen, J. 2019. Pilot allocation game: A monte carlo tree based method. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, 277–282. IEEE.

Peng, K.; Du, J.; Lu, F.; Sun, Q.; Dong, Y.; Zhou, P.; and Hu, M. 2019. A hybrid genetic algorithm on routing and scheduling for vehicle-assisted multi-drone parcel delivery. *IEEE Access* 7:49191–49200.

Runarsson, T. P.; Schoenauer, M.; and Sebag, M. 2012. Pilot, rollout and monte carlo tree search methods for job shop scheduling. In *International Conference on Learning and Intelligent Optimization*, 160–174. Springer.

Swanson, D. 2019. A simulation-based process model for managing drone deployment to minimize total delivery time. *IEEE Engineering Management Review* 47(3):154–167.

Trunda, O., and Barták, R. 2013. Using monte carlo tree search to solve planning problems in transportation domains. In *Mexican International Conference on Artificial Intelligence*, 435–449. Springer.