A method for real-time dynamic fleet mission planning for autonomous mining

Autonomous Agents and Multi-Agent Systems (2019) 33:564–590
https://doi.org/10.1007/s10458-019-09416-y

A method for real-time dynamic fleet mission planning for autonomous mining

Mattias Wahde[1] · Mauro Bellone[1] · Sina Torabi[1]

Published online: 12 June 2019

Abstract
This paper introduces a method for dynamic fleet mission planning for autonomous mining (in loop-free maps), in which a dynamic fleet mission is defined as a sequence of static fleet missions, each generated using a modified genetic algorithm. For the case of static fleet mission planning (where each vehicle completes just one mission), the proposed method is able to reliably generate, within a short optimization time, feasible fleet missions with short total duration and as few stops as possible. For the dynamic case, in simulations involving a realistic mine map, the proposed method is able to generate efficient dynamic plans such that the number of completed missions per vehicle is only slightly reduced as the number of vehicles is increased, demonstrating the favorable scaling properties of the method as well as its applicability in real-world cases.

Keywords Multivehicle trajectory planning · Genetic algorithms · Mining · Real-time planning

1 Introduction

Industry is currently fostering investments in robotics and artificial intelligence in order to solve safety problems in mining operations, and to increase efficiency, with a potential of generating billion-dollar savings. Indeed robotic machines are becoming more and more common in modern mines [1,2] and the trend is going toward virtualization [3], underground vision [4], and remote control [5]. Among the many challenges involved in mining operations,

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10458-019-09416-y) contains supplementary material, which is available to authorized users.

✉ Mattias Wahde
mattias.wahde@chalmers.se

Mauro Bellone
mauro.bellone@chalmers.se

Sina Torabi
sina.torabi@chalmers.se

[1] Department of Mechanics and Maritime Sciences, Chalmers University of Technology, Göteborg, Sweden

planning the coordinated motion of an entire fleet of vehicles is one of the most important [6], involving issues such as decision-making, path planning, and scheduling. [7].

The main contribution of this work is a novel method, based on a (modified) genetic algorithm, for efficient, centralized real-time dynamic generation of collision-free trajectories for an arbitrary number of vehicles operating in a realistic mining scenario. The principal case considered here will be that of an underground mine, in which a set of fully autonomous dumpers operates, with the task of picking up material (e.g. iron ore) from loading stations deep inside the mine, and then transferring the material to an offloading station. The effectiveness of the method is first demonstrated in a simpler static case, by means of a statistical analysis over 10,000 runs. The proposed method is compared against two classical benchmark methods in terms of mission complexity (number of segment traversals), and is shown to compare favorably with those methods.

In this paper, the planning involves both determining a vehicle's route *and* the duration of its motion along the route, a unit that will be collectively referred to as the *trajectory* of a vehicle. The planning algorithm introduced in this paper operates on topological maps, rather than metric maps. With topological maps, one can logically separate the problem of low-level vehicle control, i.e. applying control actions in order to follow a specific path, from the problem of planning the trajectory of the vehicle. Since the planning is topological, the algorithm does not specify the speed variation within a given segment (defined below). Instead, it simply specifies the total duration, which should be bounded from below by the minimum possible duration for traversing the segment in question.

The performance in the dynamic case, with frequent re-planning, is then demonstrated through a series of runs in which indicators, such as the number of traversals (per vehicle) and vehicle idle time, were measured over an increasing number of vehicles, resulting in the conclusion that the method displays good scaling properties, with only a small drop in per-vehicle performance as the number of vehicles is raised significantly.

The structure of the paper is as follows: in Sect. 2, a brief review of relevant works in the field is reported. Section 3 introduces the reader to the concepts of topological maps, missions, and fleet missions. Then, in Sect. 4 the simpler (but essential) case of *static* planning is described, and is followed, in Sect. 5, by a description of the dynamic case. The results are given in Sect. 6, and are followed by a discussion in Sect. 7 and the conclusion in Sect. 8.

2 Related work

In order for a mine to operate autonomously, two crucial aspects must be considered: namely (i) multi-vehicle path planning and (ii) dynamic scheduling. The former part deals with the generation of conflict-free paths, whereas the latter is concerned with minimizing delays and waiting times. With the increasing interest in autonomous mining, combined with improvements in computing capabilities, the full dynamic problem, *combining* path planning and scheduling, has recently attracted more attention.

2.1 Path planning

Considering first the problem of path planning in underground mining, the two most common approaches are planning with vehicle constraints [8] and planning on topological maps [9]. The use of vehicle constraints addresses the problem in the most complete way as it considers the generation of kinematically or dynamically feasible trajectories. On the other

hand, planning on topological maps unloads computational burden leaving the problem of navigation to the low level control, resulting in fast resolution. For this reason, most of the common approaches involving multiple vehicles tackle the problem using topological maps [10,11]. For topological maps, a method referred to as push-and-swap was introduced in [12]; see also Sect. 6.1.1 below.

Since the vehicles share the same environment, their coordination [13] and cooperation [14] is central when solving the planning problem. The methods presented in [14, 15], and [16] are examples of multi-vehicle path planning algorithms that use artificial intelligence-based approaches (primarily reinforcement learning), to solve the problem of planning in multi-vehicle systems, demonstrating strong scalability properties [16].

Based on the well known A* algorithm [17] that computes the optimal path for a single vehicle in a graph, in [18] an implementation of subdimensional expansion, called M*, is presented that adapts A* to be used efficiently for multivehicle planning. Since its introduction, many variants of M* have been proposed [19,20] dealing efficiently with the problem of path finding in a graph; see also Sect. 6.1.2 below. An alternative approach can also be found in [21], in which the authors proposed a so called conflict-based search algorithm. This algorithm is efficient in dealing with multivehicle path planning problems using a division between a high-level algorithm and a low-level algorithm. The low level finds the optimal path for each agent, and the high level only solves conflicts step-by-step by expanding a tree of possible solutions until a feasible one is found. Some variations of this algorithm improve performance, for example by grouping agents to reduce the number of conflicts [22], or designing the optimal assignment of targets for the agents [23]. In summary, algorithms of this kind require exploration of the solution space looking for alternative solutions every time a conflict is found.

In general, the complexity of the planning problem grows very fast with the size of the map and the number of vehicles. Hence, rather than attempting an exhaustive search, it is common to approach the problem using other search methods such as, for example, random trees [24] or genetic algorithms [25].

In the approach used here, the planning procedure starts with vehicles being assigned the individual shortest path from its start node to its end node (a path that can be computed once and for all, for any node pair in a given map, see also Sect. 3.1 below). These paths are then modified as required to meet the demands of conflict-free dynamic scheduling, described next.

2.2 Dynamic scheduling

It is very important to remark that, while the approach proposed here does handle path planning, our paper deals primarily with dynamic scheduling, solving the conflicts (i.e. avoiding collisions) efficiently, while attempting to minimize delays and waiting times. It does so using a modified genetic algorithm, thus crucially avoiding to explore the entire solution space every time a conflict is found. The dynamic fleet mission planning algorithm presented here is designed to fulfil tight time constraints in the loading and unloading of ore in a mine, also handling the fact that the required duration for those operations may vary.

A similar, but not identical, problem is addressed in a very recent work [26], in which the authors investigate the dynamic pickup and delivery problem. In that paper, a fleet of vehicles must transport customers from any starting point to their destination [27]. The problem is addressed using the so-called contract-net protocol, in which each customer publishes its transportation problem in a public auction, and each agent offers a price for the transportation

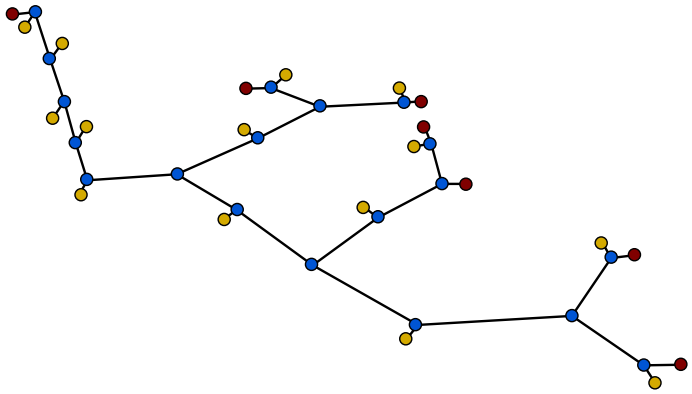

Fig. 1 The topological mine map used for dynamic planning. The red node in the upper left corner is the (prioritized) offloading site, and the remaining red nodes are the loading sites. All these nodes are collectively referred to as terminal nodes. All other nodes are either transit nodes (shown in blue), i.e. nodes at which a vehicle has a choice between two or more different future paths, or pause nodes (shown in yellow) where a vehicle can stop temporarily in order to let another vehicle pass. The vertical part in the left side of the map represents a steep, spiralling corridor leading down to the mining level, where several branches lead to the various loading sites (Color figure online)

service (moving the customer from point A to point B) according to the actual distance and traversal duration in the graph. The problem is highly complicated due to the tight scheduling and the complex bidding system. To solve the planning problem the authors use the M* algorithm, but the resolution of the *scheduling* problem requires long training in a cluster with multiple processors in parallel. In contrast, our algorithm does not require any offline training, but the calculation of all the optimal (shortest) paths between all node pairs, stored in a path matrix, is required. This operation need only be done once and for all for a given map, and takes at most a few seconds for a typical mine map.

3 Representations

This section introduces three central concepts for the planning problem considered here. First, topological maps are introduced, and then the concepts of missions and fleet missions are described.

3.1 Topological maps

The topological map is defined as a bidirectional graph $\mathcal{T} = \{\mathcal{P}, \mathcal{E}\}$, where $\mathcal{P} = \{p_1, p_2, \ldots p_n\}$ is the set of nodes (vertices) and $\mathcal{E} = \{e_1, e_2, \ldots e_m\}$ is the set of segments (edges). Every segment $e_j = (p_s, p_e, t_j)$ is fully specified by a start node p_s, an end node p_e, and the traversal time (cost) t_j. Note that the maps considered here do not contain loops.

Furthermore, $\mathcal{P}$ is the union of three disjoint subsets, that is $\mathcal{P} = T \cup S \cup Q$, where T is the set of terminal nodes, S is the set of pause (swap) nodes, and Q is the set of transit nodes. The nodes types are illustrated in Fig. 1. The set of terminal nodes (T) is further subdivided into two disjoint subsets, namely the *prioritized terminals* T^{P} and the *non-prioritized terminals* T^{NP}. As will be described below, a fully loaded vehicle, en route to the offloading station, should

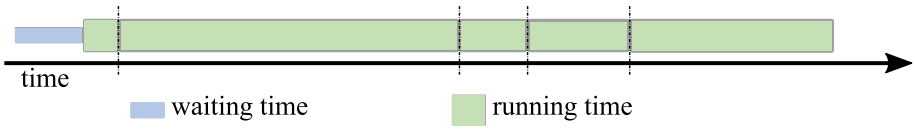

Fig. 2 An example of a mission, consisting of five mission items. In principle, each mission item consists of an initial stop time (indicated by a light-blue box) followed by the traversal (indicated by a green box) of a map segment. However, in the mission shown here, only the first mission item, where the vehicle starts at a terminal node in the map, has a non-zero initial stop time. See also Fig. 3 below (Color figure online)

generally be given higher priority than vehicles that are currently empty. Thus, offloading sites are prioritized, whereas all other terminals (i.e. the loading sites) are non-prioritized. In this paper, for the dynamic fleet mission planning (see Sect. 5) it will be assumed that there is a single prioritized terminal, but this restriction can easily be lifted (and has indeed been lifted for the static planning case described in Sect. 4). Having a single prioritized terminal is representative of the structure of many real mines, in which vehicles returning from the underground loading sites have to reach a single offloading site (where, for example, the material is crushed before being moved out of the mine), resulting in a severe bottleneck that, evidently, must be considered when planning trajectories. As indicated in Fig. 1, the severity of this bottleneck is somewhat alleviated by the presence of pause nodes *near* the offloading site, where incoming loaded vehicles can (and very often must) wait until the vehicle occupying the offloading site has been emptied and can then leave that site.

Given a topological map one can compute, once and for all (unless the map is changed) all possible paths within the map. An algorithm was written that, for all pairs (p_i, p_j), $i \neq j$ of distinct nodes, generates the shortest path (i.e. the direct path without any pause node visits), denoted $\Pi_0(p_i, p_j)$, and stores all these paths in a data structure henceforth referred to as the *path matrix*.

3.2 Missions

The topological map is populated with a set of L vehicles $\mathcal{V} = \{v_1, v_2, \ldots v_L\}$. For each vehicle v_k, one can define a *mission* $\mathcal{M}(v_k) = \{m_1, m_2, \ldots, m_q\}$ as a set of *mission items*. A mission item, in turn, consists of a segment e_i and an *initial stop time* t_w (which is non-zero only in some special cases; see below) specifying an initial standstill period before movement begins over the segment in question. The start node of the first mission item, and the end node of the last mission item, belong to the set of terminals.

When missions are initialized (before optimization), the path for any vehicle is taken from the path matrix (see above). Then, during optimization, missions are modified by inserting (or removing) pairs of mission items representing pause node visits, as described in Sect. 4.3 below. Missions are illustrated graphically as shown in Fig. 2.

3.3 Fleet missions

A *fleet mission*, denoted $\mathcal{F}$, consists of a set of missions, one for each vehicle,[1] that is $\mathcal{F} = \{\mathcal{M}(v_1), \mathcal{M}(v_2), \ldots, \mathcal{M}(v_L)\}$. Once a vehicle reaches an end node, it remains there until it is (possibly) given a new mission, which it then immediately starts executing.

[1] Note that, in the data structure used for the fleet missions, the sequence of missions carried out by each vehicle is, in fact, stored, making it possible to play back the entire history of all vehicles. However, for the purpose of planning, it is only the last (i.e. the current) mission that is relevant, for any vehicle.

A fleet mission specified in this way can be either infeasible, in case there are collisions (or near-collisions; see Sect. 4.2 below) between vehicles, or feasible, in case no such events occur. The primary goal of fleet mission planning is to find feasible missions. An important, but secondary, goal is to make those missions as efficient (i.e. with short duration) as possible.

As for the missions, an incoming (loaded) vehicle will always have the prioritized offloading site as its *primary destination* (i.e. the end node of the mission) whereas an outgoing (unloaded) vehicle will always have a loading site as its primary destination. Evidently, if the number of incoming vehicles (to the single offloading station) is larger than one, or if the number of outgoing vehicles to any given loading station is larger than one, not all vehicles can reach their primary destination. It follows from this observation that the planning procedure must allow for *secondary destinations* different from the desired primary destination. Here, only pause nodes are allowed as secondary destinations.

In the case of *static fleet mission planning*, a single fleet mission is defined, such that the vehicles simply proceed to their destinations (whether primary or secondary), and then remain there indefinitely. By contrast, in *dynamic fleet mission planning*, where vehicles reaching their primary destination are eventually assigned a new mission, a vehicle blocking any terminal will at some point move away, thus making the terminal accessible to other vehicles.

While considerably simpler than the dynamic case, the static case is in fact highly relevant since (as discussed in Sect. 5 below) the planning in the dynamic case can be cast as a sequence of static plans.

For the case of underground mines of the kind considered here, there are two more conditions (rather than mere feasibility and specification of destinations), that must be imposed on the various missions that form a fleet mission. These conditions will be described next.

3.3.1 Prioritized (incoming) missions

The first condition concerns the incoming missions[2] aiming to reach the offloading site. When a vehicle is fully loaded, it should ideally be able to drive all the way to the offloading station (or, at least, to a pause node very close to it, in case the offloading station is occupied, or will be occupied at the time of arrival) at maximum possible speed, without stopping. This is so, since the load on the gear box and engine of a fully loaded mining vehicle is typically so intense that any stop may cause mechanical failures. Thus, for incoming missions, whether they target the offloading station or a nearby secondary destination, the following conditions are applied: Such missions, denoted $\mathcal{M}^{\mathrm{P}}$, (i) have $t_{\mathrm{w}} = 0$ for all mission items, *except* at the start node (i.e. the first mission item) and (ii) always follow the path Π_0 from the start node to the destination (whether primary or secondary).

By contrast, non-prioritized (outgoing) missions may involve one or more pause node visits.

3.3.2 Initial stop times

The second condition limits the use of initial stop times (t_{w}): In practice, $t_{\mathrm{w}} > 0$ is only allowed for (a) the start node of a mission and (b) at pause nodes (only relevant for non-prioritized missions; see above). At all transit nodes, the initial stop time is set to zero, to avoid situations where a vehicle prevents other vehicles from passing.

[2] Here, in *inbound* (incoming) missions, vehicles move towards the offloading station, whereas in *outbound* (outgoing) missions vehicles move towards a loading station deep in the mine.

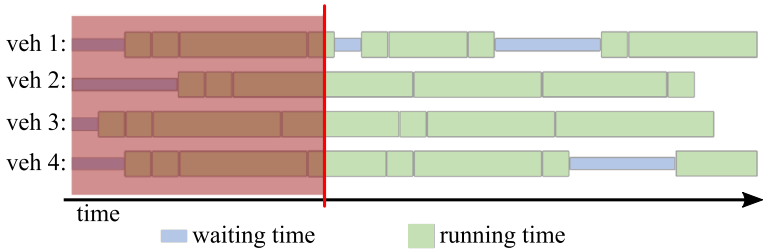

Fig. 3 A fleet mission for a fleet of four vehicles. In this particular case, the top and bottom vehicles are outbound (moving away from the offloading station), whereas vehicles two and three are inbound. The red vertical line indicates the current elapsed time. At this point, all four vehicles are moving, but the first vehicle (the top row) will soon reach a pause node where it will make a brief stop (indicated by the light-blue box), allowing an incoming vehicle to pass (Color figure online)

An example of a fleet mission is shown in Fig. 3. In this case, the fleet consists of four vehicles. The red vertical line indicates the current elapsed time. The stop time before *any* vehicle starts moving is a consequence of the optimization: This figure shows a real-time evaluation, in which the optimizer (see Sect. 4.3) starts at time 0, and then runs for a certain amount of time until the fleet mission has been optimized. In fact, once the optimization is completed, the third vehicle starts moving directly, shortly followed by the other vehicles.

4 Static fleet mission planning

The aim of fleet mission planning is to generate a solution such that all vehicles can reach their current destination, without any (near-)collisions, and doing so in minimal time, measured as the time elapsed from the start of the fleet mission until the last vehicle reaches its (primary or secondary) destination. This time interval is referred to as the fleet mission duration. In the dynamic case, described in detail in Sect. 5, (re-)planning can be requested at any time, either by a vehicle that has just been fully loaded and then requests a trajectory back to the offloading site, or a vehicle that has just been offloaded and requests a trajectory to a loading site. In this case one cannot assume that all the other vehicles are located *at* nodes. Some, or even all, other vehicles may be en route to their respective destination when the re-planning is requested.

However, for now, consider only the static case, where all vehicles start at a terminal node and end either at another terminal (the primary destination) or a pause node (the secondary destination). As mentioned above, in the case of a mine of the kind considered for the dynamic case (see Fig. 1), there is typically a single offloading site meaning, for example, that for inbound missions, only one vehicle can proceed directly to its primary destination, whereas the others must plan a route to a secondary target, i.e. a pause node near the offloading site.

For the purpose of *evaluating* the performance of the static fleet mission optimization procedure, and to make it possible to compare it to other methods, a different type of map will be considered for the static case, namely one in which there is an equal number of loading and offloading sites, and where, for any given terminal (either loading or offloading), at most one vehicle uses that terminal as its target. In other words, for *these* evaluations, all vehicles will be able to proceed directly to their primary destinations, rather than having to make use of secondary destinations.

4.1 Initial fleet mission generation

Before optimization (described below) can begin, one must generate an initial fleet mission that can act as a starting point for the optimizer. This is done as follows: For each vehicle v_k, given node pairs (p_i, p_j) (start and end), where $p_i, p_j \in T$, assign a mission $\mathcal{M}^0(v_k)$ using the path $\Pi_0(p_i, p_j)$, and with $t_w = 0$ for all mission items. The resulting fleet mission $\mathcal{F}^0 = \left\{\mathcal{M}^0(v_1), \mathcal{M}^0(v_2), \ldots, \mathcal{M}^0(v_L)\right\}$ would be ideal, since all vehicles would start directly, and then move as fast as possible along the shortest possible path. However, in all but the simplest situations, this initial fleet mission is likely to be infeasible, i.e. to involve one or more conflicts, due to the properties of the map. Hence, the next step is to tune the constituent missions in the fleet mission such that any collisions are removed while, at the same time, trying to keep the total fleet mission duration as low as possible.

Note that for a vehicle executing a prioritized mission of type $\mathcal{M}^P$ introduced above, the only possible modification (during optimization; see below) is to change t_w for the initial mission item. For non-prioritized missions, however, any number of pause node visits can be inserted as well. The procedure for evaluating and tuning (optimizing) static fleet missions will be described in the following two subsections.

4.2 Fleet mission evaluation

Taking the topological nature of the map into account, the evaluation of a fleet mission can be carried out very efficiently since it is sufficient to check for collisions and other violations only at nodes. The evaluation process proceeds as follows: At any step in the evaluation, the time Δt_k required for vehicle k to reach the next node (i.e. the end point of the segment in the current mission item) is determined for each vehicle. Thus, a list $\{\Delta t_1, \Delta t_2 \ldots\}$ is generated. Next, the smallest element $\Delta t_{\min}$ in the list is found, and time is advanced (in a single, discrete step) by this amount. At this point the vehicle (denoted v_{move}) corresponding to the smallest Δt will thus be *at* a node, namely the end node of its current segment, here denoted $p_{\min}$, whereas all other vehicles will be *between* nodes.[3]

Next, one needs the notion of a *segment pair* which is defined, for any two connected nodes p_i and p_j, as the two segments connecting these two nodes, i.e. the segment from p_i to p_j and the reverse segment. One can then check for collisions by simply investigating the situation, at the time just defined, and for the segment pairs such that one of the nodes is the node $p_{\min}$ introduced above. By construction, these are the *only* segment pairs where a collision might occur, namely if the number of vehicles is non-zero for *both* members of any such segment pair. If that is the case, a collision will occur, an event that here is referred to as a *segment violation*. Note that it *is* allowed, in principle, to have several vehicles on *one* of the segments in a segment pair, as long as there are no vehicles on the *other* (opposing) segment.

In addition, a second check must be carried out, to avoid *grazing collisions*: Even if no segment violations occur, there can still be cases where, for example, a vehicle passes a node just an instant before another vehicle passes the same node. Since the vehicles are not point particles, one must also avoid this type of grazing collision. However, as the topological map does not have a notion of euclidean distance, one must base the analysis on *time* instead.

[3] A rare special case is that in which the smallest Δt occurs for more than one vehicle. In such cases, the collision checking described in the text would be carried out at all nodes reached by those $m > 1$ vehicles. In other words, there would be a set of nodes $p_{\min,1}, p_{\min,2}, \ldots, p_{\min,m}$ for which the checking must be carried out. By far the most common case, however, is $m = 1$.

Thus, the additional check consists simply of checking, at node $p_{\min}$, where vehicle v_{move} is located at the time of checking, whether any vehicle passed that node, or will pass that node, in a given time range. If that is the case, a grazing collision is detected.

This stepping procedure is then repeated until all vehicles have reached (with or without collisions) their end nodes. With this procedure, the entire evaluation of a fleet mission consists of a rather small set of discrete steps, along with the two violation checks (at each such step) described above.

During evaluation, all collisions are detected and logged, but the fleet mission is allowed to run to completion nevertheless. If any collisions occurred (of any of the kinds described above), the score of the evaluation is set as

$$s = s_{\text{fail}} = -N_{\text{c}} + \beta t_{\text{c}} < 0 \tag{1}$$

where N_{c} is the number of collisions and t_{c} is the time to the *first* collision. β is a small constant, whose value should be smaller than the inverse of the longest possible mission duration (a number that can easily be computed from the topological map), ensuring that the score remains negative as long as there are collisions, while still giving a preference for collisions that occur late, meaning that *earlier* collisions have already been successfully avoided. If instead no collision is detected, the score is set as

$$s = s_{\text{success}} = (1/t_{\text{e}}) \times (\xi_{\min}/\xi) > 0, \tag{2}$$

where t_{e} is the time elapsed when all vehicles have completed their missions. $\xi_{\min}$ is the minimum total number of mission items, obtained for the case in which all vehicles follow their respective Π_0 path. As mentioned above, in such a case, collisions will in all likelihood occur. Thus, some vehicles (namely those for which the end node is not a prioritized terminal) will need to make one or more visits to pause nodes along the way, thus increasing the number of mission items by two for every such visit. ξ is the sum (over all missions) of the number of mission items in the fleet mission under consideration. Thus, this measure will attempt to reduce the total time, while also trying to use the shortest possible path, with minimum number of pause node visits, for each vehicle. While the incoming vehicles do not stop (by construction), the outgoing vehicles might make any number of pause node visits. The *number* of such visits should ideally be minimized, since one or a few long-duration stops would generally be preferable to making many short-duration stops, from a fuel consumption perspective.

4.3 Optimization procedure

Here, a modified genetic algorithm (GA) [28,29], shown in Algorithm 1, has been chosen for the optimization. As can be seen in Algorithm 1, the algorithm works as follows: First, the initial fleet mission is evaluated as just described, and an initial population (i.e. a set of candidate solutions, referred to as *individuals*) of size α is generated by making α copies of that fleet mission. Given its definition (see above), it is very likely that this mission will involve some collisions and will therefore receive a negative fitness score.

Next, a new generation is formed by applying tournament selection as described in Algorithm 2, followed by mutation of the selected individuals, described in Algorithm 3. The mutations are either parametric (changing the initial waiting times t_{w}; see also Fig. 4, top panel) or structural (inserting or removing pause node visits; see also Fig. 4). The parametric mutation rate is set dynamically as p_{rel}/κ, where κ is the total number of modifiable parameters for a given fleet mission, i.e. the total number of initial stop times that are allowed values

input : $\mathcal{T} = \{\mathcal{P}, \mathcal{E}\}$ (topological map)
input : $\mathcal{V} = \{v_1, v_2, \ldots, v_L\}$ (set of vehicles)
input : $\mathcal{F}^0 = \left\{\mathcal{M}^0(v_1), \mathcal{M}^0(v_2), \ldots, \mathcal{M}^0(v_L)\right\}$ (initial fleet mission)
input : α (the population size)
output: $\mathcal{F}^{\text{best}}$ (optimized fleet mission)

$s_0 = Evaluate(\mathcal{F}^0)$ (See Subsection 4.2)
for $i \leftarrow 1$ **to** α **do**
  $\mathcal{F}_i \longleftarrow \mathcal{F}^0$
  $s(\mathcal{F}_i) \longleftarrow s_0$
end
$\mathcal{F}^{\text{best}} \longleftarrow \mathcal{F}^0$
$\tau \longleftarrow 0$
while $\tau \leq \tau_{\max}$ **do**
  for $i \leftarrow 1$ **to** α **do**
    $\hat{\mathcal{F}}_i = Select(s(\mathcal{F}_1), s(\mathcal{F}_2), \ldots, s(\mathcal{F}_\alpha))$ (see Algorithm 2)
    $\mathcal{F}_i^{\text{mut}} = Mutate(\hat{\mathcal{F}}_i)$ (see Algorithm 3)
  end
  for $i \leftarrow 1$ **to** α **do**
    $\mathcal{F}_i \longleftarrow \mathcal{F}_i^{\text{mut}}$
  end
  $\mathcal{F}_1 \longleftarrow \mathcal{F}^{\text{best}}$ (elitism)
  for $i \leftarrow 1$ **to** α **do**
    $s(\mathcal{F}_i) = Evaluate(\mathcal{F}_i)$ (See Subsection 4.2)
  end
  $\mathcal{F}^{\text{best}} \longleftarrow$ argmax $s(\mathcal{F}_i)$
  $\tau \longleftarrow$ time elapsed
end

Algorithm 1: The modified GA used for optimization. τ simply denotes the (clock) time elapsed since the start of the optimization, and $\tau_{\max}$ denotes the maximum allowed optimization time. The score s is defined in Subsection 4.2 above.

input : $s(\mathcal{F}_1), s(\mathcal{F}_2), \ldots, s(\mathcal{F}_\alpha)$ (evaluation scores for all individuals)
input : p_{tour} (tournament selection parameter)
output: $\hat{\mathcal{F}}$ (the selected individual)

$i_1 \longleftarrow$random integer $\in [1, \alpha]$
$i_2 \longleftarrow$random integer $\in [1, \alpha]$
$\mathcal{F}^+ =$ argmax$(s(\mathcal{F}_{i_1}), s(\mathcal{F}_{i_2})$
$\mathcal{F}^- =$ argmin$(s(\mathcal{F}_{i_1}), s(\mathcal{F}_{i_2})$
$r \longleftarrow$ random[0,1)
if $r < p_{\text{tour}}$ **then**
  $\hat{\mathcal{F}} \longleftarrow \mathcal{F}^+$
else
  $\hat{\mathcal{F}} \longleftarrow \mathcal{F}^-$
end

Algorithm 2: The tournament selection algorithm.

input : $\mathcal{F}$ (a fleet mission)
input : p_{rel} (the relative parametric mutation rate)
output: $\mathcal{F}^{\text{mut}}$ (a mutated fleet mission)
$p_{\text{struct}} \longleftarrow p_{\text{rel}}/L$ (structural mutation rate)
$p_{\text{mut}} \longleftarrow p_{\text{rel}}/\kappa$
$\mathcal{F}^{\text{mut}} \longleftarrow \mathcal{F}$
for $\mathcal{M}(v) \in \mathcal{F}^{\text{mut}}$ **do**
  $r \longleftarrow$ random(0,1)
  if $r < p_{\text{mut}}$ **then**
    modify t_w for m_1 (the initial stop time of the first mission item)
  end
  if $\mathcal{M}$ *is non-prioritized* **then**
    for $j \leftarrow 2$ **to** q **do**
      if p_s *(in* m_j*)* $\in S$ **then**
        $r \longleftarrow$ random[0,1)
        if $r < p_{\text{mut}}$ **then**
          modify t_w for m_j (the initial stop time of the segment; see the top panel in Figure 4)
        end
      end
    end
    $r \longleftarrow$ random[0,1)
    if $r < p_{\text{struct}}$ **then**
      $r \longleftarrow$ random[0,1)
      if $r < 0.5$ **then**
        find the set T_1 (see the caption below)
        if $T_1 \neq \emptyset$ **then**
          Randomly select an element of T_1
          Insert pause node visit (see the caption below and the middle panel in Figure 4)
        end
      else
        find the set T_2 (see the caption below)
        if $T_2 \neq \emptyset$ **then**
          Remove a randomly selected pause node visit (see the caption below and the bottom panel in Figure 4).
        end
      end
    end
  end
end

Algorithm 3: The mutation algorithm. p_{struct} is the structual mutation rate, and p_{mut} is the parametric mutation rate that, in turn, depends on the number of *modifiable* parameters (κ); see also the main text. S is the set of pause nodes (see Subsection 3.1) and p_s denotes the start node of a mission item. The set T_1 is the set of all transit nodes in a mission $\mathcal{M}$ that are directly connected (via a single segment) to a pause node, after first removing from consideration transit nodes that are connected to a pause node for which a visit is already planned in the vehicle's current mission. In order words, non-sensical multiple visits to the same pause node are not allowed. T_2 is the set of all pause node visits in a mission $\mathcal{M}$, i.e. sequences of two consecutive mission items, for which the end node of the first item (and, therefore, the start node of the second item) is a pause node.

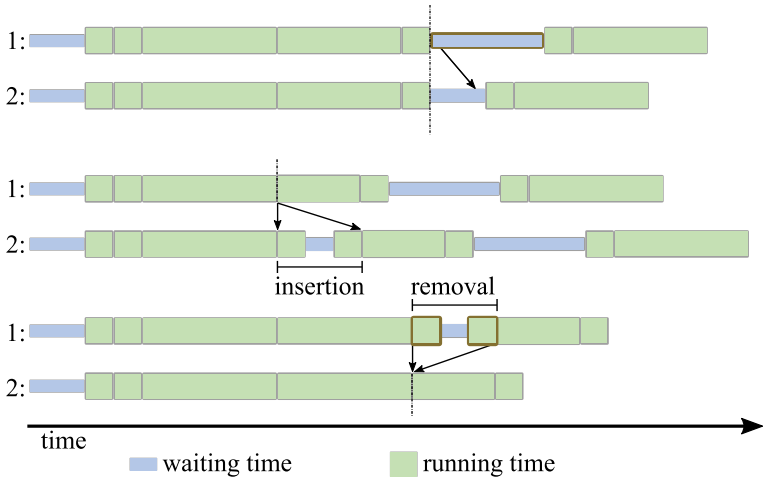

Fig. 4 The various modifications (mutations) that can be applied to a mission. The figure consists of three panels, showing a mission before and after modification, respectively. The top panel shows a parametric modification indicated by an arrow. Here, the modification reduces the initial stop time of one mission item, representing a pause node. The middle and bottom panels illustrate structural modifications. In the middle panel, a pause node visit is inserted: At a given transit node, the vehicle proceeds directly to a pause node, then remains there for a while, as indicated by the initial stop time box, and then returns to the same transit node, from which it proceeds with the mission. The bottom panel shows the *removal* of a pause node visit

different from zero (see also Sect. 3.3 above). The structural mutation rate is set as p_{rel}/L, where L is the total number of vehicles. In cases where a structural mutation is to be carried out, insertion or removal occur with equal probability. Crossover (combining two individuals [28]) is not applied, as it is doubtful whether a meaningful crossover operator could be defined that would provide any benefit beyond the structural mutations just described. Elitism [30] is applied, however, by inserting, as the first individual of the new population, a single exact copy of the best individual from the population just evaluated; see Algorithm 1.

Once the new population has thus been generated, all the new individuals are evaluated and assigned fitness values as per Eqs. (1) and (2), and the procedure of selection followed by (parametric and structural) mutation is repeated again etc. The process continues until the maximum allowed optimization time has been reached.

5 Dynamic fleet mission planning

The static fleet mission planning, described above, concerns a case where all vehicles simply move from a given start node to a given end node. By contrast, in dynamic fleet mission planning, where vehicles repeatedly drive back and forth between the offloading site and the loading sites, re-planning might be required at any time (by vehicles located at terminals), so that one cannot assume that all, or even most, vehicles will be at a node at that time, except at the very beginning.

An additional difficulty is the obvious fact that time does not stop *during* optimization. Thus, when re-planning, one must take into account the fact that the new plan (i.e. the modified fleet mission) will not be activated until the optimization has been completed. The dynamic fleet mission planning has been implemented (in C# .NET) as a multi-threaded system, depicted in Fig. 5, where one thread is responsible for running the missions and

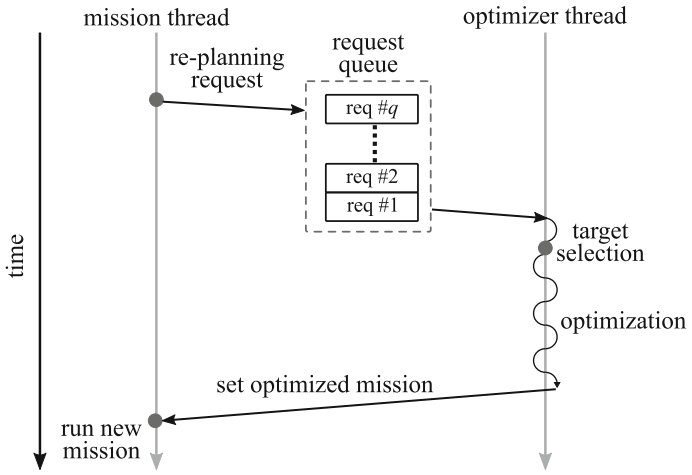

Fig. 5 Representation of threads during dynamic fleet mission planning. The mission thread handles the real-time movement of each vehicle. Every time a vehicle emits a re-planning request, a signal is sent to the fleet optimizer to generate a new (static) fleet mission that will replace the current one (if the planning is successful). A first-in-first-out queue structure is used, so that multiple requests can be queued. In addition to these threads, a third thread is responsible for visualizing the movement on-screen

another thread runs the optimizer. As will be described below, a dynamic fleet mission can be seen as a sequence of static fleet missions, in which the current (static) fleet mission can, at any time, be replaced by a modified static fleet mission obtained via the optimization procedure described below. In dynamic fleet mission, a global time variable is available, which simply measures the total elapsed time since the start, i.e. since the first optimization request was issued.

Whenever a vehicle reaches its primary destination, i.e. either the offloading site or one of the loading sites, offloading or loading will take place. The duration of those processes can (and typically does) vary: In some cases a vehicle can be unloaded or loaded quite fast, in other cases it might take longer time. Thus, from the point of view of fleet mission planning, as long as a vehicle is being processed (unloaded or loaded), one cannot set the corresponding terminal as the current destination of any *other* vehicle. In order to simulate this state of affairs, for any given loading event, the loading time is selected randomly in a range denoted $[T_{\mathrm{L}}^{\min}, T_{\mathrm{L}}^{\max}]$. Similarly, the offloading time for an offloading event is computed randomly in a range $[T_{\mathrm{O}}^{\min}, T_{\mathrm{O}}^{\max}]$.

Once the processing of the vehicle has been completed, the vehicle *requests* re-planning,[4] indicated in Fig. 5 as an event triggered by the mission thread. At this point, a mission is generated for the vehicle requesting the re-planning (hereafter referred to as v_{r}). The new mission requires target selection: If the vehicle is at a loading site, the offloading site is set as the *primary* destination. If the offloading site is not the current destination for any other vehicle, it is also set as the *current* destination for v_{r}. Otherwise, the nearest (relative to the offloading station) not currently targeted pause node is selected as the current (secondary) destination. Similarly, if the vehicle is at the offloading site, the primary destination is selected from the set of loading sites, and the current destination is then set as just explained. In the operation of an actual mine, a common situation is that in which a human operator selects the

[4] Note that, at the very start of the dynamic fleet mission, i.e. at global time 0, an optimization request is generated manually, in order to get the dynamic fleet mission started.

loading site for any outbound vehicle. Here, however, the primary destination of an outbound vehicle is selected randomly, for simplicity. The setup could easily be changed to allow for manual selection of outbound primary destinations.

Then, the shortest path is found (from the path matrix; see Sect. 3.1), connecting the current location of v_r to its current destination. Regardless of whether the mission is inbound (i.e. prioritized) or outbound, the initial stop time is set to zero for all mission items *except* the first one, for which a random initial stop time is set (allowing the vehicle to wait until a suitable time slot can be found). Ideally, the vehicle should be able to drive according to this prescription, but for *outbound* missions, in order to make sure that a feasible fleet mission can be generated, there is still (as in the static case) the possibility for the optimization procedure to insert pause node visits along the way, and also to remove such visits, if any. Then, the mission just described (for v_r) is appended, replacing the previous (now completed) mission. However, in order to retain causality, the initial stop time of the appended mission must be at least as large as the time required for the subsequent optimization. Thus, this lower limit is taken into account when setting the initial stop time of the first mission item.

Finally, before executing the optimization procedure, the entire fleet mission is copied and the current time of the *copied* mission, which will be the starting point for the optimization procedure, is set as the sum of the current global time (i.e. the elapsed time since the start of the dynamic fleet mission) *and* the time required for optimization. Then, during optimization, modifications are *only* allowed in those mission items that start beyond that time, in order to ensure causality. In other words, even though the modification process (for the missions) described above can largely be used as it is, it is only *applied* to those mission items whose start time lies beyond the time at which optimization *will* be completed. For the part of a vehicle's mission fulfilling that condition, modifications are applied precisely as in the static case, namely with the possibility of inserting or removing pause node visits for outbound vehicles. For example, the path of an outbound vehicle moving towards a loading site may undergo a change in which a pause node visit is inserted, in order to accommodate, without collision, the motion of the inbound vehicle that requested the optimization.

Thus, the optimization procedure is then executed, largely in the same way as for the static case. Meanwhile, time progresses, and the vehicles move according to their respective missions, in real time. Once the optimization has been completed, if it is successful, i.e. if a score larger than 0 has been achieved, the new, optimized fleet mission simply overwrites (instantaneously) the current fleet mission, thus *becoming* the current fleet mission. This will have no immediate effect on any of the vehicles, except possibly v_r if its motion is set to begin right after the completion of the optimization procedure. All other vehicles will (by construction, as just described above) proceed, without changes, to the next node, i.e. the end point of the segment in their current mission *item*, before perhaps taking a different path than they would have done, had the optimization not been run.

In the dynamic case, as mentioned above, vehicles *request* re-planning whenever a stationary activity (loading or offloading) has been completed. It follows that a vehicle may request re-planning while an optimization procedure, triggered by an earlier request from *another* vehicle, is in progress. In order to handle these (normally uncommon) events, optimization requests are queued, in the order that they arrive. The contents of the queue are simply the identities of the vehicles requesting optimization. Thus, once an optimization procedure has been completed, if there is one item (or more) in the queue, another optimization procedure is started right away; see also Fig. 5.

Furthermore it can happen that the optimization procedure is unsuccessful, either because the optimizer is not able to find a feasible solution over the time interval in which it is allowed to run *or* simply because there *are* no feasible solutions. The latter can, for example, happen

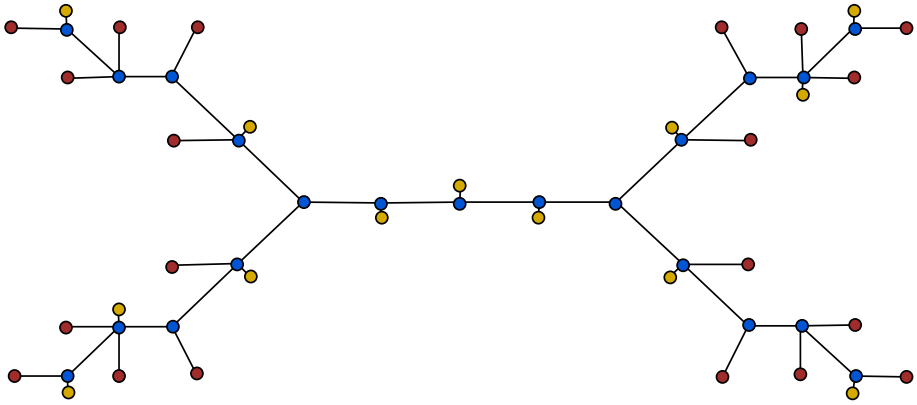

Fig. 6 The map used for the static fleet mission optimization. Two versions were made: (i) a non-prioritized (NPR) map in which no terminals were prioritized and (ii) a semi-prioritized (SPR) map in which the terminals in the left half of the map were prioritized. As in Fig. 1, the node types are indicated using different colors (Color figure online)

in cases where the vehicle requesting optimization (v_r) is at the offloading site when, at the same time, an inbound vehicle is so close to that site that v_r would not have time to reach the nearest pause node before colliding with the inbound vehicle. In either case, if an optimization procedure fails, the vehicle in question will immediately request another re-optimization, thus adding this request to the queue. However, it should be noted that, in the vast majority of cases, the optimization procedure *is* successful; see also the discussion section below.

It should be clarified that only vehicles reaching their *primary* destination can request re-planning (once their stationary activity, either loading or offloading, has been completed). Vehicles that reach a *secondary* destination, for example the pause node closest to the offloading station in the case of a loaded vehicle, cannot themselves request re-planning but they can (and will, assuming continuous operation of the mine) be given a new mission as a result of the re-planning request by another vehicle; to continue the example just given, once the vehicle at the offloading site requests re-planning, the new static fleet mission resulting from this process will involve both the outbound mission of that vehicle, as well as the inbound mission taking the paused vehicle just mentioned to the offloading site (and, possibly, changes in the missions of other vehicles as well).

6 Results

The method for dynamic fleet mission planning presented here involves both path planning and scheduling, where the latter part is of central importance. However, before the results of the full dynamic method are presented in Sect. 6.2, the *static* fleet mission planning method, described in Sect. 4, will be considered in isolation, generating solutions for a case where all vehicles move from a given start node to a given end node, and then stop there. Two different (but equally sized) maps will be considered.

The static planning is also compared, in terms of the number of mission items per plan, against the push-and-swap algorithm [12], see Sect. 6.1.1 and against the operator decomposition M* algorithm, abbreviated ODM* [19], see Sect. 6.1.2.

6.1 Static fleet mission planning

The static fleet mission planning algorithm was implemented in C# .NET and was evaluated using the map shown in Fig. 6. In fact, two instances of this map were generated: One in which *all* terminals are non-prioritized (so that pause node visits are allowed for all vehicles), and one in which half of the terminals, namely those on the left side of the map, are prioritized, and the other half are non-prioritized. These two maps will be referred to as the *non-prioritized map* (hereafter: NPR map) and the *semi-prioritized map* (hereafter: SPR map), respectively. Note that, for the latter, as described in Sect. 4, half of the vehicles will be forced to move along the shortest possible path, without any stops (except before starting the motion). It should also be noted that neither case is particularly representative of a real mine. Instead, these somewhat artificial maps are used merely to investigate the performance of the static fleet mission planning algorithm and to facilitate the comparison (carried out in the NPR map) between, on the one hand, the algorithm introduced here and, on the other hand, the push-and-swap and ODM* algorithms.

In order to carry out a thorough analysis of the static fleet mission planning method, two sets of batch runs were defined, one for each of the two maps just described. In each set, five different configurations where defined, with 2, 4, 6, 8, and 10 vehicles, respectively. For each configuration, a batch of $n = 1000$ runs were carried out, with random selection of the start and end nodes (for each vehicle) as follows: For a given run i, $i = 1, \ldots, n$, in a configuration with a total of L vehicles (where L is even), the first $L/2$ vehicles were assigned randomly selected start nodes (different for each vehicle) from terminals on the left side of the map, whereas the remaining $L/2$ vehicles were given randomly selected start nodes (also different for each vehicle) from terminals on the right side of the map. Next, the end nodes of the first $L/2$ vehicles, were taken (in order) as the start nodes of the last $L/2$ vehicles, and vice versa, thus ensuring that the initial fleet mission, before optimization, would involve conflicts. For both batch runs, vehicles moving from right to left moved with half the speed of the vehicles going the opposite direction, thus simulating a case where vehicles going from right to left are fully loaded, whereas vehicles going the other direction are empty.

In all, the total number of runs was thus equal to $2 \times 5 \times 1000 = 10,000$, where the factor 5 comes from the number of configurations considered. The duration of each individual optimization run was set at $D = L \times \tau$ (s). τ was set to 3 s. Thus, the maximum optimization time was equal to 30 s (for the case of $L = 10$ vehicles), which is an acceptable optimization duration for the case of a real mine, where typical mission durations are much longer than that.

Before starting these 10,000 runs, a brief parameter search was carried out in order to find suitable settings for the GA, varying the population size, the relative mutation rate, and the tournament selection parameter. It turned out that the GA was not very sensitive to the exact parameter settings. The selected parameter settings are shown in Table 1. For each individual run, the status of the optimized fleet mission (i.e. feasible or infeasible) was logged. For

Table 1 The GA parameter settings chosen for the batch runs, for the case of static fleet mission planning

Parameter	Value
Population size (α)	50
Tournament selection parameter (p_{tour})	0.80
Relative mutation rate (p_{rel})	2.0
Optimization time (per vehicle) τ	3 (s)

Table 2 The results obtained for the static fleet mission planning method, applied to the non-prioritized map

k:	2	4	6	8	10
Σ (%)	100.0	100.0	100.0	100.0	98.6
$\overline{\xi}$	24.6	48.8	73.7	98.5	124.7
$\overline{r}$	1.0000	1.0013	1.0054	1.0112	1.0233
$\overline{\Delta}$ (s)	173.8	183.8	202.5	252.6	297.9

Σ is the success rate. $\overline{\xi}$ is the average number of mission items (which should be minimized) for the fleet mission, whereas $\overline{r}$ is the average ratio between the actual number of mission items and the minimum number of mission items. $\overline{\Delta}$, finally, is the average duration of the fleet missions, taken as the time elapsed when the *last* vehicle reaches its end node. The averages are taken over the (successful) runs in the batch

Table 3 The results obtained for the static fleet mission planning method, applied to the semi-prioritized map

k:	2	4	6	8	10
Σ (%)	100.0	100.0	100.0	99.9	99.3
$\overline{\xi}$	24.0	48.7	73.7	97.8	122.5
$\overline{r}$	1.0000	1.0006	1.0039	1.0025	1.0031
$\overline{\Delta}$ (s)	170.5	183.3	194.3	230.1	260.0

The notation is the same as in Table 2

feasible missions, the total number of mission items (ξ; see also Eq. (2)) was logged, as well as the duration of the fleet mission, defined as the time elapsed from the start until the last vehicle reaches its end node.

The results obtained for the NPR map are given in Table 2, whereas the results from the SPR map are given in Table 3. As can be seen in the tables, the success rate (denoted Σ) was 100% (exactly) for most batch runs, except for one run with 8 vehicles in the SPR map, and for a few runs with 10 vehicles, for both maps. In the rare case of a failed run, the number of mission items is not defined, and those runs have therefore been omitted from the computation of the averages.

The absolute (average) number of mission items $\overline{\xi}$ grew (as expected) approximately linearly with the number of vehicles. An even more relevant measure is the (average) ratio $\overline{r}$ between the actual number of mission items, and the minimum number of mission items $\xi_0 = \xi_{\min} + 2 \times (L/2)$, equivalent to a case in which half of the vehicles follow the shortest possible path from start to end, and half of the vehicles make a single pause node visit. Thus, this ratio can be taken as a measure of efficiency, and it is bounded from below by 1. A value of 1 can always be reached in principle, by letting the vehicles traverse the map in pairs with all other vehicles standing still. However, such a solution would obviously not be optimal regarding the fleet mission duration; It is the task of the optimizer to find fleet missions with a minimal value of this ratio, while *also* minimizing the duration of the fleet missions; see also Fig. 7.

The larger values of $\overline{r}$ obtained for the NPR map (relative to the SPR map) are easily explained by the fact that, in the SPR map, half of the vehicles are *required* to follow the shortest possible path, whereas this is not the case for the NPR map. Another important fact is that while the average fleet mission duration ($\overline{\Delta}$) grows when the number of vehicles is increased, it grows rather slowly, increasing by less than 100% when the number of vehicles is increased by 400% (from 2 to 10).

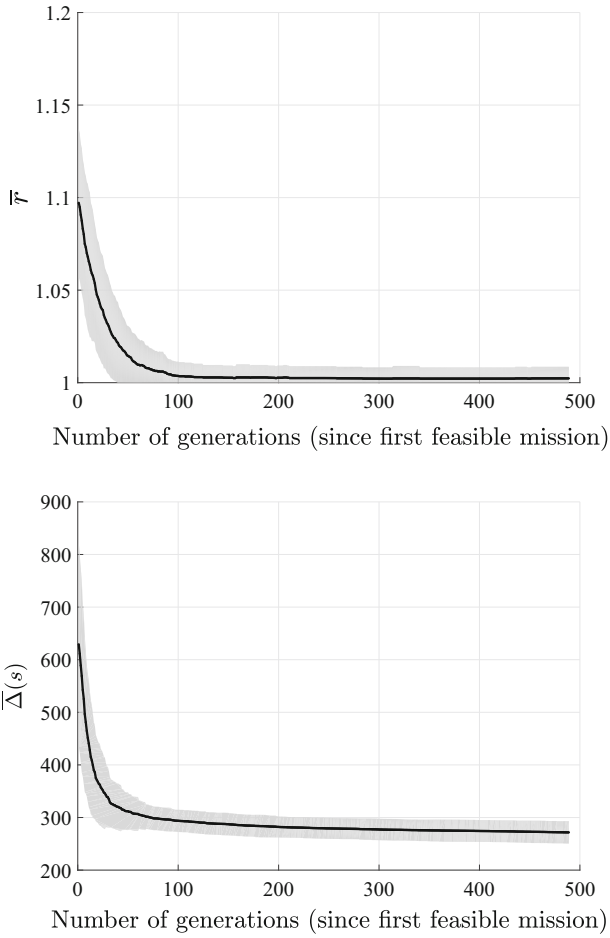

Fig. 7 Detailed analysis of the improvements achieved by the GA during static fleet mission optimization (averaged over 100 runs, in this case), for a representative configuration with 10 vehicles. The top panel shows the drop in the average relative number of mission items ($\overline{r}$), whereas the bottom panel shows the drop in the average fleet mission duration ($\overline{\Delta}$). In both panels, the horizontal axis measures the number of generations from the point (which varies between runs) at which the first feasible fleet mission was found. The gray regions represent one standard deviation around the mean

Table 4 presents an investigation into the effects of changing the optimization duration (D), for the most challenging case, involving 10 vehicles. As expected, as D is increased, both $\overline{r}$ and $\overline{\Delta}$ decrease. However, the sum of the optimization duration and the (average) mission duration reaches a minimum for $D \approx 30\,\text{s}$, at which point also the success rate is near 100%. While slightly smaller average mission durations can be obtained for larger values of D, the decrease is then offset by the increase in $\overline{\Delta}$, making $D = 30\,\text{s}$ a suitable value for practical use for the case of 10 vehicles, and motivating the empirical choice of using $\tau = 3\,\text{s}$ throughout.

In order to demonstrate the extent to which the GA is able to improve a static fleet mission, a detailed analysis was carried out in which the number of mission items and the fleet mission duration (for the current best individual in the GA) was measured as a function of the number of evaluated generations, starting at the generation at which the first feasible static fleet

Table 4 The success rate (Σ), along with $\bar{r}$, and $\overline{\Delta}$, as functions of the optimization run time (D) for the case of 10 vehicles

Run time (D)	10 s	20 s	30 s	40 s
Σ (NPR) (%)	75.3	84.9	98.6	98.1
$\bar{r}$ (NPR)	1.0363	1.0350	1.0233	1.0222
$\overline{\Delta}$ (NPR) (s)	353.5	342.3	297.9	295.6
Σ (SPR) (%)	98.7	99.3	99.3	99.6
$\bar{r}$ (SPR)	1.0039	1.0032	1.0031	1.0028
$\overline{\Delta}$ (SPR) (s)	287.4	274.4	260.0	258.1

Results are given for both maps

mission was found. This analysis was carried out for the most challenging case, with 10 vehicles, by selecting a few representative start and end node configurations among the 1000 used in the runs described above, and then running the GA 100 times for each configuration. The results, which were very similar between different configurations, are exemplified for one such configuration in Fig. 7, where the top panel shows the reduction in the number of mission items (relative to the theoretical minimum ξ_0), and the bottom panel shows the reduction in the fleet mission duration, in both cases averaged over the 100 runs.

Since the GA is a stochastic optimization method, the exact generation at which the first feasible mission is found will vary between runs. Thus, for the purpose of this figure, the origin of the horizontal axis has been taken (for each run) as the generation at which the first feasible mission was found, something that typically occurs after around 50 generations. As can be seen in the top panel of the figure, from that point onwards, the GA is able to reduce the number of mission items by around 10 %, reaching a value close to the theoretical minimum. Meanwhile, as shown in the bottom panel, the GA is also able to reduce the fleet mission duration by more than 50%.

Note that, with the population size of 50 individuals and the typical optimization duration $D = 30$ s (for the 10-vehicle case), the optimizer is able to complete several thousand generations.

6.1.1 Comparison with push-and-swap

The push-and-swap (henceforth: PAS) algorithm [12] offers a non-optimal solution with completeness guarantee for time-invariant multi-vehicle planning problems (but not for the scheduling problem). It is able to find a solution by the iteration of two simple operations; *pushing* vehicles toward their target, and *swapping* every time a conflict is found. Various versions of this algorithm improve the performance in terms of efficiency by smoothing the solution [31] or by parallelization [32].

The push-and-swap algorithm starts by setting the initial configuration of all the agents in the graph, and their respective targets. Then it iterates over all the agents, and moves them toward their targets along the shortest path in the graph, one by one according to a first-in-first-out list. Every time a conflict is found, the swap function solves the conflict by reversing the push function between the two agents until a possible deviation in the path is found and finally the actual swap between the agents takes place. This simple method makes it possible to solve the problem, but it has the drawback of adding many redundant node visits in the paths.

A comparison between the proposed GA-based algorithm and the PAS algorithm can be found in Table 5. As can be seen in the table, the number of mission items needed for PAS is consistently around twice the number of mission items required for the GA-based approach. The table also includes a comparison with the ODM* algorithm, described next.

Table 5 Summary of the comparison between the proposed GA-based algorithm, the PAS algorithm, the standard ODM* algorithm (without inflation of the heuristic), and the ODM* algorithm with the heuristic, here denoted i-ODM*, using 2, 4, 6, 8, or 10 vehicles

	2	4	6	8	10
GA					
Σ (%)	100.0	100.0	100.0	100.0	98.6
$\overline{\xi}$	23.99	48.73	73.69	97.93	123.34
σ	2.94	3.91	4.54	5.36	11.43
PAS					
Σ (%)	100.0	100.0	100.0	100.0	100.0
$\overline{\xi}$	49.39	99.05	148.17	196.81	245.48
σ	5.16	6.79	8.11	8.33	9.02
ODM*					
Σ (%)	100.0	100.0	–	–	–
$\overline{\xi}$	23.99	48.69	–	–	–
σ	2.94	3.88	–	–	–
i-ODM*					
Σ (%)	100.0	100.0	100.0	100.0	99.1
$\overline{\xi}$	23.99	48.69	73.72	105.10	142.46
σ	2.94	3.88	4.60	6.04	15.87

For each algorithm, the first row shows the percentage of successful runs (over the 1000 runs described in Sect. 6.1), the second row shows the average number of mission items, and the final row shows the standard deviation. Note that all comparisons were carried out for the NPR map

6.1.2 Comparison with ODM*

The proposed GA-based algorithm was also compared against the operator decomposition M* algorithm (henceforth: ODM*) [19] with heuristic inflation. This method is of general purpose and offers optimality of the solution. It operates by first solving individual vehicle planning, and then analyses the solution looking for conflicts. However, unlike the original version of M*, it substitutes A* with operator decomposition [33] to increase efficiency. The comparison has been carried out using the original code kindly offered by the authors of [19].

The ODM* algorithm worked very well with grid maps, where many different paths having the same cost can be found. However, our comparison using the map in Fig. 6 revealed that the high number of conflicts in the plan (when the vehicles cross the bottleneck in the middle) results in an exponential increase of the dimensionality of the expansion. Indeed, the ODM* algorithm (in its original version) relies on a subdimensional expansion, locally increasing the dimensionality of the search space, and a backpropagation each time a new conflict is found, updating the collision set. This is a recursive operation, which generates a computational cost that is exponential in the size of the collision set, an effect that is also described in [19]. To limit this effect, a scalar parameter was introduced in [19] for inflation of the heuristic, trading optimality for computational speed, thus generating faster but sub-optimal solutions.

First, the original version of ODM*, without the inflation of the heuristic (i.e. with the corresponding parameter set to one) was tested, using the map in Fig. 6. This version was able to find a solution *only* for the cases of 2 and 4 vehicles; already with 6 vehicles the number of subdimensional expansions of the search space grows so much that the solution cannot be found in the allowed time (3 s per vehicle). In fact, even with the run time extended to *hours*, no solution could be found.

The inflation of the heuristic was then tuned to make the ODM* algorithm (now abbreviated i-ODM*) able to solve the problem also with the larger number of vehicles. In this case, a sub-optimal solution could be found. Here, it is important to observe that different levels of inflation of the heuristic generate different solutions to the same problem, and the required solving time changes accordingly. As a general rule, low inflation values generate better solutions (closer to the optimum), but the algorithm takes a longer time to converge whereas high values of the inflation generate fast, but strongly sub-optimal solutions. In order to make a fair comparison, the inflation of the heuristic was chosen as the minimum value that maximizes the success rate in the given time frame, given that the timeout for the optimizer was chosen according to the criterion previously explained (3 s per vehicle). The results of the comparisons are summarized in Table 5.

6.1.3 Summary of findings for the static planning

The comparison of the algorithms, reported in Table 5, shows that the solution provided by the PAS algorithm is fast but also strongly sub-optimal and this effect increases with the number of vehicles: For PAS, the number of mission items (which should be minimized) is consistently around double that of the GA-based approach.

As for the ODM* with heuristic inflation, it can be observed that, with 2, 4, or 6 vehicles, the performance characteristics of the GA-based algorithm and i-ODM* are quite similar. However, a difference in favor of the GA-based approach is noticeable for the case of 8 vehicles, and very strongly so for the case of 10 vehicles. Here, the heuristic plays an important role in i-ODM*, in order for that algorithm to find the solution in the allowed time (3 s per vehicle), taking the solution found using the i-ODM* algorithm farther from the optimum. Indeed the GA-based algorithm browses very efficiently the solution space resulting in a competitive overall result, with fewer mission items than the solutions found by i-ODM*.

Another very important point is that the characteristics of the PAS and ODM* algorithms make them unsuitable for use in a mine of the kind considered for the dynamic planning case (Fig. 1). The reversals of direction that frequently take place in PAS and also *can* happen in i-ODM*, are not realistic in the case of an underground mine, where incoming vehicles typically must be given priority, and therefore should not stop, let alone reverse their direction of motion, until the target is reached.

6.2 Dynamic fleet mission optimization

For the dynamic case, the simulation runs are, by necessity, carried out in real time: As outlined in Sect. 5 above, in dynamic fleet mission planning, one must take into account the fact that time flows, and therefore that vehicles move, while optimization is taking place. This implies that the optimizer, triggered by an optimization request by a vehicle located at a terminal, must run for a pre-specified duration and must, at the start of the optimization process, take into account the positions at which the (moving) vehicles *will be* located at the instant when the optimization is completed.

In order to evaluate thoroughly the performance of the proposed method for dynamic planning, several runs were carried out using the C# .NET implementation, and with the optimization settings again taken from Table 1. The runs had a fixed total duration of 14,400 s (4 h), and were carried out in the map shown in Fig. 1, for which the speed was 30 km/h for outbound segments (away from the offloading station). For the inbound segments, a much lower speed (10 km/h) was used in the part of the map leading up to the offloading station, i.e. the

steep spiralling corridor referred to in Fig. 1, whereas the speed on all other inbound segments was set to 20 km/h, in both cases accounting for the lower maximum speed attainable by a fully loaded vehicle. As in the static case, only outbound vehicles were allowed to make pause node visits. The typical traversal times for outbound missions were on the order of 500 s. The exact traversal time for an outbound vehicle also depended on the duration of its pause node visits (if any). For inbound missions, the typical traversal times were on the order of 800 s.

The number of vehicles ranged from two to five: While the map has more terminals (seven in total), one should note that there is only a single offloading station, representing a significant bottleneck as soon as there are more than two vehicles in the map. Since the time required for loading and offloading is random (in this case ranging from 2 to 5 min for loading and from 30 s to one minute for offloading), every 4-h run will be different from any other such run and will also involve a non-repeating sequence of different situations to be dealt with by the optimizer, thus providing a thorough test of the proposed method.

Unlike the static case, the dynamic case does not have an obvious benchmark with which it can be compared, to the authors' knowledge. However, one can define several efficiency measures for evaluating the performance. Here, three such measures have been defined. The first measure is the number of traversals completed, defined as the number of traversals C either from the offloading station to a loading station, or in the opposite direction,[5] averaged over all vehicles, and then normalized by the number of vehicles.

Two other relevant measures are the *idle time fraction at terminals* i_t and the *idle time fraction at pause nodes* i_p. i_t is defined as the fraction of the total elapsed time (averaged over all vehicles) spent by vehicles at terminals *after* completing their necessary task (offloading or loading) at the terminal. In other words, any time spent either for mission optimization or waiting for the mission to start (i.e. in the initial stop time of the first mission item) will contribute to i_t. The unattainable ideal value of this measure would be zero. In reality, it will be larger than zero, since the optimization is not instantaneous and also since vehicles often have to wait a while before starting their motion in order to avoid collisions with other vehicles (especially for inbound vehicles, for which pause node visits are not allowed). i_p is simply defined as the fraction of the total elapsed time, again averaged over all vehicles, spent at pause nodes. Here, too, the ideal value would be zero, but the actual value will always be positive, since pause node visits are required in order to avoid collisions, at least in maps of the kind considered here.

The results for runs with two, three, four, and five vehicles are summarized in Table 6, where all entries are averages over five runs, each lasting 4 h. In all cases, the runs were all successful. An example can be seen in the accompanying video.[6]

As can be seen in the table, the most important measure of efficiency, namely the number of completed traversals *per vehicle*, went down (as expected) when the number of vehicles was increased. However, the decrease was rather small: A 150% increase in the number of vehicles (from 2 to 5) resulted in a decrease of only around 27% in the number of traversals per vehicle over the 4-h interval. The larger increases in the idle times, at pause nodes and at terminals, were also expected. A major contributing factor to the increase in the idle time at terminals was the fact that a vehicle at the (single) offloading site would often be kept waiting

[5] Here, the term *number of traversals* has been used instead of the more obvious *number of completed missions*. This is so, since a traversal between terminals may, as described at the end of Sect. 5, involve more than one mission: An inbound vehicle's first mission may take it to a secondary destination, and then be followed by a second mission emanating from that point and ending at the offloading terminal. By counting the number of traversals completed, one thus avoids artificially inflating the performance.

[6] The video can be obtained from the corresponding author upon request, and can also be found at http://www.me.chalmers.se/~mwahde/research/mining/videos.html.

Table 6 Evaluation of the dynamic planning method

# of vehicles	C	i_t	i_p
2	23.7 ± 0.4	0.018 ± 0.007	0.041 ± 0.012
3	20.9 ± 0.4	0.033 ± 0.004	0.109 ± 0.016
4	19.4 ± 0.3	0.040 ± 0.005	0.155 ± 0.011
5	17.4 ± 0.8	0.074 ± 0.010	0.213 ± 0.029

All entries are obtained as averages over five runs, each lasting 4 h. As explained in the main text, C is the number of traversals (per vehicle), i_t is the idle time fraction at terminals, and i_p is the idle time fraction at pause nodes

for quite a while, due to the barrage of incoming vehicles that sometimes made it theoretically impossible to find a solution, keeping in mind that incoming vehicles are not allowed to visit pause nodes. It should be noted that using four or five vehicles in the rather small mine defined in Fig. 1 really pushes the boundary of what is possible in a case with a single offloading site.

7 Discussion

The investigation of the static case showed that the proposed method is able to find feasible solutions, within the allotted time, in the vast majority of cases. As mentioned in connection with Table 5, with the proposed method, the number of mission items (traversed segments) is consistently around one half of the number of mission items obtained for the push-and-swap algorithm, and is also lower than the number of mission items resulting from the ODM* algorithm for cases with 6 vehicles or more.

For the most challenging case considered here, the optimizer typically found its first feasible solution already after around 50 generations or so, much smaller than the total number of evaluated generations (thousands). In a realistic case, where typical traversal times over the shortest route between terminals is on the order of 10–15 min or more, the extra time spent on optimization is, however, worth the effort. This fact is further illustrated in Fig. 7, where the improvements obtained during optimization are evident, regarding both the number of mission items and the fleet mission duration.

The exact duration of the optimization runs is a tunable parameter. The value selected here, namely $\tau = 3$ s per vehicle, is motivated by the results shown in Table 4: Summing the optimization time and the average fleet mission duration, one finds that the minimum occurs for $D \approx 30$ s for the most difficult (10-vehicle) case, for both maps considered in the static case.

Even though the proposed optimization method is *almost* always successful, as illustrated in Tables 2 and 3, one might be concerned that the stochastic nature of the optimizer may lead to failures. However, for the dynamic case, which was tested in a realistic mine map, an occasional, rare failure of the optimizer merely reduces efficiency somewhat, but does not cause any catastrophic failures: If the optimizer fails to find a solution to a given vehicle's optimization request, for instance because there *is* no feasible solution in the current situation, the request is simply repeated and placed in the queue, while the other vehicles continue moving according to the current (feasible) static fleet mission. As those vehicles gradually reach their destinations (whether primary or secondary) the optimizer eventually succeeds in one of the following attempts.

The scheduling for the dynamic case is deadlock-free: If any vehicle is moving, there is no deadlock, by construction; once a vehicle is en route, it means that it has a feasible plan to reach its destination without collisions. If instead all vehicles are all idling, either at pause

nodes or at terminals, the deadlock-free condition is ensured by the fact that a *possible* plan is always to move one vehicle at a time from its start node to its destination (such a fleet mission would be highly inefficient, and would probably never occur in practice, but the point is that it is a possible solution, guaranteeing that there will be no deadlocks), provided that the number of vehicles does not exceed the number of terminals plus the number of pause nodes, minus one.

The traversal times were set to a given value for each map segment, a value that, in a real mine, would be obtained by driving repeatedly over the segment and then taking the maximum time as the assigned segment traversal time for the segment in question. An alternative approach would have been to allow variable (i.e. longer) traversal times, which could then also be subject to optimization, for the non-prioritized missions. In fact, this approach *was* tried briefly, but was later abandoned, partly because a slower segment traversal can generally be replaced by the combination of using instead the nominal segment traversal time *and* making one or a few pause node visits, and partly because frequent speed changes could cause more wear and tear on the mechanical equipment, and may also be unpleasant for any human passengers. Finally, by removing the possibility of modulating the traversal time, one also reduces the number of optimizable parameters, thus increasing the efficiency of the optimization process.

In this paper, task allocation (i.e. selection of primary destinations) has been done randomly (see Sect. 5), which is sufficient for demonstrating the performance of the proposed method. In general, however, the problem of task allocation is a complex one [34] and, even though some solutions already exist [35,36], it is a topic that requires further research. An interesting solution is proposed in [37], in which a decentralized auction-based system is used.

The fitness measure used in this investigation is somewhat arbitrary, but was found to be efficient in minimizing both the fleet mission duration and the number of mission items; see also Fig. 7. Moreover, here, the optimization procedure ran as a single thread. It would be possible to parallelize the optimizer in order to achieve further performance improvements. These are topics for future work.

8 Conclusion

The main conclusion from this work is that the proposed method is capable of autonomously operating a fleet of robotic mining vehicles in real time, without collisions, for many hours without any human intervention. An extensive statistical evaluation showed that the modified GA is able to minimize the number of pause node visits, thereby also strongly reducing the durations of the missions. As a result, for the dynamic planning, which was tested in a realistic mine map with a single offloading site, the method showed favorable scaling properties. The number of completed terminal-to-terminal traversals *per vehicle* was reduced by only 27% as the number of vehicles was increased by 150%, demonstrating the method's capability to handle the significant bottleneck represented by the single offloading site.

The investigation presented here could be extended to larger maps, and to cases with more vehicles. As long as the ratio between the number of vehicles and the number of terminals remains in a similar range as in the cases considered here, it is expected that the method would also yield similar performance. Note, however, that with a *single* offloading terminal (as in the dynamic case above), the region around that terminal represents a significant bottleneck, limiting the number of vehicles that can reasonably be used.

The proposed method has been developed for loop-free maps, also assuming that no other (manually driven) vehicles are present in the mine. Extending the method to remove

these limitations is a relevant topic for future work. Other potential extensions would be to apply a more sophisticated approach to task allocation, and also to explore different fitness measures for optimization by including other objectives such as, for example, fuel consumption minimization.

Acknowledgements The financial support of Vinnova (Grant No. 2015-00611) is gratefully acknowledged. We would also like to thank three anonymous reviewers, whose constructive comments helped us improve the paper considerably.

Appendix A: Notation

This appendix covers general notation used throughout the paper. Note, however, that some symbols that are only used in a specific, limited context are not given below. For example, the notation related to the genetic algorithm is given (and used exclusively) in Sect. 4.3.

Symbol	Description
$\mathcal{T}$	Topological map
$\mathcal{P}$	The set of nodes (vertices) in a topological map
$\mathcal{E}$	The set of segments (edges) in a topological map
p_i	A general node, i.e. an element of $\mathcal{P}$
p_s	Start node of a segment
p_e	End node of a segment
t_j	Traversal time for a segment
T	The set of terminal nodes
S	The set of pause (swap) nodes
Q	The set of transit nodes
T^{P}	The set of prioritized terminals
T^{NP}	The set of non-prioritized terminals
$\Pi_0(p_i, p_j)$	The shortest path between nodes p_i and p_j
$\mathcal{V}$	The set of vehicles
v_k	A vehicle, i.e. an element of $\mathcal{V}$
L	The number of vehicles
$\mathcal{M}(v_k)$	A mission (for vehicle v_k)
m_i	A mission item
t_{w}	The initial stop time for a mission item
$\mathcal{F}$	A fleet mission
$\mathcal{M}^{\mathrm{P}}$	A prioritized mission
$\mathcal{F}^0$	An initial fleet mission (before optimization)
s	Evaluation score (fitness measure)
ξ	The number of mission items in a *fleet* mission
Δ	Fleet mission duration
r	Ratio between actual and minimum number of mission items
Σ	Success rate (for static optimization runs)
D	Optimization duration
C	Number of traversals (for the dynamic case)
i_t	Idle time fraction (at terminals)
i_p	Idle time fraction (at pause nodes)

References

1. Marshall, J. A., Bonchis, A., Nebot, E., & Scheding, S. (2016). Robotics in mining. In B. Siciliano & O. Khatib (Eds.), *Springer handbook of robotics* (pp. 1549–1576). Heidelberg: Springer.
2. Haviland, D., & Marshall, J. (2015). Fundamental behaviours of production traffic in underground mine haulage ramps. *International Journal of Mining Science and Technology*, *25*, 7–14.
3. Grehl, S., Sastuba, M., Donner, M., Ferber, M., Schreiter, F., Mischo, H., & Jung, B. (2015). Towards virtualization of underground mines using mobile robots—From 3D scans to virtual mines. In *Proceedings 23rd international symposium on mine planning and equipment selection* (pp. 711–722).
4. Langdon, M. (2009). Underground vision. *Engineering Technology*, *4*(20), 44–47.
5. Larsson, J., Appelgren, J., & Marshall, J. (2010). Next generation system for unmanned LHD operation in underground mines. In *Proceedings of the annual meeting and exhibition of the society for mining, metallurgy and exploration (SME)*.
6. Pasternak, M., & Marshall, J. A. (2016). On the design and selection of vehicle coordination policies for underground mine production ramps. *International Journal of Mining Science and Technology*, *26*(4), 623–627.
7. LaValle, S. M. (2006). *Planning algorithms*. Cambridge: Cambridge University Press.
8. Berglund, T., Brodnik, A., Jonsson, H., Staffanson, M., & Soderkvist, I. (2010). Planning smooth and obstacle-avoiding b-spline paths for autonomous mining vehicles. *IEEE Transactions on Automation Science and Engineering.*, *7*(1), 167–172.
9. Abraham, A. T., Ge, S. S., & Tao, P. Y. (2009). A topological approach of path planning for autonomous robot navigation in dynamic environments. In *International conference on intelligent robots and systems (IROS)* (pp. 4907–4912).
10. Murano, A., Perelli, G., & Rubin, S. (2015). Multi-agent path planning in known dynamic environments. In *International conference on principles and practice of multi-agent systems* (pp. 218–231). Springer.
11. Yu, J., & LaValle, S. M. (2013). Multi-agent path planning and network flow. In E. Frazzoli, T. Lozano-Perez, N. Roy, & D. Rus (Eds.), *Algorithmic foundations of robotics X* (Vol. 86, pp. 157–173)., Springer tracts in advanced robotics Berlin: Springer.
12. Luna, R., & Bekris, K. E. (2011). Push and swap: Fast cooperative path-finding with completeness guarantees. In *Proceedings of the 22nd international joint conference on artificial intelligence* (pp. 294–300).
13. Yan, Z., Jouandeau, N., & Cherif, A. A. (2013). A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, *10*(12), 399.
14. Wang, Q., & Phillips, C. (2014). Cooperative path-planning for multi-vehicle systems. *Electronics*, *3*(4), 636–660.
15. Chen, Y. F., Liu, M., Everett, M., & How, J. P. (2017). Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)* (pp. 285–292).
16. Sartoretti, G. A., Kerr, J., Shi, Y., Wagner, G., Kumar, T. S., Koenig, S., et al. (2019). PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, *4*(3), 2378–2385.
17. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.
18. Wagner, G., & Choset, H. (2011). M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, (pp. 3260–3267).
19. Wagner, G., & Choset, H. (2015). Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, *219*, 1–24.
20. Ferner, C., Wagner, G. & Choset, C. (2013). ODrM* optimal multirobot path planning in low dimensional search spaces. In *IEEE international conference on robotics and automation, Karlsruhe, Germany*, (pp. 3854–3859).
21. Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent path finding. *Artificial Intelligence*, *219*, 40–66.
22. Sharon, G., Stern, R., Felner, A. & Sturtevant, N.R. (2012). Meta-agent conflict-based search for optimal multi-agent path finding. In *International symposium on combinatorial search*, (pp. 39–40).
23. Ma, H., & Koenig, S. (2016). Optimal target assignment and path finding for teams of agents. In *Proceedings of the 2016 international conference on autonomous agents and multiagent systems* (pp. 1144–1152).
24. Carpin, S., & Pagello, E. (2002). On parallel RRTs for multi-robot systems. In *Proceedings of 8th conference on Italian Association for Artificial Intelligence* (pp. 834–841).

25. Zhong, W., Liu, J., & Jiao, L. (2004). A novel genetic algorithm based on multi-agent systems. In M. A. Kłopotek, S. T. Wierzchoń, & K. Trojanowski (Eds.), *Intelligent information processing and web mining, advances in soft computing* (Vol. 25, pp. 169–178). Berlin: Springer.
26. Van Lon, R. R. S., Branke, J., & Holvoet, T. (2018). Optimizing agents with genetic programming: An evaluation of hyper-heuristics in dynamic real-time logistics. *Genetic Programming and Evolvable Machines*, *19*(1–2), 93–120.
27. Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2008). A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, *58*(1), 21–51.
28. Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge: MIT Press.
29. Wahde, M. (2008). *Biologically inspired optimization methods* (pp. 40–78). Ashurst: WIT Press.
30. DeJong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. *Ph.D. Dissertation*, University of Michigan, Ann Arbor.
31. Luna, R., & Bekris, K. E. (2011). Efficient and complete centralized multi-robot path planning. In *International conference on intelligent robots and systems (IROS)* (pp. 3268–3275). IEEE/RSJ.
32. Sajid, Q., Luna, R., & Bekris, K. E. (2012). Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *International symposium on combinatorial search* (pp. 88–96).
33. Standley, T. S. (2010). Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the 24th AAAI conference on artificial intelligence (AAAI), Atlanta, Georgia, USA* (Vol. 1, pp. 28–29).
34. Mosteo, A. R., & Montano, L. (2010). A survey of multi-robot task allocation. *Technical report*. Instituto de Investigación en Ingeniería de Aragón, University of Zaragoza.
35. Matarić, M. J., Sukhatme, G. S., & Østergaard, E. H. (2003). Multi-robot task allocation in uncertain environments. *Autonomous Robots*, *14*(2–3), 255–263.
36. Kartal, B., Nunes, E., Godoy, J., & Gini, M. (2016). Monte carlo tree search with branch and bound for multi-robot task allocation. In *The IJCAI-16 workshop on autonomous mobile service robots*.
37. Nunes, E., McIntire, M., & Gini, M. (2017). Decentralized multi-robot allocation of tasks with temporal and precedence constraints. *Advanced Robotics*, *31*(22), 1193–1207.