IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan. Cite as T-ASE paper.

Optimal Path Planning for a Convoy-Support Vehicle Pair through a Repairable Network

Abhay Singh Bhadoriya[1], Christopher Montez[1], Sivakumar Rathinam[1], Swaroop Darbha[1], David W. Casbeer[2], and Satyanarayana G. Manyam[3]

***Abstract*—In this article, we consider a multi-agent path planning problem in a partially impeded environment. The impeded environment is represented by a graph with select road segments (edges) in disrepair impeding vehicular movement in the road network. A primary vehicle, which we refer to as a convoy, wishes to travel from a starting location to a destination while minimizing some accumulated cost. The convoy may traverse an impeded edge for an additional cost (associated with repairing the edge) than if it were unimpeded. A support vehicle, which we refer to as a service vehicle, is simultaneously deployed to assist the convoy by repairing edges, reducing the cost for the convoy to traverse those edges. The convoy is permitted to wait at any vertex to allow the service vehicle to complete repairing an edge. The service vehicle is permitted to terminate its path at any vertex. The goal is then to find a pair of paths so the convoy reaches its destination while minimizing the total time (cost) the two vehicles are active, including any time the convoy waits. We refer to this problem as the Assisted Shortest Path Problem (ASPP). We present a generalized permanent labeling algorithm (GPLA) to find an optimal solution for the ASPP. We also introduce additional modifications to the labeling algorithm to significantly improve the computation time and refer to the modified labeling algorithm as GPLA*. Computational results are presented to illustrate the effectiveness of GPLA* in solving the ASPP.**

***Note to Practitioners*—One motivation for this work is to improve the efficiency of autonomous warehouse operations, where multiple robots need to coordinate their plans. Take for example two robots operating in a warehouse where one robot is moving goods and the second robot is making repairs or clearing obstructions (fallen goods, objects left by workers, etc.) along the way. The presented algorithm's underlying structure is relatively simple and the algorithm itself does not require special software or solvers. The algorithm generates sub-optimal solutions as it progresses and terminates with the optimal solution. A large class of problems involving asynchronous actions between two or more agents can be handled using the presented algorithm or an extension of it. In this paper we restrict ourselves to two agents. A limitation of the presented algorithm and its possible extensions is the memory required as the graph representing the problem grows in size. We compare our work against an algorithm with similar approach (centralized A^*) and show that the presented algorithm is superior in both memory and computational time. We also present results on relatively large graphs to show the algorithm has practical value. This work can also be applied to rescue missions for people to escape wildfires, flooding or other natural disasters. A robotic agent can scout ahead for impacted pathways and assist victim(s) find the best path to escape to safety.**

***Index Terms*—Multi-agent path planning, intelligent robots, graph theory, operations research, cooperation, collaboration.**

I. Introduction

THERE has been an increasing interest in the study of cooperative behavior between multiple autonomous agents operating in a shared environment to complete some given task or set of tasks. This growing interest is due to the many applications which can make use of the cooperation of multiple agents, such as search-and-rescue [1], cooperative manipulation [2], foraging [3], surveillance [4], and safe escort using coordinated UAV and UGV systems [5]–[7] to name a few. The use of multiple agents to complete a given task or set of tasks can potentially lead to performance improvements (faster times, more tasks done per unit cost, etc.). The use of multiple agents also allows for a wider variety of situations to be addressed. Cooperative multi-agent planning can be categorized into two major classes. In the first class, agents try to acheive their individual goals while sharing some common environment or resources (see [8]–[10] for some examples). In the second class, agents work together to synthesize a joint plan that achieves some common goal. The second class of problems is also commonly described using the MA-STRIPS model [11] in conjunction with MA-PDDL [12]. For a comprehensive survey of MA-STRIPS algorithms, we refer the reader to [13]. The manner in which the agents cooperate will vary depending on the common goal considered, such as needing to directly work together to physically move an object in space [14] or performing some task(s) in some synchronized manner as in the case of this paper.

The focus of this paper is the path planning of two autonomous agents operating in a partially impeded environment. As an example, this impeded environment may be a warehouse in which agents are organizing goods. The pathways in the warehouse may have some obstructions, such as fallen goods, that hinder the agents' ability to maneuver. This impeded environment is represented by a graph where select edges represent obstructions using edge weights that will be defined later. These edges are assumed to be known *a priori* and are referred to as impeded edges. In the general case, obstructions may be physical or abstract. A designated agent, referred

[1]With Texas A & M University, College Station, TX, USA abhay.singh@tamu.edu, yduaskme@tamu.edu, dswaroop@tamu.edu, srathinam@tamu.edu

[2]David W. Casbeer is with the Controls Center, Air Force Research Laboratory, WPAFB, OH, USA david.casbeer@us.af.mil

[3]Satyanarayana G. Manyam is with Infoscitex corporation, a DCS Company, Dayton, OH, USA smanyam@infoscitex.com

This work was supported in part by AFOSR LRIR No. 21RQCOR084.

Distribution Statement A. Approved for public release, distribution unlimited. Case Number: AFRL-2022-0779.

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

to as the convoy[1] vehicle for the remainder of this paper, must leave a specified starting location and reach a given destination while minimizing some accumulated cost. The convoy will be assumed to have the capability to handle the obstructions present, but doing so will incur some cost in addition to the cost of travel. A second designated agent, referred to as the service vehicle for the remainder of this paper, can be simultaneously deployed to assist the convoy by clearing these obstructions itself as the convoy is traveling in the environment. As the service vehicle travels, it also incurs some cost. When the service vehicle clears an obstruction, it incurs some additional cost. Once an obstruction is cleared, it is assumed to remain cleared. The convoy may choose to wait at a vertex for some additional cost to allow the service vehicle to clear obstructions. The service vehicle may terminate at any vertex in the graph without continuing to incur additional cost as the convoy continues to its destination. The objective is then to find a pair of paths for the convoy and service vehicle such that the total cost accumulated for the two vehicles is minimized. For ease of discussion, we will refer to this problem as the Assisted Shortest Path Problem (ASPP).

Other practical applications of ASPP can include escort missions for the emergency vehicles such as an ambulance or a firetruck. These emergency vehicles can be assisted by a support vehicle to navigate through a traffic-congested road network to reach the desired location. The support vehicle can help by clearing the traffic in advance to save valuable time in critical situations. For such applications, the road network can be represented using a graph where the impeded edges correspond to paths with traffic congestion. Additionally, disaster rescue missions can also be modeled as an ASPP, where a rescue vehicle is assisted by a support vehicle during natural disasters such as wildfire, flooding, or earthquake. In general, ASPP can be applied to any operation where a support agent is used to improve the effectiveness/efficiency of the primary agent.

Similar problems such as multi-agent path planning in a dynamic environment or collaborative path planning have been extensively studied in recent years. Flying sidekick TSP [15] and cooperative routing for an air-ground vehicle team [16] are a few examples of collaborative path-planning problems where one agent alone might not be able to carry out the given task or be able complete it but at a very high cost. Agents are considered to be tightly coupled in such problems. For both the above-mentioned problems, authors have presented a MILP formulation to solve it optimally and provide a heuristic method to handle large-scale problems. MILP formulation is not feasible for the presented ASPP problem as the optimal solution may require the service vehicle to take a few edges multiple times. Other metaheuristic-based approaches have also been used to address similar problems. [17] presents a hybrid genetic algorithm to solve a ground vehicle-assisted multi-drone parcel delivery problem. [18] proposes a five-step path planning strategy to solve a cooperative UGV-UAV exploration problem with energy constraints. [19] combines the genetic algorithm with the estimation-of-distribution algorithm to solve a persistent surveillance problem. [20] uses a two-level memetic path planning algorithm to solve a cooperative detection problem by formulating it as a time-aware TSP.

It is also possible to draw similarities between the presented ASPP and multi-agent path planning problems in a dynamic environment. [21] presents a D^* lite [22] based exact algorithm to solve such a problem. [23] combines the Conflict-Based Search (CBS) algorithm [9] with lifelong planning A^* [24] to provide optimal solution in dynamic environment. [25] proposed a loosely synchronized search method that extends A^*-based MAPF planners to handle asynchronous actions. However, these approaches are not suitable for ASPP as they will fail to provide any plan for the service vehicle which does not have any destination and only exist to help the convoy. Other approaches include reinforcement learning based multi-agent path planning [26]–[28] but they cannot guarantee an optimal solution.

Restricted variations of the ASPP have been studied in [29] and [30]. In these papers, the convoy was not able to clear any obstructions by itself. This is the same as the cost of the convoy traversing any impeded edge being infinite. In [29], a mixed-integer linear programming formulation was presented. An approximation algorithm was presented in [30] for the case where the convoy may share an impeded edge with the service vehicle. The addition of the convoy's ability to traverse through impeded edges in this paper adds a significant level of complexity and allows for modeling more complex scenarios.

The main contribution of the paper is the development of a generalized permanent labeling algorithm (*GPLA*) that finds an optimal solution for the ASPP. Permanent labeling algorithms were introduced in [31] for the path planning of a *single* vehicle in a graph while adhering to various path structural constraints. In this approach [32], the state of the vehicle is stored as a label, and the label is updated as the vehicle moves to reflect the changes in the state. When a new label is generated by updating a previously existing label, it is referred to as a label extension. The key part of this approach lies in defining suitable labels and their extensions so that partial solutions that can potentially lead to an optimal solution are not discarded. In this paper, we extend this approach for *asynchronous* path planning for multiple agents and demonstrate this approach for two agents. We achieve this by using different clocks for the agents, where these separate clocks represent the time elapsed for each agent and are both stored in a label. With this scheme, a label extension represents a *decision* being made by one or both agents. In either case, the clocks associated with the agents may differ. These asynchronous labels and their corresponding extension step are shown in Section III. Using this scheme, we are able to significantly reduce the number of labels that would have otherwise been required to solve the ASPP using a traditional labeling algorithm. We introduce filters to further reduce the search space and improve algorithm performance. The proposed algorithm also serves as an *anytime* algorithm and can be terminated at any time to get the best feasible solution found so far. We refer to the final algorithm as *GPLA**. The procedure we present can be further generalized to handle

[1]For this problem, a convoy can be a single vehicle or a group of vehicles traveling together and therefore can be treated as a single agent.

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

more vehicles as needed.

The structure of this paper is as follows. Section II presents the mathematical formulation for the ASPP. Section III presents the *GPLA* and *GPLA**. Section IV presents the complexity analysis of the proposed algorithm. Section V presents a computational study for *GPLA**. Several families of problem instances are constructed to show the influence of various aspects of the ASPP on the runtime of *GPLA** and the structure of optimal solutions to the ASPP. *GPLA** is also compared to centralized A^* [33], which is an exact algorithm developed for solving cooperative multi-agent path planning problems using the MA-STRIPS model. It is shown that *GPLA** significantly outperforms centralized A^* both in the number of label extensions required and the overall computation time as a result. Section VI presents concluding remarks and future work.

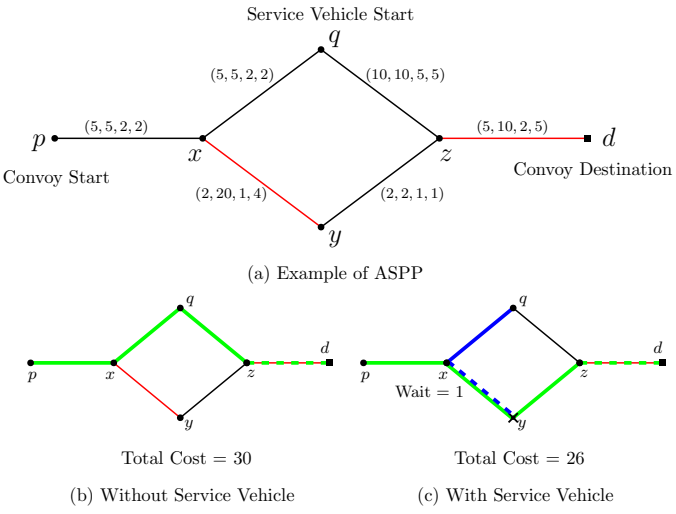
II. Problem Statement

Let $G=(V,E)$ be an undirected, connected graph representing an impeded environment. V is the set of vertices and E is the set of undirected edges. The convoy and service vehicle start at vertices p and q, respectively, where p and q need not be distinct. Let $d \in V$ be the destination of the convoy. The service vehicle may terminate at any vertex at any time and therefore has no destination. For the remainder of this paper, we will use cost and time interchangeably. Changes can be easily made to consider more general costs. Edges $K \subseteq E$ needing repair are known *a priori* and are referred to as impeded edges. The remaining edges are called unimpeded edges. We say an impeded edge is serviced if either the convoy or service vehicle has completely traversed that edge. In this paper, we use repair and service for an impeded edge interchangeably to indicate removing an obstruction (physical or abstract) in the impeded environment. We use the term impeded travel cost to refer to the cost for a vehicle to take an impeded edge that has not yet been serviced and use the term unimpeded travel cost otherwise. Each edge $e \in E$ has four positive, finite edge weights of the form $(T_e^u, T_e^i, \tau_e^u, \tau_e^i)$. T_e^u and T_e^i are the unimpeded and impeded travel cost for the convoy, respectively. The unimpeded and impeded travel costs for the service vehicle are τ_e^u and τ_e^i, respectively. All costs are non-negative and are known. For unimpeded edges, $T_e^i = T_e^u$ and $\tau_e^i = \tau_e^u$. For impeded edges, we have $T_e^i > T_e^u$ and $\tau_e^i > \tau_e^u$. Therefore, the cost of the convoy taking edge e is reduced if it is first serviced by the service vehicle. All edges are undirected so the cost in both directions are taken to be identical. When an impeded edge e is serviced, it remains serviced.

We assume the following: (1) the convoy and service vehicles can share vertices and edges without conflict, (2) the two vehicles start at the same time, (3) the two vehicles communicate at all times and information is shared in a negligible amount of time, (4) the service vehicle always travels faster than the convoy along any edge and, (5) the service vehicle will not wait for the convoy to repair an impeded edge. The convoy is allowed to wait at any vertex to give the service vehicle time to repair impeded edges. Waiting will incur some additional cost. In this paper, this waiting cost is simply the time the convoy waits.

Let X_{pa} be a path from p to $a \in V$ for the convoy and $\bar{X}_{qb}$ be a path from q to $b \in V$ for the service vehicle. We note that a vehicle's path will also potentially have waiting at one or more vertices. The two paths are coupled using the rules previously outlined. The cost of X_{pa} will depend on the decisions made by the service vehicle in path $\bar{X}_{qb}$ and vice-versa. We note that the service vehicle may also choose to remain at q. The accumulated cost of these two paths is the sum of the time the convoy reaches a and the time the service vehicle first reaches b (it may choose to terminate at b) while adhering to the motion rules and travel cost rules. This cost includes any waiting time by the convoy. In other words, the accumulated cost is the total cost of the vehicles being active. If the service vehicle never initially leaves q, the cost of $\bar{X}_{qq}$ is zero. We denote the total cost by $C(X_{pa}, \bar{X}_{qb})$. The ASPP is then to find two coupled paths X_{pd} and $\bar{X}_{qv}$, where v is any vertex in V, that minimizes $C(X_{pd}, \bar{X}_{qv})$.

A sample problem for the ASPP is shown in Fig. 1(a). Fig. 1(b) represents the least-cost path for the convoy, $X_{pd} = (p, x, q, z, d)$ with total cost 30, without the help of the service vehicle. The convoy will repair the edge (z,d) in this solution. Fig. 1(c) represents the optimal solution with the help of the service vehicle. The service vehicle will take the path $\bar{X}_{qy} = (q, x, y)$, repair the edge (x,y) and then terminate at y. The convoy will start from p and wait at x for 1 unit of time to let the service vehicle finish the repair and traverse the serviced edge (x,y). The convoy will then repair the edge (z,d) to reach the destination. This solution's cost is 26.



(a) Example of ASPP

Total Cost = 30

(b) Without Service Vehicle

Total Cost = 26

(c) With Service Vehicle

Fig. 1. (a) A sample instance for the ASPP. Black edges represent unimpeded edges and red edges represent impeded edges. The edge weights associated with each edge are of the form $(T_e^u, T_e^i, \tau_e^u, \tau_e^i)$. In (b) and (c), the convoy and the service vehicle paths are represented by green and blue colors, respectively. The dashed path represents the impeded edge is serviced by the vehicle associated with the corresponding color. In (b), the optimal solution for the convoy without the assistance of the service vehicle is shown. In (c), the optimal solution for the convoy with the assistance of the service vehicle is shown; in this case, the convoy will wait at node x for 1 unit of time. The service vehicle terminates its path at node y, which is represented by a cross. It is clear the solution in (c) is better than the solution in (b).

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

III. Generalized Permanent Labeling Algorithms

We first define an abstract object called a label to store relevant information on decisions made by both vehicles. Recall the definition of a path and the accumulated cost from Section II.

Definition 1 (Label)**.** *Suppose the convoy and service vehicle have taken paths X_{pi} and $\bar{X}_{qj}$ to i and j in G in times T_i and τ_j, including waiting, while accumulating a total cost C_{ij}. For each edge e serviced by either vehicle, create a tuple (e, s_e) where s_e is the time at which edge e was serviced. The set S_{ij} is the collection of the tuples (e, s_e), where $e \in K$. We then define a label λ_{ij} as*

$$\lambda_{ij} = (i, j, T_i, \tau_j, C_{ij}, S_{ij}).$$

For a label λ_{ij}, we define $K_S(\lambda_{ij})$ to be the set of serviced edges in S_{ij} and $\hat{K}(\lambda_{ij}) = K \setminus K_S(\lambda_{ij})$ represents the set of remaining impeded edges.

We next introduce resource extension functions (REFs) [31] associated with each label. REFs describe how a label λ_{ij} is extended to create a new label λ_{lm}. Extending a label corresponds to feasible extensions of the paths X_{pi} and $\bar{X}_{qj}$ from decisions involving edges (i,l) and (j,m), respectively. We allow for $i = l$ or $j = m$ to represent the agent's position not changing, but not both simultaneously[2].

Definition 2 (Resource Extension Functions)**.** *Suppose paths X_{pi} and $\bar{X}_{qj}$ associated with label λ_{ij} are extended by edges $e_x = (i,l)$ and $e_y = (j,m)$, respectively, to create a new label λ_{lm}. Then,*

$$T_l = \begin{cases} T_i + T^u_{e_x}, & e_x \in E \setminus K \\ T_i + \min(T^i_{e_x}, T^u_{e_x} + \max(0, s_{e_x} - T_i)), & e_x \in K_S(\lambda_{ij}) \\ T_i + T^i_{e_x}, & e_x \in \hat{K}(\lambda_{ij}) \\ \max(T_i, \tau_m), & i = l \ \wedge \ j \neq m \end{cases}$$

$$\tau_m = \begin{cases} \tau_j + \tau^i_{e_y}, & e_y \in \hat{K}(\lambda_{ij}) \\ \tau_j + \tau^u_{e_y}, & e_y \in E \setminus \hat{K}(\lambda_{ij}) \\ \max(\tau_j, T_l), & j = m \ \wedge \ i \neq l \end{cases}$$

$$C_{lm} = \begin{cases} C_{ij} + (T_l - T_i) + (\tau_m - \tau_j), & j \neq m \\ C_{ij} + (T_l - T_i), & j = m \end{cases}$$

$$E_{lm} = \begin{cases} E_{ij}, & e_x, e_y \in E \setminus \hat{K}(\lambda_{ij}) \\ E_{ij} \cup \{(e_x, T_l)\}, & e_x \in \hat{K}(\lambda_{ij}) \ \wedge \ e_y \in E \setminus \hat{K}(\lambda_{ij}) \\ E_{ij} \cup \{(e_y, \tau_m)\}, & e_x \in E \setminus \hat{K}(\lambda_{ij}) \ \wedge \ e_y \in \hat{K}(\lambda_{ij}) \\ E_{ij} \cup \{(e_x, T_l), (e_y, \tau_m)\}, & e_x, e_y \in I_{ij} \end{cases}$$

The REFs in Definition 2 encode the cost rules previously outlined in Section II and tracks the edges that have been serviced and the times they were serviced. Each label is associated with a trajectory pair, including the optimal solution. It is important to note here that the time elapsed for both vehicles need not be equal during the extension step. This is referred to as *asynchronous* path planning. Presented REFs can handle decisions from both vehicles during each label extension. This is an improvement over traditional extension strategies for asynchronous path planning where action from only one agent is considered in each extension step.

There are an infinite number of possible labels due to cycles and waiting. We introduce a dominance rule [31] to reduce the number of labels under consideration to a finite number.

Definition 3 (Dominance Rule)**.** *Consider two labels λ'_{ij} and λ_{ij}. We say λ'_{ij} dominates λ_{ij} if and only if*

1) $T'_i \leq T_i$
2) $\tau'_j \leq \tau_j$
3) $K_S(\lambda'_{ij}) \supseteq K_S(\lambda_{ij})$
4) $s'_e \leq s_e \quad \forall e \in K_S(\lambda_{ij})$

If we have equality for all the above conditions, then we consider $\lambda'_{ij} = \lambda_{ij}$, and one label can be discarded arbitrarily if only one optimal solution is required. If all optimal solutions are needed, then keep both labels.

Conditions 1 and 2 state that both vehicles have reached the same pair of vertices in less time. Note that we do not require a separate dominance condition for cost as it follows from Conditions 1 and 2. If a more general cost is used, an additional condition $C'_{ij} \leq C_{ij}$ must be included. Condition 3 states at least all serviced edges in λ_{ij} have also been serviced in λ'_{ij}. Condition 4 states each serviced edge in λ_{ij} was serviced earlier or at the same time in λ'_{ij}.

Theorem 1. *The extensions of only the non-dominated labels need to be considered to obtain the optimal solution.*

Proof. Let λ'_{ij} and λ_{ij} be two distinct labels with λ'_{ij} dominating λ_{ij}. We need to show any feasible extension of λ_{ij} will be dominated by the same extension of λ'_{ij}.

Both λ'_{ij} and λ_{ij} must have the same feasible extensions. Let the labels be extended by $e_x = (i,l)$ and $e_y = (j,m)$ for the convoy and service vehicle, respectively, resulting in new labels λ_{lm} and λ'_{lm}. We allow for $i = l$ and $j = m$, but not simultaneously. Using Definitions 2 and 3, we observe the following:

- If $i \neq l$ and $e_x \notin K_S(\lambda_{ij}) \cup K_S(\lambda'_{ij})$, then we have $T'_l \leq T_l$.
- Suppose $i \neq l$ and $e_x \in K_S(\lambda_{ij}) \cap K_S(\lambda'_{ij})$. From $s'_{e_x} \leq s_{e_x}$,

  $$T'_i + \min(T^i_{e_x}, T^u_{e_x} + \max(0, s'_{e_x} - T'_i)) \leq T_i + \min(T^i_{e_x}, T^u_{e_x} + \max(0, s_{e_x} - T_i)),$$

  which implies $T'_l \leq T_l$.
- Suppose $i \neq l$ and $e_x \in K_S(\lambda'_{ij}) \setminus K_S(\lambda_{ij})$. Then

  $$T'_i + \min(T^i_{e_x}, T^u_{e_x} + \max(0, s'_{e_x} - T'_i)) \leq T_i + T^i_{e_x},$$

  which implies $T'_l \leq T_l$.
- Suppose $j \neq m$. Since $K_S(\lambda'_{ij}) \supseteq K_S(\lambda_{ij})$ and $\tau'_j \leq \tau_j$, from Definition 2 we must have $\tau'_m \leq \tau_m$.
- Suppose $i = l$ and $j \neq m$. Then

  $$\max(T'_i, \tau'_m) \leq \max(T_i, \tau_m)$$

  which implies $T'_l \leq T_l$.
- Suppose $j = m$ and $i \neq l$. Then

  $$\max(\tau'_j, T'_l) \leq \max(\tau_j, T_l),$$

[2]Both $i = l$ and $j = m$ is not allowed because this condition implies both the vehicles are just waiting without performing any motion or servicing task. This clearly does not lead to an optimal solution.

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

which implies $\tau'_m \leq \tau_m$.

- The same extension is used so clearly $K_S(\lambda'_{ij}) \supseteq K_S(\lambda_{ij})$ implies $K_S(\lambda'_{lm}) \supseteq K_S(\lambda_{lm})$.
- Suppose $e_x \in K$.
  - If $e_x \notin K_S(\lambda_{ij}) \cup K_S(\lambda'_{ij})$, from $T'_l \leq T_l$ it follows that $s'_{e_x} \leq s_{e_x}$.
  - If $e_x \in K_S(\lambda'_{ij}) \setminus K_S(\lambda_{ij})$, then $s'_{e_x} \leq T'_i \leq T'_l \leq T_l$ and so $s'_{e_x} \leq s_{e_x}$.
- Suppose $e_y \in K$.
  - If $e_y \notin K_S(\lambda_{ij}) \cup K_S(\lambda'_{ij})$, from $\tau'_j \leq \tau_j$ it follows that $s'_{e_y} \leq s_{e_y}$.
  - If $e_y \in K_S(\lambda'_{ij}) \setminus K_S(\lambda_{ij})$, then $s'_{e_y} \leq \tau'_j \leq \tau'_m \leq \tau_m$ and so $s'_{e_y} \leq s_{e_y}$.

We see λ'_{lm} will always dominate λ_{lm}. Since the optimal solution must be a non-dominated label, we need to only consider non-dominated labels. □

From Theorem 1, we can solve the ASPP by repeatedly extending non-dominated labels, including the label corresponding to the initial configuration, until there are no more unique non-dominated labels that can be generated. Then, the non-dominated label that has the convoy position at d and with the lowest cost corresponds to an optimal solution.

We also make a few trivial observations for optimal solutions to the ASPP that will reduce the number of non-dominated labels to consider:

(i) The service vehicle, if deployed, will only terminate immediately after servicing all of its assigned impeded edges.
(ii) The convoy will only wait at an end of an impeded edge it uses later.
(iii) A convoy waiting to use an impeded edge will wait until the service vehicle has serviced that edge.
(iv) The convoy will never wait at a vertex after the service vehicle has terminated its motion.

These observations are a direct consequence of the structure of the cost rules from Section II. It should be noted these observations may not hold for more general cost structures and variants of the ASPP involving additional constraints. We note a label generated by extending a non-dominated label has the potential to never lead to an optimal solution but not be immediately discarded by the domination rule. We use the previous observations to recognize a label can be discarded despite it not currently being dominated by another previously generated label. To do so, we introduce two Boolean variables (flags), denoted by SV_{term} and δ, that are inserted into a label. These flags are then be used to recognize an extension cannot lead to an optimal solution by the previous observations. We then denote a label λ_{ij} by

$$\lambda_{ij} = (i, j, T_i, \tau_j, C_{ij}, S_{ij}, SV_{term}, \delta).$$

We note the flags themselves are not a part of the domination rule and do not affect the validity of Theorem 1 when used to discard partial solutions that only lead to sub-optimal solutions. The first flag SV_{term} denotes whether the service vehicle has terminated. Observations (i) and (iv) can be captured using SV_{term}. For the initial configuration label we set SV_{term} to be false. When extending a label with SV_{term} set to false, we generate all feasible extensions with SV_{term} set to false for each label. If any of the newly generated extensions corresponds to the service vehicle servicing an edge, we generate a copy of that extension with SV_{term} set to true. This corresponds to observation (i) above. We also create a copy of the initial label with SV_{term} set to true at the beginning of the algorithm to capture the case the service vehicle is never deployed. When extending a label with SV_{term} set to true, the service vehicle remains at its current position, the convoy will always move to a new position (i.e., we do not consider any extensions with the convoy remaining at its current position), and all extensions will have SV_{term} set to true. This corresponds to observation (iv) above. The second flag δ is used to capture the interaction between the convoy and service vehicle in a way that eliminates redundant decisions by the convoy that can never lead to an optimal solution and will correspond to observations (ii) and (iii) above. For the initial configuration label, we set δ to false. When extending a label λ_{ij} with δ set to false, if the convoy's current position i is not incident with an impeded edge that has yet to be serviced, then all generated extensions will have δ set to false. However, by observation (ii) we note we do not need to consider an extension with the convoy remaining at its current position in this case. If instead, the convoy's position i is incident with at least one impeded edge, then the extensions corresponding to the convoy remaining at i will have δ set to true. When extending a label with δ set to true, we generate labels where the convoy takes an edge that has been serviced (i.e., an edge $e \in K_S(\lambda_{ij})$) and set δ to be false for all such labels. We do not consider any labels where the convoy takes an unimpeded or impeded edge. If SV_{term} for λ_{ij} is set to false, we also generate all feasible extensions of λ_{ij} with the convoy remaining at its current position i and set δ to be true for all such labels. If a label generated in this way results in a scenario where i is no longer incident with an impeded edge that has yet to be serviced, then we set δ to be false. In a sense, δ being set to true represents the convoy "waiting" to make a decision when given the option to take an impeded edge. However, due to the asynchronous nature of the clocks associated with the two agents in each label, δ being set to true does not necessarily correspond to the convoy physically waiting at i. Rather, it represents a pause in the extension of the sequence of decisions associated with the convoy.

A. Generalized Permanent Labeling Algorithm - A^* ($GPLA^*$)

We now introduce two final modifications to the permanent labeling algorithm to further reduce the number of labels that need to be extended and the overall computation time as a result. These modifications are motivated by the A^* algorithm [34] used for solving the single agent shortest path planning problem. Therefore, we refer to *GPLA* with these modifications as $GPLA^*$. Algorithm 1 shows the pseudo-code for $GPLA^*$.

1) Early Termination: In *GPLA*, we repeatedly extend labels from L_{open} until we have exhausted all possible non-dominated labels. In general, the selection rule used to choose

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

labels from L_{open} to be extended can be arbitrary. The algorithm will always converge to the optimal solution irrespective of any selection rule. In permanent labeling algorithms, it is common to select the label most recently added to L_{open} (LIFO) or find the label in L_{open} with the least cost (best-first). The selection rule used can have a significant impact on the termination time of the algorithm.

For a single agent shortest path planning problem, the A^* algorithm uses a heuristic-cost-based extension method to reach the destination while reducing the search space. We use a similar approach. For any label λ_{ij}, we define the heuristic cost h_i to be a lower bound on the cost for the convoy to reach d from i. In our implementation, we have taken h_i to be the least cost path from i to d while treating all impeded edges as unimpeded, i.e., we set the edge weight to be T_e^u for each edge. We define f-cost for λ_{ij} to be the sum of C_{ij} and h_i and denoted by $f(\lambda_{ij})$. In other words, $f(\lambda_{ij})$ is a lower bound on the cost for the convoy to reach the destination from the state corresponding to λ_{ij}.

Lemma 1. *If a label λ_{ij} with the least f-cost is selected to be extended at every iteration, then the first such label with $i = d$ will correspond to an optimal solution to the ASPP.*

Proof. From Definition 2, for any extension from λ_{ij} to λ_{lm}, we have $C_{ij} \leq C_{lm}$. Therefore, from the definition of h_i, we can say that

$$\begin{aligned} & h_i \leq h_l + (C_{lm} - C_{ij}) \\ \Rightarrow & h_i + C_{ij} \leq h_l + C_{lm} \\ \Rightarrow & \ f(\lambda_{ij}) \leq f(\lambda_{lm}) \end{aligned}$$

which implies that the f-cost will never decrease with label extensions. Let λ_{dj} be the first label with the convoy at the destination selected to be extended. This implies that all the other labels will have f-cost greater than or equal to the f-cost of λ_{dj}. Note that f-cost of λ_{dj} is equal to C_{dj}. This implies that no other label can reach the destination with a total cost lower than C_{dj}. □

In order to utilize the result in Lemma 1, we pick the label with least f-cost in each iteration of *GPLA**. The heuristic cost h_i at each vertex $i \in V$ is computed before starting the algorithm. This is indicated by the input h in Algorithm 1.

2) Cost filter: As previously discussed, when extending a label we only keep the resulting non-dominated labels. Checking for dominance is computationally expensive due to the third and fourth conditions of Definition 3. To further reduce the number of dominance checks needed, we use the cost of a feasible solution to the ASPP as an upper bound (UB) to prematurely discard non-dominated labels. This feasible solution is computed as follows. First, we pre-compute the least cost path for the convoy to the destination d from each node $i \in V$. For this, we use Dijkstra's search [35] starting from d while assuming no help from the service vehicle, i.e. we set the edge weight to be T_e^i for each edge. Therefore, we start the *GPLA** with an upper-bound cost (UB) corresponding to the least-cost path from p to d assuming no help from the service vehicle. Next, for the selected node λ_{ij} in each iteration, we use the pre-computed least cost path for the convoy from i to d to get a new feasible solution. We accordingly update the cost of the pre-computed path if any impeded edge e in this path is also in $K_S(\lambda_{ij})$. If the cost of the new feasible solution is less than UB, then we update UB and store this solution as the best feasible solution found so far. This step is indicated by line 9 in Algorithm 1. When a label is generated, before checking for dominance we check if the f-cost exceeds the upper bound and if so we discard the label. As we keep improving on our UB, we can discard many non-dominated labels which will never lead to the optimal solution. Another advantage of this approach is that we always have a feasible solution as we go through the search space to find the optimal solution. For time-sensitive applications, such as emergency vehicle escort missions where it may not be feasible to wait for the optimal solution, we can terminate the algorithm at any time to get the best feasible solution found so far.

Algorithm 1 Generalized Permanent Labeling Algorithm - A^* (*GPLA**)

```
1:  Input: G(V,E), p, q, d, h, UB
2:  λ_start ← Initialization(p, q)
3:  D ← {λ_start}                          ▷ non-dominated labels
4:  L_open ← {λ_start}                     ▷ open list (sorted heap)
5:  while L_open ≠ ∅ do
6:      λ ← Select(L_open)
7:      if ConvoyPosition(λ) = d then
8:          return λ
9:      Update(UB)
10:     for n ← Extensions(λ) do           ▷ n = (l, m)
11:         λ' ← REF(λ, n)
12:         if f(λ') ≤ UB and λ' is NonDominated then
13:             L_open ← L_open ∪ {λ'}
14:             D ← D ∪ {λ'}
```

The generalized permanent labeling algorithm - A^* (*GPLA**) then works as follows. We create an initial label $\lambda_{pq} = (p, q, 0, 0, 0, \emptyset, \text{false}, \text{false})$. We initialize two lists L_{open} and D with λ_{pq}. L_{open} is the list of labels to be explored and D is the list of all non-dominated labels found so far. We then repeat the following steps until L_{open} is empty. In each iteration, we select the label λ_{ij} with the least f-cost from L_{open} (Lemma 1). For efficiency, we keep the L_{open} sorted based on the f-cost. If the selected label λ_{ij} has $i = d$, then we terminate the algorithm and return this label as the optimal solution. If not, we find a new feasible solution corresponding to λ_{ij} and update the UB if needed as described in Sec.III-A2. Next, we extended the selected label λ_{ij} to all pairs of neighbors (l, m) for both vehicles. These neighbors are selected based on SV_{term} and δ of λ_{ij}. Line 10 in Algorithm 1 represents this step as finding the extensions of λ_{ij}. For each extension, we get the corresponding label using the REFs of Definition 2. If the f-cost of the new label is less than the UB and is non-dominated, we add it to L_{open} and D. While adding a new label in L_{open}, to break the f-cost ties, we select the label with the highest accumulated cost (following [36]). To break any further ties, we use the most recently created label among the tied labels. The trajectories for the optimal solution

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

can be found by iteratively going through the predecessors of the label. Therefore, we require some way to store the predecessor label for any given label λ_{ij}. We have done this by directly including the parent label in the definition of λ_{ij} in our implementation. For any given graph G, the connectivity can be verified in polynomial time by using breadth-first search before running the algorithm to ensure there exists a feasible solution.

The proposed algorithm can be extended to handle multiple convoys and multiple service vehicles, as well as consider more complex interactions between the vehicles and environment. We have discussed the roadmap and associated challenges of extending this algorithm's framework to handle more difficult problems in the conclusion and future work in Section VI.

IV. Complexity Analysis

We now present the complexity associated with the proposed algorithm. Define $D = UB - LB$, where UB is the least-cost path for the convoy from p to d without deploying the service vehicle and LB is the least-cost path from p to d while treating all impeded edges as unimpeded (see Section III-A). We note UB and LB do not depend on the service vehicle. We see the number D indicates the maximum cost the service vehicle can accumulate before the combined cost of both vehicles exceeds the computed upper bound. We next define

$$k_1 = \left\lceil \frac{UB}{T_{min}} \right\rceil \tag{1}$$

and

$$k_2 = \left\lceil \frac{D}{\tau_{min}} \right\rceil \tag{2}$$

where $T_{min} = \min\{T_e^u \,|\, e \in E\}$ and $\tau_{min} = \min\{\tau_e^u \,|\, e \in E\}$. k_1 is the maximum number of edges the convoy can take before exceeding the upper bound on its own. Similarly, k_2 is the maximum number of edges the service vehicle can take before the convoy and service vehicle together will exceed the upper bound. To find the complexity, we then simply need to find an upper bound on the maximum number of labels to be explored by considering the maximum number of decisions (either taking an edge or pausing the change in state) for each vehicle.

We first focus on the convoy. For any vertex, the convoy has at most $n-1$ neighboring vertices it can move to. Additionally, as previously mentioned the convoy is able to take at most k_1 edges before exceeding the computed upper bound. Therefore, the number of paths the convoy can take is upper bounded by

$$1 + (n-1) + (n-1)^2 + \ldots + (n-1)^{k_1} = \frac{(n-1)^{k_1+1} - 1}{n-2} \tag{3}$$

Next, recall k_2 is the maximum number of edges the service vehicle can take before the total accumulated cost exceeds the upper bound solution. In the construction of the proposed algorithm, the convoy does not need to consider entering a 'waiting' state if the service vehicle has terminated its journey (see Section III, Observation V). As a result, the convoy can enter a 'waiting' state (i.e., have δ as true) at most k_2 times in the algorithm for each convoy path. Therefore, an upper bound on the number of labels with δ as true for any convoy path is given by $k_1 k_2$. Using this, we see an upper bound on the number of decisions (moves and 'waiting' states) for the convoy is given by

$$\frac{(k_1 k_2 + 1)\left[(n-1)^{k_1+1} - 1\right]}{n-2} \tag{4}$$

For the service vehicle, we note at each vertex the service vehicle has at most $n-1$ neighboring vertices it can move to and is allotted at most k_2 moves. Additionally, the service vehicle may terminate at any vertex. As a result, the service vehicle has at most n options at each vertex. Therefore, an upper bound on the number of decisions the service vehicle makes is given by n^{k_2}. Putting this together, an upper bound on the number of labels to be explored is given by

$$\frac{(k_1 k_2 + 1)\left[(n-1)^{k_1+1} - 1\right] n^{k_2}}{n-2} \tag{5}$$

These labels are effectively vertices on a higher-dimensional graph, which the proposed algorithm explores this graph using A^*. Therefore, the complexity of the algorithm is $\mathcal{O}((k_1 k_2)^2 n^{2(k_1+k_2)})$, which is pseudo-polynomial [37]. From (5), we can also say that there exists only a finite number of non-dominated labels. Therefore, from Lemma 1, we can conclude that the *GPLA** algorithm will terminate with the optimal solution in finite time.

We see k_1 and k_2 have a direct impact on the complexity. This is to be expected, as k_1 and k_2 are a measure of the number of allotted moves for both vehicles. As the number of allotted moves for both vehicles increases, the complexity of the algorithm explodes. It should be noted that k_1 and k_2 can take rather large values depending on the instance considered. This is especially true for graphs where the least-cost path for the convoy from p to d without deploying the service vehicle is long and involves many initially impeded edges (i.e., UB is large and LB is relatively small in comparison).

V. Computational Study

All algorithms were implemented[3] in Python 3.6 and the computations were done on an MSI laptop (8 Core Intel i7-7700HQ processor @ 2.80 GHz, 16 GB RAM). For the analysis, we used a grid structure (see Fig. 2) for the graph as it can easily represent a real-world scenario, like a warehouse, and is also easy to reproduce for verification. For all of the instances, the origin and the destination of the convoy, p and d, were chosen to be at diagonally opposite ends of the grid to avoid trivial cases. Impeded edges can be chosen randomly or strategically so that the convoy has to traverse through at least one impeded edge to reach the destination. This can be achieved by choosing the impeded edges to make a cut between p and d. A cut between p and d for an instance is defined as the set of edges $\{(a,b) \in E \,|\, a \in A, b \in B\}$ for any two disjoints sets $A, B \subset V$, such that $A \cup B = V$ and $p \in A$ and $d \in B$. An instance with c cuts is constructed so that

[3]Github repo: https://github.com/abhay1220/ASPP-with-GPLAstar

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

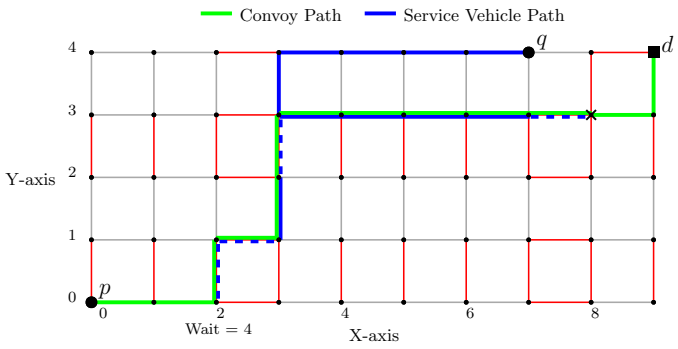


Fig. 2. A 5×10 grid instance with convoy and service vehicle paths for an optimal solution. Impeded edges are represented by the red color and the remaining edges are unimpeded. The service vehicle terminates its path at node $(8,3)$ which is represented by a cross. The dashed path represents the impeded edge is serviced by the vehicle associated with the corresponding color. In this example, the convoy waits at $(2,1)$ for 4 units of time. All edges have $T_e^u = 10$ and $\tau_e^u = 1$ and all impeded edges have $T_e^i = 40$ and $\tau_e^i = 6$. The cost of the optimal solution shown is 172.

the impeded edge set K only consists of edges from c randomly chosen cuts. To illustrate the effectiveness of *GPLA**, we compare it with centralized A^* [33], an exact algorithm for cooperative multi-agent path planning problems modeled using MA-STRIPS. Other exact centralized algorithms such as *Distoplan* [38] and *A#* [39] could have also been used, but as discussed in [13] they are not suitable references for comparison. Among distributed planners, *MAD-A** [33] is an exact algorithm known to the authors. However, as reported in [33], *MAD-A** is less efficient than centralized A^* in tightly coupled problems. This was also observed when *MAD-A** was applied to the ASPP; we therefore focus our comparison with centralized A^*.

The remainder of this section is as follows. We first show a comparison between the run times of *GPLA** and centralized A^*. Next, we analyze *GPLA** over three different classes of instances. In *Class 1*, we aim to test the computational limits of *GPLA** as we increase the graph size and the number of impeded edges. In *Class 2*, we want to study the significance of unavoidable impeded edges on the optimal solution cost. We accomplish this by increasing the number of cuts on a grid graph of fixed size. Finally, *Class 3* instances are designed to understand the impact of the service vehicle starting position on the optimal solution.

A. Comparison between GPLA* and centralized A*

To compare *GPLA** and centralized A^*, we have created instances with varied grid size and cuts as shown in Table I. Each row represents the average value over 50 randomly generated instances. The starting position of the service vehicle was chosen randomly for each instance. For the convoy, the unimpeded travel cost, T_e^u, was randomly chosen from the range $[10, 15]$ for all edges and the impeded travel cost, T_e^i, was randomly chosen from the range $[40, 50]$ for the impeded edges. For the service vehicle, the unimpeded travel cost, τ_e^u, was set to be 1 unit for all edges and the impeded travel cost, τ_e^i, was randomly chosen from the range $[2, 6]$ for the impeded edges. For unimpeded edges, $T_e^i = T_e^u$ and $\tau_e^i = \tau_e^u$. This cost structure was chosen to encourage collaboration between the vehicles while keeping the decision-making non-trivial.

TABLE I
COMPUTATIONAL TIME COMPARISON: *GPLA** VS CENTRALIZED A^*

Grid-Size	Cuts	T_{GPLA^*}	$T_{A_c^*}$	O_{GPLA^*}	$O_{A_c^*}$
6×6	2	0.05 s	0.59 s	914	2300
8×8	2	0.60 s	3.52 s	6008	15768
10×10	2	2.45 s	22.9 s	15723	46404
3×15	2	0.10 s	0.42 s	1623	4738
3×15	3	1.80 s	42.2 s	7787	28351
3×15	4	7.96 s	149 s	16972	66160

Table I shows the comparison between the two algorithms and highlights the effectiveness of *GPLA** over centralized A^*. Each row represents the average value over 50 randomly generated instances. T_{GPLA^*} and $T_{A_c^*}$ represents the average computational time for *GPLA** and centralized A^*, respectively. O_{GPLA^*} and $O_{A_c^*}$ shows the average number of extended states for both algorithms. From Table I, we see the performance advantage of *GPLA** over the centralized A^* approach. The key difference between the algorithms lies in the extension step. Centralized A^* and *MAD-A** [33] only consider a single agent's action for each extension. For *GPLA**, the REF presented in Definition 2 considers the action of both agents at once, resulting in fewer label extensions and an overall faster computation time. We now focus on *GPLA** for the remainder of this computational study.

B. Class 1 instances

To test the computational limits of the *GPLA**, we generated instances from 6×6 to 10×10 grid sizes. For each instance, the impeded edges were chosen randomly and we used the same cost structure as in Section V-A. The fraction of impeded edges is denoted by $|K|/|E|$. For each grid size, the fraction of impeded edges were varied to be 0.1, 0.2, 0.3, 0.4, and 0.5. We generated 50 random instances for each grid size and fraction of impeded edges. The starting position for the service vehicle was chosen randomly for each instance. An instance is said to be *successful* if it terminated with the optimal solution within 900 seconds. The success rate (γ) is defined as the fraction of all successful instances over the total number of instances. For a majority of the instances with $|K|/|E| = 0.1$ and 0.2, we observed that the impeded edges did not form a cut and so the convoy had at least one unimpeded path from p to d. Therefore, we do not include those results in our discussion.

The results for Class 1 are shown in Table II. In Table II, each entry in the time columns represents the average computation time of successful instances along with the standard deviation. We see the success rate decreases with increasing grid size and increasing fraction of impeded edges. The average computation time and the standard deviation increase with increase in grid size and fraction of impeded edges. For cases with relatively low success rate ($\gamma < 0.5$), the average computation time may not follow the same trend as the samples are skewed. From this set of instances, we see the computation time is significantly affected by the number of

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

TABLE II
CLASS 1 RESULTS

	$\|K\|/\|E\|$ = 0.3		$\|K\|/\|E\|$ = 0.4		$\|K\|/\|E\|$ = 0.5	
Grid Size	time (sec.)*	γ	time (sec.)*	γ	time (sec.)*	γ
6×6	0.1 ± 0.2	1.00	5.6 ± 21	1.00	27 ± 96	0.96
7×7	2.3 ± 11	1.00	39 ± 142	0.94	93 ± 180	0.66
8×8	10 ± 48	0.98	55 ± 146	0.84	100 ± 184	0.36
9×9	39 ± 96	0.98	72 ± 147	0.68	128 ± 255	0.16
10×10	51 ± 154	0.88	112 ± 223	0.48	127 ± 132	0.12

* time indicates the average computation time ± standard deviation.

impeded edges. This is to be expected, as *GPLA** must produce additional labels whenever a label is such that the convoy position is an end of an impeded edge. Similarly, an additional label is also generated whenever an extension is such that the service vehicle traverses an impeded edge, servicing it. As the number of impeded edges increases, the number of additional labels generated will begin to explode. The upper bound cost filter introduced for *GPLA** significantly reduces the number of redundant labels generated. However, as the graph size also increases the upper bound becomes less tight and so the algorithm begins to fail to discard redundant labels early. This behavior can be seen by noticing as the fraction of impeded edges increases for a fixed grid size, the computation time begins to grow at an exponential rate. Conversely, for a fixed fraction of impeded edges, the computation time grows more slowly as the grid size increases.

C. Class 2 instances

For this set of instances, we wish to determine how the presence of unavoidable impeded edges affects the optimal solution cost. To do this, we use a fixed grid of size 3×15 with the same cost structure defined in Section V-A for each instance. We then generated cuts to introduce unavoidable impeded edges for the convoy. The narrow grid structure was chosen as it is easier to produce cuts with fewer edges, which keeps the computational time reasonable as we increase the number of cuts. The service vehicle's starting position was chosen randomly for each instance. The number of cuts was varied form 1 to 5. We generated 50 random instances for each cut size.

TABLE III
CLASS 2 RESULTS

Cuts	$\|K\|/\|E\|$	OPT	OPT/UB	OPT/LB	$\sigma(OPT)$
1	0.05	191	0.89	1.05	6.19
2	0.10	196	0.82	1.08	6.48
3	0.14	201	0.77	1.10	6.94
4	0.18	205	0.71	1.12	6.90
5	0.22	207	0.68	1.14	7.56

The computational results for this set of instances is shown in Table III. Each row in Table III represents an average value over the 50 instances for a set number of cuts. Column $|K|/|E|$ indicates the fraction of impeded edges. An upper bound, UB, was computed by having the convoy take the shortest path from p to d without any assistance from the service vehicle. The gap between the cost of the optimal solution to the ASPP and UB gives an indicator of the benefit of deploying the service vehicle. Similarly, a lower bound, LB, was computed by treating all impeded edges as unimpeded and finding the shortest path for the convoy from p to d only using the unimpeded cost for all edges. Columns OPT/UB and OPT/LB represents the ratio of optimal cost, OPT, against the upper bound and the lower bound costs, respectively. We see as the number of cuts increases, the OPT/UB ratio decreases. This shows the effectiveness/benefit of the service vehicle in an impeded environment, especially as the number of unavoidable impeded edges increases. Column $\sigma(OPT)$ shows the standard deviation of the optimal cost. The increasing trend of $\sigma(OPT)$ indicates that the optimal solution cost is sensitive to the number of unavoidable impeded edges.

D. Class 3 instances

For this set of instances, we examine the impact of the starting position of the service vehicle on the optimal solution. For each instance, we again use a 3×15 grid, but we impose a fixed cost structure for all instances. All edges were assigned costs $T_e^u = 10$ and $\tau_e^u = 1$ and the impeded edges were all assigned costs $T_e^i = 40$ and $\tau_e^i = 6$. The convoy's starting position and destination were fixed at diagonally opposite ends with positions $(0,0)$ and $(14,2)$ (see Fig. 3). This cost structure and choice of positions were chosen to encourage collaboration between the convoy and service vehicle. We generated 50 different instances by randomly generating 3 cuts for each instance. For a given instance, we compute the optimal solution cost for each of the 45 possible starting positions for the service vehicle. We then computed the average optimal cost across all 50 instances for each of the 45 service vehicle starting positions.

Fig. 3 shows the average cost across 50 instances for each starting position in the grid. The convoy starting position and destination are marked at $(0,0)$ and $(14,2)$. Gaussian interpolation was used to generate a continuous map. We observe that the minimum and maximum average costs are achieved at vertex $(3,1)$ and $(14,0)$ respectively. Fig. 3 shows the starting position of the service vehicle will have a noticeable impact on the optimal solution. The magnitude of the impact will depend on the numerical values of the costs for the edges. We also note the best starting position for the service vehicle in this set of instances does not coincide with the starting position of the convoy. This is to be expected. If the service vehicle starts

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

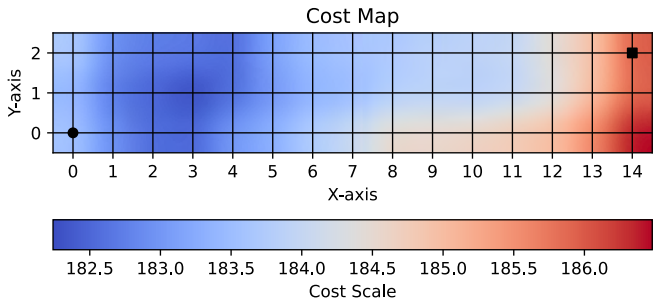


Fig. 3. Average cost map as a function of the starting position of the service vehicle.

some distance away from the convoy, it may be able to attend to more significant impeded edges before the convoy is able to reach them.

VI. Conclusion and Future Work

We have considered the problem of a convoy and service vehicle operating in an impeded environment, with the service vehicle handling the obstructions in the environment so that the convoy may reach a specified destination at a minimal cost. To solve this problem, we have introduced a labeling algorithm that is an extension of a labeling algorithm used for the single vehicle shortest path problem where various path structural constraints may be imposed. Our extension allows multiple vehicles to be considered, as well as allows for path structural constraints and non-trivial interactions between vehicles. In this paper, we only consider the case of two vehicles (the convoy and service vehicle) and a couple of non-trivial interactions between the two vehicles (the convoy is permitted to wait at any vertex and the cost of taking an impeded edge is reduced if another vehicle has taken that edge earlier in time). However, our proposed algorithm's framework can be further extended to consider more vehicles and more complicated path structural constraints and vehicle interactions. Extending this framework has significant challenges, but we believe further work towards these extensions will prove worthwhile. We now provide a roadmap to help facilitate further research in the extension of the algorithm's framework we have provided in this paper. We identify and discuss five major components of the framework and their challenges to consider more complex problems. The components are: (1) The graph, (2) the cost structure and rules of interaction, (3) the label definition, (4) label extension, and (5) filtering. For the rest of this section, we use the term "agents" rather than "vehicles" for the sake of abstraction.

The first major component of the framework is the graph definition. In the presented work, both agents existed on the same graph. However, in general, each agent may have its own graph that is not necessarily the same as the other(s). In fact, they need not share any edges in the most general case. Each agent's graph consists of vertices and edges, which represent waypoints and paths for that agent in the environment and these have been selected by a designer. Depending on the problem, these vertices and edges may have additional properties (such as a subset of edges being impeded in the presented work). For each agent's graph, there must be a source and, if applicable, a destination for the agent. The potential presence of multiple graphs, one for each agent, is the primary source of difficulty in this component when extending the framework for more complex cases. The additional attributes for vertices and edges also present difficulties, but this is more directly related to the label definition as is discussed in a moment.

The second major component is the cost structure and rules of interaction between agents. The cost of a solution must be properly defined. The cost of an agent's path may come from (1) the weights associated with the edges of the agent's graph and (2) any additional costs as a result of the agent's path structure and interactions between the other agents. Some examples of path structural costs are waiting costs, costs of revisiting vertices or re-using edges, etc. Some examples of costs for interactions between agents' paths are reducing costs of impeded edges after servicing, additional costs for using the same vertices or edges, etc. In addition to this cost structure, we must also define how the agents are permitted to move within their respective graph. We have grouped rules of interaction with the cost structure as the rules themselves may potentially lead to ambiguity in the cost of an agent's path. The designer must be aware of this and remove any such ambiguities. For example, suppose there were two service vehicles in the ASSP. If the two service vehicles are taking an impeded edge at the same time, the cost of that pair of decisions may be ambiguous and so an additional clause must be specified to remove the ambiguity. The primary difficulties when extending this framework comes from (1) stating all rules of interaction clearly and (2) removing all sources of ambiguity in the cost based on these rules.

The third major component is the label definition. The designer must identify a set of properties of the set of partial paths of the agents such that it (1) describes the cost of these paths, (2) contains necessary information of past decisions by the agents to determine if another set of decisions, one per agent, is permitted under the rules of the problem, (3) is capable of identifying all goals have been reached, and (4) is capable of retrieving all past decisions leading to these paths. There are two primary sources of difficulty in this component when extending. First, depending on the rules of the problem, it may be non-trivial to determine the information required to identify if an agent's next action will be legal. Even when this information has been identified, it may be non-trivial to determine the most efficient way to store this information for the algorithm's performance. Second, it may be difficult to make this set of properties as small as possible to improve the algorithm's performance.

The fourth major component is the label extension. The label extension allows for partial paths to be extended until a feasible solution to the problem has been found. Once optimality has been determined, the algorithm can be terminated. The extension of each agent's path given a set of actions, one per agent, is represented mathematically by resource extension functions (REFs), which depend on all of the previous components. For a given label and set of legal actions, one per agent, these REFs produce a new label representing the newly extended partial paths of the agents as

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

a result of these actions. When defining these REFs, they must (1) update all information from the input label appropriately based on the legal input actions of the agents and the past information stored in the label and (2) consider all possible combinations of legal actions of the agents for any possible scenario in the problem. The primary source of difficulty when extending the framework comes from ensuring all possible scenarios have been considered. If even a single scenario is not considered, the resulting algorithm may produce optimal solutions for many cases and only fail in a very small number of cases, making it difficult to diagnose. Additionally, it may be non-trivial to mathematically express how to update all properties within a label for different rules of interaction. This is especially true when making use of asynchronous clocks, as oftentimes path structural considerations make use of the time certain events occur (such as when an edge has been serviced in the presented work). The use of more than two agents only compounds this issue further.

The fifth and final major component is filtering. The previous components will lead to an algorithm that can solve the problem to optimality. To do so, all possible labels would need to be generated and the best solution from this exhaustive list would then need to be identified. This has two major issues. Firstly, if we generate an optimal label we would like to terminate early. Secondly, depending on the problem, there may be an infinite number of feasible solutions and so we cannot iterate through them all. We may turn to filtering to significantly reduce the search space by (1) identifying a subset of labels that need to be considered and (2) further reducing this subset by use of observations that some of these labels can never lead to an optimal solution. First, the designer may identify a dominance rule for the labels. This dominance rule must be accompanied by a proof that only non-dominated labels need to be considered (i.e., we never remove a label that could potentially lead to an optimal solution). By doing so, we significantly reduce the search space, potentially from an infinite space to a finite space. The designer should be aware that invoking the dominance check itself may be computationally expensive. Second, if the designer is able to identify certain properties of the optimal solution to their problem, they may make use of these properties to further discard newly generated labels, such as the observations used in Section III. Additionally, the designer may make use of an A^*-like approach by introducing an appropriate heuristic to further discard newly generated labels (even before invoking a dominance check). By making use of an A^*-like approach the algorithm may also be terminated significantly earlier. Identifying an appropriate dominance rule and any observations for the optimal solution structure are the primary sources of difficulty of this component. If making use of an A^*-like approach, finding a suitable heuristic may also be non-trivial.

REFERENCES

[1] J. Berger and N. Lo, "An innovative multi-agent search-and-rescue path planning approach," *Computers & Operations Research*, vol. 53, pp. 24–31, 2015.

[2] L. Parker, *Heterogeneous multi-robot cooperation*. PhD thesis, MIT, 1994.

[3] L. Steels, "Cooperation between distributed agents through self-organization," *IEEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*, pp. 8–14, 1990.

[4] L. E. Parker, "Cooperative robotics for multi-target observation," *Intelligent Automation & Soft Computing*, vol. 5, no. 1, pp. 5–19, 1999.

[5] J. Li, G. Deng, C. Luo, Q. Lin, Q. Yan, and Z. Ming, "A hybrid path planning method in unmanned air/ground vehicle (uav/ugv) cooperative systems," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 12, pp. 9585–9596, 2016.

[6] E. Garcia and D. Casbeer, "Coordinated threat assignments and mission management of unmanned aerial vehicles," *Cooperative Control of Multi-Agent Systems: Theory and Applications*, p. 141, 2017.

[7] T. Chung, "Offensive swarm-enabled tactics (offset)," *DARPA Tactical Technology Office Proposers Day, Arlington, VA, Accessed January*, vol. 30, 2017.

[8] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artificial intelligence*, vol. 219, pp. 1–24, 2015.

[9] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[10] B. De Wilde, A. W. Ter Mors, and C. Witteveen, "Push and rotate: cooperative multi-agent path planning," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pp. 87–94, 2013.

[11] R. I. Brafman and C. Domshlak, "From one to many: Planning for loosely coupled multi-agent systems.," in *ICAPS*, vol. 8, pp. 28–35, 2008.

[12] D. L. Kovacs, "A multi-agent extension of pddl3.," *WS-IPC 2012*, p. 19, 2012.

[13] A. Torreño, E. Onaindia, A. Komenda, and M. Štolba, "Cooperative multi-agent planning: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–32, 2017.

[14] B. Donald, J. Jennings, and D. Rus, "Analyzing teams of cooperating mobile robots," *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1896–1903, 1994.

[15] C. C. Murray and A. G. Chu, "The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery," *Transportation Research Part C: Emerging Technologies*, vol. 54, pp. 86–109, 2015.

[16] S. G. Manyam, K. Sundar, and D. W. Casbeer, "Cooperative routing for an air–ground vehicle team—exact algorithm, transformation method, and heuristics," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 1, pp. 537–547, 2019.

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

[18] F. Ropero, P. Muñoz, and M. D. R-Moreno, "Terra: A path planning algorithm for cooperative ugv–uav exploration," *Engineering Applications of Artificial Intelligence*, vol. 78, pp. 260–272, 2019.

[19] Y. Wu, S. Wu, and X. Hu, "Cooperative path planning of UAVs & UGVs for a persistent surveillance task in urban environments," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4906–4919, 2020.

[20] J. Li, T. Sun, X. Huang, L. Ma, Q. Lin, J. Chen, and V. C. Leung, "A memetic path planning algorithm for unmanned air/ground vehicle cooperative detection systems," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 2724–2737, 2021.

[21] K. Al-Mutib, M. AlSulaiman, M. Emaduddin, H. Ramdane, and E. Mattar, "D* lite based real-time multi-agent path planning in dynamic environments," in *2011 third international conference on computational intelligence, modelling & simulation*, pp. 170–174, IEEE, 2011.

[22] S. Koenig and M. Likhachev, "Dˆ* lite," *AAAI/IAAI*, vol. 15, pp. 476–483, 2002.

[23] Q. Wan, C. Gu, S. Sun, M. Chen, H. Huang, and X. Jia, "Lifelong multi-agent path finding in a dynamic environment," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 875–882, IEEE, 2018.

[24] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

[25] Z. Ren, S. Rathinam, and H. Choset, "Loosely synchronized search for multi-agent path finding with asynchronous actions," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9714–9719, IEEE, 2021.

[26] H. Qie, D. Shi, T. Shen, X. Xu, Y. Li, and L. Wang, "Joint optimization of multi-UAV target assignment and path planning based on multi-agent reinforcement learning," *IEEE access*, vol. 7, pp. 146264–146272, 2019.

[27] Z. Liu, B. Chen, H. Zhou, G. Koushik, M. Hebert, and D. Zhao, "Mapper: Multi-agent path planning with evolutionary reinforcement

IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2024, Yokohama, Japan.

learning in mixed dynamic environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11748–11754, IEEE, 2020.

[28] S. H. Semnani, H. Liu, M. Everett, A. De Ruiter, and J. P. How, "Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3221–3226, 2020.

[29] C. Montez, S. Rathinam, S. Darbha, and D. Casbeer, "Finding shortest paths for a team of convoy and repair vehicles," in *AIAA Scitech 2021 Forum*, p. 1769, 2021.

[30] C. Montez, S. Rathinam, S. Darbha, D. Casbeer, and S. G. Manyam, "An approximation algorithm for an assisted shortest path problem," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8024–8030, IEEE, 2021.

[31] M. Desrochers and F. Soumis, "A generalized permanent labelling algorithm for the shortest path problem with time windows," *INFOR: Information Systems and Operational Research*, vol. 26, no. 3, pp. 191–212, 1988.

[32] S. Irnich and G. Desaulniers, "Shortest path problems with resource constraints," in *Column generation*, pp. 33–65, Springer, 2005.

[33] R. Nissim and R. Brafman, "Distributed heuristic forward search for multi-agent planning," *Journal of Artificial Intelligence Research*, vol. 51, pp. 293–332, 2014.

[34] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[35] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pp. 287–290, Association for Computing Machinery, 2022.

[36] Z. Zhang, N. R. Sturtevant, R. C. Holte, J. Schaeffer, and A. Felner, "A* search with inconsistent heuristics.," in *Twenty-First International Joint Conference on Artificial Intelligence*, pp. 634–639, 2009.

[37] A. Martelli, "On the complexity of admissible search algorithms," *Artificial Intelligence*, vol. 8, no. 1, pp. 1–13, 1977.

[38] E. Fabre, L. Jezequel, P. Haslum, and S. Thiébaux, "Cost-optimal factored planning: Promises and pitfalls," in *Twentieth International Conference on Automated Planning and Scheduling*, 2010.

[39] L. Jezequel and E. Fabre, "A#: A distributed version of A* for factored planning," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 7377–7382, IEEE, 2012.

Abhay Singh Bhadoriya received the B.Tech degree in Chemical Engineering from Indian Institute of Technology, Bombay, Mumbai, India, in 2017. Currently, he is pursuing the Ph.D. degree in Mechanical Engineering at Texas A&M University, College Station, TX, USA. His research interest include path planning, controls and perception models for autonomous vehicles. He was also awarded the Emil Beuhler Aerodynamic Analog Fellowship by Texas A&M University in 2019.

Christopher M. Montez received the B.S and Ph.D. degrees in Mechanical Engineering at Texas A & M University in 2018 and 2023, respectively. He completed his Ph.D. under the guidance of Dr. Swaroop Darbha. His research interests include optimization, mathematical programming, and vehicle routing. He currently works at OpsLab as an Operations Research Engineer.

Sivakumar Rathinam (Senior Member, IEEE) received the Ph.D. degree in Civil Engineering from the University of California, Berkeley, CA, USA, in 2007. He is currently a Professor in the Department of Mechanical Engineering, Texas A&M University, College Station, TX, USA. He was a Research Scientist with the NASA Ames Research Center in California from 2007 to 2008. He has been with Texas A & M since 2009. His research interests include autonomous vehicles, motion planning, optimization, vision-based control, and air traffic control. Dr. Rathinam was the recipient of faculty fellowships from ONR and AFRL, and the best paper award in the 2015 International Conference on Unmanned Aircraft Systems. He is currently an Associate Editor for the IEEE Transactions on Automation Science and Engineering, IEEE Transactions on Intelligent Transportation Systems, and the ASME Journal on Dynamic Systems, Measurement, and Control.

Swaroop Darbha (Fellow, IEEE) received the B.Tech. degree from the Indian Institute of Technology, Madras, Chennai, India, in 1989, and the M.S. and Ph.D. degrees from the University of California at Berkeley, Berkeley, CA, USA in 1992 and 1994 respectively, all in mechanical engineering. He was a Postdoctoral Researcher with the California PATH program from 1995 to 1996. Since 1997, he has been with the faculty of Mechanical Engineering, Texas A&M University, College Station, TX, USA, where he is currently the Gulf Oil/Thomas A. Dietz Professor. His current research interests lie in the development of vehicle control and diagnostic systems for autonomous ground vehicles, development of planning, control and resource allocation algorithms for a collection of unmanned aerial vehicles.

David W. Casbeer received the B.S. and Ph.D. degrees in electrical engineering Brigham Young University, Provo, UT, USA, in 2003 and 2009, respectively. He is the team lead for the UAV Cooperative and Intelligent Control Team within the Control Science Center , Air Force Research Laboratory's Aerospace Systems Directorate, Wright-Patterson Air Force Base, OH, USA. The UAV team focuses on decision making, planning, and coordination for multiple autonomous UAVs acting and reacting in uncertain and adversarial environments. He is currently a Senior Editor for the Journal of Intelligent and Robotic Systems and an Editor for the AIAA Journal of Aerospace Information Systems. His team was awarded the AFOSR Star Team award for 2018–2020 for excellence in basic research.

Satyanarayana G. Manyam received the B.E degree in mechanical engineering from the Birla Institute of Technology and Science, Pilani, India, the M.S degree in mechanical engineering from State University of New York at Buffalo, Buffalo, NY, USA, and the Ph.D degree in mechanical engineering from Texas A&M University, College Station, TX, USA, in 2015. He is currently a Research Scientist with Infoscitex corporation, Dayton, OH, USA, and a Contractor for Air-Force Research Laboratories (AFRL), WrightPatterson Air Force Base. He was a NRC Postdoctoral Research Associate with Controls Center of Excellence, AFRL. His main research interests include cooperative routing and control of multiagent systems, optimal path and trajectory planning for autonomous vehicles and approximation algorithms. Dr. Manyam was the recipient of the Best Paper Award at the International Conference on Unmanned Aircraft Systems, 2015.