Optimal Vehicle Routing and Scheduling with Precedence Constraints and Location Choice

G. Ayorkor Korsah, Anthony Stentz, M. Bernardine Dias, and Imran Fanaswala

***Abstract*— To realize the vision of intelligent transportation systems with fully automated vehicles, there is a need for high-level planning for single vehicles as well as fleets of vehicles. This paper addresses the problem of optimally assigning and scheduling a set of spatially distributed tasks to a fleet of vehicles working together to achieve a high-level goal, in domains where tasks may be related by precedence or synchronization constraints and might have a choice of locations at which they can be performed. Such problems may arise, for example, in disaster preparedness planning, transportation of people, and delivery of supplies. We present a novel mathematical model of the problem and describe how it can be solved optimally in a branch-and-price framework.**

I. Introduction

Intelligent transportation systems comprising fully automated vehicles hold promise for improved efficiency, safety, and convenience over current systems. For this potential to be realized there is a need for algorithms for high-level planning for single vehicles and fleets of vehicles, in addition to sophisticated sensing, localization and navigation.

Traditionally, vehicle routing problems (VRPs) have addressed the problem of computing efficient routes for the transportation of people and goods by a fleet of vehicles. To realize the vision of truly intelligent transportation systems, approaches to these problems must address an increasingly richer set of constraints that may arise in various problem domains. While some tasks are independent of each other, others might be related by precedence or synchronization constraints. For example, during the evacuation of a special needs population in anticipation of a disaster, medical personal might need to visit some patients before they can be transported to shelters. Additionally, many tasks may have a pre-specified location at which they must be performed, while others may have a choice of a small set of locations where they may take place. For example, there might be choice of shelters to which individuals can be evacuated in anticipation of a disaster.

Inspired by optimal mathematical programming approaches from the Operations Research literature, this paper presents a set-partitioning model for the problem of allocating, scheduling and choosing locations for mutually-constrained tasks to be performed by a fleet of vehicles. This model enables finding a bounded optimal solution, considering the value of tasks completed, travel costs, as well as any costs due to waiting time needed to ensure that timing constraints are satisfied. We present a branch-and-bound approach to solving small instances of this problem, as well as a branch-and-price approach for larger instances.

This work is supported by the Qatar National Research Fund under contract NPRP 1-7-7-5.

G. A. Korsah (previously published as G. A. Mills-Tettey), A. Stentz, and M. B. Dias are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA. (Emails: ayorkor@cmu.edu, axs@ri.cmu.edu, mbdias@ri.cmu.edu). I. Fanaswala is with the Computer Science Department, Carnegie Mellon University, Doha, Qatar (Email: imranf@qatar.cmu.edu).

II. Problem Features and Example

When an area needs to be evacuated on the threat of a disaster such as a hurricane, the population of people with special needs, who are not able to make their own evacuation plans, requires particular attention. Individuals may have special transportation or sheltering needs that must be considered during planning. Considering available transportation options (e.g. vans, ambulances, helicopters), available support teams, and available shelters, an evacuation plan for this population will determine which vehicle will pick up which individuals and when. It will schedule any support teams (e.g. medical personnel) which need to be available before, at the time of, or after pickup or drop-off of an individual. It will also determine which shelter each individual will be taken to, considering the individual's particular requirements. With appropriate databases of the special needs population, an optimal evacuation plan can be created ahead of time, and this optimal plan can be adjusted as needed in the event of an actual disaster.

We consider a problem in which a set of agents, K (comprising automated vehicles and other entities in the team), is available to perform a collection of tasks, J. Each task $j \in J$ consists of one or more spatially distributed subtasks that must be performed in a given order. For example, a medical visit task comprises a single subtask, whereas transporting a customer comprises a pickup subtask and a drop-off subtask. Different tasks are suited to different types of agents in the system – medical tasks cannot be performed by automated transportation agents and vice-versa. Each subtask, $i \in I$ may have a fixed location at which it might be performed, or a choice of a very small set of locations L_i at which it may be performed. Subtasks might have time windows constraining their start time, and in the case of transporting items from one location to another, subtasks might use up a finite capacity available on the assigned agent. Pairs of subtasks in the problem might be related by precedence or synchronization constraints, thus creating constraints between different agents' schedules. These constraints need to be considered in assigning agents to tasks, and may result in delays in the agents' schedules, which may increase the cost, or conversely, reduce the total value of the solution.

III. Related Work

Vehicle routing problems (VRPs) address the transportation of passengers or the distribution of goods between depots and final users. VRPs can be expressed as mixed integer programming problems (MIP), defined on a graph in which the nodes correspond to locations of tasks to be performed, and edges correspond to travel segments between these locations. These mathematical models enable the formulation of optimal solution approaches. Proposed mathematical models can be broadly categorized as 3-index models and 2-index (or set-partitioning) models. For example, Cordeau [1] defines, for the dial-a-ride (DARP) problem (a variant of the VRP), a 3-index binary variable x_{ij}^k which is equal to 1 if vehicle k travels from node i to node j in the final solution. In contrast, Savelsbergh and Sol for the DARP [2] propose a set-partitioning model in which Ω_k is the set of feasible routes for vehicle k, and the 2-index variable x_r^k is a binary decision variable that takes on the value 1 if route $r \in \Omega_k$ is performed by vehicle k and 0 otherwise. Each route in Ω_k is a path through a subset of nodes, and is feasible in that all capacity and time constraints are satisfied along the route. Note that as the problem size grows, the number of feasible routes is usually too large to enumerate exhaustively. Rather a small set of feasible routes is initially used, and subsequently, additional "profitable" feasible routes are computed by a *pricing sub-problem*, and the *master* (set-partitioning) problem then selects a minimal cost set of routes satisfying the constraint that each customer must be serviced by only one vehicle.

Recent work in the vehicle routing literature has considered precedence constraints and synchronization constraints. In particular, Bredstrom and Ronnqvist present two different approaches. In one case [3], they create a three-index formulation of a vehicle routing problem, taking into consideration timing/synchronization constraints between individual tasks, each of which occurs at a fixed location. In this model, it is possible to penalize waiting time. In another case [4], they present a set-partitioning formulation that takes into consideration precedence constraints. However, in this model, waiting time cannot be penalized because time variables do not appear in the master problem formulation and so cannot be put in the objective function. Neither model addresses location choice. This paper presents a set-partitioning model which addresses location choice and precedence (and/or synchronization) constraints, while also being able to penalize waiting time as needed.

IV. Mathematical Model

We present a set-partitioning model with side constraints for this problem. The set-partitioning model, while representing complete feasible routes with single variables, also exposes time variables in the master problem formulation, thus allowing waiting time to be penalized by putting wait time variables in the objective function. We adopt the terminology of the vehicle routing literature and use the term *route* to represent a single agent's plan – that is, a sequence of subtasks that the agent / automated vehicle will perform at given locations according to the computed schedule.

In a set-partitioning approach, feasible routes for agents are represented by columns in the mixed integer linear program. In particular, a binary variable x_r^k indicates whether an agent k performs a given route r chosen from among all possible routes R_k that can feasibly be performed by agent k. In our problem, a feasible route is an ordered set of subtasks to be performed at chosen locations, such that all subtasks corresponding to the same task occur on the same route, time constraints are not violated, and agent capacity constraints are also not violated. A typical set-partitioning formulation would consist of these variables alone, with constraints specifying that each agent must perform only one route, and each task must appear on only one route.

In our formulation, however, we include additional time variables that appear in side constraints enforcing the precedence between subtasks that may appear on different routes. The real-valued variable w_i^k represents the amount of time that agent k, having arrived at the chosen location for subtask i, has to wait before it can begin execution of subtask i. This waiting time might be due to precedence constraints involving other subtasks being performed by other agents, or it might be because subtask i has a specific time window during which it must be performed. The waiting time is 0 if agent k is not assigned to subtask i. The real-valued variable t_i represents the time that execution begins on subtask i. If subtask i is not executed in the optimal solution, t_i is 0. In addition to the domain variables x_r^k, w_i^k, and t_i, the model includes helper variables $d_{i'i}$ representing the delay that the waiting time for subtask i' causes to a subtask i occurring later on the same route. If subtasks i' and i are not on the same route in the chosen solution, $d_{i'i}$ is 0. The delay variables are needed to ensure a linear formulation.

In the model below, v_j represents the value of completing a task j, which may of course comprise more than one subtask. c_{1r}^k represents the total travel cost of the route $r \in R_k$, and c_{2r}^k represents the waiting cost per unit time for agent k. The indicator π_{jr}^k is 1 if task j occurs on route $r \in R_k$ and 0 otherwise. Similarly, γ_{ilr}^k is 1 if subtask i occurs at location l on route $r \in R_k$ and 0 otherwise, and $\delta_{i'ir}^k$ is 1 if subtask i' occurs before subtask i on route $r \in R_k$ and 0 otherwise. The value τ_{ilr}^k represents the time that subtask i would be started on route $r \in R_k$ assuming no wait time was necessary, and τ_∞ represents the largest possible time in the model, that is, the end of the planning horizon. The value W_i represents the maximum allowed waiting time for subtask i; α_{il} and β_{il} represent the earliest and latest times respectively that service can begin on subtask i when it is performed at location l. λ_{il}^k represents the service time for subtask i when it is performed at location l by vehicle k. In the model below, we use λ_i and y_i respectively as placeholders for the following expressions:

$$\lambda_i = \sum_{k \in K} \sum_{r \in R_k} \sum_{l \in L_i} \lambda_{il}^k \gamma_{ilr}^k x_r^k$$

$$y_i = \sum_{k \in K} \sum_{r \in R_k} \sum_{l \in L_i} \gamma_{ilr}^k x_r^k \equiv \sum_{k \in K} \sum_{r \in R_k} \pi_{jr}^k x_r^k$$

(where j is the task to which subtask i belongs)

That is, λ_i is the service time of subtask i in the chosen solution (0 if i is not performed), and y_i indicates whether or not subtask i is performed in the selected solution. Finally, P represents the set of precedence constraints in the problem. Each precedence constraint $p = (i', i) \in P$ indicates that execution of subtask i' must end at least $\epsilon^P_{i'i}$ time units before service begins on subtask i.

The variables and defined quantities appearing in the mathematical model are summarized in Table I.

Maximize:

$$\sum_{j\in J}\sum_{k\in K}\sum_{r\in R_k} v_j \pi^k_{jr} x^k_r - \sum_{k\in K}\sum_{r\in R_k} c^k_{1r} x^k_r - \sum_{i\in I}\sum_{k\in K} c^k_2 w^k_i \quad (1)$$

Subject to:

$$1 = \sum_{r\in R_k} x^k_r \qquad \forall k \in K \qquad \text{(C1)}$$

$$1 \geq \sum_{k\in K}\sum_{r\in R_k} \pi^k_{jr} x^k_r \qquad \forall j \in J \qquad \text{(C2)}$$

$$w^k_i \leq W_i \sum_{l\in L_i}\sum_{r\in R_k} \gamma^k_{ilr} x^k_r \qquad \forall i \in I, k \in K \qquad \text{(C3)}$$

$$t_i = \sum_{l\in L_i}\sum_{k\in K}\sum_{r\in R_k} \tau^k_{ilr}\gamma^k_{ilr} x^k_r + \sum_{i'\in I} d_{i'i} + \sum_{k\in K} w^k_i \qquad \forall i \in I \qquad \text{(C4a)}$$

$$t_i \geq \sum_{l\in L_i} \alpha_{il} \sum_{k\in K}\sum_{r\in R_k} \gamma^k_{ilr} x^k_r \qquad \forall i \in I \qquad \text{(C4b)}$$

$$t_i \leq \sum_{l\in L_i} \beta_{il} \sum_{k\in K}\sum_{r\in R_k} \gamma^k_{ilr} x^k_r \qquad \forall i \in I \qquad \text{(C4c)}$$

$$d_{i'i} \geq \sum_{k\in K} w^k_{i'} - W_{i'} \sum_{k\in K}\sum_{r\in R_k} (1 - \delta^k_{i'ir} x^k_r) \qquad \forall i', i \in I \qquad \text{(C5a)}$$

$$d_{i'i} \leq W_{i_1} \sum_{k\in K}\sum_{r\in R_k} \delta^k_{i'ir} x^k_r \qquad \forall i', i \in I \qquad \text{(C5b)}$$

$$d_{i'i} \leq \sum_{k\in K} w^k_{i'} \qquad \forall i', i \in I \qquad \text{(C5c)}$$

$$y_{i'} \geq y_i \qquad \forall (i', i) \in P \qquad \text{(C7a)}$$

$$t_{i'} \leq t_i - \lambda_{i'} - \tau_\infty(y_i - y_{i'}) - \epsilon^P_{i'i}(y_i + y_{i'} - 1) \qquad \forall (i', i) \in P \qquad \text{(C7b)}$$

The objective function (1) strives to maximize the difference between overall reward and overall travel and waiting cost. (C1) specifies that each agent is assigned to exactly one route (which may be an empty route, allowing the solution to choose not to use a given vehicle). (C2) specifies that each task is assigned to at most one agent, and can in fact be rejected by assigning it to no agent. These two are the standard set-partitioning constraints. (C3) represents the bound on the waiting times. (C4a) computes the start time for a subtask, while (C4b) and (C4c) represent the time window bounds for the subtask. (C5a-C5c) represent constraints on the delay variables, which enable the start time of each subtask to be computed correctly, taking into consideration the wait time of all prior subtasks on the selected route. Finally, (C7a) and (C7b) capture the precedence constraints of the problem: (C7a) indicates that the second task i in the precedence constraint $(i', i) \in P$ is performed only if the first task i' is performed; (C7b) ensures that the start times of the task satisfy the precedence constraints. Similar constraints to (C7a) and (C7b) can represent synchronization constraints, by changing the inequalities to equalities, and removing the $\lambda_{i'}$ and $\tau_\infty(y_i - y_{i'})$ terms from C7b.

TABLE I

SUMMARY OF VARIABLES AND DEFINED QUANTITIES

Variable	Definition
x^k_r	Whether or not agent k performs route r
w^k_i	Waiting time of agent k for subtask i
t_i	Execution start time for subtask i
$d_{i'i}$	Delay for subtask i caused by i'

Quantity	Definition
v_j	Value of completing task j
R_k	Set of feasible routes for agent k
c^k_{1r}	Travel cost for route $r \in R_k$
c^k_{2r}	Wait cost per unit time for agent k
π^k_{jr}	Whether or not task j occurs on route $r \in R_k$
γ^k_{ilr}	Whether or not subtask i occurs at location l on route $r \in R_k$
$\delta^k_{i_1 i_2 r}$	Whether or not subtask i_1 occurs before subtask i_2 on route $r \in R_k$
τ^k_{ilr}	No-wait start time of subtask i at location l on route $r \in R_k$
τ_∞	End of planning horizon
W_i	maximum allowed waiting time for subtask i
$[\alpha_{il}, \beta_{il}]$	Valid time window within which to begin execution of subtask i at location l
λ^k_{il}	Service time for subtask i performed by agent k at location l
λ_i	Service time for subtask i in chosen solution
y_i	Whether or not subtask i is performed in chosen solution
P	Set of precedence constraints

In this model, the capacity and some time constraints are dealt with when generating feasible routes. The process of generating feasible routes also performs location choice by fixing the location of each subtask on the route. The solution of the set-partitioning problem then selects between all feasible routes for an agent, thus finalizing the location choice for each subtask, and also fixes the time for each task by inserting waiting times as needed to ensure that between-route timing constraints are satisfied while still respecting the within-route timing constraints.

V. BRANCH-AND-BOUND ALGORITHM

In problems that are small enough to exhaustively enumerate all feasible routes, the above mixed integer programming problem can be solved in a standard branch-and-bound framework. To start with, an upper bound on the solution is computed by relaxing the integrality constraints on the x^k_r variables, and solving the resulting linear program. Subsequently, branching decisions are made on fractional x^k_r variables (e.g., forcing them to either 0 or 1), and the solution process is repeated at each node of the branch-and-bound

tree, until a solution that satisfies the integer constraints is found. In reality, it is more efficient with this problem to make higher-level branching decisions rather than simply setting fractional x_r^k to 0 or 1. We adopt the following branching decisions, variations of which are used in several VRP solution approaches: When there are fractional routing variables, we branch by forcing two tasks to be on the same route ("together") in one branch or on different routes ("not together") in the other branch. When the fractional routing variables represent two different routes with the same subtasks performed in different orders, we branch by constraining two subtasks to occur in a specific order.

VI. Branch-and-Price Algorithm

In most situations, it will not be possible to enumerate all possible routes up front, and this is where a *column generation* process is useful. The algorithm starts out by considering only a subset of columns, and new columns are added as needed. The columns to be added are determined by solving, a problem called the *pricing subproblem*. A branch-and-price algorithm is a branch-and-bound algorithm in which column generation occurs at each node of the branch-and-bound tree.

Barnhart et al [5] provide a useful introduction to branch-and-price approaches and a detailed theoretical discussion, which is outside the scope of this paper. However, the essential idea is that the pricing subproblem finds "profitable" routes that can potentially improve the solution of the master problem. This is done by optimizing a pricing function computed from the dual variables of the master problem. For a minimization problem, a profitable column has a "price" of less than 0 (and as such, can decrease the objective function of the master problem if included), while for a maximation problem, a profitable column has a "price" of greater than 0 (and as such, can increase the objective function of the master problem if included). At each iteration of the column generation process, new columns are added to the master problem until the pricing problem, when solved to optimality, returns that there are no more profitable routes. At this point, branching can be performed on any fractional values, and the process repeated at subsequent branch-and-bound nodes.

If we designate the dual variables corresponding to constraints (C1) to (C7b) in our mathematical model as u^1 to u^{7b} respectively, then the pricing subproblem for our set-partitioning model can be derived as finding the feasible route r for agent k that maximizes the quantity:

$$\begin{aligned} p_r^k = &-(c_{1r}^k + u_k^1) + \sum_{j\in J}(v_j - u_j^2)\pi_{jr}^k \\ &+ \sum_{i\in I}\sum_{l\in L_i}(u_{ik}^3 W_i + u_i^{4a}\tau_{ilr}^k - u_i^{4b}\alpha_{il} + u_i^{4c}\beta_{il})\gamma_{ilr}^k \\ &+ \sum_{i'\in I}\sum_{i\in I}(-u_{i'i}^{5a} + u_{i'i}^{5b})W_i\delta_{i'i}^k \\ &- \sum_{(i',i)\in P}\sum_{l\in L_{i'}}(u_{(i',i)}^{7a} + u_{(i',i)}^{7b}(\lambda_{il}^k + \epsilon_{i'i}^P - \tau_\infty))\gamma_{i'lr}^k \\ &+ \sum_{(i',i)\in P}\sum_{l\in L_i}(u_{(i',i)}^{7a} - u_{(i',i)}^{7b}(\epsilon_{i'i}^P + \tau_\infty))\gamma_{ilr}^k \end{aligned} \tag{2}$$

To gain a better understanding of the pricing subproblem, recall that π_{jr}^k indicates whether a given task j is fully served on route r, γ_{ir}^{lk} indicates whether subtask i is performed at location l on route $r \in R_k$, and $\delta_{i'ir}^k$ indicates whether subtask i' occurs before subtask i on route $r \in R_k$. Also, note that for a given instance of the subproblem solution process, the dual variables u are constants. Thus, for a given agent, solving the pricing problem is equivalent to finding a feasible route that maximizes the above quantity. This can be done by searching through a graph where nodes represent feasible {location, subtask} pairs and edges indicate that it is possible for agent k to travel from one location to another. No edges connect pairs of nodes that correspond to the same subtask but different locations. The transition costs in the graph are determined by equation (2). The first term in the equation represents the cost of the route modified by a constant that depends on the agent. This route cost can be broken down into a transition cost for each edge of the graph that is traversed along the route. The second term in the equation represents a value or reward for each task completed on the route. Since a feasible route must comprise all subtasks of any task that is performed on the route, and since a feasible route consists of only one location for each subtask that is performed on the route, this term can be broken down to a value for each node of the graph visited along the route. The third term in the equation represents a cost for each node that is visited along the route: 3 of the sub-terms (involving the dual variables u_{ik}^3, u_i^{4b}, and u_i^{4c}) represent a constant cost for the node, and a fourth sub-term (involving the dual variable u_i^{4a}) represents a cost that is linear in the time that it takes to reach that node along the route, i.e., the no-wait arrival time, τ_{ilr}^k. The fourth term in the equation represents a cost for each ordered pair of tasks on the route, (i', i), such that i' precedes i. The fifth term represents a cost for each node along the route for which the corresponding subtask appears as the first subtask in a precedence constraint. The sixth and final term represents a cost for each node along the route for which the corresponding subtask appears as the second subtask in a precedent constraint.

VII. Generating New Routes: The Constrained Routing Planner (CRP)

We have formulated an optimal dynamic programming route-planning algorithm to solve the pricing problem described above. The algorithm finds the route that minimizes $\bar{p}_r^k = -p_r^k$, and hence maximizes p_r^k. Our route-planning algorithm is based on the DD* Lite algorithm [6] for incremental search with state dominance. DD* Lite performs a best-first search, focused by a heuristic, through a multi-dimensional state space, to find a path from a start node to a goal node, exploiting domain-specific dominance relationships to prune the state space where possible in order to make the search more efficient. While DD* Lite is a general search algorithm, it needs to be customized to the given domain, and we modify and customize it in the following ways:

1) State Space: Each node in the state space being searched is identified by the graph node n representing a given {subtask, location} pair, the no-wait arrival time t_a of the agent at the node along the route, the unordered set S_p of subtasks that have been previously completed along the route to that state, and a boolean variable b indicating whether the route satisfies the branching constraints of the current node in the branch-and-bound tree at which column generation is being performed. $state := \{n, t_a, S_p, b\}$

The node n is a node in the graph that is being searched, that is, the collection of {subtask, location} pairs in addition to two special nodes corresponding to the agent start and end locations. We designate the set of graph nodes as N where $|N| = \sum_{i \in I} |L_i| + 2$. While n is an *path-independent parameter* of the state, whose value does not depend on the path taken to reach the state, t_a, S_p and b are *path-dependent* parameters (as described in [7]) whose values depend on the path taken to reach the state and are computed dynamically during the search process. As such, states in this large multi-dimensional search space are not instantiated up front but are generated as they are encountered in the search.

2) Search Direction: The original version of DD* Lite searches backwards from the goal to the start in order to facilitate efficient replanning in domains where changes in the graph are likely to occur close to the start state. In this work, however, we flip the direction of search and instead search forwards from the start state to the goal node. This is because, as described in the previous section, the transition cost to a node in the graph depends on the arrival time at the node, and this can only be accurately computed if searching in a forward direction. We do not currently use the replanning functionality of DD* Lite.

3) Start and Goal States: The start state is defined as the special graph node corresponding to the agent start location for n, the earliest available time for the agent for t_a, an empty set for S_p, and a value for b that depends on the particular branch constraints in question. The goal state is defined as the special graph node corresponding to the agent end location for n, and a value of `true}`for b. The values of t_a and S_p for the goal state are not known ahead of time, but are computed during the search.

4) State Transitions: From a state s_1 in the graph, we can transition to a state s_2 corresponding to any other node $n \in N$ in the graph, with the following exceptions:

- No transitions are allowed into the start node
- No transitions are allowed out of the end node
- Transitions are not allowed from s_1 to a node corresponding to a subtask that has already been completed along this route, that is, a subtask in $s1.S_p$.
- Transitions are not allowed from s_1 to a node corresponding to a subtask whose previous subtasks have not yet been performed (for tasks with more than one subtask).

The arrival time of s_2 is computed using the arrival time of s_1, the service time at $s_1.n$, and the travel time from $s_1.n$ to $s_2.n$. Note that we are, in effect, computing the no-wait arrival time along the route. Waiting time is not included in the computation of t_a, since waiting time might be affected by the schedules of other agents and as such, is determined in the branch-and-price master problem. $s_2.S_p$ is computed as the union of $s_1.S_p$ with the subtask performed at $s_2.n$. Finally, $s_2.b$ is computed by examining whether the branching constraints are satisfied for this state. To facilitate this, we also keep track of a boolean variable for each branching constraint that needs to be satisfied, in order to be able to determine when they have all been satisfied.

The state transition function checks that agent capacity as well as maximum route length constraints are not violated. It also checks some time constraints, rejecting partial routes that arrive at a node after the relevant time window. Note that although the computation of t_a does not take waiting time into consideration, the algorithm does keep track for each state of the minimum waiting time that would be required to meet the time window constraints at the nodes along the route to that state. This enables tighter checks of time constraints and reduces the number of times the algorithm will find a route that is ultimately rejected in the branch-and-price master problem because of time constraints. The state transition function also checks that no branching constraints corresponding to the current branch of the branch-and-price tree are violated. For a "not together" constraint involving tasks j_i and j_2, it ensures that if j_1 is already on the current route, transitions to nodes corresponding to subtasks of j_2 are not allowed, and vice-versa. For an "order" constraint involving subtasks i_1 and i_2, it ensures that transitions to nodes corresponding to the later subtask are allowed only if the earlier subtask has been performed.

The transition cost from s_1 to s_2 is computed based on equation 2. Equation 2 represents the value for an entire route, but this needs to be incrementally computed as we transition from one state to another during the search. Note that negative transition costs are allowed in the graph, which is not problematic since cycling is disallowed.

5) Search Heuristic: Having computed the cost from the start state to state s, we need to compute a lower bound on the transition cost for the remainder of the route from s to the goal node. Whereas in the original version of DD* Lite a heuristic can be omitted (i.e. the heuristic value of 0 can be used), this is not possible here because transition costs can be negative. Without a heuristic that is a valid lower bound on the cost of the remainder of the route, the algorithm may terminate with a suboptimal solution. We compute a valid heuristic by computing bounds on the various dual variables, and using maximum route length constraints to bound the length of the remaining path.

6) Dominance Relationship: Assume that two states, s_1 and s_2 have the same node n, the same set of previous subtasks, S_p, but different arrival times, t_{a1} and t_{a2}. These states represent two different partial routes r_1 and r_2, reaching node n, having completed the subtasks in S_p in different orders or at different locations. Assume, without loss of generality, that $t_{a1} < t_{a2}$. Then the partial route r'_1 leaving from s_1 can visit at least every node that the partial route r'_2 leaving from s_2 can visit and will arrive at these nodes no later than

if it left from s_2. Furthermore, if the dual variable u_i^{4a} is non-positive for all subtasks, then, from the pricing equation (2), the cost of the partial route r_1' from s_1 to the goal, will be no worse than the cost of the partial route r_2' from s_2 to the goal. Thus, if the cost of the route r_1 from the start to s_1 is also better than the cost of the route r_2 from the start to s_2, s_1 *dominates* s_2, because there is no benefit in taking the route through s_2.

VIII. Results

We have implemented the branch-and-price algorithm as described. Below, we present the solution of an illustrative problem, highlighting the issues of precedence constraints, waiting costs, waiting time, and location choice. We then discuss the issue of how to handle dynamism, which is very relevant to intelligent transportation systems.

A. Example Problem

The example in figure 1 shows an evacuation problem where there are two transportation agents (squares), one medical agent (triangle), five individuals to be transported (crosses) and two shelters to which the victims can be transported (circles). Each of the five individuals requires a visit by the medical agent before being transported to a shelter. Thus, there are five precedence constraints in the problem. Figure 1 shows the routes computed for 2 different values of waiting cost, and Figure 2 show the corresponding schedules. In Figure 2, the horizontal axis represents time, and the three lines represent the schedules of the agents 0, 1 and 2. Circles indicate arrival at a location to perform a task, crosses indicate beginning execution of a task, dotted lines indicate waiting time, and solid lines indicate execution time. With a waiting cost of 0, the algorithm minimizes the travel time, regardless of how long the agent needs to wait/be idle before it can complete its task. Because there is only one medical agent, waiting times for some tasks are quite significant since the assigned transportation agent needs to wait for the medical agent to complete its task before it can transport the individual in question. When the waiting cost is increased to 0.5 (that is, half of the transportation cost per unit time), the transportation agents are less willing to wait and would prefer to travel in order to transport an individual that is ready. The overall distance traveled is increased, and the waiting time is reduced, though not completely.

For this example with 5 transportation tasks and 5 medical tasks to perform, the table below shows the effect of location choice on the problem size, and the potential advantage of column generation. As the number of subtasks in the problem that have a choice of locations at which they can be performed increases, the number of feasible routes increases drastically, but the number of routes generated in the column generation process stays roughly constant for this problem.

Location choices	0	1	2	3	4	5
Feasible routes	2261	3391	5073	7487	10813	15231
Generated routes	50	51	52	52	47	48

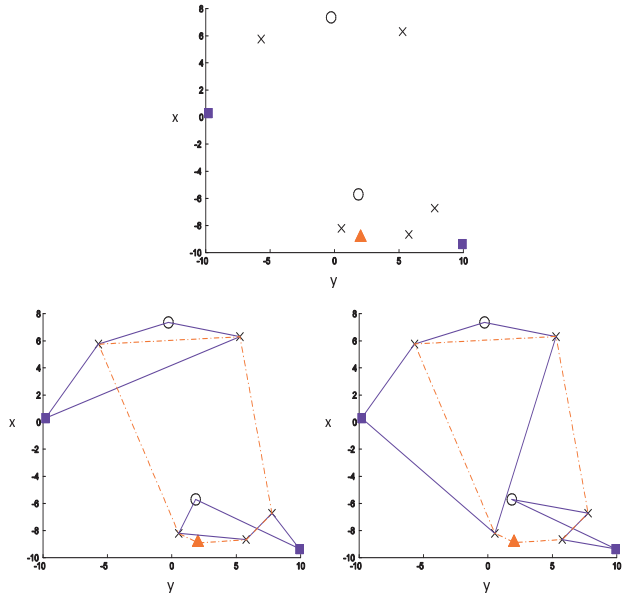

Fig. 1. Example evacuation problem and routes computed with waiting costs of 0 and 0.5

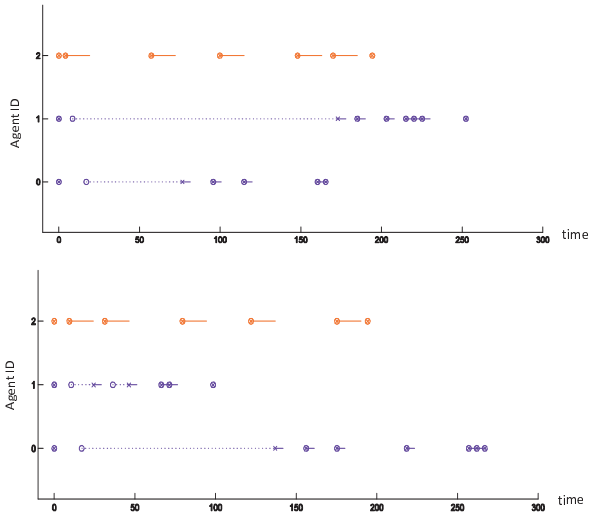

Fig. 2. Agents' schedules for waiting cost of 0 and 0.5

B. Handling dynamism

The presented approach has the advantage of computing an optimal solution to the task allocation and routing problem, but it requires that the system knows about all tasks ahead of time. By itself, it is not suited to dynamic domains in which new tasks arrive or conditions change over time. To handle dynamism, we propose a hybrid planning approach in which the optimal planner computes an initial optimal plan for the set of static tasks, that is, the tasks available at the beginning of the planning horizon. As new tasks arrive, they are incorporated into the schedule by using a decentralized market-based task allocation method [8]. The same method can be used to handle any modifications in the plan that are required as a result of unforeseen events (such as a road being closed, or a given vehicle becoming disabled).

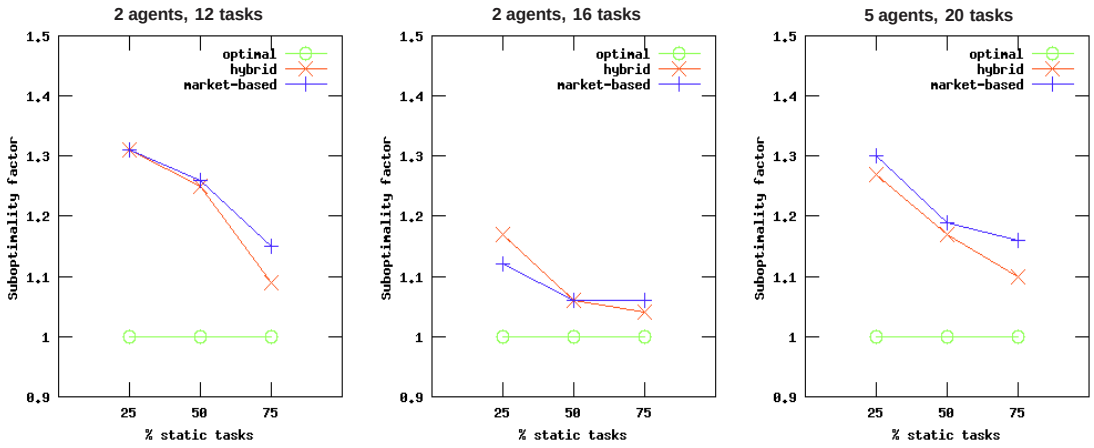

Fig. 3. Relative route costs for the optimal planner, the hybrid planner, and the market-based planner, as a function of the proportion of static tasks

Figure 3 compares the average solution cost (in this case, total team distance) for this hybrid approach to that for a solely market-based approach for three different problems configurations of a simulated scenario in which a team of vehicles needs to perform a collection of simple single-step tasks. For this scenario, there are no precedence constraints or location choices. The vehicles operate in a 40x40 area, traveling at a speed of 1 unit of distance per unit time. A set of "static" tasks is present from the begin of the planning horizon (t=0) while the remaining "dynamic" tasks come in at various times from t=1 to t=100. The optimal planner computes an initial plan for the static tasks, which the vehicles begin to execute. As they come in, new tasks are allocated and incorporated in the vehicle schedules with the market-based planner. The costs of the solutions computed by the optimal, hybrid and market-based allocation approaches (averaged over 5 random instances) are expressed as a ratio of the best solution computed by the optimal approach, in "hindsight", that is, after all dynamic tasks have arrived in the system. Note that this "hindsight" plan optimizes the team distance by inserting waiting time into the plan as appropriate, since it knows the exact times at which all tasks arrive in the system. As such, it is a plan that is unattainable without knowing the future. Note also that for this problem, a timeout is set on the solution process for the constrained route planning subproblem and so the best solution produced by the optimal planner may be slightly suboptimal in some cases. Figure 3 illustrates that having an optimized seed plan is usually, although not always, advantageous over using only a market-based method. As expected, the advantage of having a seed plan increases with the proportion of tasks that are static.

IX. Conclusion and Future Work

In this paper, we have presented a novel mathematical model for a complex task allocation and routing problem involving precedence constraints and location choice. We have presented a branch-and-price solution process and a novel dynamic programming algorithm for finding profitable routes. We also presented an approach by which this method can be combined with a market-based method to address dynamic scenarios. Our ongoing work includes a comprehensive analysis of the performance of the approach for different problem configurations. It also includes developing heuristic algorithms to solve the constrained route-planning subproblem, to widen the pool of problems for which this is a tractable planning approach. The planning framework presented contributes to enabling vehicles in an intelligent transportation system to achieve autonomy, not only on the level of navigation but also on the level of coordinating with other vehicles in a fleet to achieve high-level goals.

References

[1] J.-F. Cordeau, "A Branch-and-Cut Algorithm for the Dial-a-Ride Problem," *Operations Research*, vol. 54, no. 3, pp. 573–586, 2006.

[2] M. Savelsbergh and M. Sol, "Drive: Dynamic routing of independent vehicles," *Operations Research*, vol. 46, no. 4, pp. 474–490, 1998.

[3] D. Bredström and M. Rönnqvist, "Combined vehicle routing and scheduling with temporal precedence and synchronization constraints," *European Journal of Operations Research*, vol. 191, pp. 19–31, 2008.

[4] ——, "A branch and price algorithm for the combimed vehicle routing and scheduling problem with synchronization constraints," Norwegian School of Economics and Business Administration, Department of Finance and Management Science, Discussion Paper Number FOR7 2007, 2007.

[5] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Operations Research*, vol. 46, pp. 316–329, 1998.

[6] G. A. Mills-Tettey, A. T. Stentz, and M. B. Dias, "DD* lite: Efficient incremental search with state dominance," in *Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, July 2006, pp. 1032–1038.

[7] ——, "Continuous-field path planning with constrained path-dependent state variables," in *ICRA 2008 Workshop on Path Planning on Costmaps*, May 2008.

[8] M. B. Dias, "Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 2004.