Integrating Vehicle Routing and Motion Planning

Scott Kiesel and **Ethan Burns** and **Christopher Wilt** and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
`skiesel`, `eaburns`, `wilt`, `ruml` at `cs.unh.edu`

Abstract

There has been much interest recently in problems that combine high-level task planning with low-level motion planning. In this paper, we present a problem of this kind that arises in multi-vehicle mission planning. It tightly integrates task allocation and scheduling, who will do what when, with path planning, how each task will actually be performed. It extends classical vehicle routing in that the cost of executing a set of high-level tasks can vary significantly in time and cost according to the low-level paths selected. It extends classical motion planning in that each path must minimize cost while also respecting temporal constraints, including those imposed by the agent's other tasks and the tasks assigned to other agents. Furthermore, the problem is a subtask within an interactive system and therefore must operate within severe time constraints. We present an approach to the problem based on a combination of tabu search, linear programming, and heuristic search. We evaluate our planner on representative problem instances and find that its performance meets the demanding requirements of our application. These results demonstrate how integrating multiple diverse techniques can successfully solve challenging real-world planning problems that are beyond the reach of any single method.

Introduction

As techniques for high-level task planning and low-level motion planning each mature, there has been interest in how they might be integrated together to improve overall system performance. Often, decisions at the high level, such as who will do what and in what order, depend on low-level considerations, such as the existence or cost of feasible motions for particular tasks. A straightforward approach would be to combine both the task and motion planning problems and then solve them all at once with a single search algorithm such as A* (Hart, Nilsson, and Raphael 1968). Because of the exponential nature of such problems, however, this approach is intractable for even small instances. Alternatively, both the task and motion problems could each be solved independently by first finding a task-level plan and then solving the motion planning problem for each task. While this approach is usually feasible, it can lead to poor solutions, or even incompleteness. This is because the task planner has incomplete knowledge of the cost and dynamics that will be utilized by motion plans when achieving its tasks, and furthermore, the motion planner is focused on the individual tasks without considering a global perspective.

This paper makes two main contributions. First, we present a new problem that requires the combination of task and motion planning, called Waypoint Allocation and Motion Planning (WAMP). While the problem is easy to understand and compact to specify, it presents timely research challenges. It consists of scheduling a fixed set of vehicles to achieve different waypoint locations according to given temporal constraints. At the high-level, it is a resource allocation problem in which waypoints must be assigned to vehicles. For each vehicle, an ordering of the waypoints must be found such that temporal constraints can be met. At the low level, it is a difficult motion planning problem where a physically feasible path that respects the vehicles' motion models must be found such that each waypoint is visited and, again, all temporal constraints are met. The solution cost depends on the low-level paths that are selected. As we describe below, many of the subproblems of WAMP are known to be NP-hard. We also prove that the target value search problem (Kuhn et al. 2008), which is related to WAMP's routing subproblem, is NP-complete.

The second contribution is a planner that we have developed to solve WAMP instances involving fixed-wing aircraft. We combine tabu search for waypoint allocation, linear programming for scheduling, and heuristic search for route planning. The planner separates the high-level scheduling and resource allocation from the low-level routing by using a *surrogate objective* that is optimized by the high-level solver as a proxy for the true objective of the problem. This greatly reduces the number of times the router needs to be called. The low-level planner has the ability to give feedback to the high-level sequencer to help improve the accuracy of the surrogate objective. We present experiments that demonstrate the infeasibility of using one single A* search to solve this problem. Then, we test the scalability of our planner and evaluate the performance of its major components. We also show that our planner is able to solve realistic problems within the required time limit. This work illustrates how real world applications can feature the combination of multiple interacting planning problems, requiring the integration of diverse solution techniques.

Problem Formulation

WAMP is directly motivated by an application faced by our industrial partner. An instance of WAMP is given by a 6-tuple $\langle Size, V, W, R, C, K\rangle$ where $Size = \langle x_{max}, y_{max}\rangle$ is the problem's x and y dimensions, V is a set of vehicles, W is a set of waypoints, R is a set of relative temporal constraints between waypoints, C is a set of high cost regions and K is a set of non-traversable regions. As advised by our partner's domain expertise, the state space is restricted to two dimensions and vehicle collisions are not considered.

Vehicles In our instances, all vehicles are airplanes, so each element of the set V is a 5-tuple $\langle x_0, y_0, \theta_0, v, r\rangle$ where x_0, y_0 and θ_0 define the vehicle's initial position and heading, v is the vehicle's velocity and r is the minimum turn radius. In this paper only fixed velocity vehicles are considered. In Figure 1a, the vehicles start poses are depicted by small black triangles.

Waypoints Each waypoint in the set W is represented as a circle and is defined by the 8-tuple $\langle x, y, r, t_s, t_e, \theta_0, \theta_1, A\rangle$, where x, y and r give the center point and radius, t_s and t_e give the start and end times of the window during which the waypoint must be achieved, θ_0 and θ_1 define a range of headings that the vehicle must be within when the waypoint is achieved and $A \subseteq V$ is a set of vehicles that are not allowed to achieve that waypoint. The waypoints can be seen in Figure 1a as numbers with circles. Each waypoint must be achieved by being within the circle at a legal time at a legal heading.

Relative Temporal Constraints In addition to each waypoint having an absolute time window, the set R defines a set of relative constraints. Each is a 4-tuple $\langle u, v, min, max\rangle$ where $u, v \in W$ are waypoints and min and max are the minimum and maximum allowable time difference between when u is achieved and v is achieved.

Costs C is a set of two dimensional Gaussians: $\langle x, y, h, \sigma_x, \sigma_y, c\rangle$, where x and y give the center location, h specifies the 'height', or the cost that will be incurred at the center, the σ terms give the standard deviation in the x and y directions respectively, and c is the correlation. These are used to determine the cost of vehicle motion. There is also a minimum cost present everywhere representing fuel consumption and time. Every vehicle traverses a path, and the cost incurred by the vehicle is the time it spends in each location multiplied by the cost of being in that location. As the time discretization approaches zero, the limit is the line integral along the vehicle's path divided by the vehicle's speed. In Figure 1a, the cost of each cell is represented by the shade of red in the cell. Being in a white area incurs low cost, and being in a red area incurs high cost.

Keepouts Lastly, K is a set of 'keepout' zones that cannot be traversed. Each zone is a triangular area defined by three points $\langle x_0, y_0, x_1, y_1, x_2, y_2\rangle$. These shapes can be composed to create more elaborate regions. In Figure 1a, the gray area is a keepout zone.

The cost of a solution is the sum of the cost incurred by all vehicles. However, after a vehicle achieves its final waypoint it no longer accumulates cost. The objective of WAMP is to find a minimal cost solution, using the available vehicles, that achieves the given waypoints and meets all of the problem constraints.

Application Context

The planner must solve problems within ten seconds because it is part of an interactive decision-support aid with a human in the loop, who edits the resulting plans. The planner's solution might not be immediately acceptable because the Gaussian cost model is only an approximation of the real cost model and there may be other assets that are not modeled in the instance. Our partner also was interested in pseudo-balanced vehicle makespans. Therefore our planner's objective was altered to take into account not only cost but an adjustable ratio between path cost and makespan.

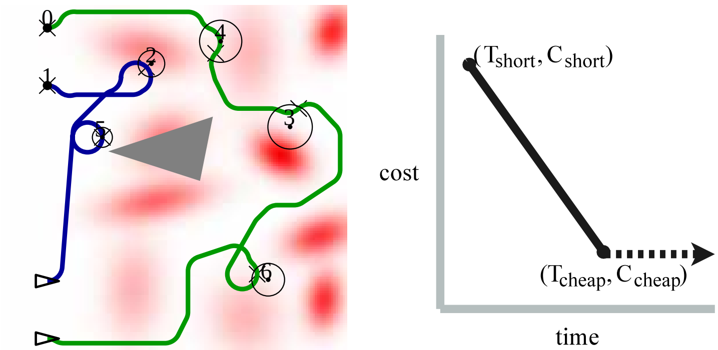
Relations to Other Problems

We provide brief sketches showing how WAMP can be seen as a composition of several problems that are known to be NP-hard. We also provide an NP-completeness proof for one subproblem that, as far as we are aware, was not previously known to be NP-complete.

Vehicle Routing Problem with Time Windows VRPTW is a popular problem in the operations research community. While it now has a large number of variants, the classic VRPTW (Dantzig and Ramser 1951) is: given an infinite fleet of vehicles and a set of service requests with known fixed distances between request locations, find a schedule such that the number of vehicles and total travel cost are minimized (in that order) and all requests are serviced within their given time windows. One variant that is closely related to our problem is the m-VRPTW problem (Lau, Sim, and Teo 2003) where the number of vehicles is fixed at some value m and the goal is to find the minimum cost schedule with this fixed size fleet. The decision variant of this problem (determining if a feasible schedule exists) has been shown to be NP-complete (Savelsbergh 1985; Lau, Sim, and Teo 2003).

The m-VRPTW problem can be reduced to a WAMP instance if each of the m vehicles has infinite capacity and the delivery destinations reside in the Euclidean plane. The reduction can then be achieved by setting the turning radius of each of the m vehicles to ϵ with a fixed velocity of 1. All vehicles share a start and end location, the location of the depot. Each delivery destination location and time window are direct mappings from the original problem. Using a large Gaussian to distribute cost uniformly over the map, such that the cost of each point on the map evaluates to 1, will result in a WAMP objective function that directly minimizes the overall distance traveled.

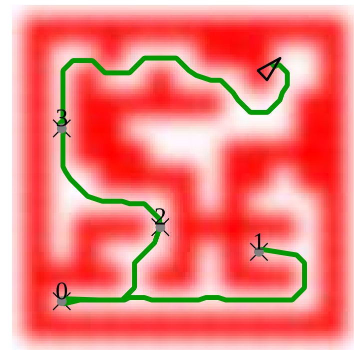
Jobshop Scheduling Problem JSP is perhaps the most well-known scheduling problem. The JSP is an NP-complete problem (Garey and Johnson 1991) concerning a given set of jobs, each composed of a set of activities that each have a given length. All activities must be assigned time on a given set of machines so that no two activities use the same machine at the same time and each activity must be serviced by a specified machine. The problem is to determine whether or not a feasible schedule exists within a given deadline.



(a) (b)

n	Failure Rate
1	24%
2	64%
3	88%
4	98%
5	98%
6	100%

(c)



(d)

Figure 1: Example solution with 2 vehicles and 6 waypoints (a), example time/cost trade-off (b), A* results (c), a maze (d).

We reduce the JSP to WAMP by associating vehicles with machines and waypoints with tasks. For each machine m, there is a special location l_m located sufficiently far from the special locations for all other vehicles that flying between any two special locations takes longer than the deadline. Each activity to be scheduled on machine m corresponds to three unique waypoints, the first one is placed at location l_m, the second is placed at a distance from l_m that corresponds to half of the length of the activity and the final one is placed at l_m. These waypoints have temporal constraints such that the first must be achieved first, the second one, that is not located at l_m, must be achieved exactly half the activity duration after the first and the final one must be achieved exactly half of the activity duration after the second. When a vehicle chooses to achieve the first waypoint for this activity, it cannot achieve any other waypoints besides the remaining two for this activity and the entire time to achieve all three must be equal to the activity duration. Finally, the activities are ordered within their respective tasks by constraining the last waypoint for an activity to proceed the first waypoint for the activity that follows it within the task.

Traveling Salesman Problem TSP is a classic NP-complete problem (Garey and Johnson 1991). The Euclidean variant of the TSP may be reduced to WAMP by creating an instance with uniform cost, a single vehicle, a turn radius that is infinitely small (the vehicle can turn and point itself directly at its next waypoint) and by placing the cities of the TSP at their respective x and y locations. The vehicle is able to traverse this set of waypoints within the given cost bound if and only if there is a solution to the TSP within the bound.

Target Value Search While WAMP is defined on a continuous space, our solution uses discrete search-based methods, thus it is useful to understand the complexity of related discrete problems such as this. TVSP Kuhn et al.; Schmidt et al. 2008; 2009 is the problem of finding a path from start to goal whose length is equal to the target value. As far as we are aware, there are no theoretical results about the complexity of this problem. Schmidt et al. (2009) conjecture that the optimization variant of the TVSP (i.e., finding a path with cost as close as possible to the target value) is in EXPTIME. We will show that the decision problem of determining whether or not a path with the exact target value exists is, in fact, NP-complete.

We specify a TVSP instance as a 4-tuple $\langle G, s, g, T\rangle$, where $G = \langle V, A\rangle$ is a finite graph with vertices V and a set of weighted, directed arcs $A \subseteq V \times V \times \mathbb{N}$, $s \in V$ and $g \in V$ are the start and goal nodes in the graph, and $T \in \mathbb{N}$ is the target value.

Theorem 1 *The target value search problem in a graph is NP-complete.*

Proof The problem is in NP since, given a solution, one can easily check the validity of the path and sum the edge weights in polynomial time. We show it is NP-hard by reducing from SUBSETSUM (Garey and Johnson 1991). Given an instance of SUBSETSUM — a finite set $S \subseteq \mathbb{N}$ and a positive integer B — we formulate a target value search problem as follows: $T = B$ is our target value. For each $s_i \in S$, we create a vertex v_i. The vertices are then linked together in a chain with two arcs between adjacent pairs of vertices, one arc has cost 0 and the other has a cost equal to the element of S corresponding to the first vertex of the arc: $(v_i, v_{i+1}, 0) \in A$ and $(v_i, v_{i+1}, s_i) \in A, 0 \leq i < |S|$. Finally, our start vertex is $s = s_0$ and our goal vertex is $g = s_{|A|-1}$.

There is a path in this graph that achieves the target value T if and only if there is a subset of S whose sum is equal to B, with the non-zero-cost arcs corresponding to the elements included in the subset. □

Our Approach

Our approach is guided by four features of the application context that we exploit to make the problem easier to solve: first, the cost function is relatively smooth, meaning that similar paths will often have similar costs. This allows us to approximate final path cost by evaluating the cost of a simple 8-way grid path. This implies that we can postpone detailed motion planning until we have a promising candidate solution. Second, there are many possible low-level paths, so we can make the assumption that any schedule will be routable given sufficient time per leg. This allows us to assume that a spectrum of paths exists between the fastest

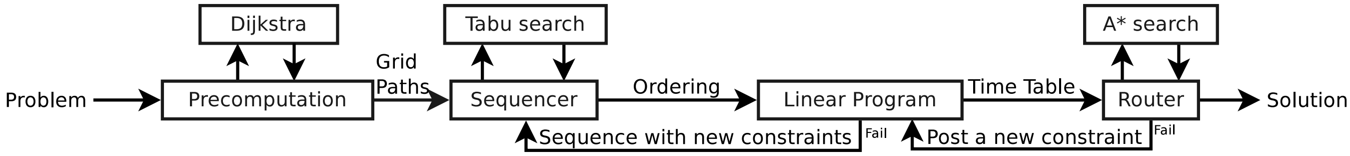


Figure 2: Overview of our system.

(most expensive) and the cheapest (relatively long) (see Figure 1b). As we explain below, this spectrum is constructed optimistically and we therefore will incorporate feedback from motion planning as necessary to refine the estimates of achievable paths. Third, making a leg longer can usually decrease cost of the final route, as the vehicle has more time to navigate around high cost regions. Finally, it is easy to make a leg of a route longer, because if the route arrives at the destination waypoint too early, then extra time can easily be added by inserting loops into the route at low-cost locations. This means we can focus on trying to arrive early.

More specifically, our planner uses four stages: precomputation, scheduling, building a timetable and routing (see Figure 2). First, we pre-compute information about times and costs between pairs of waypoints. This information will be used by the later stages to approximate the time and cost between pairs of waypoints. Next, the sequencer assigns waypoints to vehicles and then orders them such that there should be a feasible route for each vehicle that obeys the problem constraints. After an assignment and ordering are found, we use a linear program (LP) to find a *timetable* that specifies, for each waypoint, the time at which its assigned vehicle should arrive. The timetable is then passed to the router to find a flyable path for each vehicle that achieves the given times.

The information about routability used by the sequencer and LP is approximate, so there are a two places where the procedure may fail. When this happens, the router posts additional problem constraints, which are then used by the LP and sequencer to improve the accuracy of their estimates. The following subsections describe each of these steps in greater detail.

Pre-computation

Both the sequencer and the timetable generation phases need to estimate the cost and duration of possible routes between each pair of waypoints, as depicted in Figure 1b. These estimates are represented by a linear interpolation between the quickest path and the cheapest path between each set of waypoints. The slope of this line represents an estimate of the rate at which adding additional time navigating on a leg can be converted into cost reduction, which we call the *improvement slope*. These shortest and the cheapest paths between the waypoints are computed in an 8-connected grid discretization of the problem where the discretization is the size of the smallest vehicle's turning radius. Each grid cell uses a single traversal cost estimate given by the mean of the true cost sampled at a fixed number of points distributed uniformly over the cell. In our implementation, the cost between two adjacent cell centers is the distance multiplied by half of the cost of each cell. To find the shortest and cheapest paths to each waypoint, we use Dijkstra's algorithm from each waypoint to all cells in the grid. Since the costs and distances are invertible, this gives an estimation of the shortest and cheapest paths to the given waypoint from anywhere in the problem. While the sequencer only needs waypoint to waypoint estimates, the router needs a heuristic value for every cell.

Sequencer

The sequencer finds an ordered assignment of waypoints to vehicles that is thought to be feasible given the problem constraints. For this step, we use a tabu search based on the technique for m-VRPTW described by Lau, Sim, and Teo. WAMP however, has a handful of additional constraints such as allowable vehicle constraints and relative temporal constraints. The search is over ordered partial assignments of waypoints to vehicles. In each state, there is a set of waypoints that have yet to be assigned called the *holding list* and there is a set of ordered waypoints assigned to each vehicle. The neighborhood of a state is given by five operators: *relocate* a waypoint by moving it from one vehicle to a specific location in the ordering for another vehicle, *exchange* two waypoints in the ordering on a single vehicle, *unassign* a waypoint by moving it from a vehicle's ordering to the holding list, *assign* a waypoint to a specific location in the ordering for a vehicle and *exchange* a waypoint on the holding list with an assigned waypoint on a vehicle.

The search begins from the initial state where all waypoints are unassigned. The neighborhood of the initial state is evaluated to find the best neighbor using an ordering predicate described below. As neighbors are generated, they are tested for validity in two ways. First, constraints imposed by the ordering of each schedule are tested for feasibility using a simple temporal network (STN, Cervoni, Cesta, and Oddi 1994). In order to account for the distance between waypoints, we use the pre-computed shortest path distances to constrain each pair of waypoints to be separated by at least the time required to traverse the shortest path between them. If the STN reports that the ordering constraints are inconsistent with the constraints of the problem, then the neighbor is discarded as it cannot lead to a valid solution.

The second test is to see if the neighbor is *tabu* by checking if any of the waypoints that moved while generating the neighbor are included in a *tabu list*. The tabu list contains waypoints that are temporarily disallowed from being moved. If a neighbor fails the tabu test, then it is considered as a candidate for the best neighbor, only if there are no other feasible neighbors. The tabu list helps to prevent the search from getting stuck in local minima by causing it to explore new portions of the space.

Once the best feasible neighbor is found, then the waypoints that were moved to generate that neighbor are added to the tabu list. If the size of the tabu list becomes greater than a fixed size (7 in our experiments), then entries are removed in first-in-first-out order. Finally, the search iterates with the best neighbor as the new current state. The best state ever encountered by the search is maintained as an incumbent, giving the sequencer an anytime behavior. The sequencer is stopped when either a maximum time limit has been reached or, if a full schedule has been found, it is stopped when no new incumbent arrives for a quarter of a second.

Following Lau, Sim, and Teo (2003), the ordering function used by the sequencer to estimate the quality of a state is hierarchical. First, the ordering function prefers states in which more waypoints have been scheduled. This helps encourage the sequencer to find total assignments of all of the waypoints to vehicles. In order to allow the user to make a trade-off between inexpensive and short schedules, we break ties using our version of the WAMP objective.

These costs approximate the actual makespan and cost of the final flyable route for the given schedule. Since the sequencer finds an ordering over the waypoints and not a fully instantiated timetable, there is some question as to how time may be allocated among the different legs of each route if there is flexibility in the temporal constraints. For estimating the cost of a state during the tabu search, we can use one of three techniques. The first approximation is optimistic and assumes that each leg will always use a path with the cost and duration of the minimum cost grid path. We call this the *min* estimation technique.

The *greedy* technique assigns time to each leg greedily. Each leg has an associated improvement slope which we use to estimate the rate at which we can convert extra time into cost reduction. The greedy technique greedily allocates more time to legs for which additional navigation time is likely to reduce cost the most. As we describe below, this greedy strategy is optimal in certain situations.

The final estimation technique is based on linear programming and is fairly expensive when evaluated on each state generated by the tabu search. We describe it in the next section as it is the same technique used to generate the timetable of the final solution returned from the sequencer.

Generating a Timetable

Once a waypoint ordering has been found for each vehicle, we generate a timetable that assigns the time when each waypoint should be achieved. This timetable will be used by the router to find a flyable path for each vehicle that achieves each waypoint at its designated time. In order to decide where time should be allocated along each vehicle's route, we again use an estimation of the time/cost trade-off for each leg of the route. The objective of this part of the solver is to assign each leg a time such that the sum of the associated costs is minimized.

In order to meet the problem's time constraints, more time may need to be spent on a leg than would be taken by the cheapest path. If this happens, the cost of the leg will generally be greater than the cheapest path cost due to cost incurred while waiting for time to pass. Currently, our implementation uses an optimistic approximation in which additional time can be added for free.

The LP uses two base variables for each leg: reduction duration $dur_{red(i)}$ and free duration $dur_{free(i)}$. $dur_{red(i)}$ represents the additional time that is devoted to avoiding high cost areas, and is required to be larger than the minimum travel time between the two waypoints, and smaller than the travel time of the cheapest path between the two waypoints. $dur_{free(i)}$ represents time beyond the time required for the cheapest path. $dur_{red(i)} + dur_{free(i)} = dur_i$, where dur_i is the duration spent getting to waypoint i from the previous place, either the previous waypoint or the starting location. t_i is the time at which waypoint i was achieved, and it is equal to the sum dur_j for all waypoints that the vehicle services up to and including waypoint i. The objective function of the LP is minimizing $\sum_{waypoints} dur_{red(i)} \cdot red_i + 0 \cdot dur_{free(i)}$ where red_i is the improvement slope. Temporal restrictions from the problem all restrict t_i so these can be entered into the LP directly, restricting the legal values of the derived variables.

An alternative method for solving the timetable problem is to use the greedy estimation method used by the scheduler.

Theorem 2 *In the case where there are no relative constraints in the problem or when all relative constraints are subsumed by the absolute constraints on each waypoint, the solution produced by the greedy algorithm is optimal.*

Proof Suppose we have a potentially optimal solution that is not the greedy solution. The fact that this solution is not greedy means there exists a pair of legs S and S' such that S' offers a worse return on investment of time, and S' was allocated time that could possibly have gone to S. This possibility implies that it is possible to shift time from S' to S by simply moving all the waypoints between S and S' by some nonzero amount, leaving the duration of all other legs the same. This solution cannot be optimal, because we can improve it by moving some time from S' to S. This reduces the cost of the solution because S' offers a worse return on investment of time than S, and all other legs remained the same duration. □

Theorem 3 *In the general case, the problem requires a non-greedy method, such as linear programming.*

Proof We exhibit an instance with three vehicles (and some relative constraints) that defies greedy scheduling. Vehicle v_1 must visit waypoint w_1, which is at least 2 minutes away. Vehicles v_2 and v_3 each start one minute from w_2 and w_3, respectively, and must visit them exactly 1 minute before v_1 visits w_1. Anytime after v_1 visits w_1, v_2 must visit $w_{2'}$ and v_3 must visit $w_{3'}$. $w_{2'}$ is at least 1 minute from w_2 and $w_{3'}$ is at least 1 minute from w_3. All waypoints must be visited before time 7. The traversal costs are such that giving v_1 more time for w_1 lowers cost by 6 per minute, giving v_2 or v_3 more time for w_2 or w_3 doesn't lower cost at all, and giving v_2 or v_3 more time for $w_{2'}$ or $w_{3'}$ lowers cost by 5 per minute. The greedy scheduler will put w_1 at $7 - \epsilon$, w_2 and w_3 at $6 - \epsilon$, and $w_{2'}$ and $w_{3'}$ at 7. This lowers cost for v_1 by $(5 - \epsilon) \cdot -6$ and for v_2 and v_3 by $\epsilon \cdot -5$, for a total of $-30 - 4\epsilon$. The optimal solution puts w_1 at 2, w_2 and w_3 at 1, and $w_{2'}$ and $w_{3'}$ at 7, which lowers cost for v_1 by 0 and for v_2 and v_3 by $5 \cdot -5$, for a total of -50. □

Routing

The router constructs flyable paths that meet the timetable while minimizing cost. The router performs this task one vehicle at a time, one leg at a time. Each invocation has three phases: finding a grid path, smoothing the grid path, and adding additional travel if necessary to match the timetable.

Finding a Path The first step in constructing each leg is to use a discretized version of the problem to find an 8-way grid path that connects the cells containing the leg's start and goal. All grid cells whose center point is within the radius of the leg's goal waypoint count as goals. If the waypoint's radius does not contain a grid cell center, the grid cell that contains the waypoint's center point is used as the goal. Any cell touched by a keepout zone is marked as impassable in the grid search. Technically this approximation makes the planner incomplete, however, this was not an issue in practice.

Grid paths are found using A* search, modified to account for time constraints. The modified A* search prunes any state whose travel time so far t_{cur} and estimated remaining travel time t_{rem} (from the pre-computed shortest 8-way grid path times) are greater than the deadline d_i imposed by the timetable, $t_{cur} + t_{rem} > d_i$. The cost of each grid cell is determined in the pre-computation phase. The heuristic used during search is based on the pre-computed costs. The pre-computed cheapest path is used if its length is less than that required to meet the deadline.

Smoothing If used directly, the 8-way grid path is usually dynamically infeasible and might not intersect the waypoint's radius or take into account heading constraints from the waypoint or the previous leg (or start position).

The grid path is smoothed by substituting arcs at sharp turns, resulting in a smooth path that is traversable by the current vehicle. This smooth path may not achieve the waypoint correctly (incorrect heading and/or incorrect position) and may not line up correctly with the exit trajectory from the previous leg. This is resolved by constructing dynamically feasible Dubins paths (Dubins 1957; LaValle 2006) to match up the ends of the smooth path with the previous leg and the goal waypoint. This is done by constructing a Dubins path that connects a point on the smooth path to either the goal waypoint or the previous leg.

The connection point choice has very visible impact on the resulting path. Choosing a point too close may result in large turns to correct heading discrepancies. Choosing a point too far away can remove too much of the cheaply routed path. We iteratively try several lengths, keeping the best path according to a weighted combination of cost and distance.

Constructing the connection from the smooth path to the goal waypoint has one more free variable, the heading at the waypoint. The waypoint may have an associated heading constraint so any values chosen must be within the specified range. The same iterative technique is used to evaluate connection points along the smooth path.

The heading at which a non-goal waypoint is achieved affects both the cost of the segment entering the waypoint as well as the cost of the segment exiting the waypoint. We would like to achieve the waypoint at a heading which is expected to have a cheap ingress as well as a cheap egress. To account for both of these costs, we consider a small set of pairs of Dubins curves where one curve in each pair is entering the waypoint and the other is exiting. The set is constructed using all combinations of a discrete set of starting points along the smoothed path entering the waypoint, ending points along the grid path exiting the waypoint, and headings at the waypoint. Of this set, we choose the curve that enters the waypoint from the pair that minimizes the weighted sum of cost and makespan.

Extending a Route The smoothing process can result in paths longer or shorter than the grid path solution. When a path whose increased length results in missing the deadline, the A* search is continued to find a faster path. If no such path can be found, the router will fail back to the timetable stage with a new constraint bounding the problematic leg.

If smoothing results in a path that arrives at the waypoint before the deadline, the path is lengthened. If the time required to arrive at the deadline is at least the circumference of a tight loop of the vehicle, loops are added to the leg in the area where they will increase the cost least.

When Routing Fails

The timetable is generated using only an approximation of the routability between waypoints, so it may happen that the router is unable to meet the given deadlines. This can occur when the shortest 8-way grid path between two waypoints is shorter than the shortest flyable path. When the router fails to successfully route a leg, it passes the true minimum distance and cost of the failed leg back to the LP and sequencer. Using this new distance constraint, a new timetable is found and routing restarts. Additionally, if the updated LP has become infeasible, then the ordering produced by the sequencer is invalid and the sequencer is restarted to find a different ordering.

To avoid re-planning the same legs again in an updated timetable, the router caches the route for each successful leg. When a new timetable requires a leg that has already been routed with the same time constraint, this leg is re-used from the cache.

Evaluation

We now present the results from experiments we performed to evaluate our planner.

A Single Unified A* Search

Our first experiment verified that solving WAMP by running an A* search on the combined task and motion planning problem would quickly become infeasible. The state space included the airplane's position, heading, and time. The available operators were turn left or right $45°$ and go straight. For a heuristic, we calculated a minimum spanning tree of the 8-way grid path costs between waypoints on a discretized version of the problem, and added the distance of the vehicle to the nearest waypoint. The A* solver was written in Java, and we define failure as filling a 7GB object heap. Figure 1c shows the failure percentages (right column) as the number of waypoints scales from 1 to 6 (left column). Each of these problems used a single vehicle and had no temporal constraints. The A* solver fared extremely poorly on this problem, and was unable to successfully solve a full set of these very small instances even with a single waypoint.

Scaling Behavior

We now turn to evaluating the approach discussed in this paper, which was implemented in C++ and run on a 3.16 GHz machine with 8GB of RAM. Our first evaluation measured solution time and cost when scaling both the number of vehicles and the number of waypoints. Both of these parameters have a large effect on the difficulty of problem. The plots in Figure 3 show the results of these experiments. The left-most plot shows the scaling behavior of the min, greedy and LP surrogates as a function of the number of waypoints. Each glyph represents the mean time and cost over a set of instances with a number of waypoints given by the label (10, 20, 30 or 40) and 4 vehicles. The error bars give the 95% confidence interval on the means. A line connects each mean in order of increasing number of waypoints. As can be seen, the problems require more time and accrue more cost as the number of waypoints increases. When using both the min and greedy surrogates, we are able to solve the instances within our 10 second time frame even with up to 40 waypoints. We were surprised by the good performance of the min approximation. The LP approximation requires more time and is only able to solve up to 30 waypoint instances within the 10 second time frame.

The center plot of Figure 3 shows the scaling behavior as the number of vehicles increases. These instances had 20 waypoints. As the number of vehicles increases, the planning time increases. Again, the min and greedy surrogates give the best performance. Both the greedy and min techniques easily solve all problems within the 10 second time frame. The LP technique requires more than 10 seconds for some 16 vehicle instances.

Evaluating the Scheduler

Next, we considered synthetic instances that stress each major component of the system separately. To evaluate the sequencer, we created a set of instances for which we could find optimal solutions to the scheduling problem. We converted sets of TSP instances with 40 and 100 cities into WAMP instances for a vehicle with turning radius ϵ in order to compare the solutions found by our sequencer to the optimal TSP solutions. For comparison, we also implemented a simple nearest-neighbor TSP solver which chooses to visit the nearest unvisited city next.

The results of this experiment are shown on the right plot in Figure 3. We compared the min (M) and greedy (G) approximation techniques and the nearest neighbor TSP solver (NN). The y axis shows the factor of the optimal cost, so 1 is optimal and 1.2, for example, is 20% over optimal. Each box surrounds the middle half of the results, the horizontal line represents the median value, the 'whiskers' extend to the min and max. Circles beyond the whiskers show outliers. This plot does not include any results for the LP-based approximation as it was unable to solve any of the 100 city problems within a 120 second time limit. We can see from this plot that our ordering search tends to find solutions that are 20% above the optimal cost. For the more difficult 100 city instances, both the min and greedy approximations tend to outperform the nearest neighbor solver. Additionally, the 100 city instances seem to skew a bit more toward low-cost solutions than the easier 40 waypoint instances. We interpret these as positive results because they show that our sequencer is able to find reasonable solutions to these TSP instances.

Evaluating the Router

To evaluate the router, we created instances that required traversal of a maze of high-cost regions. Figure 1d shows the path found for one such instance. While we did not have any simple way to quantify these results, it is visibly clear that the router was able to find its way through the mazes while avoiding high-cost regions.

Application

Finally, we evaluated on a set of instances that were similar to those used by our industrial partner. These instances were 200x200 miles, with 3 vehicles, and 41 waypoints. Our industrial partner's current system, which we do not have access to for reasons of intellectual property and security classification, solves instances like these in approximately 7 seconds. On this set of instances, our solver had a mean solution time of 2.5 seconds. We have designed our implementation such that we expect near linear time speedup on a multi-core machine; so these results could be improved even further. Due to confidentiality reasons we were unable to directly compare solutions on quality, however we generally received positive feedback.

Related Work

Rapidly-exploring Random Trees (RRTs, Lavalle 1998) are a popular technique for finding dynamically feasible motion plans, however they do not minimize path cost. The RRT* algorithm (Karaman and Frazzoli 2010) minimizes cost, but does not handle constraints.

Bhatia, Kavraki, and Vardi (2010) combine sample-based motion planning with temporal goals by employing a geometry based multi-layered synergistic approach. Unlike the temporal constraints of WAMP, their goals are given by linear temporal logic formula.

Dornhege et al. (2009) describe how to combine low-level motion planning with high-level task planning via semantic attachment to a PDDL planner. In their approach, the lower

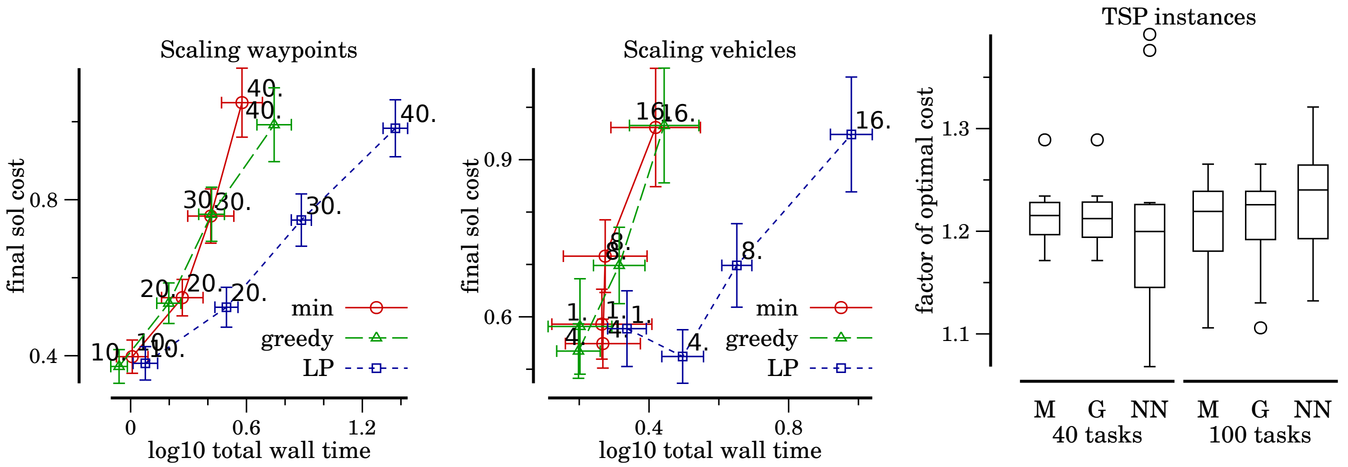


Figure 3: Scaling the number of waypoints and vehicles (left and center) and TSP instances (right).

level planner is used to check action applicability and compute effects whenever certain high level actions are used. In our approach, we use pre-computed minimum travel times to allow quick feasibility checking during high level planning, reserving the low level planner for computing the true cost of a solution.

Kaelbling and Loranzo-Pérez (2011) present a more flexible technique for combining both task and motion planning called "hierarchical planning in the now." The technique generates a hierarchy dynamically. When refining a transition at one level in the hierarchy, a planner is used where the goal specification is given by the preconditions of the destination node of the transition. This technique does not handle temporal constraints or a cost metric other than makespan.

Frank et al. (2011) make use of surrogate objective for motion planning for a robotic arm in the face of deformable objects. Their technique uses a surrogate objective to avoid using a computationally intensive finite element methods simulation to compute the cost of object deformations.

There has also been previous work in routing for aerial vehicles. McVey et al. (1999) present the Worldwide Aeronautical Route Planner (WARP) that plans fuel-efficient airplane routes around the globe. Štěpán Kopřiva et al. (2010) present Iterative Accelerated A* (IAA*) which is a technique developed for flight-path planning that increases the distance covered by each action primitive when the vehicle is far from surrounding obstacles. However, neither of these techniques consider temporal constraints.

Possible Extensions

Our current surrogate objective optimistically assumes that additional time can be added to a route free of charge. We plan to explore better approximations for this issue in the future. One possible improvement is to estimate that additional time adds cost at a fixed rate. As long as the slope is greater than the original it can still be captured in a linear program.

An additional improvement to our current system would be to allow the router to pass more information back to the sequencer and linear programming layers. Currently, the router only sends accurate time/cost information back to these layers when it determines that a leg is unroutable. One may imagine a more complex system, however, where information flows back to the sequencer and linear programming layer for every successfully routed leg too.

Conclusion

We introduced the problem of Waypoint Allocation and Motion Planning, which requires integration of high-level task planning with low-level motion planning. WAMP models a real-world application, moving beyond the classic planning problems and raising interesting issues that have not received much attention, including how to partition effort in a multi-level planner and how to trade plan cost for execution time in a time-constrained context. WAMP contains many subproblems that are well-known to be NP-complete and we proved that the target value search problem is NP-complete. We described an approach for the WAMP problem that takes advantage of the characteristics of the problem in order to separate the solving into three distinct stages: scheduling, building timetables, and routing. Our approach makes use of a surrogate objective in the high-level layers in order to avoid calls to the more expensive low-level route planner. Using this hybrid approach we are able to meet the demanding requirements imposed by the application.

Acknowledgements

We graciously acknowledge funding from the DARPA CSSG program (grant HR0011-09-1-0021) and NSF (grant IIS-0812141).

References

Bhatia, A.; Kavraki, L. E.; and Vardi, M. Y. 2010. Sampling-based motion planning with temporal goals. 2689–2696. Anchorage, Alaska: IEEE.

Cervoni, R.; Cesta, A.; and Oddi, A. 1994. Managing dynamic temporal constraint networks. In *Proceedings of AIPS-94*, 13–18.

Dantzig, G., and Ramser, J. 1951. The truck dispatching problem. *Management Science* 6(1):80–91.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebe, B. 2009. Integrating symbolic and geometric planning for mobile manipulation. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS-09)*.

Dubins, L. E. 1957. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics* 79:497–516.

Frank, B.; Stachniss, C.; Abdo, N.; and Burgard, W. 2011. Using gaussian process regression for efficient motion planning in environments with deformable objects. In *Automated Action Planning for Autonomous Moblie Robotics*.

Garey, M. R., and Johnson, D. S. 1991. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Company.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2):100–107.

Kaelbling, L. P., and Loranzo-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *IEEE Conference on Robotics and Automation*.

Karaman, S., and Frazzoli, E. 2010. Incremental sampling-based algorithms for optimal motion planning. *CoRR* abs/1005.0416.

Kuhn, L.; Schmidt, T.; Price, B.; Zhou, R.; and Do, M. 2008. Heuristic search for target-value path problem. In *First International Symposium on Search Techniques in Artificial Intelligence and Robotics*.

Lau, H. C.; Sim, M.; and Teo, K. M. 2003. Vehicle routing problem with time windows and a limited number of vehicles. *Europen Journal of Operational Research* 148:559–569.

Lavalle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. Technical report, Department of Computer Science, Iowa State University.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press. Available at http://planning.cs.uiuc.edu/.

McVey, C. B.; Clements, D. P.; Massey, B. C.; and Parkes, A. J. 1999. Worldwide aeronautical route planner. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*.

Savelsbergh, M. 1985. Local search in routing problems with time windows. *Annals of Operations Research* 4:285–305.

Schmidt, T.; Kuhn, L.; Price, B.; de Kleer, J.; and Zhou, R. 2009. A depth-first approach to target-value search. In *Proceedings of the Second Symposium on Combinatorial Search*.

Štěpán Kopřiva; Šišlák, D.; Pavlíček, D.; and Pěchouček, M. 2010. Iterative accelerated A* path planning. In *Proceedings of the Foty-nineth Conference on Decision and Control*.