On a Traveling Salesman Problem with Dynamic Obstacles and Integrated Motion Planning

Anja Hellander and Daniel Axehill

Hellander, A., Axehill, D., (2022), On a Traveling Salesman Problem with Dynamic Obstacles and Integrated Motion Planning, *2022 AMERICAN CONTROL CONFERENCE (ACC)*, , 4965-4972.
https://doi.org/10.23919/ACC53348.2022.9867369

Original publication available at:
https://doi.org/10.23919/ACC53348.2022.9867369

http://www.ieee.org/

On a Traveling Salesman Problem with Dynamic Obstacles and Integrated Motion Planning*

Anja Hellander[1] and Daniel Axehill[2]

Abstract— **This paper presents a variant of the Traveling Salesman Problem (TSP) with nonholonomic constraints and dynamic obstacles, with optimal control applications in the mining industry. The problem is discretized and an approach for solving the discretized problem to optimality is proposed. The proposed approach solves the three subproblems (waypoint ordering, heading at each waypoint and motion planning between waypoints) simultaneously using two nested graph-search planners. The higher-level planner solves the waypoint ordering and heading subproblems while making calls to the lower-level planner that solves the motion planning subproblem using a lattice-based motion planner. For the higher-level motion planner A* search is used and two different heuristics, a minimal spanning tree heuristic and a nearest insertion heuristic, are proposed and optimality bounds are proven. The proposed planner is evaluated on numerical examples and compared to Dijkstras algorithm. Furthermore, the performance and observed suboptimality are investigated when the minimal spanning tree heuristic cost is inflated.**

I. INTRODUCTION

One of the classical problems within combinatorial optimization, with important optimal control applications, is the Traveling Salesman Problem (TSP), where, given a set of waypoints and their pairwise distances, the objective is to find the shortest tour starting at one waypoint that visits all other waypoints and then returns to the first waypoint. In this work we will consider a variant of the TSP that is similar to a TSP variant of the multi-vehicle routing problem called MVRP-DDO [1]. We will not consider the problem of finding a tour, but rather a path from a given initial waypoint to a given terminal waypoint that visits all waypoints in a given set. We will consider a nonholonomic car-like vehicle that can move forwards, as well as in reverse, i.e., a Reeds-Shepp car [2]. Like the MVRP-DDO, we will assume that waypoints are dense compared to the size and the minimal turning radius of the vehicle, and that waypoints once they have been visited can not be visited again, i.e., they become obstacles that the vehicle must not collide with. We further constrain the vehicle such that it may only leave the waypoints moving in the forward direction.

The studied optimization problem is of practical interest for the mining industry, more specifically open-pit drill mining as described in [1]. A drilling machine is given a set of drill targets located within a given area. The drilling machine must navigate to each drill target, drill a hole and fill it with explosives. When the drilling machine has drilled a hole a small pile of dust is created around the hole. Drilling machines such as the Pit Viper 351 manufactured by Epiroc have a dust guard around the drill which can only be raised in one direction; hence the drilling machine can only leave the drill target in one direction. Driving over a hole that has been drilled will result in the hole being filled again; hence holes that have already been drilled become obstacles.

In [1], which considers motion planning for the open-pit mine, considers a TSP instance where visited waypoints become obstacles similat to that of this work. However, their approach does not guarantee that a feasible solution is found if one exists. Furthermore, the approach they present has been tailored to a particular type of problem, by assuming that waypoints are placed in roughly parallel lines, attempting to find these lines and moving along them. The solution presented in this work makes no such assumptions and can therefore easily be generalized to other waypoint placements. Another difference is that they consider a multi-vehicle problem where scheduling has to be taken into account, whereas this work only considers a single-vehicle problem. Furthermore, their presented solution only attempts to find a feasible solution to a discretized problem rather than a solution that is optimal or within proven suboptimality bounds as considered in this work.

A related TSP variant is the Dubins TSP (DTSP) which considers a Dubins car [3], i.e., only forward driving is allowed. This requires the determination not only of the relative ordering of the waypoints, but also of the vehicle heading at each waypoint which requires continuous decision variables. Most of the previous research on the DTSP has focused on obstacle-free environments, such as [4], [5], [6], [7], [8], [9]. In [10] static obstacles are considered; however, the vehicle heading at each waypoint is the same and given in advance. In [11] a Reeds-Shepp car was considered and the headings at each waypoint were constrained to an interval. In [12] a TSP problem with a unicycle model and obstacles is solved to optimality without discretization by reformulating the problem as a quadratically constrained quadratic program. Scenarios with both static and dynamic obstacles are considered, but unlike the problem considered in this work the location of the obstacles depend only on time and not on the order in which waypoints are visited. Hence it would not solve the problem considered here.

Early work on the DTSP usually disregarded the nonholonomic constraints initially and solved the problem as if it were a Euclidean TSP (ETSP) using standard methods

*This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

[1]Anja Hellander is with the Division of Automatic Control, Linköping University, 581 83 Linköping, Sweden anja.hellander@liu.se

[2]Daniel Axehill is with the Division of Automatic Control, Linköping University, 581 83 Linköping, Sweden daniel.axehill@liu.se

for TSP such as the Lin-Kernighan heuristic [13] or the Concorde TSP solver [14]. This gives the ordering of the waypoints, and the headings were determined in a later step. Examples of work that used this separation include [4], [6] and [15]. In [16] the ordering is determined first and the headings are determined in a separate step, although the ordering is not based on the solution of the ETSP. Solving the two problems separately is known to give a suboptimal solution [5], so later work has focused more on solving both the ordering and the heading problems simultaneously. A common approach is to discretize the heading angles, compute all possible distances and treat the problem as a generalized traveling salesman problem, such as in [5] and [11]. One approach that does not discretize the headings was presented in [7], where a genetic algorithm approach is used.

Adding the constraint that the vehicle may not pass over waypoints that have already been visited makes the problem more difficult. Given two waypoints and their respective headings, the shortest path in an obstacle-free environment for a Dubins or Reeds-Shepp car can be computed analytically and is well known [3] [2]. When obstacles are present the computational effort to find an optimal or near-optimal path is much greater. Additionally, since the obstacles in the current work are not static but depend on which waypoints have already been visited it is not possible to compute the distances between pairs of waypoints and headings, since the optimal feasible path will depend on which obstacles are present. Instead, the states between which distances are computed must also include information about previously visited waypoints, which increases the number of possible states and hence the number of distances to compute substantially. Suppose that there are n waypoints in total. Then for any waypoint there are 2^{n-1} possible combinations of previously visited nodes. Due to the large number of nodes and the comparatively high computational effort of computing arc costs, it might not be very effective to compute all arc costs and use a standard TSP solver.

The contributions in this work are the following. We present a new approach to the problem that discretizes the problem and uses discrete graph search to find an optimal solution to the discretized problem, whereas previous work only attempted to find a feasible solution. Our approach combines two different planners, one that plans in the space of waypoint orderings and (discretized) headings, and one that plans the (continuous) motion between given pairs of waypoints and headings. For the first planner, we propose the A* search algorithm [17] and propose two heuristics to help guide the search: one based on the nearest insertion algorithm, and one based on minimal spanning trees. We show that the use of the minimal spanning tree heuristic results in an optimal solution for the discretized search problem and prove suboptimality bounds when the proposed nearest insertion heuristic is used. We also consider using the minimal spanning tree heuristic inflated with a factor and show on numeric examples that this can result in much faster computation time with small or no sacrifices on optimality.

II. PROBLEM FORMULATION

For a car-like vehicle an initial configuration $q_s = (x_s, y_s, \theta_s)$ and a final configuration $q_g = (x_g, y_g, \theta_g)$ are given, as well as a set of target points $\{p_1, \ldots, p_n\}$ where the coordinates (x_i, y_i) are given for each target point p_i. The objective is to find the shortest path for the vehicle from q_s to q_g that passes through each target $p_1, \ldots, p_n$ under the constraint that when a target has been visited it becomes an obstacle and the vehicle may not pass over that point again. This requires the determination of the vehicle heading θ_i at each point p_i. Furthermore, although the vehicle in general can drive forwards as well as backwards it must drive forwards as it leaves a target point due to the fact that the target point once reached is converted into an obstacle. There may also be other (static) obstacles that limit the space that the vehicle can freely move in. In addition, the path must obey kinematic constraints.

The problem therefore consists of three dependent subproblems:

1) Decide the order in which $p_1, \ldots, p_n$ should be visited.
2) Decide the headings $\theta_i \in [0, 2\pi)$ for each target point p_i.
3) Decide the shortest path between the chosen configurations.

Note that the problem contains a mixture of discrete and continuous decision problems; the target ordering is a discrete problem whereas the choice of headings as well as the path planning are continuous problems.

III. METHOD

We start by restricting the headings to a set consisting of k possible angles, thereby discretizing the heading problem. To solve the three subproblems simultaneously, we use two nested planners: one higher-level planner that searches in the space of possible (discretized) headings and target orderings, and one discrete lower-level planner that performs the planning of the continuous motion between a pair of configurations given a set of obstacles.

A. The higher-level planner

The higher-level planner is an A* planner [17]. A search state is a triple $z = (p, \theta, S)$. The current position is $p = (x, y)$, which must coincide with either a target point, the start configuration or the end configuration. The current discretized heading is θ, and S is a set of positions (target points, start configuration, end configuration) that have been visited previously (the order in which they were visited does not matter).

The search is performed backwards, i.e. starting from a state $z_{root} = (p_{root}, \theta_{root}, S_{root})$ where $p_{root} = p_g = (x_g, y_g)$, $\theta_{root} = \theta_g$, $S = \{p_g\}$. The goal state is $z_{goal} = (p_{goal}, \theta_{goal}, S_{goal})$ where $p_{goal} = p_s = (x_s, y_s)$, $\theta_{goal} = \theta_s$, $S = \{p_s, p_g, p_1, \ldots, p_n\}$. The reason for performing backward search rather than forward search is the same as in [1]: if targets are dense in comparison to the vehicle's size and minimum turning radius, then obstacles will be dense as well for states where many targets have already been

visited. It can be expected that there will be many such states where there are no feasible collision free paths to unvisited targets. By performing the search backwards such states will be encountered early during the search and pruned, whereas if the search would have been performed forwards such states would not have been found until much later in the search, resulting in unnecessary exploration of states that will lead to dead ends further on.

To be able to perform the A* search the planner must be able to compute the cost-to-go function $h(z)$ as well as the cost-to-come function $g(z)$. For the cost-to-go function several different possible choices of heuristics are discussed later in Section III-C. To update the cost-to-come function, it is necessary to have a cost function $c(z_1, z_2)$ that gives the cost of moving from state z_1 to z_2, i.e., the length of the shortest feasible path from (x_2, y_2, θ_2) to (x_1, y_1, θ_1). (Note that due to the backward search, the vehicle will visit the states in reverse order when executing the plan, hence the path is planned from p_2 to p_1.) This path is computed by the lower-level motion planner.

B. The lower-level planner

To compute the cost of moving between configurations (x, y, θ), we use a lattice-based planner [18] based on A* and motion primitives.

The heuristic used for the A* search is the Euclidean distance to the goal state, which is an admissible heuristic and guarantees that the resulting solution is optimal. In order to handle obstacles the lower-level planner is called with an initial state $z = (p, \theta, S)$. All target points in S other than p are added as obstacles in the lattice-based planner. The current position p can not be added as an obstacle immediately since this would make the current state infeasible. Instead, the lattice-based planner first performs a single node expansion. After this first node expansion the position p is added as an obstacle and the lattice-based planner continues with the search. Since the vehicle is only allowed to leave an obstacle driving forwards, only motion primitives where the vehicle is driving forward are allowed for the first expansion. For subsequent expansions motion primitives where the vehicle is reversing are allowed. Since the position p has then been added as an obstacle the planner is prevented from trying to plan paths where p is passed. The lattice-based planner itself prevents p being passed with the same heading, since states cannot be revisited, but adding p as an obstacle also prevents the planner from trying to pass p at a different heading.

C. Heuristics for the higher-level planner

An important part of an A* planner is the choice of heuristic h. Two important properties of a heuristic are *admissibility* and *consistency*. A heuristic h is said to be *admissible* if for any node n it holds that $h(n) \leq h^*(n)$ where $h^*(n)$ is the cost of an optimal path from n to a goal state, i.e., an admissible heuristic never overestimates the cost-to-go. When an admissible heuristic is used, the resulting solution is optimal. A heuristic is said to be *consistent* if for any pair of nodes n, n' it holds that $h(n) \leq \text{cost}(n, n') + h(n')$. A consistent heuristic is guaranteed to only expand each node once, whereas heuristics that are not consistent may expand the same node multiple times, resulting in inefficiency. Consistency implies admissibility, but the converse does not hold [19].

For the problem of interest, given a current state consisting of current position and a list of positions that have previously been visited, the chosen heuristic should estimate the cost to get from this state to the goal state. Since backward search is used, the goal is the state where the current position is the start configuration and no other positions have been visited. The heuristic should therefore estimate the cost of the shortest path from the start configuration to the current configuration that passes through a given set of positions. We consider three different heuristics: $h = 0$, a minimal spanning tree heuristic and a nearest insertion heuristic.

If $h = 0$ is chosen, then A* reverts to Dijkstras algorithm [20]. This heuristic is admissible, so the resulting solution is optimal.

For the other two heuristics, consider the set of positions $S = \{\tilde{p}_0, \ldots, \tilde{p}_{m-1}\}$ that have been visited, where $\tilde{p}_0$ corresponds to the initial configuration and $\tilde{p}_{m-1}$ is the current configuration. Let $G = (V, E)$ be a complete undirected graph where each vertex v_i corresponds to $\tilde{p}_i$ and the set of edges E corresponds to the paths between them. Let the cost $d(v_i, v_j)$ of each edge (v_i, v_j) be the Euclidean distance between $\tilde{p}_i$ and $\tilde{p}_j$. Since this distance does not consider any obstacles or any vehicle kinematics it is a lower bound on the actual distance that the vehicle would have to travel between the two positions.

For the minimal spanning tree heuristic we compute the cost of a minimal spanning tree which can be found using for example Prim's algorithm [21] [20] or Kruskal's algorithm [22]. Since the optimal path from v_0 to v_{m-1} that passes through all other nodes is also a spanning tree, but not necessarily minimal, the cost of a minimal spanning tree gives an underestimate of the length of an optimal path and hence the heuristic is admissible and optimality of the solution is guaranteed.

For the nearest insertion heuristic, we modify the nearest insertion heuristic for the standard (tour) TSP [23], such that it returns a path with given start and end nodes instead of a tour. The resulting algorithm becomes as follows: start with a path T_1 consisting only of v_0 (the node that corresponds to the start configuration), and create the path T_2 by adding v_{m-1} (the node that corresponds to the current configuration in the A* search, and the end node for the nearest insertion heuristic) such that $T_2 = \{v_0, v_{m-1}\}$. For each $i = 2, \ldots, m-1$, find the vertex $v_k \in V \backslash T_i$ with the shortest distance to any node in T_i. Create the path T_{i+1} by inserting $a_i = v_k$ into T_i at the place that gives the lowest increase of cost. Here, the cost of a path is the sum of the distances along the path. The cost of a path consisting of a single node is defined as 0. When all nodes have been added to the path, the algorithm returns the cost of the path. This may very well result in an overestimate of the cost-to-go

and consequently the heuristic is not admissible. Although the nearest insertion heuristic for the standard tour-TSP is well-known, we are not aware of any work where the path-TSP variant of it has been studied, nor of any previous work where it has been used as a heuristic to guide a search algorithm such as A*. We will show that there is a bound on the suboptimality of the returned solution for the path-TSP problem, namely that the cost of the returned solution is strictly less than 3·OPTCOST where OPTCOST is the cost of the optimal path. Furthermore, we will show that this suboptimality bound extends to the solution returned by the A* search algorithm.

D. Suboptimality bound for nearest insertion

To prove the suboptimality bound we will first establish some lemmas.

Lemma 1. Let $h(n)$ be a heuristic, not necessarily admissible. Then, at any point before the search has terminated there exists a node n' in the set of open nodes such that n' is on an optimal path and the predecessors of n' are along an optimal path, i.e., $g(n') = g^*(n')$, where $g(n)$ is the cost-to-come and $g^*(n)$ is the optimal cost-to-come.

Proof of Lemma 1. See the proof of Lemma 1 in [17].

Lemma 2. Let $h(n)$ be a heuristic such that $h(n) \leq wh^*(n)$ for all n for some $w > 1$. For the cost of the solution returned by A* using $h(n)$ as heuristic, COST, and the cost of an optimal solution OPTCOST the relation COST $\leq w\cdot$ OPTCOST holds.

Proof of Lemma 2. Suppose that the expansion of nodes is along an optimal path. Then the cost of the found solution is equal to the optimal cost and the lemma holds trivially. Suppose therefore that the resulting solution was not expanded along an optimal path. Let the node in the goal set that was expanded be denoted n_g. Immediately before n_g was expanded, by Lemma 1 there was some node n' on an optimal path, in the set of open nodes, such that $g(n') = g^*(n')$. Since A* expands the node in the open set that minimizes $f(n) = g(n) + h(n)$, and n_g was expanded but n' was not, it follows that we must have

$$\begin{aligned} f(n_g) \leq f(n') &\iff \\ g(n_g) + h(n_g) \leq g(n') + h(n') &\iff \\ g(n_g) \leq g^*(n') + wh^*(n') &\iff \\ \text{COST} \leq w(g^*(n') + h^*(n')) = w \cdot \text{OPTCOST}, \quad (1) \end{aligned}$$

which proves the lemma.

For TSP problems on complete graphs where the distance function satisfies the triangle inequality it is known that the ratio between the cost of tours obtained by the nearest insertion algorithm and the cost of the optimal tour is bounded from above by 2 [23]. However, the TSP considered in this work is not such a TSP since we are looking for a path that starts in a given node and ends in a given node (distinct from the start node). There are two differences between the nearest insertion algorithm proposed in this work and the standard TSP nearest insertion algorithm. Firstly, in this work, the second node to be added is required to be the end node which means that this node is not chosen in accordance to the same criterion as the remaining nodes (i.e. that the node closest to the path should be added). Secondly, since the path considered here is a path and not a closed tour, nodes may not be inserted between the end node and the start node.

Lemma 3. Let a_i be the node added to path T_i, $i \geq 2$. Then, for any node $a_j \notin T_i$ and any node $a_k \in T_i$ it holds that

$$cost(T_{i+1}) - cost(T_i) \leq 2d(a_k, a_j), \quad (2)$$

where $d(a_j, a_k)$ is the cost of the edge between a_j and a_k.

Proof of Lemma 3. For $i \geq 2$ the edge (a_k, a_l) in T_i that minimizes $d(a_k, a_i) + d(a_i, a_l) - d(a_k, a_l)$ is chosen. So for any edge (a_k, a_l) we have $cost(T_{i+1}) - cost(T_i) \leq d(a_k, a_i) + d(a_i, a_l) - d(a_k, a_l)$. Since the edge costs satisfy the triangle inequality we have $d(a_i, a_l) \leq d(a_i, a_k) + d(a_k, a_l) \iff d(a_i, a_l) - d(a_k, a_l) \leq d(a_i, a_k)$ which gives us $cost(T_{i+1}) - cost(T_i) \leq d(a_k, a_i) + d(a_i, a_k) = 2d(a_i, a_k)$. The triangle inequality also gives $d(a_k, a_i) \leq d(a_k, a_l) + d(a_l, a_i) \iff d(a_k, a_i) - d(a_k, a_l) \leq d(a_l, a_i)$ which gives $cost(T_{i+1}) - cost(T_i) \leq d(a_i, a_l) + d(a_l, a_i) = 2d(a_i, a_l)$. Since this holds for any edge (a_k, a_l) it means that $cost(T_{i+1}) - cost(T_i) \leq 2d(a_i, a_k)$ for any node a_k in T_i. In particular it holds for whichever node $a_{k'}$ that minimizes $d(a_i, a_k)$. For all nodes a_j in $V \backslash T_i$ and all nodes a_k in T_i we have, $d(a_k, a_j) \geq d(a_{k'}, a_i)$ and consequently $cost(T_{i+1}) - cost(T_i) \leq 2d(a_i, a_{k'}) \leq 2d(a_k, a_j)$ and the lemma is proven.

Theorem 1. If a path of length NEAREST is obtained by the proposed nearest insertion algorithm, and the length of the optimal path is OPTIMAL, then $\frac{\text{NEAREST}}{\text{OPTIMAL}} \leq 3$.

Proof of Theorem 1. Let M be a minimal spanning tree on G, and let as before a_i be the node that is added to the path T_i in the nearest insertion algorithm. For any a_i there is a unique edge e_i in M such that e_i is an edge between one node in T_i and one node not in T_i, and if $i \neq j$ then $e_i \neq e_j$. The proof for this can be found in the proof of Lemma 3 in [23]. Then, by Lemma 3 it holds that for $i \geq 2$,

$$cost(T_{i+1}) - cost(T_i) \leq 2d(e_i). \quad (3)$$

The cost of the resulting path can be rewritten as

$$\begin{aligned} \text{NEAREST} &= \sum_{i=1}^{n-2} cost(T_{i+1}) - cost(T_i) = \\ &= cost(T_2) + \sum_{i=2}^{n-2} cost(T_{i+1}) - cost(T_i) \leq \\ &\leq cost(T_2) + 2\sum_{i=2}^{n-2} d(e_i). \quad (4) \end{aligned}$$

The sum on the right hand side is the sum of all edges except for one of M, which is clearly no more than OPTIMAL. The cost $cost(T_2)$ is the distance between the start and end nodes. Since all distances obey the triangle inequality, no path that starts at the start node and ends at the end node

can be shorter, so this is not greater than OPTIMAL. This gives us

$$\text{NEAREST} \leq \text{OPTIMAL} + 2 \cdot \text{OPTIMAL} = 3 \cdot \text{OPTIMAL} \tag{5}$$

which proves the statement.

It should be noted that this bound is conservative, i.e., will not be achieved. If the cost $cost(T_2)$ is in fact equal to the optimal length, then that must mean that all remaining nodes can be inserted at zero cost, and the heuristic will find the optimal path.

Theorem 2. Let NEAREST be the cost of a path obtained by A* using the nearest insertion algorithm as heuristic, and let OPTCOST be the cost of an optimal path. Then NEAREST $\leq 3 \cdot$OPTCOST.

Proof of Theorem 2. By Theorem 1 the nearest insertion heuristic, h_n satisfies $h_n(n) \leq 3h^*(n)$ for all nodes n, where $h^*(n)$ is the true cost-to-go. It also satisfies $h_n(n_g) = 0$ for all nodes n_g in the goal region. The result now follows from Lemma 2.

IV. NUMERICAL EXPERIMENTS

For numerical experiments we considered a drill rig modeled as a simple car with state $(x, y, \theta, \alpha, \omega)$, where (x, y) is the position of the center of the rear axle of the vehicle, θ is the heading, α is the steering wheel angle, and ω is the steering wheel angular rate. The control u is the derivative of ω, i.e. the steering wheel angular acceleration. The velocity is taken as 1 when driving forward and -1 when driving in reverse. In forward mode the model dynamics are

$$\dot{x} = \cos\theta \tag{6a}$$
$$\dot{y} = \sin\theta \tag{6b}$$
$$\dot{\theta} = \frac{\tan\alpha}{L} \tag{6c}$$
$$\dot{\alpha} = \omega \tag{6d}$$
$$\dot{\omega} = u \tag{6e}$$

where L is the wheel base of the vehicle.

The steering angle was constrained to $-\frac{\pi}{4} \leq \alpha \leq \frac{\pi}{4}$ and the wheel base was set to $L = 2.912$, which then equals the minimal turning radius. The width of the vehicle was set to $w = 1.3$, slighty less than half the wheel base. For the lower-level lattice planner a grid resolution of 0.2 was used and motion primitives were generated using CasADi [24] and IPOPT [25]. The proposed planner was applied to a few small-size examples with varying number of target points and different feasible areas. In the examples, targets were placed on a regular grid with resolution 3. This is similar to the vehicle length and minimal turning radius, and about twice the vehicle width, and the targets can therefore be considered to be densely placed. For each example, the three heuristics $h = 0$, the minimal spanning tree heuristic and the nearest insertion heuristic were used in the higher-level planner. The resulting computation times, path lengths, number of node expansions performed by the higher-level planner and number of calls to the lower-level planner from the higher-level planner were noted.

TABLE I

COMPUTATION TIME, NUMBER OF NODE EXPANSIONS, NUMBER OF CALLS TO THE LOWER-LEVEL PLANNER AND RESULTING PATH LENGTHS FOR THE THREE HEURISTICS $h = 0$, MINIMAL SPANNING TREE (MST) AND NEAREST INSERTION (NI) FOR AN EXAMPLE WITH 12 NODES.

	h=0	h = MST	h = NI
Computation time [s]	195.501	83.1209	60.7963
Node expansions	756	380	276
Calls to lower-level planner	13616	7264	5744
Path length	115.538	115.538	115.538

The first example considered consisted of twelve target points placed regularly as shown in Figure 1. The start configuration is $(-3, 0, \frac{\pi}{2})$ and the end configuration is $(3, 9, \frac{\pi}{2})$. Heading angles were constrained to $\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$. The resulting computation times, path lengths, number of node expansions in the higher-level planner, and number of calls to the lower-level planner for the three heuristics are shown in Table I. All three heuristics result in the same path length, 115.538, which is optimal since two of the heuristics are admissible. The found path is shown in Figure 2. The path is more complex than what might have been expected. However, since the feasible area is so restricted and the target points are so close to each other relative to the length and width of the vehicle, the vehicle becomes very restricted in which maneauvers that are feasible. For instance, if target 7 were drilled after target 3, it would not be possible for the vehicle to drive to target 4 without leaving the feasible area or driving over a target that has already been drilled, which would not be a feasible solution. The proposed nearest insertion heuristic has the lowest computation time (61 seconds) as well as the lowest number of node expansions and calls to the lower-level planner. The basic heuristic $h = 0$ is by far the slowest, resulting in about 3 times as long compution time as when the nearest insertion heuristic is used.

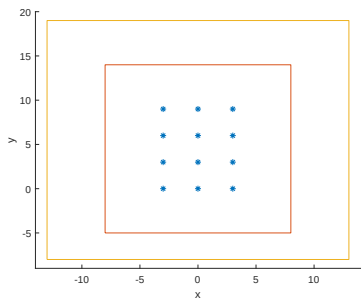

Fig. 1. The setup for the first example. The target points (blue stars) and the boundary of the feasible area (red lines) as well as the boundary of the increased feasible area (yellow lines).

A modification of this example where the feasible area was increased was considered as well. The results are shown in Table II. All heuristics resulted in the same optimal path with length 72.219, which is about 60 % of the previous path length. The resulting optimal path is shown in Figure 3. It can be seen that the ordering of the target points is different

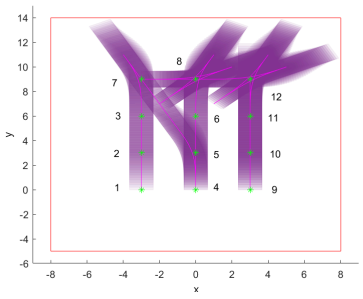

Fig. 2. The optimal path of the rear axle center found for the first example. Waypoints are numbered according to the order in which they are visited. The swept path of the vehicle is shown in purple.

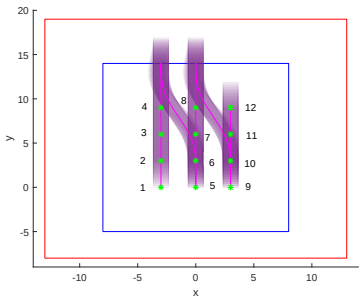

Fig. 3. The optimal path of the rear axle center found for the first example with increased feasible area. Waypoints are numbered according to the order in which they are visited. The swept path of the vehicle is shown in purple. The blue lines denote the original feasible area.

compared to the previous example. With the increased area it is now possible to continue straight a bit after the fourth target point before reversing to the fifth target point without driving over any previous target point or leaving the feasible area, something that due to the length of the vehicle was not previously possible. For both the minimal spanning tree heuristic and the nearest insertion heuristic, the number of node expansions decreased drastically compared to when the previous feasible area was used, from 380 to 130 expansions and from 276 to 87 expansions respectively. The number of calls to the lower-level planner also decreased on similar scales. For the $h = 0$ heuristic there was a slight decrease in the number of node expansions and an increase in the number of calls to the lower-level planner. For all three heuristics the computation was much higher compared to the previous example. For all heuristics, this indicates that the average computation time for a call to the lower-level planner increased when the feasible area increased. This can be explained by that with an increased feasible area there are now feasible paths between more states than previously. The proportion of calls to the lower-level planner that actually returns a feasible path would then be higher, resulting in longer computation times since a path has to be computed instead of the planner finding early in the search that the problem is infeasible. With a larger area there are also more possible nodes for the lattice-based planner to explore, and it might therefore take longer time to discover that a problem is infeasible.

TABLE II

COMPUTATION TIME, NUMBER OF NODE EXPANSIONS, NUMBER OF CALLS TO THE LOWER-LEVEL PLANNER AND RESULTING PATH LENGTHS FOR THE THREE HEURISTICS $h = 0$, MINIMAL SPANNING TREE (MST) AND NEAREST INSERTION (NI) FOR AN EXAMPLE WITH 12 NODES WITH INCREASED FEASIBLE AREA.

	h=0	h = MST	h = NI
Computation time [s]	1645.78	245.949	196.773
Node expansions	733	113	87
Calls to lower-level planner	17916	3112	2428
Path length	72.219	72.219	72.219

A second example with 16 target points was considered as well. The target points were placed as shown in Figure 4. The start configuration is $(-6, 0, \frac{\pi}{2})$ and the end configuration is $(3, 0, \frac{3\pi}{2})$, which coincide with two of the target points. The heading angles were discretized in the same way as in the previous example. The results are shown in Table III. All heuristics found the same path which is optimal. The found path is shown in Figure 5. Although the number of targets has only increased with 33 %, the number of node expansions is almost 10 times higher for both the minimal spanning tree heuristic and the nearest insertion heuristic and more than 10 times higher for the zero heuristic. The increase in computation time is around 15 times for the nearest insertion and the minimal spanning tree heuristics, and slightly higher for the zero heuristic. This means that the average computation time per node expansion has also increased, which for the nearest insertion and minimal spanning tree heuristics is expected since they run in $O(n^2)$ and $O(n^2 \log n)$ respectively. The feasible area is also larger than in the first example, which might contribute to the increased computation time as seen previously. In this example the zero heuristic is still by far the slowest, and the nearest insertion heuristic the fastest.

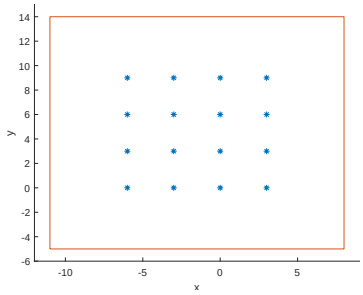

Fig. 4. The setup for the second example. The target points (blue stars) and the boundary of the feasible area (red lines).

Since the studied problem clearly is a very difficult problem, the possibility for computing suboptimal solutions was also considered. Inflating an admissible heuristic h with a factor $\epsilon > 1$, i.e., using the heuristic function $h_\epsilon = \epsilon h$, is known to result in a heuristic that returns a solution with a cost no greater than ϵ times the optimal one [19]. Since h is admissible, $h_\epsilon(n) \leq \epsilon h^*(n)$, and Lemma 2 can be applied. For the two numerical examples, the minimal spanning tree heuristic inflated with $\epsilon = 1.5, \epsilon = 2, \epsilon = 3$ were used. The

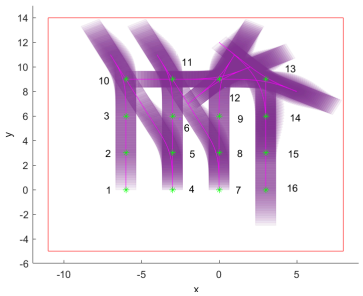

Fig. 5. The optimal path of the rear axle center found for the second example. Waypoints are numbered according to the order in which they are visited. The swept path of the vehicle is shown in purple.

TABLE III

COMPUTATION TIME, NUMBER OF NODE EXPANSIONS AND RESULTING PATH LENGTHS FOR THE THREE HEURISTICS $h = 0$, MINIMAL SPANNING TREE (MST) AND NEAREST INSERTION (NI) FOR AN EXAMPLE WITH 16 NODES.

	h=0	h = MST	h = NI
Computation time [s]	3581.17	1325.85	964.674
Node expansions	8827	3621	2721
Calls to lower-level planner	217036	94252	74232
Path length	145.741	145.741	145.741

results for the example with 12 targets are shown in Table IV. For all three values of ϵ, the optimal path length is found. For $\epsilon = 1.5$ the performance in terms of computation time and number of node expansions is about the same as for the nearest insertion heuristic. For $\epsilon = 2$ the computation time is slightly below 75 % of the computation time for the nearest insertion heuristic, and for $\epsilon = 3$ the computation time is about one third of the computation for the nearest insertion heuristic.

The results for the example with 16 targets are shown in Table V. Again, the optimal path length is found for all three values of ϵ. Computation times are improved compared to the uninflated minimal spanning tree heuristic; the computation time for $\epsilon = 1.5$ is 67 % of the computation time for the uninflated heuristic (92 % of the the computation time for the nearest insertion heuristic), the computation time for $\epsilon = 2$ is about 28 % of that of the uninflated heuristic, and for $\epsilon = 3$ the computation time is only 5 % of the computation time for the uninflated heuristic.

The results from both examples show that although the

TABLE IV

COMPUTATION TIME, NUMBER OF NODE EXPANSIONS, NUMBER OF CALLS TO THE LOWER-LEVEL PLANNER AND RESULTING PATH LENGTHS FOR THE INFLATED MINIMAL SPANNING TREE HEURISTIC FOR AN EXAMPLE WITH 12 NODES.

	$\epsilon = 1.5$	$\epsilon = 2$	$\epsilon = 3$
Computation time [s]	57.2963	44.9409	19.8687
Node expansions	282	224	101
Calls to lower-level planner	5676	4648	1932
Path length	115.538	115.538	115.538

TABLE V

COMPUTATION TIME, NUMBER OF NODE EXPANSIONS, NUMBER OF CALLS TO THE LOWER-LEVEL PLANNER AND RESULTING PATH LENGTHS FOR THE INFLATED MINIMAL SPANNING TREE HEURISTIC FOR AN EXAMPLE WITH 16 NODES.

	$\epsilon = 1.5$	$\epsilon = 2$	$\epsilon = 3$
Computation time [s]	885.436	376.218	60.8159
Node expansions	2400	1265	264
Calls to lower-level planner	63696	33516	6844
Path length	145.741	145.741	145.741

number of node expansions and the computation time do not scale well with the problem size for the minimal spanning tree heuristic or the nearest insertion heuristic, they offer great improvement compared to using no heuristic at all. For the examples that were tested the nearest insertion heuristic resulted in optimal path length in all cases, despite that the theory presented only guarantees a path length no greater than 3 times the optimal one. Since the examples used have densely placed targets the actual distance required for the vehicle to move between targets can sometimes be much greater than the Euclidean distance which is used in the heuristics. This could then balance the fact that the heuristics do not always return the optimal path given Euclidean distances, and the actual distance returned might still be an underestimate of the true cost-to-go, making the heuristics admissible in practice in such cases.

The results from the numerical experiments also indicate that the nearest insertion heuristic results in a computationally faster search than the minimal spanning tree, but at the cost of the loss of optimality guarantees. Inflating the minimal spanning tree with a factor $\epsilon > 1$ greatly improves the computation time and number of node expansions, performing better than the nearest insertion for all three values of ϵ in both examples. This comes at the cost of loss of optimality guarantees, but the cost of the found path is still guaranteed to be within a factor ϵ of the optimal. In all cases the optimal path length was achieved, indicating that the approach has the potential to have a practical value. This also shows that there is potential to increase the target size; even if the computation times for the uninflated minimal spanning tree and nearest insertion heuristics become too long for those heuristics to be tractable, an inflated minimal spanning tree heuristic could still be possible to use and might give optimal or near optimal solutions in practice.

V. CONCLUSIONS

We have proposed an approach to a TSP with nonholonomic constraints and dynamic constraints, with optimal control applications. The solution proposed is a higher-level planner that uses A* to search in the space of waypoints ordering and discretized headings, and a lower-level planner in the form of a lattice-based motion planner that performs the motion planning between waypoints. Furthermore, a minimal spanning tree heuristic as well as a path-TSP variant of the nearest insertion heuristic that can be used as heuristics

in the higher-level planner in order to guide the A* search are proposed. The minimal spanning tree heuristic was shown to be admissible, and hence the resulting solution is guaranteed to be optimal. For the nearest insertion heuristic, theoretical bounds on the suboptimality were proven.

Results from the numerical experiments show that the proposed method is of practical relevance, in particular for small-size problems or if suboptimal solutions can be accepted. The proposed minimal spanning tree and nearest insertion heuristics offer great improvement in terms of computation time and node expansions compared to using no heuristic at all. Furthermore, numerical experiments show that inflating the minimal spanning tree heuristic greatly improves computation times while still returning optimal or near optimal solutions in practice.

As future work, the path obtained from the discrete planner might also be improved by adding a final optimization step where local optimization is performed on the path, as in [26]. Another possible future extension is to consider larger problems where the targets are placed regularly according to some pattern, and turn the higher-level planner into a lattice-based planner with advanced motion primitives, computed using the proposed solution in this work. Future work could also include investigating if there are other problems where the approach presented in this work could be applied.

ACKNOWLEDGMENT

The lattice-based planner used as the lower-level planner was originally implemented by Kristoffer Bergman and Oskar Ljungqvist.

REFERENCES

[1] Masoumeh Mansouri, Fabien Lagriffoul, and Federico Pecora. Multi vehicle routing with nonholonomic constraints and dense dynamic obstacles. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3522–3529. IEEE, 2017.

[2] James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[3] Lester E Dubins. 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(3):497–516, 1957.

[4] Xiang Ma and David A Castanon. Receding horizon planning for Dubins traveling salesman problems. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 5453–5458. IEEE, 2006.

[5] Jerome Ny, Eric Feron, and Emilio Frazzoli. On the Dubins traveling salesman problem. *IEEE Transactions on Automatic Control*, 57(1):265–270, 2011.

[6] Ketan Savla, Emilio Frazzoli, and Francesco Bullo. Traveling salesperson problems for the Dubins vehicle. *IEEE Transactions on Automatic Control*, 53(6):1378–1391, 2008.

[7] Xin Yu and John Y Hung. A genetic algorithm for the dubins traveling salesman problem. In *2012 IEEE International Symposium on Industrial Electronics*, pages 1256–1261. IEEE, 2012.

[8] Luitpold Babel. New heuristic algorithms for the Dubins traveling salesman problem. *Journal of Heuristics*, pages 1–28, 2020.

[9] Izack Cohen, Chen Epstein, and Tal Shima. On the discretized dubins traveling salesman problem. *IISE Transactions*, 49(2):238–254, 2017.

[10] Wang Zhong and Li Yan. A target visiting path planning algorithm for the fixed-wing UAV in obstacle environment. In *Proceedings of 2014 IEEE Chinese Guidance, Navigation and Control Conference*, pages 2774–2778. IEEE, 2014.

[11] Sivakumar Rathinam, Satyanarayana G Manyam, and Yuntao Zhang. Near-optimal path planning for a car-like robot visiting a set of waypoints with field of view constraints. *IEEE Robotics and Automation Letters*, 4(2):391–398, 2019.

[12] Chuangchuang Sun, Nathaniel Kingry, and Ran Dai. A Unified Formulation and Nonconvex Optimization Method for Mixed-Type Decision-Making of Robotic Systems. *IEEE Transactions on Robotics*, 37(3):831–846, 2020.

[13] Keld Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[14] Concorde TSP Solver. `http://www.math.uwaterloo.ca/tsp/concorde/index.html`. Accessed: 2021-03-12.

[15] Sivakumar Rathinam, Raja Sengupta, and Swaroop Darbha. A resource allocation algorithm for multivehicle systems with nonholonomic constraints. *IEEE Transactions on Automation Science and Engineering*, 4(1):98–104, 2007.

[16] Zhijun Tang and Umit Ozguner. Motion planning for multitarget surveillance with mobile sensor agents. *IEEE Transactions on Robotics*, 21(5):898–908, 2005.

[17] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[18] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[19] Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

[20] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[21] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[22] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[23] Daniel J Rosenkrantz, Richard E Stearns, and Philip M Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

[24] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[25] COIN-OR Interior Point Optimizer IPOPT. `https://github.com/coin-or/Ipopt`. Accessed: 2021-03-11.

[26] Kristoffer Bergman, Oskar Ljungqvist, Torkel Glad, and Daniel Axehill. An Optimization-Based Receding Horizon Trajectory Planning Algorithm. *arXiv preprint arXiv:1912.05259*, 2019.