2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

On Integrated Optimal Task and Motion Planning for a Tractor-Trailer Rearrangement Problem

Anja Hellander, Kristoffer Bergman and Daniel Axehill

Abstract— **In this work, a combined task and motion planner for a tractor and a set of trailers is proposed and it is shown that it is resolution complete and resolution optimal. The proposed planner consists of a task planner and a motion planner that are both based on heuristically guided graph-search. As a step towards tighter integration of task and motion planning, we use the same heuristic that is used by the motion planner in the task planner as well. We further propose to use the motion planner heuristic to give an initial underestimate of the motion costs that are used as costs during the task planning search, and increase this estimate gradually by using the motion planner to verify the cost and feasibility of actions along paths of interest. To limit the time spent in the motion planner, the use of time and cost limits to pause or prematurely abort the motion planner is proposed, which does not affect the resolution completeness or resolution optimality. The planner is evaluated on numerical examples and the results show that the proposed planner can significantly reduce the execution time compared to a baseline resolution optimal task and motion planner.**

I. INTRODUCTION

There are many planning problems which require both high-level planning (task planning) in the form of deciding which sequence of actions to take in order to reach some goal, as well as low-level planning in the form of motion planning in order to determine how to execute the actions. For some problems it is possible to take a hierarchical approach, first performing task planning and then performing motion planning for each action in the plan. However, since the task planning does not consider geometrical, kinematical or dynamical constraints there is no guarantee that a feasible motion plan exists for the found task plan or that optimizing a motion plan for a given task plan leads to an optimal solution to the joint task and motion planning problem [19]. For this reason it is often desirable to perform joint task and motion planning.

A. Task and motion planning

Recent years have seen a lot of interest in integrated task and motion planning (TAMP). A survey of existing literature can be found in [6]. Most of the early work focused on finding a feasible but not necessarily optimal solution, which is a difficult problem in itself. One example is the so-called *semantic attachments* [4], where the task planner makes calls to external procedures such as a motion planner in order to evaluate the feasibility of an action. Another common approach is to first find a task plan using symbolic references to continuous variables corresponding to the continuous states of the platform in the world, and later attempt to find values for those symbolic references that result in a feasible task and motion plan [18].

A type of problem that has been commonly studied is problems in which a manipulator manipulates or rearranges moveable objects [21], [7], [11]. In this work we consider an example of one such problem, where the manipulator is a car-like tractor (truck) and the moveable objects are trailers that can be connected to the tractor.

Recently, there has been an increased interest in finding solutions to TAMP problems that are not only feasible but also optimal. One such early work was [22] where the problem is formulated as a multi-level optimization problem. An asymptotically optimal sampling-based approach is proposed in [19]. In [8] optimization over the values for symbolic references are performed, but there is no joint optimization of task and motion. [20] considers a more general form of TAMP where the objective is not expressed in terms of reaching a set of goal states but in terms of maximizing the value of a function, e.g., maximizing the height of a tower being built.

B. Contributions

We propose a method for integrated task and motion planning for a vehicle application involving a tractor and a set of trailers, which is shown to be resolution complete and resolution optimal. The combined task and motion planner consists of a task planner and a motion planner that are both graph-search based and guided by admissible, consistent heuristics. In this work we take steps towards integrating the task and motion planners more tightly, e.g., by basing the heuristic used by the task planner on the heuristic used by the motion planner. To improve efficiency and limit the time spent in the motion planner we propose to initially use underestimates of the true costs, computed by the motion planner heuristic, as the costs used in the task planning search and gradually use the motion planner to compute better estimates for paths of particular interest. We also introduce upper bounds on the cost-to-come during the search in order to be able to safely prematurely abort the motion planner, similar to what is known from mixed-integer programming and branch-and-bound [5].

II. GRAPH-SEARCH METHODS PRELIMINARIES

In practice, most methods for task planning as well as motion planning are based on graph-search. The most commonly used method for finding optimal cost paths on a graph is A* [10]. The A* algorithm incrementally builds and

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

A. Hellander and D. Axehill are with the Division of Automatic Control, Linköping University, Sweden {anja.hellander, daniel.axehill}.@liu.se

K. Bergman is with RISE Research Institutes of Sweden, kristoffer.bergman@ri.se

maintains a tree of nodes starting from the initial node until it reaches a goal node. In each iteration, a new node n in the tree is chosen for expansion from the so-called open list of nodes, and its successors, i.e., nodes v such that an edge (n, v) exists, are generated and added to the tree. Nodes in the open list are sorted based on the function

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the cost-to-come of node n, i.e., the lowest cost of a path from the initial node to n, and $h(n)$ is a heuristic function that estimates the cost-to-go, i.e., the lowest cost of a path from n to the goal node. A heuristic is said to be *admissible* if for all nodes n

$$h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost-to-go. It is said to be *consistent* if for all nodes n and all successors m of n the inequality

$$h(n) \leq h(m) + c(n, m)$$

holds, where $c(n, m)$ denotes the cost of the edge (n, m). It is well-known that if the heuristic is admissible, the solution returned by A* is optimal, and if it is consistent then the f-values of the nodes chosen for expansion by A* are monotonically non-decreasing [10].

One extension of A* is Lifelong Planning A* (LPA*) [13] which is designed to efficiently repeatedly solve graph-search problems with the same initial and goal nodes but with varying edge costs. In addition to $g(n)$, each node maintains an additional estimate $rhs(n)$ of the cost-to-come. By comparing $g(n)$ and $rhs(n)$, nodes where the cost-to-come may have changed can be identified. In LPA*, the open list consists of those nodes where $g(n) \neq rhs(n)$, and the nodes in the open list are sorted by a two-dimensional key $k(n) = [\min(g(n), rhs(n)) + h(n); \min(g(n), rhs(n))]$. Sorting is done lexicographically, i.e., $[k_1, k_2] < [k_1', k_2']$ if $k_1 < k_1'$ or if $k_1 = k_1'$ and $k_2 < k_2'$. Like A*, if a consistent and admissible heuristic is used, LPA* finds the optimal path given the current edge costs, and the key values of the nodes chosen for expansion are monotonically non-decreasing during each re-planning loop [14].

One application for A* is motion planning, where it can be used in lattice-based motion planners [17]. In a lattice-based motion planner, the continuous search space is discretized and the resulting discretized states are connected using pre-computed so-called motion primitives. The motion primitives can be computed by, e.g., using numerical optimization as in [17]. A* search is then performed on the resulting graph where the nodes correspond to the discretized states and the edges correspond to the motion primitives.

Another application of graph search is task planning. In classical task planning [9], the state space is discrete and a state is typically represented as a set of propositions that hold. The state can be changed by executing actions. Actions are defined by the parameters they take, their preconditions (propositions that must hold in order to execute the action) and their effect (how the state changes when the action is executed). This can be modelled as a graph where the nodes correspond to the discrete states and the edges correspond to actions. A* can then be applied to this graph in order to find a sequence of actions that transforms a given initial state to a given goal state, or more commonly to a state belonging to a set of goal states.

III. PROBLEM FORMULATION

We consider a problem in which a car-like tractor (truck) moves in a world containing static obstacles as well as movable obstacles in the form of trailers which can be connected to the tractor one at a time, thus forming a 2-trailer system [1]. Models for the car-like tractor as well as the tractor-trailer can be found in [3]. The task of the tractor is to move some or all of the trailers to some user-specified locations. While the application used in this work is that of a tractor moving trailers, the approach can be applied to other problems in which a manipulator rearranges moveable objects, provided that a motion planner with the same properties as in Section IV-B is used.

An important aspect, that we would like to stress in this work, is that as little explicit information as possible about the geometric state of the world should be encoded into the task planning problem. Instead, the motion planner should be used to obtain such information. This is motivated by that in advanced problems this would potentially require a significant effort of manually modelling as well as computing the information. In this spirit, the task planner has no explicit information about geometry in the form of, e.g., predicates indicating whether a trailer is blocked by another trailer. The task state only includes symbolic information about the location of the truck and all trailers, information about which trailer (if any) the trailer is connected to, and for each location if there is currently a trailer there or not. All other information must be discovered during the planning. For each symbolic location the corresponding continuous state (coordinates) for both the trailer and the tractor are stored by the task planner. We will refer to this continuous state as the motion planning state.

The objective is to find a *joint* task and motion plan, consisting of a sequence of actions that will take the system from a given initial task state to a given goal state (or set of goal states) together with associated motion plans detailing the motions corresponding to the move actions such that the total cost of the actions in the plan is minimized. Denoting the space of possible task states as $\mathcal{S}$, the space of possible actions as $\mathcal{A}$, the space of plans $\Pi = \{< a_0, a_1, \cdots >\}$, the initial state as s_{init} and the set of goal states as S_{goal} this can be formulated as

$$\begin{aligned} \min_{\pi \in \Pi} \quad & \sum_{i=0}^{|\pi|} c(s_i, a_i) \\ s.t. \quad & s_0 = s_{init} \\ & s_{i+1} = \gamma(s_i, a_i) \quad i = 0, \ldots, |\pi| \\ & s_{|\pi|} \in S_{goal} \\ & \pi = < a_0, a_1, \ldots, a_{|\pi|} > \end{aligned} \tag{1}$$

where $c(s_i, a_i)$ denotes the cost of applying action a_i in state s_i, $\gamma(s, a)$ is the resulting state when a is applied in state s and π denotes an action sequence.

We have currently only considered three different actions in the task planner: connect a trailer to a tractor, disconnect

a trailer from a tractor and move the tractor (possibly with a trailer connected) from one location to another, but our approach can easily be applied where other actions exist as well. For connect and disconnect actions a, the cost at state s is

$$c(s,a) = \begin{cases} C & \text{if } a \text{ is applicable in } s \\ \infty & \text{if } a \text{ is not applicable in } s \end{cases} \quad (2)$$

for some constant $C > 0$.

For a move action a and task state s we have

$$\begin{aligned} c(s,a) = \min_{x(\cdot),u(\cdot),\Sigma_f,q(\cdot)} \quad & J = \int_0^{\Sigma_f} l(x(\sigma),u(\sigma),q(\sigma))\mathrm{d}\sigma \\ s.t. \quad & x(0) = x_{init} = x(s) \\ & x(\Sigma_f) = x_{term} = x(\gamma(s,a)) \\ & x'(\sigma) = f_{q(\sigma)}(x(\sigma),u(\sigma)) \\ & x(\sigma) \in \mathcal{X}_{free}(s) \\ & q(\sigma) \in \mathcal{Q}, \quad u(\sigma) \in \mathcal{U} \end{aligned} \quad (3)$$

where $\sigma > 0$ is defined as the distance travelled by the system, Σ_f is the total path length, $x \in \mathcal{X}$ is the state vector, $u \in \mathcal{U}$ is the control input and $q \in \mathcal{Q}$ is a discrete input signal which selects the current mode of the system such as, e.g., driving forward or in reverse, and if a trailer is connected or not. The obstacle-free part of the state space is denoted $\mathcal{X}_{free}$ and depends on s. The continuous state, or motion planning state, of the vehicle when in task state s is denoted $x(s)$ and will be the initial state of (3). The running cost used to define the performance measure J is $l(x,u,q)$, where $l(x,u,q) > 0$ to ensure $c(s,a) > 0$.

Solving (3) to global optimality is difficult due to the aspect of choosing the system mode q, and the non-convex constraints introduced by the obstacles and the nonlinear system model [3]. Solving (1) to global optimality is even more difficult as it involves solving (3) to global optimality as well as finding an action sequence and choosing continuous coordinates that correspond to the symbolic positions used by the task planner.

IV. COMBINED TASK AND MOTION PLANNER

As solving the combined task and motion planning problem to global optimality is a difficult problem we make simplifications to the problem that allows us to solve the resulting simpler optimization problem. This section first describes the simplifications applied to the problem before describing the proposed task and motion planner.

A. Discretization of the problem

We assume that the geometric coordinates for the initial and (when applicable) goal positions of the tractor and trailers are known and given. In order to solve the problem it is likely that additional positions to, e.g., temporarily place a trailer at, will be needed. We will assume that such positions are already given as part of the problem. If not, we propose to use an additional function that adds a number of additional positions to the task-planning problem and samples their corresponding geometric coordinates. In that case, the algorithm can be extended by resampling and increasing the number of positions of the planning fails. This has however not currently been implemented as the main focus of this work is to achieve resolution optimality.

The continuous state space corresponding to the physical states of the platform is discretized as well, which is a common approach in motion planning [15]. All state variables are discretized to finite sets of possible values in order to create a state lattice that can be used by a lattice-based motion planner.

B. The motion planner

A motion planning algorithm is used to solve (3) to resolution optimality. Two separate instances of a lattice-based motion planner are used; one with motion primitives for the combined tractor-trailer vehicle and one with motion primitives for only the tractor. The motion primitives are kinematically feasible and are computed by numerically solving the optimal control problem (3) for a pre-defined set of initial and start states. Each motion planner uses as heuristic a heuristic lookup table (HLUT) [12]. The HLUT stores the optimal cost for a number of pre-computed problems, where the optimal solution from the original to all other states on the state lattice within a certain radius is computed, in an obstacle-free and unlimited environment. The heuristics are consistent and admissible, hence, A* is guaranteed to find a resolution optimal solution to (3) (if such a solution exists). With resolution optimal it is meant that the solution is optimal given the discretization of the search space and the choice of motion primitives.

C. Task planner

The task planner attempts to find a plan in the form of a sequence of actions that solve (1). To do so, it uses the motion planner to solve (3) in order to compute the costs of move-related actions while the costs of actions for (dis)connecting a trailer are already known.

If $c(s,a)$ are known for all states s and all actions a, it is possible to treat the problem as a pure task-planning problem and apply standard task-planning methods. However, the cost for moving between two motion-planning states depends on the task-planning state, as the placement of trailers not involved in the motion will affect the optimal motion since they will be considered obstacles. Computing these costs in advance would therefore often be intractable due to the high number of combinations of possible states and actions.

It is generally computationally expensive to make calls to the motion planner, and in particular many calls. For this reason, it will still be too expensive even if the motion planner is invoked only for those states that are explored during the actual search. To decrease the number of calls to the motion planner, we initially use a computationally inexpensive underestimate of the cost for each move action which is computed by the heuristic function of the motion planner, and improve on this estimate by calling the motion planner only when necessary. This means that the estimated cost of moving between two states may increase monotonically during planning, and even become infinite if it is later discovered that there is no feasible motion plan between two states.

1) LPA and the ComputeShortestPlan procedure:* Due to the fact that costs are initially estimated and may change during planning as improved estimates of the true costs are computed, the task planner is based on LPA* with some minor changes. Pseudocode for the task and motion planner is shown in Algorithm 1 and Algorithm 2. The algorithm relies on the existence of a priority queue $\mathcal{V}$ which sorts nodes n, representing task states, based on their priority $k(n)$ as defined in Section II. $\mathcal{V}$.Top() returns the node with the smallest priority, $\mathcal{V}$.TopKey() returns the smallest priority of any node in the queue, $\mathcal{V}$.Insert(n, k) inserts node n with priority k into the queue, $\mathcal{V}$.Remove(n) removes node n from the queue and $\mathcal{V}$.Update(n, k) changes the priority of node n to k in the queue. Further, $pred(n)$ indicates the predecessors of node n and $succ(n)$ the successors of n.

The pseudocode in Algorithm 1 contains the procedures that are unchanged, or almost unchanged, from the optimized version of LPA* [14]. The only addition is the variables $ub(n)$ and $p_{ub}(n)$ (lines 7-8) that will be explained later in Section IV-C.2. For simplicity, all nodes n in the set of possible nodes N are initialized here in the formal algorithm and $pred(n)$ and $succ(n)$ are used in the same way as in LPA*. In the implementation, nodes are only initialized when they are needed since any node that has not been initialized will have the key $[\infty, \infty]$. The first time $succ(n)$ is called for a node n, the successors are computed by iterating over all a that are applicable in $s(n)$ and finding the nodes that correspond to $\gamma(s(n), a)$. A simple collision check on the motion-planning state corresponding to $\gamma(s(n), a)$ is performed, and if it fails the successor is ignored.

Computing $pred(n)$ would be difficult, so it is stored as a list by each node n. Whenever $succ(n)$ is computed, each node $v \in succ(n)$ adds n to its list of predecessors $pred(v)$. The list $pred(v)$ may therefore not include all possible predecessors, but it will include all predecessors that have a finite key value. The cost of moving from node n to node v is denoted $c(n, v)$ and is identical to $c(s_n, s_v)$ where s_n and s_v are the task-planning states corresponding to n and v respectively. The variable $exactCost(n, v)$ denotes if $c(n, v)$ is exact or an estimate. Another implementation difference is that n_{goal} does not refer to a single goal node (as in LPA*) as there may be more than one possible goal state for the task-planning problem. Instead n_{goal} is a special node that represents the whole set of goal states.

2) The main loop: Algorithm 2 shows how the LPA* procedures are used in the task and motion planner. In the main loop, lines 1-10, ComputeShortestPlan is called to compute the best plan given the current action-cost estimates. This plan is not necessarily a feasible plan since all actions might not correspond to feasible motion plans, and the cost of the plan, $g(n_{goal})$, is an underestimate that may not be the true cost. Thus, the feasibility of the plan and the values of the cost estimates need to be verified. The introduced variable $ub(n)$ represents an upper bound on the true cost-to-come of the state n, and $p_{ub}(n)$ represents the predecessor of the node for which the upper bound is attained. Note that as the node n_{goal} represents the set of all goal states, $ub(n_{goal})$ gives an upper bound on the cost of any solution to the problem.

If $g(n_{goal}) = ub(n_{goal})$ the search can terminate (line 6 in Algorithm 2). If $ub(n_{goal}) < \infty$, the found plan is feasible (since it has a finite upper bound) and no other plan can have a lower cost since ComputeShortestPlan returns the plan with the lowest cost. The resulting path can then be extracted by starting at n_{goal} and for each node n move to $p_{ub}(n)$. Note that $exactCost(p_{ub}(n), n)$ is guaranteed to hold, as otherwise p_{ub} will not have been set, but that $exactCost(p(n), n)$ is not and therefore $p_{ub}(n)$ should be chosen rather than $p(n)$. If instead $g(n_{goal}) < ub(n_{goal})$, the VerifyPlan procedure is called to verify that the plan is feasible and compute the exact cost of the plan.

The VerifyPlan procedure first extracts the plan by calling a function ExtractPlan, which finds all nodes in the plan by starting at n_{goal} and moving from the current node n to $p(n)$ until the start node is reached. The resulting path is then reversed to obtain the actual plan. Starting from the beginning of the plan, for all nodes n where $g(n) < ub(n)$ and the exact cost is not known the motion planner is called to compute the cost of moving from $p(n)$ to n (line 19). The motion planner will return both the cost as well as a value that represents if the cost is exact or only an estimate, where the latter is stored as $exactCost(p(n), n)$. If there is no feasible path, the cost is infinite. If the resulting cost is higher than the previously known cost the same update as in LPA* is performed (lines 20-25). If there is a feasible path, the upper bound of the node is updated by calling the procedure UpdateUpperBound (line 28). Whenever a node achieves a lower upper bound than before, any successors for which the exact cost is known also update their upper bounds.

After the VerifyPlan procedure has terminated, the main loop repeats. If no costs were changed, ComputeShortestPlan will return the same plan again.

3) Pausing and aborting the motion planner: Determining infeasibility is often a time consuming process for the motion planner. To avoid spending too much time on a single motion-planning problem, we propose two different improvements: pausing the motion planner if a certain time limit has been reached, and prematurely aborting the motion planner if the cost exceeds a maximum cost which is computed based on the upper bounds.

We introduce the variable $timeLimit(n, v)$ which denotes the maximum time the motion planner may spend to attempt to find a motion plan from the state corresponding to n to the state corresponding to v. If that limit is reached, the motion planner will return the f-value of the last node it expanded and the search is paused. Since the motion planner uses a consistent and admissible heuristic, the f-values of the nodes it expands are monotonically non-decreasing and an underestimate of the optimal cost. If the motion planner fails to either find a solution or determine that the problem is infeasible, the time limit for that problem is increased for the next time the motion planner attempts that particular problem (line 31). If the internal state of the motion planner is stored, the search can resume, i.e., warm-start, from where it was paused.

Additionally, the second time the motion planner attempts the same problem, we propose to search backward instead of forward. The rationale behind this is that if a problem is infeasible due to obstacles near the goal it can take a lot of time to discover this when searching from the initial state, but

it may be discovered quickly when searching from the goal state. To keep the pseudocode simple the backward search has been omitted from Algorithm 2.

The upper bounds on the cost-to-come are also used to limit the time spent in the motion planner. The motion planner when called from Algorithm 2 approaches the optimal objective function value from below, similar to a dual numerical optimization method. Furthermore, the upper bound is a true upper bound on the optimal objective function value. Hence, performance-increasing ideas known from mixed-integer optimization can be employed, where the solver for the subproblems (here motion planner) can be prematurely aborted as soon as it reaches a known upper bound without losing optimality guarantees [5].

The function MaxCost(n, v), presented in Algorithm 3, returns a maximum value for the cost $c(n, v)$ in order for the edge (n, v) to be part of an optimal plan. If $c(n, v)$ is higher than m_1 computed on line 2 then any plan that contains the edge (n, v) will result in a cost higher than the global upper bound. If $c(n, v)$ is higher than m_2 computed on line 3 then there is another path to v that has a lower cost-to-come. Hence, if the motion planner attempts to compute $c(n, v)$ and extracts a node from its queue with an f-value higher than MaxCost(n, v) the search can be aborted and this f-value returned as the new cost.

4) Heuristic: The heuristic for the task planner utilizes the same heuristic that is used within the tractor-trailer motion planner. It only considers the cost of the move actions required to move the trailers. For any trailer that is not currently at its goal location, the heuristic for the tractor-trailer motion planner is used to compute an underestimate of the cost of moving it to its goal, i.e., its cost-to-go. The heuristic function used by the task planner is the sum of all such cost-to-go estimates. Let $h_{mp}(x, y)$ denote the heuristic cost of moving the tractor-trailer from the motion-planning state x to the motion-planning state y, and let $x_i(n)$ denote the motion-planning state of trailer i in the task-planning state corresponding to node n, assuming that the truck is connected to it. Let T be the set of trailers for which there is a specified location in the task-planning goal. The heuristic is then

$$h(n) = \sum_{t \in T} h_{mp}(x_t(n), x_t(n_{goal})). \tag{4}$$

As will be shown later in Section V, the heuristic in (4) is admissible and consistent.

V. Planner properties

This section establishes some theoretical properties of the planner in Algorithm 2.

Lemma 1: The heuristic $h(n)$ as defined in (4) is consistent and admissible for any cost estimate $c(n, v)$ used by the planner in Algorithm 2.

Proof: Consider any task state s and corresponding node n, and any successor v of n with corresponding task state s'. As $s' = \gamma(s, a)$ for some action a and each action can at most move one trailer, at most one trailer T can have been moved between s and s'. If no trailer in T is moved between s and s' it is clear that $h(n) = h(v)$, and since $c(n, v) \geq 0$ it follows that $h(n) \leq h(v) + c(n, v)$. Suppose

Algorithm 1 Procedures from LPA* used by the task and motion planner

```
1:  procedure CalculateKey(n)
2:      return [min(g(n), rhs(n)) +h(n); min(g(n), rhs(n))]
3:  end procedure
4:  procedure Initialize
5:      V = ∅
6:      for all n ∈ N do
7:          rhs(n) = g(n) = ub(n) = ∞
8:          p(n) = p_ub(n) = NULL
9:      end for
10:     rhs(n_start) = 0
11:     V.Insert(n_start, [h(n_start); 0])
12: end procedure
13: procedure UpdateVertex(n)
14:     if g(n) ≠ rhs(n) AND n ∈ V then
15:         V.Update(n, CalculateKey(n))
16:     else if g(n) ≠ rhs(n) AND n ∉ V then
17:         V.Insert(n, CalculateKey(n))
18:     else if g(n) = rhs(n) AND n ∈ V then
19:         V.Remove(n)
20:     end if
21: end procedure
22: procedure ComputeShortestPlan
23:     while V.TopKey() < CalculateKey(n_goal) OR rhs(n_goal) > g(n_goal) do
24:         n = V.Top()
25:         if g(n) > rhs(n) then
26:             g(n) = rhs(n)
27:             V.Remove(u)
28:             for all n' ∈ succ(n) do
29:                 if rhs(n') > g(n) + c(n, n') then
30:                     p(n') = n
31:                     rhs(n') = g(n) + c(n, n')
32:                     UpdateVertex(n')
33:                 end if
34:             end for
35:         else
36:             g(n) = ∞
37:             for all n' ∈ succ(n) ∪ {n} do
38:                 if n' ≠ n_start AND p(n') = n then
39:                     p(n') = arg min_{v∈pred(n')} (g(v) + c(v, n'))
40:                     rhs(n') = g(p(n')) + c(p(n'), n')
41:                 end if
42:                 UpdateVertex(n)
43:             end for
44:         end if
45:     end while
46: end procedure
```

Algorithm 2 The main loop of the task and motion planner

```
1: procedure MAIN
2:     Initialize()
3:     while TRUE do
4:         ComputeShortestPlan()
5:         if g(n_goal) = ub(n_goal) then
6:             break
7:         end if
8:         VerifyPlan(n_goal)
9:     end while
10: end procedure
11: procedure VERIFYPLAN(n)
12:     plan = ExtractPlan(n)
13:     for i := 0 to length(plan)-1 step 1 do
14:         v = plan[i]
15:         v' = plan[i + 1]
16:         if g(v') = ub(v') OR exactCost(v, v') then
17:             continue
18:         end if
19:         cost, isExact = MotionPlanner(v, v', timeLimit(v, v'), MaxCost(v, v'))
20:         if cost > c(v, v') then
21:             c(v, v') = cost
22:             p(v') = arg min_{n' ∈ pred(v')} (g(n') + c(n', v'))
23:             rhs(v') = g(p(v')) + c(p(n), v')
24:             UpdateVertex(v')
25:         end if
26:         if isExact and cost < ∞ then
27:             exactCost(v, v') = TRUE
28:             UpdateUpperBound(v', v)
29:         else
30:             if NOT isExact then
31:                 increase timeLimit(v, v')
32:             end if
33:             break
34:         end if
35:     end for
36: end procedure
37: procedure UPDATEUPPERBOUND(v, n)
38:     if ub(n) + c(n, v) < ub(v) then
39:         ub(v) = ub(n) + c(n, v)
40:         p_ub(v) = n
41:         for v' ∈ succ(v) do
42:             if exactCost(v, v') then
43:                 UpdateUpperBound(v', v)
44:             end if
45:         end for
46:     end if
47: end procedure
```

Algorithm 3 The MaxCost procedure

```
1: procedure MAXCOST(n, v)
2:     m_1 = ub(n_goal) − h(v) − g(n)
3:     m_2 = ub(v) − g(n)
4:     return min(m_1, m_2)
5: end procedure
```

that trailer $t \in T$ is moved. For all other trailers $\tau \in T$ that are not moved $x_\tau(n) = x_\tau(v)$. By (4), $h(n) - h(v) = h_{mp}(x_t(n), x_t(n_{goal})) - h_{mp}(x_t(v), x_t(n_{goal}))$.

As $h_{mp}(x, y)$ is the cost of moving from x to y when the feasible area is unlimited, it is clear that $h_{mp}(x_t(n), x_t(n_{goal})) \leq h_{mp}(x_t(n), x_t(v)) + h_{mp}(x_t(v), x_t(n_{goal}))$ which gives $h(n) - h(v) \leq h_{mp}(x_t(n), x_t(v)) \leq c(n, v)$. The last inequality holds as the cost estimates are initialized using h_{mp} and can only increase. This proves the consistency of $h(n)$, which implies admissibility. ■

Lemma 2 (Properties of LPA):* Assume that a consistent and non-negative heuristic is used. Then, the ComputeShortestPlan procedure in Algorithm 1 is complete and optimal.

Proof: See [14]. ■

Theorem 3: The task and motion planner in Algorithm 2 is resolution complete, i.e., it will terminate in finite time and, given the discretization of the original problem (placement of possible trailer locations, resolution of the state lattice, choice of motion primitives), it will return a solution if one exists.

Proof: The motion planner is resolution complete and resolution optimal [16]. By Lemma 2 the function ComputeShortestPlan is complete. For each candidate plan produced by ComputeShortestPlan, it will either be found to be a valid solution with the correct cost, or at least one edge cost will be increased. Due to the possibility of pausing the motion planner, it is possible that the motion planner sometimes fails to increase the edge costs, but in that case the same candidate plan is returned immediately again and the motion planner is given more time (line 31 in Algorithm 2). Since the motion planner is complete it will eventually either find a feasible solution or determine that there is none. Hence, for each candidate plan that is not feasible at least one edge cost will be raised. Also, in the worst case scenario, all edge costs will eventually increase to the exact motion costs in which case $g(n_{goal}) = ub(n_{goal})$ when ComputeShortestPlan completes and then Algorithm 2 completes on line 6. ■

Theorem 4: The task and motion planner in Algorithm 2 is resolution optimal, i.e., given the discretization of the original problem (placement of possible trailer locations, resolution of the state lattice, choice of motion primitives) the solution returned by Algorithm 2 is optimal.

Proof: A solution is returned if it has been returned by ComputeShortestPlan and if $g(n_{goal}) = ub(n_{goal})$. Since $g(n_{goal}) = ub(n_{goal})$ the cost is exact. By Lemma 2, ComputeShortestPlan is optimal so no other plan has a lower cost given the current cost estimates. Since the exact costs cannot be lower than the current estimates, all other plans must have an exact cost that is no lower than that of the plan. Hence, it is optimal. ■

VI. NUMERICAL EXPERIMENTS

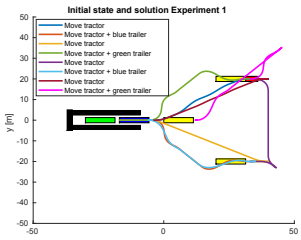
For the numerical experiments we consider the task planning problem described by (1)-(3), with $C = 0.1$. The models used for the tractor-trailer vehicle and the car-like tractor with no trailer connected are the same as in [3]. The cost function used is $l(x, u, q) = 1 + \alpha^2 + 10\omega^2 + u_\omega^2$ where α is the steering angle, $\omega(\sigma) = \alpha'(\sigma)$, and u_ω is the input signal

used to control ω. The first term penalizes the path length and the following terms penalize non-smooth movement. The resolution of the state-lattice grid is $r = 1.0$ m and 16 different discretized values of the heading θ are used. The motion primitives were computed using the CasADi [2] framework. A HLUT for all states such that the absolute difference in x and y is less than 100 m was constructed for each motion planner. The feasible area was constrained to $-50 \leq x, y \leq 50$. The initial time limit for the motion planner was set to 1 s, and doubled every time the motion planner failed to complete within the time limit.

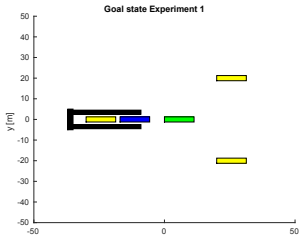
The performance of the proposed task and motion planner is compared to the performance of a baseline resolution optimal task and motion planner where the task planner calls the motion planner to evaluate the exact cost and feasibility during node expansion (similar to the idea of semantic attachments), as well as to the performance of our task and motion planner when no time limit or maximum cost is used for the motion planner. As described in Section IV-C.3, we stress that these limits are enforced in a way that *does not affect optimality*. The task and motion planners are compared on two different examples where the truck has to move trailers to specified places. The setups of the examples are shown in Figure 1. Both examples have been selected to represent challenging problems with non-trivial solutions where trailers cannot immediately be moved to their designated locations due to other trailers blocking the path. Instead, trailers may need to be placed at one or several temporary locations before they can be moved to the correct location. In each example, execution is interrupted after 2400 s if no solution has been found within that time limit.

The resulting plan for the first example is the same for all three methods as they are all resolution optimal. The plan is 15 actions long with a cost of 536.72. The execution times are shown in Table I where it can be seen that our method significantly outperforms the baseline and the version with no time or cost limit for the motion planner. The execution time of our method is about 3 % of the execution time when no limits are used for the motion planner, and less than 1 % of the execution time for the baseline. For all three methods the majority of the execution time is spent in the motion planner. It can also be seen in Table I that allowing the motion planner to pause or abort prematurely leads to 4 additional calls to the ComputeShortestPlan and 4 additional calls to the motion planner. This is reasonable, as pausing the motion planner may result in that the same problem is attempted again. If the green trailer is placed as in Figure 1b it is not possible for a tractor-trailer to move between a position outside of the area surrounded by obstacles to a position within. This takes a lot of effort to discover when searching forward, but is quickly discovered when searching backward which explains the large difference in execution time.

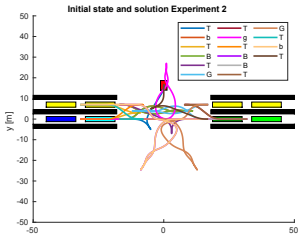
The resulting plan for the second example is 29 actions long with a cost of 1100.58. The baseline had not finished within 2400 s and was aborted without reaching a solution, but both of our methods found the same resolution optimal plan. The execution times are shown in Table II. Both of our versions show similar performance in terms of nodes explored and number of calls to the motion planner. The method with time and cost limits is in this example about 5 %, or about than 3 s, faster. The reason for the similar performance is that no or few motion planning problems were sufficiently difficult for the time limit to make a difference. The number of nodes is much higher than in the first example, as well as the time spent in the task planner. Both our versions first found the solution after about 6 s, and the remaining time was spent verifying that no better solution exists. Hence, this is promising for the ability to compute suboptimal solutions, for which a guaranteed suboptimality level can be computed using $g(n_{goal})$ as a lower bound on the optimal cost.



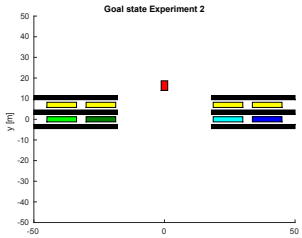
(a) Initial state and resulting trajectory, first example



(b) Goal state, first example



(c) Initial state and resulting trajectory, second example



(d) Goal state, second example

Fig. 1: Experimental setups. Yellow indicates empty positions where a trailer could be placed, red indicates the tractor, and each trailer is shown in a different blue or green shade. Obstacles are shown in black. The absence of a vehicle in the goal indicates that no position is specified for it. The trajectories shown are for the center of the rear axle of the tractor. Note that trajectories may sometimes be partially or completely obscured by overlapping trajectories. In (c), T indicates the tractor whereas B, b, G, g indicate the tractor with the dark blue, the light blue, the dark green and the light green trailer respectively.

VII. CONCLUSIONS AND FUTURE WORK

A. Conclusions

A combined task and motion planner for a tractor and a set of trailers is proposed and shown to be resolution complete and resolution optimal. Several methods to increase the efficiency are proposed including using cost estimates based on the heuristic used in the motion planner, pausing the motion planner if a given time limit has been reached and aborting the motion planner prematurely when maximum cost has been exceeded. These limits are enforced in a way that preserves optimality.

We have numerically evaluated the proposed planner on two different scenarios of practical relevance and shown that the proposed planner can significantly reduce the execution

TABLE I: Results from the first experimental setup

	Baseline	Our (no time/cost limits)	Our
Execution time [s]			
Total	968.53	210.29	6.46
Motion planner	968.51	210.26	6.42
Task planner	0.02	0.03	0.03
Explored nodes			
Including re-exploration	92	645	583
Unique nodes	92	108	108
Calls to motion planner	336	41	45
Calls to ComputeShortest-Plan	1	13	17
Plan cost	536.72	536.72	536.72

TABLE II: Results from the second experimental setup

	Baseline	Our (no time/cost limits)	Our
Execution time [s]			
Total	> 2400	62.05	59.14
Motion planner	> 2396.5	43.51	45.71
Task planner	> 3.73	16.17	15.63
Explored nodes			
Including re-exploration	> 5637	67246	67306
Unique nodes	> 5637	12200	12200
Calls to motion planner	> 37688	339	337
Calls to ComputeShortest-Plan	1	37	39
Plan cost	-	1100.58	1100.58

time compared to a baseline planner that uses the motion planner to compute the exact cost during node expansion. We have also shown on the evaluated examples that the proposed method of pausing and aborting the motion planner prematurely based on time and cost limits can improve the execution time without sacrificing optimality.

B. Future Work

Future work includes extending the planning framework to compute locally optimal task and motion plans by applying an additional numerical optimization step on the obtained task and motion plan in a similar manner as in [3].

Another direction for future work is to improve the scalability of the planner by improving the heuristic as well as to investigate how to efficiently determine when a motion planning problem is infeasible. One possible extension is to make the planner more domain independent and extend a planning language such as Planning Domain Definition Language (PDDL) to include information about the motion planning problem, which could then be used as input to the planner.

REFERENCES

[1] Claudio Altafini, Alberto Speranzon, and Karl Henrik Johansson. Hybrid control of a truck and trailer vehicle. In *Hybrid Systems: Computation and Control: 5th International Workshop, HSCC 2002 Stanford, CA, USA, March 25–27, 2002 Proceedings*, pages 21–34. Springer, 2002.

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

[3] Kristoffer Bergman, Oskar Ljungqvist, and Daniel Axehill. Improved path planning by tightly combining lattice-based path planning and optimal control. *IEEE Transactions on Intelligent Vehicles*, 6(1):57–66, 2020.

[4] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 19, pages 114–121, 2009.

[5] Roger Fletcher and Sven Leyffer. Numerical experience with lower bounds for MIQP branch-and-bound. *SIAM Journal on Optimization*, 8(2):604–616, 1998.

[6] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[7] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. FFrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136, 2018.

[8] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 440–448, 2020.

[9] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[10] 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.

[11] Jennifer E King, Marco Cognetti, and Siddhartha S Srinivasa. Rearrangement planning using object-centric and robot-centric action spaces. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3940–3947. IEEE, 2016.

[12] Ross A Knepper and Alonzo Kelly. High Performance State Lattice Planning Using Heuristic Look-Up Tables. In *IROS*, pages 3375–3380. Citeseer, 2006.

[13] Sven Koenig and Maxim Likhachev. Incremental A*. *Advances in neural information processing systems*, 14, 2001.

[14] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning A*. *Artificial Intelligence*, 155(1-2):93–146, 2004.

[15] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[16] Oskar Ljungqvist, Niclas Evestedt, Marcello Cirillo, Daniel Axehill, and Olov Holmer. Lattice-based motion planning for a general 2-trailer system. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 819–824. IEEE, 2017.

[17] 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.

[18] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 639–646. IEEE, 2014.

[19] Wil Thomason, Marlin P Strub, and Jonathan D Gammell. Task and motion informed trees (TMIT*): Almost-surely asymptotically optimal integrated task and motion planning. *IEEE Robotics and Automation Letters*, 7(4):11370–11377, 2022.

[20] Marc Toussaint. Logic-Geometric Programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936, 2015.

[21] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. Combined task and motion planning for mobile manipulation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 20, pages 254–257, 2010.

[22] Chongjie Zhang and Julie A Shah. Co-optimizing task and motion planning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4750–4756. IEEE, 2016.