A Language For Robot Path Planning in Discrete Environments: The TSP with Boolean Satisfiability Constraints

Frank Imeson Stephen L. Smith

***Abstract*— In this paper we introduce a new language in which discrete path planning problems for mobile robots can be specified and solved. Given an environment represented as a graph and a Boolean variable for each vertex to represent its inclusion/exclusion on the path, we consider the problem of finding the shortest path (or tour) in the graph subject to a Boolean satisfiability (SAT) formula defined over the vertex variables. We call this problem SAT-TSP. We show the expressiveness of this language for specifying complex motion planning objectives in a discrete environment. We then present three solution techniques for this problem, including a novel reduction to the well known travelling salesman problem (TSP). We present extensive simulation results which compare the performance of the three solvers on standard benchmarks from TSP, SAT, and Generalized TSP (GTSP) literature.**

I. INTRODUCTION

A key problem in robotics is in providing a natural and expressive language in which a user can specify a desired task, and from which a planner can compute a robot motion plan. A common approach is to represent the environment as a finite transition system (i.e., a graph) and specify a task in a formal language such as linear temporal logic (LTL). Linear temporal logic contains the usual Boolean operators—and, or, not—along with temporal operators—next, always, eventually, and until. Early work looked at finding a satisfying motion plan for a given task specification [1], [2] and recent work has looked at optimizing over the set of satisfying motion plans [3], [4].

A typical problem with the LTL language is computational complexity—satisfying an LTL formula is PSPACE-complete [5]. To combat this, researchers have proposed to look at fragments of the LTL language, such as general reactivity [2] and the fragment introduced in [4], for which satisfaction and optimization can be performed more efficiently. This highlights the inherent trade-off between expressivity of the language, and the complexity of computing a solution.

An interesting feature of temporal logic based motion planning is that it generates plans over an infinite horizon. This is a result of the temporal operators, "always" and "eventually", which specify logic over infinite time. Thus, any optimization objective must be defined over an infinite horizon. Common objective functions for LTL problems are discounted costs, average time between repeating events, or the worst time between repeating events. Of course, in application, robot plans will be finite in duration.

Many finite path planning problems can be cast as optimization problems on graphs. Finding a shortest path between a start and end vertex can be solved in polynomial time using Dijkstra's algorithm. Finding a shortest path that visits all vertices in a set is known as the travelling salesman problem (TSP), which is **NP**-hard. However, if the graph is metric, then good approximation algorithms exist [6]. Problems that are easily expressed as TSP arise in surveillance and monitoring applications where a robot needs to visit the set of all vantage points [7]. A more general problem is the generalized TSP (GTSP), in which there are several sets of vertices, and the goal is to find the shortest path that visits at least one vertex in each set. The GTSP naturally arises in several applications, including surveillance problems [8], and even certain instances of temporal logic motion planning [4].

In this paper we seek to explore the middle-ground between these two extremes in path planning (i.e., simple graph path planning vs. LTL planning). To this end, we look at finding the shortest path in a graph subject to a set of Boolean constraints on the vertices that indicate their inclusion or exclusion from the path. We call this problem SAT-TSP. In this problem we can no longer express temporal (i.e., ordering) constraints, but the Boolean operators on their own are quite expressive—Boolean satisfiability (SAT) is an **NP**-complete problem, and thus can efficiently express any problem in **NP**.

What is interesting, though, is the notion of expressivity. The decision versions of TSP, SAT, GTSP, and SAT-TSP are all in the same complexity class, **NP**-complete. Thus, they are all, in a theoretical sense, equally expressive – if we had a polynomial-time algorithm for one problem, then we would have a polynomial-time algorithm for all the other problems. However, we believe that expressing complex path planning problems of this nature is more naturally achieved using the SAT-TSP language instead of attempting to encode them with the TSP, GTSP, or SAT language.

The contribution of this paper is follows. We introduce a new language SAT-TSP to allow a user to more "easily" express a class of high-level path planning problems. We demonstrate the expressiveness of this language and provide three methods for solving SAT-TSP instances: 1) an efficient and novel reduction to TSP, 2) a reduction to the constraint satisfaction problem (CSP), and 3) a mixed solver approach which leverages standard SAT and TSP solvers independently. We provide simulation results benchmarking these three approaches and give a coarse classification of the types of problem instances that best suit each solver approach.

II. BACKGROUND

In this paper we present a new language SAT-TSP which is based on the SAT and TSP languages, and we compare our generalization to the GTSP language. To do so we must first provide background on the languages SAT, TSP and GTSP and

This research is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

The authors are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo ON, N2L 3G1 Canada (fcimeson@uwaterloo.ca; stephen.smith@uwaterloo.ca)

provide a means of comparing GTSP to TSP, which we do in the context of expressivity.

A. Languages

The Boolean satisfiability problem SAT is expressed as a Boolean formula which contains literals and operators. A literal is either a Boolean variable (x_i) or its negation ($\neg x_i$). The operators are conjunction ($\wedge$, and), disjunction ($\vee$, or) and negation ($\neg$, not) which operate on the literals and other Boolean formulae. An assignment of the variables (true or false) will now result in the formula being satisfied or not (true or false). The conjunctive normal form (CNF-SAT) is the standard form of SAT in which the formula F has the form $F = c_1 \wedge c_2 \wedge \cdots \wedge c_n$ and each clause $c_i = l_{i.1} \vee l_{i.2} \vee \cdots \vee l_{i.|c_i|}$ is a disjunction of literals. The problem language is defined as follows:

$\textbf{SAT} = \{\langle F \rangle : F \text{ is a satisfiable Boolean formula}\}$.

The travelling salesman problem TSP is traditionally posed as the following: given a list of cities and distances between each pair of cities, what is the shortest possible path that the salesman can take to visit each city exactly once and return to the first city? GTSP is the variation on the TSP where the salesman has to visit at least one city in each set of cities. A tour in a graph that visits each vertex exactly once is called a Hamiltonian tour. The languages can be written as follows:

$\textbf{TSP} = \{\langle G, c \rangle : G = \langle V, E, w \rangle$ is a complete graph with edge weights $w : E \to \mathbb{R}_{\geq 0}$ and G contains a Hamiltonian tour with cost at most $c\}$.

$\textbf{GTSP} = \{\langle G, S, c \rangle : G = \langle V, E, w \rangle$ is a complete and weighted graph, $S = \{S_1, S_2, ..., S_m\}$ where $S_i \subseteq V$ for each $i \in \{1, \ldots, m\}$, G contains a Hamiltonian tour that visits at least one vertex in each set S_i and has cost at most $c\}$.

The optimization versions of TSP and GTSP finds a solution that minimizes c.

B. Expressivity

To compare GTSP to TSP we use a concept known as expressivity. Theoretical expressivity represents the breadth of problems that can be encoded in the language [9], which is the same as the problem's complexity class. However there is an informal notion of expressivity which measures how *practical* the language is to use.

In terms of theoretical expressivity, GTSP is no more or less expressive than TSP but it can be argued that GTSP is more expressive than TSP in the practical sense. This is apparent since the GTSP language can easily express a TSP instance via a trivial reduction[1] and it can also express one in a set TSP problems. Conversely to express GTSP problems as TSP problems we have an entire area of research dedicated to the subject – the Noon-Bean transformation [10] being one of the most widely used. This indicates how non-trivial this task is. We thus concluded that GTSP is more expressive in the practical sense and we will make a similar argument in Section III as to how SAT-TSP is more expressive than GTSP.

[1]For each vertex $v_i \in V$ create a set $S_i = \{v_i\}$ and add it to the group of sets S, yielding a GTSP instance $\langle G, S \rangle$ that solves the TSP problem.

III. PROBLEM STATEMENT

Our goals in this paper are two fold: we study TSP problems with additional constraints of inclusion/exclusion on the vertices in the TSP graph, and we provide the language SAT-TSP which can be used to easily express these instances.

To define SAT-TSP we consider a complete and weighted graph $G = (V, E, w)$, where $V = \{v_1, \ldots, v_n\}$. For each vertex $v_i \in V$ we associate a Boolean variable $x_i \in \{0, 1\}$, when the tour visits v_i the assignment of x_i is one, zero otherwise. We write the corresponding set of Boolean variables as $X(V) = \{x_1, \ldots, x_n\}$. Given a graph G, a SAT formula F defined over $X(V)$ and a possible set of additional Boolean variables (which can be used to define extra constraints), our task is to find the shortest cycle in G that satisfies F. The language is defined as follows:

$\textbf{SAT-TSP} = \{\langle G, F, c \rangle : G = \langle V, E, w \rangle$ is a complete and weighted graph, F is a CNF-SAT formula defined over $X(V)$ along with a possible set of auxiliary variables. Then G contains a Hamiltonian cycle over $V' \subseteq V$ of cost at most c and there exists a satisfying assignment of $X(F)$ such that $X(V') = 1$ and $X(V/V') = 0\}$.

We have defined SAT-TSP as finding a cycle, but we could equivalently define the problem as finding a path.

We claim that SAT-TSP is more expressive than GTSP in the practical sense and thus more general. To show this consider the following trivial reduction from GTSP to SAT-TSP: given a GTSP instance $\langle G, S, c \rangle$ we construct the formula $F = \prod_{j=1}^{|S|} \sum_{v_i \in S_j} x_i$, where each S_j is a set in S of the GTSP instance and now we have a SAT-TSP instance $\langle G, F, c \rangle$[1]. On the other hand, given an instance of SAT-TSP, there does not appear to be a straightforward reduction to GTSP.

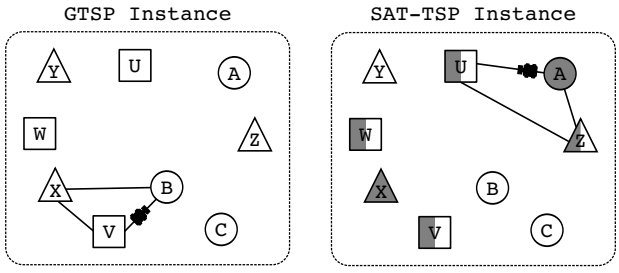

Fig. 1: Visual representation of our GTSP and SAT-TSP example. Shapes represent the different groups, circles for scoop suppliers, squares for motor suppliers and triangles for subframe suppliers. The colour in each shape represents the bracket compatibility, grey for type 1 and white for type 2. The shortest tour is shown for each instance.

To demonstrate the expressiveness of SAT-TSP, consider the following illustrative example.

Example III.1 (Expressivity of SAT-TSP)**.** Let us first consider a GTSP example where a robot has suffered severe damage to its collection unit and needs to retrieve a set of

[1]Summation is used to represent a series of disjunctions $\sum a_i = a_1 \vee a_2 \vee \cdots \vee a_n$, while product is used to represent a series of conjunctions $\prod a_i = a_1 \wedge a_2 \wedge \cdots \wedge a_n$.

parts from different suppliers. The required parts are 1) a collection scoop, 2) a motor, and 3) a subframe. The robot can visit suppliers A, B, C for the scoop, suppliers U, V, W for the motor and X, Y, Z for the subframe. Each supplier has a physical location on a map as shown in Figure 1 and the robot must choose a tour with minimal travel distance to a set of suppliers that retrieves all three parts. This problem has a natural encoding as a GTSP instance but what if the problem contained incompatibilities in the choices?

Consider the senario where the scoop bracket is incompatible with certain subframes: supplier A has scoops with a type 1 bracket, while suppliers B and C have scoops with a type 2 bracket, supplier X supplies a subframe that accepts a type 1 bracket, supplier Y supplies a subframe that accepts a type 2 bracket, and supplier Z supplies a subframe that accepts type 1 or a 2 bracket. It is not obvious how to express these additional constraints into the GTSP language. However, we can easily encode this as a SAT-TSP instance: let G be the graph with edge weights equal to the travel times between suppliers and the formula $F = (A \vee B \vee C) \wedge (U \vee V \vee W) \wedge (X \vee Y \vee Z) \wedge ((A \wedge (X \vee Z)) \vee (\neg A \wedge (Y \vee Z)))$ which we easily translate to CNF-SAT [11] for our language. □

While the above example is simple, it demonstrates that complex constraints consisting of dependencies and incompatibilities can be easily represented in the SAT-TSP language.

IV. APPROACH

In this section we give an overview of our three approaches for solving SAT-TSP instances. In our explanation we use the set of symbols x_i to represent the boolean variables in the SAT formula F, c_j to represent the clauses in the formula, l_k to represent literals in a clause and v_i to represent vertices in the TSP graph G.

A. Approach 1: Reduction of SAT-TSP To TSP

To reduce SAT-TSP to TSP we have constructed a Cook reduction **??** called REDUCE2TSP that translates a SAT-TSP instance into a TSP instance that forces v_s to be the starting vertex. Since we do not know which vertices are included in the optimal tour, we check all n possibilities for v_s, and then return the optimal result. This procedure is shown in Algorithm 1.

Algorithm 1: TSP APPROACH(F, G)

```
⟨Φ_min, c_min⟩ ← ⟨∅, ∞⟩
for v_s ∈ V[G] do
    G' ← REDUCE2TSP(F, G, v_s)
    ⟨Φ, c⟩ ← SOLVETSP(G')
    if c < c_min then
        ⟨Φ_min, c_min⟩ ← ⟨Φ, c⟩
return ⟨Φ_min, c_min⟩
```

The procedure REDUCE2TSP shown in Algorithm 2 constructs a set of "widgets" for each vertex, this allows the TSP language to represent the inclusion or exclusion of a vertex in the SAT-TSP language depending on which direction the widget is traversed as illustrated in Figure 2. Clause vertices are appropriately connected to the widgets to indicate satisfaction or not – if every clause vertex is visited then F is satisfied. The addition of padding vertices between sections of the chain connected to clause vertices ensures that a feasible tour that visits a clause vertex must immediately return to the chain – thus the path only enters and exits the chain from the extremes (not the middle).

The inner connections between widgets are constructed to only allow a tour of the following form: first the start vertex (v_s) is visited, then all included vertices are visited followed by all excluded vertices. The connections and their weights shown in Figure 3 will produce a tour of the same cost as the corresponding SAT-TSP tour. Take special note of the connection cost between included widget y and excluded widget z (bottom left corner of figure), this is the closer cost between the last included vertex and the start vertex.

For conciseness this reduction does not include the widgets for auxiliary variables. However, the widgets for auxiliary variables would be constructed in the same manner, with the inner connections forcing these widgets to be visited at the end of the tour. This design is inspired by the reduction from SAT to the Hamiltonian Cycle Problem [12].

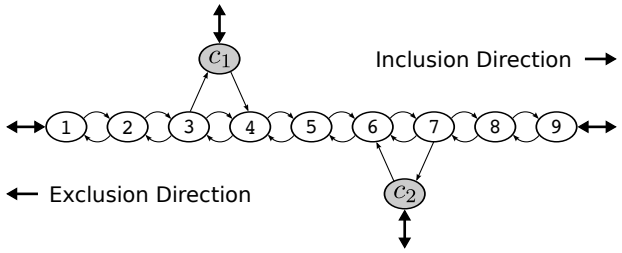

Fig. 2: Above we show an example of a widget for vertex v_i attached to two clause vertices c_1 and c_2. The clause c_1 contains literal x_i and the clause c_2 contains literals $\neg x_i$. A path that traverses the vertices from left to right ($1 \rightarrow 9$) indicates that v_i is included in the SAT-TSP tour.

Remark IV.1 (Creating a Complete Graph). The input to a TSP solver is a complete graph, and thus for any missing edges in our construction, we add edges with infinite costs to the graph.

We now prove that Algorithm 2 is indeed a reduction. We begin with a definition and some useful results.

Definition IV.2 (Feasibility)**.** A feasible TSP solution is a tour that has non-infinite cost.

Lemma IV.3 (Reduction Results)**.** *For the reduction from SAT-TSP to TSP in Algorithm 2 the following hold:*

(i) *Algorithm 2 runs in $O(|V||L| + |V|^2)$ where $|V|$ is the number of vertices in G and $|L|$ is the number of literals in F.*
(ii) *A feasible TSP tour must traverse a chain from one end to the other before visiting another widget.*
(iii) *A feasible TSP tour must visit all included vertex widgets followed by all excluded vertex widgets.*
(iv) *A feasible TSP tour translates to a solution for the SAT-TSP instance.*
(v) *A SAT-TSP solution translates to a feasible tour for the TSP instance.*

Algorithm 2: REDUCE2TSP(F, G, v_s)

1 Create graph H with vertices $\{v_s, v_{c_1}, v_{c_2}, ..., v_{c_m}\}$
2 **for** *each* $v_i \in V[G]/v_s$ **do**
3 $C_i \equiv \{c_j \in F | x_i \in c_j \text{ or } \neg x_i \in c_j\}$
4 Create a widget of length $3|C_i| + 3$
   // Clause vertices connected to the widget are spaced with one vertex in between
5 **for** *each* $c_j \in C$ **do**
6 **if** $x_i \in c_j$ **then**
7 Connect c_j to widget using included edges
8 **else**
9 Connect c_j to widget using excluded edges
   // $I(w_i, in) \equiv$ input vertex of widget i for inclusion, conversely $X(w_i, in)$ represents exclusion
10 **for** *each* $v_i \in V[G]/v_s$ **do**
11 Connect:
12 v_s to $I(w_i, in)$, cost $w_G(v_s, v_i)$
13 $I(w_i, out)$ to v_s, cost $w_G(v_i, v_s)$
14 $X(w_i, out)$ to v_s, cost 0
15 **for** *each* $v_j \in V[G]/\{v_s, v_i\}$ **do**
16 Connect:
17 $I(w_i, out)$ to $I(w_j, in)$, cost $w_G(v_i, v_j)$
18 $I(w_i, out)$ to $X(w_j, in)$, cost $w_G(v_i, v_s)$
19 $X(w_i, out)$ to v_s, cost 0
20 **return** H

Proof. We will establish each of the five results in turn.

Proof of (i): In Algorithm 2, lines 3-9 traverses the set of literals L to create a widget, doing at most $O(|L|)$ work. Lines 10-19 populates the inner connections between widgets $O(|V|^2)$. Thus the entire algorithm does at most $O(|V||L| + |V|^2)$ work.

Proof of (ii): Refer to Figure 2 for a concrete example: this widget only has two possible directions ($1 \rightarrow 9$ or $9 \rightarrow 1$) for a tour. Each path starts and ends at opposite extremes of the chain and by considering all possible paths from the vertex before clause c_1 to the vertex after the clause ($\{2, 3, 4, 5\}, \{2, 3, c_1, 4, 5\}, \{5, 4, 3, 2\}$) we see that there does not exists a feasible tour that visits another widget. Since this argument holds for any clause, a tour must traverse the entire widget before visiting another widget.

Proof of (iii): Feasible tours include the starting vertex v_s, there are no connections from v_s to any excluded vertices and there are no connections from any excluded vertices to any included vertices. Thus all tours must start with v_s and visit all included vertices before excluded vertices.

Proof of (iv): The traversal direction of the chain indicates the inclusion or exclusion of the vertex in the tour. Since the chain must be traversed from one end to the other, the only clause vertices that can be visited are those that are satisfied by the direction chosen (inclusion or exclusion) and since a feasible tour must visit all clause vertices, then the corresponding SAT instance must be satisfied.

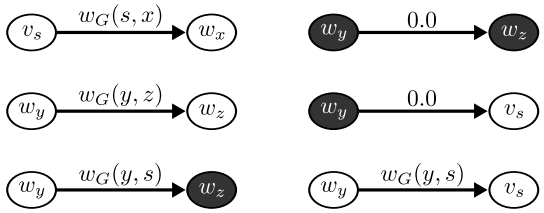

Fig. 3: This diagram is a legend of the different edge weights connecting widgets to widgets and widgets to the start vertex. A widget w_y that is white in colour indicates inclusion of the vertex y and black represents exclusion. The function $w_G(x, y)$ maps edge tuples to real valued weights found in graph G of the SAT-TSP instance.

Proof of (v): Construct a tour for the TSP instance to visit all included vertices followed by excluded vertices (arbitrary order). For each clause vertex, find a pair of edges that the tour can deviate from to visit the clause vertex. This deviation must exist since there is at least one true literal in each clause. □

Theorem IV.4. *(Reduction) Algorithm 2 is a reduction from SAT-TSP to TSP with vertex v_s assumed to be in the tour. Specifically the algorithm runs in polynomial time and there exists a solution to the SAT-TSP instance if and only if there is a solution to the TSP instance. Moreover, the TSP and SAT-TSP solutions have equivalent costs and thus have the same minimum and maximum solutions.*

Proof. The proof has two parts, the first part follows directly from Lemma IV.3. The second part follows from the fact that a feasible tour visits all included vertices followed by excluded vertices and the edge weights between included widgets $\langle w_i, w_j \rangle$ are the same as the edge weights between vertices $\langle v_i, v_j \rangle$ in the SAT-TSP instance. The edge weight of the last included vertex v_l to any other vertex (including v_s) is $w_G(v_l, v_s)$. Thus the set of non-zero edges weights are equivalent which implies the costs are equivalent. □

B. Approach 2: Reduction of SAT-TSP to the Constraint Satisfaction Problem

To reduce SAT-TSP to the constraint satisfaction problem (CSP) [13] we have constructed a Cook reduction that translates the SAT-TSP instance into a CSP instance with a fixed tour length l, i.e., solutions must have l vertices in the tour. Since we do not know the length of the optimal tour, we compare all $|V[G]|$ possibilities and return the best solution. This reduction uses techniques from [13] to translate the SAT formula and basic techniques to translate the TSP instance and combine the two; as such we omit the details here.

C. Approach 3: MIXED Approach

A naive approach to solving SAT-TSP instances is to solve the (SAT and TSP) instances separately. We have implemented this approach by first enumerating all solutions of the SAT formula, then for each solution constructing a sub-graph $G' \subseteq G$ of the included vertices, and finally computing the TSP tour on the subgraph. We call this approach the MIXED approach. This is a naive approach since there may

exist an exponential number of solutions to the SAT formula. However, for many instances in which the SAT formula has few solutions, this approach will perform well, and so we use it as a comparison for the other two approaches.

V. RESULTS

In this section we show simulation results for four types of instances: 1) EASYSAT-HARDTSP, which uses TSP instances from TSPLIB [14] and constructs the SAT formula to be $F = \prod_{i \in |V[G]|} x_i$; 2) HARDSAT-EASYTSP, which uses SAT instances from SATLIB [15] and constructs a complete graph to have all the same edge weights $w(v_i, v_j) = 1$; 3) HARDSAT-HARDTSP, which uses SAT instances from the SATLIB library and constructs a subgraph from a TSP instance in TSPLIB, where all vertices with index labels larger then the number of variables in F are excluded; and 4) GTSP instances obtained from GTSPLIB [16] are reduced to SAT-TSP using the reduction from Section II-B. We use the LKH solver for TSP instances, the MINISAT solver for SAT instances, and GeCode for CSP instances.

A. Optimality

Given enough time, each approach would yield the optimal answer. However due to time limitations we cap the solver time of each sub-instance to 60 seconds. Due to this time restriction the results are not always found and when they are, they may be sub-optimal. We have used data from all three approaches as well as the data from the GTSPLIB to compile the best know results.

In our simulations we have observed that a solution from the TSP or the MIXED approach is almost always optimal but the CSP approach will produce sub-optimal solutions for EASYSAT-HARDTSP and HARDSAT-HARDTSP instances.

We also note that no one approach is able to find solutions for all of our libraries as shown in Table I. In fact the reduction to TSP seems to produce difficultly for the TSP solver even when the SAT formula is easy, the CSP solver does not seem to be very effective at finding an optimal tour for TSP problems and the mixed approach does not work on GTSP instances since there are often exponential number of solutions to enumerate.

The positive results are that the CSP approach seems to work well on HARDSAT-EASYTSP and GTSPLIB instances. For hard SAT formulae—formulae with a few or no solutions—the mixed approach works very well, motivating the question, might there exist an efficient mixed approach?

Approach	EASYSAT HARDTSP	HARDSAT EASYTSP	HARDSAT HARDTSP	GTSPLIB
TSP	34%	0%	0%	43%
CSP	4%	71%	17%	90%
MIXED	100%	100%	99%	0%

TABLE I: This table presents the percentage of instances that were able to obtain the best known solution for the respective approaches .

B. Efficiency

Our first comparison seeks to study how well the TSP approach handles EASYSAT-HARDTSP instances. Ideally the solver performance would compare to that of the MIXED approach since the added complexity is trivial. However due to the Cook reduction the TSP approach solves $|V[G]|$ instances and so for a fair comparison the results shown in Figure 4 are normalized. With this normalization we see that this approach is often $10\times$ slower than the MIXED approach.

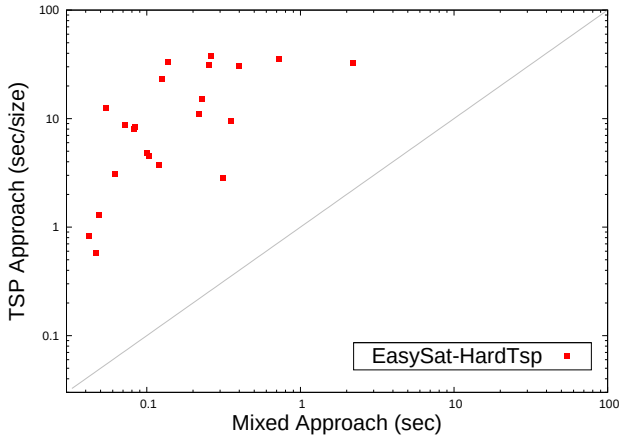

Fig. 4: Comparison of the (normalized) time taken for the TSP approach with respect to the time taken for the MIXED approach on the set of EASYSAT-HARDTSP instances. Each data point shows one problem instance in the EASYSAT-HARDTSP library.

We conjecture that the loss of efficiency is due to the loss of the metric property of the TSP instances (most TSP instances in TSPLIB are metric and the LKH solver works well with metric instances). Unfortunately the reduction from SAT-TSP to TSP does not preserve the metric property even for trivial SAT instances.

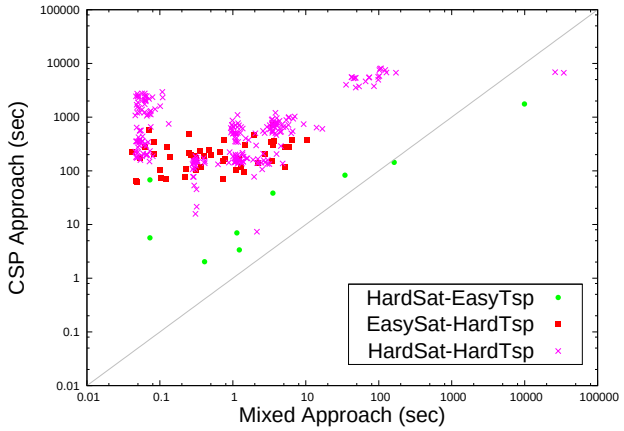

Fig. 5: Comparison of the time taken (including reduction) for the CSP approach with respect to the time taken for the MIXED approach on the set of input instances: EASYSAT-HARDTSP, HARDSAT-EASYTSP and SATLIB. Each data point corresponds to one problem instance.

Figure 5 compares the CSP approach to the MIXED approach. The CSP approach is slower on most of our instances by many orders of magnitude when compared to the MIXED approach, but it is comparable on the HARDSAT-EASYTSP, which is somewhat expected as CSP solvers are known to be competitive with SAT solvers [13]. We also note that our CSP approach is not implementing heuristics that take advantage of the metric property found in many TSP instances and so

we are not surprised that a native TSP solver (MIXED approach) greatly outperforms this approach on the EASYSAT-HARDTSP.

VI. DISCUSSION

Given a SAT solver and a TSP solver, there is a subset of instances SAT* $\subseteq$ SAT and TSP* $\subseteq$ TSP which can be solved efficiently[2]. We have observed that the majority of our instances from the SATLIB and TSPLIB library are in SAT* and TSP*. This implies that our reduction from SAT-TSP to TSP does not in general map into TSP* instances. However since the MIXED approach does not translate the TSP instance, we expect it to be handled efficiently, which is what we observed. Recall that the MIXED approach is not in general efficient and so we wonder if there is an efficient implementation that avoids translating the instances and hence avoids the possibility of mapping an instance in SAT-TSP* to an instance in TSP $\setminus$ TSP*. The answer to this question is no, as shown in the following theorem, but first we need two definitions. A sub-instance of a SAT instance F is any SAT instance F' obtained by fixing an assignment to a sub-set of variables in F. A sub-instance of a TSP instance $\langle G, c\rangle$ is any instance $\langle G', c'\rangle$ where G' is a complete subgraph of G.

Theorem VI.1. *The decision version of SAT-TSP is **NP**-complete even if for each SAT-TSP instance we have access to oracles for the SAT instance, the TSP instance or any of the SAT or TSP sub-instances.*

This result states that unless **P = NP**, we have no hope of finding an efficient algorithm that does not include a translation of either the SAT instance and or the TSP instance. We now prove this result:

Proof. To prove the above result let us reduce a known **NP**-complete problem to SAT-TSP, namely SET-COVER.

SET-COVER $= \{\langle S, k\rangle : S$ is a collection of subsets of elements from a finite universe U for which there exists a subset $S' \subset S$ which covers all the elements in U and $|S'| \leq k\}$.

The reduction maps the sets in SET-COVER to vertices in a complete graph G, in which all edges have a weight of 1. The inclusion/exclusion of a set S_i is indicated by a variable x_i, which is used to ensure that each element u_j is covered. This is accomplished with the formula $F = \prod \sum_{u_j \in S_i} x_i$. Now a minimal solution to the SAT-TSP instance $\langle G, F\rangle$ has a tour of length l which is also the minimum number of sets needed to cover all the elements U.

The TSP instance and sub-instances have trivial solutions (an arbitrarily ordering of the vertices). The SAT instance or a sub-instance has a trivial solution, which is to arbitrarily pick an unassigned variable to be true in each clause (there are no negative literals). Both SAT and TSP instances are solved in linear time which is the same time that it would take an oracle to read off the solutions. Thus in this case, it is as if we have oracles for our instances and sub-instances, and since the SET-COVER problem is **NP**-hard, it must be that SAT-TSP is **NP**-hard even when we have oracles for our SAT and TSP instances (or any sub-instance). Since SAT-TSP is in **NP** then SAT-TSP is **NP**-complete. □

[2]For this argument, we adopt a crude classification of efficiency to be the set of instances that are on average solved in less then one second.

VII. CONCLUSION

We introduced a new problem language SAT-TSP which enables us to easily express discrete robotic path planning problems with complex constraints. We provided three approaches and evaluated them on different types of instances through simulation.

We plan to continue our development of solver(s) for SAT-TSP by use of reductions (to TSP, SAT and CSP) to better understand what kind of millage we can get from existing solvers before turning to a custom solver. We are particularly interested to know if we can construct an efficient algorithm that reduces SAT-TSP* to TSP* and or SAT* instances. We are also interested in exploring approximation algorithms that do not require a translation of the sub-problems (SAT and TSP).

REFERENCES

[1] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *Robotics, IEEE Transactions on*, vol. 21, no. 5, pp. 864–874, 2005.

[2] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *Robotics, IEEE Transactions on*, vol. 25, no. 6, pp. 1370–1381, 2009.

[3] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[4] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimal control of non-deterministic systems for a computationally efficient fragment of temporal logic," in *IEEE Conf. on Decision and Control*, 2013.

[5] A. P. Sistla and E. M. Clarke, "The complexity of propositional linear temporal logics," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 733–749, 1985.

[6] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 4th ed., ser. Algorithmics and Combinatorics. Springer, 2007, vol. 21.

[7] Y. Chevaleyre, "Theoretical analysis of the multi-agent patrolling problem," in *IEEE/WIC/ACM Int. Conf. Intelligent Agent Technology*, Beijing, China, Sep. 2004, pp. 302–308.

[8] K. J. Obermeyer, P. Oberlin, and S. Darbha, "Sampling-based path planning for a visual reconnaissance unmanned air vehicle," *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 2, pp. 619–631, 2012.

[9] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, "Complexity and expressive power of logic programming," *ACM Computing Surveys (CSUR)*, vol. 33, no. 3, pp. 374–425, 2001.

[10] C. Noon and J. Bean, "An efficient transformation of the generalized traveling salesman problem," *Ann Arbor*, vol. 1001, pp. 48 109–2117, 1989.

[11] M. N. Velev, "Efficient translation of boolean formulas to cnf in formal verification of microprocessors," in *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*. IEEE Press, 2004, pp. 310–315.

[12] J. Kleinberg and É. Tardos, *Algorithm design*. Addison-Wesley, 2006.

[13] T. Walsh, "Sat v csp," in *Principles and Practice of Constraint Programming–CP 2000*. Springer, 2000, pp. 441–456.

[14] G. Reinelt, "Tspliba traveling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.

[15] H. H. Hoos and T. Stützle, "Satlib–the satisfiability library," *Web site at: http://www. satlib. org*, 1998.

[16] G. Gutin and D. Karapetyan, "A memetic algorithm for the generalized traveling salesman problem," *Natural Computing*, vol. 9, no. 1, pp. 47–60, 2010.