Multi-Robot Task Planning and Sequencing using the SAT-TSP Language

Frank Imeson Stephen L. Smith

***Abstract*— The SAT-TSP language was recently proposed [1] for expressing and solving high-level robotic path planning problems. In this paper we show how different constraints that commonly appear in path planning problems, such as set constraints, counting constraints, and ordering constraints can all be expressed in the SAT-TSP language. We also show how the language can be used to express multi-robot path planning problems. We evaluate our existing solver approaches on test problems that include a variety of complex constraints and we demonstrate the language through a ROS implementation. We also provide a new approach that reduces the SAT-TSP language to the generalized traveling salesman problem language. We show that this new approach outperforms our existing approaches on problems that contain one-on-a-set constraints.**

I. Introduction

High-level path planning arises in many robotic applications, from surveillance and monitoring for security and law enforcement, to pickup and delivery problems in automated warehousing. Much of the difficulty arises in finding a language in which the problems can be specified, and for which an algorithm exists to compute optimal (or near optimal) paths for the robot. Researchers often leverage a set of common languages for their high-level path planning problems, such as the traveling salesman problem (TSP), the generalized traveling salesman problem (GTSP) and Linear Temporal Logic (LTL).

The TSP language is commonly used to minimize the path length through a set of waypoints, such as reducing the amount of motion of a robotic arm [2], minimizing the path to search an area [3], or patrolling an environment [4]. The GTSP language is often used to express one-in-a-set problems where, for example, pictures need to be taken of several different objects, and each object can be viewed from one of several vantage points [5]. The LTL language has traditionally been used for general tasks in which there are constraints on the time-evolution of the robot [6], [7]. More recent work has used LTL to specify optimization problems that also include time-evolution constraints [8]. In addition, the generalized TSP has been leveraged to solve some fragment problems of the LTL language [9].

The practical design goals for a new language is to find a balance between the expressivity of the language, i.e., what problems can be easily expressed, and the computational efficiency of the solvers, i.e., what problems can be efficiently solved. In [1] we introduced the new language SAT-TSP to allow for the natural expression of logic and transition costs — both commonly needed for path planning problems. The language does this by taking as input a graph G, a Boolean formula F and a budget c. The input instance is satisfiable if there exists a tour on the graph G of cost less than or equal to c such that a vertex is included in the tour if and only it's corresponding variable in the solution to F is assigned true. The SAT-TSP language is expressive in the practical sense, because it is easy to express logic and transition costs. However, one may be unfamiliar with how to express logic constraints in the SAT language or how the combination of the SAT formula and the TSP graph can be used to express more complex problems. In this paper we aim provide a series of examples that demonstrate the expressivity of the language by showing how common path planning problems and constraints can be expressed in the SAT-TSP language.

We also introduce a new approach to better handle GTSP instances. In [1] we have shown that all our existing solver approaches have difficulty on GTSP problems. However, these problems occur frequently in monitoring/surveillance applications [5], [9] and in problems where vehicles have non-trivial dynamics [10]. Our new approach consists of reducing the SAT-TSP language to the GTSP language, which we show performs well on GTSP instances and continues to perform well even when the GTSP instance has some additional SAT constraints.

The *contributions* of this paper are as follows. We provide a new reduction of the SAT-TSP language to the GTSP language and evaluate the performance of this approach. We also provide a series of application examples that demonstrate how to express robotic path planning problems in the SAT-TSP language. Specifically, we demonstrate expressions of set constraints, counting constraints, ordering constraints and multi-robot path planning problems. We also provide a video demonstration of our ROS implementation on a subset of these applications using Gazebo [11].

II. Background

In this section we review the previous work [1] and some background concepts. The concepts reviewed include the SAT, TSP, GTSP and SAT-TSP languages.

The Boolean satisfiability problem SAT is expressed as a Boolean formula that contains literals and operators. A literal is either a Boolean variable (x_i) or its negation ($\neg x_i$). The operators conjunction ($\wedge$, and), disjunction ($\vee$, or) and negation ($\neg$, not) operate on the literals and other Boolean formulae. An assignment of the variables (true or false) results in the formula being satisfied (true) or not (false).

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

The traveling salesman problem (TSP) is traditionally posed as the following problem: given a set of cities and distances between each pair of cities, find the shortest possible path that the salesman can take to visit each city

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)

exactly once and return to the first city. The generalized traveling salesman problem (GTSP) is an extension of TSP to allow for sets of cities such that a solution only visits one city in each set.

TSP $= \{\langle G, c\rangle : G = \langle V, E, w\rangle$ is a complete graph with edge weights $w : E \to \mathbb{R}_{\geq 0}$. Then G contains a cycle that visits every vertex exactly once with cost at most $c\}$.

GTSP $= \{\langle G, S, c\rangle : G = \langle V, E, w\rangle$ is a complete and weighted graph, $V = S_1 \cup S_2 \cup ... \cup S_m$ and $S_i \cap S_j = \emptyset$ for every pair $S_i, S_j \in S$. Then G contains a cycle that visits exactly one vertex in each set S_i and has cost at most $c\}$.

We introduced the language SAT-TSP in [1]. It is a combination of the SAT and TSP language, such that the subset of the variables $\{x_1, x_2, \ldots, x_{|V|}\}$ in the SAT instance represent the inclusion/exclusion of vertices in the TSP tour.

SAT-TSP $= \{\langle G, F, c\rangle : G = \langle V, E, w\rangle$ is a complete and weighted graph, F is a Boolean formula with variables X such that $|X| \geq |V|$. Then there exists a satisfying assignment of X and G contains a cycle of $\{v_i \in V | x_i = 1\}$ with cost at most $c\}$.

III. A GTSP APPROACH FOR SOLVING SAT-TSP

In this section we present the translation of a SAT-TSP instance $\langle G, F, c, v_s\rangle$ to a GTSP instance $\langle G', S'\rangle$, where $v_s \in V$ is assumed to be in the SAT-TSP solution. Each assumption for $v_s \in V$ is tested, until either a solution is found or an unsatisfiable result is returned. The translation reduces clauses and variable assignments to sets. A graph G' is also created to accommodate the sets and mimic transitions in G. In the graph G' we create two vertices v_i^T, v_i^F for each variable $x_i \in X$, which we refer to as the root vertices. We also create a vertex for every literal in the formula, specifically the vertex $v_{i,a}^T$ refers to the literal x_i in clause a and $v_{i,b}^F$ refers to the literal $\neg x_i$ in clause b. These vertices are referred to as the vertex literals and we use them to encode each clause c_a as a set $\{v_{i,a}^\delta \in V'\}$ for $\delta \in \{T, F\}$ and fixed a. We also have a set $\{v_i^T, v_i^F\}$ for each variable $x_i \in X$ to ensure either a true or false assignment.

Before we continue with the construction of graph G', let us create to two sets of vertices $V'_\alpha = \{v_k^T \in V' | v_k \in V\}$ and $V'_\beta = \{v_k^\delta \in V' | v_k^\delta \notin V'_\alpha\}$ for $\delta \in \{T, F\}$, to help us describe how the root vertices are connected. We use these sets to explicitly describe how the root vertices pertaining to included vertices V'_α in G' are connected as V is in G and that the remaining root vertices V'_β are fully connected with zero weight edges. Then these two sets are connected in such a way to allow for the solution tour to close with the same cost as it would in the SAT-TSP instance. Figure 1 shows a small example of the connections between root vertices and the following list explicitly enumerates the details of these connections:

1) $\langle v_i^T, v_j^T\rangle \in E'$ with weight $w(v_i, v_j)$ for $v_i^T, v_j^T \in V'_\alpha$ if and only if $\langle v_i, v_j\rangle \in E$
2) $\langle v_i^\delta, v_j^\theta\rangle \in E'$ with weight 0 for $v_i^\delta \in V'_\beta,\ v_j^\theta \in V'_\beta \setminus v_i^\delta$
3) $\langle v_i^T, v_j^\delta\rangle \in E'$ with weight $w(v_i, v_s)$ for $v_i^T \in V'_\alpha,\ v_j^\delta \in V'_\beta$
4) $\langle v_i^\delta, v_s^T\rangle \in E'$ with weight zero for $v_i^\delta \in V'_\beta,\ v_s^T \in V'_\alpha$

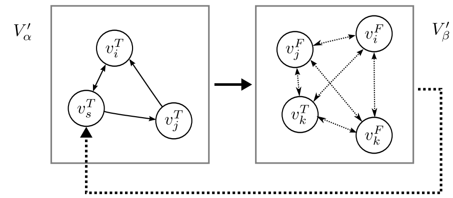


Fig. 1: This figure shows an example of the connections between the root vertices in the GTSP graph instance. Every vertex $v_i^T \in V'_\alpha$ connects to every vertex in $v_j^\delta \in V'_\beta$ with edge weight $w(v_i, v_s)$ and every $v_i^\delta \in V'_\beta$ connects to vertex $v_s^T \in V'_\alpha$ with zero edge weight. The dotted edges all have zero edge weights.

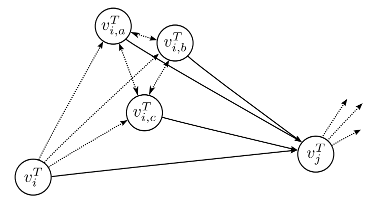


Fig. 2: This figure shows an example of the connections between the root vertex and it's literals. The vertices connected with dotted edges have zero edge weight and the solid edges have weight $w(v_i^T, v_j^T)$.

Each vertex root v_i^δ is connected to it's vertex literals $v_{i,a}^\delta$ with zero weight edges and that group of vertex literals is fully connected to each other with zero weight edges. Furthermore a vertex literal is connected to other root vertices if and only if it's root vertex is connected. Figure 2 shows a small example of the vertex literal connections and the following list explicitly enumerates these connections:

1) $\langle v_i^\delta, v_{i,a}^\delta\rangle \in E'$ with weight 0 for $v_i^\delta, v_{i,a}^\delta \in V'$
2) $\left\langle v_{i,a}^\delta, v_{i,b}^\delta\right\rangle \in E'$ with weight 0 for $v_{i,a}^\delta \in V',\ v_{i,b}^\delta \in V' \setminus v_{i,a}^\delta$
3) $\langle v_{i,a}^\delta, v_j^\theta\rangle \in E'$ with weight $w'(v_i^\delta, v_j^\theta)$ for $v_{i,a}^\delta \in V',\ v_j^\theta \in V' \setminus v_i^\delta$ if and only if $\langle v_i^\delta, v_j^\theta\rangle \in E'$

This construction ensures that a vertex literal $v_{i,a}^\delta$ can only be visited if the vertex v_i^δ is first visited.

In the special case that the set of literal vertices $\{v_{i,a}^\delta \in V'\}$ for fixed i and fixed $\delta \in \{T, F\}$ has cardinally one, then the root vertex can be replaced with the vertex literal. Proper book keeping will be needed to reflect this change.

Lemma III.1. *The following hold for the GTSP translation:*

(i) *The instance is constructed in $O(|X \cup L|^2)$ time, where X is the set of variables in F and L is the set of literals in F.*
(ii) *A GTSP solution translates to a solution for the SAT-TSP instance.*
(iii) *A SAT-TSP solution including v_s in the solution translates to a solution for the GTSP instance.*

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

Proof of (i): The graph G' has $2|X| + |L|$ vertices and thus at most $O(|X \cup L|^2)$ edges. The sets have $2|X| + |L|$ elements. Since the graph and the sets are constructed with no additional calculations the translation requires $O(|X \cup L|^2)$ time to construct.

Proof of (ii): Given a GTSP solution $\zeta = \zeta_1^\delta, \zeta_2^\delta, \zeta_3^\delta$, it translates to a SAT-TSP solution as follows: For every $\zeta_i^\delta = v_j^\delta$, if $\delta = T$ assign $x_j = 1$ otherwise assign $x_j = 0$ (False). Construct the SAT-TSP tour as $\eta = \zeta_1, \zeta_2, \zeta_3, \ldots$ for all $\zeta_i^\delta = v_j^T$ if $v_j \in V$ — preserve the ordering of ζ_i in the tour. This assignment and tour η is a solution to the SAT-TSP instance. Each variable in F only has one assignment since only one vertex in the set $\{v_i^T, v_i^F\}$ can be visited and each clause c is satisfied by this assignment since there is a vertex literal visited in each clause set that corresponds with a true literal in the clause. The cost constraint is satisfied since the GTSP tour has the same cost as the SAT-TSP tour. This is apparent from the construction of G', the cost to transition from an included vertex to an included vertex is the same as transitioning from vertex to vertex in G and the cost to transition from an included vertex to a non-included vertex is the same as transitioning back to the starting vertex.

Proof of (iii): Given a SAT-TSP solution $\zeta = \zeta_1, \zeta_2, \ldots, \zeta_m$ for the tour and a set of assignments for variables $x_i \in F$. The solution translates to GTSP solution as follows: construct the tour of the included root vertices (V'_α) followed by the rest of the root vertices (V'_β). Then traverse the tour and insert visits to vertex literals $v_{i,a}^\delta$ in between root vertices if no other $v_{j,a}^\sigma$ has been visited in the clause set. This tour is a solution to the GTSP problem instance since for each clause there must exist at least one true literal, thus the corresponding set has at least one vertex literal it could visit in each clause set(for no additional cost). Finally the sets $\{v_i^T, v_i^F\}$ are visited only once since every variable x_i only has one assignment. The cost constraint is satisfied since the costs are the same as discussed in (ii). □

Theorem III.2. *This translation is a reduction from SAT-TSP to GTSP with vertex v_s assumed to be in the solution. Specifically, the translation requires polynomial time and there exists a solution to the SAT-TSP instance if and only if there is a solution to the GTSP instance. Moreover, the GTSP and SAT-TSP solutions have the same costs and thus have the same minimum and maximum solutions.*

Proof. The proof follows directly from Lemma III.1 and by the construction of the graph — it has edges with the same costs. □

IV. APPLICATIONS

In this section we demonstrate how to use the SAT-TSP language by expressing a set of example path planning problems in the SAT-TSP language. The SAT-TSP language is comprised of both the SAT and TSP languages and as a rule of thumb the SAT language expresses logic well and the TSP language expresses transition costs well.

In the rest of this section we demonstrate the expression of set constraints, counting constraints, ordered constraints and multi-robot constraints in the SAT-TSP language. All of these constraints can be combined with each other or with any other set of constraint expressible in the SAT-TSP language.

A. Sets Constraints

Many problems often have set constraints. To express set constraints in SAT-TSP we start with an example, suppose we have a set $S = \{e_1, e_2, \ldots, e_n\}$ and we wish to include at least one of these elements in the solution. This is accomplished by adding the clause $(x_{e_1} \vee x_{e_2} \vee \cdots \vee x_{e_n})$ to the formula, where x_{e_i} represents the inclusion/exclusion of e_i in the solution. Suppose instead we wish to include exactly one element from this set. Then the formula $(\bigvee_{i=1}^n (x_{e_i} \bigwedge_{j=1, j \neq i}^n \neg x_{e_j}))$ encodes this constraint, where the notation $\bigvee_{i=1}^n$ represents disjunction of the series of elements over index i and the notation $\bigwedge_{j=1}^n$ represents the conjunction of the series of elements over index j. We may also be interested in creating set constraints that visit at most one in a set (possibly none). This is done by adding the clause: $(\neg x_{e_1} \vee \neg x_{e_2} \vee \cdots \vee \neg x_{e_n})$ to the exactly one in a set constraint formula. The at least one in a set constraint yields a formula of size $O(n)$ and the exactly one in a set or at most one in a set constraint has size $O(n^2)$.

We can introduce additional variables to create more sophisticated yet efficient set constraints. For example suppose we have the set of items $\{o_1, o_2, \ldots, o_6\}$, where the items ($o_1$ and o_2) are square in shape, (o_3 and o_4) are cylindrical, (o_5 and o_6) are spherical, (o_1, o_3 and o_5) are red in color and (o_2, o_4 and o_6) are green. Then we may wish for the robot to collect one of each shape, all of the same color. To express this problem we first create two indicator variables x_r and x_g to represent the collection of at least one red and at least one green item respectively. Then the following set of clauses encode the above constraints. The clauses $(x_{o_1} \vee x_{o_2}) \wedge (x_{o_3} \vee x_{o_4}) \wedge (x_{o_5} \vee x_{o_6})$ constrains the solution to contain at least one of each shape, the clauses $(x_r \vee x_g) \wedge ((x_r \wedge \neg x_g) \vee (\neg x_r \wedge x_g))$ restricts the choice of color to either red or green (exactly one in a set) and the clauses $((x_{o_1} \vee x_{o_3} \vee x_{o_5}) \implies x_r) \wedge ((x_{o_2} \vee x_{o_4} \vee x_{o_6}) \implies x_g)$ constrain the color indicator variables to be true if an item of that color is visited.

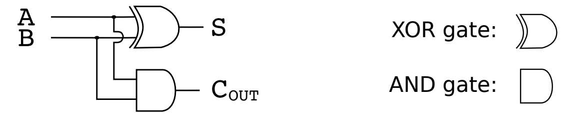


Fig. 3: Adder circuit: Takes Boolean input variables A and B, outputs summation S and the carry out bit C_{OUT}.

B. Counting Constraints

We can also express counting constraints as a Boolean formula. To construct the formula we look to digital circuits for inspiration. Figure 3 shows a digital adder circuit that takes two inputs A and B, adds them together and outputs the sum S and the carry bit C_{OUT}. To construct a larger adder circuit, we chain a series of basic adder circuits together as shown in Figure 4.

Suppose we wish for our robot to collect three items from the set of red items o_1, o_2, o_3, o_4. Then we can construct a circuit to count the number of collected items, Figure 4 shows the summation circuit counting the red items for the least significant digit of the count. The circuit takes as inputs, signals that represents the inclusion (high for included, low for excluded) and outputs the zero bit r_0 used in the binary representation of the count (number of true inputs = $r_0 +$

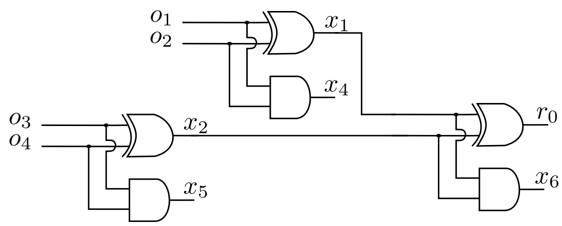


Fig. 4: A sub-circuit of the red counter circuit that represents the the least significant digit of the count.

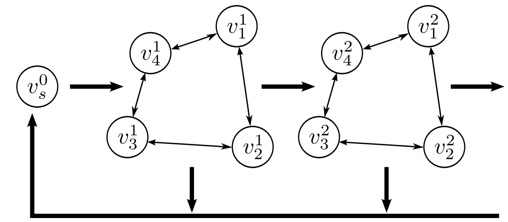


Fig. 5: An example of how levels (copies of G) are connected in the SAT-TSP graph G'. The bold arrows indicate that every vertex connected to the tail end of the arrow is connected to every vertex connected to the head of the arrow.

$2r_1 + 2^2r_2$). The carry out bits x_4, x_5 and x_6 are used as input for the next significant digit summation circuit.

To constrain the red count to be exactly three, the following clauses $(r_0) \wedge (r_1) \wedge (\neg r_2)$ are added to the SAT formula.

Each significant digit in the counting circuit requires $n-1$ adders for n inputs, which adds up to $n(n-1)/2$ adders for the entire circuit. The transformation of the circuit to a SAT formula is linear [12]. Therefore, counting constraints on n variables requires a formula of size $O(n^2)$.

Instead of an exact count, we may have instead wanted to constrain the count to be three or less. To accomplish this we would force the most significant bits to be false (the third bit and up) by adding the following clauses to the formula: $(\neg r_2)$. Or if one wanted a greater than constraint one would count the number of low (false) inputs and negate the higher bits.

C. Ordering Constraints

The SAT-TSP language can also be used to express ordering constraints of the form $v_i < v_j$ (v_i must precede v_j). To do so, the knowledge of a starting vertex v_s is required (first vertex in solution tour), if one does not have this knowledge then a separate reduction can be done for each $v_s \in V$ and the best result is returned.

This method produces a SAT-TSP instance $\langle G', F' \rangle$. The graph G' is constructed with $|V \setminus \{v_s\}|$ copies of the induced subgraph of G with vertices $V \setminus \{v_s\}$. Each copy represents a level and has it's own set of unique labels. The solution can only traverse the levels in sequential order. As such, the vertices in level λ are labeled v_i^λ. Vertex v_s is added to the graph as v_s^0 to represent the starting vertex v_s on level 0. Each adjacent level is connected in increasing order with directed edges and each vertex is also connected back to v_s^0. Figure 5 shows a small example of the connections and the following list explicitly enumerates the details of these connections:

1) $\langle v_i^\lambda, v_j^\lambda \rangle \in E'$ with weight $w(v_i, v_j)$ for $v_i^\lambda, v_j^\lambda \in V'$ if and only if $\langle v_i, v_j \rangle \in E$
2) $\langle v_i^\lambda, v_j^{\lambda+1} \rangle \in E'$ with weight $w(v_i, v_j)$ for $v_i^\lambda, v_j^{\lambda+1} \in V'$ if and only if $\langle v_i, v_j \rangle \in E$
3) $\langle v_i^\lambda, v_s^0 \rangle \in E'$ with weight $w(v_i, v_s)$ for $v_i^\lambda \in V'$ if and only if $\langle v_i, v_s \rangle \in E$

The entire graph has $O(|V|^2)$ vertices and $O(|V||E|)$ edges.

To ensure at most one vertex in the set of copies $\{v_i^1, v_i^2, \ldots, v_i^{|V \setminus \{v_s\}|}\}$ is visited for every $v_i \in V$ we add the at most one constraint to the formula for each set as described in Section IV-A, which contributes $O(|V|^3)$ literals to the formula F'. To ensure the vertex orderings are not violated we add negation clauses to disallow any vertex pair that violates any ordering. For example suppose vertex v_a must precede vertex v_b in the solution, then we negate every combination of v_a^λ and v_b^σ such that $\lambda \geq \sigma$ (negation clause: $\neg(x_{v_a^\lambda} \wedge x_{v_b^\sigma})$). There are $O(|V|^2)$ such negation clauses for each ordering constraint. In the case that we have a precedence constraint for a set of vertices $A = \{v_{a_1}, v_{a_2}, \ldots, v_{a_{|A|}}\}$ and a set $B = \{v_{b_1}, v_{b_2}, \ldots, v_{b_{|B|}}\}$, such that every vertex $v_a \in A$ must precede every vertex $v_b \in B$, then we can use indicator variables to reduce the number of negation clauses from $O(|A||B||V|^2)$ to $O(|V|^2)$.

To demonstrate let us create the set of indicator variables $x_A^\lambda \in \{x_A^0, x_A^1, \ldots, x_A^{|V|}\}$ and $x_B^\lambda \in \{x_B^0, x_B^1, \ldots, x_B^{|V|}\}$ to represent if a vertex in set A or B is visited on level λ respectively. Then we can negate the indicators instead of the vertex pairs to produce only $O(|V|^2)$ negation clauses. This is done with the clauses: $\neg(x_A^\lambda \wedge x_B^\sigma)$ for all $\lambda \geq \sigma$. The indicator variable x_A^1 is constrained with the clauses $(x_{v_{a_1}}^1 \vee x_{v_{a_2}}^1 \vee \ldots \vee x_{v_{a_{|A|}}}^1) \implies x_A^1$ and the other indicators are likewise constrained to add to the formula size $O(|V|^2)$ literals. Therefore using this method keeps the size of the ordered constraint formula down to $O(|V|^2)$.

A solution to the above SAT-TSP expression will satisfy the orderings since all sub-graphs derived from solutions to the SAT formula cannot violate the ordering. The solution also has the equivalent cost since the corresponding transition in G' are equivalent to the original graph G.

D. Multiple Robots

Many multi-robot path planning problems can be expressed in the SAT-TSP language. Specifically, we can express problems consisting of heterogeneous robots with different capabilities. We can also express location-robot conflicts that state which locations must be visited by the same robot or which locations cannot be visited by the same robot. Capability constraints can be as simple as a location v_i can be visited by a robot only if the robot possesses a set of capabilities $a(v_i)$. Alternatively, the requirements of location v_i can be some arbitrarily complex SAT formula of the abilities. The location-robot constraints may also be arbitrarily complex.

Given a problem with $V = \{v_1, v_2, \ldots, v_{|V|}\}$ locations, $R = \{r_1, r_2, \ldots, r_{|R|}\}$ robots, a transition graph G^x, a start and a finish location v_s^x, v_f^x and a set of abilities $a(r^x) \subseteq \{a_1, a_2, \ldots, a_o\}$ for each robot x. We setup the SAT-TSP instance $\langle G', F' \rangle$ by first constructing the graph G' as the union of the robot's transition graphs $G^1 \cup G^2 \cup \ldots \cup G^{|R|}$, with all incoming edges to the set of start locations and all outgoing edges from the set of finish locations removed and then for each robot r^x we connect the finish vertex v_f^x to the next robot's start vertex v_s^{x+1} with zero weight.

The formula F' requires an at-most-one-in-a-set constraint for each set of variables representing the set $\{v_i^1, v_i^2, \ldots, v_i^{|R|}\}$ for each $v_i \in V$ — no two robot's are allowed to visit the same vertex. Then the instance is constrained to start with the the first robot's start vertex and finish with the last robot's finish vertex. Now the instance is setup to to handle the set of ability and location-robot constraints. As an example of a multi-robot path planning problem that can be expressed in this way, refer to Section V where we restrict a class of items (shapes) from the robot's abilities and we restrict visiting more than one location that contains an item of the same color.

V. Simulations

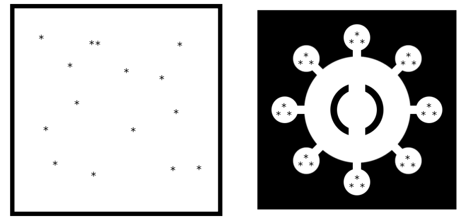
In this section we present a set of simulations to benchmark the GTSP approach against the CSP approach found in [1]. We choose to compare the GTSP approach against the CSP approach because it is our most successful solver on general SAT-TSP instances. In each simulation we have a set of locations for one or more robots to visit. At each location (excluding the home locations) there is an item for the robot to retrieve. This item has both shape and color. There are three possible shapes: cube, ball or cylinder and eight possible colors: red, green, blue.... Both the shape and color aspects are used to help constrain the problems. The robot is then tasked with finding a tour of minimum length that collects a set of items to satisfy the problem constraints. For example, find a minimum length tour that collects one cube, one ball and one cylinder.

In the rest of this section we present four types of path planning problems, the two simulated physical environments used for the problems, and the performance results of the GTSP and CSP approaches on these problems. We have also provided an attachment video that demonstrates two problem solutions in the ROS environment.

A. The Environments

To construct problems that resemble real-world robotic applications we convert the following environments to graphs with edge weights that represent the shortest distance in free space between locations (the geodesic distance) to be used in our simulations.

1) The Unit Square: Shown on the left of Figure 6, the robot is able to move within the two dimensional space $x \in [0, 1]$ and $y \in [0, 1]$. The locations are randomly placed within the square. There is one location dedicated to each robot as home, the rest of the locations are for items. Each item location has an item of random shape and color. This type of environment emulates the types of problem environments that have little to no obstacles for the robot to avoid and little to no predictability of the tasks.



(a) Square Environment (b) Pod Environment

Fig. 6: The two environments used for testing. Each environment represents a two dimensional space. The robot can drive in the white areas, black areas are obstacles. The small dots represent the item locations.

2) The Pod Environment: Shown on the right of Figure 6, there are eight circular "pods", each of which contains three item locations — 24 item locations in total. There is a predetermined set of items: one of each shape/color combination. The set of items are randomly distributed among the 24 item locations and a home location near the center of the environment is added for each robot. This environment is meant to emulate the office or industrial environment where there is a predetermined set of locations and tasks. This environment is also used in our ROS simulation.

B. The Problems

The set of problems we use in our simulations are the set of applications we demonstrated in Section IV. Each problem uses the shape and color information to construct the constraints. The problems are as follows.

1) Sets: The robot must retrieve one item of each shape (cube, ball, cylinder). The set of items the robot brings back must all have the same color.

2) Counting: The robot must retrieve one cube, two balls and three cylinders.

3) Ordered: The robot must retrieve one cube, two balls and three cylinders. All cubes must be visited before balls and all balls must be visited before cylinders.

4) Two Robots: The robots must retrieve one cube, two balls and three cylinders. The first robot cannot pickup up cubes, the second robot cannot pickup balls and no robot can pickup more than one item of the same color.

C. GTSP Simulations

To explore how well the GTSP approach works, we constructed a series of randomly generated GTSP instances with 100 vertices divided into a random number of sets ranging from 10 to 20. The graph is generated from our unit square environment and the GTSP instance is encoded as a SAT-TSP instance. We then add a series of randomly generated negation constraints. The constraints negate pairs of vertices in separate sets (both vertices cannot appear in the solution). The addition of these clauses help us explore how the solver performance degrades (if at all) as the instance transitions from a GTSP instance to a more constrained problem.

D. The Results

The results we show in this section are from the solver approaches GTSP and CSP. We show the cost of the best solutions and the time to find these solutions.

TABLE I: This table presents the time taken for the GTSP and CSP solvers to find the optimal solution in the square environment. Results are averaged over ten runs. The size of the problem indicates how many locations are in the problem and a '-' is used to indicate that no solution was found.

Problem	Size	Time (seconds) GTSP	CSP
Sets	20	48.4	0.572
Sets	50	105	45.2
Counting	20	-	40.1
Counting	50	-	486
Ordered	10	-	42.4
Two Robots	20	48.5	0.806
Two Robots	50	105	39.9

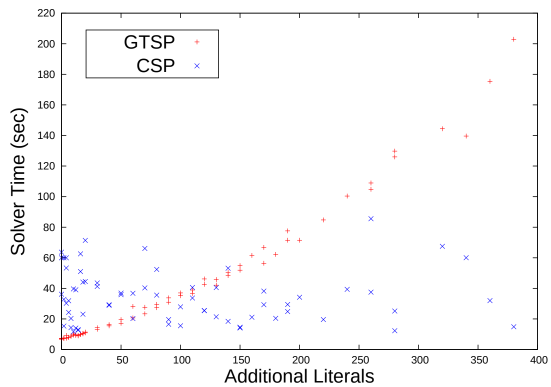


Fig. 7: This graph plots the solver times of the GTSP approach and the CSP approach with respect the number of added literals to the GTSP instance.

The CSP solver we used is Gecode's flatzinc's interpreter [13]. The GTSP solver we use is a custom solver based on large neighborhood search, which is currently being developed by the authors and appears competitive with the state-of-the-art solvers [14], [15].

Table I shows average time of the two approaches on the application problems. As we can see the GTSP approach did not perform as well as the CSP approach on our application problems. Typically, the solver took both more time and yielded a lower quality solution. This is likely due to the fact that the transformation to GTSP produces many zero and infinite cost edges, which is typically challenging for neighborhood search-based solvers. In general neither the GTSP or CSP approach was able to find optimal solutions for the larger ordered instances.

As we can see in Figures 7 and 8 the GTSP approach performs better on problems that initially have some structure similar to a GTSP. These figures compare CSP to GTSP performance for instances that are more or less constrained GTSP instances (more or less additional literals). These figures also show that as the problem becomes more constrained, the benefit of choosing the GTSP approach diminishes and as in the case of our application examples the problem is constrained enough to warrant the choice of the CSP approach over the GTSP approach.

VI. Conclusion

We have provided a series of application examples to help the user understand what types of problems can be expressed in the SAT-TSP language and what types of problems the SAT-TSP solvers handle well. We have also provided a

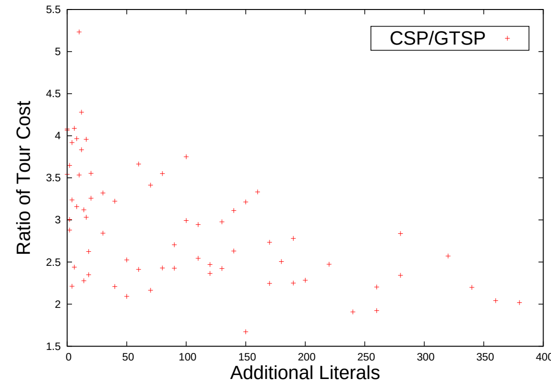


Fig. 8: This graph plots the CSP:GTSP cost ratio with respect to the number of additional literals in the GTSP instance.

new approach to solving SAT-TSP instances that works well on problems that have structure similar to GTSP instances. Specifically, we have shown that when the user has a GTSP instance with some additional constraints, then the GTSP approach is a good place to start.

References

[1] F. Imeson and S. L. Smith, "A language for robot path planning in discrete environments: The tsp with boolean satisfiability constraints," in *Proceedings of the IEEE Conf. on Robotics and Automation*, 2014.

[2] A. Zavichi, K. Madani, P. Xanthopoulos, and A. A. Oloufa, "Enhanced crane operations in construction using service request optimization," *Automation in Construction*, vol. 47, pp. 69–77, 2014.

[3] H. Choset, "Coverage for robotics–a survey of recent results," *Annals of mathematics and artificial intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] J. Le Ny, E. Feron, and E. Frazzoli, "On the dubins traveling salesman problem." *IEEE Transactions on Automatic Control*, vol. 57, no. 1, pp. 265–270, 2012.

[11] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.

[12] 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.

[13] G. Team, "Gecode: Generic constraint development environment, 2006," 2008.

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

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