An SMT-Based Approach to Motion Planning for Multiple Robots With Complex Constraints

Frank Imeson and Stephen L. Smith, *Senior Member, IEEE*

***Abstract*—In this paper, we propose a new method for solving multirobot motion planning problems with complex constraints. We focus on an important class of problems that require an allocation of spatially distributed tasks to robots, along with efficient paths for each robot to visits their task locations. We introduce a framework for solving these problems that naturally couples allocation with path planning. The allocation problem is encoded as a Boolean Satisfiability problem (SAT) and the path planning problem is encoded as a traveling salesman problem (TSP). In addition, the framework can handle complex constraints such as battery life limitations, robot carrying capacities, and robot-task incompatibilities. We propose an algorithm that leverages recent advances in Satisfiability Modulo Theory (SMT) to combine state-of-the-art SAT and TSP solvers. We characterize the correctness of our algorithm and evaluate it in simulation on a series of patrolling, periodic routing, and multirobot sample collection problems. The results show our algorithm significantly outperforms state-of-the-art mathematical programming solvers.**

***Index Terms*—Autonomous agents, path planning for multiple mobile robot systems, scheduling and coordination, surveillance systems.**

I. Introduction

ROBOTS are increasingly being asked to perform complex missions that require the efficient coordination of multiple robots over large and complex physical spaces. In this paper, we focus on motion planning problems where a robot or group of robots are dispatched to complete a set of spatially distributed and interdependent tasks. Such problems arise in a variety of robot applications. In environmental monitoring and patrolling, robots with on-board sensing are tasked with selecting and visiting viewpoints to assess traffic conditions [1], crop health [2], security threats [3], [4], or for general data collection [5]. In exploration tasks, robots are used to observe and potentially collect samples from a previously unexplored environment [6]–[8] (an example sample collection task for two robots is shown in Fig. 1). In flexible manufacturing and material transport, a group of robots must repeatedly visit a set of locations with a specified frequency to move and deliver materials within a factory [9]–[11].

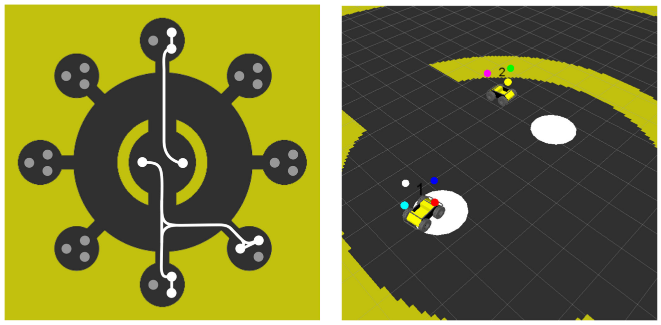

Fig. 1. Robot operating system (ROS) simulation for a sample collection problem. The left image shows the environment, the layout of the samples, and the robot solution paths. The right shows the robots returning home after collecting one of each mineral type from a subset of the samples.

In all of these applications, the common aspects are that tasks must be allocated to the robots and each robot must plan a route to accomplish its set of tasks (note the problems of allocation and routing are typically coupled). Each problem has additional constraints such as battery life, visit frequencies, robot-task incompatibilities, and carrying capacities. The typical approach for solving these problems is to create a roadmap for each robot that captures the environment and its transition costs [12]. This separates the planner into two parts, a high-level planner, which is responsible for optimizing motion plans within the roadmap, and a low-level planner [12]–[14], which tracks the resulting path and avoids collisions. In this paper, we focus on the high-level planner and introduce a common framework in which a broad class of motion planning problems (including their additional constraints) can be expressed and efficiently solved. Additionally, for this paper, we assume collision avoidance between multiple robots is handled by the low-level planner.

Research in high-level motion planning has followed two main approaches. The first is to design a custom algorithm for specific problems. These custom algorithms draw from a broad range of areas including approximation algorithms [2], [15], branch-and-bound techniques [3], and advanced heuristics [4]. The algorithms typically leverage some specific structure in the problem, and thus a drawback is that they are often not portable to related problems in which a subset of the constraints differ. The second approach is to express the problem in a standard form on which a powerful commercial-grade solver can

Manuscript received March 9, 2018; accepted January 20, 2019. This paper was recommended for publication by Associate Editor K. Hauser and Editor A. Billard upon evaluation of the reviewers' comments. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada. *(Corresponding author: Frank Imeson.)*

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

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. This video shows a simulation for the sample collection problem as described in this paper. This material is 8.73 MB in size.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2019.2896401

be used. For example, a commonly used standard is integer linear programming (ILP), for which several advanced and user-friendly solvers exist, including CPLEX [16] and Gurobi [17]. However, a common difficulty in this approach is scalability, since there is no way to leverage problem specific structures such as allocation and routing aspects that are present in multirobot planning problems. It is this observation that we seek to address in our proposed solution methodology.

In this paper, we introduce a framework for expressing and solving high-level motion planning problems. The framework is designed to leverage the problem-specific structure of high-level planning problems, while maintaining the ability to express a wide variety of additional constraints such as interdependencies between tasks, incompatibilities between certain robots and tasks, battery life, and capacity constraints. To this end, we introduce a general problem called SAT-TSP and an algorithm for solving these problems, called cbTSP. The problem takes as input a set of weighted graphs, a Boolean formula, and a cost budget. The graphs are used to represent the motion roadmap for each robot. The Boolean formula is used to express the logical constraints on the robots' motion, such as task dependencies, incompatibilities, and capacity constraints. The cost budget is a constraint on the robots' total transition costs (distance or time). A SAT-TSP solution is a set of paths for each robot that visits a subset of the locations in the environment, satisfy the constraints of the problem, and stay within the cost budget. To solve this problem, we propose an algorithm based on Satisfiability Modulo Theory (SMT). The algorithm leverages a state-of-the-art Boolean Satisfiability (SAT) solver [18] to satisfy the constraints of the problem, and a traveling salesman problem (TSP) solver [19] to route the robots through their assigned tasks. The SMT approach enables these two solvers to coordinate their efforts, with the SAT solver proposing candidate allocations, and the TSP solver evaluating their cost, helping to guide the search.

A. Contributions

The contributions of this paper are as follows. We formally introduce the SAT-TSP problem and propose the novel solver cbTSP. We characterize the complexity of this problem, and show that the proposed algorithm is correct for instances in which the robotic roadmaps are TSP-monotonic (a generalization of metric graphs). We compare cbTSP to a commercial-grade ILP solver on a set of three important motion planning problems: patrolling, sample collection with multiple robots, and periodic routing. Our results show that cbTSP often outperforms the ILP solver. Additionally, we have included a supplementary video, which demonstrates a solution for a multirobot sample collection problem in a high-fidelity ROS [20] simulation.

The SAT-TSP problem was first introduced in our preliminary works [21] and [22]. These papers proposed solving problems expressed in SAT-TSP through a reduction to TSP, a reduction to the generalized version of TSP, and a reduction to the constraint satisfiability problem (CSP). This paper also proposed a preliminary version of our brute force solver as presented in Section IV-A. The contribution of this paper over the preliminary work is to propose a powerful solution methodology based on SMT that far outperforms the methods in this early work. We also prove that SAT-TSP is NP-hard, even when the constraints and the routing aspect of the problem can be efficiently solved independently. Finally, we express three planning problems in SAT-TSP and compare the performance of the proposed solver against a state-of-the-art commercial-grade ILP solver.

B. Organization

The rest of this paper is organized as follows. In the remainder of this section, we review methods for solving high-level planning problems, and the use of SMT-based solvers. In Section II, we cover the necessary background needed for this paper. In Section III, we formally introduce the SAT-TSP problem. In Sections IV and V, we describe the cbTSP solver and our ILP formulation respectively used for the simulations. In Section VI, we detail the three different path planning problems used for our simulations and describe how they are expressed in SAT-TSP and ILP. In Section VII, we detail the simulation setup for each problem and compare the solution quality of cbTSP to the ILP solver. In Section VIII, we conclude this paper and review future work.

C. Related Work

There are a number of methods for solving high-level path planning problems. One of the most common is to use mixed integer linear programming (MILP) or ILP [23]. This framework allows the user to express their problem as a series of linear constraints along with an optimization objective. There are a number of powerful solvers that one can use to find optimal solutions such as CPLEX [16] and Gurobi [17]. In [24], Park *et al.* use MILP to plan optimal trajectories, and in [25] and [26] ILP is used to solve a multirobot charging problems. In [27], Yu and LaValle give an ILP solution for collision-free multirobot planning. In this paper, we provide a direct comparison of our SAT-TSP solver, cbTSP, to the well-known ILP solver, Gurobi, on a similar set of robotic path planning problems and our results show that cbTSP often outperforms the ILP solver.

Another common approach for high-level path planning is to use linear temporal logic (LTL) [28]. The LTL language allows a user to express a problem as a set of state transitions such as "if the robot is at location x then the next location it will visit is y." Solvers developed in the model checking community can then be used to compute *runs* (expressed as an automaton) that satisfy the LTL formula [29], [30]. In [31], LTL is used to express a class of persistent patrolling problems, and Smith *et al.* propose a method for computing optimal plans rather than just feasible plans. In [32], LTL is used to express multirobot planning problems. A drawback of LTL is that the problem of determining if an LTL formula is satisfiable is in the complexity class PSpace-complete [33], while we show that deciding if a SAT-TSP problem is satisfiable (the decision version of SAT-TSP) is in the complexity class NP-complete (see Section IV). The LTL language is likely more expressive than SAT-TSP, since it is believed that PSpace is a strict superset of NP. For example, LTL can be used to express planning problems that consist of infinite length solution paths (i.e., persistent problems), where SAT-TSP cannot. However, many important path planning problems lie in NP, and thus our goal in this paper is to produce a solver that is tailored to problems in this class.

Other approaches for high-level planning include the stanford research institute problem solver (STRIPS) problem specification language [34] and its successor planning domain definition language (PDDL) [35]. Like LTL, the language allows a user to express a problem as a set of state transitions, although PDDL is even more expressive than LTL. There are a number of good solvers for these expressions such as the fast-forward planning system (FF) [36] and LAMA [37]. These languages are capable of expressing any problem in EXPSpace [38], which is widely believed to be a strict superset of PSpace and thus NP. In this paper, we focus on problems in NP, and thus ILP (which lies in the same complexity class as SAT-TSP) will serve as our point of comparison.

Our main solver, cbTSP is based on SMT, which is an extension of SAT that allows for first-order logic. SMT solvers have previously been proposed for solving path planning problems. In [39] and [40], the authors solve the high-level and low-level path planning problems simultaneously. This is unlike our approach where we leverage the SMT framework to better solve the high-level problem. In [41]–[43], the authors use an off-the-shelf SMT solver to find solutions for the high-level path planning problem. Our approach differs from these by using a custom SMT theory that specializes in handling the combinatorial nature of sequencing locations. To the best of our knowledge, we are the first to solve high-level path planning problems using a custom SMT theory to handle the combinatorial aspect of sequencing.

The CSP is a further generalization of SMT and thus SAT. Both SMT and CSP are capable of expressing and solving the types of problems proposed by this paper. In [44, Appendix B] we expressed SAT-TSP instances in both SMT and CSP and then solved instances with state-of-the-art SMT and CSP solvers. Both methods performed poorly compared to an ILP solver, and thus we do not use these approaches as a comparison in this paper.

II. Background

In this section, we define SAT and TSP, review SMT and the aspects of the SMT solver approach DPLL(T) used by cbTSP. As well, we introduce some notation for expressing Boolean formulas.

A. SAT, Graphs, and TSP

Decision problems are problems that have a yes or no answer (satisfiable or unsatisfiable). In this paper, we also study optimization problems that aim to minimize the cost of the solution and, for simplicity, we formally define these optimization problems as their decision problem versions.

The Boolean satisfiability problem SAT is expressed as a Boolean formula that contains literals and operators. A literal is either a Boolean variable (x) or its negation ($\neg x$). The operators are conjunction ($\wedge$, and), disjunction ($\vee$, or), and negation ($\neg$, not), which may operate on the literals or other Boolean formulas. An assignment of the variables (true or false) results in the formula being satisfied (true) or not (false). The conjunctive normal form (CNF-SAT) is the canonical form of SAT and a formula F is in its canonical form if the formula is a conjunction of clauses, where each clause is a disjunction of literals. In this paper, we allow for formulas to be expressed in noncanonical form.

Definition II.1 (SAT): Given a SAT Boolean formula F, determine if it is satisfiable.

A graph is expressed as a tuple, $\langle V, E, w\rangle$, where V is the set of vertices, E is the set of edges, and w is the weight function that assigns a cost in $\mathbb{R}_{\geq 0}$ to each edge. Note for this paper, we assume all edges are directed ($\langle v_i, v_j\rangle$ is not the same as $\langle v_j, v_i\rangle$). The next definition is needed for future sections.

Definition II.2 (Induced Subgraph): An induced subgraph of a graph $G = \langle V, E\rangle$ for $V' \subseteq V$ is a graph $G' = \langle V', E'\rangle$ with $E' = \{\langle v_i, v_j\rangle \in E | v_i, v_j \in V'\}$. We say that G' is induced by V'.

A cycle in a graph that visits each vertex exactly once is called a Hamiltonian cycle, or a tour. The TSP is typically defined as finding the shortest tour in a graph, and its decision version is as follows.

Definition II.3 (TSP): Given a complete and weighted graph $\langle V, E, w\rangle$ and a cost budget c, determine if there exists a Hamiltonian cycle with total cost c or less.

The optimization version of TSP minimizes c.

B. SMT and DPLL(T)

The SMT problem is an extension of SAT that additionally allows for the expression of multiple decidable first-order logic problems, such as arithmetic logic or quantifiable Boolean logic. We refer to each additional first-order formal t as a theory and the set of theories as T. Each theory $t \in T$ is linked through the propositional logic formula F (the SAT formula) using a Boolean variable $x_t \in X$, where the value of x_t must agree with the evaluation of t (x_t is true if and only if t evaluates to true). The power of this approach is that a SAT solver can be used to solve the SAT-aspect of the problem, while a set of specialized solvers can be leveraged for each class of theory (specializations of first-order logic).

Definition II.4 (SMT): An SMT formulation, $\langle F, T\rangle$, is satisfiable if and only if

1) F is a propositional formula defined over X;
2) T is a set of decidable first-order logic problems defined over variables Q such that $X \subseteq Q$;
3) there exists an assignment of Q satisfying F such that for every $t \in T$, the corresponding predicate variable x_t agrees with the evaluation of t (true or false).

The DPLL(T) algorithm for solving SMT instances is based on the DPLL algorithm for solving SAT instances (propositional formulas). The DPLL algorithm solves F by building a list of assignments for the Boolean variables in F (a partial solution). The DPLL(T) algorithm extends DPLL to allow for theories by linking the predicates $x_t \in X$ to the theories $t \in T$. Informally, the algorithm works as follows. As a partial solution for F is constructed by the DPLL algorithm, the theory solvers are called to confirm that the partial SAT solution is consistent with the theory instances. If the theory instances are consistent, then the DPLL algorithm continues, if not, then the theory solver that detected the conflict constructs a learnt clause f_{conflict}, which captures the conflict over the Boolean variables X. The learnt clause is added to $F \leftarrow F \wedge f_{\text{conflict}}$ and the algorithm backtracks some or all of its assignments until the partial solution no longer conflicts with the new F. A trimmed down algorithm for DPLL(T) is shown in Algorithm 2.

Note II.5: The above description of DPLL(T) only captures the mechanisms of DPLL(T) that are used by cbTSP, a full description of DPLL(T) can be found in [45].

III. Problem Statement

In this paper, we consider path planning problems that require a set of robots to visit a set of locations in the environment to accomplish the problem goals (the robots complete a set of tasks at their locations). An optimal solution is one that allows the robots to accomplish the goals of the problem, while minimizing their travel costs. We model these problems with SAT-TSP by capturing the transition costs of the robots in their environment as a set of discrete graphs (one for each robot) and the logic of the problem is encoded as a SAT formula. The formula is used to dictate which locations are visited by which robots. In this way, we can have a location/task visited by multiple robots or just one robot. Additionally, the use of multiple graphs for multiple robots allows for heterogeneous robots (the transition costs can be different for each robot).

In the rest of this section, we introduce SAT-TSP and show how it is used to model a simple path planning example. Additionally, we classify SAT-TSP's complexity and show that even when it is composed of easy to solve SAT and TSP problems, the combination can still be hard.

A. SAT-TSP Definition

Before we define the decision version of SAT-TSP and its two optimization problems, we start with some notation. A SAT-TSP instance can take as input multiple graphs G_i, each with a cost budget c_i. To simplify the notation, we sometimes absorb the cost budget c_i into the graph's tuple $G_i = \langle V_i, E_i, w_i, c_i\rangle$ (when there is only one graph, we do not absorb the cost budget). A formula F has a solution or partial solution M, which is a collection of variable assignments. A variable x is assigned true if $x^T \in M$ and false if $x^F \in M$, otherwise x is unassigned.

Definition III.1 (SAT-TSP): The SAT-TSP decision problem takes as input $\langle G_1, G_2, \ldots, G_n, F, C\rangle$, where:

1) $G_i = \langle V_i, E_i, w_i, c_i\rangle$ is a directed, weighted graph with edge weights $w_i : E_i \rightarrow \mathbb{R}_{\geq 0}$ and cost budget c_i;
2) F is a Boolean formula defined over $X \supseteq V_1, V_2, \ldots V_n$;
3) C is a budget imposed on the total path cost.

Then, the instance is satisfiable if and only if:

1) there exists a tour of each graph G_i over a subset $V_i' \subseteq V_i$ with cost $c_i' \leq c_i$;
2) such that $\sum_{i=1}^{n} c_i' \leq C$;
3) and there exists an assignment M of X satisfying F such that a vertex variable $v = 1$ ($v^T \in M$) if and only if $v \in V_1' \cup V_2' \cup \ldots \cup V_n'$.

Note III.2: The above definition implies that there are two main types of variables, vertex variables, and auxiliary variables. Vertex variables are used to indicate if a location is visited or not and auxiliary variables are used to construct the logic of the problem into a concise formula (all problems presented in this paper utilize auxiliary variables). Vertex variables share the same labels as the vertices to emphasize the link between the variable and vertex, where auxiliary variables take on arbitrarily labels such as x_i.

In this paper, we consider two optimization problems for SAT-TSP: 1) minimize the total cost budget minimize C and 2) minimize the maximum cost budget of any graph G_i minimize $\max c_i$.

Furthermore, the instances studied in this paper can be expressed as metric-SAT-TSP instances.

Definition III.3 (Metric-SAT-TSP): A metric-SAT-TSP instance is one where each input graph $G = \langle V, E, w\rangle$ is complete and the edges satisfy the triangle inequality: $w(v_i, v_k) \leq w(v_i, v_j) + w(v_j, v_k)$ for all $v_i, v_j, v_k \in V$ such that $v_i \neq v_k$.

Note that robotic roadmaps are typically metric, as the time and/or distance to transition from a configuration A to another configuration C is less than or equal to that of transitioning from A to an intermediate point B, followed by transitioning from B to C. Additionally, if the roadmap is missing edges (not complete) then one can fill in the missing edges with shortest paths. For example, if $\langle v_i, v_j\rangle \notin E$ then we can add $\langle v_i, v_j\rangle$ to E and set $w(v_i, v_j)$ to the cost of the shortest path between v_i and v_j in the roadmap.

Example III.4 (Modeling a Problem in SAT-TSP): Consider two robots: robot 1 has a battery life of 10 min and can travel at 2 m/s, while robot 2 has a battery life of 12 min and can travel at 1 m/s. The environment contains locations $L = \{1, 2, \ldots, |L|\}$. Each location must be visited by either robot 1 or 2, but not both. We encode this as a SAT-TSP instance by first creating two graphs (i.e., roadmaps), $G_1 = \langle V_1, E_1, w_1, c_1\rangle$ and $G_2 = \langle V_2, E_2, w_2, c_2\rangle$ to capture the transition costs for each robot. Specifically, each graph G_r contains a vertex $v_i \in V_r$ for each location $i \in L$ and an edge $\langle v_i, v_j\rangle$ in the graph represents the transition from location i to j. The weight of the edge $\langle v_i, v_j\rangle$ in each graph is given by the time for the corresponding robot to travel from location i to location j. If the distance from i to j is $d_{i,j}$, then the weight for robot 1 is $w_1(i,j) = d_{i,j}/2$ and the weight for robot 2 is $w_2(i,j) = d_{i,j}$. Now the tuple $\langle V_1, E_1, w_1, 600\rangle$ captures the transition system for robot 1 and its battery budget, similarly tuple $\langle V_2, E_2, w_2, 720\rangle$ for robot 2.

Let $R = \{1, 2\}$ be the set of robots. Then, we construct the formula F using the set of variables $X = \{v_{i,r} | i \in L, r \in R\}$ to represent if vertex $v_i \in G_r$ is in the solution or not (true or false). We start by adding the set of clauses $(v_{i,1} \vee v_{i,2})$ to F for each $i \in L$, to express that each location $i \in L$ must be visited by at least one robot. Then, we add the clauses $(\neg v_{i,1} \vee \neg v_{i,2})$ to express that each location $i \in L$ can be visited by at most one robot.

Finally, we choose a value for the total cost budget C, to be any value ≥ 1320, since any solution satisfying the individual robot budgets will also satisfy this C. If we wish to find the solution with the lowest total cost, we search for feasible solutions that minimize C. ●

Remark III.5 (Modeling Path Planning Problems): SAT-TSP as a modeling language is most suited for expressing planning constraints as a SAT formula and motion transitions as one or more TSPs. Theoretically, like ILP, SAT-TSP can efficiently model any decision problem in NP, but practically there are limitations. For example, it is arguably cumbersome to use SAT-TSP for counting constraints (e.g., visit five locations). This is due to the simple structure that SAT uses for expressing logic. However, this structure has led to the success of modern SAT and SMT solvers [18], [45], [46], which cbTSP takes advantage of. Furthermore, we demonstrate in Sections VI and VII that SAT-TSP can be successfully used for small counting constraints.

B. Complexity of SAT-TSP

The decision version of SAT-TSP is NP-complete. This follows from the fact that SAT (an NP-hard problem) reduces to SAT-TSP and a SAT-TSP solution can be verified in polynomial time.

We also classify the complexity of SAT-TSP when it is composed of easy SAT and TSP problems. Let SAT* $\subseteq$ SAT and TSP* $\subseteq$ TSP be the set of problem instances that are solvable in polynomial time. An example of a TSP instance that is easy to solve (in TSP*) is an instance with a complete graph that has all of its edge weights equal to 1. Finding an optimal solution of cost $|V|$ is accomplished in polynomial time by choosing any ordering of the vertices. We are interested in SAT-TSP instances composed of easy SAT and TSP instances because it is often the case that TSP solvers work very well on TSP problems encountered in practice, such as those in the TSP library [47]. Similarly, SAT solvers are quite efficient on practical SAT problems, such as instances in the SAT library [48]. So does this mean that if our SAT-TSP problem is composed of easy instances in SAT* and TSP*, then it is easy to solve?

Theorem III.6: Consider the subset of SAT-TSP problems composed of instances from SAT* and TSP*, then this subset of SAT-TSP remains NP-complete.

Proof: We prove the above result by reducing a NP-complete problem to a SAT-TSP problem composed of SAT* and TSP* problems. Specifically, we do this for the SET-COVER problem. This problem takes in as input $\langle U, S, C\rangle$, where U is a universe of finite elements (a set), S is a collection of sets, for which each set $S_i \in S$ contains a subset of the elements from U ($S_i \subseteq U$), and C is a cost budget, then a solution is a subset $S' \subseteq S$ that covers all of the elements in U with S' and $|S'| \leq C$. The reduction to SAT-TSP maps the sets $S_i \in S$ to vertices v_i in the complete graph $G = \langle V, E, w\rangle$, where the edges in the graph all have a weight of 1. The inclusion/exclusion of a set S_i is indicated by the SAT-TSP tour visiting the vertex $v_i \in V$. The SAT-TSP formula

$$F = \bigwedge_{u_j \in U} \left(\bigvee_{\{i | u_j \in S_i\}} v_i \right)$$

is used to ensure that each element $u_j \in U$ is covered by at least one set S_i. A solution to the SAT-TSP problem $\langle G, F, C\rangle$ (C is given as input to the set cover problem) is a tour of length $c' \leq C$, which translates to a set cover solution with c' sets ($|S'| = c'$).

The TSP instance G and subinstances have the trivial solution of any tour (all tours have the same cost since all edges have weight 1). The SAT instance F and subinstances also have trivial solutions since there are no negative literals in the formula (we simply assign all the literals to be true). Thus, both the SAT and TSP instances are solved in linear time (polynomial time) and since SET-COVER is NP-hard, then it must be the case that SAT-TSP remains NP-complete despite the fact that the SAT and TSP problems are in SAT* and TSP*, respectively. ■

Theorem III.6 proves that SAT-TSP is NP-hard even when its subproblems are easy. Thus, we cannot expect to construct a polynomial time solver for SAT-TSP even when we have access to polynomial time solvers for its subproblems (unless P = NP). One could speculate that a similar consequence of the theorem would be that a good SAT-TSP solver would require more sophistication than a naïve combination of a SAT and TSP solver, which solves the SAT instance first then solves the TSP instance.

In Section IV, we start by exploring the BRUTE solver, which naïvely solves the SAT and TSP instances separately. The result is that it explores every SAT solution (possibly an exponential number of solutions). The cbTSP solver improves upon the BRUTE solver by adding the ability to negate partial solutions and leverages the sophistication of the SAT solver's branch heuristics. The ability to negate partial solutions allows cbTSP to more effectively prune the search space and the branch heuristics prioritizes the assignment of variables that are involved in the most number of conflicts.

IV. cbTSP: An SMT-Based Approach for SAT-TSP

In this section, we provide a simple BRUTE solver as a lead-in to the cbTSP solver. Then, we provide a high-level description of cbTSP and the conditions under which it can be used.

A. BRUTE Approach, a Lead-in to cbTSP

The BRUTE approach decouples the SAT-TSP instance by first solving the SAT instance and then the TSP instance. For simplicity, Algorithm 1 implements a SAT-TSP solver that only takes instances with one input graph. The algorithm is easily extended to take multiple graphs by replacing Line 5 with multiple calls to the TSP solver and bookkeeping the additional cost budgets.

The sequence of the BRUTE solver is as follows. First, it uses a SAT solver to find a feasible set of included vertices V' (Lines 2 and 3). Next, it uses the TSP solver, TSP-Solve, to find the minimum cost tour p', with cost $c' \leq C$ (Line 5), of the induced subgraph G' (Line 4). It negates the solution to prohibit it from reoccurring (Line 10) and repeats the process until it has checked every solution (Line 1). The problem with this approach is that there may be an exponential number of solutions to find and negate.

If we were solving metric instances, a less naïve approach would be to replace Line 9 with $f' \leftarrow (\bigwedge_{v \in V'} v)$. This would allow the algorithm to negate the solution, as well as all supersets of V', thus more effectively pruning the solution space. This approach would be valid for metric instances since we cannot lower the solution cost by adding vertices to the solution tour.

Algorithm 1: BRUTE-SAT APPROACH(G, F, C).

```
1  while SATISFIABLE$(F)$ do
2      $M' \leftarrow$ SOLVE$(F)$
3      $V' \leftarrow \{v \in V | v^T \in M'\}$
4      $G' \leftarrow$ SUBGRAPH$(G, V')$
5      $\langle p', c' \rangle \leftarrow$ TSP-SOLVE$(G', C)$
6      if $c' \leq C$ then
7          return $\langle M', p', c' \rangle$
8      else
9          $f' \leftarrow (\bigwedge_{v \in V'} v) \wedge \left(\bigwedge_{v \in V \setminus V'} \neg v\right)$
10         $F \leftarrow F \wedge \neg f'$
11 return $\emptyset$
```

The cbTSP approach works in this way, but it is also able to negate partial solutions. In fact, the cbTSP solver's parameters (found in Appendix B) allow it to be configured as a non-naïve brute approach (negate supersets of full solutions but ignore partial solutions). In this way, we can start to see how cbTSP is a sophisticated extension of the BRUTE approach.

B. cbTSP Solver

The cbTSP solver builds on the BRUTE approach by using partial solutions to prune the search space. This allows for significant computation savings. We begin by casting the SAT-TSP problem in the SMT framework. We use the DPLL(T) algorithm (see Algorithm 2) to couple a SAT and TSP solver. The cbTSP solver only works for TSP-monotonic instances (see Definition IV.2), which essentially means that the cost of partial solutions cannot be reduced by adding vertices. TSP-monotonicity is a necessary property needed to prune partial solutions. Additionally, metric SAT-TSP instances are TSP-monotonic (see Theorem IV.6).

Definition IV.1 (Partial Solution): Given a SAT-TSP instance $\langle G_1, G_2, \ldots, G_n, F, C \rangle$, a *partial solution* M is a true/false assignment for a subset of the variables in X. The subgraph G_i' *induced* by M is the subgraph induced by the vertices $V_i' \subset V_i$ when the corresponding variables in M are true.

Definition IV.2 (TSP-Monotonicity): A graph $G = \langle V, E, w \rangle$ is TSP-monotonic, if for any vertex subsets of the following form $V_1 \subset V_2 \subseteq V$ the induced subgraphs G_1 and G_2 have TSP costs $c_1 \leq c_2$.

The TSP-monotonic property allows for the negations of partial solutions that exceed the cost budget(s), since including more vertices cannot lower the solution cost. Specifically, if the partial solution exceeds the cost budget(s), then we exclude the responsible vertices and their supersets from reoccurring (the negation details are given in Algorithm 3, Lines 6 and 9).

To find optimal solutions, the cbTSP approach constructs and solves a series of SAT-TSP problems $\langle G_1, G_2, \ldots, G_n, F, C \rangle$ formulated as SMT problems (each with different cost budgets). The SMT formulation uses a custom TSP theory (see Algorithm 3) to construct the induced subgraph (Lines 2 and 3) and answer the decision problem of whether or not a TSP solution exceeds the cost budget(s). The SMT problem is as follows: the propositional formula is $F \wedge x_{\text{tsp}}$, where F is the Boolean formula in the SAT-TSP instance and the predicate x_{tsp} is decided by the custom TSP theory (x_{tsp} is true if and only if the theory can find a TSP tour of the included vertices within the cost budgets). The SMT theories have prior knowledge of the ground variables X, the graphs G_i, and the cost budget(s). The cost budget(s) are set by the user or a binary search algorithm used to find optimal solutions. The TSP theory is as follows.

Algorithm 2: Overview of DPLL(T) on F.

```
   precondition: TSP-Theory.setup$(G_1, \ldots, G_n, C)$
1  $M \leftarrow \emptyset$
2  while $\exists\ x \in X$ s.t. $\{x^T, x^F\} \cap M = \emptyset$ do
3      Add a new variable assignment to $M$
4      $f_{\text{conflict}} \leftarrow$ TSP-Theory$(M)$
5      if $f_{conflict} \neq \emptyset$ then
6          $F \leftarrow F \wedge f_{\text{conflict}}$
7          Backtrack $M$ to some point that does not
             conflict with $F$
8  if $M$ solves $F$ then
9      return $M$
10 else
11     return $\emptyset$
```

Algorithm 3: Overview of TSP-Theory (M).

```
1  for each graph $G_i$ do
2      $V_i' \leftarrow \{v_j \in V_i | v_j^T \in M\}$
3      $G_i' \leftarrow$ INDUCEDGRAPH$(G_i, V_i')$
4      $c_i' \leftarrow$ TSPSOLVE$(G_i', c_i)$
5      if $c_i' > c_i$ then
6          return $\neg \left(\bigwedge_{v_j \in V_i'} v_j\right)$
7  if $\sum c_i' > C$ then
8      $V' \leftarrow V_1' \cup \ldots \cup V_n'$
9      return $\neg \left(\bigwedge_{v_j \in V'} v_j\right)$
10 return $\emptyset$
```

Definition IV.3: The TSP-Theory takes as input the tuple $\langle G_1, G_2, \ldots, G_n, C \rangle$ and M, where

1) $\langle G_1, G_2, \ldots, G_n, C \rangle$ is the input to setup the theory and M is the input when called by the SMT solver;
2) each $G_i = \langle V_i, E_i, w_i, c_i \rangle$ is a TSP-monotonic graph with a cost budget c_i,
3) C is the total cost budget;
4) and M is a partial or full assignment of the ground variables in F.

Then, the theory is satisfiable ($x_{\text{tsp}} = 1$) if and only if

1) there exists a tour for each graph G_i of cost c_i or less over the set of vertices $\{v \in V_i | v^T \in M\}$;
2) and the total cost of the solution does not exceed C.

In the SMT formulation, the x_{tsp} predicate is forced to be true, thus all solutions and partial solutions must not exceed the cost budget(s). If the TSP theory finds that there is no solution, then a learnt clause is constructed (Lines 6 and 9 of Algorithm 3) and added back to the formula F (Line 6 of Algorithm 2). The

DPLL(T) solver is subsequently tasked with solving the new formula, which includes the learnt clause.

At a high level, cbTSP solves decision instances as follows:

1) set up the TSP-Theory with the graphs and budgets $\langle G_1, G_2, \ldots, G_n, C\rangle$;
2) call DPLL(T) on F (Algorithm 2);
3) build partial solutions (Line 3);
4) check the consistency of the TSP-Theory (Line 4);
5) negate partial solutions if there is a conflict (Line 6);
6) return the solution if satisfiable, otherwise return $\emptyset$.

The optimization version of cbTSP uses binary search to find a solution with the minimum cost budget (minimize C or minimize the maximum c_i) by solving a series of SAT-TSP decision instances with different cost budgets. Details of the binary search algorithm is found in Appendix A.

Note IV.4: Algorithm 2 is a simplified version of the DPLL(T) logic used in cbTSP. The real algorithm is more sophisticated than what is depicted. For example, in addition to keeping track of M, we also keep track of the solution paths and their costs, and we only call the TSP theory if M contains a change to the set of included vertices.

Example IV.5 (Illustration of the cbTSP Approach): This example demonstrates the interaction between the SAT solver (based on DPLL) and the TSP solver (the TSP theory) in cbTSP. Suppose we have one input graph $G = \langle V, E, w, c\rangle$ and suppose the solver has constructed a partial solution $M = \{x_{99}^T, v_2^F, v_1^T, x_{67}^T, v_4^T, v_5^T\}$. Then, suppose that the SAT solver extends the partial solution by assigning v_6 to be true. Once the assignment v_6^T is added to M, the TSP theory solver is called to make a consistency check. The TSP theory uses M to construct the TSP problem G' for the set of included vertices v_1, v_4, v_5, and v_6. Note x_{99} and x_{67} are not vertex variables (they are auxiliary variables) and so they do not appear in this set. The TSP theory then calls the TSP solver with input $\langle G', c\rangle$ and if a tour is found within the budget the theory returns true (consistent). The DPLL solver (SAT solver) continues to build upon the partial solution. If no tour is found within the budget, the check is inconsistent and the learnt clause $f_{\text{conflict}} = \neg(v_1 \wedge v_4 \wedge v_5 \wedge v_6)$ is constructed to be added to F so that the SAT solver avoids this solution in the future. The SAT solver then backtracks (revert some of the partial solution) to avoid the inconsistency and searches for a new solution that satisfies $F \wedge f_{\text{conflict}}$. •

C. Correctness of cbTSP

In this section, we show that metric instances are TSP-monotonic (see Definition IV.2) and prove that cbTSP yields the correct solution for TSP-monotonic instances.

Theorem IV.6: Metric graphs are TSP-monotonic.

Proof: We use contradiction to prove the above. Assume that there is some subset $V_1 \subset V$ and $V_2 = V_1 \cup \{v_j\}$ such that the induced subgraphs G_1 and G_2 have TSP costs $c_1 > c_2$. This means that the tour of G_1 can be shortened if we include the vertex v_j. Suppose the shortest tour of G_2 has the edge $\langle v_i, v_j\rangle$ in the path. We construct the graph G_1' to be a copy of graph G_1 and replace the weights of the outgoing edges for v_i with $w_1'(i,k) = w_2(i,j) + w_2(j,k)$. Now the graph G_1' will have the same optimal tour cost as G_2, but since the edge weights of G_1' compared to G_1 are equal or larger (triangle inequality) the optimal tour cost of G_1' cannot be lower than the optimal tour cost of G_1.

It follows that we cannot incrementally add vertices to V_1 to lower the TSP cost of the new graph. Consequently, the TSP costs c_1 and c_2 for V_1 and V_2 satisfies $c_1 \leq c_2$. Therefore, metric graphs are TSP-monotonic. ■

Theorem IV.7: The cbTSP approach is correct (i.e., sound and complete) on TSP-monotonic decision instances.

Proof: We first prove that the SMT formulations used by cbTSP are sound and then we prove that the SMT solver approach (DPLL(T) on F) used by cbTSP is sound and complete.

The soundness of the SMT formulation follows from the definition of SAT-TSP (see Section III) and Definition IV.2. Specifically, the SMT formulation allows for the negation of partial solution vertex sets and supersets for induced subgraphs that exceed the TSP budget(s). This does not remove any solutions in the search space that could have TSP costs within the budget, since the graph(s) are TSP-monotonic. Furthermore, a full SMT solution consists of TSP tour(s) of the included vertices and a satisfying assignment of F, which is a solution to the SAT-TSP instance. Therefore, the SMT formulation is sound, since it has the same solution set as the SAT-TSP formulation.

It follows that the solver approach for SMT instances is sound and complete, since it is based on the sound and complete algorithm DPLL(T) [45]. Therefore, the cbTSP approach is correct on TSP-monotonic instances. ■

D. Relaxing TSP-Monotonicity

In this section, we expand on the class of instances solvable by cbTSP (TSP-monotonic instances). Specifically, we describe a relaxation of TSP-monotonicity that can be used in practice with cbTSP when we have prior knowledge of a vertex set $A \subseteq V$ that must be included in the solution. The knowledge of this set allows us to relax the TSP-monotonicity property to apply to sets $V_1 \supseteq A$. The cbTSP solver in turn avoids partial solutions that exclude vertices in the set A and, thus, the correctness of negating partial solutions is still valid.

To motivate this direction, let us consider the following example. Let G be a metric graph with a start vertex v_s and a goal vertex v_g. Next, remove all the incoming edges of v_s and all the outgoing edges of v_g other than the one edge connecting v_g to v_s. Also, modify F to be $F = F \wedge v_s \wedge v_g$. This example is not TSP-monotonic. However, the addition to the formula is so simple that the cbTSP solver's preprocessing will assign v_s and v_g to be true (include v_s and v_g in the solution). Thus, all partial solutions that cbTSP checks will satisfy the TSP-monotonicity property, since the graph is otherwise metric (preprocessing happens before any calls to the TSP theory).

The set of required vertices A must not conflict with the formula F, i.e., F is satisfiable if and only if

$$F \wedge \left(\bigwedge_{v \in A} v\right)$$

is satisfiable. However, for this to work in practice, the decision to include the vertices in A would need to be done in the preprocessing step of cbTSP (before the DPLL solver is called,

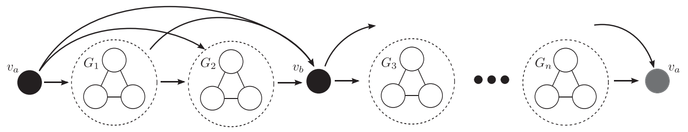

Fig. 2. A-TSP-monotonic example, where the black locations represent the set A. Interconnecting edges to or from a graph G_i represents a collection of edges that connect every vertex in the source to every vertex in the destination. The edge weights of the interconnecting edges satisfy the triangle inequality.

which is not discussed in this paper). This happens for atomic clauses (clauses with one literal), e.g., given the clause (v), the prepossessing phase would assign a true value to v so that the clause is satisfied.

Definition IV.8 (A-TSP-Monotonicity): Given a graph $G = \langle V, E, w\rangle$ and a subset $A \subset V$, then G is *A-TSP-monotonic* if for any vertex subsets $V_1 \subset V_2 \subseteq V$ such that $A \subseteq V_1$, the induced subgraphs G_1 and G_2 have TSP costs $c_1 \leq c_2$.

Proposition IV.9: The cbTSP approach is correct on A-TSP-monotonic decision instances if the decision to include the vertices in A is done so in the preprocessing phase of the solver (before DPLL is called).

Proof: The proof follows Theorem IV.7's proof—the instance behaves like a TSP-monotonic instance once DPLL is called. ■

Remark IV.10 (A Sufficient Construction for Achieving A-TSP-Monotonicity): Given a graph $G = \langle V, E, w\rangle$, the problem is A-TSP-monotonic for a set A if the following is true for every pair of edges $\langle v_i, v_j\rangle, \langle v_j, v_k\rangle \in E$ such that $v_i, v_k \in V$ and $v_j \in V \setminus A$:

1) the edge $\langle v_i, v_k\rangle$ must also be in E;
2) and $w(v_i, v_k) \leq w(v_i, v_j) + w(v_j, v_k)$.

●

Notice that our motivating example with G, v_s, and v_g satisfies the above construction for $A = \{v_s, v_g\}$. However, if we were to remove the edge $\langle v_s, v_g\rangle$, then the instance would not satisfy the above construction and it would not be A-TSP-monotonic. This is due to the fact that the infeasible partial solution of v_s and v_g can be made feasible by adding any additional vertex to the solution.

Fig. 2 illustrates a more complex example that was constructed using Remark IV.10. Here, a set of metric subgraphs $G_1, G_2, \ldots, G_n$ are connected together with a ring graph to dictate the order that the metric graphs are to be traversed. The ring graph is constructed from vertices in A (the black vertices) and the metric subgraphs are connect to A in between adjacent vertices in the ring.

V. Integer Program Formulation for SAT-TSP

In this paper, we are interested in path planning problems that require robots to travel between multiple task locations (patrolling, sample collection, and periodic routing). Specifically, all of the path planning problems seek to find shortest tours over a subset of the vertices (not necessarily a fixed set of vertices) in each input graph (i.e., robotic roadmap). This section describes the standard method for expressing such problems as an integer linear program. The additional constraints that define the specific problem and guide the solution to choose the set of visited locations are added to the formulation in subsequent sections. The ILP formulation for the TSP aspect of the problem is drawn from the vehicle routing literature [49] and is a standard method used in the operations research community [49].

To be able to solve multiple TSP instances simultaneously, we require that each instance has a home location that must be visited. In this way, we are able to eliminate subtours that do not visit one of the home locations.

Let the set of Boolean variables $e^k_{i,j} \in \{0,1\}$ represent the inclusion or exclusion of the edge $\langle v_i, v_j\rangle \in E_k$ from the solution ($e^k_{i,j} = 1$ indicates inclusion). Similarly, let v^k_i represent the set of Boolean variables representing the inclusion/exclusion of the vertex $v_i \in V^k$ from the solution. Additionally, without loss of generality, let v^k_1 be the home location in each graph G_k. The following ILP template encodes TSP problems with multiple graphs over a nonfixed set of vertices and minimizes the max cost of the individual tours. The formulation for minimizing the total cost simply replaces the objective with $\sum_{k=1}^{n}\sum_{i=1}^{|V^k|}\sum_{j=1}^{|V^k|} w_k(v_i, v_j)\, e^k_{i,j}$

minimize

$$c_{\max} \tag{1}$$

subject to

$$\sum_{k=1}^{n}\sum_{i=1}^{|V^k|}\sum_{j=1}^{|V^k|} w_k(v_i, v_j)\, e^k_{i,j} \leq C \tag{2}$$

subject to, for each $k \in \{1, 2, \ldots, n\}$

$$v^k_1 = 1, \tag{3}$$

$$\sum_{i=1}^{|V^k|} e^k_{i,j} = v^k_j, \text{ for each } v_j \in V_k \tag{4}$$

$$\sum_{j=1}^{|V^k|} e^k_{i,j} = v^k_i, \text{ for each } v_i \in V_k \tag{5}$$

$$\sum_{i\in P}\sum_{j\in P} e^k_{i,j} \leq \sum_{i\in P} v^k_i - v^k_l,$$

$$\text{for each } P \subseteq \{2, \ldots, |V^k|\}, \text{ and some } l \in P \tag{6}$$

$$\sum_{i=1}^{|V^k|}\sum_{j=1}^{|V^k|} w_k(v_i, v_j)\, e^k_{i,j} \leq \min(c_k, c_{\max}). \tag{7}$$

Constraint (2) ensures the total solution cost is within the budget C. Constraint (3) forces the solution to include the home vertices. Constraints (4) and (5) restrict the incoming and outgoing degree for each location to be one only if the vertex is included. Constraint (6) is the subtour elimination constraint, where the subtour is represented by the set P, which cannot include the home location. The subtour is eliminated by ensuring that the number of edges between vertices in P is less than $|P|$ (a subtour would have $|P|$ edges). If one or more locations are not visited in P, i.e., P is not a subtour, then the inequality is satisfied by the fact that each included vertex has at most one incoming and one outgoing edge (a strict subset of P can have at most

$|P| - 1$ edges). In practice, the subtour elimination constraint is lazily implemented (if a constraint in (6) is violated during the construction of the solution, then the constraint is added to the formulation) to avoid expressing an exponential number of constraints. Finally, constraint (7) ensures the individual tours are within the budgets c_k and it ensures that the individual tour costs are equal or lower than the optimal value $c_{\max}$.

VI. Robotic Applications and Expressions

In this section, we describe three robotic path planning problems (patrolling, sample collection, and periodic routing) and show how they are expressed in both SAT-TSP and ILP. The SAT-TSP expressions for these problems utilize at-most-one-in-a-set constraints (also known as mutual exclusivity). We define this type of constraint here and use it throughout the rest of the section.

Definition VI.1 (At-Most-One-in-a-Set Constraint): Given a set $X' \subset X$ of variables, the *at-most-one-in-a-set constraint* states that at most one variable in the set X' can be assigned true. The following Boolean formula expresses the constraint:

$$\bigwedge_{x,y \in X' | x \neq y} \neg(x \wedge y).$$

A. Patrolling Problem

We consider patrolling problems that require each point of interest to be observed from multiple viewpoints. These problems were inspired by [50] and [51]. Our patrolling problem is defined on a metric environment that contains m points of interest, $n-1$ "observation locations" and one home location. The robot must visit a subset of the observation locations to observe all of the points of interest and return home. A point of interest p is observable from a location v if the distance between p and v is less than or equal to a threshold (provided by the user). Each point of interest p must be observed by at least two complementary locations, referred to as a "complementary pair." An observation location v_j is complementary to v_i if both points observe p from perspectives that are separated by a minimum threshold angle (provided by the user).

A solution is a tour that visits a set of observation locations with minimum length such that each point of interest is observed by at least one complementary pair. An illustrative example of this problem is given in Fig. 3 (left illustration).

To aid in the expression of the problem, we let the set V_p represent the set of observation locations that can observe the point of interest p. The set $\hat{V}_{p,i}$ represents the set of locations that are complementary observation locations of v_i for p and the set $P = \{1, 2, \ldots, m\}$ represents points of interest.

1) SAT-TSP Expression: This problem is encoded into SAT-TSP by constructing a formula F that captures the logic of visiting the home location and at least one complementary pair for each point of interest. The clause (v_h) captures the home location requirement (v_h is the robot's home) and the set of clauses

$$\bigvee_{v_i \in V_p} v_i \wedge \left(\bigvee_{v_j \in \hat{V}_{p,i}} v_j \right), \text{ for each } p \in P$$

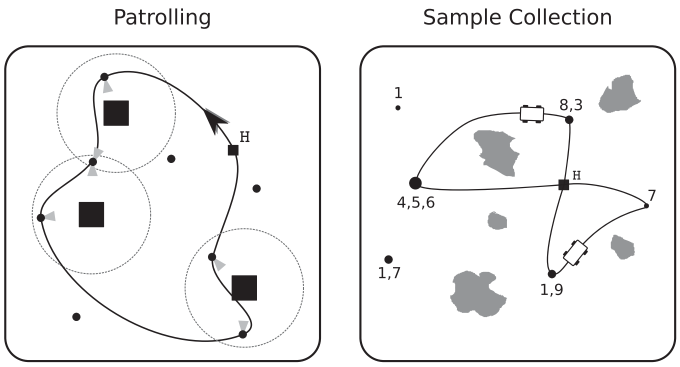

Fig. 3. On the left is a UAV patrolling example and on the right is a UGV sample collection example. The robot's path is indicated with a solid line in both illustrations and the location marked by an "H" represents the robot's home. For the patrolling problem, points of interest are buildings represented by large squares, the faint circles represent the radius that the building can be observed from, and the gray triangles indicate the different observation perspectives. In the sample collection problem, the labels above the locations represent the mineral types that are within the sample (large samples have three minerals and small samples have one). The gray contours represent obstacles.

captures the logic of visiting at least one complementary pair for each point of interest (p is observed from v_i only if it is observed by at least one of its complements v_j). Now the tuple $\langle G, F, C \rangle$ encodes the patrolling problem as a SAT-TSP instance, where G is the discrete graph representing the distances between observation locations and C represents the maximum cost of the solution tour (finding the minimum C solves the optimization problem).

2) ILP Expression: To express this problem as an ILP, we build upon the formulation given in Section V. Let the set of Boolean variables $\hat{v}_{p,i} \in \{0, 1\}$ represent whether or not v_i is part of a complementary pair observing the point of interest p. The following are the additional constraints in the ILP formulation:

$$\text{for each } p \in \{1, 2, \ldots, m\}$$

$$\hat{v}_{p,i} \leq v_i, \text{ for each } v_i \in V_p \tag{8}$$

$$\hat{v}_{p,i} \leq \sum_{v_j \in V_{p,i}} v_j, \text{ for each } v_i \in V_p \tag{9}$$

$$\sum_{v_i \in V_p} \hat{v}_{p,i} \geq 1. \tag{10}$$

Constraint (8) restricts the indicator $\hat{v}_{p,i}$ from being true ($\hat{v}_{p,i} = 1$) if the location v_i itself is not visited, Constraint (9) restricts the indicator from being true if none of the complementary locations of v_i are visited, and Constraint (10) ensures that at least one complementary pair is visited.

B. Sample Collection Problem

The following problem is inspired by sample collection problems that arise for science rovers, as described in [6]. In this problem, we have a set of R robots, $n-1$ samples, one home location, and a set of m different minerals that can appear in the $n-1$ samples. Each sample in the environment is either small, medium, or large. Small samples have within them one type of mineral, medium samples have up to two different minerals, and large samples have up to three different minerals. There are two types of robots used to collect samples, small, and large. A small robot can collect an unlimited number of small samples and up to one medium sample but no large samples. A large robot cannot collect small samples, but it can collect an unlimited number of medium samples and up to one large sample. The small and large robots have different speeds. Each robot is given the same time budget to collect samples and return home. Since each location only contains one sample, multiple robots are restricted from visiting the same location.

A solution to this problem is a tour for each robot that satisfies the time budget, starts at the home location, and visits a set of locations in the environment that allows the robots to collect a set of samples that contain one of each type of mineral found in the environment. An optimal solution minimizes the total time taken by the set of robots. An illustrative example of this problem is given in Fig. 3 (on the right). In this example, there are no samples with mineral 2 and so feasible solutions do not collect mineral 2. Additionally, we demonstrate a sample collection solution in our supplementary video.[1] This solution utilizes one small and one large robot to collect seven different mineral types.

To express this problem, we introduce the following. Let V_t be the set of locations that contains a mineral of type t; let V_S, V_M, and V_L be the set of locations that respectively have small, medium, and large samples; let R_S, R_L, and $R = \{1, 2, \ldots, |R|\}$, respectively, be the set of small, large, and all robots; and let $T = \{1, 2, \ldots, m\}$ be the set of minerals that exist in the environment.

1) SAT-TSP expression: The problem is encoded as a SAT-TSP instance by constructing a graph G_r for each robot $r \in R$ and a formula F that captures the goals and capacity restrictions of the robots. Each graph G_r contains all of the locations in the environment and its edge weights are given by transitions for robot r to move within its environment. The set of variables $v_{i,r}$ is used to indicate if robot r visits location i. When the location/robot pair is incompatible, such as location i contains a large sample and robot r is small, the variable is negated. We chose this approach instead of constructing graphs without incompatible location/robot pairs to simplify the SAT-TSP expression for this paper.

We start our expression by forcing each robot to visit the home location using the clauses $\bigwedge_{r \in R} v_{h,r}$, where $v_h \in V$ is the home location for the robots. Next, we negate all incompatible location/robot pairs (e.g., small robot and large sample) $\bigwedge_{v_{i,r} \in X_I} \neg v_{i,r}$, where $X_I = \{v_{i,r} | v_i \in V_L, r \in R_S\} \cup \{v_{i,r} | v_i \in V_S, r \in R_L\}$. We encode the goal of collecting at least one of each mineral, with a disjunctive clause for each mineral

$$\bigwedge_{t \in T} \left(\bigvee_{v_i \in V_t, r \in R} v_{i,r} \right).$$

To restrict multiple robots from visiting the same location v_i, we use at-most-one-in-a-set constraints (Definition VI.1) for the sets $\{v_{i,r} | r \in R\}$ for each $v_i \in V$. Finally, we restrict the robots' carrying capacity with at-most-one-in-a-set constraints on the sets $\{v_{i,r} | v_i \in V_M\}$ for each small robot $r \in R_S$ and $\{v_{i,r} | v_i \in V_L\}$ for each large robot $r \in R_L$.

The tuple $\langle G_1, G_2, \ldots, G_{|R|}, F, C \rangle$ now encodes the problem as a SAT-TSP instance, where $G_r = \langle V_r, E_r, w_r, c_r \rangle$ is the discrete graph for robot r that captures the time transitions, c_r is the robot's time budget, F captures the goals and constraints of the problem, and C encodes the maximum total time incurred by all of the robots. The total time budget C can be minimized to find the optimal solution.

2) ILP expression: To express this problem as an ILP, we build upon the formulation given in Section V. Similar to the SAT-TSP expression, we negate incompatible location/robot pairs instead of altering the previously established ILP formulation. For example, a location i would be incompatible with the small robot r if it contained a large sample. The incompatibility is negated with the assignment $v_i^r = 0$ (notation defined in Section V). The following is the extension of the ILP formulation:

$$\sum_{v_i \in V_t} \sum_{r \in R} v_i^r \geq 1, \text{ for each } t \in \{1, 2, \ldots, m\} \tag{11}$$

$$\sum_{r \in R} v_i^r \leq 1, \text{ for each } v_i \in V \tag{12}$$

$$\sum_{v_i \in V_M} v_i^r \leq 1, \text{ for each } r \in R_S \tag{13}$$

$$\sum_{v_i \in V_L} v_i^r \leq 1, \text{ for each } r \in R_L \tag{14}$$

$$v_i^r = 0, \text{ for each } v_i^r \in I \tag{15}$$

where

$$I = \{v_i^r | v_i \in V_L, r \in R_S\} \cup \{v_i^r | v_i \in V_S, r \in R_L\}$$

is used to represent the set of incompatible location/robot pairs. Constraint (11) encodes the goal of retrieving at least one of each mineral type, Constraint (12) restricts multiple robots from retrieving the same sample, Constraint (13) restricts small robots from collecting more than one medium sample, Constraint (14) restricts large robots from collecting more than one large sample, and Constraint (15) negates the incompatible location/robot pairs.

C. Period Routing Problem

The period routing problem [9], [11] requires a robot to service a set of $n - 1$ locations over a set of m periods. In each period, the robot starts from a designated home location and travels to a subset of the service locations. Each location v requires $f(v) \leq m$ out of m periods of service and has restrictions on which periods or period combinations are allowable. For this paper, all period combinations that avoid back-to-back visits are allowed (the first and last period are considered back-to-back). This problem arises in collection [11] and delivery [9] tasks. Also period routing problems often contain a capacity that limits the number of tasks that can be performed.

A solution to this problem is a set of tours, one for each period that meets the service demands of the locations, respects the capacity limits, and respects the service restrictions. The

[1] See supplementary video attachment.

optimal solution minimizes the maximum length tour over the m periods.

1) SAT-TSP expression: The problem is encoded as a SAT-TSP instance by constructing a graph for each period and a formula that captures the goals and restrictions of the problem. Each graph G_p represents the transition costs for the robot to move within its environment during period p (all graphs are the same).

The problem goals of providing a specific number of service periods to the locations is captured with the standard technique of using adder circuits [52], which are efficiently translated to a Boolean formula [53] and added to F. A brief overview of how adder circuits are used within this context can be found in Appendix C. These adder circuits take as input the set of location/period variables for each location $\{v_{i,p} | p \in P\}$ and output a set of Boolean variables $\{b_{i,1}, b_{i,2}, b_{i,4}, \ldots\}$ that capture the binary encoding of how many of the inputs are true. Next, the outputs of the adder circuit are forced to the desired service demands $f(v_i)$. As an example, if v_i requires two visits, then we force the twos bit $b_{i,2}$ to be true and the remaining bits to be false. This is accomplished by adding the following to $F \leftarrow F \wedge \neg b_{i,1} \wedge b_{i,2} \wedge \neg b_{i,4} \wedge \ldots$.

We force the robot to visit the home locations with the clause $\bigwedge_{p \in P} v_{h,p}$, where $v_h \in V$ is the home location of the robot. The back-to-back period restriction is exhaustively handled by negating every possible illegal combination

$$\neg(v_{i,p} \wedge v_{i,p^+}) \text{ for all } v_i \in V, p \in P$$

where $p^+ = (p+1) \mod m$.

The tuple $\langle G_1, G_2, \ldots, G_m, F, C \rangle$ now encodes the problem as a SAT-TSP instance, where: $G_p = \langle V_p, E_p, w_p, c_p \rangle$ is the discrete graph for period p; the cost budget C is set to ∞ to indicate that there is no budget; F captures the goals and constraints of the problem; and c_p encodes the maximum cost of graph G_p's path which is minimized to find the optimal solution.

2) ILP expression: To express this problem as an ILP, we build upon the formulation given in Section V. The following are the additional constraints in the ILP formulation:

$$\sum_{p \in P} v_i^p = f(v_i), \text{ for each } v_i \in V \tag{16}$$

$$v_i^p + v_i^{p^+} \leq 1, \text{ for each } v \in V \text{ and } p \in P \tag{17}$$

where $p^+ = (p+1) \mod m$. Constraint (16) encodes the goal of servicing each location the proper number of times and Constraint (17) restricts back-to-back visits.

VII. Simulation Results

In this section, we present simulation results that compare cbTSP to an ILP solver on the instances presented in Section IV. Additionally, we explore the scalability of SAT-TSP with respect to the number of robots. The problem instances in this section all have metric travel costs and are thus solvable by cbTSP (metric instances are TSP-monotonic).

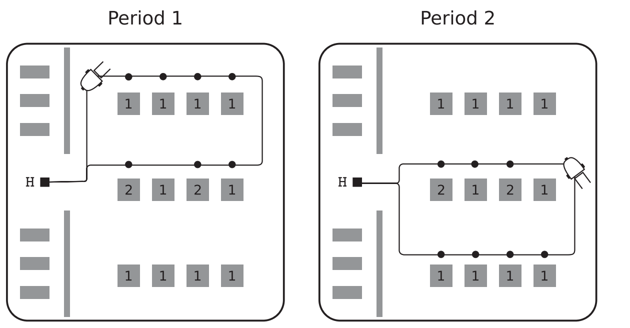

Fig. 4. Simple period routing example for material transport within a factory. There are only two periods of service and a location can either require service for one or both periods (as indicated on the graph). The home location is labeled with an "H."

A. Simulations

We ran all simulations on an Intel Core i7-4600U, 2.10 GHz with 16 GB of RAM. The cbTSP solver used a custom DPLL(T) solver (based on MiniSat [18]) to test partial solutions by making external callbacks to a TSP solver (a version of LKH [19] we modified to solve decision TSP problems). The cbTSP solver takes as input the SAT-TSP instance, a time budget, and a set of parameters used to configure the solver. The best solution found within the time budget is returned. The solver parameters are used to configure the behavior of cbTSP's SAT and TSP solvers as well as a few cbTSP specific behaviors. The details of these parameters and their values are given in Appendix B. The ILP solver, Gurobi [17], was accessed through Python and also takes as input a time budget. To make a fair comparison, we restricted Gurobi to a single process (thread). Additionally, we seeded the solver with a random seed each time it was called to ensure each run was different from the last (set using Gurobi's parameters).

All of our simulations use a 300 s time budget and the best solution found within the budget is reported in our results. As well, we track the solvers progress within its time budget to compare the solver's results as they are found. The use of a fixed time budget allows us to simulate real-world conditions, where the robot must make decisions while it operates as opposed to letting the solvers run overnight (or longer) to completion. In the latter case of running the solvers to completion, comparing solution quality would be meaningless since both approaches would find optimal solutions. Thus, using a fixed time budget allows us to compare the solver's ability to find solutions quickly. Typically, both solvers consume their entire time budget and so we do not explicitly report solver times as they would all be 300 s. Instead, we compare the average solution quality found over ten separate runs for each solver in Tables I-III and track the solution quality found over time in Fig. 5.

The time budget does not include the time taken to create the SAT-TSP and ILP instances (reduction time), as we do not wish to benchmark this process. However, the reduction times are polynomial with respect to the input size.

B. Patrolling Problem Setup and Results

The patrolling problem instances were generated as follows. All of the locations (the $n-1$ observation locations, the robot's home, and the m points of interest) were uniformly

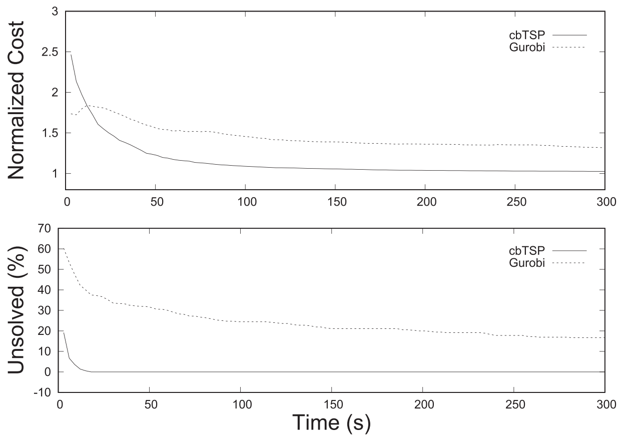

Fig. 5. Comparison of cbTSP to Gurobi of how solutions evolve over time. The top plot captures the average solution quality, normalized with the best known result and the bottom plot captures the number of solver runs that do not have feasible solutions.

TABLE I
EXPERIMENTAL RESULTS FOR PATROLLING SIMULATIONS

			Gurobi		cbTSP	
No.	n	m	Best Cost	Avg. Cost	Best Cost	Avg. Cost
1	40	5	2268	2292	**2260**	**2260**
2	40	10	1812	1812	1812	1812
3	40	20	3171	3171	**3111**	**3111**
4	60	7	1825	1925	**1801**	**1801**
5	60	15	2615	2845	**2430**	**2430**
6	60	30	3252	3346	**3176**	**3176**
7	80	10	2020	2227	**1903**	**1903**
8	80	20	2948	4681	**2533**	**2602**
9	80	40	4640	6920	**3244**	**3585**
10	100	12	2398	3265	**1937**	**1939**
11	100	25	2825	4324	**2431**	**2473**
12	100	50	7385	8273	**3803**	**4104**

randomly distributed in a 1000×1000 meter two-dimensional (2-D) square. A point of interest $p = (x_p, y_p)$ is observable from location $v = (x_v, y_v)$ if

$$|p - v| \leq \frac{4000}{5m\sqrt{m}}$$

for which the equation was designed to help ensure that most randomly generated instances will have a feasible solution (one where each point of interest can be observed by a complementary pair). A location u is complementary to v for p if

$$(\theta_v - \theta_u) \mod 360 \geq 60$$

where

$$\theta_v = \tan^{-1}\left(\frac{y_p - y_v}{x_p - x_v}\right) \text{ and } \theta_u = \tan^{-1}\left(\frac{y_p - y_u}{x_p - x_u}\right).$$

An illustrative example of this problem is given in Fig. 3 (on the left).

We compared cbTSP to the ILP solver, Gurobi, on a set of 12 instances. The results are shown in Table I, where the left three columns indicate the instance id, and the number of observation locations (n) and points of interest (m) in the environment. The best results are highlighted. As we can see cbTSP outperforms Gurobi on almost every instance and as the instances get more difficult (lower in the table) the performance gap increases as shown by the first and last instances in the table—Gurobi has an average cost of 1.01 times more than cbTSP on the first instance and an average cost of 2.01 on the last.

TABLE II
EXPERIMENTAL RESULTS FOR SAMPLE COLLECTION SIMULATIONS

			Gurobi		cbTSP	
No.	n	m	Best Cost	Avg. Cost	Best Cost	Avg. Cost
1	10	10	3500	3500	3500	3500
2	20	10	3355	3356	3355	**3355**
3	20	20	8563	8563	8563	8563
4	40	10	1876	1893	1876	**1880**
5	40	20	5251	5429	**5117**	**5397**
6	40	40	9117	9682	**8812**	**9235**
7	60	10	1613	1730	**1591**	**1603**
8	60	30	-	-	**11 809**	**13 077**
9	80	10	1382	1580	**1299**	**1332**
10	80	40	10 941	**11 009**	**10 453**	12 073
11	100	10	1883	2863	**1363**	**1419**
12	100	50	-	-	**14 668**	**14 949**

C. Sample Collection Setup and Results

The sample collection problem instances we tested were generated as follows. There is one home location and $n - 1$ sample locations in the environment and the set of samples contain up to m different mineral types. All locations (sample locations and the home location) were uniformly randomly distributed in a 1000×1000 m 2-D square. The distribution of sample sizes are as follows: 3:1 for small to large and 2:1 for small to medium. The types of minerals in each sample are uniformly randomly distributed (the same mineral type may reoccur in a sample). There are two types of robots used to collect samples, small and large, three of each, six in total. Each robot has a time budget of 25 min to collect samples and return home and the small robots travel at a speed of 2 m/s, while the large robots travel at half that speed (1 m/s).

Note VII.1: Although the large robots can complete the task by collecting about half the samples, it costs double to travel the same distance. Furthermore, it is equally likely for a mineral type to be found in a small, medium, or large sample.

We compared cbTSP to the ILP solver Gurobi on 12 sample collection instances. The results are reported in Table II, where the left three columns indicate the instance id, and the number of samples n and maximum number of different minerals m in the environment. The best results are highlighted. Dashes in the table indicate that the solver was not able to find a feasible solution. The results show that cbTSP is competitive with Gurobi. Specifically, both solvers find the same quality solutions for the easy instances (instances 1–3); then as the instances get more difficult (instances 4–7), cbTSP starts to outperform Gurobi; and for the most difficult instances (instances 8–12), cbTSP often outperforms Gurobi by quite a bit. Gurobi only outperforms cbTSP on one instance and fails to find feasible solutions for two out of the five difficult instances.

1) Physics-Based Simulations: We have also performed simulation experiments for sample collection in more complex environments (shown in Fig. 1) where travel costs are shortest collision free paths. These simulations utilize the Clearpath Husky robot model within Gazebo. The robots use the ROS

TABLE III
EXPERIMENTAL RESULTS FOR PERIOD ROUTING SIMULATIONS

		Gurobi		cbTSP	
No.	n	Best Cost	Avg. Cost	Best Cost	Avg. Cost
1	15	2278	2392	**2212**	**2212**
2	15	1968	2055	**1904**	**1904**
3	20	2200	2382	**2022**	**2022**
4	20	1807	2016	**1670**	**1670**
5	25	2706	3718	**2310**	**2347**
6	25	2638	2869	**2057**	**2057**
7	30	2877	3133	**2342**	**2358**
8	30	3040	3811	**2433**	**2439**
9	35	3738	3944	**2502**	**2610**
10	35	3667	3902	**2404**	**2506**
11	40	4369	4680	**2727**	**2954**
12	40	3934	4529	**2594**	**2678**

navigation stack [20] to move within the physics simulator. An example simulation, visualized with RViz, is included as a supplementary video. In the video there are two robots, each with different speed. The slower Husky emulates the large sample collecting robot and the faster Husky emulates the small robot. The collection task requires collecting seven different mineral types from the environment.

D. Period Routing Setup and Results

The period routing problem instances were generated as follows. All of the locations (the $n-1$ service locations and the robot's home) were uniformly randomly distributed in a 1000×1000 m 2-D square. There are six service periods and each service location may require either one, two, or three periods of service (uniformly randomly assigned) out of the six periods. An illustrative example is shown in Fig. 4.

We have compared cbTSP to the ILP solver, Gurobi, on 12 period routing instances. The results are shown in Table III, where the left two columns indicate the instance id and the number of locations in the environment. The best results are highlighted. The table shows that cbTSP outperforms Gurobi on the majority of the instances and like the other two applications, as the instances become more difficult the performance gap between cbTSP and Gurobi grows. This can be seen as a trend between the first and last instance in the table. Gurobi starts out by finding solutions that are close to the quality of cbTSP (1.08 times the cost of cbTSP) then degrades (1.69 times the cost) as the instances become more difficult.

E. Solution Quality as a Function of Runtime

Both cbTSP and Gurobi provide updates on the quality of the solutions found during run time. This is particularly useful for online robotic applications where the robot can decide to accept a solution that is good enough instead of waiting for a better solution.

Fig. 5 compiles all of the solver data for all three libraries. From the data, we can see that cbTSP is able to able to find more feasible solutions than Gurobi with less time. Additionally, we can see that the quality of these solutions quickly improve within the first 60 s, while the solutions found by Gurobi improve at a slower rate. One thing to note is that the solution quality data is an average of the solutions found by these approaches, so making a direct comparison of the two approaches can be misleading since the data for one solver may include an average of an instance that the other solver has not been able to solve yet. This accounts for the degradation of Gurobi's solution quality at around 10 s. Here, new solutions are added to the average that were not part of the average before and in the case that these new solutions have a larger normalized cost, the new results will degrade the overall average.

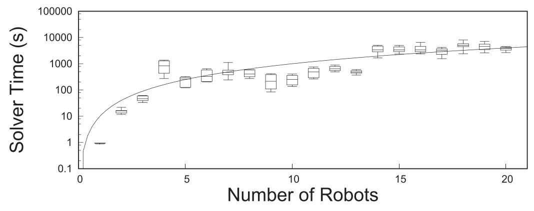

Fig. 6. Runtime to solve patrolling problems as a function of the number of robots. The boxes capture 50% of the trials, while the tails capture 100% of the trials. The curve shown for scale is $10|R|^2$.

F. Scaling With Number of Robots

In this section, we explore scalability with the number of robots. We consider the patrolling problem as described in Section VI-A. To scale the problem, we allow multiple robots to visit the same location and set the objective to minimize the maximum length robot tour. This is a natural alternative to minimize the sum of tour lengths, which encourages all robots to be utilized in the solution, rather than having a single robot tour the observation locations. The formula is updated with the following set of clauses for each $v_i \in V$:

$$v_i \iff \left(v_{i,1} \vee v_{i,2} \vee \ldots \vee v_{i,|R|}\right)$$

to capture whether or not a location was visited by any robot.

The simulations range from 1 to 20 robots and the solver is allowed to run to completion. The environment is scaled as follows: the number of points of interest in the environment is $10|R|$ and the number of observation locations is $20|R|$, where R is the set of robots available. With this scaling, each robot visits approximately four locations. The constructed SAT-TSP instance has both its environment size (input graphs) and its formula size proportional to R^2. Thus, the problem size grows quadratically with R while its search space grows exponentially with R. Fig. 6 plots the time needed to solve the instances as a function of the number of robots available. The runtimes range from a few seconds for instances with a single robot to approximately 5000 s for instances with close to 20 robots. Note that the log scale in the plot highlights that the growth in runtime is not exponential even though the search space grows exponentially, instead the solver time approximately grows quadratically with the number of robots, like the problem size.

VIII. CONCLUSION

In this paper, we introduced the SAT-TSP problem and the effective solver cbTSP, which enabled us to express and solve discrete robotic motion planning problems with complex constraints. We have demonstrated that cbTSP often outperforms a commercial grade ILP solver, showing that the cbTSP is a good choice for the problems demonstrated in this paper. One can obtain the cbTSP solver from https://github.com/fcimeson/cbTSP.

There are a number of directions we are pursuing for our future work. One direction is to implement some of the advanced techniques used by DPLL and DPLL(T), such as clause forgetting, propagations, and custom search heuristics. Although, we have not found memory blow up to be a problem, clause forgetting would help avoid this potential problem by forgetting conflict clauses that our method added to the formula but are rarely if ever part of a conflict. With propagations, we could implement custom SMT theories useful for planning such as location ordering constraints, which in turn would offload some of the logic from the SAT formula and put it into a more compact form that the custom SMT theory can solve. Currently, cbTSP uses search heuristics from the SAT solver to choose the order of variable assignments, which does not utilize knowledge of the TSP problem. We believe we could use this knowledge to improve the search heuristic and thus improve the solver's performance. Additionally, we plan to investigate the effectiveness of using SAT-TSP for collision avoidance problems. Our approach would extend the environment graph to incorporate discretized time steps. In this way, we can use the location/time vertices to prohibit multiple robots from occupying the same location during the same discrete time step, as in [27]. Finally, we are working on a replanning scheme that preserves aspects of the solution that can be used for the new problem.

Appendix A
cbTSP: Binary Search

The cbTSP solver uses a modified version of binary search to find optimal solutions. Algorithm 4 shows this approach for optimization problems that aim to minimize the cost C. The `bdiv` parameter is configured through the cbTSP interface (default setting is 10) and `upper_bound` is an upper bound on the worst cost solution that can be set to $\sum_{i=1}^{n} \max_{\langle v_a, v_b\rangle \in E_i} w_i(v_a, v_b)|V_i|$. The same basic algorithm is used to find optimal solutions for problems that aim to minimize the maximum cost c_i (subgraph path cost) by replacing C with c_1 and for each $G_i = \{V_i, E_i, w_i, c_i\}$ replace c_i with c_1 so that c_1 now constrains the maximum subgraph cost for each G_i.

Algorithm 4: Binary Search($G_1, G_2, \ldots, G_n, F$).

```
1  $C^- \leftarrow 0$
2  $C^+ \leftarrow$ upper_bound
3  $C^* \leftarrow$ upper_bound
4  while $C^- < C^+$ do
5      $C \leftarrow C^+ - \left\lfloor \frac{C^+ - C^-}{\texttt{bdiv}} \right\rfloor$
6      $C' \leftarrow$ cbTSP $(G_1, G_2, \ldots, G_n, F, C)$
7      if $C' \geq 0$ then
8          $C^+ \leftarrow C' - 1$
9          $C^* \leftarrow C'$
10     else
11         $C^- \leftarrow C + 1$
12 return $C^*$
```

Appendix B
cbTSP: Solver Parameters

The cbTSP solver can take in a number of parameters to configure the TSP solver, the SAT solver, and the cbTSP solver. This section details the cbTSP specific parameters as well as the default settings.

A. MiniSat

The configurable parameters of MiniSat are the conflict budget and the propagation budget. Both parameters by default are unlimited. All other MiniSat parameters are used as default and are nonconfigurable through the cbTSP interface.

TABLE IV
Default LKH Parameters

PRECISION	10	PATCHING_A	2
MOVE_TYPE	5	PATCHING_C	3
RUNS	1		

TABLE V
Experimental Results for Varying cb_interval

instance/`cb_interval`	1	2	10	BRUTE
patrolling06	**3176**	3182	3631	3631
patrolling10	**3442**	3908	4018	4120
sample04	**1883**	2089	4207	5324
sample08	**12 395**	13 380	16 157	-
period11	**2804**	2896	1387	4071
period12	**2669**	2697	3247	4078

B. LKH

The LKH parameters configured by cbTSP are: `PROBLEM_FILE`, `TOUR_FILE`, `TIME_LIMIT`, `STOP_AT_MAX_COST`, and `MAX_COST`. The last two parameters are customizations we added to allow for LKH to solve decision problems. All other LKH parameters are configurable, the default settings that deviate from the LKH default settings are given in Table IV.

C. cbTSP

This section documents the configurable parameters for the cbTSP solver. Default values are given in parenthesis.

1) The callback interval (`cb_interval = 1`)*:* This parameter configures the TSP callback interval. A setting of x indicates that the TSP theory consistency checked is performed when the following is satisfied: $|V| \mod x = 0$. A value of $x > |V|$, behaves, as a non-naïve BRUTE solver (described in Section IV-A). The default settings for the parameter `cb_interval` was chosen after comparing different values of `cb_interval` on a set of small experiments. We took two instances from each problem application (patrolling, sample collection, and period routing), six in total and compare the quality (cost) of the solver's results with the same setup as in Section VII. The results are given in Table V, which represents

TABLE VI
EXPERIMENTAL RESULTS FOR VARYING bdiv

instance/bdiv	2	10	20	Linear
patrolling06	3177	**3176**	3177	3177
patrolling10	3653	**3442**	3496	3972
sample04	1899	1883	**1882**	2291
sample08	12 993	12 395	**12 051**	14 731
period11	2925	**2804**	2937	3578
period12	2973	**2669**	2708	3148

the average of four runs. The best results are highlighted. Dashes indicate that the solver was not able to find a feasible solution. As we can see from the Table, a value of `cb_interval` $= 1$ has the best performance. One thing to note from the results is how the non-naïve BRUTE approach (`cb_interval` $= 999$) fails on sample instance 08. We believe this is due to the battery constraint aspect of the problem. In this problem, unlike the other two problems, the initial SAT solutions that the BRUTE approach would find are unlikely to represent feasible TSP solutions due to the battery budget and so the solver in this configuration wastes lots of time searching nonfeasible TSP solutions. Thus, without the guide of the TSP theory, the solver seems to struggle finding feasible solutions for the sample collection problem.

2) Conflicts (`nConflicts = -1`*):* The cbTSP solver adds conflict clauses back to the sat formula to narrow down the search space, as described in Section IV. The number of conflict clauses cbTSP adds back to the formula can be exponential in size, thus the parameter `nConflicts` $= x$ limits the number of additional clauses to x (-1 for unlimited) and once that number is reached, it returns with the best known solution. One can check the output text of the solver to confirm whether or not the budget was reached.

3) Query budget (`max_query_time = -1`*):* The cbTSP solver searches for solutions with a series of SAT-TSP decision queries, where each search has a different cost budget set by either a binary or linear search algorithm. By default (-1) the queries are given an unlimited amount of time, but if this parameter has a positive value of x then each query is terminated after x seconds and is assumed to be unsatisfiable.

4) Search method (`search_method = binary`*):* This parameter is used to choose between binary and linear search.

5) The binary search parameter (`bdiv = 10`*):* This parameter configures the division size of the binary search as shown in Algorithm 4. Due to the fact that most unsatisfiable instances are harder to prove than satisfiable instances, it is desirable to have this parameter larger than the nominal value of two. The default settings for the binary search parameter `bdiv` was chosen after comparing different values of `bdiv` on a set of small experiments. As with the `cb_interval` experiments, we took two instances from each problem application (patrolling, sample collection, and period routing), six in total and compare the quality of the solver's results with the same setup as in Section VII. The results are given in Table VI, which represent the average of four runs. The linear configuration (in the last column) represents linear search and is configured using a value of `bdiv = 999999`. The best results are highlighted. As we can see from Table VI, a value of `bdiv = 10` yields a good result.

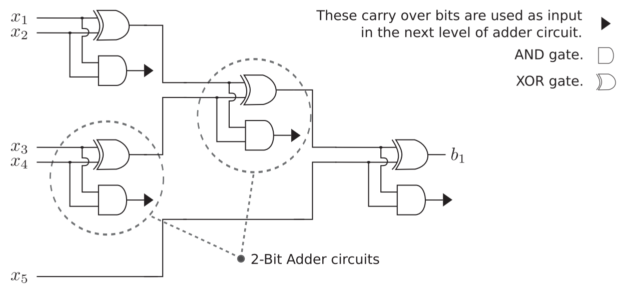
x_1
x_2
x_3
x_4
x_5
b_1
These carry over bits are used as input in the next level of adder circuit.
AND gate.
XOR gate.
2-Bit Adder circuits

Fig. 7. Adder circuit summing up Boolean input variables $X' = \{x_1, x_2, \ldots, x_5\}$ and outputting the Boolean variable b_1, as well all the carry out bits.

APPENDIX C
ADDER CIRCUITS

In this section, we provide an overview of how adder circuits are used to count the number of true variables in a set X'. In general, we can input the negation of a variable (its negative literal) but for this document we only count variable inputs if they are assigned true. Adder circuits are used in electronics to do rudimentary mathematical operations [52]. The example circuit shown in Fig. 7 takes as input $X' = \{x_1, x_2, \ldots, x_5\}$ and outputs b_1. This circuit is composed of four two bit adder circuits and correctly constrains the binary one bit b_1 to the number of true input variables in X', there would be a similar circuit for the twos bit that takes in as input all of the carry out bits from the example. The complete binary circuit can be constructed using techniques in [52]. The circuit is translated to a Boolean formula (SAT) in polynomial time using methods from [53].

REFERENCES

[1] J. Yu, S. Karaman, and D. Rus, "Persistent monitoring of events with stochastic arrivals at multiple stations," *IEEE Trans. Robot.*, vol. 31, no. 3, pp. 521–535, Jun. 2015.
[2] P. Tokekar, J. Vander Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic UAV and UGV system for precision agriculture," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1498–1511, Dec. 2016.
[3] G. A. Hollinger and G. S. Sukhatme, "Sampling-based robotic information gathering algorithms," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1271–1287, 2014.
[4] D. Portugal and R. P. Rocha, "Cooperative multi-robot patrol with Bayesian learning," *Auton. Robots*, vol. 40, no. 5, pp. 929–953, 2016.
[5] R. Wang, M. Veloso, and S. Seshan, "Active sensing data collection with autonomous mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2016, pp. 2583–2588.
[6] M. J. Gallant, A. Ellery, and J. A. Marshall, "Rover-based autonomous science by probabilistic identification and evaluation," *J. Intell. Robot. Syst.*, vol. 72, no. 3–4, pp. 591–613, 2013.
[7] M. Eich, R. Hartanto, S. Kasperski, S. Natarajan, and J. Wollenberg, "Towards coordinated multirobot missions for lunar sample collection in an unknown environment," *J. Field Robot.*, vol. 31, no. 1, pp. 35–74, 2014.
[8] A. Das *et al.* "Mapping, planning, and sample detection strategies for autonomous exploration," *J. Field Robot.*, vol. 31, no. 1, pp. 75–106, 2014.
[9] N. Christofides and J. E. Beasley, "The period routing problem," *Networks*, vol. 14, no. 2, pp. 237–256, 1984.
[10] L. Bertazzi and M. G. Speranza, "Inventory routing problems with multiple customers," *EURO J. Transp. Logistics*, vol. 2, no. 3, pp. 255–275, 2013.

[11] B. Yao, P. Hu, M. Zhang, and S. Wang, "Artificial bee colony algorithm with scanning strategy for the periodic vehicle routing problem," *Simulation*, vol. 89, no. 6, pp. 762–770, 2013.

[12] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[13] C. Xie, J. van den Berg, S. Patil, and P. Abbeel, "Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 4187–4194.

[14] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[15] F. Pasqualetti, A. Franchi, and F. Bullo, "On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms," *IEEE Trans. Robot.*, vol. 28, no. 3, pp. 592–606, Jun. 2012.

[16] "IBM ILOG CPLEX Optimizer Studio," 2019. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[17] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2018. [Online]. Available: http://www.gurobi.com

[18] N. Sorensson and N. Een, "MiniSat v1.13—A SAT solver with conflict-clause minimization," *SAT*, vol. 2005, p. 53, 2005.

[19] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *Eur. J. Oper. Res.*, vol. 126, no. 1, pp. 106–130, 2000.

[20] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, 2009, vol. 3.2, p. 5.

[21] F. Imeson and S. L. Smith, "A language for robot path planning in discrete environments: The TSP with Boolean satisfiability constraints," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 5772–5777.

[22] F. Imeson and S. L. Smith, "Multi-robot task planning and sequencing using the SAT-TSP language," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 5397–5402.

[23] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. New York, NY, USA: Dover, 1998.

[24] J. Park, S. Karumanchi, and K. Iagnemma, "Homotopy-based divide-and-conquer strategy for optimal trajectory planning via mixed-integer programming," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1101–1115, Oct. 2015.

[25] N. Mathew, S. L. Smith, and S. L. Waslander, "Multirobot rendezvous planning for recharging in persistent tasks," *IEEE Trans. Robot.*, vol. 31, no. 1, pp. 128–142, Feb. 2015.

[26] N. Kamra and N. Ayanian, "A mixed integer programming model for timed deliveries in multirobot systems," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, 2015, pp. 612–617.

[27] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Trans. Robot.*, vol. 32, no. 5, pp. 1163–1177, Oct. 2016.

[28] A. Pnueli, "The temporal logic of programs," in *Proc. IEEE Symp. Found. Comput. Sci.*, 1977, pp. 46–57.

[29] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Reading, MA, USA: Addison-Wesley, 2004, vol. 1003.

[30] A. Cimatti *et al.*, "NuSMV 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*. New York, NY, USA: Springer, 2002, pp. 359–364.

[31] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *Int. J. Robot. Res.*, vol. 30, no. 14, pp. 1695–1708, 2011.

[32] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Trans. Robot.*, vol. 26, no. 1, pp. 48–61, Feb. 2010.

[33] A. P. Sistla and E. M. Clarke, "The complexity of propositional linear temporal logics," *J. ACM*, vol. 32, no. 3, pp. 733–749, 1985.

[34] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artif. Intell.*, vol. 2, no. 3/4, pp. 189–208, 1971.

[35] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains." *J. Artif. Intell. Res.*, vol. 20, pp. 61–124, 2003.

[36] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *J. Artif. Intell. Res.*, vol. 14, pp. 253–302, 2001.

[37] S. Richter and M. Westphal, "The LAMA planner: Guiding cost-based anytime planning with landmarks," *J. Artif. Intell. Res.*, vol. 39, no. 1, pp. 127–177, 2010.

[38] K. Erol, D. S. Nau, and V. S. Subrahmanian, "Complexity, decidability and undecidability results for domain-independent planning," *Artif. Intell.*, vol. 76, no. 1, pp. 75–88, 1995.

[39] Y. Shoukry *et al.* "Scalable lazy SMT-based motion planning," in *Proc. 55th Conf. Decis. Control*, 2016, pp. 6683–6688.

[40] W. N. Hung *et al.* "Motion planning with satisfiability modulo theories," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 113–118.

[41] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki, "SMT-based synthesis of integrated task and motion plans from plan outlines," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 655–662.

[42] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe LTL specifications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2014, pp. 1525–1532.

[43] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Implan: Scalable incremental motion planning for multi-robot systems," in *Proc. 7th Int. Conf. Cyber-Physical Syst.*, 2016, pp. 1–10.

[44] F. Imeson, "Robotic path planning for high-level tasks in discrete environments," Ph.D. dissertation, Univ. Waterloo, Waterloo, ON, Canada, Apr. 2018. [Online]. Available: http://hdl.handle.net/10012/13082.

[45] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)," *J. ACM*, vol. 53, no. 6, pp. 937–977, 2006.

[46] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. 38th Annu. Des. Autom. Conf.*, 2001, pp. 530–535.

[47] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, 1991.

[48] H. H. Hoos and T. Stützle, "SATLIB–The satisfiability library," 1998. [Online]. Available: http://www.satlib.org

[49] L. C. Coelho and G. Laporte, "The exact solution of several classes of inventory-routing problems," *Comput. Operat. Res.*, vol. 40, no. 2, pp. 558–565, 2013.

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

[51] G. Best, J. Faigl, and R. Fitch, "Multi-robot path planning for budgeted active perception with self-organising maps," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 3164–3171.

[52] M. M. Mano and M. D. Ciletti, *Digital Design*, 4th ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2006.

[53] P. Jackson and D. Sheridan, "Clause form conversions for Boolean circuits," in *Proc. Theory Appl. Satisfiability Testing*, 2005, pp. 183–198.

Frank Imeson received the B.Sc. degree in physics, the M.A.Sc. degree in electrical and computer engineering, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2003, 2013, and 2018, respectively.

He worked in the automation industry from 2003 to 2011. His main research interests include artificial intelligence for robotics. Specifically, he has been developing high level robotic controls for solving complex optimization path planning problems.

Stephen L. Smith (S'05–M'09–SM'15) received the B.Sc. degree in engineering physics from Queen's University, Kingston, ON, Canada, in 2003, the M.A.Sc. degree in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada, in 2005, and the Ph.D. degree in mechanical engineering from UC Santa Barbara, Santa Barbara, CA, USA, in 2009.

From 2009 to 2011, he was a Postdoctoral Associate in the Computer Science and Artificial intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA. He is currently an Associate Professor in Electrical and Computer Engineering, University of Waterloo, Canada, and a Canada Research Chair in autonomous systems. His main research interests include control and optimization for autonomous systems with an emphasis on robotic motion planning and coordination.