2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Assigning Multi-Robot Tasks to Multitasking Robots

Winston Smith and Yu Zhang

Abstract— **One simplifying assumption in existing and well-performing task allocation methods is that the robots are single-tasking: each robot operates on a single task at any given time. While this assumption is harmless to make in some situations, it can be inefficient or even infeasible in others. In this paper, we consider assigning multi-robot tasks to multitasking robots. The key contribution is a novel task allocation framework that incorporates the consideration of physical constraints introduced by multitasking. This is in contrast to the existing work where such constraints are largely ignored. After formulating the problem, we propose a compilation to weighted MAX-SAT, which allows us to leverage existing solvers for a solution. A more efficient greedy heuristic is then introduced. For evaluation, we first compare our methods with a modern baseline that is efficient for single-tasking robots to validate the benefits of multitasking in synthetic domains. Then, using a site-clearing scenario in simulation, we further illustrate the complex task interaction considered by the multitasking robots in our approach to demonstrate its performance. Finally, we demonstrate a higher-complexity simulation to demonstrate the scalability and applicability of our approach.**

I. Introduction

In prior work on task allocation, it has been a near-omnipresent assumption that robots are single-tasking [1], [2]. When considering task allocation practically, it is often more convenient to have robots work on multiple tasks simultaneously to improve efficiency, and may be required in some situations. For example, consider a scenario where a robot needs to swipe a keycard to unlock a door temporarily and also open that door at the same time. We note that it is not always possible for multiple robots to "divide and conquer" these tasks due to a space constraint; under certain conditions (e.g., robots with large footprints), *one* robot must do *both* tasks since two robots cannot fit in the space around the door simultaneously. The ability for a multi-robot system to automatically identify this infeasibility so that a bi-manual robot can be assigned instead of two robots is missing. Redesigning this scenario such that there is a single "swipe keycard and open door" task to force the task to be assigned to a single robot is possible, but this approach requires the multitasking solution to already be known so that we can combine each set of tasks that require multitasking into a single task. A generic solution for multi-robot tasks with multi-tasking robots requires the automatic decomposition and combination of tasks, which goes beyond simply adding up or splitting task requirements and is even more complicated. Hence, compiling such problems to problems with single-tasking robots is utterly impractical in general.

Winston Smith and Yu Zhang are with the School of Computing and Augmented Intelligence, Arizona State University, Tempe, USA {wtsmith7, yzhan442}@asu.edu.

The key observation here is that achieving multitasking robots with multi-robot tasks is not only about the ability of each robot to perform its tasks or parts of the tasks assigned to it (in the case of multi-robot tasks) in terms of having appropriate sensors, motors, etc., but also about the ability of each robot to perform its assignments in terms of *physicality* – that is, whether these assignments introduce physical constraints that prevent the robot from achieving them simultaneously (in the case of multitasking) or hinder robots from cooperating on a task (in the case of multi-robot tasks). In the previous example, the space constraint, a type of physical constraint, introduced the required multitasking. At the same time, multitasking can introduce other physical constraints. Following from above, to be able open the door, a bi-manual robot must be flexible enough to use one arm for swiping the keycard and another arm for opening the door simultaneously, which imposes a constraint on the arm span.

Hence, the key to enable multitasking with multi-robot tasks lies in the ability to identify and exploit synergies among these physical constraints to ensure their compatibility. A significant challenge is that constraints can be inter-dependent, meaning that they may interact with others in complex ways, such as implying other constraints recursively. For example, if the robot has a fixed camera for vision, a constraint on the robot's viewing angle as a result of the keycard task would constrain the robot's facing and one axis of its position. This complexity makes it substantially more challenging to anticipate the influence of a task assignment on future assignments when multitasking is considered with multi-robot tasks, not to mention the significantly grown number of candidate assignments. Prior works on multitasking robots with multi-robot tasks [3], [4] ignore the inter-dependencies among constraints and hence can introduce invalid solutions.

To the best of our knowledge, this work represents the first framework for assigning multi-robot (MR) tasks to multitasking (MT) robots with instantaneous assignment (IA) (MT-MR-IA in [2]) while considering physical constraints. It is a significant extension of prior work [5] for determining task feasibility under a given set of task assignments to robots. Since the mechanism for considering the physical constraints with multitasking robots and multi-robot tasks are the same, we focus on the more unique multitasking capability in the following discussion. We present a general problem definition, explore a compilation approach to the weighted MAX-SAT problem first, and then a greedy method. Simulation results in synthetic domains demonstrate that our approach can handle medium-size problems effectively. In particular, we show that a modern baseline for task allocation with

979-8-3315-8160-2/26/$31.00 ©2026 IEEE

single-tasking robots performs poorly compared with our method, verifying the advantage of multitasking. Using a site clearing scenario in simulation, we further illustrate the complex task interactions between multitasking robots that can be handled by our approach to demonstrate its capability. Finally, we show how our approach leads to increased task efficiency in a complex setting with realistic simulations.

II. Related Work

The multi-robot task allocation (MRTA) problem has been a subject of prior research, reviewed in [2], [1]. For multi-robot systems, one class of problems of particular interest involves multi-robot tasks (MR). Since even the simplest subclass that addresses single-tasking robots (ST) and instantaneous assignments (IA) is NP-hard, the focus has been on approximate solutions or heuristic methods [3], [6], [7], [8], [9], [10], [11], [12], [13]. The more general subclass with multitasking robots (MT) has not seen nearly as much examination, with some notable exceptions [3], [4], [14], [15], [11]. Some recent works also approach the MT-MR domains. For example, [16] solves the MT-MR-TA problem as posed in [2], and [17] considers precedence constraints between tasks. None of these prior works are general enough under the presence of physical constraints, which are in addition to resource or precedence constraints to fulfill task requirements and can introduce complex task dependencies.

Physical constraints may assume various forms, and are especially prevalent with multi-robot tasks and multitasking robots. Co-location constraints on sensor and motor capabilities, and spatial constraints due to physical presence, are examples of physical constraints. Most prior works on task allocation (especially with multi-robot tasks) only implicitly consider physical constraints and thus have limited applicability. For example, previous works have considered overlapping coalitions [18], [19] but assume the influences of these constraints are captured by the utilities of coalitions. That approach not only imposes a challenge on the domain designer but also implies that the constraints are independent across assignments, which is restrictive.

Co-location constraints have been explicitly considered in [6] and generalized in [14], [15], [11]. However, one common limitation of these prior works is that they all ignore the inter-dependencies between these constraints, which can result in infeasible solutions. For multitasking, in particular, its feasibility depends critically on the compatibility of these constraints in complex and interactive ways, which are required to handle multiple tasks simultaneously. Although multitasking robots have been considered before, such as in [4], no physical constraints were considered. A prior work [5] that looked into generally formulating physical constraints, however, operates under a restricted problem setting where the set of assignments is given, which we will extend in this work to create and optimize assignments. A general approach to considering physical constraints would enable a multi-robot system to express real-world problems and generalize to novel scenarios more effectively.

The way in which physical constraints are formulated in this work resembles how works in action languages [20], [21], [22], [23] deal with indirect action effects in planning, but our work examines these effects in robot task allocation: these prior works only examine single robots, whereas ours allows for multiple agents. [24] and [25] applied a similar concept for event modeling, which is applied to logic programs. In addition, the indirect action effects in these prior works are generally not allowed to interact with each other.

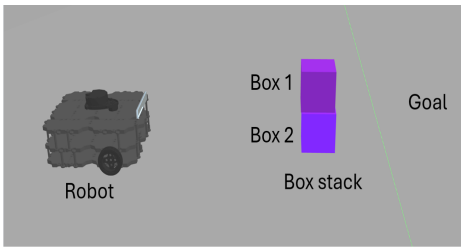

Fig. 1: Running example in ROS with the boxes stacked.

III. Approach

A. Running Example

As a running example, we will examine a scenario in which a robot must push several boxes to a goal area (see Fig. 1); each box corresponds to a single task. The two boxes are initially on the ground. Consider a situation where the robot only has enough battery power for one trip, or where the robot must push the boxes through a one-way door. In such a case, assuming the robot is capable of pushing both boxes at the same time, the robot is required to perform multitasking by stacking one box on top of the other and pushing them together by pushing the bottom box. While this may be intuitive from a human's perspective, it involves substantial challenges that are representative of both the synergistic and restrictive aspects in multitasking. The robot must:

1) (***Synergistic***) Understand that multiple tasks can be achieved simultaneously, e.g., when pushing the bottom box, the top box is also pushed if they are stacked;
2) (***Restrictive***) Identify and respect any additional constraints as a result of multitasking, e.g., having a box on top of the bottom box effectively increases the weight of the bottom box and the pushing power required.

With existing approaches, considering these aspects would require building the conditions of each aspect above into problem specifications on a problem-by-problem basis: e.g., combining the two tasks by redefining the box stack to be a single task with an updated weight. This becomes infeasible when there are more boxes and robots, boxes have different weights, heavy boxes should not be stacked on lighter ones, and robots have different pushing powers. In this paper, we introduce a formalism that represents such knowledge in a *problem-independent* way. As a result, our formalism and solution methods together allow robots to flexibly reason about and solve novel scenarios in the same domain.

We assume that this knowledge is provided by users and leave learning it to future work. To focus on the task allocation challenge, we assume that the initial state of any given problem can be converted to a set of candidate initial

states that are conducive to instantaneous assignment (IA). For example, Fig. 1 could be a candidate initial state for the problem initial state where both boxes are on the ground. This means that, before task execution, a preprocessing step must exist that can stack the boxes after task allocation is made. These steps require the consideration of complementary tasks as well as time-extended assignment. Thus, it is more natural to address them in future work when we extend our formulation to time-extended domains.

We build our work on [5] to model physical constraints, even though only the restrictive aspect above was considered there. Our contributions are a) extending the formalism from [5] to consider synergies so that robots can reason about *how* to perform multiple tasks simultaneously (instead of being told how), b) providing the first general solution method for the MT-MR-IA problem considering physical constraints, and c) evaluating the effectiveness of multitasking in synthetic and simulated domains.

B. Preliminaries & Prior Work

A predicate in logic is a boolean function that represents a property or relation [26]. It may evaluate to true or false depending on its arguments. In the planning community, predicates are used to specify states or partial states. In the running example, the predicate $F_{On}(o_1, o_2)$ is used to indicate whether or not o_1 is on top of o_2. o_1 and o_2 are both arguments to the predicate F_{On}, also called *referents*, which can remain uninstantiated, indicated by capital letters, or instantiated to domain elements, indicated by lowercase letters with numeric subscripts, which are mapped to objects in the environment. In Fig. 1, $F_{On}(o_1, o_2)$ is true when o_1 is mapped to Box 1 and o_2 to Box 2.

A physical constraint is a constraint on part of the physical configuration of the world. We conservatively assume that a single physical constraint may arbitrarily restrict the *physical configuration* that satisfies the constraint. For example, a constraint on $F_{On}(o_1, o_2)$ restricts o_1 to be on o_2. The exact physical configuration that satisfies o_1 on o_2, however, may vary arbitrarily over time (i.e., where o_1 is on o_2 exactly). As a result, we do not allow two physical constraints to apply to the same predicate since they could require incompatible physical configurations to be assumed. For example, two physical constraints on the location of a box could require the box to be placed at different locations at the same time. Such a stipulation is *conservative* since two constraints on the same predicate may still be compatible, e.g., when both constraints require the box to always be at exactly the same location or be at different locations at different times.

It may be the case that a set of constraints implies additional constraints that are not in the original set. These must also be considered. For example, a constraint on $F_{Pos}(o_2)$ (i.e., the position of o_2) and another on $F_{On}(o_1, o_2)$ together imply a constraint on $F_{Pos}(o_1)$. To capture this, we use CIRs, which are analogous to inference rules from [5]:

Definition 3.1 (Constraint Implication Rule (CIR)): Given a set of predicates S and a predicate f, a constraint implication rule specifies a relationship where the constraints on all of S imply a constraint on f, written as $S \rightarrow f$.

For example, the example CIR described above could be written in its uninstantiated form as:

$$\{F_{Pos}(Y), F_{On}(X, Y)\} \rightarrow F_{Pos}(X)$$

Note that the same (different) uninstantiated referent label must be instantiated to the same (different) domain element in a CIR, e.g., Y must instantiate to the same domain element for a given application of the CIR above. CIRs may be given as uninstantiated or partially instantiated, but all referents must be instantiated when applying a CIR. A CIR with uninstantiated or partially instantiated predicates thus specifies a set of instantiated CIRs. We require any referents to f to appear in S - that is, any set of constraints cannot imply a constraint on new domain elements that are not referenced in that set of constraints. A CIR is automatically triggered whenever constraints corresponding to its left hand side are all present. When specifying CIRs, it is important to consider the physical processes behind how the environment acts upon itself, and to specify the effects of these processes in an abstract way as opposed to being exact.

Definition 3.2 (Minimally Implying Subset [5]): A minimally implying subset M of a set of predicates S is any subset of S such that removing any element from M makes it no longer contain or imply all predicates in S.

Finally, we define compatibility, which determines whether a set of constraints can be satisfied simultaneously:

Definition 3.3 (Compatibility [5]): A set of constraints S is compatible iff no constraint in S is implied by more than one unique minimally implying subset of S.

C. Problem Definition

We may now provide a general formulation of our MT-MR-IA task allocation problem, referred to as "Task Allocation for Multitasking robots under Physical Constraints" or "TAMPiC" for short.

Definition 3.4 (TAMPiC): An instance of TAMPiC is a tuple $(F, Q, R, C, T, I, \Delta)$ defined as follows:

- F: The set of all unique instantiated predicates.
- Q: A set of CIRs $\{q_l\}$. By definition, each q_l takes the form $S_l \rightarrow f_l (S_l \in 2^F, f_l \in F)$.
- R: A set of robots $\{r_i\}$. Each r_i has a set of capabilities $\{c_j\}$ and each c_j has an associated set of predicates $P_j = \{f_k \in F\}$, denoted as $c_j \rightarrow P_j$, that will become constrained if c_j is activated and a cost k_j expressed as a nonnegative integer which will be subtracted from solution utility if c_j is activated. Each k_j is assumed zero if not specified.
- T: A set of tasks $\{t_m\}$. Each $t_m = (Y_m, u_m)$ where Y_m is a nonempty set of predicates to be constrained and/or capabilities to be activated in order to fulfill the task, and u_m is the utility for fulfilling the task.
- I: A set of fully instantiated predicates $I \in 2^F$ representing constraints present in the initial state.
- Δ: A function that returns a set of candidate initial states that are conducive to instantaneous assignment, given the problem initial state: $\Delta(I) = \{\delta\}, \delta \in 2^F$.

Then, the goal of the problem is to find $\delta_{max} \in \Delta(I)$ that maximizes the following objective:

$$\text{maximize} \sum_m x_m u_m - \sum_{i,j} y_{i,j} k_{i,j}$$

s.t. the set of all constraints from the activated capabilities based on δ_{max} must be compatible, where x_m = 1 if t_m is assigned or 0 otherwise and $y_{i,j}$ = 1 if c_j is activated on r_i or 0 otherwise.

The definition above bears some similarities to the planning problem. However, states can change over planning steps; for task allocation with instantaneous assignments, we only take a "snapshot" at a specific planning step. Each δ can be considered as moving the step at which the snapshot is taken. Note how such a definition prepares us to consider extended assignments in future work. In addition, CIRs may appear to be similar to actions/operators in planning. From this perspective, in contrast to actions, CIRs are passively activated whenever the required preconditions are satisfied. This observation makes CIRs more akin to derived predicates or events in PDDL+ [27], though adapted for multiple agents.

Since we consider instantaneous assignments in this work, we consider the constraints associated with a task for the entire duration. The domain-dependent function Δ is assumed to be given and applied once before task allocation to update the initial state to make it conducive to instantaneous assignment. When multiple candidate initial states (after update) are possible, we will check for each separately. Addressing time extended assignment in future work will eliminate the need for Δ entirely. Note that Δ only updates the initial state for task allocation purposes; it does not handle the steps required to stack the boxes in the physical environment. To simplify the following discussion, we assume that the boxes are initially stacked as in Fig. 1 and applying Δ returns a set of a single element that is the problem initial state (i.e., $\Delta(I) = \{\delta\}$). We may now give a formal specification of our running example:

$R = \{r_1\}$, $r_1 = \{C_{Push}(r_1, X), C_{StrongPush}(r_1, X)\}$

This specifies that the single robot in the scenario has two capabilities: $Push$ and $StrongPush$, each taking the robot itself and the object to be pushed as arguments. We specify capabilities with their owner robot as the first argument to differentiate the same capabilities on different robots. There are two capabilities $C = \{C_{Push}(X, Y), C_{StrongPush}(X, Y)\}$:

$C_{Push}(X, Y) \rightarrow$

$F_{Pos}(X) \wedge F_{Pos}(Y) \wedge \neg F_{Weight+}(Y) \wedge \neg F_{On}(Y, Z)$

$C_{StrongPush}(X, Y) \rightarrow F_{Pos}(X) \wedge F_{Pos}(Y) \wedge \neg F_{On}(Y, Z)$

The positions of both the pusher robot and the object being pushed are constrained, and the object being pushed must be not on top of another object. $Push$ also requires the object to be not heavy (i.e., $\neg F_{Weight+}(Y)$). $T = \{t_1, t_2\}$:

$t_1 = (\{F_{Pos}(o_1)\}, u_1 = 1)$, $t_2 = (\{F_{Pos}(o_2)\}, u_2 = 3)$

There are two tasks considered. The first task requires the position of a specific object (o_1) to be constrained (i.e., it must be moved to a specific position). The utility is 1. The second task is similarly defined with utility 3. There are two CIRs $Q = \{q_1, q_2\}$:

$q_1 = \{F_{On}(X, Y), F_{Pos}(Y)\} \rightarrow F_{Pos}(X)$

$q_2 =$

$\{F_{On}(X, Y), F_{Weight}(Y), F_{Weight}(X)\} \rightarrow F_{Weight+}(Y)$

The first rule specifies that X being on Y constrains X's position if Y's position is constrained. This rule is used to infer that a stack of objects can be pushed by pushing only the bottom object, resulting in task synergies. See Sec. III-A Item 1). The second rule specifies that if X is on top of Y, and both objects have a weight, the effective weight of Y is increased. This rule suggests the additional constraints introduced by multitasking. See Sec. III-A Item 2). The initial state is:

$$I = \{F_{On}(o_2, o_1), F_{Weight}(o_1), F_{Weight}(o_2)\}$$

In the initial state, o_2 is on top of o_1; both objects have a weight. This means the robot cannot directly push o_2 because it is out of reach. Single-tasking approaches would fail to recognize that t_2 can be completed at all. In TAMPiC, this can be accomplished by activating the $C_{StrongPush}(r_1, o_1)$ capability to push both boxes simultaneously.

D. Solution Method

The MT-MR-IA problem without considering physical constraints, equivalent to the well-known NP-complete Set Covering Problem [2], is a special case of TAMPiC when $Q = \varnothing$ and $\Delta(I) = \{I\}$. Thus, TAMPiC is NP-hard. Despite the computational challenge, we explore a general process for converting instances of TAMPiC to Weighted MAX-SAT, which is a well-studied NP-complete problem and has efficient solutions in practice. The converted problem can be given to any Weighted MAX-SAT solver. We provide a definition of Weighted MAX-SAT, similar to [28]:

Definition 3.5: The Weighted MAX-SAT problem is a tuple (V, S, W) where V is the set of variables, S is a set of disjunctive clauses of literals of V, and W is a set of non-negative weights assigned to each clause in S.

A literal refers to a variable v or its negation $\neg v$. The goal of Weighted MAX-SAT is to find truth values for variables in V to maximize the combined weight of the satisfied clauses in S. (s, w) denotes a clause and its associated weight.

The full process of conversion and querying the SAT solver for a solution to a TAMPiC problem is referred to as SAT-based Task Assignment with Multitasking Robots, or STAMR. Throughout the conversion, unless otherwise specified, every clause has weight α that is strictly greater than the sum of the utilities of all tasks. α is used to ensure the satisfaction of these clauses. Alternatively, many Weighted MAX-SAT solvers allow for "hard" clauses, which must be satisfied. Either method may be used in practice. F, C, I, and Δ: F and C are redundant given the other components for ease of reference so we do not discuss their conversion. For I, and Δ, We will need to apply the conversion for each candidate initial state in $\Delta(I)$.

δ: Each predicate in $\delta, \delta \in \Delta(I)$, introduces its own clause.

Q: To convert CIRs, a clause for each possible instantiation of each CIR must be created. We will use a certain Boolean identity extensively for the entire STAMR conversion process: $(p \rightarrow q) \equiv (\neg p \vee q)$. We use this equivalence to ensure that the Weighted MAX-SAT formula is in the

conjunctive normal form (CNF), which is required for many Weighted MAX-SAT solvers. For example, one instantiation of q_1 is:

$$\{F_{On}(o_1, o_2), F_{Pos}(o_2)\} \rightarrow F_{Pos}(o_1)$$

and the corresponding weighted clause would be:

$$(\neg F_{On}(o_1, o_2) \vee \neg F_{Pos}(o_2) \vee F_{Pos}(o_1))), \alpha)$$

R: Each r_i is defined as a set of capabilities. To convert capabilities, create a clause for each instantiated capability that requires the predicates constrained by that capability to be true if the capability is activated. For example, robot r_1 has a capability $C_{StrongPush}(r_1, Y)$, and one instantiation of this capability might be $C_{StrongPush}(r_1, o_1)$. Then the instantiated capability can be converted to:

$$((\neg C_{StrongPush}(r_1, o_1) \vee (F_{Pos}(r_1) \wedge F_{Pos}(o_1) \wedge \neg F_{On}(o_1, o_2)), \alpha)$$

which can then be converted to multiple disjunctive clauses using the distributive law, each with weight α.

To handle capability costs, however, we must make use of the following theoretical result since MAX-SAT does not natively support costs (negative weights). First, we introduce the following notations. Let $\beta = \{\{s_1, w_1\}, \{s_2, w_2\}, ...\}$ for a given weighted MAX-SAT problem. Note that V is implied given S and W in Def. 3.5. We define $\rho(\{s_i, w_i\}) = \{\neg s_i, -w_i\}$ and denote an interpretation of all the predicate variables in β as X. We further define $opt(\beta, X) = 1$ if X is an optimal solution to β or 0 otherwise. Let β' be β after applying ρ to an element in β.

Theorem 3.1: $opt(\beta, X) = opt(\beta', X)$.

The proof is given in a longer version of this paper [29]. We can apply the above theorem multiple times to apply ρ to multiple clauses. Note that negating a clause in CNF causes that clause to potentially no longer be in CNF. While it is trivial to convert the negated clause back to CNF by breaking it into multiple clauses, the weight of each resulting clause cannot be easily resolved. To address this problem, we attach the weight to a marker clause for each instantiation of each capability to denote if it is activated:

$$\{c_{i,j}, -k_{i,j}\}$$

The marker clauses are then transformed according to ρ. Note the change of sign above w.r.t. the original problem.

T: To convert tasks, there are two steps for each task. First, create a clause with weight equal to the task's utility and which contains only an assignment literal corresponding to whether the task is assigned. For example, for t_1, the corresponding Weighted MAX-SAT clause would be:

$$((t_1), 1)$$

Then, similar to Q, create a clause for each instantiation of each task, which forces the predicates for that instantiation to be true, and also another clause which forces at least one of the instantiations to be satisfied if the task's assignment literal is true. In the case of t_1, it only has a single instantiation.

$$((\neg t_{1,1} \vee F_{Pos}(o_1)), \alpha), ((\neg t_1 \vee t_{1,1}), \alpha)$$

Finalizing: To prevent constraints being created without the corresponding capabilities or CIRs being activated to create them, each constraint must imply at least one of the ways in which it could have been generated. For example, we have $((\neg F_{Pos}(o_2) \rightarrow (C_{Push}(r_1, o_2) \vee C_{StrongPush}(r_1, o_2) \vee q_1)), \alpha)$, which can be converted into:

$$((\neg F_{Pos}(o_2) \vee C_{Push}(r_1, o_2) \vee (C_{StrongPush}(r_1, o_2) \vee (F_{On}(o_1, o_2) \wedge F_{Pos}(o_2)), \alpha)$$

Additionally, each CIR and capability must set all other CIRs and capabilities (including different instantiations) that constrain the same predicates to false to prevent incompatibility (Def. 3.3). For example:

$$((\neg C_{Push}(r_1, o_2) \vee (\neg C_{StrongPush}(r_1, o_2) \wedge \neg(F_{On}(o_1, o_2) \wedge F_{Pos}(o_2))), \alpha)$$

The number of clauses in the Weighted MAX-SAT problem corresponding to each TAMPiC instance is on the order of the k-permutation of the set of possible instantiations for each argument to any predicate in F, where k is the maximum number of arguments any predicate takes. Because k is a domain constant, the conversion is polynomial w.r.t. the size of the TAMPiC instance. The converted problem can be given to any Weighted MAX-SAT solver and the resultant solution gives us a solution to the original TAMPiC problem.

Theorem 3.2: The compilation solution is both sound and complete for TAMPiC, assuming the Weighted MAX-SAT solver used is also both sound and complete.

The proof is given in a longer version of this paper [29].

Corollary 3.3: TAMPiC is NP-complete.

It follows immediately given that TAMPiC is in NP and a solution to TAMPiC is a solution to Weighted MAX-SAT and vice-versa based on the conversion (reduction).

E. Greedy Heuristic

We also propose an approximation method using a simple greedy heuristic. It chooses the task with the highest utility, and only converts TAMPiC with that single task into the corresponding weighted MAX-SAT problem and solves it. Then, it retrieves any implied constraints from the solution, and incorporates these constraints into future iterations as clauses with weight α, which simply state the presence of these constraints. This process is repeated for each task, ordered from the highest utility to the lowest. Any tasks that were successfully fulfilled are fulfilled in the same way in the returned solution for the original problem. This method is referred to as STAMR-Greedy, or STAMR-G for short.

IV. Results

For evaluation, we first evaluate our methods in synthetic domains and compare with a baseline to validate the benefits of multitasking. We then further illustrate the complex task interaction among the robots using a site-clearing scenario in simulation, similar to our running example. Finally, we demonstrate a higher-complexity simulation to show the scalability and applicability of our approach. In all cases, we only list key components of the problem for brevity.

A. Synthetic Evaluation

Our baseline is the *ResourceCentric* method from [13] that assumes single-tasking robots since it has been shown to outperform other commonly used methods for multi-robot task allocation with IA. We will use "RC" to refer to this method. Because our problem definition poses a hurdle to

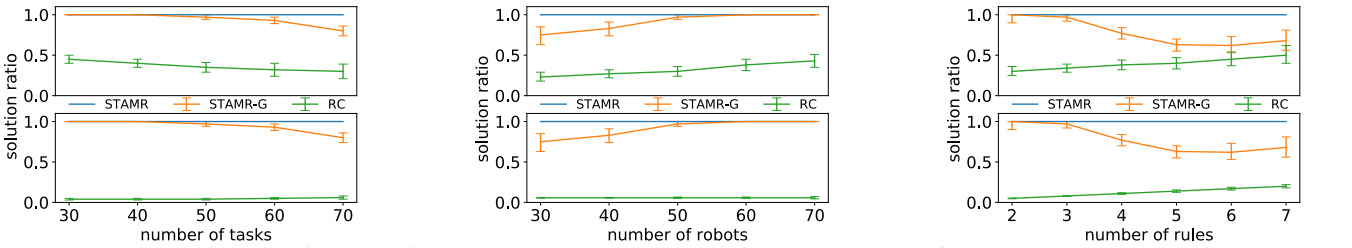

Fig. 2: Solution ratio as # of tasks (left), robots (center), and CIRs (right) increases in first (top) and second (bottom) setting.

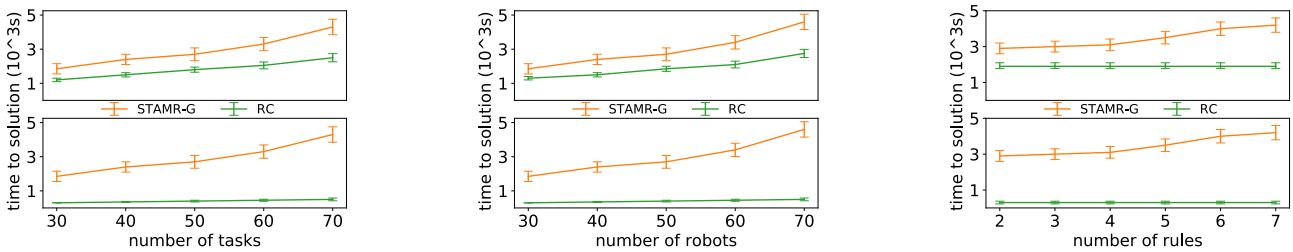

Fig. 3: Time taken as # of tasks (left), robots (center), and CIRs (right) increases in the first (top) and second (bottom) setting. The solution time for STAMR is not plotted since it required substantially more time than the others.

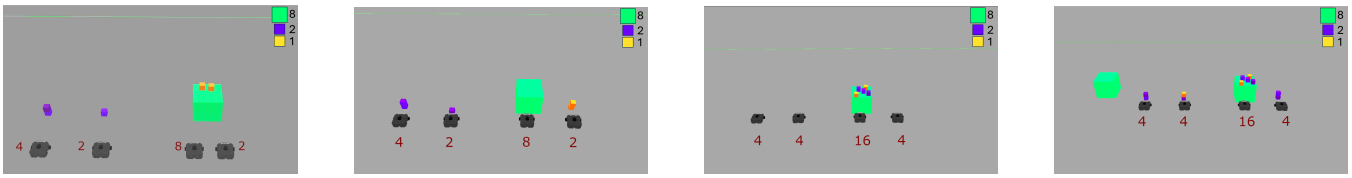
Fig. 4: Four site clearing scenarios with one of their optimal solutions illustrated, respectively.

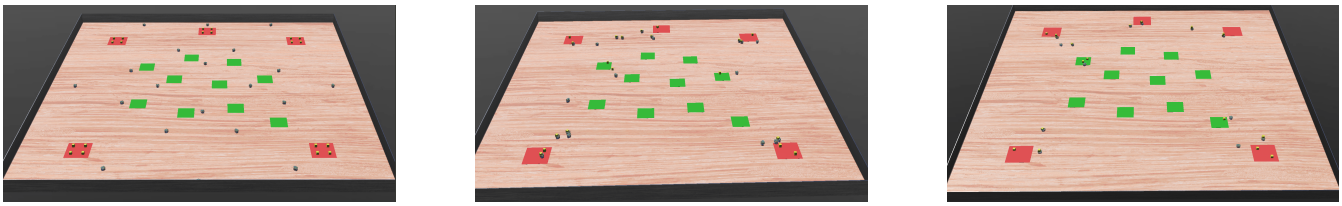
Fig. 5: The product delivery scenario in its initial state. The entire simulation environment (left). The baseline, midway through execution; note the clumps of robots (center). Our congestion-aware approach (right).

running RC, we evaluated RC in two different settings. The issue here is that tasks in TAMPiC may be specified as requirements for constraints, such as moving the box expressed as a constraint on the position of the box. RC is only able to consider tasks that only require capabilities to be activated since dealing with constraints requires CIRs. Hence, we evaluated RC in two settings. In the first setting, tasks generated did not require any constraints, whereas in the second setting, each requirement of each task had an equal chance to be a capability or a constraint. For the second setting, because RC cannot consider tasks with constraints, all such tasks are discarded before running RC.

We analyzed randomly generated problem instances. Because calculating the optimal solution is infeasible for large problems, we limited our examination to medium-sized problem instances. First, at most five unique uninstantiated predicates were generated ($1 \leqslant |F| \leqslant 5$), each with one or two arguments. Each problem had $|T| = 50$, each t_m with $1 \leqslant |Y_m| \leqslant 3$ and $1 \leqslant u_m \leqslant 30$. Each problem had $|R| = 50$, each r_i with $1 \leqslant |\{c_j\}| \leqslant 3$, and each type of capability constrained one or two predicates ($1 \leqslant |P_j| \leqslant 3$). Each problem had $|Q| = 2$, each q_l with $1 \leqslant |S_l| \leqslant 3$. $\Delta(I) = \{I\}$. In all cases where a range of possible outcomes is given, random generation was used with equal chance given to each outcome. For each evaluation setting, we used the parameter ranges as discussed above by default, except for the parameter that we chose to vary.

We tested the effects of increasing the number of tasks, robots and CIRs on the three methods (RC, STAMR, STAMR-G) in both settings. We define "solution ratio" as the utility achieved by the given approach divided by the utility achieved by the optimal solution. For each data point, 100 randomized runs were performed. The results are in Fig. 2 and Fig. 3. Note that the difference in settings had a negligible effect on the performance of STAMR and STAMR-G; this is expected since STAMR and STAMR-G handle capability- and constraint-based tasks identically. As more tasks were added, the relative performance of STAMR-G and RC dropped, likely due to the increased difficulty in optimization given that more tasks were available but only some of them could be assigned. Increasing the number of robots yielded the opposite effects. RC's performance

in the second setting was consistently weak such that little difference was observed as more tasks or robots were added.

Finally, we tested all three methods in both settings with varying number of CIRs. Increasing the number of CIRs worsened the relative performance of STAMR-G initially and, unexpectedly, *improved* it as the number continued to climb. As the number of CIRs increased, the restrictive aspects played a more substantial role, limiting multitasking such that RC approached STAMR and STAMR-G.

B. Site Clearing

We used STAMR with simulated robots in site-clearing scenarios where we solved a more complex version of our running example (Fig. 4). We used a green line at the top to denote a "point of no return" for the robots: they may each only cross the line once. Shown in Fig. 4 are scenarios involving four robots and six boxes (except for the last). The box weights (black) and robot pushing powers (red) are labeled in the figure. We assume that pushing powers are additive. Every box is associated with its own task and gives the same utility. The problem is more formally specified as follows for the first scenario (leftmost) in Fig. 4:

Q: contains CIRs to simulate addition of weights down stacks of boxes. For example: $\{F_{On}(X,Y), F_{Weight1}(X), F_{Weight2}(Y)\} \rightarrow F_{Weight3}(Y)$: stacking a box of weight 1 on top of a box of weight 2 results in an effective weight of 3 for the bottom box. Each combination of two weights totaling less than or equal to 16 is simulated in this way. Similarly, Q contains a second type of CIRs for pushing. For example, $\{F_{PushHard}(X,Y), F_{Push}(Z,Y), F_{Weight10}(Y)\} \rightarrow F_{Moved}(Y)$: a strong push and a normal push together are sufficient for a box of weight 10. Finally, $\{F_{On}(X,Y), F_{Moved}(X)\} \rightarrow F_{Moved}(Y)$, as before.

R: $r_1 = (C_{Push}(r_1, X))$, $r_2 = (C_{PushWeak}(r_2, X))$, $r_3 = (C_{PushHard}(r_3, X))$, $r_4 = (C_{PushWeak}(r_4, X))$

Δ: produces candidate initial states; each scenario in Fig. 4 corresponds to a chosen candidate initial state.

Assume all the boxes are on the ground initially. Under the box weights and robot pushing powers shown in the first scenario (leftmost figure) in Fig. 4, one of the optimal solutions involves stacking the boxes as shown in the same figure before task allocation (via Δ). Next, we show that 1) our approach can generate various solutions under different candidate initial states to show its flexibility, and 2) our approach can generalize to novel scenarios with changes to the original problem. We first examined the effect of a different candidate initial state (second scenario). Our approach was flexible enough to generate both solutions even though they were quite different (i.e., one involving multi-robot tasks and the other one with only single-robot tasks). In the third scenario, we changed the pushing powers of the robots, resulting in a situation where the optimal solution had a single robot push all the boxes. In the fourth and last scenario, we added more boxes than that could be pushed and our approach still had no problems. In all cases, both STAMR and STAMR-G were able to find an optimal solution without changes other than modifications to the problem described. In every case, STAMR took significantly more time than other methods; this is expected and STAMR should only be used when computation time is not a concern.

C. Order Delivery

We additionally performed an evaluation to examine both the scalability of our approach and to demonstrate its applicability. In this scenario, robots are to deliver orders of boxes from stores to houses (Fig. 5). While a robot can deliver a single order quickly, delivering more than one order at a time (but no more than two for simplicity) can be more efficient but requires the robots to move slower, due to stacking one order on another, to avoid dropping the orders. Too many robots delivering to the same house could contribute to congestion around that house and slowness could aggravate congestion, which can lead to safety issues (i.e., collisions).

We created a virtual environment in the Webots simulator [30] where the stores are indicated by red zones, and the houses by green zones. Local obstacle avoidance was implemented for the robots to coordinate with each other. Order pickup and delivery were simulated: when a robot got close to the store or house, orders would be automatically loaded/unloaded. A simple baseline would be for each robot to deliver one order at a time. We show how our approach handled such scenarios more flexibly. First, we use a rule to express that robots are slower with more than one product:

$$F_{On}(X,Y), F_{On}(Y,R) \rightarrow F_{RequireSlow}(R)$$

To ensure delivering no more than two orders at a time, we introduce a CIR to create a logically invalid state on a special predicate, which the assignment algorithm recognizes as signaling an invalid assignment:

$$F_{On}(X,Y), F_{On}(Y,Z), F_{On}(Z,R) \rightarrow \bot$$

Such an invalid state is recognized automatically because of the SAT conversion. These CIRs will become implication clauses such that the assignment algorithm will avoid setting the antecedent (left side) to true. There are CIRs to calculate congestion; robots moving slowly cause more congestion at a location and robots moving quickly cause less:

$$F_{MovingQuickly}(R,L) \rightarrow F_{Congested1}(L)$$
$$F_{MovingSlowly}(R,L) \rightarrow F_{Congested2}(L)$$

There are CIRs to add up congestion levels to a store or house, similar to the weight CIRs in Site Clearing. Finally, there is a CIR for safety and efficiency by preventing excessive congestion at a location:

$$F_{Congested4}(L) \rightarrow \bot$$

where we prohibit a congestion level above 4, which is a controllable variable. Our approach can strike a balance between efficiency and safety by avoiding excessive congestion.

We randomly generated the starting positions of the robots, stores, and houses with regional constraints to ensure a roughly even distribution within their designated areas. Each order was pseudo-randomly assigned both a starting store and a destination house while ensuring that each store received 4 orders and each house placed 2 orders. We generated a single scenario with 5 stores, 10 houses, and 20 robots, and ran it 10 times. In these scenarios, each order corresponds to

a task and all tasks share the same utility. A robot moving quickly is associated with a lower cost (but it can deliver only one order) while moving slowly is associated with a slightly higher cost (while delivering two).

Our approach recognized that the superior solution was to have most of the robots deliver two orders to the same houses and move slowly. Note that due to our focus on instantaneous assignment, the allocation solution is based on some *conservative* assumptions, such as a robot that is assigned to deliver two orders always moves slowly even though in practice the robot can speed up after a delivery; similarly, two robots delivering to the same house are assumed to increase congestion even when they may arrive at different times.

Because we used E-puck robots that lack strong mobility, collisions are nearly unavoidable in complex situations such as this. However, our approach resulted in a significantly lower average number of collisions, as well as a lower average time-to-completion, indicative of fewer avoidance maneuvers. The baseline averaged 10 collisions ($\sigma \approx 0.5$), while ours yielded an average of 1 collision ($\sigma \approx 0.08$). Additionally, our approach completed the deliveries approximately 24% faster ($\sigma \approx 3\%$) on average.

V. Conclusions

In this paper, we presented a novel framework for task allocation with multitasking robots, considering both the synergistic and restrictive aspects of multitasking. It addresses a gap in existing methods by considering physical constraints as the key to enable multitasking in multi-robot systems. We formulated the problem and presented a compilation method to convert it to Weighted MAX-SAT. We further presented a greedy method. We compared both methods with a modern ST-MR-IA baseline to demonstrate the benefits of multitasking in synthetic domains and with two simulations to demonstrate its scalability, flexibility, and applicability.

One major limitation of our work is an absence of examination into true numeric constraints, including temporality. Notably, addressing this would enable us to simplify the specification of CIRs substantially and consider TA task assignment, instead of being limited to IA. Considering numericity in constraints would remove the need for, i.e., the separate "Push" and "StrongPush" capabilities in our running example. An extension into temporality would require a consideration of required concurrency [31] as it applies to CIRs as well as to actions, but would significantly expand the expressiveness of our approach. Finally, the specification of CIRs is not wholly intuitive; it would be a worthy extension to our work simply to create a formulation which is easier to intuit, perhaps via learning from examples.

Acknowledgment: We thank the anonymous reviewers for their helpful comments and suggestions. This research is supported in part by NSF grant 2047186.

References

[1] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[2] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[3] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *AIJ*, vol. 101, no. 1-2, pp. 165–200, 1998.

[4] P. M. Shiroma and M. F. Campos, "Comutar: A framework for multi-robot coordination and task allocation," in *IROS*. IEEE, 2009, pp. 4817–4824.

[5] W. Smith and Y. Zhang, "Achieving multitasking robots in multi-robot tasks," *ICRA*, 2019.

[6] L. Vig and J. Adams, "Multi-robot coalition formation," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 637–649, 2006.

[7] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme, "Coalition structure generation with worst case guarantees," *Artificial Intelligence*, vol. 111, no. 1-2, pp. 209–238, 1999.

[8] J. A. Adams *et al.*, "Coalition formation for task allocation: Theory and algorithms," *JAAMAS*, vol. 22, no. 2, pp. 225–248, 2011.

[9] S. Liemhetcharat and M. Veloso, "Weighted synergy graphs for effective team formation with heterogeneous ad hoc agents," *Artificial Intelligence*, vol. 208, pp. 41–65, 2014.

[10] B. Gerkey and M. Mataric, "Sold!: Auction methods for multi-robot coordination," *IEEE Transactions on Robotics and Automation, Special Issue on Multi-robot Systems*, pp. 758–768, 2001.

[11] L. Parker and F. Tang, "Building multirobot coalitions through automated task solution synthesis," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1289–1305, Jul. 2006.

[12] R. Zlot and A. Stentz, "Complex task allocation for multiple robots," in *ICRA*, Apr. 2005, pp. 1515–1522.

[13] Y. Zhang and L. E. Parker, "Considering inter-task resource constraints in task allocation," *JAAMAS*, vol. 26, no. 3, pp. 389–419, 2013.

[14] Y. Zhang and L. Parker, "IQ-ASyMTRe: Synthesizing coalition formation and execution for tightly-coupled multirobot tasks," in *IROS*, 2010.

[15] ——, "IQ-ASyMTRe: Forming executable coalitions for tightly coupled multirobot tasks," *TRO*, vol. 29, no. 2, pp. 400–416, 2013.

[16] B. Miloradović and A. Papadopoulos, "A formal definition of the multi-robot multi-task time-extended assignment problem configuration," in *CASE*, 2025.

[17] E. Bischoff, F. Meyer, J. Inga, and S. Hohmann, "Multi-robot task allocation and scheduling considering cooperative tasks and precedence constraints," in *IEEE International Conference on Systems, Man, and Cybernetics*, 2020.

[18] O. Shehory and S. Kraus, "Formation of overlapping coalitions for precedence-ordered task-execution among autonomous agents," in *Proc. of ICMAS-96*. Citeseer, 1996, pp. 330–337.

[19] V. D. Dang, R. K. Dash, A. Rogers, and N. R. Jennings, "Overlapping coalition formation for efficient data fusion in multi-sensor networks," in *AAAI*, vol. 6, 2006, pp. 635–640.

[20] M. Gelfond and V. Lifschitz, *Action Languages*. Linköping University Electronic Press, 1998.

[21] P. Khandelwal, F. Yang, M. Leonetti, V. Lifschitz, and P. Stone, "Planning in action language bc while learning action costs for mobile robots," in *Proceedings of ICAPS*, vol. 24, 2014, pp. 472–480.

[22] M. Fox and D. Long, "Pddl+: Modeling continuous time dependent effects," *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, vol. 4, 2017.

[23] W. M. Piotrowski, M. Fox, D. Long, D. Magazzeni, and F. Mercorio, "Heuristic planning for pddl+ domains," in *AAAI Workshop: Planning for Hybrid Systems*, vol. 16, 2016.

[24] J. J. Alferes, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski, "Lups-a language for updating logic programs," *Artificial Intelligence*, vol. 138, pp. 87–116, 2002.

[25] Y. Qiao, K. Zhong, H. Wang, and X. Li, "Developing event-condition-action rules in real-time active database," in *Proceedings of the ACM Symposium on Applied Computing*. ACM, 2007, pp. 511–516.

[26] M. Huth, *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

[27] M. Fox and D. Long, "Modelling mixed discrete-continuous domains for planning," in *JAIR*, vol. 27, 2006, pp. 235–297.

[28] F. Heras, J. Larrosa, and A. Oliveras, "Minimaxsat: A new weighted max-sat solver," in *Theory and Applications of Satisfiability Testing*, 2007, pp. 41–55.

[29] W. Smith and Y. Zhang, "Assigning multi-robot tasks to multitasking robots," 2026. [Online]. Available: https://arxiv.org/abs/2506.15032

[30] Webots, "http://www.cyberbotics.com," open-source Mobile Robot Simulation Software. [Online]. Available: http://www.cyberbotics.com

[31] W. Cushing, M. Kambhampati, S., and D. S. Weld, "When is temporal planning really temporal?" in *IJCAI*, 2007, pp. 1852–1859.