Exact Approaches to the Multi-Agent Collective Construction Problem

Edward Lam[1,2][0000−0002−4485−5014], Peter J. Stuckey[1][0000−0003−2186−0459], Sven Koenig[3][0000−0002−5458−094X], and T. K. Satish Kumar[3]

[1] Monash University, Melbourne, Victoria, Australia
[2] CSIRO Data61, Melbourne, Victoria, Australia
[3] University of Southern California, Los Angeles, California, USA
edward.lam@monash.edu,peter.stuckey@monash.edu,
skoenig@usc.edu,tkskwork@gmail.com

Abstract. The multi-agent collective construction problem tasks agents to construct any given three-dimensional structure on a grid by repositioning blocks. Agents are required to also use the blocks to build ramps in order to access the higher levels necessary to construct the building, and then remove the ramps upon completion of the building. This paper presents a mixed integer linear programming model and a constraint programming model of the problem, either of which can exactly optimize the problem, as previous efforts have only considered heuristic approaches. The two models are evaluated on several small instances with a large number of agents. The plans clearly show the swarm behavior of the agents. The mixed integer linear programming model is able to find optimal solutions faster than the constraint programming model and even some existing incomplete methods due to its highly-exploitable network flow substructures.

Keywords: classical planning · multi-agent planning · multi-agent path finding · blocksworld · swarm robotics

1 Introduction

The multi-agent collective construction (MACC) problem tasks a set of cooperative robots in a blocksworld with the construction of a given three-dimensional structure. The structure is built from blocks, which must be carried and rearranged by the robots. The problem aims to determine minimum-cost paths for the robots to perform this task while avoiding collisions.

The problem is best explained by example. Figure 1 illustrates a solution to a toy instance. Blocks are shown in gray. Two robots are shown in black and yellow. In timesteps 1 to 9, the black robot brings three blocks into the world and then exits. Outside the grid, robots are assumed to operate infinitely fast; i.e., an infinite number of actions can be performed in one timestep outside the grid. (Alternatively, the black robot can be assumed to be different robots in practice.) In timesteps 1 to 4, the yellow robot enters the world with a block and

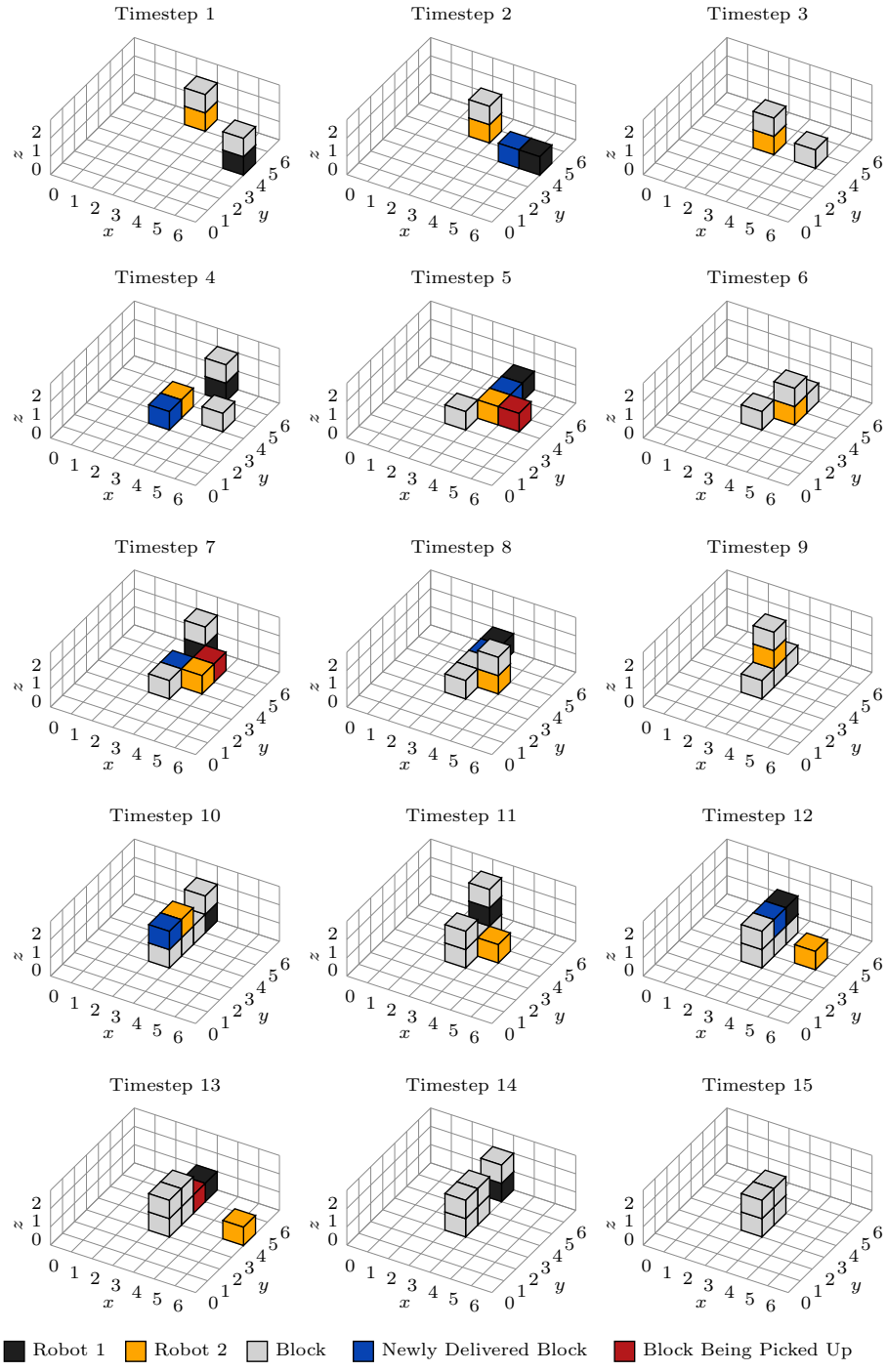

Fig. 1. A solution to a toy MACC instance.

delivers it at coordinate $(3, 3)$. In timesteps 5 to 10, the yellow robot rearranges the blocks previously brought into the world by the black robot. It then proceeds to exit the world in timesteps 11 to 13. In timesteps 10 to 12, the black robot brings in another block for delivery. In timestep 14, the black robot removes the ramp that it previously delivered in timestep 8 and used in timestep 11 to access the top level of the structure. The black robot then exits in timestep 15, leaving the structure fully assembled.

MACC is relevant to many applications, such as robotics [8] and open pit mining [6]. The problem is relatively simple to understand, yet poses many interesting questions for combinatorial optimization. In particular, questions about symmetries and dominance rules are highly non-trivial. The main contributions of this paper are a mixed integer linear programming (MILP) model and a constraint programming (CP) model of the problem. Using either of these two models, this paper is the first one to exactly optimize the problem, as all previous approaches are heuristics, which find high-quality solutions without proof of optimality. Experimental results show that the MILP model substantially outperforms the CP model because of its network flow substructures, which are easily exploited by MILP solvers. The remainder of the paper discusses these results in detail.

2 Problem Definition

Consider a planning horizon of $T \in \mathbb{Z}_+$ timesteps, and let $\mathcal{T} = \{0, \dots, T-1\}$ be the set of timesteps. The problem is stated on a three-dimensional grid that is divided into cells. Let the grid be $X \in \mathbb{Z}_+$ cells wide, $Y \in \mathbb{Z}_+$ cells deep and $Z \in \mathbb{Z}_+$ cells high. Let $\mathcal{X} = \{0, \dots, X-1\}$, $\mathcal{Y} = \{0, \dots, Y-1\}$ and $\mathcal{Z} = \{0, \dots, Z-1\}$ be the sets of coordinates in the three dimensions. Define $\mathcal{C} = \mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$ as the set of all cells. Then, every cell $(x, y, z) \in C$ is a triple of coordinates in the grid. Define the border cells $\mathcal{B} = \{(x, 0, 0) : x \in \mathcal{X}\} \cup \{(x, Y-1, 0) : x \in \mathcal{X}\} \cup \{(0, y, 0) : y \in \mathcal{Y}\} \cup \{(X-1, y, 0) : y \in \mathcal{Y}\}$ as the perimeter cells on the ground level. Define the positions $\mathcal{P} = \mathcal{X} \times \mathcal{Y}$ as the projection of the cells onto the first two dimensions. That is, the positions lie on the two-dimensional grid corresponding to the top-down view of the three-dimensional grid. Define the neighbors of a position $(x, y) \in \mathcal{P}$ as the set of positions $\mathcal{N}_{(x,y)} = \{(x-1, y), (x+1, y), (x, y-1), (x, y+1)\} \cap \mathcal{P}$.

Consider a problem with $A \in \mathbb{Z}_+$ identical robots. A robot is of the size of a cell. Each robot can carry up to one block at any given time. Similar to robots, a block is the size of a cell. Robots start and finish outside the grid. A robot can enter and exit the world at any border cell, with or without carrying a block. (An infinite reservoir of blocks lies beyond the grid.) During every timestep that a robot is on the grid, it must take one of the following four actions:

- If the robot is carrying a block, it can *deliver* the block to a neighboring position of the same height as its current cell, raising the height at the delivery position by one. (See timesteps 1 and 2 in Figure 1.)

- If the robot is not carrying a block, it can *pick up* a block in a neighboring position at the same height as the robot, decreasing the height at the removal position by one. (See timesteps 13 and 14 in Figure 1.)
- The robot can *move* from its current cell to a cell in a neighboring position provided that the difference in height of both cells is within one level; i.e., robots can climb up or down at most one block.
- The robot can *wait* at its current cell.

Blocks are stationary unless moved by a robot. Blocks can be stacked on any position except the border positions, which are reserved for entry and exit. Up to $Z - 1$ blocks can be stacked in any one position. In any position, a block can only be placed on the ground level or on top of the top-most block. Only the top-most block can be removed.

Borrowing terminology from multi-agent path finding [10], robots must avoid *vertex collisions* and *edge collisions*. A vertex collision occurs when two or more robots attempt to occupy, pick up from or deliver to a position. An edge collision occurs when two or more robots attempt to cross through each other to swap positions. Robots can enter and exit the grid as many or as few times as necessary. If a robot exits the grid, it must spend at least one timestep outside the grid (e.g., to pick up another block) before returning.

The aim of the problem is to find paths for the robots to construct a given three-dimensional structure by collectively rearranging blocks. By the end of the planning horizon, all robots must have exited the world, leaving the completed structure behind. The input structure is given as a desired height $\bar{z}_{(x,y)} \in \mathcal{Z}$ for every position $(x, y) \in \mathcal{P}$. That is, every block must be supported from below. Structures cannot be hollow like a cave.

In many related problems, such as multi-agent path finding and vehicle routing, two common cost/objective functions are makespan and sum-of-costs. Minimizing makespan is equivalent to compressing the time horizon so that the structure is completed as soon as possible at the expense of more actions. Minimizing sum-of-costs minimizes the total number of actions taken by the robots regardless of the time taken.

Since all robots are identical, the sum-of-costs is minimized by deploying one robot to build the structure. Since there is at most one robot in the world at any time, collisions never occur. At the cost of a higher makespan, using fewer robots always dominates using more robots for the sum-of-costs objective. On the contrary, solely using the makespan objective is problematic because unnecessary robots can aimlessly wander the world, incurring penalties in the sum-of-costs objective but having no impact on the makespan objective (besides possibly colliding with other robots and extending the makespan). Therefore, this paper argues for a two-tier lexicographic objective that first minimizes the makespan and then minimizes the sum-of-costs. This objective finds solutions that can construct a structure in the least amount of time and, with second priority, the fewest number of actions.

3 Background and Related Work

Teams of smaller robots are often more effective than a few larger robots. Smaller robots are usually cheaper, easier to program and easier to deploy. Despite their possibly limited sensing and computational capabilities, teams of smaller shape-shifting/self-reconfiguring robots are more fault tolerant and provide more parallelism than a few larger robots. A good example of the effectiveness of teams of smaller robots is in the domain of collective construction [3,2,7,8].

Inspired by termites building mounds, the Harvard TERMES project investigated how teams of robots can cooperate to build user-specified three-dimensional structures much larger than themselves [8]. The TERMES hardware consists of small autonomous robots and a reservoir of passive "building blocks", simply referred to as "blocks". The robots are of roughly the same size as the blocks. Yet, they can manipulate these blocks to build tall structures by stacking the blocks on top of each other and building ramps to scale greater heights. The three basic operations of a TERMES robot are: (1) climbing up or down blocks one block-height at a time; (2) navigating with proper localization on a partially-built structure without falling down; and (3) lifting, carrying and putting down a block to attach it to or detach it from a partially-built structure. MACC approximately models the TERMES robots.

A collective construction problem in the TERMES domain is to build a user-specified three-dimensional structure, assuming that the reservoir is unlimited and that the initial world is empty of blocks, i.e., all blocks are initially in the reservoir. A decentralized reactive algorithm for constructing any given structure is presented in [8]. While this algorithm succeeds in building the structure, it does not treat MACC as a rigorous combinatorial optimization problem.

A rigorous formulation of the TERMES collective construction problem as a combinatorial optimization problem is provided in [5]. The formulation exploits the fact that the three basic operations of the TERMES robots are almost always successful. The high reliability of these operations provides a nice abstraction for centralized planning algorithms, allowing for the assumption that the robots are ideal. Under these idealistic assumptions, the paper presents an algorithm that achieves a small number of pickup and drop-off operations. The algorithm solves the single-robot construction problem using dynamic programming carried out on a tree spanning the cells of a workspace matrix that represent physical locations on a grid frame of reference. The use of dynamic programming exploits common substructures and significantly reduces the number of operations on blocks. Their algorithm is polynomial-time and performs well in practice but does not guarantee optimality. In fact, the paper does not characterize the complexity of the problem.

This algorithm has been extended to the case of multiple robots in [1]. Here, the idea is to use different robots for different branches of the tree with the intuition that they can largely operate in independent regions. However, a drawback of this approach is that the regions of the tree close to its root quickly become a bottleneck and not much parallelism is achieved. Higher parallelism is achieved in [9]. Inspired by recent advances in single-agent reinforcement learning, this approach

extends the single-agent asynchronous advantage actor-critic (A3C) algorithm to enable multiple agents to learn a homogeneous, distributed policy, where agents work together toward a common goal without explicitly interacting. It relies on centralized policy and critic learning, but decentralized policy execution, in a fully-observable system. Neither of the two algorithms is guaranteed to generate optimal solutions.

Finally, since blocksworld domains are well studied in the area of automated planning and scheduling, the International Planning Competition (IPC)[4] now includes the collective construction problem in the TERMES domain as a benchmark problem because of its interesting properties [4].

4 The Mixed Integer Linear Programming Model

The MILP model is based on network flow. Network flow problems generalize shortest path problems. Using network flow, the MILP model treats all robots as one flow through a time-expanded graph that is further complicated by the states necessary to track whether a robot is carrying a block from one timestep to the next.

Let $\mathcal{K} = \{\mathrm{M}, \mathrm{P}, \mathrm{D}\}$ be the types of actions, where M indicates that a robot is moving from one cell to another or waiting at the same cell, P indicates that a robot is picking up a block, and D indicates that a robot is delivering a block.

Define an action as a nine-tuple $i = (t, x, y, z, c, a, x', y', z')$, whose elements are given as follows:

- $t \in \mathcal{T}$ is the timestep of the action.
- $x \in \mathcal{X} \cup \{\mathrm{S}\}$, $y \in \mathcal{Y} \cup \{\mathrm{S}\}$ and $z \in \mathcal{Z} \cup \{\mathrm{S}\}$ are the coordinates of the robot taking the action, where S is a special symbol indicating that the robot is at a start location off the grid and will move into a border cell.
- $c \in \{0, 1\}$ indicates whether the robot is currently carrying a block.
- $a \in \mathcal{K}$ denotes the action type.
- $x' \in \mathcal{X} \cup \{\mathrm{E}\}$, $y' \in \mathcal{Y} \cup \{\mathrm{E}\}$ and $z' \in \mathcal{Z} \cup \{\mathrm{E}\}$ are the coordinates of the cell where the action occurred, where E is a special symbol indicating that the robot is moving from a border cell to an end location off the grid.

For instance, the action $(5, 1, 2, 3, 0, \mathrm{M}, 1, 3, 3)$ represents a robot standing in cell $(1, 2, 3)$ at timestep 5 moving to $(1, 3, 3)$ while not carrying a block, and the action $(5, 1, 2, 3, 1, \mathrm{D}, 1, 3, 3)$ represents a robot in cell $(1, 2, 3)$ at timestep 5 delivering a block it is carrying to cell $(1, 3, 3)$.

Not every possible $(t, x, y, z, c, a, x', y', z')$ in the Cartesian product is a valid action. For example, $(3, 0, 0, 0, 0, \mathrm{M}, 5, 5, 5)$ indicates a robot teleporting from cell $(0, 0, 0)$ at timestep 3 to cell $(5, 5, 5)$. Define the set of valid actions $\mathcal{R} = \mathcal{R}_1 \cup \ldots \cup \mathcal{R}_6$ made up of six subsets of different actions:

- Robots can enter the world at a border cell: $\mathcal{R}_1 = \{(t, \mathrm{S}, \mathrm{S}, \mathrm{S}, c, \mathrm{M}, x', y', z') : t \in \{0, \ldots, T-4\} \wedge c \in \{0, 1\} \wedge (x', y', z') \in \mathcal{B}\}$.

[4] `https://ipc2018.bitbucket.io`

- Robots can move to a neighboring cell at the same level, one level above or one level below: $\mathcal{R}_2 = \{(t, x, y, z, c, \mathrm{M}, x', y', z') : t \in \{1, \ldots, T-3\} \wedge (x, y, z) \in \mathcal{C} \wedge c \in \{0, 1\} \wedge (x', y') \in \mathcal{N}_{(x,y)} \wedge z' \in \mathcal{Z} \wedge |z' - z| \leq 1\}$.
- Robots can wait at the same cell: $\mathcal{R}_3 = \{(t, x, y, z, c, \mathrm{M}, x, y, z) : t \in \{1, \ldots, T-3\} \wedge (x, y, z) \in \mathcal{C} \wedge c \in \{0, 1\}\}$.
- Robots can exit the world at a border cell: $\mathcal{R}_4 = \{(t, x, y, z, c, \mathrm{M}, \mathrm{E}, \mathrm{E}, \mathrm{E}) : t \in \{2, \ldots, T-2\} \wedge (x, y, z) \in \mathcal{B} \wedge c \in \{0, 1\}\}$.
- While not carrying a block, robots can pick up a block from a neighboring cell at the same level: $\mathcal{R}_5 = \{(t, x, y, z, 0, \mathrm{P}, x', y', z) : t \in \{1, \ldots, T-3\} \wedge (x, y) \in \mathcal{P} \wedge z \in \{0, \ldots, Z-2\} \wedge (x', y') \in \mathcal{N}_{(x,y)}\}$.
- While carrying a block, robots can deliver the block to a neighboring cell at the same level: $\mathcal{R}_6 = \{(t, x, y, z, 1, \mathrm{D}, x', y', z) : t \in \{1, \ldots, T-3\} \wedge (x, y) \in \mathcal{P} \wedge z \in \{0, \ldots, Z-2\} \wedge (x', y') \in \mathcal{N}_{(x,y)}\}$.

The timesteps in $\mathcal{R}_1, \ldots, \mathcal{R}_6$ are chosen carefully since, e.g., all robots must be off the world by timestep $T-1$, they must be moving from a border cell off the world by timestep $T-2$, hence $T-3$ is the latest that a block can be delivered.

Every position $(x, y) \in \mathcal{P}$ is modeled as a shortest path from height 0 to the desired height $\bar{z}_{(x,y)} \in \mathcal{Z}$ at the final timestep $T-1$. Similar to $\mathcal{R}$, define $\mathcal{H} = \{(t, x, y, z, z') : t \in \{0, \ldots, T-2\} \wedge (x, y, z) \in \mathcal{C} \wedge z' \in \mathcal{Z} \wedge |z' - z| \leq 1\}$ to represent the actions of growing or shrinking the height of a position. An action $(t, x, y, z, z') \in \mathcal{H}$ indicates that position (x, y) currently has height z at timestep t and height z' at timestep $t+1$.

In this model, the problem can be thought of in terms of two groups of interacting agents: (1) pillars that need to grow or shrink to their target heights in the least amount of time (pillars might need to grow higher than their target height), and (2) robots that assist the pillars by picking up and delivering blocks around the world. For pillars to grow upward, they need robots to stack blocks at their positions. For robots to place blocks on a pillar, a neighboring pillar needs to be of a similar height. Hence, in some sense, the problem involves a complicated interaction between two sets of agents, both of which co-operate to achieve their goals.

The model captures the actions of all robots in one network flow and the actions of each pillar in a shortest path (a special case of network flow). These two substructures are coupled by interdependency constraints. In the absence of the interdependencies, the model separates into a number of independent network flows. Hence, the idea behind the MILP model is for the solver to first resolve the interdependencies to simplify the problem, and then the problem becomes much easier since pure network flow problems can be solved in polynomial time by linear programming [11]. Of course, resolving the interdependencies remains a major challenge.

The MILP model is written using non-standard wildcard notation. Let $\mathcal{U}$ be a set containing tuples $(u_1, u_2, \ldots, u_n)$. For constants $u_1, u_2, \ldots, u_n$, we use the notation $\mathcal{U}_{u_1, u_2, \ldots, u_n}$ as a shorthand for the set $\{(u'_1, u'_2, \ldots, u'_n) \in \mathcal{U} : u'_1 = u_1 \wedge u'_2 = u_2 \wedge \ldots \wedge u'_n = u_n)\}$, which is equal to the singleton $\{(u_1, u_2, \ldots, u_n)\}$ if the element exists and equal to the empty set $\emptyset$ otherwise. Let $*$ denote a

wildcard symbol for matching any value in a dimension of the tuples. For example, $\mathcal{U}_{*,u_2,\ldots,u_n}$ is shorthand for the set $\{(u'_1, u'_2, \ldots, u'_n) \in \mathcal{U} : u'_2 = u_2 \wedge \ldots \wedge u'_n = u_n\}$, and $\mathcal{U}_{u_1,u_2,*,\ldots,*}$ represents the set $\{(u'_1, u'_2, u'_3, \ldots, u'_n) \in \mathcal{U} : u'_1 = u_1 \wedge u'_2 = u_2\}$. This wildcard notation is used to pick subsets of $\mathcal{R}$ and $\mathcal{H}$.

Figure 2 shows the model. For every robot action $i \in \mathcal{R}$, define a binary decision variable $r_i \in \{0, 1\}$ to indicate whether the action occurred. Similarly, define a binary decision variable $h_i \in \{0, 1\}$ for every height action $i \in \mathcal{H}$.

Objective Function (1) minimizes the sum-of-costs objective, i.e., the total number of cells occupied by robots throughout the planning horizon. The makespan is minimized external to the model by sequentially increasing T, as described later.

Constraints (2) to (6) define a path for each position. Constraint (2) prevents blocks from being placed at the border positions because robots must enter and exit the world on the ground level. Constraint (3) starts the world devoid of blocks. This constraint states that all cells have height 0 in the first two timesteps because the earliest a robot can appear in the world is in timestep 1; hence blocks cannot be placed in the world until timestep 2. Constraint (4) enforces the completion of the structure. Robots must have exited the world by timestep $T - 1$. So they must be at a border cell exiting before timestep $T - 2$. Therefore, the structure must be built before the last two timesteps. Constraint (5) flows the height of each position from one timestep to the next. Constraint (6) enforces one value of height for every position in each timestep.

Constraints (7) to (11) govern the actions of the robots. Constraint (7) flows a robot not carrying a block in and out of a cell. The first summation accounts for a robot moving without a block from any cell at timestep t into cell (x, y, z) at timestep $t + 1$. The second summation counts whether a robot standing at (x, y, z) has just deposited a block nearby. After taking any of these actions, the robot will be at cell (x, y, z) at timestep $t + 1$ without a block. It then has to either move to another cell (the third summation) or pick up a nearby block (the fourth summation). Constraint (8) is a similar constraint for robots carrying a block. This constraint states that, if a robot carrying a block is in cell (x, y, z) at timestep $t + 1$ (either by moving into the cell with a block or by picking up a block nearby), then it must afterward move while continuing to carry the block or deliver the block. Constraints (7) and (8) implicitly require robots to start and end outside the grid. Constraint (9) prevents vertex collisions. It permits at most one robot to be at position (x, y) or to pick up from or deliver a block to position (x, y) at any timestep. Constraint (10) is the edge collision constraint, which prevents robots from exchanging positions. Constraint (11) limits the number of robots. By also including robots at the dummy start cell $(\mathrm{S}, \mathrm{S}, \mathrm{S})$, this constraint also requires robots that have left the world to spend at least one timestep outside the world (e.g., to pick up another block) before returning.

Constraints (12) to (14) couple the robots and the pillars. Without these three constraints, the problem separates into two independent parts. Constraint (12) states that, if a robot is at cell (x, y, z), then the height of the pillar at position

$$\min \sum_{\substack{i=(t,x,y,z,c,a,x',y',z')\in\mathcal{R}:\\ (x,y,z)\neq(\mathrm{S},\mathrm{S},\mathrm{S})}} r_i \tag{1}$$

subject to

$$h_{t,x,y,z,z} = 1 \qquad \forall t \in \{0,\dots,T-3\}, (x,y,z) \in \mathcal{B}, \tag{2}$$

$$h_{0,x,y,0,0} = 1 \qquad \forall (x,y) \in \mathcal{P}, \tag{3}$$

$$h_{T-2,x,y,\tilde{z}_{(x,y)},\tilde{z}_{(x,y)}} = 1 \qquad \forall (x,y) \in \mathcal{P}, \tag{4}$$

$$\sum_{i\in\mathcal{H}_{t,x,y,*,z}} h_i = \sum_{i\in\mathcal{H}_{t+1,x,y,z,*}} h_i \qquad \forall t \in \{0,\dots,T-3\}, (x,y,z) \in \mathcal{C}, \tag{5}$$

$$\sum_{i\in\mathcal{H}_{t,x,y,*,*}} h_i = 1 \qquad \forall t \in \{0,\dots,T-2\}, (x,y) \in \mathcal{P}, \tag{6}$$

$$\sum_{i\in\mathcal{R}_{t,*,*,*,0,\mathrm{M},x,y,z}} r_i + \sum_{i\in\mathcal{R}_{t,x,y,z,1,\mathrm{D},*,*,*}} r_i = \sum_{i\in\mathcal{R}_{t+1,x,y,z,0,\mathrm{M},*,*,*}} r_i + \sum_{i\in\mathcal{R}_{t+1,x,y,z,0,\mathrm{P},*,*,*}} r_i \qquad \forall t \in \{0,\dots,T-3\}, (x,y,z) \in \mathcal{C}, \tag{7}$$

$$\sum_{i\in\mathcal{R}_{t,*,*,*,1,\mathrm{M},x,y,z}} r_i + \sum_{i\in\mathcal{R}_{t,x,y,z,0,\mathrm{P},*,*,*}} r_i = \sum_{i\in\mathcal{R}_{t+1,x,y,z,1,\mathrm{M},*,*,*}} r_i + \sum_{i\in\mathcal{R}_{t+1,x,y,z,1,\mathrm{D},*,*,*}} r_i \qquad \forall t \in \{0,\dots,T-3\}, (x,y,z) \in \mathcal{C}, \tag{8}$$

$$\sum_{i\in\mathcal{R}_{t,x,y,*,*,*,*,*,*}} r_i + \sum_{i\in\mathcal{R}_{t,*,*,*,*,\mathrm{P},x,y,*}} r_i + \sum_{i\in\mathcal{R}_{t,*,*,*,*,*,\mathrm{D},x,y,*}} r_i \leq 1 \qquad \forall t \in \{1,\dots,T-2\}, (x,y) \in \mathcal{P}, \tag{9}$$

$$\sum_{i\in\mathcal{R}_{t,x,y,*,*,\mathrm{M},x',y',*}} r_i + \sum_{i\in\mathcal{R}_{t,x',y',*,*,\mathrm{M},x,y,*}} r_i \leq 1 \qquad \forall t \in \{1,\dots,T-2\}, (x,y) \in \mathcal{P}, (x',y') \in \mathcal{N}_{(x,y)}, \tag{10}$$

$$\sum_{i\in\mathcal{R}_{t,*,*,*,*,*,*,*,*}} r_i \leq A \qquad \forall t \in \mathcal{T}, \tag{11}$$

$$\sum_{i\in\mathcal{H}_{t,x,y,z,*}} h_i \geq \sum_{i\in\mathcal{R}_{t,x,y,z,*,*,*,*,*}} r_i \qquad \forall t \in \{0,\dots,T-2\}, (x,y,z) \in \mathcal{C}, \tag{12}$$

$$h_{t,x,y,z+1,z} = \sum_{i\in\mathcal{R}_{t,*,*,*,0,\mathrm{P},x,y,z}} r_i \qquad \forall t \in \{0,\dots,T-2\}, (x,y) \in \mathcal{P}, z \in \{0,\dots,Z-2\}, \tag{13}$$

$$h_{t,x,y,z,z+1} = \sum_{i\in\mathcal{R}_{t,*,*,*,1,\mathrm{D},x,y,z}} r_i \qquad \forall t \in \{0,\dots,T-2\}, (x,y) \in \mathcal{P}, z \in \{0,\dots,Z-2\}, \tag{14}$$

$$h_i \in \{0,1\} \qquad \forall i \in \mathcal{H}, \tag{15}$$

$$r_i \in \{0,1\} \qquad \forall i \in \mathcal{R}. \tag{16}$$

Fig. 2. The MILP model.

(x, y) must be z. Constraints (13) and (14) respectively equate pickup and delivery actions to a decrease and increase in the height of a pillar.

Constraints (15) and (16) specify the domains of the variables.

5 The Constraint Programming Model

Standard CP models of routing problems are based on a sequence of actions performed by each robot. This modeling indexes every robot individually and, hence, introduces robot symmetry. This section presents a CP model that forgoes the sequence-based modeling and, instead, adopts a network flow structure to eliminate robot symmetry. Compared to the MILP model, the CP model uses a simpler network that omits the vertical dimension, actions and block-carrying state. It models these aspects using logical and ELEMENT constraints, which better exploit the strengths of CP.

Assign every position $p = (x, y) \in \mathcal{P}$ an identifier $i_p = Y \cdot x + y$ that maps the two-dimensional positions to a one-dimensional index. Let $\mathcal{I} = \{0, \ldots, X \cdot Y - 1\}$ denote the set of position identifiers. Let $\bar{z}_i \in \mathcal{Z}$ be the height of the desired structure at position $i \in \mathcal{I}$.

Define the border positions as $\mathcal{B} = \{i_{(x,0)} : x \in \mathcal{X}\} \cup \{i_{(x,Y-1)} : x \in \mathcal{X}\} \cup \{i_{(0,y)} : y \in \mathcal{Y}\} \cup \{i_{(X-1,y)} : y \in \mathcal{Y}\}$, and the interior positions as $\overline{\mathcal{B}} = \{i \in \mathcal{I} : i \notin \mathcal{B}\}$. Define two dummy positions -1 and -2 off the grid and group them in $\mathcal{O} = \{-1, -2\}$. Let $\mathcal{E} = \mathcal{I} \cup \mathcal{O}$ denote every position (on and off the grid).

For any position $i = i_{(x,y)}$, let $\mathcal{N}_i = \{i_{(x-1,y)}, i_{(x+1,y)}, i_{(x,y-1)}, i_{(x,y+1)}\} \cap \mathcal{I}$ be its neighbors. Define a set of neighbors that includes the off-grid positions as

$$\mathcal{N}_i^{\mathcal{E}} = \begin{cases} \mathcal{N}_i & i \in \overline{\mathcal{B}}, \\ \mathcal{N}_i \cup \mathcal{O} & i \in \mathcal{B}. \end{cases}$$

Using $\mathcal{N}_i^{\mathcal{E}}$, robots are able to move off the grid from a border position.

Let $\mathcal{K} = \{\mathrm{M}, \mathrm{B}, \mathrm{U}\}$ be the types of actions, where M indicates that a robot is moving from one position to another or waiting at the same position, B indicates that a robot is picking up or delivering a block, and U indicates that a position is unoccupied by a robot.

Let $r_{t,i} \in \mathcal{K}$ be a decision variable representing the action taken by the robot in position $i \in \mathcal{E}$ at timestep $t \in \mathcal{T}$. Define $n_{t,i} \in \mathcal{E}$ as a variable denoting the next position of the robot in position $i \in \mathcal{E}$ at timestep $t \in \mathcal{T}$. Define $b_{t,i} \in \mathcal{I}$ as the position of the pick-up or delivery by the robot in position $i \in \mathcal{I}$ at timestep $t \in \mathcal{T}$. Define $c_{t,i} \in \{0, 1\}$ as a variable that indicates whether the robot in position $i \in \mathcal{E}$ at timestep $t \in \mathcal{T}$ is carrying a block. Let $p_{t,i}, d_{t,i} \in \{0, 1\}$ be variables that, respectively, indicate whether the robot in position $i \in \mathcal{I}$ at timestep $t \in \{0, \ldots, T-2\}$ is picking up or delivering a block. Let $h_{t,i} \in \mathcal{Z}$ be a variable that stores the height at position $i \in \mathcal{E}$ in timestep $t \in \mathcal{T}$. The meaning of the variables for any position $i \in \mathcal{I}$ is clear. The variables are also defined for $i \in \mathcal{O}$ to ensure that the problem is satisfiable by giving the ELEMENT constraints an end-point; these variables do not carry much meaning.

The CP model is shown in Figures 3 and 4. Objective Function (17) minimizes the number of occupied positions at all timesteps. The ELEMENT global constraint is used in Constraints (30), (32), (35), (39) and (41) to (44).

Constraint (18) fixes the height of all off-grid positions. Constraint (19) disallows blocks to be delivered to the border positions. Constraint (20) states that the world is devoid of blocks in the first two timesteps. Constraint (21) requires the building to be completed before the last two timesteps. Constraint (22) allows the height at any position to change by at most one level.

Constraints (23) to (26) fix the robot variables at the off-grid positions. Constraints (27) and (28) disallow robots on the grid in the first and last timesteps. Constraint (29) requires robots to stay at the same position when picking up or delivering a block. Constraint (30) maintains the block-carrying states of robots when moving. Constraint (31) changes the block-carrying states of robots after picking up or delivering blocks.

Constraints (32) and (33) are the flow constraints. Constraint (32) states that every position is either unoccupied or the robot in the position must take an action in the next timestep. Constraint (33) states that an interior position is unoccupied or a robot reached it from a nearby position in the previous timestep. Constraint (34) prevents vertex collisions by disallowing more than one robot from being in a position, picking up a block from the position or delivering a block to the position at timestep $t + 1$. Constraint (35) prevents edge collisions. Constraint (36) limits the number of robots on the grid during each timestep. This constraint also requires robots to spend at least one timestep outside the grid, as discussed in Section 2.

Constraints (37) and (38) compute whether a robot is picking up or delivering a block. If a robot moves, Constraint (39) requires the height of its next position to be within one level of the height of its current position. If a robot waits at the same position, Constraint (40) states that the height must remain the same. Constraint (41) states that the height of the block being picked up must be one level higher than the pillar at the position of the robot (i.e., the block is at the same level as the robot). Constraint (42) decreases the height at the position of a block after a pick up. Constraints (43) and (44) are the equivalent constraints for deliveries. Constraint (45) counts the changes to the height at a position. Constraints (46) and (47) are redundant constraints, which improve the filtering.

Constraints (48) to (54) specify the domains of the variables. Constraint (50) requires robots to move to a neighboring position, the same position or off the grid. Constraint (51) states that blocks must be picked-up from or delivered to a neighboring position.

6 Experimental Results

The experiments compare the run-time of the two models. The MILP model is solved using Gurobi 9.0.2, a state-of-the-art mathematical programming solver that regularly outperforms its competitors in standard benchmarks. The CP model is solved using OR-Tools 7.6, an open-source CP solver that has won many

$$\min \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}} (r_{t,i} \neq \mathrm{U}) \tag{17}$$

subject to

$$h_{t,i} = 0 \qquad \forall t \in \mathcal{T}, i \in \mathcal{O}, \tag{18}$$
$$h_{t,i} = 0 \qquad \forall t \in \mathcal{T}, i \in \mathcal{B}, \tag{19}$$
$$h_{t,i} = 0 \qquad \forall t \in \{0,1\}, i \in \mathcal{I}, \tag{20}$$
$$h_{t,i} = \bar{z}_i \qquad \forall t \in \{T-2, T-1\}, i \in \mathcal{I}, \tag{21}$$
$$h_{t,i} - 1 \leq h_{t+1,i} \leq h_{t,i} + 1 \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{22}$$
$$r_{t,i} = \mathrm{M} \qquad \forall t \in \mathcal{T}, i \in \mathcal{O}, \tag{23}$$
$$n_{t,i} = i \qquad \forall t \in \mathcal{T}, i \in \mathcal{O}, \tag{24}$$
$$c_{t,-1} = 1 \qquad \forall t \in \mathcal{T}, \tag{25}$$
$$c_{t,-2} = 0 \qquad \forall t \in \mathcal{T}, \tag{26}$$
$$r_{0,i} = \mathrm{U} \qquad \forall i \in \mathcal{I}, \tag{27}$$
$$r_{T-1,i} = \mathrm{U} \qquad \forall i \in \mathcal{I}, \tag{28}$$
$$(r_{t,i} = \mathrm{B}) \rightarrow (n_{t,i} = i) \qquad \forall t \in \mathcal{T}, i \in \mathcal{I}, \tag{29}$$
$$(r_{t,i} = \mathrm{M}) \rightarrow (c_{t+1,n_{t,i}} = c_{t,i}) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{30}$$
$$(r_{t,i} = \mathrm{B}) \rightarrow (c_{t+1,i} = \neg c_{t,i}) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{31}$$
$$(r_{t,i} = \mathrm{U}) \vee (r_{t+1,n_{t,i}} \neq \mathrm{U}) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{32}$$
$$(r_{t+1,i} = \mathrm{U}) \vee \bigvee_{j \in \mathcal{N}_i \cup \{i\}} (r_{t,j} \neq \mathrm{U} \wedge n_{t,j} = i) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \overline{\mathcal{B}}, \tag{33}$$
$$\sum_{j \in \mathcal{N}_i \cup \{i\}} (r_{t,j} = \mathrm{M} \wedge n_{t,j} = i) + (r_{t,i} = \mathrm{B}) + \sum_{j \in \mathcal{N}_i} (r_{t+1,j} = \mathrm{B} \wedge b_{t+1,j} = i) \leq 1$$
$$\forall t \in \{1, \ldots, T-2\}, i \in \mathcal{I}, \tag{34}$$
$$(r_{t,i} = \mathrm{M} \wedge n_{t,i} \neq i \wedge r_{t,n_{t,i}} = \mathrm{M}) \rightarrow (n_{t,n_{t,i}} \neq i) \qquad \forall t \in \{1, \ldots, T-2\}, i \in \mathcal{I}, \tag{35}$$
$$\sum_{i \in \mathcal{I}} (r_{t,i} \neq \mathrm{U}) + \sum_{i \in \mathcal{B}} (r_{t-1,i} = \mathrm{M} \wedge n_{t-1,i} < 0) \leq A \qquad \forall t \in \{1, \ldots, T-1\}, \tag{36}$$
$$p_{t,i} \leftrightarrow (r_{t,i} = \mathrm{B} \wedge c_{t+1,i} \wedge \neg c_{t,i}) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{37}$$
$$d_{t,i} \leftrightarrow (r_{t,i} = \mathrm{B} \wedge \neg c_{t+1,i} \wedge c_{t,i}) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{38}$$
$$(r_{t,i} = \mathrm{M}) \rightarrow (h_{t,i} - 1 \leq h_{t+1,n_{t,i}} \leq h_{t,i} + 1) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{39}$$
$$(r_{t,i} = \mathrm{M} \wedge n_{t,i} = i) \rightarrow (h_{t+1,i} = h_{t,i}) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{40}$$
$$p_{t,i} \rightarrow (h_{t,b_{t,i}} = h_{t,i} + 1) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{41}$$
$$p_{t,i} \rightarrow (h_{t+1,b_{t,i}} = h_{t,b_{t,i}} - 1) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{42}$$
$$d_{t,i} \rightarrow (h_{t,b_{t,i}} = h_{t,i}) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{43}$$
$$d_{t,i} \rightarrow (h_{t+1,b_{t,i}} = h_{t,b_{t,i}} + 1) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{44}$$
$$h_{t+1,i} = h_{t,i} - \sum_{j \in \mathcal{N}_i} (p_{t,j} \wedge b_{t,j} = i) + \sum_{j \in \mathcal{N}_i} (d_{t,j} \wedge b_{t,j} = i) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \tag{45}$$

Fig. 3. The CP model. Continued in Figure 4.

$$h_{t+1,i} = h_{t,i} - 1 \rightarrow \bigvee_{j \in \mathcal{N}_i} (p_{t,j} \wedge b_{t,j} = i) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \quad (46)$$

$$h_{t+1,i} = h_{t,i} + 1 \rightarrow \bigvee_{j \in \mathcal{N}_i} (d_{t,j} \wedge b_{t,j} = i) \qquad \forall t \in \{0, \ldots, T-2\}, i \in \mathcal{I}, \quad (47)$$

$$h_{t,i} \in \mathcal{Z} \qquad \forall t \in \mathcal{T}, i \in \mathcal{E}, \quad (48)$$

$$r_{t,i} \in \mathcal{K} \qquad \forall t \in \mathcal{T}, i \in \mathcal{E}, \quad (49)$$

$$n_{t,i} \in \mathcal{N}_i^{\mathcal{E}} \cup \{i\} \qquad \forall t \in \mathcal{T}, i \in \mathcal{E}, \quad (50)$$

$$b_{t,i} \in \mathcal{N}_i \qquad \forall t \in \mathcal{T}, i \in \mathcal{I}, \quad (51)$$

$$c_{t,i} \in \{0, 1\} \qquad \forall t \in \mathcal{T}, i \in \mathcal{E}, \quad (52)$$

$$p_{t,i} \in \{0, 1\} \qquad \forall t \in \mathcal{T}, i \in \mathcal{I}, \quad (53)$$

$$d_{t,i} \in \{0, 1\} \qquad \forall t \in \mathcal{T}, i \in \mathcal{I}. \quad (54)$$

Fig. 4. The CP model. Continued from Figure 3.

of the MiniZinc Challenges in recent times. The two solvers are run for up to seven days in parallel mode with 20 threads on an Intel Xeon E5-2660 v3 CPU at 2.60 GHz with 64 GB of memory.

Even though traditional finite-domain CP solvers are most effective with a hand-tailored variable and value selection heuristic (i.e., a branching rule), nogood learning solvers like OR-Tools perform best when using their preferred search strategies, as evidenced in the MiniZinc Challenge, where OR-Tools performs better in the free search category. Therefore, we do not specify a variable and value selection heuristic.

The two models minimize the number actions, i.e., the sum-of-costs. To lexicographically minimize makespan as well, the two models are run with sequentially increasing T. That is, the experiments begin with $T = 4$, which is the smallest possible value, and progressively increase T until the problem becomes satisfiable (i.e., the solver finds a feasible solution). Then, the solver proceeds to find an optimal solution. Upon completion, the solution is guaranteed to have the lowest possible makespan and the lowest number of actions for that makespan.

The six structures from [9] are used for evaluation. They are shown in Figure 5. Up to 50 robots are permitted. Table 1 shows the results for the six instances. For both models, the table gives the time to prove optimality, the optimal makespan, the best available sum-of-costs and its lower bound, and the number of robots required to execute the plan. (Unused robots never enter the world.)

The MILP model solves all six instances exactly within six days. Instances 1, 2 and 6 are trivial for the MILP model to solve, while the run-times of the other three instances span a large range. The CP model solves Instance 2 exactly but is substantially slower than the MILP model. For the remaining five instances, it finds feasible solutions with the optimal makespan.

Figure 6 plots the twelve timesteps for executing the optimal plan to the first instance. The large number of robots swarming into the world makes it difficult

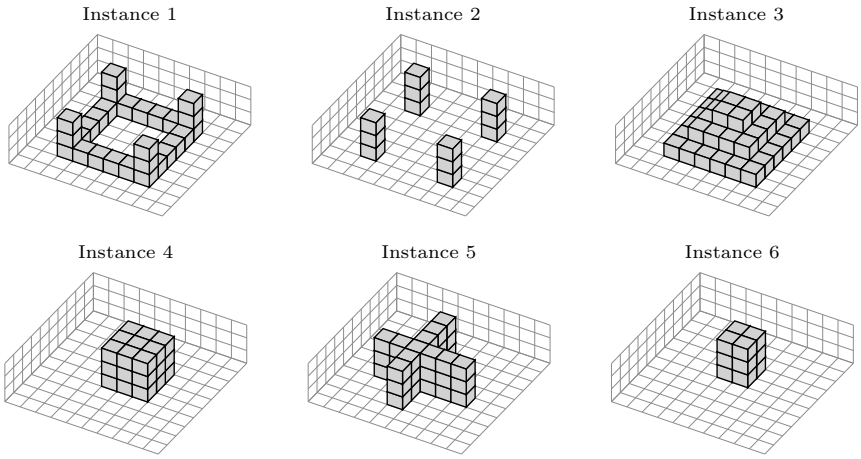

Fig. 5. The structures to construct in the six instances.

Model	Instance	Run-time	Makespan	Sum-of-costs	Sum-of-costs LB	Robots
MILP	1	29s	11	176	176	34
	2	3s	11	128	128	28
	3	1.2hr	13	344	344	44
	4	5.5hr	17	429	429	42
	5	5.7d	17	368	368	37
	6	183s	15	234	234	27
CP	1	> 7d	11	178	107	30
	2	1.2hr	11	128	128	28
	3	> 7d	13	354	164	44
	4	> 7d	17	452	189	50
	5	> 7d	17	395	39	41
	6	> 7d	15	245	154	28

Table 1. Best available solution to the six instances.

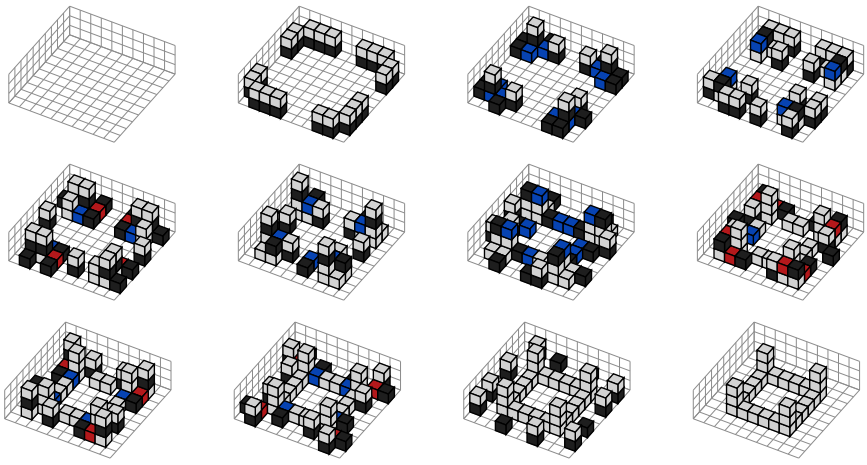

Fig. 6. The twelve timesteps in the optimal plan to Instance 1.

to analyze any emergent macro-level behavior. Nonetheless, minor bucket-brigade behavior is already demonstrated in the toy example from Figure 1.

The state-of-the-art reinforcement learning method [9] produced solutions taking up to 2,000 timesteps (for many fewer robots). The two optimization models found feasible solutions to all instances with a makespan of less than twenty timesteps, indicating that these small structures are simple to construct as they do not rely on long chains of interdependent blocks.

7 Conclusions and Future Work

The MACC problem is a relatively new problem that is starting to gain attention in the multi-agent planning community. The problem tasks a group of co-operating robots to construct a three-dimensional structure by rearranging blocks in a blocksworld. Robots are required to build ramps to access the upper levels of the structure, and then remove the ramps after assembling the structure.

This paper models the problem using MILP and unexpectedly reveals the two interacting network-flow substructures hidden in the problem. A CP model of the problem is also developed but is slower than the MILP model because it lacks easily exploitable structure. This preliminary study shows that small instances of complex path-finding problems with an extremely large state space can be solved exactly today, as previous solution methods were all heuristics, which aim to find high-quality but not provably optimal solutions.

Scaling the MILP model to long time horizons remains a major challenge. Early experiments show that using more robots shortens the makespan, and hence, makes the model smaller and easier. Surprisingly, it is the number of timesteps that makes the problem difficult, rather than the number of robots. Buildings taller than five blocks require long ramps for accessing the higher levels. These ramps take many timesteps to build, resulting in a makespan much longer and a model much larger than what is possible for exact optimization. For these instances, the only viable methods are the existing heuristics.

Early experiments show that the CP model presented in Section 5 and solved using OR-Tools is fastest among eleven models solved using both OR-Tools and Chuffed. One interesting finding is that sequence-based models (often using the Regular global constraint) are faster in Chuffed but slower in OR-Tools. Whereas, network flow models (e.g., the model in Section 5) are faster in OR-Tools, which has a linear relaxation propagator. Future studies should investigate whether CP-like sequencing models can obtain competitive performance.

As suggested by a reviewer, MaxSAT models can also be considered since the problem mainly contains Boolean variables and clauses. The difficulty would be encoding the cardinality constraints, but how to do this is well-known.

Acknowledgments

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987, 1817189, 1837779, and 1935712.

References

1. Cai, T., Zhang, D., Kumar, T.K.S., Koenig, S., Ayanian, N.: Local search on trees and a framework for automated construction using multiple identical robots. In: Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (2016)
2. Grushin, A., Reggia, J.: Automated design of distributed control rules for the self-assembly of prespecified artificial structures. In: Robotics and Autonomous Systems, 56(4):334-359 (2008)
3. Jones, C., Mataric, M.: Automatic synthesis of communication-based coordinated multi-robot systems. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (2004)
4. Koenig, S., Kumar, T.K.S.: A case for collaborative construction as testbed for cooperative multi-agent planning. In: Proceedings of the ICAPS-2017 Scheduling and Planning Applications Workshop (2017)
5. Kumar, T.K.S., Jung, S., Koenig, S.: A tree-based algorithm for construction robots. In: Proceedings of the International Conference on Automated Planning and Scheduling (2014)
6. Lambert, W., Brickey, A., Newman, A., Eurek, K.: Open-pit block-sequencing formulations: A tutorial. Interfaces **44**, 127–142 (2014)
7. Napp, N., Klavins, E.: Robust by composition: Programs for multi-robot systems. In: Proceedings of the IEEE International Conference on Robotics and Automation (2010)
8. Petersen, K., Nagpal, R., Werfel, J.: TERMES: An autonomous robotic system for three-dimensional collective construction. In: Proceedings of Robotics: Science and Systems (2011)
9. Sartoretti, G., Wu, Y., Paivine, W., Kumar, T.K.S., Koenig, S., Choset, H.: Distributed reinforcement learning for multi-robot decentralized collective construction. In: Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (2018)
10. Stern, R., Sturtevant, N., Felner, A., Koenig, S., Ma, H., Walker, T., Li, J., Atzmon, D., Cohen, L., Kumar, S., Boyarski, E., Bartak, R.: Multi-agent pathfinding: Definitions, variants, and benchmarks. In: Proceedings of the Symposium on Combinatorial Search (2019)
11. Vaidyanathan, B., Ahuja, R.K.: Minimum cost flows. In: Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Inc. (2011)