Action Duration Generalization for Exact Multi-Agent Collective Construction

Martin Rameš[a] and Pavel Surynek[b]
Faculty of Information Technology, Czech Technical University, Thákurova 9, 160 00 Prague 6, Czech Republic

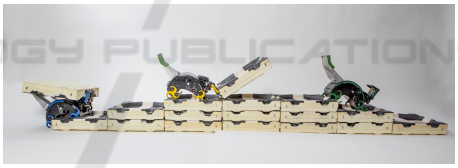
Keywords: Multi-Agent Construction, Multi-Agent Planning, Mixed Integer Linear Programming.

Abstract: This paper addresses exact approaches to multi-agent collective construction problem which tasks a group of cooperative agents to build a given structure in a blocksworld under the gravity constraint. We propose a generalization of the existing exact model based on mixed integer linear programming by accommodating varying agent action durations. We refer to the model as a fraction-time model. The introduction of action durations enables one to create a more realistic model for various domains. It provides a significant reduction of plan execution duration at the cost of increased computational time, which rises steeply the closer the model gets to the exact real-world action duration. We also propose a makespan estimation function for the fraction-time model. This can be used to estimate the construction time reduction size for cost-benefit analysis. The fraction-time model and the makespan estimation function have been evaluated in a series of experiments using a set of benchmark structures. The results show a significant reduction of plan execution duration for non-constant duration actions due to decreasing synchronization overhead at the end of each action. According to the results, the makespan estimation function provides a reasonably accurate estimate of the makespan.

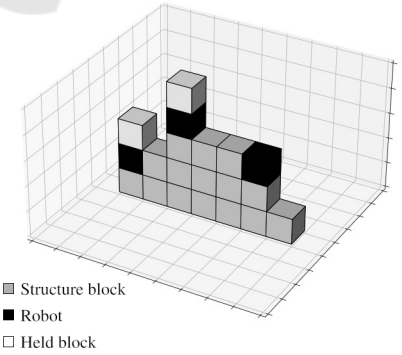
1 INTRODUCTION

The multi-agent collective construction (MACC) problem tasks a group of cooperative agents to build a given structure in a blocksworld. Agents can pick up, move, and place blocks, which are used as the only building material for a three-dimensional structure. Both blocks and agents are moving under the condition of gravity. The problem aims to determine a collision-free plan for the agent movement, which would perform the construction task while minimizing the execution time (makespan) and the sum of durations the agents spend on the grid (sum-of-costs).

Previously, such problems were solved heuristically with no proof of optimality. Recently, a new branch of research emerged, aiming to study optimal MACC problem solutions. This paper aims to further this research by generalizing the currently best optimal model – MILP model by (Lam et al., 2020) – further referred to as the constant-time model. The generalization, further referred to as the fraction-time model, allows agent actions to differ in duration to better map real-world multi-agent systems.



(a) "Termes robot 01" by Forgemind ArchiMedia is licensed under CC BY 2.0 (Forgemind ArchiMedia, 2014).



(b) Visualization of the world state in the MACC problem.

Figure 1: Example of the TERMES system and its MACC representation. Three robotic agents are building a long stair structure. The middle agent is performing a deliver-block action.

[a] https://orcid.org/0009-0000-3301-6269
[b] https://orcid.org/0000-0001-7200-0542

Rameš, M. and Surynek, P.
Action Duration Generalization for Exact Multi-Agent Collective Construction.
DOI: 10.5220/0012385600003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 718-725
ISBN: 978-989-758-680-4; ISSN: 2184-433X

The constant-time model (Lam et al., 2020) exactly optimizes the MACC plan to achieve minimum makespan and sum-of-costs as primary and secondary optimization criteria, respectively, assuming all actions to have the same duration. This is not the case in real life and the robots are forced to wait for the end of the longest action. This includes the TERMES robots (Petersen et al., 2011), which inspired the constant-time model. Notably, the difference between average block pick-up time (15s) and placement time (24s), measured in (Petersen et al., 2011), is 46%. The paper also shows that agents carrying a block or moving on unstructured terrain (i.e. gravel, grass) move slower.

To better utilize the information about mean action durations, we propose the fraction-time model. The model will accept a structure height map along with action durations as input and provide a MACC plan as output. We modify the constant-time model to include the action durations. The modification is non-trivial, as the generalized model needs to prevent mid-action agent collisions and maintain the network flow substructure of the base model to remain computationally viable. We demonstrate the improvement of plan execution duration on three instances with three non-constant action durations, including TERMES durations. We provide two makespan lower bounds, one based on fraction-time model relaxation and one based on the fraction-time model with a less detailed action duration assignment, which can also be used to get the makespan upper bound. To solve the fraction-time MILP model, we use the state-of-the-art solver Gurobi (Gurobi Optimization, LLC, 2023).

2 RELATED WORK

The MACC problem is part of a wider research branch, which studies various forms of collaborative construction. These forms include, for instance, self-assembly, where the agents themselves act as building blocks, once they reach their destination (Piranda and Bourgeois, 2018; Romanishin et al., 2013). Another uses quadrotors to assemble the structure from beams and columns (Barros dos Santos et al., 2018). Robotic arms with connectors on both sides are proposed by (Jenett and Cheung, 2017) to build and move atop lightweight lattice structures like an inchworm.

All these approaches focus on robots with unique capabilities and limitations, requiring dedicated algorithms. We focus on the TERMES, a termite-inspired multi-agent system by (Petersen et al., 2011). The TERMES robots can pick up, carry, and place similar-sized blocks. Each robot can carry at most one block, and climb the difference of one block. The robots can build complex structures based on simple sensor feedback and high-level instructions – move forward by one block, turn 90° left, turn 90° right, pick up the block in front of the robot, and deliver a carried block at a position in front of the robot (Petersen et al., 2011). The MACC discretizes the movement of the TERMES robot and makes the problem partially hardware-independent by limiting the robots to the above high-level actions. Image 1 shows the TERMES robots and their MACC representation.

Currently, there are only two models that can solve the MACC makespan-optimally (Lam et al., 2020) – a mixed integer linear programming (MILP) model and a constraint programming model. The results of both models in the paper match on finished instances, with the MILP model being much faster to compute. Both models assume a constant action duration.

There are also multiple heuristic/sub-optimal approaches to the MACC problem. The Compiler for Scalable Construction by the TERMES Robot Collective heuristically minimizes the number of agents, who pass through the construction site (Deng et al., 2019). Another approach presents an algorithm minimizing the number of pick-up and deliver actions, performing dynamic programming on a spanning tree to find paths for a TERMES robot (Cai et al., 2016).

There are also two approaches, which utilize a solver/planner but sacrifice the optimal solution to gain order-of-magnitude better computation speed (Srinivasan et al., 2023; Singh et al., 2023). Both compare their solution to the constant-time model.

3 THE FRACTION-TIME MODEL

To allow more precise modeling of non-constant action durations of real-world robots, we propose the fraction-time model. The model replaces the timestep t of each action by t_s and t_e, which denote the timestep of action start (inclusive) and action end (exclusive), respectively. This notation mirrors non-preemptive scheduling notation, where the action is executed within a time interval $[t_s, t_e)$.

We use the non-standard notation of (Lam et al., 2020). For set $\mathcal{U}$ of tuples $(u_1, u_2, \ldots, u_n)$ with length $n \in \mathbb{Z}_+$ notation $\mathcal{U}_{u_1,u_2,\ldots,u_n}$ is short for $\{(u'_1, u'_2, \ldots, u'_n) : u'_1 = u_1 \wedge u'_2 = u_2 \wedge \cdots \wedge u'_n = u_n\}$. Let $*$ be a wildcard symbol matching any value at its position in the tuple (i.e. $\mathcal{U}_{*,u_2,\ldots,u_n}$ is short for $\{(u'_1, u'_2, \ldots, u'_n) : u'_2 = u_2 \wedge \cdots \wedge u'_n = u_n\}$ and $\mathcal{U}_{u_1,u_2,*,\ldots,*}$ is short for $\{(u'_1, u'_2, \ldots, u'_n) : u'_1 = u_1 \wedge u'_2 = u_2\}$).

Let $\widehat{a}$ be shorthand for $\{0, \ldots, a-1\}, a \in \mathbb{Z}_+$. Let (X, Y) be the size of the building area and $(Z-1)$ be the height of the target structure in blocks, the ex-

tra layer allows travel on top of the structure. Let $\mathcal{C} = \widehat{X} \times \widehat{Y} \times \widehat{Z}$ be the set of all positions within the grid, $\mathcal{P} = \widehat{X} \times \widehat{Y}$ is the projection of $\mathcal{C}$ into the first two dimensions, $\mathcal{B} = \{(x,0,0) : x \in \widehat{X}\} \cup \{(x,Y-1,0) : x \in \widehat{X}\} \cup \{(0,y,0) : y \in \widehat{Y}\} \cup \{(X-1,y,0) : y \in \widehat{Y}\}$ is the set of border cells at the perimeter of the building area. Let $\overline{z}_{(x,y)}$ be the target height of the block column at position (x,y). Let $\mathcal{C}' = \mathcal{C} \cup \{(\mathrm{S},\mathrm{S},\mathrm{S}),(\mathrm{E},\mathrm{E},\mathrm{E})\}$ be a set of all agent-accessible positions – including two special positions $(\mathrm{S},\mathrm{S},\mathrm{S}),(\mathrm{E},\mathrm{E},\mathrm{E})$, which symbolize the start and the end position outside the grid. Let $\mathcal{K} = \{\mathrm{M},\mathrm{P},\mathrm{D}\}$ be a set of agent action type distinguishers – M for move action (used for "entry", "leave", "move_block", "move_empty" and "wait" action types), P for "pick_up" action type and D for "deliver" action type. Let $\mathcal{N}_{(x,y)} = \{(x-1,y),(x+1,y),(x,y-1),(x,y+1)\} \cap \mathcal{P}$ be the set of neighbor positions of (x,y) and $\mathcal{T} = \widehat{T}$ be the planning horizon of T timesteps. The actions are tuples $i = (t_s,t_e,x,y,z,c,k,x',y',z')$, which consist of values:

- start time $t_s \in \mathbb{Z}_+$ (inclusive)
- end time $t_e = t_s + d, d \in \mathbb{Z}_+$ (exclusive), where d is the action duration
- start position $(x,y,z) \in \mathcal{C} \cup \{(\mathrm{S},\mathrm{S},\mathrm{S})\}$
- indicator $c \in \{0,1\}$ if agent carries a block
- action type distinguisher $k \in \mathcal{K}$
- end position $(x',y',z') \in \mathcal{C} \cup \{(\mathrm{E},\mathrm{E},\mathrm{E})\}$
  - position of affected block for pick_up and deliver actions (agent stays at the same position)
  - marks agent position at the end of the action for other action types

Let duration d_i of action $i = (t_s,t_e,\dots)$ be $d_i = t_e - t_s$. Let $f_d : \mathbb{Z}_+ \times \mathcal{C}' \times \{0,1\} \times \mathcal{K} \times \mathcal{C}' \longrightarrow \mathbb{Z}_+$ be a function of action duration, based on the rest of action tuple – that is t_s, (x,y,z), c, k and (x',y',z') respectively.

Let $\mathcal{A} = \{\text{entry}, \text{leave}, \text{move\_block}, \text{move\_empty}, \text{deliver}, \text{pick\_up}, \text{wait}\}$ be the set of all action types.

- $Q_{\text{entry}} = \{(t_s,\mathrm{S},\mathrm{S},\mathrm{S},c,\mathrm{M},x,y,z) : t_s \in \widehat{T-3} \wedge c \in \{0,1\} \wedge (x,y,z) \in \mathcal{B}\}$
- $Q_{\text{move\_empty}} = \{(t_s,x,y,z,0,\mathrm{M},x',y',z') : t_s \in \{1,\dots,T-3\} \wedge (x,y,z) \in \mathcal{C} \wedge (x',y') \in \mathcal{N}_{(x,y)} \wedge |z'-z| \leq 1\}$
- $Q_{\text{move\_block}} = \{(t_s,x,y,z,1,\mathrm{M},x',y',z') : t_s \in \{1,\dots,T-3\} \wedge (x,y,z) \in \mathcal{C} \wedge (x',y') \in \mathcal{N}_{(x,y)} \wedge |z'-z| \leq 1\}$
- $Q_{\text{wait}} = \{(t_s,x,y,z,c,\mathrm{M},x,y,z) : t_s \in \{1,\dots,T-3\} \wedge c \in \{0,1\} \wedge (x,y,z) \in \mathcal{C}\}$
- $Q_{\text{leave}} = \{(t_s,x,y,z,c,\mathrm{M},\mathrm{E},\mathrm{E},\mathrm{E}) : t_s \in \{2,\dots,T-2\} \wedge c \in \{0,1\} \wedge (x,y,z) \in \mathcal{B}\}$
- $Q_{\text{pick\_up}} = \{(t_s,x,y,z,0,\mathrm{P},x',y',z) : t_s \in \{1,\dots,T-3\} \wedge (x,y) \in \mathcal{P} \wedge (x',y') \in \mathcal{N}_{(x,y)} \wedge z \in \widehat{Z-1}\}$
- $Q_{\text{deliver}} = \{(t_s,x,y,z,1,\mathrm{D},x',y',z) : t_s \in \{1,\dots,T-3\} \wedge (x,y) \in \mathcal{P} \wedge (x',y') \in \mathcal{N}_{(x,y)} \wedge z \in \widehat{Z-1}\}$

Let $Q = \bigcup_{a \in \mathcal{A}} Q_a$. Let $f_q : \mathbb{Z}_+ \times \mathcal{C}' \times \{0,1\} \times \mathcal{K} \times \mathcal{C}' \longrightarrow \mathbb{Q}_+$ be a function, which assigns each action a duration $f_q(t_s,\dots,z'),(t_s,\dots,z') \in Q$. If not specified otherwise, the f_q function must be part of the model input. Let m be the least common multiple of denominators in $\{f_q(t_s,\dots,z') : \forall (t_s,\dots,z') \in Q\}$. Then, unless otherwise specified, the f_d function is defined as $f_d(t_s,\dots,z') = m f_q(t_s,\dots,z'), \forall (t_s,\dots,z') \in Q$. Let us introduce the following simplifying notation – let $f_d(v) = f_d(t_s,\dots,z'), \forall v = (t_s,\dots,z') \in Q$. Let us define a set of all actions for each action type $a \in \mathcal{A}$ as $\mathcal{R}_a = \{(t_s,t_e,x,y,z,c,k,x',y',z') : v = (t_s,x,y,z,c,k,x',y',z') \in Q_a \wedge t_e = t_s + f_d(v)\}$.

The proposed model has seven disjoint sets of agent action types, which in union give the new set of agent actions. Agent action types are derived from six different action subsets in the original model:

Set $\mathcal{R}_{\text{entry}}$ of "entry" type actions features all actions, where the agent enters from the starting position outside the grid to a border cell. The agent might carry a block when it enters. This action type is the only source of new blocks for the construction site.

The set of agent actions, where the agent moves to the neighbor grid position is divided into two action types – "move_block" and "move_empty" for moving to the neighbor position while carrying / not carrying a block, respectively. This distinction is made for cases where the carried block requires the agent to move slower. Set $\mathcal{R}_{\text{move\_empty}}$ is for "move_empty", set $\mathcal{R}_{\text{move\_block}}$ is for "move_block" action type. Both action types can be implemented on TERMES robots as a combination of turn and move-forward actions.

Action type "wait" with action set $\mathcal{R}_{\text{wait}}$ has a duration of $T_{\text{wait}} = 1$ timestep, chosen because the minimum wait-time of agents is generally not limited. Action type "leave" is for agents leaving the grid from a border cell (going to the end position $(\mathrm{E},\mathrm{E},\mathrm{E})$). The associated action set is $\mathcal{R}_{\text{leave}}$. The agents can carry a block when leaving the grid, the only way to remove blocks from the construction site. Set $\mathcal{R}_{\text{pick\_up}}$ of "pick_up" type actions features all actions, where the agent picks up a block from a neighbor position. Set of "deliver" type actions $\mathcal{R}_{\text{deliver}}$ features all actions, where the agent delivers a block to a neighbor position. Let the set of all actions be $\mathcal{R} = \bigcup_{a \in \mathcal{A}} \mathcal{R}_a$.

Let $\mathcal{H}$ be a set of block-actions. Block-action is a tuple $(t,x,y,z,z') \in \mathcal{H}$, where $(x,y,z) \in \mathcal{C} \wedge z' \in \{z-1,z,z+1\} \cap \widehat{Z} \wedge t \in \widehat{T}$. All block-actions last one timestep, meaning that only action start time t

is required. Indicators $r_i \in \{0,1\}, i \in \mathcal{R}$ and $h_i \in \{0,1\}, i \in \mathcal{H}$ decide, which action is part of the plan. For instance, $r_j = 1, j = (5,7,1,2,0,1,D,2,2,0) \in \mathcal{R}$ in the solution indicates that an agent at timestep 5 delivers block to position $(2,2,0) \in \mathcal{C}$ while standing at $(1,2,0) \in \mathcal{C}$. The action is finished by timestep 7.

The objective function 1 minimizes the sum-of-costs (the sum of timesteps each robot spends on the grid). The proposed model counts the "entry"-type action timesteps into the objective function, because the agent is considered to partially be on the grid, when the action starts (blocking the border cell as part of the action exclusion zone).

The exclusion zones are proposed as a measure to avoid agent collisions. Each exclusion zone is created at the start of an action, removed at the end of the same action and grants the agent performing the action exclusive access to start- and end-position columns, if the position is within grid. The exclusive access does not allow the remaining agents to perform actions, which start or end within the exclusion zone.

$$\min(\sum_{i \in \mathcal{R}} r_i d_i) \tag{1}$$

$$h_{t,x,y,z,z} = 1, \forall t \in \widehat{T}, (x,y,z) \in \mathcal{B} \tag{2}$$

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

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

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

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

Constraints 2 – 6 are the same as in the original MILP model. Constraint 2 forbids placement of blocks at border cells, constraint 3 starts the world devoid of blocks, constraint 4 ensures that the target structure is finished at the end of construction, constraint 5 flows the column height from one timestep to the next and constraint 6 forces every position to have one height.

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

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

Constraints 7 and 8 flow the agents from one action to the next. Semi-closed interval of action execution $[t_s, t_e)$ is exploited for a seamless transition between actions. Similarly to the base model, constraint 7 flows agents without block and ensures that the number of agents ending their action without block at position (x,y,z) at timestep t is the same as the number of agents without block starting their action at the same position and in the same timestep. Constraint 8 does the equivalent for agents carrying a block.

$$\sum_{\substack{i \in \mathcal{R}_{I_s,t_e,} \\ x,y,*, \\ *,*, \\ *,*,* \\ t_s \le t < t_e}} r_i + \sum_{\substack{i \in \mathcal{R}_{I_s,t_e,} \\ *,*,*, \\ *,*, \\ x,y,* \\ t_s \le t < t_e}} r_i - \sum_{\substack{i \in \mathcal{R}_{I_s,t_e,} \\ x,y,*, \\ *,*, \\ x,y,* \\ t_s \le t < t_e}} r_i \le 1, \forall t \in \widehat{T}, (x,y) \in \mathcal{P} \tag{9}$$

Constraint 9 addresses vertex collision of agents during action execution. Since the only requirement for the agent is to perform the action a between t_s and t_e, the exact position of the agent is unknown and both start and end position are made into the exclusion zone, where no other action can take place (for timesteps within interval $[t_s, t_e)$). To avoid agents removing blocks under moving agents, the constraint makes the whole block column at position (x,y) of both the start and end of the action an exclusion zone. Separate constraint to prevent edge collisions (like in the base model) is no longer necessary, as two actions can no longer share vertices while executing simultaneously, due to the exclusion zones.

$$\sum_{i \in \mathcal{R}_{t_s,t_e,*,*,*,*,*,*,*,*:t_s \le t < t_e}} r_i \le A, \forall t \in \mathcal{T} \tag{10}$$

Constraint 10 limits the number of agents on the grid to at most $A \in \mathbb{Z}_+$, given as part of model input.

$$\sum_{i \in \mathcal{H}_{t,x,y,z,*}} h_i \ge \sum_{i \in \mathcal{R}_{t_s,t_e,x,y,z,*,*,*,*,*:t_s \le t < t_e}} r_i, \quad \forall t \in \widehat{T}, (x,y,z) \in \mathcal{C} \tag{11}$$

$$h_{t,x,y,z+1,z} = \sum_{i \in \mathcal{R}_{*,t+1,*,*,*,0,\mathrm{P},x,y,z}} r_i, \quad \forall t \in \widehat{T-1}, (x,y) \in \mathcal{P}, z \in \widehat{Z-1} \tag{12}$$

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

$$h_i \in \{0,1\}, \forall i \in \mathcal{H} \tag{14}$$

$$r_i \in \{0,1\}, \forall i \in \mathcal{R} \tag{15}$$

Constraint 11 forces agents to always stand on the highest block in the column. Constraints 12 and 13 govern decreases and increases in block column height, respectively. Constraint 12 ties every decrease by one block to pick_up action. Constraint 13 ties increase by one block to deliver action. Constraints 14 and 15 specify the variable domains.

The model is used to exactly optimize makespan and sum-of-costs, the primary and secondary optimization criteria, respectively. Optimization of the makespan is done by starting at the minimum possible value (estimated by a lower bound function described below) and sequentially increasing the makespan by one timestep until a solution is found.

We expect that the most often used f_d assignment will give each action-type a fixed duration (i.e. all actions in Q_{deliver} will have the duration T_{deliver}, the "pick_up" actions will have the duration $T_{\text{pick\_up}}$ etc. – see equation 16). Namely, in the case of the TERMES robots, the first two action type durations would be $T_{\text{deliver}} = 3$ and $T_{\text{pick\_up}} = 2$. The times assume one timestep is 10 s and are from (Petersen et al., 2011), where the block pick-up time is measured to be 15 ± 5 s and the block delivery time 24 ± 5 s. The rest of the action type durations are derived similarly, the full list is in table 1 in column "TERMES".

$$f_d(v) = \begin{cases} T_a & \text{if } v \in Q_a, a \in \mathcal{A} \\ 1 & \text{otherwise.} \end{cases} \quad (16)$$

4 MAKESPAN ESTIMATION, UPPER AND LOWER BOUND

Due to the generalization, the fraction-time model is expected to be more computationally demanding than the base model, when the action durations are not constant. To allow for an informed decision, whether the use of a fraction-time model is required, a lower bound, heuristic estimate, and upper bound of the fraction-time model makespan are presented.

A lower bound is computed using the relaxed problem MACC$_r$. MACC$_r$ relaxes the requirement for the maximum number of agents (allowing an unlimited number of agents), the constraint of agent collisions (allowing multiple agents to stand on one block), exclusion zone constraint (allowing agents to place blocks under other agents) and constraints limiting changes of agent vertical position (allowing agents to stay on top of a column, while it is being built and place blocks to the neighbor column at the same time). The only optimization criterion for MACC$_r$ is makespan. MACC$_r$ has a trivial solution – for each column in the building area add the number of agents equal to the column height and assign them to the column, starting at the border position closest to the column. For all agents – enter at the assigned starting position with a block, move to the assigned column (simultaneously), and one by one place the held block at the top of the assigned column. Once all agents assigned to a column place their blocks, move all those agents to the starting border cell and leave the building area. The makespan of MACC$_r$ is the lower bound of fraction-time MACC (because MACC$_r$ is the relaxation of MACC). Let T_r be the optimum makespan of MACC$_r$. Let $s_{(x,y)}$ be the minimum L_1 distance from a border cell to $\mathcal{N}_{(x,y)}$ (a neighbor of (x,y)). Let $d_a^{\min} = \min_{i \in \mathcal{R}_a} d_i, a \in \mathcal{A}$ be the minimum duration of action type a. Let $d_{\text{move}}^{\min} = \min\{d_{\text{move\_block}}^{\min}, d_{\text{move\_empty}}^{\min}\}$. Let $T_{(x,y)} = d_{\text{entry}}^{\min} + s_{(x,y)} d_{\text{move}}^{\min} + \bar{z}_{(x,y)} d_{\text{deliver}}^{\min} + s_{(x,y)} d_{\text{move}}^{\min} + d_{\text{leave}}^{\min}$ be the duration of the described action sequence for building the column at (x,y) ($\bar{z}_{(x,y)}$ is the desired height of the column). Then $l_r = \max\{T_{(x,y)} \mid \forall (x,y) \in \mathcal{P} \wedge \bar{z}_{(x,y)} > 0\} \leq T_r$ is a MACC lower bound (building of each column is independent in MACC$_r$, due to the relaxations, l_r is equal to the longest duration $T_{(x,y)}$).

A simple mission duration improvement function is proposed to assist in cost-benefit analysis and makespan estimation for more precise action duration mapping of the fraction-time model. Let $d_a^{\text{avg}} = (\sum_{i \in \mathcal{R}_a} d_i)/|\mathcal{R}_a|, a \in \mathcal{A}$. Let $\alpha = \sum_{a \in \mathcal{A}} d_a^{\text{avg}} / |\mathcal{A}|$ be the estimation of the relative makespan increase. Let T_{b} be the makespan computed by the fraction-time model where all actions last one timestep. The estimated makespan of the fraction-time model with a more detailed action duration mapping is defined as $T_h = \max(l_r, \min(u_f, \lceil \alpha T_{\text{b}} \rceil))$, where u_f is an upper bound defined later in the chapter.

Finally, the fraction-time model can be used for its own makespan estimation. Let r_i, $\mathcal{R}$, T and d_i for the fraction-time model with less precise action duration mapping be marked as r'_i, $\mathcal{R}'$, T' and d'_i, respectively. Since the more precise mapping of action durations makes the model more computationally demanding, a model with less precise mapping – defined as $\max_{i \in \mathcal{R}'_a} d'_i \leq \min_{i \in \mathcal{R}_a} d_i, \forall a \in \mathcal{A}$ – can be used for estimating the makespan of the more computationally demanding task. In this regard, the model with $d'_i = 1, \forall i \in \mathcal{R}'$ is especially interesting, as constant action durations should provide the smallest runtime of the fraction-time model for given height-map. Let $l_f = T'$ be a lower bound gained using fraction-time model with less precise action duration mapping.

Let u_f be an upper bound, defined as the execution duration of a plan by the fraction-time model with less precise action duration mapping, where the actions use durations of the more precise mapping and the agents wait at the beginning of each timestep until all actions that were supposed to end at that timestep are performed. This waiting strategy can also be used when executing the plan on real hardware. When $d'_i = 1, \forall i \in \mathcal{R}'$, u_f is defined by equation 17.

Let $u_c = T' * \max_{v \in Q} f_d(v)$ be a naive upper bound, where T' is the makespan of fraction-time

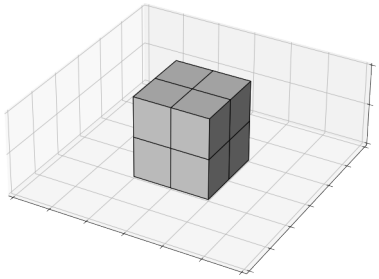


Figure 2: Instance used for makespan estimation precision measurement.

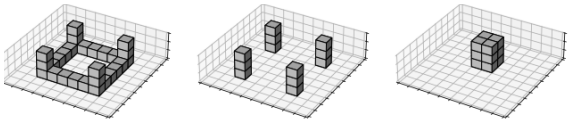


Figure 3: Three different instances used in experiments.

model with $d'_i = 1, \forall i \in \mathcal{R}'$. In practice, we multiply the makespan of unit-action-duration fraction-time model with the duration of the longest action.

$$u_{f_0} = 0$$
$$u_{f_n} = u_{f_{n-1}} + \max_{\substack{(t_s,t_e,x,\ldots,z') \\ \in \mathcal{R}'_{n-1,*,\ldots,*}}} f_d(u_{f_{n-1}}, x, \ldots, z'), \tag{17}$$

5 EXPERIMENTS

The first experiment aims to demonstrate construction time reduction for non-constant action durations. It is performed with the Gurobi 10.0.3 solver (Gurobi Optimization, LLC, 2023), limited to 32 threads on a 3 GHz Intel Skylake processor with 16 physical cores and hyper-threading, along with 132 GB of RAM.

For evaluation, we translate the fraction-time model to Python Gurobi API, which we use to describe the objective function 1 and all the constraints (2 – 15). The resulting translation requires a known makespan and optimizes only the sum-of-costs. To optimize makespan, we add an outer loop, iterating over makespans starting at lower bound l_r and increasing by 1 until a solution (construction plan) is found. The input for the Python program is specified as a 2-dimensional height-map of the structure, durations for enter, leave, move_block, move_empty, deliver and pick_up action types. The output of the model is a set of indicators r_i and h_i, which action tuples were used. Each action tuple contains a start and an end position, which is used to assign the actions to agents and compile an action sequence for each agent.

We experiment with three structures shown in figure 3. These instances represent a subset of instances from (Lam et al., 2020) that have the lowest solver computation time, with part of the area without blocks around the third structure removed to reduce access time for the agents. The fraction-time model is used to compute plans for the construction of each structure instance. Four action duration sets (i.e. sets of action durations) are used for each structure – the action durations can be seen in table 1. The maximum number of agents is 50 for instances 1 and 2; 20 for the third instance (due to the smaller construction area).

The table 2 shows the results of the first experiment, measuring solver runtime, solution makespan, and sum-of-costs for each instance–action-type-set duo. The construction plans are saved as well, the 1-timestep plan is used for u_f and u_c upper bound computation (result in parentheses next to makespan). The u_f and u_c values are very similar, showing that in most cases, the more easily computed u_c value is sufficient. The makespan values show a decrease in construction duration in comparison with the wait-action padded plan by u_f. The construction duration decreases on average by approximately 19% for *1-2 timestep*, 6% for *1-2-3 timestep*, and 9% for *TERMES timestep*.

The table also contains the average value and sampling variance of the runtime, showing a steep growth in computational complexity. The results suggest an exponential increase of computation time in regards to makespan, which was also observed in a similar model by (Srinivasan et al., 2023). This is likely caused by linear dependency between the number of MILP variables (r_i and h_i) and model makespan (also noted by (Srinivasan et al., 2023)) and a general MILP problem being NP-hard (Bulut and Ralphs, 2021).

Table 1: Action type durations for used action duration sets.

	Action duration sets			
Action	1	1-2	1-2-3	TERMES
enter	1	2	3	3
leave	1	1	2	3
move_block	1	1	3	3
move_empty	1	1	1	2
pick_up	1	2	3	2
deliver	1	2	3	3

The second experiment aims to evaluate the effects of varying action durations on the makespan estimation accuracy of T_h. For this purpose, all action types, but T_{wait}, are assigned durations $T_a \in \{1,2,3\} = \beta, \forall a \in \mathcal{A} \setminus \{\text{wait}\}$ and all combinations of those durations are measured (requiring $|\beta|^{|A|-1} = 729$ construction plans to be computed). T_{wait} is left as 1, as the minimum robot wait time is not limited. We select a $2 \times 2 \times 2$ cube target structure (figure 2) due to the high number of required measurements.

The results in figure 4 show that T_h both can

Table 2: Experimental results.

Instance	Action duration set	Run-time mean [s]	Run-time sampling variance [s^2]	Makespan lower bound	Makespan with (u_f;u_c upper bound)	Sum-of-costs	Robots
 1	1-timestep	26.75	1.41	6	11 (11; 11)	232	32, 33, 34, 37
	1-2-timestep	355.33	95.16	10	17 (21; 22)	316	32, 33
	1-2-3-timestep	426.32	28.48	15	30 (32; 33)	576	32
	TERMES-timestep	322.73	9.19	18	30 (33; 33)	648	32, 33, 34
 2	1-timestep	24.10	1.05	6	11 (11; 11)	196	32
	1-2-timestep	58.88	0.93	10	17 (21; 22)	284	32
	1-2-3-timestep	287.24	15.76	15	30 (32; 33)	508	32
	TERMES-timestep	264.28	2.83	18	30 (33; 33)	548	32
 3	1-timestep	239.21	47.98	6	14 (14; 14)	142	16, 17
	1-2-timestep	303.92	43.62	10	21 (27; 28)	213	17
	1-2-3-timestep	3807.62	414.87	15	37 (41; 42)	352	17
	TERMES-timestep	6152.27	1132.99	18	38 (41; 42)	398	16, 17

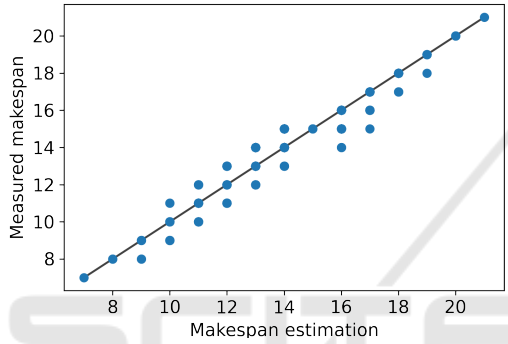


Figure 4: Scatter plot of makespanheuristic relation.

under- and over-estimate the actual makespan value. The root mean square error (RMSE) of T_h is 0.655, with minimum estimation error absolute value, relative to makespan, being 0 and maximum 0.143. The error stems from the simplicity of T_h – it depends only on the action type duration mean and makespan of the generalized model with unit action durations. It does not take into account the usage of the action types.

The last test aims to show another beneficial action duration set for the fraction-time model. The real TERMES use marine foam blocks with indentations/protrusions and neodymium magnets to ensure block alignment and stability (Petersen et al., 2011). The resulting block columns are stable enough that the agents do not need to adapt their movement speed to their height on the structure – at least at the scales used in the paper. For TERMES-like multiagent systems with less stability, the fraction-time model allows to adjust the agent movement speed according to the vertical position of the agent. This is shown on a task with action durations given by the equation 18, ensuring that the action durations grow linearly with the vertical position of the robot at the action's end (except the wait action, where the robot stays still).

The experiment measures 10 run-times with action durations f_{d_h} (referred to as "Unstable columns" in the results) on the instance 1. Makespan, sum-of-costs and run-time are compared with results for "TERMES-timestep" action duration set of instance 1. The "TERMES-timestep" results are called "Base" in the results table 3. The makespan of unstable columns task has increased by 40%, the sum-of-costs by 7.4%, relative to the base. The increase of makespan was expected, due to f_{d_h} consisting of TERMES-timestep f_d with an added positive linear component. The interesting result is the small 7.4% increase in sum-of-costs in relation to the 40% relative increase in makespan. This indicates a lower agent utilization in case of the unstable columns task.

The run-time increased with the makespan. However, it is still notably lower than runtime of instance 3 with TERMES-timestep in table 2, which has makespan 38. This indicates, that while run-time is greatly dependent on makespan, the remaining task-dependent constraints, such as the structure height map and the agent count, also affect the computational complexity to a relatively large degree.

$$f_{d_h}(v) = \begin{cases} 3 & \text{if } v \in Q_{\text{entry}} \cup Q_{\text{leave}} \\ 2+z' & \text{if } v = (\ldots, z') \in Q_{\text{move\_empty}} \\ 3+z' & \text{if } v = (\ldots, z') \in Q_{\text{move\_block}} \\ 2+2z' & \text{if } v = (\ldots, z') \in Q_{\text{pick\_up}} \\ 3+2z' & \text{if } v = (\ldots, z') \in Q_{\text{deliver}} \\ 1 & \text{otherwise.} \end{cases} \quad (18)$$

6 CONCLUSION

A new branch of the multi-agent construction has recently emerged – exact optimization of the problem. The current state-of-the-art exact approach provides solutions with optimal makespan and sum-of-costs

Table 3: Comparison of TERMES task solution where movement and block manipulation speed depends on agent height (unstable columns) with task solution, where movement speed is independent of the agent height (base).

TERMES	Base	Unstable columns
Makespan	30	42
Sum-of-costs	648	696
Mean run-time [s]	322.73	3837.64

but assumes all agent high-level actions to have the same duration. This is not the case in the real world. For instance, the mean duration of block manipulation actions of the TERMES robot differs by 46%.

This has motivated us to generalize the current state-of-the-art exact MILP model (Lam et al., 2020) into the fraction-time model with action durations specified as part of the model input. The generalization is non-trivial – upholding the agent collision constraint has necessitated the redefinition of agent collision using exclusion zones while keeping the model computationally viable required the preservation of the two network flow structures within the model. We test the generalized model on various action durations (including TERMES), studying its behavior. The experiment shows a 9% decrease in construction duration (makespan vs u_f upper bound, which can provide a non-optimal plan) for the TERMES robots. Other action duration sets also show similar results.

The second experiment analyses the heuristic makespan estimation function T_h and its behavior with different action duration sets. The experiment uses toy instance – $2 \times 2 \times 2$ cube and showcases the spread of estimated makespan values in comparison with the real ones. The RMSE of the T_h is 0.665, making it a viable source of information for cost-benefit analysis when considering a more computationally demanding action duration mapping. The last experiment showcases the ability of the model to adjust to agents moving slower at greater heights. The optimal solutions show a notably lower relative utilization of the agents in comparison with a non-height-dependent action duration assignment.

While the model is computationally demanding, it can provide reference solutions for both existing and future heuristic models. These solutions can be used to estimate the heuristic model efficiency and as long-term targets for models optimizing makespan (i.e. construction duration).

ACKNOWLEDGEMENTS

This work has been supported by the project number 22-31346S of the Czech Science Foundation GA ČR and by the CTU project SGS23/210/OHK3/3T/18.

An extended version of this paper is available at (Rameš and Surynek, 2023).

REFERENCES

Barros dos Santos, S. R., Givigi, S., Nascimento, C. L., Fernandes, J. M., Buonocore, L., and de Almeida Neto, A. (2018). Iterative decentralized planning for collective construction tasks with quadrotors. *Journal of Intelligent & Robotic Systems*, 90(1):217–234.

Bulut, A. and Ralphs, T. K. (2021). On the complexity of inverse mixed integer linear optimization. *SIAM Journal on Optimization*, 31(4):3014–3043.

Cai, T., Zhang, D. Y., Kumar, T. S., Koenig, S., and Ayanian, N. (2016). Local search on trees and a framework for automated construction using multiple identical robots. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1301–1302.

Deng, Y., Hua, Y., Napp, N., and Petersen, K. (2019). A compiler for scalable construction by the termes robot collective. *Robotics and Autonomous Systems*, 121:103240.

Forgemind ArchiMedia (2014). Termes robot 01.

Gurobi Optimization, LLC (2023). Gurobi Optimizer Reference Manual.

Jenett, B. and Cheung, K. (2017). Bill-e: Robotic platform for locomotion and manipulation of lightweight space structures. In *25th AIAA/AHS Adaptive Structures Conference*, page 1876.

Lam, E., Stuckey, P., Koenig, S., and Kumar, T. (2020). Exact approaches to the multi-agent collective construction problem. In Simonis, H., editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 743–758. Springer. International Conference on Principles and Practice of Constraint Programming 2020, CP2020 ; Conference date: 07-09-2020 Through 11-09-2020.

Petersen, K. H., Nagpal, R., and Werfel, J. K. (2011). Termes: An autonomous robotic system for three-dimensional collective construction. *Robotics: science and systems VII*.

Piranda, B. and Bourgeois, J. (2018). *Geometrical Study of a Quasi-spherical Module for Building Programmable Matter*, pages 387–400. Springer International Publishing, Cham.

Rameš, M. and Surynek, P. (2023). Action duration generalization for exact multi-agent collective construction. arXiv.org. https://arxiv.org/abs/2312.13485.

Romanishin, J. W., Gilpin, K., and Rus, D. (2013). M-blocks: Momentum-driven, magnetic modular robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4288–4295.

Singh, S., Gutow, G., Srinivasan, A. K., Vundurthy, B., and Choset, H. (2023). Hierarchical propositional logic planning for multi-agent collective construction. In *Construction Robotics Workshop*.

Srinivasan, A. K., Singh, S., Gutow, G., Choset, H., and Vundurthy, B. (2023). Multi-agent collective construction using 3d decomposition.