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

Eventually Optimal and Scalable Multi-Agent Planning for Block Cave Mining

Christopher Leet[1], Paolo Forte[2], Uwe Köckemann[2], Henrik Andreasson[2], Sven Koenig[1,2]

***Abstract*— Automation in underground mining has the potential to significantly enhance safety, operational efficiency, and sustainability. However, effectively coordinating fleets of autonomous vehicles in dynamic mine environments introduces substantial challenges in both optimization and motion planning. To address these challenges, we introduce and formalize the *Block Cave Mining (BCM)* problem, which focuses on computing a transport plan that maximizes ore throughput while satisfying draw ratio constraints. To solve this problem, we propose SAMM, an eventually optimal anytime solver that jointly integrates task assignment, scheduling, and path planning via a mixed-integer linear programming formulation. To improve scalability, we also introduce SAMMS, a variant of SAMM that trades optimality guarantees for efficiency by decomposing the problem into shorter planning subcycles. Experimental evaluations using realistic industrial mine scenarios demonstrate that SAMMS achieves near-optimal throughput and scales effectively to larger fleets and mine layouts.**

I. Introduction

Autonomous material transport systems are becoming increasingly important to mining [1], as they can improve efficiency, safety, and reduce costs [2]. Among the mining methods that benefit from automation is block cave mining (see Fig. 1), a mining technique used to extract large volumes of ore from underground mines [3]. It begins with the excavation of an undercut layer ① and extraction tunnels ② beneath an ore body. Vertical drawbells ③ connect the undercut layer to the extraction tunnels. The undercut layer is then blasted ④ causing the base of the ore body to fracture and collapse. The ore is transported from the drawbells to crushers which pulverize it and deposit it into containers for removal. As ore is extracted, a hollow ⑤ forms at the base of the ore body. This loss of support causes the remainder of the ore body to slowly crumble under its own weight and fall into the drawbells for collection.

To automate the ore collection process, a block cave mine operator must generate a plan that specifies how autonomous agents transport ore to crushers. This production plan is to alleviate stress level build up in the mine as well as fragmentation of the ore, the amount of material drawn from each drawbell is important to keep within bounds and changes slowly over time [4]. In this work we assume the rate at which ore is extracted is a fixed percentage of the total extraction rate (if the rates are updated we treat it as a new problem). These percentages vary between drawbells, for example, one drawbell might contribute 7% of the extracted ore while another might only contribute 1%. We define the task of finding a transport plan which maximizes the overall extraction rate as the *Block Cave Mining (BCM)* problem. The BCM problem presents three challenges. First, since agents must haul ore for an indefinite amount of time, a solver must construct an arbitrarily long plan. Second, the BCM problem is composed of three, tightly coupled, components: task assignment (which drawbell to visit), scheduling (when to visit) and path planning (how to get there). Finally, BCM agents have heavy restrictions on their movement. Most prior work on *Multi-Agent Path Finding (MAPF)* models the environment as a 2D grid graph with holes and assumes that agents can move in any cardinal direction in constant time. BCM agents, however, are bulky, center articulated vehicles only slightly narrower than the mine's tunnels. Even basic actions such as turning around require a complex series of maneuvers at a junction, complicating path planning.

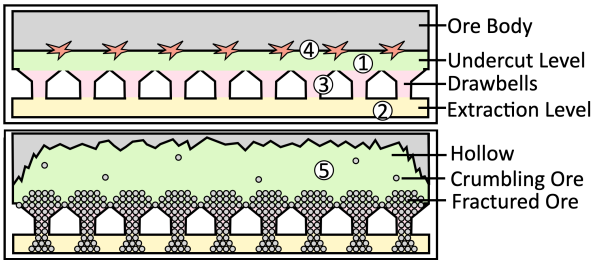

Fig. 1. A side view of the block cave mining process. Top: building the infrastructure. Bottom: extracting ore.

To the best of our knowledge, no optimal solution to the BCM problem exists. Current solutions use heuristics such as statically assigning agents to fixed subsets of drawbells, which limits optimality. To bridge this gap, we introduce SAMM (Simple Anytime Mine MAPF), an eventually optimal method designed to meet the challenges above. SAMM achieves continuous operation by producing cyclic plans, plans that can be looped because they start and end with the team of agents in the same configuration. It avoids the pitfalls of decomposition by casting task assignment, scheduling and path planning into a single *Mixed Integer Linear Program (MILP)* which captures their inter-dependencies and decides the optimal number of agents for a given cycle length. Finally, to model maneuvering in confined tunnels, SAMM constructs a state graph over discrete poses (position–orientation pairs), where transitions encode feasible motion primitives executable in constant time. The industrial applicability of MILP approaches, such as SAMM, to MAPF

This work was partially supported by the European Union's Horizon Europe Framework Programme under grant agreement No 101070596 (euROBIN). *(Corresponding author: Christopher Leet)*

[1]University of Southern California Los Angeles, California, USA {cjleet,skoenig}@usc.edu

[2]Centre for Applied Autonomous Sensor Systems, Örebro University, Örebro, Sweden <name>.<surname>@oru.se

is limited since they typically do not scale well.

Therefore, we propose an additional, scalable solver, SAMMS (SAMM with Subcycles). The difficulty of generating a cyclic plan using a ILP grows exponentially with the plan's length. Instead of generating one long cycle which services every drawbell, SAMMS generates a sequence of subcycles, each of which services a subset of drawbells. Each subcycle starts and ends in the same state, allowing them to be seamlessly concatenated. SAMMS constructs a transport plan online by concatenating these subcycles. Whenever a subcycle completes, SAMMS selects a new subcycle to execute. We prove that SAMMS is always asymptotically feasible and converges rapidly. The longer it runs, the closer it approximates each drawbell's targeted draw rate.

We evaluate SAMM and SAMMS on benchmark scenarios derived from real-world mine layouts and demonstrate their ability to solve industrial scale BCM instances.

II. Related Work

Multi-Agent Path Finding (MAPF) [5] is the problem of computing collision-free paths for multiple agents in a shared environment. MAPF has been solved in different industrial domains, including warehouse logistics [6], transportation systems [7], and construction [8]. Classical MAPF solutions mainly rely on two main paradigms: search-based and compilation-based approaches. Search-based methods emphasize solution optimality by employing graph search, heuristics, and specialized techniques to minimize collisions and search overhead. Examples include prioritized planning [9], conflict-based search [10], and priority-based search [11]. Compilation-based approaches reformulate MAPF as an optimization problem, such as Boolean satisfiability [12], mixed-integer linear programming [13], or answer set programming [14]. While these methods can yield high-quality or even provably optimal solutions, they often face scalability issues on larger instances. More recently, the field has witnessed an increasing interest in learning-based methods, including reinforcement learning [8] and foundation model driven approaches [6]. However, most existing MAPF algorithms make simplifying assumptions that do not hold in block cave mining. They generally assume grid-based navigation, a fixed number of agents, and that task assignment and scheduling are precomputed, focusing solely on path planning. Our approach jointly optimizes the number of agents, task assignment, scheduling, and path planning, and produces plans that can be executed continuously.

Various approaches have been explored for underground mine planning and scheduling [15], [**?**]. Production scheduling methods can be broadly categorized into heuristic methods and exact optimization techniques. Heuristic approaches aim to generate near-optimal solutions within a reasonable time, particularly when identifying the optimal solution under complex constraints is computationally infeasible. Examples include block aggregation heuristics [16] and optimization-based decomposition heuristics [17]. While computationally efficient, these methods cannot guarantee globally optimal schedules. Conversely, exact mathematical programming methods have been widely applied to block-caving production scheduling, as they allow systematic exploration of feasible solutions while accounting for operational constraints and providing bounds on optimality. Techniques such as linear programming [4], and quadratic programming [18] have been proposed to optimize production plans. However, these approaches neither generate optimal transport plans nor scale.

III. Problem Formulation

We now formally present the block cave mining problem.

A. Mine

A block cave mine is a collection of drawbells and ore crushers connected by a network of narrow tunnels.

Drawbells. A drawbell is a shaft filled with crushed ore. Let $D := \{d_1, d_2, \ldots\}$ be the set of drawbells in a mine. Let draw rate λ_i be the average rate at which ore is extracted from drawbell d_i. Each drawbell's draw rate is coupled to the draw rates of the other drawbells. Let $\Lambda := \sum_{d_i \in D} \lambda_i$ be the total ore extraction rate. Each drawbell d_i's extraction rate must be a fixed fraction $r_i := \lambda_i / \Lambda$ of the total. Let r_i be the drawbell d_i's draw ratio, and the vector of draw ratios $R := \langle r_1, r_2, \ldots \rangle$ be the mine's draw ratio configuration.

Crushers. A crusher grinds fractured ore and then deposits it in a container for extraction. Let $C := \{c_1, c_2, \ldots\}$ be the set of crushers in the mine. Ore can be transported from any drawbell to any crusher.

Constant Time Roadmap. We model the routes that agents in the mine can follow using a Constant Time Roadmap (CTR), i.e., a directed graph $G := (Q, E)$ where each vertex $q_i \in Q$ represents a pose (a position and orientation pair). There is an edge $(q_i, q_j) \in E$ iff an agent can move from pose q_i to pose q_j within τ seconds and without passing through any other pose. Time in a block cave mine is discretized. One timestep lasts τ. Thus an agent can always move from any pose to any adjacent pose in exactly 1 timestep.

A drawbell and a crusher is accessible from a pose q_i if an agent at q_i can load ore into it and unload ore from it. Let $\bar{D}(q_i) \subseteq D$ and $\bar{C}(q_i) \subseteq C$ denote the drawbells and crushers serviceable from q_i. A valid CTR has the following properties: (1) The roadmap is strongly connected. If the roadmap is disconnected, it may be impossible to carry ore from some drawbell to a crusher; and (2) each drawbell/crusher must be accessible from at least one pose, that is: $C = \bigcup_{q_i \in Q} \bar{C}(q_i)$ and $D = \bigcup_{q_i \in Q} \bar{D}(q_i)$.

B. Agents

A team of agents $A := \{a_1, a_2, \ldots\}$ transports ore between drawbells and crushers. Each agent is an autonomous vehicle. An agent can perform four types of actions:

Wait. The agent idles in its current pose for one timestep.

Move. The agent moves to an adjacent pose in the roadmap.

Pick Up. The agent picks up ore from a drawbell $d_j \in \bar{D}(q_i)$ accessible from its current pose q_i.

Deposit. The agent deposits ore into a crusher $c_j \in \bar{C}(q_i)$ accessible from its current pose q_i.

Each agent is equipped with a shovel that has a capacity of exactly one ore unit. Shovels cannot be partially filled, and each pickup or deposit action transfers precisely one unit between a drawbell and the agent, or the agent and a crusher. Picking up ore from drawbell d_j takes T_{pk} timesteps, and depositing it into crusher c_j takes T_{dp} timesteps. If an agent at pose q_i begins a pickup or deposit action, it occupies pose q_i until that action is completed. The state of an agent a_i at timestep t is a pair (π_{it}, σ_{it}) where $\pi_{it} \in Q$ denotes the agent's pose and σ_{it} denotes the state of its shovel (0 for empty, and 1 for full).

C. Collision

Two poses $q_i, q_j \in Q$ conflict iff two agents cannot occupy them simultaneously without colliding. We represent pose conflicts with a pose conflict family $\mathcal{C}^Q \subseteq 2^Q$. Each element of $\mathcal{C}^Q$ is a pose conflict set S, a set of pairwise conflicting poses. Any set of pairwise conflicting poses $S' \subseteq Q$ is a subset of some pose conflict set $S \in \mathcal{C}^Q$. A pose conflicts with itself. If a pose q_i does not conflict with any other pose, the singleton set $\{q_i\}$ is contained in $\mathcal{C}^Q$ to represent q_i's self-conflict. Consequently, every pose appears in a set in $\mathcal{C}^Q$ at least once.

Similarly, two edges $(q_h, q_i), (q_j, q_k) \in E$ conflict iff two agents cannot traverse them simultaneously without colliding. We represent edge conflicts with an edge conflict family $\mathcal{C}^E \subseteq 2^E$. Each element of $\mathcal{C}^E$ is an edge conflict set of pairwise conflicting edges. Any set S of pairwise conflicting edges in E is a subset of some edge conflict set in $\mathcal{C}^E$. To prevent collisions, no two agents may occupy poses in the same pose conflict set in $\mathcal{C}^Q$ at the same timestep or move along edges in the same edge conflict set in $\mathcal{C}^E$ from the same timestep to the next one.

D. Transport Plan

A transport plan $(\boldsymbol{\pi}, \boldsymbol{\sigma})$ describes how agents move ore from drawbells to crushers. It defines the state (π_{it}, σ_{it}) of each agent $a_i \in A$ at every timestep t. As ore must be transported continuously, we generate a cyclic transport plan that can be repeated indefinitely. A cyclic transport plan is associated with a cycle time $T \in \mathbb{N}$ and an agent permutation $\Omega : A \to A$. The transport plan runs from $t = 0$ to $t = T$. The state of each agent $a_i \in A$ at $t = 0$ must be the same as the state of agent $\Omega(a_i)$ at $t = T$, that is,

$$\forall\ a_i \in A, (\pi_{i0}, \sigma_{i0}) = (\pi_{jT}, \sigma_{jT}) \text{ for } a_j = \Omega(a_i).$$

A cyclic transport plan is repeated indefinitely by executing it from $t = 0$ to $t = T$, relabeling its agents according to Ω, and then repeating the process. A cyclic transport plan is valid iff: (1) it can be realized by assigning each agent a sequence of wait, move, pickup and deposit actions, and (2) no two agents occupy conflicting poses or move along conflicting edges at any time t.

E. The Block Cave Mining (BCM) Problem

A BCM problem can be formalized as a 5-tuple $\mathcal{I} := (Q, E, D, C, R)$, where (Q, E) is a constant time roadmap, D and C are its drawbells and crushers, and R is a draw ratio configuration. The throughput of a cyclic transport plan $(\boldsymbol{\pi}, \boldsymbol{\sigma})$ with cycle length T is the average units of ore deposited per timestep. If $\text{DEPOSITS}(\boldsymbol{\pi}, \boldsymbol{\sigma})$ is the number of deposit actions in the plan $(\boldsymbol{\pi}, \boldsymbol{\sigma})$, the plan's throughput is $\text{DEPOSITS}(\boldsymbol{\pi}, \boldsymbol{\sigma})/T$. In the Block Cave Mining problem, we are given an BCM instance $\mathcal{I}$ and asked to find a valid cyclic transport plan that maximizes throughput. Let $\Xi(\mathcal{I}, T)$ be the set of valid transport plans for instance $\mathcal{I}$ with cycle length T. The BCM is formulated as:

$$\max_{T \in \mathbb{N}, (\boldsymbol{\pi}, \boldsymbol{\sigma}) \in \Xi(\mathcal{I}, T)} \frac{\text{DEPOSITS}(\boldsymbol{\pi}, \boldsymbol{\sigma})}{T}.$$

IV. Block Cave Mine Benchmarks

In this section, we describe a family of benchmark block cave mine layouts [19]. This family illustrates the layout of real-world block cave mines, composed of three components: drawbell tunnels, crusher tunnels, and a main tunnel.

Drawbell Tunnels. Drawbell tunnels are linear, dead-end tunnels with drawbells positioned at regular intervals along both sides. These tunnels are narrow, preventing agents from turning around and restricting the drawbells that they can access. In our mine layout, the drawbells on one side of a tunnel can only be accessed by agents facing its entrance, while the drawbells on the opposite side can only be accessed by agents facing its dead end. Consequently, an agent must reverse into a tunnel to access half of its drawbells.

There is a bijection between the poses and the drawbells in a drawbell tunnel. Each pose lies directly outside its associated drawbell. Each drawbell is only accessible from its associated pose. The transitions between these poses form two simple paths: one for poses oriented toward the entrance and the other for poses oriented toward the dead end. Because drawbell tunnels are narrow, an agent following one path cannot pass an agent following the other. Similarly, an agent picking up ore from a drawbell blocks traffic from moving past it. Fig. 2 depicts a small mine which has two drawbell tunnels and eight drawbells. The paths of poses oriented towards the entrance and dead end of each drawbell tunnel are shown in red and yellow.

Main Tunnel. In a block cave mine, the drawbell tunnels run parallel to one another. Each drawbell tunnel connects to a main tunnel which runs perpendicular to them. Two paths of poses oriented in opposite directions run the length of the main tunnel. An agent on one of these paths cannot pass an agent on the other. Each path of poses in a drawbell tunnel connects to both paths of poses in the main tunnel. Fig. 2 depicts the poses in the mine's main tunnel in orange.

Crusher Tunnels. Crusher tunnels are short passages that terminate at a crusher. Each crusher tunnel contains a single pose that provides access to its crusher. The three-way junction at the mouth of each drawbell and crusher tunnel allows agents to reorient. The mine in Fig. 2 has a single crusher tunnel, whose pose is shown in blue.

We present a benchmark generatorfor block cave mines based on the layout described above. It allows for systematic

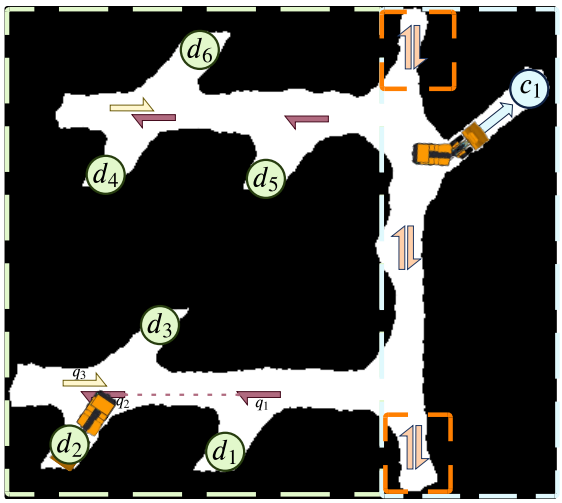

Fig. 2. An example mine. Drawbell tunnels are circled in green; the main tunnel and crusher tunnel are circled in blue. Vehicles cannot turn around without complicated maneuvers, facilitated by the poses circled in orange.

variation of the number and density of drawbell tunnels, drawbells per tunnel, and crusher tunnels.

V. The Simple Anytime Mine MAPF Solver

In this section, we introduce SAMM (the Simple Anytime Mine MAPF), an anytime BCM solver, and prove that it is eventually optimal. We propose a way to accelerate SAMM, and prove that it remains eventually optimal.

We construct SAMM as follows. Let the fixed cycle length (F-BCM) BCM problem be the problem of finding the best solution to a BCM instance $\mathcal{I}$ which has cycle length T. We introduce FLIMM (Fixed Length ILP-based Mine MAPF),a solver designed to find the optimal solution to the F-BCM. SAMM solves the BCM by running FLIMM with progressively higher values of T until time runs out.

A. The Fixed Length ILP-based Mine MAPF Solver

FLIMM solves the F-BCM problem by formulating it as an Integer Linear Program (ILP) as follows.

Variables. FLIMM's ILP introduces the following variables. Let $\Phi \in \{0,1\}$ be set of possible shovel states and $\phi \in \Phi$ be an arbitrary shovel state.

Occupancy Tensor. The tensor $\boldsymbol{at}$ is a $|Q| \times |\Phi| \times T$ tensor. The field $at(q_i, \phi, t) \in \{0,1\}$ indicates if an agent with shovel state ϕ is at pose q_i at timestep t.

Movement Tensor. The tensor $\boldsymbol{mv}$ is a $|E| \times |\Phi| \times T$ tensor. The field $mv(q_i, q_j, \phi, t) \in \{0,1\}$ indicates if an agent with shovel state ϕ moves from pose q_i to pose q_j on timestep t.

Pickup and Deposit Tensors. The tensors $\boldsymbol{pk}$ and $\boldsymbol{dp}$ are $|Q|\times|D|\times T$ and $|Q|\times|C|\times T$ tensors. The fields $pk(q_i, d_j, t)$ and $dp(q_i, c_j, t)$ indicate if an agent at pose q_i picks up ore from drawbell d_j and deposits it in crusher c_j on timestep t.

Objective Function. The objective is to maximize the rate at which ore is removed from the mine. Since the ratio between each draw rate $\lambda_i \in \{d_i\}_{i=2}^{|D|}$ and λ_1 is fixed, this is equivalent to maximizing the rate at which ore is picked up from drawbell d_1. Let $\bar{Q}_i$ be the set of poses from which a drawbell $d_i \in D$ can be accessed.

$$\max \sum_{q_j \in \bar{Q}_1} \sum_{t=0}^{T-1} pk(q_j, d_1, t)$$

Action Selection Constraints. These constraints limit the actions that an agent may take in a given state. Let $\text{ADJ}(q_i)$ denote the set of poses adjacent to pose q_i in the CTR.

Constraint 1. An agent at pose q_i on timestep t with an empty shovel must move to an adjacent pose $q_j \in \text{ADJ}(q_i)$, wait at q_i, or pick up ore from a drawbell $d_j \in \bar{D}(q_i)$ accessible from pose q_i. Every pose in the CTR has a self-loop edge since an agent at pose q_i can always reach q_i in τ time. The set of variables $\{mv(q_i, q_j, 0, t)\}_{q_j \in \text{ADJ}(q_i)}\}$ thus represents both move actions and wait actions.

$$\forall\ q_i \in Q, \forall\ t \in [0..T-1],$$
$$at(q_i, 0, t) = \sum_{q_j \in \text{ADJ}(q_i)} mv(q_i, q_j, 0, t) + \sum_{d_j \in \bar{D}(q_i)} pk(q_i, d_j, t).$$

Constraint 2. An agent at pose q_i on timestep t with a full shovel must move to an adjacent pose, wait at pose q_i, or deposit ore into a crusher accessible from q_i.

$$\forall\ q_i \in Q, \forall\ t \in [0..T-1],$$
$$at(q_i, 1, t) = \sum_{q_j \in \text{ADJ}(q_i)} mv(q_i, q_j, 1, t) + \sum_{c_j \in \bar{C}(q_i)} dp(q_i, c_j, t).$$

Constraint 3. FLIMM models a T_{pk} timestep pickup action as a one timestep pickup action followed by $T_{pk} - 1$ one timestep wait actions. An agent picking up ore from a drawbell $d_j \in \bar{D}(q_i)$ at pose q_i on timestep t must therefore wait at q_i for the next $T_{pk} - 1$ timesteps.

$$\forall\ q_i \in Q,\ \forall\ d_j \in \bar{D}(q_i),\ \forall\ s \in [1..T_{pk} - 1],$$
$$\forall\ t \in [0..T-1],\ mv(q_i, q_i, 0, (t+s)\%T) \geq pk(q_i, d_j, t).$$

Constraint 4. Similarly, FLIMM models a T_{dp} deposit action as a one timestep deposit action followed by $T_{dp} - 1$ one timestep wait actions. An agent depositing ore into a crusher $c_j \in \bar{C}(q_i)$ at pose q_i on timestep t must thus have waited in pose q_i for the preceding $T_{dp} - 1$ timesteps.

$$\forall\ q_i \in Q,\ \forall\ c_j \in \bar{C}(q_i),\ \forall\ s \in [1..T_{dp} - 1],$$
$$\forall\ t \in [0..T-1],\ mv(q_i, q_i, 0, (t-s)\%T) \geq dp(q_i, c_j, t).$$

Since picks up and depositing ore is usually quick, T_{pk} and T_{dp} is usually 1 (as in our evaluations). When $T_{pk} = 1$ and $T_{dp} = 1$, the range of values $[1..0] = \{\}$ that s takes in Constraints 3 and 4 is empty and the constraint vanishes.

Next State Constraints. These constraints determine the locations of the team of agents on timestep $t+1\%T$ based on the actions that were taken on timestep t.

Constraint 5. There is an agent at pose q_i with an empty shovel on timestep $t+1\%T$ if an agent with an empty shovel moved to q_i or an agent at q_i deposited ore on timestep t.

$$\forall\ q_i \in Q, \forall\ t \in [0..T-1], at(q_i, 0, t+1\%T)$$
$$= \sum_{q_j \in \text{ADJ}(q_i)} mv(q_j, q_i, 0, t) + \sum_{c_j \in \bar{C}(q_i)} dp(q_i, c_j, t).$$

Algorithm 1 CONVFLIMM($\boldsymbol{at}, \boldsymbol{mv}, \boldsymbol{pk}, \boldsymbol{dp}$)

1: $n \leftarrow \sum_{q_i \in Q} \sum_{\phi \in \Phi} at(q_i, \phi, t)$
2: $(\boldsymbol{\pi}, \boldsymbol{\sigma}) \leftarrow$ two empty $n \times |T|$ matrices
3: $j \leftarrow 0$
4: **for** $(q_i, \phi) \in Q \times \Phi$ **do**
5: **if** $at(q_i, \phi, 0) = 1$ **then**
6: $(\pi_{j0}, \sigma_{j0}) \leftarrow (q_i, \phi)$
7: $j \leftarrow j + 1$
8: **for** $(j, t) \in [0..n-1] \times [0..T-1]$ **do**
9: $(\pi_{j(t+1)}, \sigma_{j(t+1)}) \leftarrow$ NXTSTATE$(\pi_{jt}, \sigma_{jt}, t, \boldsymbol{mv}, \boldsymbol{pk}, \boldsymbol{dp})$
10: $\Omega \leftarrow$ an empty length n vector
11: **for** $j, k \in [0..n-1] \times [0..n-1]$ **do**
12: **if** $(\pi_{jT}, \sigma_{jT}) = (\pi_{k0}, \sigma_{k0})$ **then**
13: $\Omega(a_j) \leftarrow a_k$
14: **return** $(\boldsymbol{\pi}, \boldsymbol{\sigma}, \Omega)$
15: **function** NXTSTATE$(q_i, \phi, t, \boldsymbol{mv}, \boldsymbol{pk}, \boldsymbol{dp})$
16: **for** $q_j \in$ ADJ(q_i) **do**
17: **if** $mv(q_i, q_j, \phi, t) = 1$ **then**
18: **return** (q_j, ϕ)
19: **for** $d_j \in \bar{D}(q_i)$ **do**
20: **if** $\phi = 0 \wedge pk(q_i, d_j, t) = 1$ **then**
21: **return** $(q_i, 1)$
22: **for** $c_j \in \bar{C}(q_i)$ **do**
23: **if** $\phi = 1 \wedge dp(q_i, d_j, t) = 1$ **then**
24: **return** $(q_i, 0)$

Constraint 6. There is an agent at pose q_i with a full shovel on timestep $t + 1\%T$ if an agent with a full shovel moved to q_i or an agent at q_i picked up ore on timestep t.

$$\forall\ q_i \in Q, \forall\ t \in [0..T-1],\ at(q_i, 1, t + 1\%T) = \sum_{q_j \in \text{ADJ}(q_i)} mv(q_j, q_i, 1, t) + \sum_{d_j \in \bar{D}(q_i)} pk(q_i, d_j, t).$$

Collision Constraints. These constraints prevent collisions.
Constraint 7. At most one agent may occupy a pose in any pose conflict set $S \in \mathcal{C}^Q$ on any timestep t.

$$\forall\ S \in \mathcal{C}^Q, \forall\ t \in [0..T-1], \sum_{q_i \in S} \sum_{\phi \in \Phi} at(q_i, \phi, t) \leq 1.$$

Constraint 8. At most one agent may transition across an edge in any edge conflict set $S \in \mathcal{C}^E$ on any timestep t.

$$\forall\ S \in \mathcal{C}^E, \forall\ t \in [0..T-1], \sum_{(q_i, q_j) \in S} \sum_{\phi \in \Phi} mv(q_i, q_j, \phi, t) \leq 1.$$

Draw Ratio Constraint. This constraint enforces the relative rates at which ore is removed from each drawbell.
Constraint 9. The ratio between the draw rate from each drawbell d_i and the reference drawbell d_1 must be r_i. Recall that $\bar{Q}_i$ be the set of poses from which d_i can be accessed.

$$\forall d_i \in D, \sum_{q_j \in \bar{Q}_i} \sum_{t=0}^{T-1} pk(q_j, d_i, t) = r_i \sum_{q_j \in \bar{Q}_i} \sum_{t=0}^{T-1} pk(q_j, d_1, t).$$

From Decision Variables to a Plan. Once FLIMM's ILP is solved, the values assigned to its decision variables encode an optimal solution to the F-BCM. We translate these decision variable values into a formal plan using the function CONVFLIMM, shown in Algorithm 1.

Algorithm 2 SAMM($\mathcal{I}, timer$)

1: $(\boldsymbol{\pi}^*, \boldsymbol{\sigma}^*, \Omega^*, T) \leftarrow$ (NULL, NULL, NULL, 1)
2: **while** $timer$ has not expired **do**
3: $(\boldsymbol{at}, \boldsymbol{mv}, \boldsymbol{pk}, \boldsymbol{dp}) \leftarrow$ FLIMM$(\mathcal{I}, T, timer)$
4: $(\boldsymbol{\pi}, \boldsymbol{\sigma}, \Omega) \leftarrow$ CONVFLIMM$(\boldsymbol{at}, \boldsymbol{mv}, \boldsymbol{pk}, \boldsymbol{dp})$
5: **if** $\boldsymbol{\pi} =$ NULL $\vee$ THRU$(\boldsymbol{\pi}, \boldsymbol{\sigma}, T) >$ THRU$(\boldsymbol{\pi}^*, \boldsymbol{\sigma}^*, T)$ **then**
6: $(\boldsymbol{\pi}^*, \boldsymbol{\sigma}^*, \Omega^*) \leftarrow (\boldsymbol{\pi}, \boldsymbol{\sigma}, \Omega)$
7: $T \leftarrow T + 1$
8: **return** $(\boldsymbol{\pi}^*, \boldsymbol{\sigma}^*, \Omega^*)$
9: **function** THRU$(\boldsymbol{\pi}, \boldsymbol{\sigma}, T)$
10: **return** DEPOSITS$(\boldsymbol{\pi}, \boldsymbol{\sigma})/T$

CONVFLIMM iterates over the variables $\{at(q_i, \phi, 0) : q_i \in Q, \phi \in \phi\}$ (Lines 4-5). Whenever it encounters a variable with the value 1, it instantiates a new agent a_j. Agent a_j's initial pose and shovel state are set to $\pi_{j0} = q_i$ and $\sigma_{j0} = \phi$ (Line 6). Next, CONVFLIMM determines the action assigned to agent a_j on timestep $t = 0$. If:

- $mv(q_i, q_i, \phi, 0)$ is true, agent a_j waits at pose q_i. Thus $\pi_{j1} = q_i$ and $\sigma_{j1} = \phi$. Since each pose has a self-loop, this check occurs in Lines 16-18.
- $mv(q_i, q_j, \phi, 0)$ is true for some pose $q_j \in$ ADJ(q_i), agent a_j moves from q_i to q_j. Thus $\pi_{j1} = q_j$ and $\sigma_{j1} = \phi$ (Lines 16-18).
- agent a_j's shovel is empty $\sigma_{j0} = 0$ and $pk(q_i, d_k, 0) = 1$ for some $d_k \in \bar{D}(q_i)$, agent a_j picks up ore from drawbell d_k. Thus $\pi_{j1} = q_i$ and $\sigma_{j1} = 1$ (Lines 19-21).
- agent a_j's shovel is full $\sigma_{j0} = 1$ and $dp(q_i, c_k, 0)$ is true for some $c_k \in \bar{C}(q_i)$, then a_j deposits ore at crusher c_k. Thus $\pi_{j1} = q_i$ and $\sigma_{j1} = 0$ (Lines 22-24).

Constraint 5 ensures that exactly one of these conditions holds. ConvFLIMM repeats this procedure to determine the state of agent a_j on timesteps $t = 2$ to T. After extracting FLIMM's transport plan from its decision variables, ConvFLIMM determines its agent permutation Ω. An agent $a_j \in A$ is mapped to the agent a_k if its terminal state (π_{jT}, σ_{jT}) matches its initial state (π_{k0}, σ_{k0}) (Line 10-13).

B. The Simple Anytime Mine MAPF (SAMM) solver

We construct SAMM using our fixed-cycle-length BCM solver, as outlined in Algorithm 2. SAMM iteratively runs FLIMM with progressively increasing cycle lengths until the time limit $timer$ is reached, then returns the best plan found. If $timer$ expires while FLIMM is running, it returns NULL.

Theorem 1: SAMM is eventually optimal.

Proof. Let $(\boldsymbol{\pi}^*, \boldsymbol{\sigma}^*)$ be the optimal transport plan for the BCM instance $\mathcal{I}$. Let T^* be the cycle length associated with this plan. Since SAMM enumerates cycle lengths in increasing order, it will eventually call FLIMM on cycle length T^*. FLIMM will return $(\boldsymbol{\pi}^*, \boldsymbol{\sigma}^*)$ since it is guaranteed to find the optimal plan for any fixed cycle length. □

It is usually unnecessary to initialize SAMM with a cycle length of $T = 1$ since very short cycle lengths are typically impractical. Our evaluations initialize SAMM with $T = 6$.

C. Accelerating SAMM

FLIMM constrains each drawbell's draw rate d_i to r_i times the reference drawbell's draw rate d_1. If each ratio r_i is

Algorithm 3 SAMMS($\mathcal{I} := (Q, E, D, C, R), timer$)

1: $\mathcal{D} \leftarrow$ PARTITIONDRAWBELLS(D)
2: $\{D'_1, D'_2, \ldots\} \leftarrow \mathcal{D}$
3: $\xi_1 \leftarrow$ GENREFSUBCYCLE($\mathcal{I}, D'_1, , timer$)
4: $(\boldsymbol{\pi}^{(1)}, \boldsymbol{\sigma}^{(1)}, \Omega^{(1)}) \leftarrow \xi_1$
5: $\bar{\boldsymbol{\pi}} \leftarrow \{\pi^{(1)}_{j0} : j \in [0..n-1]\}$
6: $\bar{\boldsymbol{\sigma}} = \{\sigma^{(1)}_{j0} : j \in [0..n-1]\}$
7: **for** $D'_i \in \mathcal{D}$ *s.t.* $i \neq 1$ **do**
8: $\quad \xi_i \leftarrow$ GENSUBCYCLE($\mathcal{I}, D'_i, \bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\sigma}}, timer$)
9: **return** COMPOSESUBCYCLES($\{\xi_i\}_{D'_i \in \mathcal{D}}$)
10: **function** GENREFSUBCYCLE($\mathcal{I}, D'_1, timer$)
11: $\quad \mathcal{I}'_1 \leftarrow$ REDUCEBCM($\mathcal{I}, D'_1$)
12: $\quad$ **return** SAMM($\mathcal{I}'_1, timer$)
13: **function** GENSUBCYCLE($\mathcal{I}, D'_i, \bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\sigma}}, timer$)
14: $\quad \mathcal{I}'_1 \leftarrow$ REDUCEBCM($\mathcal{I}, D'_i$)
15: $\quad$ **return** FS-SAMM($\mathcal{I}'_i, \bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\sigma}}, timer$)

simple (e.g., 3/2), these rates can often be achieved using a plan with a short cycle length. However, complex ratios (e.g., 97/100) may require a long cycle length. FLIMM's worst case runtime grows exponentially with cycle length.

To address this issue, we introduce Flexible Draw Rate FLIMM (FDR-FLIMM), a variant of FLIMM that allows each drawbell's draw rate d_i to be higher than the target rate $r_i d_1$, relaxing Constraint 9 to read:

$$\forall d_i \in D, \sum_{q_j \in \bar{Q}_i} \sum_{t=0}^{T-1} pk(q_j, d_i, t) \geq r_i \sum_{q_j \in \bar{Q}_i} \sum_{t=0}^{T-1} pk(q_j, d_1, t).$$

We enforce the target draw rate for each drawbell at execution time by occasionally skipping pickup actions. Let $z_i(t)$ be the average rate at which ore has been extracted from drawbell d_i from timestep 0 to timestep t. On any timestep t, if executing a pickup action would cause $z_i(t)$ to exceed the target rate $r_i d_1$, we replace it with T_{pk} wait actions.

Theorem 2: The throughput of FDR-FLIMM's solution to any fixed cycle length BCM instance $(\mathcal{I}, T)$ is no worse than the throughput of FLIMM's solution.

Proof. FDR-FLIMM is obtained by relaxing one of FLIMM's constraints. Relaxing a constraint cannot eliminate any feasible solutions. Thus any feasible solution to FLIMM is a feasible solution to FDR-FLIMM. Both FLIMM and FDR-FLIMM maximize the reference draw rate d_1. It follows that the draw rate d_1 achieved by FDR-FLIMM is no lower than the draw rate achieved by FLIMM.

The throughput of the solutions produced by both solvers is a multiple of d_1. FLIMM enforces this relationship through its constraints, while FDR-FLIMM enforces it by selectively skipping pickup actions at execution time. Therefore, the throughput of FDR-FLIMM's solution is no lower than the throughput of FLIMM's solution. □

Let FDR-SAMM be the variant of SAMM in which FLIMM is replaced by FDR-FLIMM. Theorem 2 implies that FDR-SAMM is also eventually optimal.

VI. SAMM WITH SUBCYCLES

In this section, we introduce SAMMS (SAMM with Subcycles), a scalable solution to the block cave mining problem. We then establish that SAMMS is complete.

Algorithm 4 REDUCEBCM($\mathcal{I} := (Q, E, D, C, R), D'_i$)

1: **for** $q_i \in Q$ **do**
2: $\quad \bar{D}'_i(q_i) \leftarrow \bar{D}(q_i) \setminus D'_i$
3: $\psi_i \leftarrow \arg\max\{r_i : d_i \in D\}$.
4: $\rho_i \leftarrow \max\{r_i : d_i \in D\}$.
5: $R'_i \leftarrow \{r_j / \rho_i : d_j \in D'_i\}$
6: **return** (Q, E, D'_i, C, R'_i)

The computational complexity of constructing a cyclic plan with an ILP grows exponentially with its cycle length T. Instead of producing a single cyclic plan that services every drawbell, SAMMS constructs a sequence of subcycles $\Xi := \{\xi_1, \xi_2, \ldots\}$, each of which services a subset of drawbells. SAMMS constructs a transport plan online by concatenating subcycles from Ξ. Whenever a subcycle completes, SAMMS selects a new subcycle to execute. SAMMS is shown in Alg. 3. It has four main steps:

Step 1.) SAMMS partitions the set of drawbells D into families using the function PARTITIONDRAWBELLS (Line 1). Let the set of drawbell families be $\mathcal{D} := \{D'_1, D'_2, \ldots\}$ (Line 2). Each drawbell d_j belongs to exactly one family.

Step 2.) Let the reference family D'_1 be the family that contains the reference drawbell d_1. SAMMS constructs a subcycle $\xi_1 := (\boldsymbol{\pi}^{(1)}, \boldsymbol{\sigma}^{(1)}, \Omega^{(1)})$ that services the reference family using the function GENREFSUBCYCLE (Line 3). This subcycle is termed the reference subcycle.

Step 3.) SAMMS constructs subcycles $\{\xi_i\}_{i \neq 1}$ for the remaining families $\{D'_i\}_{i \neq 1}$. Let n be the number of agents used by the reference subcycle. Let $(\bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\sigma}})$ be the state of these agents at $t = 0$ (Lines 5-6):

$$\bar{\boldsymbol{\pi}} = \{\pi^{(1)}_{j0} : j \in [0..n-1]\}, \quad \bar{\boldsymbol{\sigma}} = \{\sigma^{(1)}_{j0} : j \in [0..n-1]\}$$

To ensure that the subcycles that SAMMS constructs can be seamlessly composed into a single global plan, each of these subcycles must: (a) use the same number of agents as the reference subcycle and (b) start and end with their team of agents in the state $(\bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\sigma}})$. SAMMS constructs these subcycles using the function GENSUBCYCLE (Lines 7-8).

Step 4.) Finally, SAMMS generates a transport plan online by selecting subcycles to run in real time using the function COMPOSESUBCYCLES (Line 9). SAMMS ensures each drawbell's draw rate conforms to the draw ratio r_i by running each subcycle at the appropriate frequency. We now describe the functions associated with each of SAMMS's four steps.

PARTITIONDRAWBELLS. SAMMS assigns each subcycle g drawbells chosen at random from each drawbell tunnel, where g is a tunable hyperparameter. In block cave mining, overall throughput is typically limited by crusher access. Because drawbells are selected randomly, subcycles typically include drawbells both near and farther away from the main tunnel. This variation in travel distances helps stagger agent arrivals at the crushers, improving their utilization.

REDUCEBCM. Step 2.) and 3.) rely on the function REDUCEBCM, shown in Alg. 4. REDUCEBCM removes every drawbell in an BCM instance $\mathcal{I}$ not in a family D'_i. The resulting reduced BCM instance is denoted $\mathcal{I}'_i$.

First, REDUCEBCM computes the set of drawbells $\bar{D}'_i(q_i)$ accessible from each pose $q_j \in Q$ in $\mathcal{I}'_i$ by removing every

drawbell that is not in D'_i from the set $\bar{D}(q_i)$ (Line 1-2).

Second, it selects a reference drawbell for $\mathcal{I}'_i$. The drawbell in D'_i with the highest draw rate is chosen as D'_i's reference drawbell. This drawbell is denoted ψ_i (Line 3). Since the reference drawbell d_1 has the highest draw rate of any drawbell in $\mathcal{I}$, d_1 is the reference drawbell in $\mathcal{I}'_1$.

Third, it re-expresses the draw ratios of the drawbells in D'_i relative to the new reference drawbell ψ_i. The draw ratio between a drawbell $d_j \in D$ and the original reference drawbell d_1 is $r_j := \lambda_j/\lambda_1$. Thus, the draw ratio between two arbitrary drawbells $d_j, d_k \in D$ is $\lambda_j/\lambda_k = \lambda_j/\lambda_1 \cdot \lambda_1/\lambda_k = r_j/r_k$. Let ρ_i be the draw ratio between D'_i's reference drawbell ψ_i and the original reference drawbell d_1 in $\mathcal{I}$ (Line 4). The draw ratio between a drawbell $d_j \in D'_i$ and ψ_i is thus r_j/ρ_i. The set of rescaled draw ratios R'_i in $\mathcal{I}'_i$ is thus $R'_i := \{r_j/\rho_i : d_j \in D'_i\}$ (Line 5).

GENREFSUBCYCLE. This function is shown in Alg. 3 (Lines 10-12). Recall that D'_1 is the family containing the original reference drawbell d_1. GENREFSUBCYCLE uses REDUCEBCM to obtain the reduced BCM instance $\mathcal{I}'_1$ (Line 11). It then uses SAMM to solve $\mathcal{I}'_1$ (Line 12).

GENSUBCYCLE. This function also generates a reduced BCM instance $\mathcal{I}'_i$ by applying REDUCEBCM to $\mathcal{I}$ (Line 14). However, it solves $\mathcal{I}'_1$ using a modified version of SAMM called Fixed Start SAMM (FS-SAMM) (Line 15).

FS-SAMM is constructed by replacing SAMM's call to FLIMM with a call to a variant of FLIMM, Fixed Start FLIMM (FS-FLIMM). FS-FLIMM considers only solutions to the F-BCM problem that start and end with its agents in the state $(\bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\sigma}})$. FS-FLIMM is obtained by augmenting FLIMM with the following constraints.

Constraint 12. There is an agent in pose $q_i \in Q$ with shovel state $\phi \in \Phi$ on timestep $t = 0$ iff there is an agent in that state in $(\bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\sigma}})$.

$$\forall\ q_i \in Q, \forall\ \phi \in \Phi,\ at(q_i, \phi, 0) \\ \Leftrightarrow \exists\ j \in [0..n-1]\ s.t.\ (\bar{\pi}_j, \bar{\sigma}_j) = (q_i, \phi).$$

COMPOSESUBCYCLES. This function constructs a transport plan online by concatenating subcycles from Ξ. Whenever a subcycle completes, the function selects the next subcycle to execute. Because all subcycles share the same start and end state, they can be seamlessly composed in any order.

COMPOSESUBCYCLES ensures that the draw rates of all drawbells conform to the draw ratios in R by adjusting how frequently each subcycle is executed. Let f_i be subcycle ξ_i's execution frequency. Let PICKUP(ψ_i) be the amount of ore that each execution of ξ_i removes from its reference drawbell ψ_i. The draw rate from ψ_i is thus $f_i \cdot$ PICKUP(ψ_i).

The required draw ratio between ψ_i and subcycle ξ_1's reference drawbell $\psi_1 = d_1$ is ρ_i. To produce this draw ratio, COMPOSESUBCYCLES must ensure that:

$$f_i \cdot \text{PICKUP}(\psi_i) = f_1 \cdot \text{PICKUP}(\psi_1) \cdot \rho_i$$

Thus, ξ_i must be executed η_i times for every execution of ξ_1 where:

$$\eta_i := \frac{f_i}{f_1} = \frac{\text{PICKUP}(\psi_1) \cdot \rho_i}{\text{PICKUP}(\psi_i)}$$

Algorithm 5 COMPOSESUBCYCLES$(\Xi, \boldsymbol{\eta})$

```
1: for ξ_i ∈ Ξ do
2:     f_i ← 0
3: while true do
4:     for ξ_i ∈ Ξ do
5:         if (f_i + 1)/f_1 ≤ η_i then
6:             RUN(ξ_i)
7:             f_i ← f_i + 1
```

We term η_i subcycle ξ_i's execution ratio. Let the set of all execution ratios be $\boldsymbol{\eta} := \{\eta_i : \xi_i \in \Xi\}$.

In practice, achieving the exact execution ratio η_i that each subcycle $\{\xi_i\}_{i \neq 1}$ requires is difficult when these ratios are not simple. Consequently, COMPOSESUBCYCLES is designed to achieve asymptotic feasibility. As the plan length increases, COMPOSESUBCYCLES's draw ratio constraint violations converge to zero at the rate $O(1/n)$, where n is the number of subcycles that have been executed. COMPOSESUBCYCLES is shown in Alg. 5.

COMPOSESUBCYCLES continously loops over the subcycles in Ξ. When it reaches each subcycle $\xi_i \in \Xi$, it runs ξ_i unless doing so would cause ξ_i's realized execution ratio to exceed its target execution ratio (Lines 4-7).

A. Analysis

Let L be the number of times that COMPOSESUBCYCLES's inner loop (Lines 4-7) has been run.

Theorem 3: SAMMS can generate an asymptotically feasible solution for any BCM instance $\mathcal{I}$. After COMPOSESUBCYCLES loops L times, each drawbell's draw rate within $O(1/L)$ of the target draw rate.

Proof. The realized draw ratio between a drawbell $d_j \in D'_i$ and the reference drawbell d_1 is directly proportional to the realized execution ratio between D'_i and D'_1. Thus, if the realized execution ratio of D'_i converges to η_i at rate $O(1/L)$, the realized draw ratio of drawbell d_j converges to r_j at the same asymptotic rate. After L loops, each subcycle $\xi_k \in \Xi$ has been executed exactly $\lfloor \eta_k \cdot L \rfloor$ times. Since $\eta_1 = 1$, the realized execution ratio of ξ_i is $\lfloor \eta_i \cdot L \rfloor / L$. The error $\eta_i - \lfloor \eta_i \cdot L \rfloor / L$ converges to 0 at rate $O(1/L)$. □

VII. EXPERIMENTAL RESULTS

We evaluate SAMM and SAMMS from two points of view: (a) the computational time, and (b) solution quality, expressed as throughput, i.e. the total amount of ore deposited.

Setup. SAMM and SAMMS are evaluated on five block cave mine scenarios. These scenarios were chosen to have a good mix of drawbells and crushers, maximizing the potential effects of optimization. SAMM and SAMMS were given a time limit of 60 seconds in each experiment. This time limit is realistic since embeddings are computed offline. Both SAMM and SAMMs were set to terminate after two runs of FLIMM without any improvement. Pickup and deposit times were both set to one timestep. The scenarios [20] are produced by a benchmark generator whose parameters are calibrated from layout statistics reported in [19]. We implement[??] Alg. 2 and 3 in Python 3.11 [21]. We represent the layout graph with the NetworkX [22] library, and solve

TABLE I

SAMM VS. SAMMS THROUGHPUT AND RUNTIME ACROSS SCENARIOS.

Scenario	Drawbells	Crushers	SAMM Throughput	SAMM Runtime (s)	SAMMS Throughput	SAMMS Runtime (s)
1	8	4	0.89	11.0	0.56	0.96
2	14	7	1.56	49.1	0.92	1.69
3	16	8	1.78	57.8	1.10	2.92
4	18	4	N/A	60	0.40	11.28
5	24	4	N/A	60	0.34	55.8
6	36	6	N/A	60	0.59	13.9
7	48	8	N/A	60	0.73	34.0

FLIMM as an ILP using Gurobi [23]. Our implementation and the code to run our evaluations is publicly available at [24]. Each evaluation was performed on a Intel Core Ultra 9 275HX $\times$ 24 CPU, a NVIDIA GeForce RTX 5080 Laptop GPU and 32 GiB of RAM running Ubuntu 24.04.3 LTS.

A. Results.

The goal of our evaluation is to determine whether SAMM and SAMMS can scale to scenarios representative of industrial use cases. We ran SAMM and SAMMS 5 times on 7 scenarios of increasing complexity. Table VI-A shows each solvers: (i) mean throughput, measured across the runs where it terminated successfully, and (ii) mean runtime, where a failed run had a mean runtime of 60sec. SAMMS solved 5/5 runs on each scenario except for scenarios 5 and 7, where it solved 4/5 runs. It consistently achieved runtimes an order of magnitude faster than SAMM by decomposing the BCM problem into smaller, independent subproblems. SAMM, however, was not able to scale to the two larger scenarios. Notably, throughput is always relatively low, even in scenarios with a large number of crushers. In the tight mine geometries used in our evaluations, maneuvering agents into and out of a crusher is time-consuming. Consequently, whenever an agent deposits ore into a crusher, the crusher is effectively occupied or an extended period of time, even though a deposit action only takes a single timestep.

VIII. CONCLUSIONS

In this paper, we formalize the *Block Cave Mining (BCM)* problem, which focuses on coordinating autonomous vehicles in underground mines. In order to solve the problem, we propose SAMM, an eventually optimal anytime solver, which jointly solves task assignment, scheduling, and path planning via MILP. To address scalability limitations inherent in MILP approaches, we also present SAMMS, which decomposes the planning into shorter subcycles, trading optimality for computational efficiency. Experimental results on real-world mine scenarios demonstrate that SAMMS can be applied to industrial-scale BCM instances.

REFERENCES

[1] L. M., S. Schafrik, P. Kolapo, Z. Agioutantis, and J. Sottile, "Equipment and Operations Automation in Mining: A Review," *Machines*, vol. 10, no. 12, p. 713, 2024.

[2] B. Gold, "Rio Tinto's Autonomous Haulage Achieves 1 Billion Tons," *Eng. and Mining J.*, vol. 219, pp. 4—-5, 2018.

[3] L. Orellana, R. Castro, A. Hekmat, and E. Arancabia, "Productivity of a Continuous Mining System for Block Caving," *Rock Mechanics and Rock Eng.*, vol. 50, pp. 657—-663, 2016.

[4] A. R. Guest, G. J. van Hout, A. V. Johannides, and L. Scheepers, "An application of linear programming for block cave draw control," in *Int. Mass Mining Conference and Exhibition.*, 2006.

[5] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Barták, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *ArXiv*, vol. abs/1906.08291, 2019.

[6] A. Agaskar, S. Siva, W. Pickering, K. O'Brien, C. Kekeh, A. Li, B. G. Sarker, A. Chua, M. Nemade, C. Thattai, J. Di, I. Iyengar, R. Dharoor, D. Kirouani, J. Erskine, T. Hegazy, S. Niekum, U. A. Khan, F. Pecora, and J. W. Durham, "Deepfleet: Multi-agent foundation models for mobile robots," *ArXiv*, 2025.

[7] J. Li, T. A. Hoang, E. Lin, H. L. Vu, and S. Koenig, "Intersection coordination with priority-based search for autonomous vehicles," in *AAAI Conf. Artif. Intell.*, 2023.

[8] G. Sartoretti, Y. Wu, W. Paivine, T. Kumar, S. Koenig, and H. Choset, "Distributed Reinforcement Learning for Multi-Robot Decentralized Collective Construction," *The Int. Symp. on Distributed Auton. Robot. Syst.*, pp. 35–49, 2018.

[9] M. A. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, pp. 477–521, 1986.

[10] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-Based Search for Optimal Multi-Agent Path Finding," *Artif. Intell.*, vol. 219, p. 40–66, 2012.

[11] S.-H. Chan, R. Stern, A. Felner, and S. Koenig, "Greedy priority-based search for suboptimal multi-agent path finding," in *Symp. on Combinatorial Search*, 2023.

[12] J. Yu and S. LaValle, "Multi-agent Path Planning and Network Flow," *Algorithmic Foundations of Robot. X*, pp. 157–173, 2013.

[13] E. Lam, P. L. Bodic, D. D. Harabor, and P. J. Stuckey, "Branch-and-cut-and-price for multi-agent pathfinding," *Comput. Oper. Res.*, vol. 144, p. 105809, 2019.

[14] V. Nguyen, P. Obermeier, T. C. Son, T. Schaub, and W. Yeoh, "Generalized Target Assignment and Path Finding Using Answer Set Programming." *Int. Joint Conf. Artif. Intell.*, pp. 1216–1223, 2017.

[15] F. Khodayari and Y. Pourrahimian, "Mathematical programming applications in block-caving scheduling: a review of models and algorithms," *Int. J. of Mining and Mineral Eng.*, 2015.

[16] M. Tabesh and H. Askari-Nasab, "Two-stage clustering algorithm for block aggregation in open pit mines," *Mining Technol.*, vol. 120, pp. 158 – 169, 2011.

[17] D. O'Sullivan and A. M. Newman, "Optimization-based heuristics for underground mine scheduling," *Eur. J. Oper. Res.*, vol. 241, pp. 248–259, 2015.

[18] S. Donaldson, W. Harney, and T. Cox, "Operational and real-time lhd dispatch at rio tinto's argyle diamond mine," *MassMin 2020: Proc. of the Eighth Int. Conf. & Exhibition on Mass Mining*, 2020.

[19] R. Gómez, K. Saéz, N. Pino, E. Labbe, and E. Marambio, "Analysis of extraction level layouts for block caving," *MassMin 2020: Proc. of the Eighth Int. Conf. & Exhibition on Mass Mining*, 2020.

[20] P. Forte, *Atlantis Collection*, 2025, https://github.com/PaoloForte95/atlantis_collection.

[21] Python Software Foundation, "Python 3.11.10 Documentation," 2024, accessed: 2024-09-15.

[22] A. A. Hagberg, D. A. Schult, and P. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," *The 7th Python in Sci. Conf.*, pp. 11–15, 2008.

[23] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024, accessed: 2024-09-15.

[24] C. Leet, "SAMM-block-cave-mining," https://github.com/chrisleet/SAMM-block-cave-mining, 2026, gitHub repository, accessed March 6, 2026.