Proceedings of the Eighteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment
(AIIDE 2022)

Minimizing Coordination in Multi-Agent Path Finding with Dynamic Execution

Aidan Wagner[*], **Rishi Veerapaneni**[*], **Maxim Likhachev**

Robotics Institute, Carnegie Mellon University
{ajwagner, rveerapa, mlikhach}@andrew.cmu.edu

Abstract

Multi-agent Path Finding (MAPF) is an important problem in large games with many dynamic agents that need to follow space-time trajectories without inter-agent collisions. Modern MAPF solvers plan assuming that agents directly follow the space-time trajectories at known constant speeds without delays or speedups, resulting in rigid plans which need to be replanned if there are changes during execution. Instead we would like agents to be able to follow their computed paths with dynamic velocities while requiring minimal coordination with others to prevent collisions and deadlocks. One way to address this problem is to first produce collision free space-time paths and then compute a coordination controller that prevents collisions and deadlock during dynamic execution. This two step process prevents fully minimizing coordination as the initially planned space-time paths do not reason about coordination and can be arbitrarily bad. We introduce a novel paradigm and show how planning in *space-coordination level*, rather than space-time, allows us to simultaneously plan paths and a coordination controller. Our method, Space-Level Conflict-Based Search (SL-CBS), builds on the Conflict-Based Search framework and allows us to reason explicitly about coordination, producing paths as well as a coordination controller with bounded suboptimal minimal coordination. We show experimentally that this results in a 20-50% reduction in coordination compared to the closest state of the art solver.

Introduction

Multi-agent coordination is required for many real-world tasks as well as many video games. Multi-Agent Path Finding (MAPF) tries to compute collision-free paths for groups of agents while minimizing a certain objective, which is usually the sum of path costs.

In large strategy games with hundreds of diverse agents, like Age of Empires or Rise of Nations, it is important for agents to traverse the landscape in a realistic way while avoiding obstacles and other moving agents. Since each agent has their own movement abilities, we would like each agent to be able to traverse their paths with *dynamically execution*, speeding up and slowing down arbitrarily as desired during execution. In known deterministic environments, we would like to change agent velocities during runtime without replanning agents as replanning requires extra computation, and can lead to unnatural movement and deadlock in the multi-agent setting. Instead, planning an initial robust plan and a coordination protocol could let agents follow their paths with dynamic velocities, eliminating any additional replanning and leading to natural looking coordination at the expense of computing the initial robust plan and the coordination protocol to prevent collisions and deadlock.

Motivating example: Imagine a game with a large number of diverse agents that can move at different speeds, e.g. infantry, knights, elephant riders, catapults. Computing collision free paths in space-time without velocity reasoning would require each agent to follow a uniform constant unit velocity, preventing dynamic speed differences between different agents. On the other hand, incorporating velocity into the state would increase our state space and planning time. Finally, neither of these two solutions output space-time trajectories that would allow dynamic changes in velocity during execution (like horses getting tired and needing to slow down) and would require these to be explicitly planned initially or additional replanning afterwards. We would like to just plan 1) A set of paths for agents to follow and 2) A coordination controller that sends coordination commands during execution to prevent collisions. Therefore during execution, agents can follow their paths as slow or fast dynamically as wanted, and only need to stop when requested by the coordination controller. Crucially both these are computed only once initially, removing the need for replanning under dynamic execution.

Our goal in this work is specifically to find a set of collision free paths and a coordination controller that minimizes the amount of coordination required for agent to dynamically follow the paths. To this end, we reformulate the classic MAPF problem to explicitly reason about coordination by planning in respect *space-coordination level* instead of space time. Our key insight is that planning in space-level allows us to *simultaneously* plan paths and a coordination controller. Our contributions are

1. Defining the minimum coordination problem under dynamic execution.
2. Demonstrating how planning in space-level rather than space-time allows reasoning about both paths and coordination simultaneously.

[*]Both authors contributed equally.

3. Building on top of the Conflict-Based Search framework and creating Space-Level Conflict-Based Search (SL-CBS) which is a bounded suboptimal solver that can return bounded minimum coordination paths.

Related Work

There are several MAPF works that compute collision free paths. Conflict-Based Search (CBS) is a popular complete and optimal MAPF solver that uses a high-level constraint tree (CT) to resolve conflicts and low level planner than plans individual agents in respect to the constraints. There have been many follow up works that speed up CBS by reducing the CT size by employing suboptimality, explicitly pruning branches, selectively expanding branches, adding sets of constraints, detecting symmetries, and improving high-level heuristics (Barer et al. 2014; Li, Ruml, and Koenig 2021; Boyarski et al. 2015, 2021; Li et al. 2019, 2020, 2021).

Reasoning about computing paths and coordination that can be followed at dynamic unknown velocities has been substantially less studied. Several works have worked on "K-Robust" MAPF which compute paths that can be executed with k delays without collisions (Atzmon et al. 2018; Chen et al. 2021). These works do not require additional coordination during execution, but are limited to instances where each agent is delayed at most k times. We instead want a method that can handle arbitrary delays, speed-ups, and general heterogeneous execution speeds. Various other works tackle non-constant velocities (van den Berg, Lin, and Manocha 2008; Kou et al. 2019) or reason about inter-agent communications (Boardman, Harden, and Martínez 2021; Wang, Sahin, and Bhattacharya 2022) but none reason about coordination and dynamic execution without the need for replanning.

Temporal Plan Graph

The closest relevant work is Hönig et al. (2016) which introduces a post-processing technique MAPF-POST that allows agents to follow the initial plan with their own dynamic executions, i.e. with their own velocity constraints, as long as they are never making negative progress (moving in the opposite direction of the initial planned path). Specifically, they create a Temporal Plan Graph (TPG) that maintains visitation orders of agents on states of contention to prevent deadlock. During execution, agents just need to query the next state of contention and check if traversing it will violate the visitation order or not. If traversing the next state would violate the visitation, the agent is required to wait until it does not. Reducing the number of these states of contention therefore reduces the amount of coordination overhead as well as decreases the number of instances where agents may need to wait. Using the TPG and the corresponding TPG protocol allows agents to dynamically traverse their paths faster or slower without needing to replan.

Planning a set of collision free paths using any MAPF method, computing the TPG, and then following the paths with arbitrary velocities during execution using the TPG for coordination therefore describes the type of solution we want. However, this procedure does not attempt to reduce the amount of coordination needed.

Coordination Under Dynamic Execution

The standard MAPF problem takes in a graph G with vertices V and edges E, and N agents $A_i, i \in \{1...N\}$ with unique start and goal locations, and returns collision free space-time trajectories $\pi_i(t)$ for each agent. Typical MAPF methods work on a graph defined by a 2D gridworld with vertices corresponding to non-obstacle cells, unary cost edges between adjacent vertices (4-connected), point-mass robots occupying a single vertex, and aim to minimize the maximum or sum of path travel times. Time is discretized into unit timesteps with agents taking exactly one timestep to traverse edges, requiring agents during execution to follow the paths with a constant known velocity.

We aim to remove the constraint of agents following their paths at velocities known during planning. Instead, we allow *dynamic execution* of agents running at arbitrary non-negative speeds during execution, and use a *coordination controller* to prevent collisions and deadlock. Our goal is to compute the paths as well as a coordination controller that minimizes the amount of coordination required. We change the standard MAPF problem to explicitly reason about coordination of paths alongside the actual paths themselves. This initially dramatically complicates our planning problem as we must plan paths as well as the coordination protocol. We start by first formally defining our problem set-up by defining dynamic execution, coordination, and our objective of minimizing coordination.

Problem Set-up

Definition 1 (Dynamic execution)**.** *For a given set of N agents, $A_i, i \in \{1...N\}$, and agent paths π_i, dynamic execution allows agents to traverse their paths with arbitrary velocity ≥ 0 across the path to their goal unless specified to stop and wait by a coordination controller.*

Let each agent A_i have a path π_i consisting of adjacent states, with $\pi_i(0) = Start_i$ and $\pi_i(end) = Goal_i$. Dynamic execution allows agents to dynamically/arbitrarily speed up and slow down as needed, so the location of an agent at time t, $\pi_i(t)$, can only be determined during execution. Agents are not allowed to backtrack as this is unwanted and suboptimal behaviour. This fits our video game use case of having different diverse agents travelling at different dynamic velocities during execution.

Definition 2 (Coordination controller)**.** *The coordination controller specifies stop and continue coordination commands to agents based on the locations of other agents.*

The coordination controller (CC) aims to prevent collisions and deadlock during dynamic execution. Notice how the controller cannot specify concrete timesteps when agents should stop and continue, as with dynamic execution, agents locations are not tied to specific timesteps. Also note that the controller cannot modify any paths but only acts as a sort of smart traffic light. Therefore the controller can only control *where* agents should stop and continue based on the locations of other agents during dynamic execution.

Definition 3 (Coordination commands). *A stop command for agent A_i along their path π_i is defined by a stop location $s \subset \pi_i$ and the locations of other agents $S_j \subseteq \pi_j$. Formally, a stop command is issued during dynamic execution at timestep t for A_i when $\pi_i(t) = s$ and $\exists j \neq i, \pi_j(t) \in S_j$. Every stop command has an corresponding continue command for agent A_i to continue past s when $\forall j \neq i, \pi_j(t) \notin S_j$.*

We abuse notation and let $S_j \subset \pi_j$ denote a subset of locations along the sequence π_i. Conceptually, the use of the controller can prevent collisions in the same way as a smart traffic light; stop an agent (A_i) before an intersection (s) until all opposing agents (A_j) pass it, and then allow the original agent to continue. This example demonstrates why the stop/continue command for A_i depends on the locations of other agents A_j; we need to know if they have crossed the intersection yet or not. This coordination prevents replanning under dynamic execution as desired and instead reduces the computational burden to figuring out the controller during planning and having the controller send coordination commands during execution. However note that this definition requires agents to be able to stop arbitrarily fast. Additionally note that MAPF-POST (Hönig et al. 2016) defines a CC via their Temporal Plan Graph.

Definition 4 (Coordination Objective). *Our goal is to minimize the total coordination as measured by the number of stop commands that the coordination controller needs to compute.*

We cannot define the coordination as the sum of wait times of agents as that can only be determined during dynamic execution. We therefore aim to reduce the number of instances that agents need to wait and coordinate with other agents.

Our problem is to compute a set of paths as well as a coordination controller that minimizes our total coordination. Computing an arbitrary set of collision free space-time paths and then post-processing them to generate a coordination controller does not allow minimizing coordination as the space-time paths can be arbitrarily bad in respect to post-processing for coordination under dynamic execution.

Method

Our main idea is that paths and the coordination controller can be planned simultaneously. As defined, the CC specifies stop commands for agents at specific locations based on the location of other agents. Between stop commands, all agents must be in spatially independent path segments, otherwise under dynamic execution there could be a collision. Our key innovation is that we can represent a plan and a coordination controller in a single "space-coordination level" representation.

Space-Level Example

Imagine the blue agent (B) and red agent (R) in Figure 1 that need to plan paths from their respective start S to goal G locations. A path without any coordination would be fully spatially independent, but this is not possible. Instead B's path must intersect with agent R's path at multiple states, requiring the coordination controller to send stop commands during dynamic execution. At minimum, the CC must stop R from blocking (4,3) before B moves out of the way, and likewise must stop B from blocking (1,3) before R traverses it. Existing methods first compute space-time paths and then independently post-process them to determine a CC.

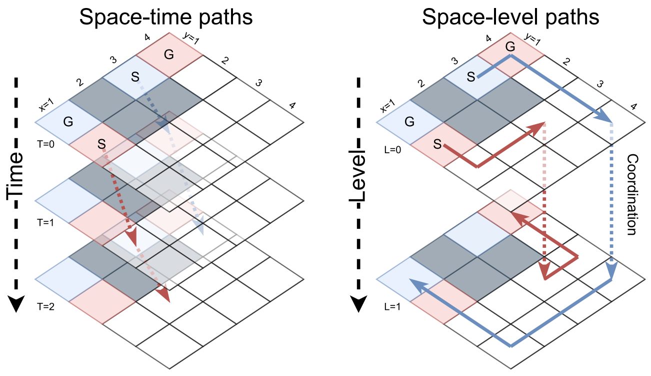

Figure 1: We plan space-level paths rather than space time paths. Space-time paths can be viewed as planning trajectories between copies of the map, with each action necessarily leading to a deeper layer. On the other hand, space-level paths allow consecutive actions in the same level as well as an action moving to a deeper level. The transitions between levels define explicit coordination commands.

We show that space-level paths contain information for the individual paths as well as the CC using our example in Figure 1. Imagine we are given the collision free space-level paths for R and B. Each path consists of spatially independent path segments per level, which connect across layers to form a path from start to goal. Coordination is required at these transitions between layers, otherwise for example R could collide at (4,3) if it does not wait. Each level transition therefore explicitly defines a CC stop command, with continue commands given when all agents reach the end of their path segment on the level (e.g. when R reaches (3,3) and B reaches (4,4)).

Following Definition 3, our stop condition for R is $s = (3,3)$, $S_B = \{(3,1),(4,1),(4,2),(4,3)\}$. I.e. if R reaches $(3,3)$ while B is in a location in S_B, R must stop to prevent a collision with B. Similarly, our continue command for R is sent by the coordination controller when B leaves S_B, allowing R to continue on its path after B clears the intersection. These two commands together with the stop and continue command for B at $(4,4)$ define the coordination controller.

Coordination Controllers From Space-level Paths

Space-level paths in our space-level representation contain all the necessary information for the spatial paths as well as the corresponding CC commands. Mathematically, for a space-level path consisting of path segments π_j^l at level l, the aggregate path is $\pi_j = [\pi_j^0, \pi_j^1, ...]$. Coordination commands for A_i occur at each transition between levels, at level l with $s = \pi_i^l(end)$ and $S_j = \pi_j^l \setminus \pi_j^l(end), \forall j \neq i$. Each

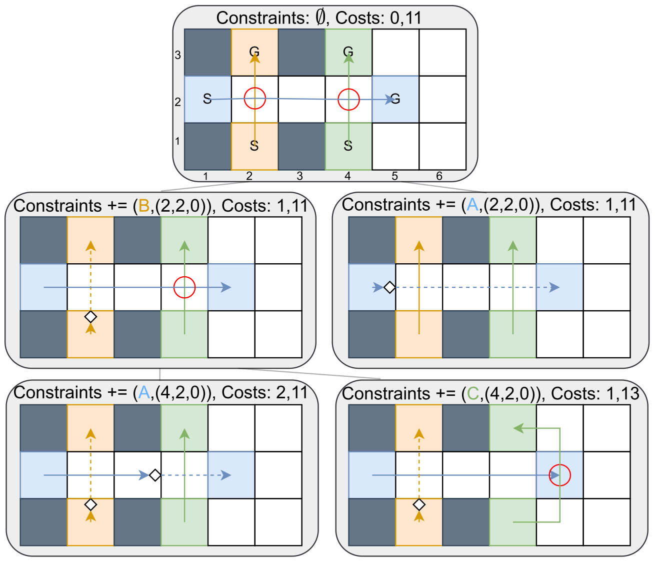
Constraints: ∅, Costs: 0,11
Constraints += (B,(2,2,0)), Costs: 1,11
Constraints += (A,(2,2,0)), Costs: 1,11
Constraints += (A,(4,2,0)), Costs: 2,11
Constraints += (C,(4,2,0)), Costs: 1,13
S
G

Figure 2: We show an example of Space-Level CBS running on a scenario with 3 agents (A is blue, B is orange, C is green). Paths segments at level=0 are in solid and level=1 are dashed with the transition between levels marked by the small diamond. Conflicts are circled in red, costs are (coordination cost, path cost) tuples. We initially start with no constraints and have two conflicts. Arbitrarily resolving the conflict at (2,2,0) with constraints results in either A or B needing to go to a deeper level, as (2,2,0) cannot be bypassed other than coordinating. Since both branches have identical costs, we arbitrarily choose to branch the left node and resolve the conflict at (4,2,0). Agent A avoid this conflict by going a level deeper, while B can spatially avoid it, resulting in another conflict. At this point we expand the (A,(2,2,0)) high level node which has the least cost and results in a valid solution, terminating search.

level denotes collision free path segments for each agent to follow at arbitrary speed until they reach the end of the path segment, which then denotes a stop command sent by the CC. Once all agents reach the end of their path segments at that level, the CC sends continue commands allowing agents to continue on the rest of their path until the next stop command. Therefore the transition between levels denotes an explicit coordination stop and continue command, allowing us to reason with coordination along paths.

Space-Level Conflict-Based Search

Conflict Based Search is a popular complete and optimal space-time MAPF planner. The CBS framework employs a high-level constraint tree that searches over conflicts in a best-first manner, and a low level A* search that searches individual paths while respecting constraints. Our method, Space-Level CBS, utilizes the CBS framework to plan conflict free space-level paths instead of space-time trajectories. Algorithm 1 shows the standard high level CBS psuedo-code as described in Barer et al. (2014) with space-level differences underlined. Figure 2 walks through a small example of Space-Level CBS.

Low-level planner Instead of searching for space-time paths, our low-level planner searches for space-level paths. The key difference is that unlike space-time search where a successor of state s and time t, (s,t), leads to $(s',t+1)$, we explicitly reason about levels and can choose to remain in the same level (s',l) or increase our level $(s,l+1)$. We do not allow actions decreasing levels. Therefore within a level l, all successors are either 1) adjacent states, resulting in a spatial path segment within the level, or 2) a successor to the level $l+1$, resulting in the end of the path segment at level l. Note that regular space-time planning is a subset of our formulation where each level has only one other adjacent successor.

Sum of Individual Costs (SIC) Following our definitions, the coordination controller is defined by the transitions between levels with the coordination cost equal to the sum of levels across agents. Agents resting at their goal accrue no additional cost. Therefore we compute and minimize the sum of levels rather than path cost. However in some instances this may dramatically increase path lengths, so we more generally consider minimizing a weighted sum $(1-w)*P+w*L$ of the sum of path lengths P and sum of levels L. $w=1$ corresponds to purely minimizing coordination. Manipulating w allows the user to directly trade-off reducing coordination at the expense of path length. The low-level planner finds a space-level path for a single agent that minimizes this objective given the other agents' path and CC.

Space-level conflicts and constraints Space-level conflicts and constraints work identically to space-time conflicts and constraints. If two agents A_i, A_j conflict at the same space-level location (s,l), then a constraint is placed on each agent at (s,l) and the low level planner is called to replan while satisfying the new constraints. Conceptually, when satisfying the constraint, A_i can 1) avoid coordinating with A_j by finding a spatial path around s at the same level l, or 2) coordinate with A_j by going to a deeper level and then traversing over s. Note that since our low-level planner action space does not allow swapping (i.e. agent A_i goes $(s,l)\rightarrow(s',l+1)$ while A_j goes $(s',l)\rightarrow(s,l+1)$) as changing levels requires staying at the same location $(s'=s)$, we do not need edge conflicts.

Enhanced CBS Enhance Conflict Based Search (Barer et al. 2014) uses a w_{so} bounded suboptimal focal search on the low-level planner and high-level planner based on conflicts to speed up search. $w_{so}\geq 1$ is a hyper-parameter that controls the suboptimality of the search solution. We incorporate this out of the box in SL-CBS using space-level conflicts. With $w=1$, SL-CBS will compute paths and a CC that are w_{so} suboptimal in respect to total coordination cost.

Proving Optimal Minimum Coordination

We prove that with $w=1, w_{so}=1$, our SL-CBS method of using CBS for computing space-level plans and the CC based on the levels minimizes the total coordination. Our proof follows CBS's main optimality logic with changes due to our space-level planning and coordination controller. The proof for $w_{so}>1$ bounded suboptimality follows sim-

Algorithm 1: SL-CBS high-level

```
Input: MAPF instance
 1: $R.constraints = 0$
 2: $R.solution$ = find individual paths using low-level()
 3: $R.cost = SIC(R.solution)$
 4: Insert R to OPEN
 5: while OPEN $\neq \emptyset$ do
 6:     $P \leftarrow$ OPEN.pop()
 7:     if $P$ has no conflicts then
 8:         return $P.solution$
 9:     $C \leftarrow$ conflict($A_i, A_j, (s, \underline{l})$) in P
10:     for agent $A_i$ in C do
11:         A $\leftarrow$ new node
12:         A.constraints $\leftarrow$ P.constraints + ($A_i, (s, \underline{l})$)
13:         A.solution $\leftarrow$ P.solution
14:         Update $A.solution$ invoking low-level($A_i$)
15:         $A.cost = SIC(A.solution)$
16:         Insert A to OPEN
17: return No solution
```

ilarly with ECBS's suboptimality logic and is omitted for brevity. In practice running with $w_{so} = 1$ timesout but running slightly larger w_{so} produces improved results.

Lemma 1. *Our space-level representation contains all minimal length paths and a sufficient subset of CC's whith contains an optimal solution.*

Proof. Consider a set of paths π_i^* and CC* that result in a optimal minimum cost solution. Observe that π_i^* will not repeat states between coordination commands as otherwise we can obtain a cheaper path by shortcutting the repeated sections. Our space-level representation does not change the spatial path planning space of the problem; SL-CBS will still consider all minimal length paths including π_i^*.

However, our space-level representation does not allow all possible CC's as it only considers coordination commands based on *all* agents, as opposed to CC* which can send CC commands on subsets of agents.

Concretely, let each CC* command depend on a subset of agents $Y \subset \{A_0, ...A_N\}$, which means the command will be sent for arbitrary locations of agents in $Y' = \{A_0, ..., A_N\} \setminus Y$. We can construct a CC*′ which adds arbitrary location constraints on agents in Y', resulting on CC commands dependent on all agents' locations and has the same optimal cost as CC*. Our space-level representation contains CC*′ but not CC*. Thus our space-level representation does not contain all possible CC's but contains a sufficiently large set that covers a CC with optimal cost. □

Lemma 2. *At all times, there exists a node in the constraint tree that leads to the optimal solution.*

Proof. We use induction on the nodes in the constraint tree.

Base case: SL-CBS's root CT node permits any possible solution as there are no constraints.

Inductive step: Assume we have a node in the CT that leads to the optimal solution. If we have no conflicts then this node is the optimal solution. If we have a conflict at some (s, l, A_i, A_j), then the optimal solution must have at least one agent avoid the location. Each child node does exactly this, therefore at least one child node will lead to the optimal solution. □

Lemma 3. *The cost of children CT nodes are $\geq$ than the cost of the parent.*

Proof. Let us expand a CT node Q by adding the constraint A_i to avoid a conflict (s, l) in π_i and replanning A_i to get a child CT node Q' with new low-level path π_i'. Suppose π_i' has less cost than π_i. Then π_i' is also a valid path for Q as Q no additional constraints on A_i than Q'. This is a contradiction as π_i was generated via an optimal low level search for Q and thus must have a lower cost than π_i'. Thus π_i' must result in an overall cost for $Q' \geq Q$. □

Theorem 1. *Space-level CBS returns the optimal solution.*

Proof. By Lemma 1, space-level CBS is searching over a sufficient space that contains a minimum coordination solution. By Lemma 2, a node that leads to the optimal solution is always in the CT. By Lemma 3, the cost of nodes are non-decreasing. Therefore searching the CT in order of lowest costs as describe in Algorithm 1, we are guaranteed that the first valid node we expand will result in a set of paths and CC that has the optimal minimum cost. With $w = 1$, the cost of each CT node is exactly equal to the coordination cost, and we return a minimal coordination cost solution. □

Experimental Results

Our main idea is that we can simultaneously optimize coordination and path lengths. We therefore compare against a method that first computes paths and then post-processes them afterwards rather than jointly reasoning about them. Since our SL-CBS method uses ECBS bounded suboptimal focal searches, we compare against computing paths via ECBS and then post processing using MAPF-POST (Hönig et al. 2016). MAPF-POST (Hönig et al. 2016) introduces a CC based on a temporal plan graph post-processing step on a given space-time plan. However, MAPF-POST only post processes a given path to construct some CC for dynamic execution rather than a low-cost CC. We therefore post process MAPF-POST to produce a lower coordination cost CC for fair comparison.

Modified MAPF-POST: The raw MAPF-POST CC does not minimize the number of spatial coordinates where coordination is required and instead specifies coordination at all states with spatial overlap. In Figure 1, MAPF-POST's CC would have coordination commands for R at $(4, 1), (4, 2), (4, 3)$ and for B at $(1, 3), (1, 2)$ rather than our two required commands. We post process MAPF-POST's CC by computing spatially independent path segments to get a more compact representation that reduces total coordination for more fair comparison. Specifically, given a set of paths and the corresponding MAPF-POST CC, we run all agents as far as possible until the the MAPF-POST CC sends a stop command to all of them. This defines a set of spatially independent path segments and levels, and produces

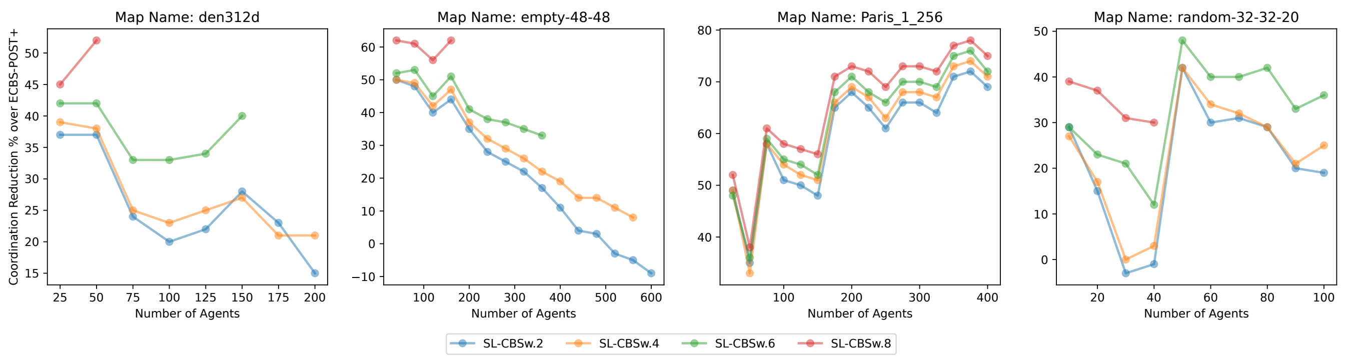

Figure 3: We see how SL-CBS with $w = 0.2, 0.4, 0.6, 0.8$ is able to provide consistent improvement over ECBS-POST+ across all maps, with larger w values improving coordination reduction. This confirms that jointly reasoning about paths and coordination can lead to substantial decreases in coordination compared to our ECBS-POST+ baseline. SL-CBS produces large improvements in Paris_1_256 as it is a big map where SL-CBS can fully utilize its space to avoid extra coordination.

ht_chantry	100	200	300	400	100	200	300	400	100	200	300	400
Method	Average Coordination				Average Path Length				Runtime			
ECBS	88.4	90.1	93.3	96.1	88.4	90.1	93.3	96.1	1.38	9.72	26.9	83.5
ECBS-POST+	11.6	17.1	24.6	31.2	Same as ECBS				Same as ECBS			
SL-CBSw0.2	4.93	10.22	15.19	20.9	91.9	99.5	106	114	0.48	2.69	9.17	21.91
SL-CBSw0.4	4.66	9.31	13.5	**18.1**	91.6	98.9	105	112	0.52	3.01	10.6	25.4
SL-CBSw0.6	4.35	**8.49**	**12.1**	-	91.7	98.7	105	-	0.53	3.13	14.8	-
SL-CBSw0.8	**3.98**	-	-	-	92.1	-	-	-	0.8	-	-	-

Table 1: We compare our method SL-CBS against our baselines on a ht_chantry, across different number of agents (top row). We observe that as w increases, we reduce average coordination at the expense of longer paths compared to ECBS. Additionally, our runtime is smaller when $w = 0.2, 0.4$, but becomes significantly slower for larger w that timeout (dashed entries).

a CC with less coordination required. We denote post processing paths found by ECBS as ECBS-POST+. If ECBS produces the space-time paths corresponding to Figure 1, ECBS-POST+ would just have a coordination command for R at $(3, 3)$ and B at $(1, 4)$, which has the optimal coordination cost of 2.

Experiments: We compare SL-CBS with $w = 0.2$, 0.4, 0.6, 0.8 to ECBS and ECBS-POST+ across 8 different maps from Stern et al. (2019). All statistics and plots are the means across 5 seeds of each method with $w_{so} = 1.5$ and a timeout of 2 minutes. SL-CBS with $w = 1$ timed out on all instances and is omitted accordingly. ECBS's coordination cost is equal to the path length as the amount of coordinated waits is equal to the path length (as at each timestep, each agent needs to sync/wait for all other agents to reach their corresponding location at that timestep). ECBS-POST+ acts as our primary baseline as described earlier. Along with coordination, we include statistics on the path length and runtime to see how SL-CBS trades these off. We did not include the runtime for POST+ in ECBS-POST+ as we did not optimize its implementation.

Minimizing Coordination: We see in Figure 3 that SL-CBS has a large coordination reduction across many maps. We additionally observe that increasing w increases coordination reduction as expected. We also witness that for SL-CBS with $w = 0.4, w = 0.8$, this comes with increased computational burden, with those methods timing out on 3 of the 4 maps as the number of agents increase. Figure 3 additionally shows that the trends of coordination reduction can change between maps. Paris_1_256 is the largest map where additional agents cause more congestion in ECBS's paths and requires more coordination while SL-CBS is able to utilize the free space to plan around them. On the otherhand, empty-48-48 has less free space and with more agents SL-CBS approaches ECBS's performance. For some Real-Time Strategy context, the random-32-32-20 map with low numbers of agents is similar to Age of Empires scenarios which have a density of roughly 25 agents per 30x30 tiles; we see our methods can reduce coordination by 10-40% in these scenarios. Overall across all 8 maps, SL-CBS with $w = 0.4$ reduces coordination by an average of 39% with stddev of 17% compared to ECBS-POST+.

Path Cost and Runtime: We analyze the results on the ht_chantry map in Table 1 to understand how SL-CBS is able to reduce coordination. The appendix contains additional results showing how this holds across different maps. We observe that SL-CBS produces consistently longer paths than ECBS, showing how SL-CBS's joint reasoning allows

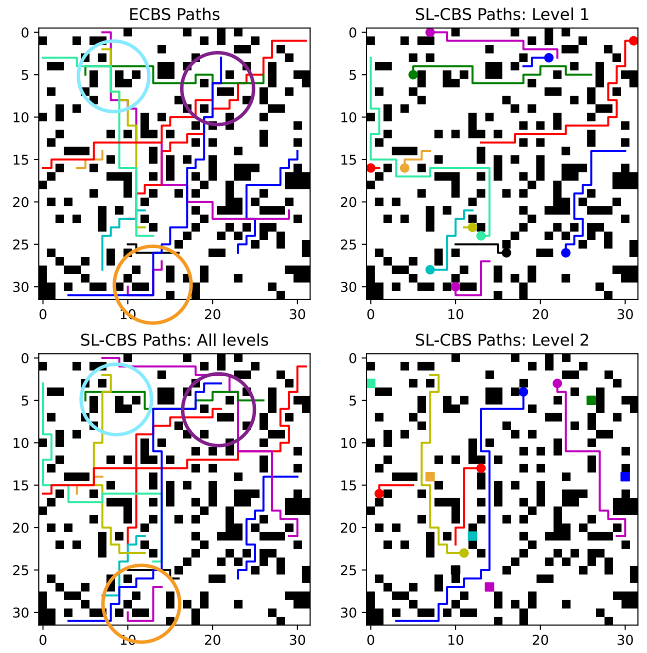

Figure 4: We compare running SL-CBS with $w = 0.85$ and ECBS on random-32-32-20 with $w_{so} = 1.2$. The left images compare ECBS's and SL-CBS's path. The right images show the spatially independent path segments found for level 1 and level 2 of the SL-CBS solution, denoting segments for dynamic execution along with coordination commands at the end. Agents resting at their goal are shown as squares. The circled areas highlight congested regions with overlapping paths in ECBS that have been spaced out in SL-CBS due to its joint coordination and path reasoning. The orange circle demonstrates a prime example where ECBS's blue path almost entirely overlaps the magenta path, requiring the two agents to coordinate along almost the entirety of the magenta agent's path. SL-CBS neatly sidesteps this by planning the blue agent around magenta.

it to reduce coordination by avoiding possible coordination points with longer paths. Figure 4 visualises this on a small scenario. We see that a small increase in average path cost per agent can lead to significant coordination reduction. In general across all 8 maps, SL-CBS with $w = 0.4$ increases average path cost by 10.6% with stddev of 9.1% while reducing coordination by an average of 39% with stddev of 17%.

SL-CBS's main bottleneck is its the runtime as w increases or as the scene gets more congested. We see that $w = 0.8$, SL-CBS times out at 100 agents, and that with SL-CBS with $w = 0.6$ take significantly longer than ECBS with 200 agents. This occurs as SL-CBS with high w's needs to reason more and more about coordination, which means it has to increasingly search over all coordination commands along with paths themselves. ECBS-POST+ with its separate CC generation does not suffer from this runtime issue at the expense of not being able to change paths to minimize coordination. However, we see that with lower values, $w = 0.2, 0.4$, SL-CBS is able to actually *decrease* planning time by 3-10x. This behaviour likely arises due to SL-CBS's ability to avoid conflicts by assigning agents different coordination levels (as opposed to ECBS which has all agents in space time at the same level), but was not explored deeper as reducing runtime is not the aim of our work.

Limitations, Future Work, and Conclusion

Our work demonstrates an initial methodology to simultaneously optimize for paths length and coordination, and has several limitations.

The most severe limitation is that our coordination commands are based on the location of *all* other agents as opposed to a subset. This means that agents will not move as independently as possible, likely resulting in some overally cautious behaviour as they may wait when on agents they do not need to. However, computing coordination on subsets of agents would require a different approach with likely harsh runtime implications as we would then need to additionally reason about which subsets of agents to coordinate/wait on (exponential such combinations per wait) rather than just if we should coordinate/wait (boolean per possible wait). Note we still minimize coordination as the number of coordination instances are the same regardless of if commands wait for a subset of or all agents.

Our method is also limited to reasoning about spatial paths and does not take other dynamic constraints like velocity or acceleration into account. These could be incorporated in SL-CBS by keeping the same high level structure and solely modifying the low-level planner to include these constraints which would likely slow down planning.

Future work could tackle these two main limitations to create more realistic trajectories and coordination controllers. An exciting different direction could also modify the definition of dynamic execution and coordination with a different set of assumptions. For example, dynamic execution with velocity bounds could allow for a coordination controller that modulates the velocity of agents rather than completely stopping them. Including priors on velocity could also enable methods to reason about and minimize wait time rather than the number of waits. In general, we hope this work spurs future investigations into jointly reasoning about coordination during dynamic execution of multi-agent systems, enabling dynamic movement of agents without needing to replan.

Our work defines the multi-agent dynamic execution scenario and the corresponding need for a coordination controller under dynamic execution. Under our definition of coordination, we show that reasoning in space-level rather than space-time allows us to jointly reason about paths and the coordination controller, and minimize the total coordination required. Our method SL-CBS based on the CBS/ECBS framework produces paths and a controller that provably minimizes coordination. Experimentally, we validate that SL-CBS simultaneously optimizes paths and coordination, and is able to minimize coordination by planning slightly longer paths that bypass congestion.

den312d	50	100	150	200	50	100	150	200	50	100	150	200
Method	Average Coordination				Average Path Length				Runtime			
ECBS	53.0	54.7	55.8	57.0	53.0	54.7	55.8	57.0	0.12	0.55	2.97	16.3
ECBS-POST+	6.29	10.7	16.9	20.5	Same as ECBS				Same as ECBS			
SL-CBSw0.2	3.99	8.75	12.2	17.4	56.5	62.5	65.4	70.9	0.06	0.32	0.81	2.01
SL-CBSw0.4	3.95	8.36	12.5	**16.2**	56.4	62.2	66.2	70.3	0.06	0.4	1.63	5.16
SL-CBSw0.6	3.7	**7.26**	**10.2**	-	56.2	61.6	64.3	-	0.08	0.65	23.15	-
SL-CBSw0.8	**3.07**	-	-	-	56.3	-	-	-	2.33	-	-	-

ht_chantry	100	200	300	400	100	200	300	400	100	200	300	400
Method	Average Coordination				Average Path Length				Runtime			
ECBS	88.4	90.1	93.3	96.1	88.4	90.1	93.3	96.1	1.38	9.72	26.9	83.5
ECBS-POST+	11.6	17.1	24.6	31.2	Same as ECBS				Same as ECBS			
SL-CBSw0.2	4.93	10.22	15.19	20.9	91.9	99.5	106	114	0.48	2.69	9.17	21.91
SL-CBSw0.4	4.66	9.31	13.5	**18.1**	91.6	98.9	105	112	0.52	3.01	10.6	25.4
SL-CBSw0.6	4.35	**8.49**	**12.1**	-	91.7	98.7	105	-	0.53	3.13	14.8	-
SL-CBSw0.8	**3.98**	-	-	-	92.1	-	-	-	0.8	-	-	-

room-32-32-4	25	50	75	100	25	50	75	100	25	50	75	100
Method	Average Coordination				Average Path Length				Runtime			
ECBS	27.6	29.4	30.7	34.6	27.6	29.4	30.7	34.6	0.01	0.45	2.86	18.1
ECBS-POST+	3.35	7.31	9.85	14.1	Same as ECBS				Same as ECBS			
SL-CBSw0.2	2.78	5.28	8.17	11.34	30.2	32.9	35.3	39.2	0.01	0.1	0.6	1.47
SL-CBSw0.4	2.74	5.14	7.64	**10.3**	30.1	32.9	34.6	37.9	0.01	0.08	0.45	2.87
SL-CBSw0.6	2.5	**4.71**	**6.37**	-	30.0	32.3	33.1	-	0.01	0.11	0.71	-
SL-CBSw0.8	**2.1**	-	-	-	29.9	-	-	-	0.02	-	-	-

Table 2: We compare our method SL-CBS against our baselines on a large (den312), medium (ht_chantry), and small (room-32-32-4) map across different number of agents (top row of each). We observe that as we increase w, we reduce average coordination at the expense of longer paths compared to ECBS. Additionally, our runtime is smaller when $w = 0.2, 0.4$, but becomes significantly slower for larger w with timeouts denoted by dashed entries.

Appendix: Quick Recap

We provide a quick recap for the skimming reader.

Recommended Background Reading

Readers new to Conflict-Based Search are recommended to read CBS or ECBS (Sharon et al. 2015; Barer et al. 2014). Readers unfamiliar with incorporating coordination for dynamically following MAPF paths should read (Hönig et al. 2016).

Intended Takeaways

Motivation: Imagine a game with a large number of diverse agents that can move at different speeds, e.g. infantry, knights, elephant riders, catapults. Computing collision free paths in space-time without velocity reasoning would require each agent to follow a uniform constant unit velocity, preventing dynamic speed differences between different agents. On the other hand, incorporating velocity into the state would increase our state space and planning time. Finally, neither of these two solutions output space-time trajectories that would allow dynamic changes in velocity during execution (like horses getting tired and needing to slow down) and would require these to be explicitly planned initially or additional replanning afterwards.

We would like to just plan 1) A set of paths for agents to follow and 2) A coordination controller that sends coordination commands during execution to prevent collisions. Therefore during execution, agents can follow their paths as slow or fast dynamically as wanted, and only need to stop when requested by the coordination controller. Crucially both these are computed only once initially, removing the need for replanning under dynamic execution.

Main idea: Our main idea is that paths and coordination can be planned simultaneously. Our key innovation is that we can represent paths and a coordination controller in a single "space-coordination level" representation. Agents can move dynamically at arbitrary speeds between coordination commands unlike following space-time paths at predetermined velocities, while also eliminating the need to replan.

Coordination & Space-Level CBS: We define coordination as stop and continue commands that agents respect during dynamic execution. Our objective is to simultaneously compute paths and a coordination controller than minimizes the number of these commands. Building on the

ECBS framework, we can plan w_{so} bounded suboptimal space-level paths where each agent's path contains spatially independent (from other agents) path segments connected by transitions between adjacent levels. We show how we can explicitly reason about coordination commands by interpreting each transition between space-levels as a coordination command. This formulation allows us to minimize the total coordination (levels) L or a weighted objective $w * L + (1 - w) * P$ of coordination L and path cost P. We subsequently prove that with $w = 1, w_{so} = 1$ SL-CBS can theoretically return minimal coordination solutions, but note in practice only the bounded suboptimal version was able to solutions in reasonable runtimes.

Results: We test our method on 8 diverse maps and compare it against an improved MAPF-POST baseline from Hönig et al. (2016). We first show that bounded suoptimal SL-CBS can return solutions that minimize coordination in competitive times (and sometimes even faster). In general we see that SL-CBS reduces coordination by slightly increasing our path cost, highlighting how our joint reasoning allows us to reduce coordination by planning slightly longer paths that avoid congestion. Table 2 provides detailed results on a large, medium, and small map to show how this holds and varies across different map sizes and scenarios.

With $w = 0.2$, across all 8 maps we generally reduce coordination by 20-40% with 5-15% increase in path cost while planning faster than the baseline. With $w = 0.4, w = 0.6$ we can get an even larger coordination reduction of 30+% while runtime is comparable for a smaller number of agents and only gets significantly slower for a larger number of agents. For context, Age of Empires scenarios have a density of 25 agents per 30x30 tiles. In our experiments on 32x32 maps, SL-CBS with $w = 0.2, w = 0.4$ always runs as fast as ECBS and reduces coordination 15-60% with a path cost increase of at max 6%.

Acknowledgements

This material is partially supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE1745016 and DGE2140739. Rishi V. and Aidan W. would like to thank Dhruv S. and Shivam V. for their helpful feedback for writing this paper, and Li, Ruml, and Koenig (2021) for their open-source codebase.

References

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2018. Robust multi-agent path finding. In *Eleventh Annual Symposium on Combinatorial Search*.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Seventh Annual Symposium on Combinatorial Search*.

Boardman, B.; Harden, T.; and Martínez, S. 2021. Multiagent motion planning with sporadic communications for collision avoidance. *IFAC Journal of Systems and Control*, 15: 100126.

Boyarski, E.; Felner, A.; Le Bodic, P.; Harabor, D. D.; Stuckey, P. J.; and Koenig, S. 2021. f-Aware Conflict Prioritization; Improved Heuristics For Conflict-Based Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14): 12241–12248.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, E. 2015. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Chen, Z.; Harabor, D.; Li, J.; and Stuckey, P. J. 2021. Symmetry Breaking for k-Robust Multi-Agent Path Finding. *CoRR*, abs/2102.08689.

Hönig, W.; Kumar, T. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-agent path finding with kinematic constraints. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*.

Kou, N. M.; Peng, C.; Yan, X.; Yang, Z.; Liu, H.; Zhou, K.; Zhao, H.; Zhu, L.; and Xu, Y. 2019. Multi-Agent Path Planning with Non-Constant Velocity Motion. AAMAS '19, 2069–2071. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450363099.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *IJCAI*, volume 2019, 442–449.

Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2020. New techniques for pairwise symmetry breaking in multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 193–201.

Li, J.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021. Pairwise Symmetry Reasoning for Multi-Agent Path Finding Search. *CoRR*, abs/2103.07116.

Li, J.; Ruml, W.; and Koenig, S. 2021. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 12353–12362.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219: 40–66.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*, 151–158.

van den Berg, J.; Lin, M.; and Manocha, D. 2008. Reciprocal Velocity Obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, 1928–1935.

Wang, X.; Sahin, A.; and Bhattacharya, S. 2022. Coordination-free Multi-robot Path Planning for Congestion Reduction Using Topological Reasoning. *CoRR*, abs/2205.00955.