Multi-Agent Pathfinding with Real-Time Heuristic Search

Devon Sigurdson*, Vadim Bulitko*, William Yeoh†, Carlos Hernández‡ and Sven Koenig§
* Computing Science, University of Alberta
Email: {dbsigurd,buliko}@ualberta.ca
† Computer Science and Engineering, Washington University in St. Louis
Email: wyeoh@wustl.edu
‡ Ciencia de la Ingenieria, Universidad Andres Bello
Email: carlos.hernandez.u@unab.cl
§ Computer Science, University of Southern California
Email: skoenig@usc.edu

***Abstract*—Multi-agent pathfinding, namely finding collision-free paths for several agents from their given start locations to their given goal locations on a known stationary map, is an important task for non-player characters in video games. A variety of heuristic search algorithms have been developed for this task. Non-real-time algorithms, such as Flow Annotated Replanning (FAR), first find complete paths for all agents and then move the agents along these paths. However, their searches are often too expensive. Real-time algorithms have the ability to produce the next moves for all agents without finding complete paths for them and thus allow the agents to move in real time. Real-time heuristic search algorithms have so far typically been developed for single-agent pathfinding. We, on the other hand, present a real-time heuristic search algorithm for multi-agent pathfinding, called Bounded Multi-Agent A* (BMAA*), that works as follows: Every agent runs an individual real-time heuristic search that updates heuristic values assigned to locations and treats the other agents as (moving) obstacles. Agents do not coordinate with each other, in particular, they neither share their paths nor heuristic values. We show how BMAA* can be enhanced by adding FAR-style flow annotations and allowing agents to push other agents temporarily off their goal locations, when necessary. In our experiments, BMAA* has higher completion rates and lower completion times than FAR.**

I. Introduction

Pathfinding is a core task in many video games, for example, to allow *non-player characters* (NPCs) to move to given goal locations on a known stationary map. A* [1] is a classic algorithm for single-agent pathfinding. The artificial intelligence algorithms in video games, however, often need to find collision-free paths for several agents to their given goal locations. Figure 1 illustrates *multi-agent pathfinding* (MAPF) [2] on a map from the *Dragon Age: Origins* video game [3], where NPCs (green dots) have to move to their given goal locations (red dots).

The constraints on MAPF algorithms depend on the application. For example, real-time strategy games, such as *StarCraft* [4], require the NPCs to move simultaneously in real time, which limits the amount of time available to compute the next moves for all NPCs before they need to start moving. Video games can generate maps procedurally to create new game levels on the fly, which makes it impossible to preprocess the maps. Players can often re-task NPCs at will or the map can change, rendering their previously calculated paths obsolete on a moment's notice. Finally, game settings can limit the amount of coordination allowed among characters in the game (such as sharing their paths or heuristic values), and some characters might not even be under the complete control of the system (because they are on an opposing team).

These constraints motivated our development of *Bounded Multi-Agent A** (BMAA*) — a MAPF algorithm that operates in real time, loses only a small amount of search in case players re-task NPCs or the map changes and neither requires explicit inter-agent coordination, complete control of all NPCs nor preprocessing of maps. BMAA* works as follows: Every agent treats the other agents as (moving) obstacles, runs an individual real-time heuristic search that searches the map around its current location within a given lookahead to select the next move and updates heuristic values assigned to locations to avoid getting stuck. We show how BMAA* can be enhanced by, first, adding flow annotations from the MAPF algorithm FAR [5] (that impose move directions similar to one-way streets) and, second, allowing agents to push other agents temporarily off their goal locations, when necessary, if agents are allowed to send each other move requests. In our experiments, BMAA* has higher completion rates and smaller completion times than FAR, thus demonstrating the promise of real-time heuristic search for MAPF.

II. Problem Formulation

A MAPF problem is defined by a pair (G, A). $G = (N, E, c)$ is an undirected weighted graph of nodes N connected via edges $E \subseteq N \times N$. The costs $c(e)$ of all edges $e \in E$ are strictly positive with the following exceptions: There exists an edge for every node that connects the node to itself, allowing the agent to always wait in its current node. The costs of these edges are zero. $A = \{a^1, \ldots, a^n\}$ is a set of agents. Each agent $a^i \in A$ is specified by the pair $(n^i_{\text{start}}, n^i_{\text{goal}})$ of its start node n^i_{start} and goal node n^i_{goal}. We use graphs that

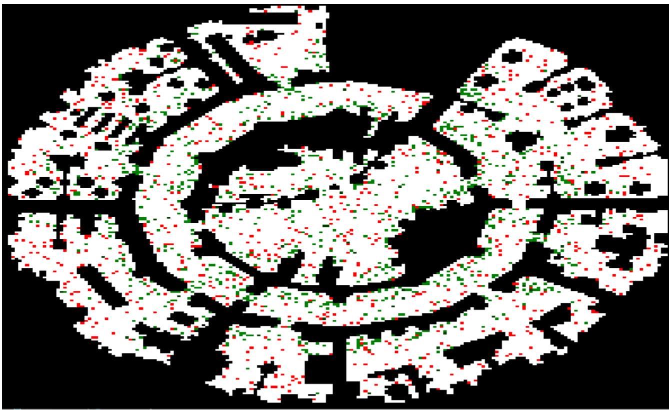

Fig. 1: NPCs on a *Dragon Age: Origins* map [3].

correspond to rectangular 8-neighbor grids, as is common for video games. The nodes correspond to the cells not blocked by stationary obstacles. The nodes of two neighboring cells are connected with an edge. The costs of these edges are one for cardinal neighbors and $\sqrt{2}$ for diagonal neighbors.

Time advances in discrete steps. Every agent always occupies exactly one node at every time step. We use $n^i_{\text{curr}} \in N$ to refer to the current node of agent a^i. The agent determines a prefix P of a path from its current node to its goal node and sends it to a central NPC controller. $P(n)$ is the successor node of node n on the path. The central NPC controller then moves the agent from node n^i_{curr} to node $P(n^i_{\text{curr}})$ with the following exceptions: The agent waits in its current node if $P(n^i_{\text{curr}})$ is undefined or the agent would collide with another agent. Two agents collide iff they swap nodes or move to the same node from one time step to the next one.

We use the following performance measures: The *completion rate* is the percentage of agents that are in their goal locations when the runtime limit has been reached [5], [6]. The *completion time* for an agent is the time step when that agent last reached its goal location. If an agent leaves its goal and does not return the completion time is undefined. Finally, the *travel distance* of an agent is the sum of the costs of the edges traversed by that agent. We consider the mean of all agents' travel distance and the mean of all agents' completion time as the performance measures in our MAPF problems. These performance measures cannot be optimized simultaneously. Their desired trade-off can be game specific. We choose to maximize the completion rate (because players will notice if NPCs do not reach their goal locations) but report on the other two performance measures as well.

III. Related Work

We now review search algorithms that are related to BMAA*, focusing on pathfinding with heuristic search algorithms, which use heuristic values to focus their search.

A. A*

A* [1] provides the foundation for our BMAA* and many other MAPF algorithms, even though it was developed for single-agent pathfinding. An A* search for an agent explores the search space starting at its current node. The exploration is informed by heuristic values and driven toward nodes with a low estimate of the estimated cost of moving from the current node via them to the goal node. Algorithm 1 shows the pseudo-code for a version of A* that finds a cost-minimal path for agent a^i from its current node n^i_{curr} to its goal node n^i_{goal} under mild assumptions about the graph and the heuristic values.[1] It maintains two lists of nodes, namely the closed and open lists. The closed list is initially empty (line 3), and the open list contains the current node (line 4). The closed list is an unordered set of nodes that A* has already expanded. The open list is an ordered set of nodes that A* considers for expansion. A* always expands a node in the open list with the lowest f-value next, where the f-value of node n is $f(n) = g(n)+h(n)$. Its g-value $g(n)$ is the cost of the lowest-cost path from the current node to node n discovered so far, and its h-value $h(n)$ (or, synonymously, heuristic value) is the heuristic estimate of the cost of a lowest-cost path from node n to the goal node. (The g-values are initially zero for the start node and infinity for all other nodes.) A* removes node n from the open list and adds it to the closed list (lines 11 and 12). It then expands the node by iterating over all of its neighbors n'. It updates the g-value of node n' if node n' has not yet been expanded (i.e., it is not yet in the closed list) and the g-value of node n' can be decreased due to the fact that the cost of the path from the current node via node n to node n' is smaller than the g-value of node n' (because the search has then discovered a lower-cost path from the current node to node n') (line 19). In this case, it also updates the parent of node n' to node n (line 20) and adds it to the open list if it is not already in it (line 22). A* continues its search until either the open list is empty (line 6) or the node about to be expanded is the goal node (line 7). In the former case, no path exists from the current node to the goal node. In the latter case, the path P that is obtained by repeatedly following the parents from the node about to be expanded to the current node is a cost-minimal path from the current node to the goal node in reverse (line 8).

B. Online MAPF Algorithms

We focus on online MAPF algorithms, where there entire problem is not required to be solved before agents begin moving. since we are interested in MAPF algorithms that operate in a short amount of time, lose only a small amount of search in case players re-task NPCs or the map changes and neither require explicit inter-agent coordination, complete control of all NPCs nor preprocessing of maps. We describe only the most suitable online MAPF algorithms below.

*Windowed Hierarchical Cooperative A** (WHCA*) [6] finds collision-free paths for all agents for their next window of

[1] In our pseudo-code, *First* returns a node with the smallest f-value in the open list (breaking ties in favor of a node with the largest g-value, with any remaining ties broken by first-in first-out); *Pop* removes a node with the smallest f-value from the open list (breaking ties in favor of a node with the largest g-value) and returns it; *Add* adds an element to a list; and *GetNeighbors* returns all neighboring nodes of a node in the graph.

Algorithm 1 A*.

```
1:  procedure A*
2:      P ← ()
3:      closed ← ∅
4:      open ← {n^i_curr}
5:      g(n^i_curr) ← 0
6:      while open ≠ ∅ do
7:          if open.First() = n^i_goal then
8:              calculate P
9:              break
10:         end if
11:         n ← open.Pop()
12:         closed.Add(n)
13:         for n′ ∈ n.GetNeighbors() do
14:             if n′ ∉ closed then
15:                 if n′ ∉ open then
16:                     g(n′) ← ∞
17:                 end if
18:                 if g(n′) > g(n) + c(n, n′) then
19:                     g(n′) ← g(n) + c(n, n′)
20:                     n′.parent ← n
21:                     if n′ ∉ open then
22:                         open.Add(n′)
23:                     end if
24:                 end if
25:             end if
26:         end for
27:     end while
```

moves. It shares the paths of all agents up to the given move limit through a reservation table, which adds a time dimension to the search space and thus results in expensive searches. Beyond the move limit, WHCA* simply assumes that every agent follows the cost-minimal path to its goal node and thus ignores collisions among agents. The move limit needs to be sufficiently large to avoid conflicts among agents, resulting in searches that might exceed the amount of time available to compute the next moves for all NPCs before they need to start moving. Furthermore, WHCA* requires all NPCs to be under its complete control.

Flow Annotated Replanning (FAR) [5] combines the reservation table from WHCA* with flow annotations that make its searches less expensive since no time dimension has to be added to the search space. Each agent has to reserve its next moves before it executes them. Agents do not incorporate these reservations into their search but simply wait until other agents that block them have moved away, similar to waiting at traffic lights. FAR attempts to break deadlocks (where several agents wait on each other indefinitely) by pushing some agents temporarily off their goal nodes. However, agents can still get stuck in some cases. The flow annotations of FAR [5] change the edges of the original graph G in order to reduce the number of collisions among agents. They effectively make the undirected original graph directed by imposing move directions on the edges, similar to one-way streets, which reduces the potential for head-to-head collisions among agents. This annotation is done on a grid in a way so that any node remains reachable from all nodes from which it could be reached on the original graph, as follows: The new graph initially has no edges. The first row of nodes is connected via westbound edges, the second row is connected via eastbound edges, and so on. Similarly, the first column of nodes is connected via northbound edges, the second column is connected via southbound edges, and so on. Sink nodes (with only in-bound edges) and source nodes (with only out-bound edges) are handled by adding diagonal edges adjacent to them. If sink and source nodes are in close proximity of each other, the diagonal edges can end up pointing at each other and result in a loss of connectivity, in which case additional undirected edges are added around them. Corridor edges (that is, edges on paths whose interior nodes have degree two) of the original graph remain undirected, which is important in case the corridor is the only connection between two components of the original graph. A standard implementation of A* is then used to search for a path to the goal in this restricted graph.

C. Real-time Heuristic Search

Video games often require NPCs to start moving in a short amount of time, which may not be possible with any of the search algorithms reviewed above since they need to compute a complete path before an agent can execute the first move. *Real-time heuristic search* (RTHS) algorithms, on the other hand, perform a constant amount of search per move regardless of the size of the map or the distance between the start and goal nodes. They have been studied for single-agent pathfinding [7]–[10], starting with the seminal work by Korf [11]. They need to compute only the prefix of a path before the agent can execute the first move — and repeat the operation until the agent reaches the goal node. To avoid cycling forever without reaching the goal node due to the incompleteness of the searches, the algorithms update the heuristic values over time by making them locally consistent [11], incrementally building the open and closed lists [12] or ignoring parts of the map [13]. There are two benefits to using RTHS algorithms in video games. First, an NPC can start moving in a short amount of time. Second, only a small amount of search is lost in case a player re-tasks NPCs or the map changes.

A well-known RTHS algorithm *Real-Time Adaptive A** (RTAA*) [14] performs an A* search, limited to a given number of node expansions. RTAA* then uses the f-value of the node A* was about to expand to update the heuristic values of all expanded nodes (that is, all nodes in the closed list *closed*) as shown in Procedure *Update-Heuristic-Values* in Algorithm 4. The agent then moves along the path from its current node to the node A* was about to expand, limited to a given number of moves — and RTAA* repeats the operation.

IV. Our Approach: BMAA*

Our *Bounded Multi-Agent A** (BMAA*) is a MAPF algorithm where every agent runs its own copy of RTAA*. BMAA* satisfies our requirements: It operates in real-time,

loses only a small amount of search in case players re-task NPCs or the map changes. Additionally, it does not requires explicit inter-agent coordination, complete control of all NPCs or preprocessing of maps. The design of BMAA* is modular to allow for extensions by adding or changing modules. For example, BMAA* can be enhanced by, first, adding flow annotations from FAR and, second, allowing agents to push other agents temporarily off their goal nodes, when necessary, if agents are allowed to send each other move requests.

We parameterize BMAA* as follows in the spirit of recent research in the context of *Parameterized Learning Real-Time A** [15]: First, $expansions$ is the limit on the number of node expansions of the A* search of RTAA*. Second, $vision$ is the distance within which agents can see other agents. Third, $moves$ is the number of moves that each agent makes along its path before RTAA* determines a new path for the agent. Fourth, $push$ is a Boolean flag that determines whether agents can push other agents temporarily off their goal nodes. Finally, $flow$ is a Boolean flag that determines whether RTAA* uses the flow annotations from FAR.

A. Procedure NPC-Controller

Algorithm 2 shows the pseudo-code for the central NPC controller. The time step $time$ is initialized to zero at the start of BMAA*, and the central NPC controller is then invoked at every time step with A, the set of agents currently under the control of the system. In the search phase, the central NPC controller lets every agent under the control of the system use the Procedure *Search-Phase* shown in Algorithm 3 to find a prefix of a path from its current node to its goal node (line 3, Algorithm 2). In the subsequent execution phase, the central NPC controller iterates through all agents under the control of the system: First, it retrieves the node that the agent should move to next, which is the successor node of the current node of the agent on its path (line 7, Algorithm 2). Second, if the desired node is blocked by an agent that has reached its own goal node already and agents can push other agents temporarily off their goal nodes ($push = true$), it can push the blocking agent to any neighboring node (line 9, Algorithm 2). The blocking agent returns to its own goal node in subsequent time steps since all agents always execute RTAA* even if they are in their goal nodes. Finally, it moves the agent to the desired node if that node is (no longer) blocked (line 12, Algorithm 2) and increments the current time step (line 16, Algorithm 2).

B. Procedure Search-Phase

Algorithm 3 shows the pseudo-code for the search phase. It finds a new prefix of a path from the current node of the agent to its goal node when it has reached the end of the current path, the current node is no longer on the path (for example, because the agent has been pushed away from its goal node), or the agent has already executed $moves$ moves along the path. (The "expiration" time step $limit$ for the path keeps track of the last condition on line 2 and is set on line 8.) If so, then it uses Procedure *Search* in Algorithm 5 to execute an RTAA* search (line 3) and uses Procedure *Update-Heuristic-Values* to update the heuristic values afterward (lines 5-7).

Algorithm 2 BMAA*'s NPC Controller.

```
1:  procedure NPC-CONTROLLER(A)
2:      for all a^i ∈ A do
3:          a^i.Search-Phase()
4:      end for
5:      for all a^i ∈ A do
6:          if a^i.P(n^i_curr) is defined then
7:              n ← a^i.P(n^i_curr)
8:              if push ∧ n is blocked by agent a^j then
9:                  a^j.PushAgent()
10:             end if
11:             if n is not blocked by an agent then
12:                 a^i.MoveTo(n)
13:             end if
14:         end if
15:     end for
16:     time ← time + 1
```

Algorithm 3 BMAA*'s Search Phase.

```
1:  procedure SEARCH-PHASE
2:      if Search.P(n^i_curr) is undefined or time ≥ limit then
3:          Search()
4:          if Search.open ≠ ∅ then
5:              n ← Search.open.First()
6:              f ← g(n) + h(n)
7:              Update-Heuristic-Values(Search.closed, f)
8:              limit ← time + moves
9:          end if
10:     end if
```

Algorithm 4 BMAA*'s Update Phase.

```
1:  procedure UPDATE-HEURISTIC-VALUES(closed, f)
2:      for n ∈ closed do
3:          h(n) ← f − g(n)
4:      end for
```

C. Procedure Search

Algorithm 5 shows the pseudo-code for an A* search, as discussed before, but with the following changes: First, each agent maintains its own heuristic values across all of its searches. Second, the search also terminates after it has expanded $expansions$ nodes. Thus, the path P obtained on line 9 by repeatedly following the parents from the node about to be expanded to the current node is now only the prefix of a path from the current node of the agent to its goal node. Finally, $GetNeighbors$ returns a node's neighbours that are not blocked by stationary obstacles. However, other agents within the straight-line distance $vision$ within which agents can see other agents are treated as obstacles as long as they do not block its goal node. Thus, the corresponding nodes are immediately discarded (lines 16-18). If RTAA* uses the

Algorithm 5 BMAA*'s Version of A*.

1: ***procedure*** SEARCH
2: $P \leftarrow ()$
3: $exp \leftarrow 0$
4: $closed \leftarrow \emptyset$
5: $open \leftarrow \{n^i_{curr}\}$
6: $g(n^i_{curr}) \leftarrow 0$
7: ***while*** $open \neq \emptyset$ ***do***
8: ***if*** $open.First() = n^i_{goal} \vee exp \geq expansions$ ***then***
9: *calculate* P
10: *break*
11: ***end if***
12: $n \leftarrow open.Pop()$
13: $closed.Add(n)$
14: ***for*** $n' \in n.GetNeighbors(flow)$ ***do***
15: $d \leftarrow distance(n^i_{curr}, n')$
16: ***if*** n' *is blocked by an agent* $\wedge\, d \leq vision$ ***then***
17: ***if*** $n' \neq n^i_{goal}$ ***then***
18: *continue*
19: ***end if***
20: ***end if***
21: ***if*** $n' \notin closed$ ***then***
22: ***if*** $n' \notin open$ ***then***
23: $g(n') \leftarrow \infty$
24: ***end if***
25: ***if*** $g(n') > g(n) + c(n, n')$ ***then***
26: $g(n') \leftarrow g(n) + c(n, n')$
27: $n'.parent \leftarrow n$
28: ***if*** $n' \notin open$ ***then***
29: $open.Add(n')$
30: ***end if***
31: ***end if***
32: ***end if***
33: ***end for***
34: $exp \leftarrow exp + 1$
35: ***end while***

flow annotations from FAR ($flow = true$), then $GetNeighbors$ returns only those neighboring nodes of a node of the graph which are reachable from the node via the flow annotations from FAR. The flow annotations are not computed in advance but generated the first time the node is processed and then cached so that they can be re-used later.

V. Experimental Evaluation

We experimentally evaluate four versions of BMAA* both against FAR and against A*-Replan, which is equivalent to FAR with no flow annotations. BMAA* cannot push other agents temporarily off their goal locations ($push = false$) and uses no flow annotations ($flow = false$), BMAA*-c can push other agents temporarily off their goal locations, BMAA*-f uses flow annotations, and BMAA*-f-c combines both features. All BMAA* versions use the parameters $lookahead = 32$, $moves = 32$ and $vision = \sqrt{2}$. We choose these parameters on the basis of preliminary experiments. Increasing

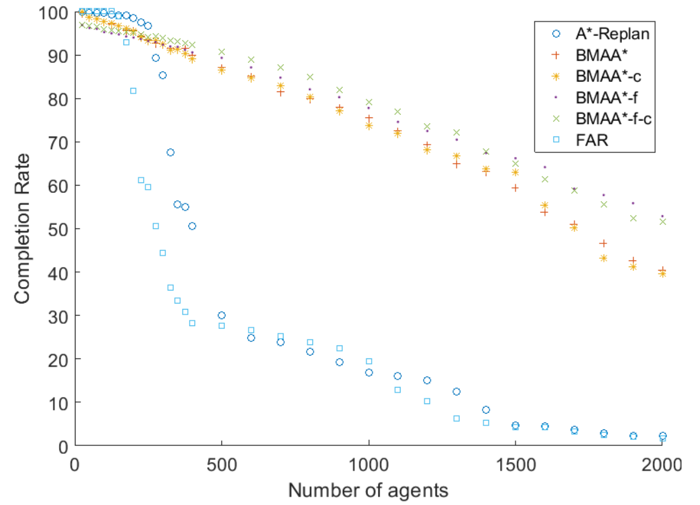

Fig. 2: Completion rates averaged over all MAPF instances.

$lookahead$ often decreases the travel distance at the cost of increasing the search time per move. Increasing $vision$ often reduces the completion rate since it makes agents react to far-away agents. FAR and A*-Replan use a reservation size of three, as suggested by the creators of FAR, meaning that agents must successfully reserve their next three moves before they execute them. All MAPF algorithms use the octile heuristic values as heuristic values (or, in case of BMAA*, to initialize them), are coded in C# and are run on a single Intel Broadwell 2.1Ghz CPU core with 3GB of RAM and a runtime limit of 30 seconds per MAPF instance, which is sufficiently large to allow for full A* searches.

We evaluate them on ten maps from the MovingAI benchmark set [16]. We use three maps from *Dragon Age: Origins* (DAO), three maps from *WarCraft III* (WCIII), three maps from *Baldur's Gate II* (BGII) (resized to 512×512) and one map from *Baldur's Gate II* in its original size. We create ten MAPF instances for each map with the number of agents ranging from 25 to 400 in increments of 25 and from 400 to 2000 in increments of 200. We assign each agent unique randomly selected start and goal locations which are reachable from each other in the absence of other agents.

A. Aggregate Completion Rate Results

Figure 2 shows the completion rates of all MAPF algorithms averaged over all MAPF instances on all maps. The completion rates of all MAPF algorithms decrease as the number of agents increases because the congestion and amount of search (since every agent has to search) increase. A higher congestion makes it more difficult for agents to reach their goal locations, and a higher amount of search makes it more likely that the runtime limit is reached.

All BMAA* versions have substantially higher completion rates than FAR and A*-Replan for more than 200 agents, with the BMAA* versions that can push other agents temporarily off their goal locations being slightly better than the other BMAA* versions. This can be explained as follows: FAR and

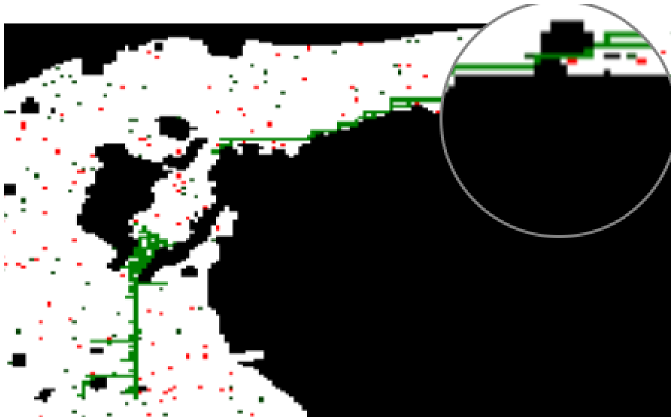

Fig. 3: Issue for FAR: One-cell-wide corridors.

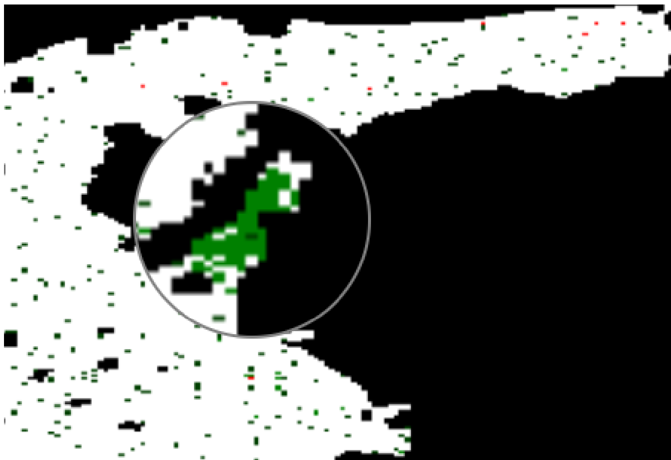

Fig. 4: Issue for BMAA*: Dead ends.

A*-Replan determine complete paths for the agents, which results in many agents sharing path segments and thus creates congestion around choke points, such as the one-cell-wide corridor in Figure 3. The BMAA* versions often avoid this behavior, for two reasons: First, the agents of the BMAA* versions have larger travel distances than the ones of FAR and A*-Replan. While the large travel distances of RTHS algorithms are viewed as a major deficiency in the context of single-agent pathfinding, they are beneficial in the context of MAPF since they avoid congestion around choke points. Second, the agents of the BMAA* versions treat the other agents as (moving) obstacles and thus find paths that avoid choke points that are blocked by other agents and thus appear impassable, while the agents of FAR and A*-Replan assume that they can resolve conflicts in their paths with those of other agents and thus move toward choke points.

However, the BMAA* versions also have disadvantages: First, they might move agents into dead ends, such as the one shown in Figure 4, if the initial heuristic values are misleading (resulting in depressions in the heuristic value surface). This well-known issue for RTHS algorithms is addressed by them updating their heuristic values. Several recent RTHS techniques attempt to reduce the travel distances but, of course, agents exploring new areas in imperfect manners could also be viewed as realistic in some cases. Second, even the BMAA* versions that can push other agents temporarily off their goal locations might not be able to move all agents to their goal locations when other agents on their paths are unable to vacate their own goal locations (Figure 5).

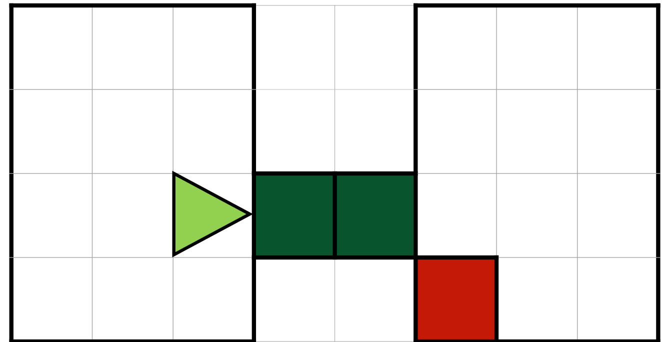

Fig. 5: Unsolvable MAPF instance for the BMAA* versions, where the triangular agent has to move to its red goal location while the dark green square agents are already at their own goal locations in a one-cell-wide corridor.

B. *Per-Map Results*

Tables I-III show the three performance measures for all MAPF algorithms averaged over all MAPF instances for each of the maps separately since the map features affect the performance of the MAPF algorithms. The best results are highlighted in bold.

1) Per-Map Completion Rate Results: Table I shows that BMAA*-f-c has the highest completion rates on seven out of the ten maps but the completion rate of BMAA*, for example, is 15 percent larger than the one of BMAA*-f-c on map DAO-lak307d.

2) Per-Map Completion Time Results: The completion rates of FAR and A*-Replan drop substantially for more than 200 agents, as shown in Figure 2. We thus limit the number of agents to 200 since most agents then reach their goal locations. We assign the remaining agents a completion time of 30 seconds. Table II shows that BMAA*-f has the lowest completion times on five maps and BMAA* has the lowest completion times on the remaining four maps.

3) Per-Map Travel Distance Results: We again limit the number of agents to 200 since most agents then reach their goal locations. We assign the remaining agents their travel distances when the runtime limit is reached. Table III shows that FAR has the lowest travel distances on nine maps.

VI. Conclusions

Our paper considered an important problem faced by artificial intelligence in many video games, namely MAPF. We reviewed recent related work and argued for the use of

TABLE I: Completion rates averaged over all MAPF instances for each map.

Map Name	A*-Replan	BMAA*	BMAA*-c	BMAA*-f	BMAA*-f-c	FAR	Overall
BGII-AR0414SR (320*281)	45	87	87	85	**89**	32	71
BGII-AR0414SR (512*512)	14	80	79	82	**83**	07	58
BGII-AR0504SR (512*512)	08	51	51	**62**	**62**	05	40
BGII-AR0701SR (512*512)	08	48	49	64	**65**	06	40
WCIII-blastedlands (512*512)	14	**85**	**85**	78	80	03	58
WCIII-duskwood (512*512)	08	58	58	**67**	**67**	03	43
WCIII-golemsinthemist (512*512)	10	59	59	**72**	**72**	04	46
DAO-lak304d (193*193)	19	39	38	**53**	51	27	38
DAO-lak307d (84*84)	60	**79**	77	68	64	60	68
DAO-lgt300d (747*531)	12	65	65	**77**	**77**	10	51
Overall	20	65	65	71	71	16	51

TABLE II: Completion times (in seconds) averaged over all MAPF instances for each map.

Map Name	A*-Replan	BMAA*	BMAA*-c	BMAA*-f	BMAA*-f-c	FAR	Overall
BGII-AR0414SR (320*281)	2.8	**1.2**	5.1	2.2	5.6	3.8	3.5
BGII-AR0414SR (512*512)	8.8	3.6	6.6	**3.0**	6.8	12.9	7.0
BGII-AR0504SR (512*512)	12.3	8.6	12.7	**6.3**	12.5	16.0	11.4
BGII-AR0701SR (512*512)	12.7	4.0	5.4	**3.2**	4.5	15.0	7.5
WCIII-blastedlands (512*512)	8.8	**1.4**	1.5	2.2	2.3	21.0	6.2
WCIII-duskwood (512*512)	12.5	4.1	5.8	**3.7**	5.5	21.1	8.8
WCIII-golemsinthemist (512*512)	11.1	4.2	5.9	**3.0**	4.2	19.0	7.9
DAO-lak304d (193*193)	4.5	6.7	15.1	7.9	11.4	**3.2**	8.1
DAO-lak307d (84*84)	**0.2**	**0.2**	**0.2**	0.5	0.3	0.6	0.3
DAO-lgt300d (747*531)	8.3	**1.4**	1.6	2.2	2.4	10.5	4.4
Overall	8.2	3.5	6.0	**3.4**	5.5	12.3	6.5

TABLE III: Travel distances averaged over all MAPF instances for each map.

Map Name	A*-Replan	BMAA*	BMAA*-c	BMAA*-f	BMAA*-f-c	FAR	Overall
BGII-AR0414SR (320*281)	663	554	557	620	639	**130**	527
BGII-AR0414SR (512*512)	661	1538	1557	2080	2115	**224**	1363
BGII-AR0504SR (512*512)	407	2167	2231	3671	3783	**227**	2089
BGII-AR0701SR (512*512)	562	973	967	1267	1287	**322**	896
WCIII-blastedlands (512*512)	299	376	376	775	784	**268**	480
WCIII-duskwood (512*512)	367	1179	1188	1712	1737	**257**	1073
WCIII-golemsinthemist (512*512)	530	1205	1206	1371	1369	**285**	994
DAO-lak304d (193*193)	2154	1425	1460	1258	1295	**148**	1290
DAO-lak307d (84*84)	578	**38**	39	125	95	47	154
DAO-lgt300d (747*531)	435	403	404	592	603	**289**	454
Overall	666	986	998	1347	1371	**225**	932

real-time heuristic search. We then contributed a new real-time MAPF algorithm, BMAA*, which is of modular design and can be enhanced with recent flow-annotation techniques. BMAA* has higher completion rates and smaller completion times than FAR at the cost of longer travel distances, which is a good trade-off since NPCs reaching their goal locations via possibly longer paths is less noticeable by players than NPCs not reaching their goal locations at all. Finally, we discussed what makes MAPF difficult for different algorithms, paving the road to per-problem algorithm selection techniques in the spirit of recent research in the context of single-agent pathfinding [17], [18].

Overall, BMAA* demonstrates the promise of real-time heuristic search for MAPF. Its main shortcoming is its large travel distances compared to the ones of FAR. Several recent RTHS techniques attempt to reduce the travel distances for single-agent pathfinding [19] and thus might also be able to reduce the travel distances for BMAA*. Examples include search space reduction techniques [13], [20], precomputation techniques [21], [22] and initialization techniques for the heuristic values, which might help to reduce the dead-end problem shown in Figure 4.

ACKNOWLEDGMENTS

Devon Sigurdson and Vadim Bulitko appreciate support from NSERC and Nvidia. Carlos Hernández was partially funded by Fondecyt grant number 1161526. Sven Koenig was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987 and 1319966 as well as a gift from Amazon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed

or implied, of the sponsoring organizations, agencies or the U.S. government.

References

[1] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] S. Koenig and H. Ma, "AI buzzwords explained: Multi-agent path finding (MAPF)," *AI Matters*, 2017.

[3] BioWare, "Dragon Age: Origins," 2009.

[4] "Starcraft," https://starcraft2.com/en-us/, accessed: 2018-03-14.

[5] K.-H. C. Wang and A. Botea, "Fast and memory-efficient multi-agent pathfinding," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2008, pp. 380–387.

[6] D. Silver, "Cooperative pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2005, pp. 117–122.

[7] T. Ishida, *Real-time search for learning autonomous agents*. Springer Science & Business Media, 1997, vol. 406.

[8] V. Bulitko and G. Lee, "Learning in real time search: A unifying framework," *Journal of Artificial Intelligence Research*, vol. 25, pp. 119–157, 2006.

[9] S. Koenig and X. Sun, "Comparing real-time and incremental heuristic search for real-time situated agents," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 3, pp. 313–341, 2009.

[10] B. Cserna, M. Bogochow, S. Chambers, M. Tremblay, S. Katt, and W. Ruml, "Anytime versus real-time heuristic search for on-line planning," in *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2016.

[11] R. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2–3, pp. 189–211, 1990.

[12] Y. Björnsson, V. Bulitko, and N. Sturtevant, "TBA*: Time-bounded A*," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2009, pp. 431–436.

[13] C. Hernandez, A. Botea, J. A. Baier, and V. Bulitko, "Online bridged pruning for real-time search with arbitrary lookaheads," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017, pp. 510–516.

[14] S. Koenig and M. Likhachev, "Real-time Adaptive A*," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2006, pp. 281–288.

[15] V. Bulitko, "Evolving real-time heuristic search algorithms," in *Proceedings of the International Conference on the Synthesis and Simulation of Living Systems*, 2016.

[16] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144–148, 2012.

[17] V. Bulitko, "Per-map algorithm selection in real-time heuristic search," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2016, pp. 143–148.

[18] D. Sigurdson and V. Bulitko, "Deep learning for real-time heuristic search algorithm selection," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2017, pp. 108–114.

[19] V. Bulitko and A. Sampley, "Weighted lateral learning in real-time heuristic search," in *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2016.

[20] C. Hernandez and J. A. Baier, "Avoiding and escaping depressions in real-time heuristic search," *Journal of Artificial Intelligence Research*, vol. 43, pp. 523–570, 2012.

[21] R. Lawrence and V. Bulitko, "Database-driven real-time heuristic search in video-game pathfinding," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 3, pp. 227–241, 2013.

[22] L. Cohen, T. Uras, T. Kumar, H. Xu, N. Ayanian, and S. Koenig, "Improved solvers for bounded-suboptimal multi-agent path finding," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2016, pp. 3067–3074.