Towards Smart Behavior of Agents in Evacuation Planning Based on Local Cooperative Path Finding

Róbert Selvek[1] and Pavel Surynek[1][0000−0001−7200−0542]

Faculty of Information Technology, Czech Technical University in Prague
Thákurova 9, 160 00 Prague, Czechia
{selverob,pavel.surynek}@fit.cvut.cz

Abstract. We address engineering of smart behavior of agents in evacuation problems from the perspective of *cooperative path finding* (CPF) in this paper. We introduce an abstract version of evacuation problems we call *multi-agent evacuation* (MAE) that consists of an undirected graph representing the map of the environment and a set of agents moving in this graph. The task is to move agents from the endangered part of the graph into the safe part as quickly as possible. Although the abstract evacuation task can be solved using centralized algorithms based on network flows that are near-optimal with respect to various objectives, such algorithms would hardly be applicable in practice since real agents will not be able to follow the centrally created plan. Therefore we designed a decentralized evacuation planning algorithm called LC-MAE based on local rules derived from local cooperative path finding (CPF) algorithms. We compared LC-MAE with near-optimal centralized algorithm using agent-based simulations in multiple real-life scenarios. Our finding it that LC-MAE produces solutions that are only worse than the optimum by a small factor. Moreover our approach led to important observations about how many agents need to behave rationally to increase the speed of evacuation. A small fraction of rational agents can speed up the evacuation dramatically.

Keywords: evacuation planning · cooperative path-finding · local algorithms · decentralized algorithms · agent-based simulations · real-life scenarios · network flows

1 Introduction

We address in this paper the evacuation problem from the point of view of engineering of smart behavior of individual evacuated agents. Evacuation planning represents an important real-life problem and is increasingly studied as a topic in artificial intelligence [4, 16]. The evacuation task consists of evacuation of people or other agents from the endangered area into the safe zone. The important computational challenge in evacuation represents the fact that centralized planning can hardly be applied as the task involves individual self-interested agents usually not willing to follow a centrally created plan.

Various techniques have been applied to address the evacuation problem both from the *centralized* and *decentralized* point of view including modeling the problem as *network flows* [1] or nature inspired computation such as *bee colony* optimization. The important distinguishing feature of evacuation planning algorithms is whether single evacuation route is being planned [20] or the problem is regarded as multi-agent scenario [19]. In multi-agent evacuation scenarios sometimes multiple types of agents are present such as those assisting the evacuation [15] and those being evacuated.

The environment where the evacuation task takes place is often modeled as a graph where vertices represent locations for agents, and edges model the possibility of moving between pairs of locations [14] (directed case may be used for representing one way path - a case often appearing in practice). Hence the evacuation problem can be interpreted as a variant of *path finding* or *cooperative path finding* [21, 25, 24, 23, 6] (CPF).

1.1 Related Work

Specifically in these problems graphs are used as abstractions for the environment. Similarly as in CPF, the evacuation modeling must take into account potential collisions between agents and solving techniques must ensure proper avoidance [7]. The collision avoidance in CPF is usually represented by a constraint of having at most one agent per vertex (in some versions of CPF more than one agent is allowed per vertex).

In contrast to CPF, where agents have unique individual goals (location/vertex), we usually do not distinguish between individual agents in the evacuation task. That is, an agent can evacuate itself to anywhere in the safe zone (not to a specific location in the safe zone). From the theoretical point of view, this feature makes evacuation planning algorithms similar to single commodity network flows [3] while the standard CPF is reducible to the *multi-commodity flow* problem [2].

Another important challenge in evacuation planning is represented by the execution of a plan by real agents. In real-life evacuation scenarios, we cannot simply assume that all agents will want to follow the plan. Centralized control of all agents is not feasible in the setup with self-interested agents. The real agent in evacuation scenario may for example prefer the nearest exit or a path through which it has arrived while a centrally created plan could force the agent go elsewhere, thus not being believable for the agent. This differs from classical planning [8], where the planning authority fully observes the environment, actions are assumed to be deterministic, and plans created in advance are assumed to be perfectly executed.

1.2 Contribution and Organization

Therefore in this work we focus on local evacuation planning relying on local cooperative path finding techniques and agent-based simulations. Our assumption is that evacuation paths planned locally using information available to the agent will be more realistic and could be executed by the real agent. At the same time we do not rule out the central aspect completely, as we also consider some agents to be more informed (about alternative exits, through a communication device) than others.

This paper is an extension of the original conference paper where the idea of evacuation planning using local cooperative path finding algorithms has been presented first time [18]. In this revised version we have extended the set of experiments and added more detailed explanation of network flow based algorithm that was omitted in the conference paper.

The organization of the paper is as follows. We begin with a formal introduction to the concept of evacuation planning, followed by a short summary of local cooperative path finding algorithms. Local CPF algorithms represent a basis of our novel evacuation algorithm called Local Cooperative Multi-agent Evacuation (LC-MAE) described next. Finally we present extensive experimental evaluation of LC-MAE in multiple scenarios using agent-based simulations. As part of the simulations, the algorithm is compared to a network-flow based algorithm which produces near-optimal plans.

2 Background

2.1 Evacuation Planning Formally

We introduce formal definition of evacuation task in this section. The abstract *multi-agent evacuation problem* (MAE) takes place in an undirected graph $G = (V, E)$. The set of vertices is divided into a set of *endangered* (D) vertices and a set of *safe* (S) vertices together modeling the zone to be evacuated and the safe zone. Agents from a set, $A = \{a_1, a_2, ..., a_k\}$, are distributed among the vertices and the task is to evacuate them from endangered to safe vertices.

The crisp variant of the problem requires that all agents are evacuated while in the optimization variant we want to have as many as possible agents in safety.

The MAE problem is similar to *cooperative path finding* (CPF) [21] from which we took the model for agent movement. Each agent is placed in a vertex of G so that there is at most one agent per vertex. The configuration of agents in vertices of the graph at time t will be denoted as $c_t : A \rightarrow V$. Similarly to CPF, an agent can move into a vacant adjacent vertex [1].

Multiple agents can move simultaneously, provided they do not collide with each other (that is, no two agents enter the same target vertex at the same time) and agents only enter vacant vertices.

Definition 1. Multi-agent evacuation (MAE) *is a 5-tuple* $\mathcal{E} = [G = (V, E), A, c_0, D, S]$, *where* G *represents the environment,* $A = a_1, a_2, ..., a_k$ *is a set of agents,* $c_0 : A \rightarrow V$ *is the initial configuration of agents,* D *and* S *such that* $D \subseteq V$, $S \subseteq V$, $V = D \cup S$ *with* $D \cap S \neq \emptyset$, *and* $|S| \geq k$ *represent a set of endangered and a set of safe vertices respectively.*

The task in MAE is to find a plan that moves all agents into the safe vertices (the crisp variant). That is we are searching for a plan $\pi = [c_0, c_1, ..., c_m]$ so that $c_m(a) \in S$ $\forall a \in A$. The total time until the last agent reaches the safe zone is called a *makespan*; the makespan of π is m. An illustration of simple evacuation problem is shown in Figure 1 and 2.

The assumption is that everything in the endangered zone will be destroyed at some unknown point in the future. Hence, to increase chances of evacuation, the makespan

[1] Alternative definitions of possible movements in CPF exist that for example permit train of agents to move simultaneously atc.

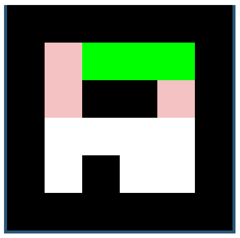


Fig. 1: Grid map depicting a multi-agent evacuation instance. Green squares represent agents that need to be evacuated from the pink endangered area to the white safe zone.

should be small. An evacuation plan with the near optimal makespan can be found in polynomial time using *network flow* techniques [26, 27, 1]. However, these algorithms require a centralized approach where agents perfectly follow the central plan which is hardly applicable in real-life evacuation scenarios [7, 11].

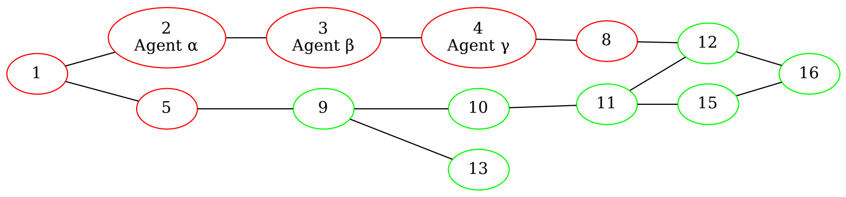


Fig. 2: MAE instance shown using undirected graph with the endangered and safe zone depicted using red and green vertices respectively .

2.2 Cooperative Path Finding Algorithms

We address MAE from a local point of view inspired by CPF algorithms like Local Repair A* (LRA*) and Windowed Hierarchical Cooperative A* (WHCA*) [21]. These algorithms feature a *decentralized* approach to cooperative path finding. Instead of using optimization techniques or network flow, each agent's path is found separately by using local rules, resolving conflicts between agents trying to enter the same vertex as they occur. In other words, each agent's next move is derived from the knowledge of the relative position of the goal vertex and agents in the neighborhood of the agent the move is planned for.

While LRA* only resolves conflicts at the moment agent tries to enter an occupied vertex, a naive strategy which can easily lead to deadlock, WHCA* is more advanced. Instead of only planning agents' paths in space (in graph), agents plan their paths in space-time and share them with other agents using a data structure called *reservation* *table* which is fact is expanded underlying graph over time. The time expansion is done by making a copy of the graph for each time step. In this data structure, we cab reserve space/time point for an agent once it goes through. When planning, vertices reserved by another agent at a given time are considered to be impassable.

If each agent planned and reserved its whole path to destination, agents planning later would have information about its complete route and their needs and goals would not be taken into account, leading to selfish behavior and deadlocks. The fix to this behavior is *windowing*, in which agents plan their paths only for a certain, small, number of time units and the planning is staggered, so that each agent has an option of reserving a certain vertex. The windowing mechanism supports local behavior of agents which is in line with our assumptions about real-agents in evacuation scenarios.

3 Local Multi-agent Evacuation (LC-MAE)

Our novel *local multi-agent evacuation* (LC-MAE) algorithm divides the evacuation task into three sub-problems:

- *Evacuation destination selection:* This sub-problem arises from the most important difference between MAE and simple CPF as in MAE agents' destinations/goals are not specified. Hence first we need to specify individual destination vertex for each agent.
- *Path-finding to safety:* Once each agent has picked its destination vertex in S, it has to find a collision-free path to the selected destination. At this stage the task is identical to CPF.
- *Behavior in the safe zone:* Agents that have left D and arrived to their destination vertex do not disappear from the map. We need to ensure that their behavior will not block other agents from entering S.

Agent movement in the last two sub-problems is based on modified versions of WHCA* algorithm, described in their respective sections.

3.1 Evacuation Destination Selection

The basic data structure used by LC-MAE when choosing an evacuation destination vertex is the *frontier* denoted F, $F \subseteq S$. The frontier is a set of safe vertices which separates the endangered zone from the safe zone.

In other other words, removing F from G will separate D and S into disconnected components. F is created on algorithm initialization. It holds that an agent must enter S by passing a vertex from F. So the frontier can be constructed as a standard *vertex cut* in a graph [17].

Destination selection uses a modified A* algorithm, inspired by the RRA* algorithm [21]. Agent's position is set as the path-finding goal, while all nodes in F are added to the initial *open set*. *Manhattan distance* [12] is used as the heuristic guiding the search of A*.

The result is a vertex in F that is located at the shortest true distance from the agent while, at the same time, being reachable by the agent. With the vertex at hand, the algorithm returns its true distance from the agent. This matches many real-world evacuation scenarios in which people are being evacuated from an area they know and thus have a mental map of the nearest exits [13].

While evacuating, agents keep track of the number of steps they have taken to reach their destination. Since the goal's true distance from the starting position is known, they can compare these two numbers. If the number of steps taken is significantly higher than the distance, it may indicate the agent has veered off the optimal path. This could be, for example, because the path to the chosen destination is congested. In that case, the agent repeats the destination selection process, an action that we call *retargeting*.

3.2 Reservation Table

In LC-MAE we use a variant of the reservation table used in the Cooperative A* algorithm [21]. Every vertex in G is associated with a mapping of time units to reservation structures. Every reservation structure includes a reference to the reserved vertex, the ID of the agent that created the reservation and a priority.

Associating priority with reservations is our primary distinguishing feature. Agents can make a reservation for vertex v and time step t provided they fulfill one of the following conditions:

1. No reservation exists yet for v at time t and the vertex can be reserved at time $t+1$.
2. A lower-priority reservation exists for v at time t and the vertex can be reserved at time $t+1$.
3. The agent holds a reservation for vertex v at $t-1$.

Condition 3 ensures that CPF algorithms used in LC-MAE can always perform at least one action, staying in place, without having to dynamically change the order in which agents' paths are planned or performing invalid actions, like colliding with another agent.

The $t+1$ reservability requirement in conditions 1 and 2 prevents the creation of so-called *trains* - lines or crowds of agents which move in the same direction at the same time and which reduce the simulation realism. A simple example of a train being formed is shown in the right column of figure 3.

3.3 Path-finding To Safety

Once the agent has picked its destination vertex s in S, it plans a path towards s using WHCA* with the RRA* heuristic. An agent plans the next part of its path on-demand when it has to return an action for the next time step and fulfills any of these conditions:

- Is more than halfway through its planned path[2]
- Has to retarget
- Has lost a reservation for a vertex on its planned path
- Is making its first step

[2] Only using half of the planned path before replanning is a simple way of improving agent cooperation described in [22]

Endangered agents are processed before agents located in S, thus being prioritized relative to agents that are in S. This ensures that endangered agents generate their plans first.

As soon as an agent enters S (even if it is not its current destination vertex), it stops following the planned path and switches to the behavior described in the next section.

3.4 Safe Zone Behavior

While some parts of the behavior of an endangered agent, e.g. on-demand planning for a specified number of steps in advance and reserving the vertices for given times do not change once the agents enters a safe zone, the costs for different actions that WHCA* algorithm considers for each step are different and more dynamic.

The major difficulty in the safe zone is that freshly arriving agents must not impede the ongoing evacuation. A simple approach is to move agents as far from D as possible. However, knowing whether an agent is getting away from D is not a trivial problem from the local point of view.

The behavior agents adopt after entering S in LC-MAE (called *surfing*) is based on modified WHCA* algorithm. Costs for the passage from one vertex to another are computed dynamically, depending on the positions of other agents and the type of the agent's target vertex.

Algorithm 1: The algorithm for computing the number of agents following agent a [18]

```
1 previous-reserved ($\mathcal{E}$, $\pi$, $a$, $t$)
2     let $\pi = [c_0, c_1, ..., c_t]$
3     $V_a \leftarrow \{c_{t-b}(a) \mid b = 0, ..., \frac{l}{2}\}$
4     for $v \in V_a$ do
5         if reserved(v, t) then
6             $r \leftarrow r + 1$
7     return $r$
```

The basic idea is that an agent's priorities should vary according to the number of agents following behind, on the same path. When no other agents are following, it will prefer staying in place. With an increasing number of agents behind, the cost of staying in place also increases.

The agent determines the number of following agents by checking whether there are reservations created for positions it has passed before. The process is formalized in pseudo-code as Algorithm 1.

Since agents only make this check when they start planning their next few steps, the results have to be adjusted to account for the increasing uncertainty about the steps that other agents will take. This is done by subtracting the number of future time steps from the number of following agents (see line 11 of Algorithm 2).

Algorithm 2: Computing costs of different actions for agents in the safe zone. Actions are considered relative to agent a located at v at time t. t_c specifies the time at which the plan is generated [18].

```
neighbors ($\mathcal{E}$, $\pi$, $a$, $t$, $v$, $t_c$)
    let $\pi = [c_0, c_1, ..., c_t]$
    $A_p \leftarrow$ previous-reserved($\mathcal{E}$,$\pi$,$a$,$t_c$)
    costs $\leftarrow$ []
    foreach $u \in S \mid \{v,u\} \in E$ do
        if reservable($u$, $t+1$)$\wedge u \in S$ then
            if $u \in \{c_t(a) \mid t = 0,1,...,t\}$ then
                costs $\leftarrow$ costs $\cup\{(u,3)\}$
            else
                costs $\leftarrow$ costs $\cup\{(u,2)\}$
    $b \leftarrow \max(1, |A_p| - (t - t_c))$
    if reservable($v$, $t+1$) then
        costs $\leftarrow$ costs $\cup\{(u, 1*b)\}$
    else
        costs $\leftarrow$ costs $\cup\{(u, 4*b)\}$
    return costs
```

The complete cost-calculation algorithm can be found in Algorithm 2. With increasing back-pressure, moving into a reservable adjacent vertex becomes the cheapest option. The agent keeps a list of positions it has already visited and assigns them a higher cost. This leads to better diffusion of agents through S as endangered agents can ”nudge” safe agents deeper.

4 Centralized Evacuation Based on Network Flows

Near optimal solutions of MAE can be found by modeling an MAE instance as network flow [1, 26] and planning it centrally.

The core concept is to construct a time expanded network having m copies of G in which a flow of size $|A|$ exists if and only if the corresponding *relaxed MAE* has a solution with makespan m. In the relaxed MAE we do not require that agents only enter vacant vertices which is a condition hard to model in the context of network flows.

Only the requirement that there is at most one agent located on a vertex in a single time unit is kept. Hence train-like movements of agents are possible in the relaxed MAE. The illustration of *train like* movement and corresponding movement following the *move to unoccupied* rule is shown in figure 3 possible.

The process of construction of flow network G_m based on time expansion of the underlying graph G for m steps is described as follows:

- We add vertices z and s into G_m representing global source and global sink.

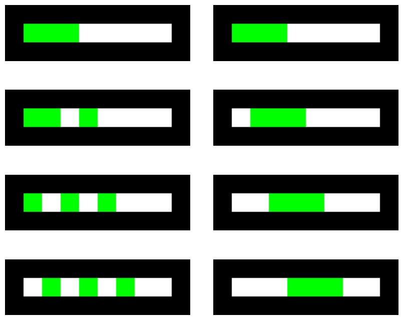


Fig. 3: Movement of agents in ordinary MAE (left) and in relaxed MAE (right) [18].

- For each vertex v_j from G we add into G_m vertices i_j^0 and o_j^0 and edge (i_j^0, o_j^0). For each j such that v_j contains an agent we add edge (z, i_j^0) into G_m. Intuitively this construction ensures interconnection with the source.
- For each $t \in 1, \dots, m-1$ and each vertex v_j from G we add into G_m vertices i_j^t and o_j^t connected by an edge. Moreover for each edge $e = \{v_x, v_y\} \in E$ we add into G_m edges $(o_x^{t-1}, i_y^t), (o_y^{t-1}, i_x^t)$.
- For each o_j^{m-1} such that v_j is a safe vertex in G we add edge (o_j^{m-1}, s) into G_m.
- Set the capacity of every edge in G_m to 1.

Finding the mininum makespan for the relaxed MAE can be done for example by the modified binary search algorithm as shown in Algorithm 3. The algorithm uses multiple yes/no queries about the existence of a solution for a specified number of steps to find the optimum.

A solution of a relaxed MAE problem generated by the network flow algorithms can be post-processed into a solution of ordinary MAE by postponing moves that would violate the invariant of not entering a vertex that has just been left by an agent. However, postponing moves may lead to deadlocks, so the post-processing algorithm swaps the paths planned for deadlocked agents when a deadlock is detected. We're calling this planning algorithm based on post-processed network flows *POST-MAE*; the idea of post processing follows the scheme from Figure 3.

5 Experimental Evaluation

We implemented LC-MAE in Python and evaluated it in multiple benchmark scenarios. We also implemented the POST-MAE algorithm on top of Push-relabel max-flow algorithm [9] that has been used to find the minimum makespan for the relaxed MAE.

In order to estimate the difference between the makespans of plans generated by LC-MAE and makespans of optimal (if completely unrealistic) plans, we also benchmarked POST-MAE without the post-processing, denoted as *flow* in comparison tables.

Our implementation relies on data structures implemented in the `networkx` library [10]. The visualization is implemented on top of the `arcade` library [5]. In the LC-MAE implementation, the look-ahead window was set to 10 steps.

Algorithm 3: Algorithm for finding minimum evacuation time.

```
minimum-makespan ($\mathcal{E}$)
    $f \leftarrow 0$
    $m \leftarrow |A|$
    $w_{hi} \leftarrow 0$
    while $f \neq |A|$ do
        $\mathcal{E}_e \leftarrow$ expand($\mathcal{E}, m$)
        $f \leftarrow$ maximum-flow($G_e$)
        if $f \neq |A|$ then
            $w_{hi} \leftarrow m$
            $m \leftarrow m * \frac{3}{2}$
    while true do
        $m_{new} \leftarrow w_{hi} + \lfloor (m - w_{hi})/2 \rfloor$
        $\mathcal{E}_e \leftarrow$ expand($\mathcal{E}, m_{new}$)
        $f \leftarrow$maximum-flow($G_e$)
        if $f = |A|$ then
            $m \leftarrow m_{new}$
            if $m = w_{hi} + 1$ then
                break
        else
            $w_{hi} = m_{new}$
            if $m_{new} = m - 1$ then
                break
    return m
```

5.1 Agent Types

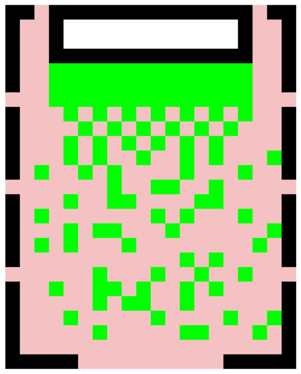
To simulate real-life scenarios with higher fidelity we used agents of two types. They differ in their behavior in D, while in S all agents rely on *surfing*. *Retargeting* agents fully implement the destination selection algorithm while *static* agents plan a path to a vertex specified in advance at scenario creation time.

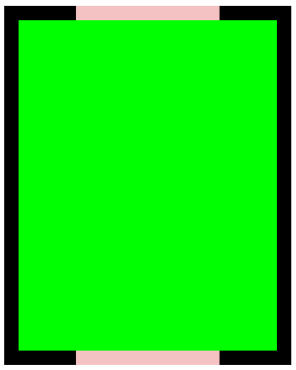
5.2 Setup of Experiments

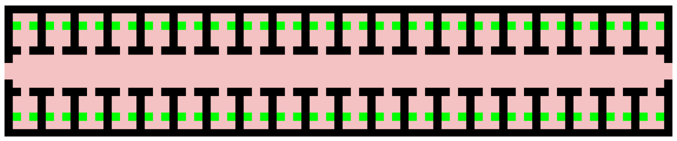
We used 4 different maps in our evaluations as shown in Figure 4 - they represent 4-connected grids with obstacles. Free and safe vertices are white (surrounding area), free and endangered vertices are pink. Vertices occupied by agents are green. Black squares signify walls, so no vertices are present in the underlying graph at those positions.

The respective test scenarios try to show evacuation on 3 realistic and 1 synthetic map:

- *Concert* (Figure 4a) representing a concert hall with an unevenly distributed crowd of 118 agents.


(a) Concert scenario


(b) Blocker scenario


(c) Office scenario

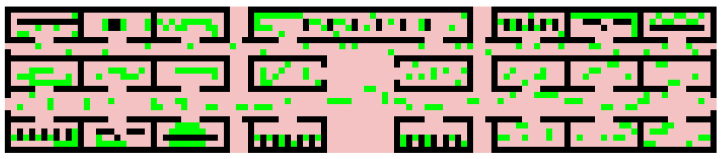

(d) Shops scenario

Fig. 4: Maps on which different evacuation scenarios were tested [18].

Scenario	Agents	LC-MAE	Flow	POST-MAE
Concert	118	90	17	33
Office	80	94	47	62
Shops	299	129	36	75
Blocker	414	146	23	69

Table 1: Makespans for evacuation plans [18].

- *Office* (Figure 4c) representing an office building corridor flanked on both sides by small offices. Exits are located on both ends of the corridor. There are 2 agents in each office, the corridor is empty.
- *Shops* (Figure 4d) representing a shopping center with complicated layout and many exits. 299 agents are present on the map, located both in the shops an on the corridors.
- *Blocker* (Figure 4b) an unrealistic map of a room with two emergency exits. It's completely filled with 414 agents.

5.3 Experimental Results

We first compared the makespan of evacuation plans generated by LC-MAE with optimal makespans calculated by the flow-based algorithm for relaxed MAE and with makespans of solutions post-processed with POST-MAE - see Table 1. LC-MAE generates plans that are only worse by a small constant factor (ranging from 1.52 to 2.73) than those generated by POST-MAE, which indicates that LC-MAE solutions are close to the true optimal makespan. Moreover, plans generated by LC-MAE are more realistic as they need local communication only.

The performance of LC-MAE is better than the performance of flow algorithm and than that of POST-MAE, as demonstrated in Table 2. An additional advantage is that since LC-MAE is a local algorithm and uses windowing, the plans can be used while they are being generated, since the steps taken by agents do not change.

5.4 Agent-based Simulations

We also performed a series of experiments to understand the real process of evacuation in scenarios in which various types of agents are mixed together, that is, when some agents are better informed than others.

Scenario	LC-MAE	Flow	POST-MAE	Speedup
Concert	7	52	62	8.9×
Office	4	57	61	15.3×
Shops	20	379	403	20.2×
Blocker	27	220	243	9.0×

Table 2: Seconds taken by plan generation [18].

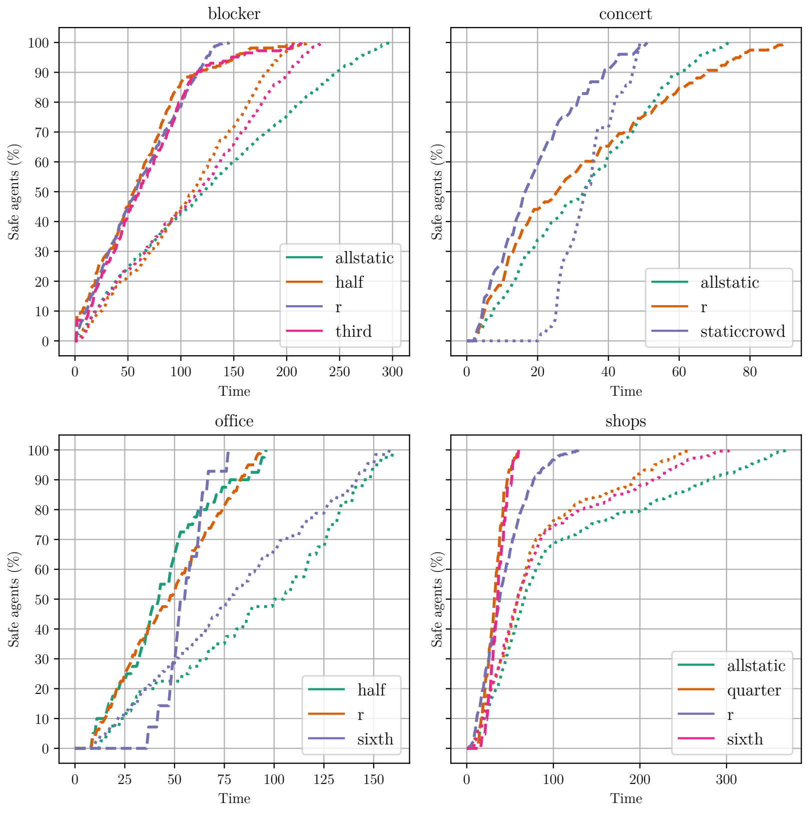


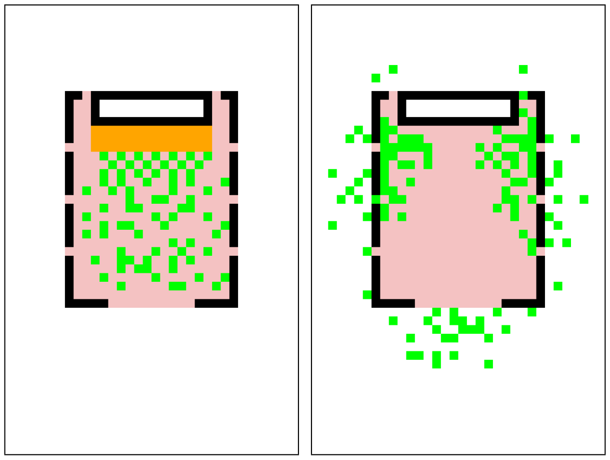
Fig. 5: The percentage of safe agents in time. Line colors differentiate between different scenarios. Dashed lines represent percentage of retargeting agents in *S*, dotted lines represent the percentage of static agents in *S* [18].

For each map, we created multiple scenarios with some of the retargeting agents replaced by static agents. These static agents try to exit through the largest opening between the safe and endangered zone (which could be described as the main exit from the area) and ignore all other exits. With this setup, retargeting agents could be considered to be better informed, given they take all the possible exits into account.

The percentage of agents that have reached safety as a function of time is shown in Figure 5.

5.5 Concert

The largest discrepancy between makespans of evacuations planned by POST-MAE and LC-MAE occurred on the Concert map. Our hypothesis was that small dimensions of side emergency exits and limited space in the safe zone behind them quickly caused congestion and hindered the evacuation, as can be seen in Figure 6b.



(a) *Static Crowd* scenario for the Concert map

(b) Congestion occuring in the *All Retargeting* scenario of the Concert map

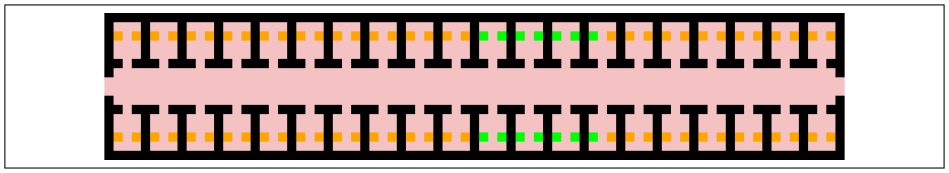
Fig. 6: Situations from the Concert map

To verify this hypothesis, we created two other scenarios, called *Static Crowd* and *All Static*. In *Static Crowd* there are 42 agents standing in front of the stage. Those agents try to escape through the main exit located on the bottom of the map (see Figure 6a). In the *All Static* scenario all agents use this exit.

Our hypothesis was confirmed (see Table 3). While a scenario with only retargeting agents has a makespan of 90 time units, when all agents are static, the makespan is only 74 time units. The shortest makespan, 51 time units, occurs in the *Static Crowd* scenario, since informed agents use side exits which don't get congested and free the space in the center of the map for the crowd escaping through the main exit.

	Retargeting Agents		**Static Agents**	
Scenario	**Count**	**Makespan**	**Count**	**Makespan**
All Retargeting	118	90	0	
Static Crowd	76	51	42	49
All Static	0		118	74

Table 3: Number of agents and evacuation makespan for different scenarios on the Concert map broken down by agent type


(a) The initial map state

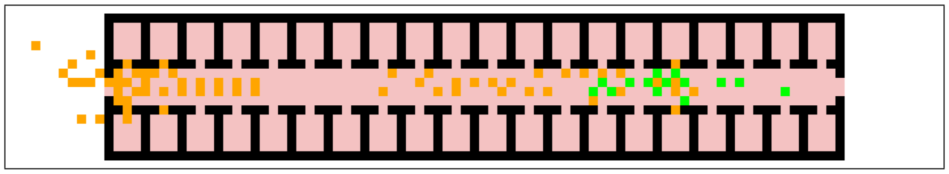

(b) Map state after the collision of retargeting and static agents in the corridor

Fig. 7: The *Sixth* scenario of the Offices map

5.6 Offices

For the Offices map, we created two modified scenarios, *Half* and *Sixth*. Their names indicate the fraction of agents which was left as retargeting. The rest of the agents is static, using the left exit to evacuate.

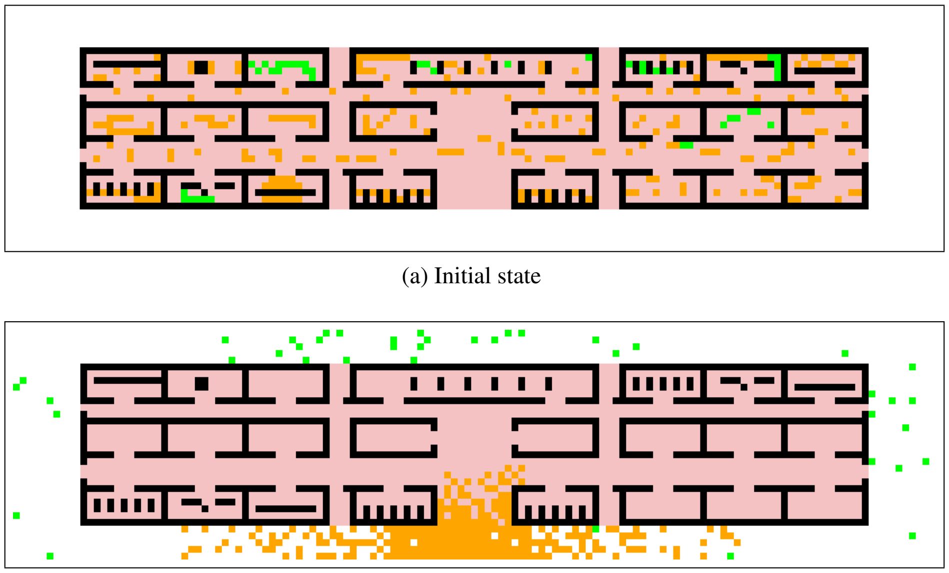
In *Sixth*, 14 retargeting agents are located in 7 columns to the right of the map center (see Figure 7a). We expected the retargeting agents heading right and static agents heading left to collide in the narrow corridor. This expectation was fulfilled. The evacuation of 14 informed agents took 77 time units, only 17 time units less than the evacuation of 80 retargeting agents in the original scenario. Both this collision and the congestion occurring at the left exit impeded the evacuation of static agents, causing their evacuation to take 158 time units. The gap in evacuation flow caused by the collision can be seen in Figure 7b.

In *Half*, there was one static and one retargeting agent in each office. In the makespan plot for this scenario, there is a significant slowdown around time unit 60, caused by a crowd forming in front of left exit and making both static and retargeting agents evacuate at the same rate (see Table 4).

	Retargeting Agents		**Static Agents**	
Scenario	**Count**	**Makespan**	**Count**	**Makespan**
All Retargeting	80	94	0	
Half	40	96	40	160
Sixth	14	77	66	158

Table 4: Number of agents and evacuation makespan for different scenarios on the Offices map broken down by agent type

5.7 Shopping Center



(a) Initial state

(b) Crowding in front of the main exit

Fig. 8: The *Sixth* scenario of the Offices map

For the Shopping Center map, we created 3 modified scenarios, called *Quarter*, *Sixth* and *All Static*, named in the same pattern as scenarios for the Offices map. Main exit, used by static agents, is located on the bottom of the map. The different types of agents are distributed randomly throughout the map.

The safe zone around the map is narrow so this scenario tests how the spread of agents between different exits influences the makespan. As can be seen on the plot the evacuation slows down around time unit 90 in all scenarios, because the area in front of the main exit gets filled and agents are not dispersing fast enough. This situation can be seen occurring in *Sixth* scenario in figure 8b. Makespan results are summarized in Table 5.

5.8 Blocking

The Blocking map is specific due to being an unrealistic map used to test agent behavior in a completely filled area. We created 3 modified scenarios, called *Quarter*, *Sixth* and *All Static*, named in the same pattern as scenarios for the Offices map. The exit, used by static agents, is located on the bottom of the map. In each row there is only one type of agents (see Figure 9a).

Due to map's regularity, the results are unsurprising. The evacuation in *All Static* has a makespan 2.06 times as long as evacuation using both exits (see Table 6). Safe

	Retargeting Agents		Static Agents	
Scenario	**Count**	**Makespan**	**Count**	**Makespan**
All Retargeting	299	129	0	
Quarter	75	61	224	257
Sixth	42	60	257	303
All Static	0		299	368

Table 5: Number of agents and evacuation makespan for different scenarios on the Shopping Center map broken down by agent type

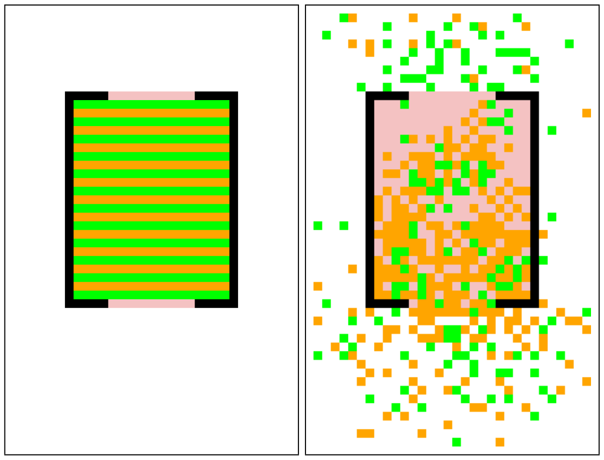

(a) *Half* scenario (b) Evacuation process during *Third* scenario

Fig. 9: Situations from *Blocking* map

zone saturation effects are similar to the Shopping Center map, being more pronounced for retargeting agents stuck in the crowd in front of the bottom exit.

An interesting situation can be seen in Figure 9b. Retargeting agents from the upper part of the map choose the upper exit and create reservations for their paths. These reservations block static agents trying to reach the bottom exit. Thus even some of the static agents use the upper exit to get to safety. On the other hand, some of retargeting agents, which get blocked by static agents, use the bottom exit, even though they are closer to the upper one.

6 Conclusion

We introduced an abstraction for evacuation problems called *multi-agent evacuation* (MAE) based on graph theoretical concepts similar to cooperative path finding. We suggested a new local algorithm called LC-MAE for solving MAE that produces solutions in which individual agents try to behave smartly during the evacuation process.

	Retargeting Agents		Static Agents	
Scenario	**Count**	**Makespan**	**Count**	**Makespan**
All Retargeting	414	146	0	
Half	216	207	198	219
Third	144	214	270	236
All Static	0		414	301

Table 6: Number of agents and evacuation makespan for different scenarios on the Blocking map broken down by agent type

LC-MAE uses a modification of WHCA* as the underlying path-finding process but also introduces several high-level procedures that guide agents' behaviour depending on whether they are in the endangered or the safe zone. We performed experimental evaluation with multiple scenarios including scenarios inspired by real-life evacuation cases as well as synthetic scenarios. The experimental evaluation indicates that LC-MAE generates solutions with makespan that is only a small factor worse than the optimum. We also studied how different ratios of less informed agents affect the process of evacuation. We found that depending on the scenario, the presence of some informed agents can improve the evacuation outcome for the whole group of agents. Additionally, even large numbers of uninformed agents don't impede informed agents from reaching the correct exit. For the future work we would like to continue with a framework for automated inference of simple local movement rules from solutions generated by optimal centralized evacuation algorithms. We expect that such rules could mimic the evacuation process produced by the centralized algorithm at the local level.

Acknowledgements

This research has been supported by GAČR - the Czech Science Foundation, grant registration number 19-17966S. We would like to thank anonymous reviewers for their valuable comments.

References

1. Arbib, C., Muccini, H., Moghaddam, M.T.: Applying a network flow model to quick and safe evacuation of people from a building: a real case. In: Proceedings of the GEOSAFE Workshop on Robust Solutions for Fire Fighting, RSFF 2018, L'Aquila, Italy, July 19-20, 2018. pp. 50–61 (2018)
2. Bompadre, A., Orlin, J.B.: A simple method for improving the primal simplex method for the multicommodity flow problem. Networks **51**(1), 63–77 (2008)
3. Cacchiani, V., Jünger, M., Liers, F., Lodi, A., Schmidt, D.R.: Single-commodity robust network design with finite and hose demand sets. Math. Program. **157**(1), 297–342 (2016)
4. Chalmet, L.G., Francis, R.L., Saunders, P.B.: Network models for building evacuation. Fire Technology **18**(1), 90–113 (Feb 1982)
5. Craven, P.V.: The python arcade library. `http://arcade.academy` (03 2019)

6. Felner, A., Li, J., Boyarski, E., Ma, H., Cohen, L., Kumar, T.K.S., Koenig, S.: Adding heuristics to conflict-based search for multi-agent path finding. In: Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018. pp. 83–87 (2018)
7. Foudil, C., Djedi, N., Sanza, C., Duthen, Y.: Path finding and collision avoidance in crowd simulation. CIT **17**, 217–228 (2009)
8. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning and Acting. Cambridge University Press (2016), `http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB`
9. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum-flow problem. J. ACM **35**(4), 921–940 (1988). https://doi.org/10.1145/48014.61051, `https://doi.org/10.1145/48014.61051`
10. Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using networkx. In: Proceedings of the 7th Python in Science Conference. pp. 11 – 15 (2008)
11. Hudziak, M., Pozniak-Koszalka, I., Koszalka, L., Kasprzak, A.: Comparison of algorithms for multi-agent pathfinding in crowded environment. In: Nguyen, N.T., Trawiński, B., Kosala, R. (eds.) Intelligent Information and Database Systems. pp. 229–238. Springer International Publishing (2015)
12. Korf, R.E., Taylor, L.A.: Finding optimal solutions to the twenty-four puzzle. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2. pp. 1202–1207 (1996)
13. Kurdi, H.A., Al-Megren, S., Althunyan, R., Almulifi, A.: Effect of exit placement on evacuation plans. European Journal of Operational Research **269**(2), 749–759 (2018)
14. Liu, C., li Mao, Z., min Fu, Z.: Emergency evacuation model and algorithm in the building with several exits. Procedia Engineering **135**, 12 – 18 (2016), 2015 International Conference on Performance-based Fire and Fire Protection Engineering (ICPFFPE 2015)
15. Liu, Z., Wu, B., Lin, H.: Coordinated robot-assisted human crowd evacuation. In: 57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018. pp. 4481–4486 (2018)
16. Mishra, G., Mazumdar, S., Pal, A.: Improved algorithms for the evacuation route planning problem. In: Combinatorial Optimization and Applications. pp. 3–19. Springer International Publishing (2015)
17. Sánchez-Oro, J., Duarte, A.: An experimental comparison of variable neighborhood search variants for the minimization of the vertex-cut in layout problems. Electronic Notes in Discrete Mathematics **39**, 59–66 (2012)
18. Selvek, R., Surynek, P.: Engineering smart behavior in evacuation planning using local cooperative path finding algorithms and agent-based simulations. In: Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2019, Volume 2: KEOD, Vienna, Austria, September 17-19, 2019. pp. 137–143 (2019)
19. Sharma, S.: AVATARSIM: A multi-agent system for emergency evacuation simulation. In: 17th International Conference on Software Engineering and Data Engineering (SEDE-2008), June 30 - July 2, 2008, Omni Los Angeles Hotel at California Plaza, Los Angeles, California, USA, Proceedings. pp. 163–167 (2008)
20. Shekhar, S., Yang, K., Gunturi, V.M.V., Manikonda, L., Oliver, D., Zhou, X., George, B., Kim, S., Wolff, J.M.R., Lu, Q.: Experiences with evacuation route planning algorithms. International Journal of Geographical Information Science **26**(12), 2253–2265 (2012)

21. Silver, D.: Cooperative pathfinding. In: Young, R.M., Laird, J.E. (eds.) Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA. pp. 117–122. AAAI Press (2005)
22. Silver, D.: Cooperative pathfinding. In: AI Game Programming Wisdom 3 (2006)
23. Surynek, P.: Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In: IJCAI. pp. 1916–1922 (2015)
24. Surynek, P.: Solving abstract cooperative path-finding in densely populated environments. Computational Intelligence **30**(2), 402–450 (2014)
25. Wang, K., Botea, A.: MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. JAIR **42**, 55–90 (2011)
26. Yu, J., LaValle, S.M.: Multi-agent path planning and network flow. In: Algorithmic Foundations of Robotics X - Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics, WAFR 2012. pp. 157–173 (2012)
27. Yu, J., LaValle, S.M.: Optimal multi-robot path planning on graphs: Structure and computational complexity. CoRR **abs/1507.03289** (2015), `http://arxiv.org/abs/1507.03289`