BioPlan: An API for Classical Planning on BioCrowds

Maurcio C. Magnaguagno
Pontifcia Universidade Catlica
do Rio Grande do Sul (PUCRS)
Email: mauricio.magnagugno@acad.pucrs.br

Felipe Meneguzzi
Pontifcia Universidade Catlica
do Rio Grande do Sul (PUCRS)
Email: felipe.meneguzzi@pucrs.br

Abstract

Crowd simulation for evacuation situations often assumes that all agents are trying to reach a single point within an environment. Although such an assumption is not entirely wrong, human agents often exhibit more complex behaviors, even if deviations from the standard behavior are not particularly frequent. Classical planning is far from the best way to achieve the minimal path or correct behavior for agents, but adds a deeper level of reasoning about complex goal-achievement and about actions that are more complex than simply moving about. In this paper, we describe a crowd simulation experiment that uses classical AI planning to enrich the behavior of the agents in the scenario. Using this approach, we can express not only the target destination of the agents, but also (sub)goals and path preferences.

Keywords:: AI Planning, Crowd, Simulation

Author's Contact:

1 Introduction

Computer games and simulations are often concerned with virtual crowds to populate their simulated environments, with each specific application focusing on different concerns. Whereas games are generally concerned with computational efficiency aimed at achieving a responsive experience, even at the cost of some of its realism, simulations are concerned with realistic virtual crowds that respond in a way that is compatible with real humans. Common to all of these applications, the most basic problem to be solved is to make individual agents navigate through a set of waypoints that are either dynamically generated or previously established by a designer. However, when an agent has more complex goals than simply arriving at one or more destinations, the problem becomes one in which an agent needs to specify additional (sub)goals which need to be exhibited in the simulation. To achieve these types of goal without having to specify, at design-time, exactly the actions taken by an agent, a planner is often required. In fact, AI planners have been extensively used to compute the behavior of individual agents in computer games [Orkin 2006]. However, computing individual plans for large numbers of agents with many possible actions is prohibitively time-consuming, which leads to the issue of using planning algorithms efficiently to generate behavior for several similar agents.

In this paper, we describe a crowd simulation approach that uses classical AI planning to enrich the behavior of the agents in the scenario through a path influenced by agent desires shared by groups of agents. Using this approach, we can express not only the target destination for the agents, but also (sub)goals as desired states to be reached, including parts of the map as path preferences. Our approach consists of converting agent preferences to a classical planning problem and employing a classical planner before the simulation to generate paths with subgoals for agents within a crowd to follow once the simulation starts. Although deterministic plans from classical planners are often not suitable for generating behavior in complex spatial environments, we use the higher level of abstraction of such plans in a way that allows rich crowd behaviour without a substantial addition in complexity to the simulator. We achieve this by letting the collision avoidance mechanism already present in the simulation add uncertainty to the resulting plan-driven behavior of the agents. This makes the solutions more flexible, changing the initial position of a group can render an entire different path, with agents always looking for alternative ways to reach their desires while maintaining a least-effort path.

Traditional approaches to crowd simulation have avoided the issue of planning global behaviors more complex than moving agents from one point in a map to another, thus focusing reasoning only on the path itself [Sud et al. 2007]. This type of reasoning prevents agents from employing actions more complex than movement and pursuing desired states more complex than map positions. Such a choice is motivated by the fine-grained representation of the simulated maps, which avoids discretizing the space into a coarse grid to achieve a realistic representation of individual agent movement. Using such a state representation, reasoning about agent desired-states becomes infeasible, since scalability issues arise once thousands of agents have a single location as their desired destination. Treuille et al. [Treuille et al. 2006] points out that global planning with local collision avoidance may lead to unrealistic situations such of large crowd concentrations with agents feeding into the congested mass of agents, whereas real humans would try to avoid the crowded region before getting stuck in it rather than blindly following their path and ending up stuck. In response, the authors of *Continuum Crowds* have developed a method to plan globally so that no agent ever gets stuck in the environment. Alternatively, Li et al. [Li et al. 2001] try to achieve complex behavior while minimizing the global planning effort by employing the notion of a group leader that performs complex reasoning, while a crowd of agents conceptually follows this leader through a series of checkpoints as they move through the scenario. In this setting, the leader not only plans for itself, but also needs to make sure that its behavior includes tolerances so that the crowds that follow it do not get stuck. By contrast, our approach aims to use a classical planner to generate plans for a group of agents in a single execution, generating a single plan that is executed by each individual agent in a group and letting the existing collision avoidance algorithms deal with the details of movement at runtime. Although the use of planning capabilities to expand behavior has been used previously as a cognitive layer for animated characters [Funge et al. 1999], to the best of our knowledge, its efficient use to drive behavior in crowd simulation is novel.

Thus, our main contribution is an approach that uses classical planning to efficiently enrich a traditional crowd simulation model with the notion of declarative goals while maintaining a substantial degree of scalability. We demonstrate the applicability of our approach through an implementation based on the BioCrowds simulator [de Lima Bicho 2009] and a classical planner that does STRIPS-style planning [Fikes and Nilsson 1971] using heuristic search [Bonet and Geffner 2001]. The resulting system simplifies the effort of crowd design and control while speeding up the computation of multiple group paths by automatically generating the data required by the planner whenever new paths are required.

2 Background

Path planning[1] algorithms are commonly used within simulations to compute paths from a starting position to a goal position for agents to use according to their behaviors. Thus, in this section we review relevant prior work on path planning and agent behavior. One of the key aspects of a crowd simulation is the way in which agents move through the simulated space and interact with each other when collisions may occur. One of the first crowd sim-

[1] Path planning is often referred to simply as planning in crowd simulation literature as in [Treuille et al. 2006], in this paper, to avoid confusion, we shall differentiate path planning from AI planning (or simply planning).

ulation systems, Boids [Reynolds 1987], uses a behaviour model based on simple rules to generate actions based on agent perception. These rules controlled attraction and angle of movement of bird-like creatures and later evolved to more human-like features [Flach et al. 2013]. By contrast, we focus on the model used by the BioCrowds simulator [de Lima Bicho 2009], which is based on a biological phenomenon, comparing the growth of veins in leaves to social interaction the idea of competition for space and creation of trajectory guides the behavior of agents.

Although guiding an agent's path using predetermined tasks (sequences of actions to be executed) is not new in BioCrowds [Flach et al. 2013], its agents' behavior is limited to a single destination during the simulation. The model proposed by [Flach et al. 2013] selects a random action to be performed and a path planning algorithm computes a path to the point where the action is possible. Thus, BioCrowds agents lack the ability to reason about higher-level goals (world-states to be achieved) and multiple goals and subgoals. In order to address this limitation, we aim to use a more advanced reasoning mechanism, whereby agents have a desired world-state and use a domain-independent planning algorithm to compute a sequence of complex actions in order to transition from the current world-state into the desired world-state. Here, a world-state may refer not only to a position in the environment, but also other, more abstract goals such as avoiding a certain position. Planning algorithms search using a specification of the environment dynamics using transition rules described in a flexible formal language as well as a specification of the problem to be solved [Ghallab et al. 2004]. This allows very different problems and domains to be solved by the same, efficient, algorithm. On the other hand, path finding problems are usually solved by tailored search algorithms, which generate a plan with a good response time but much less flexibility. For example, adding keys and closed doors to an environment requires the reconstruction of the entire search algorithm instead of a minimal reformulation of the domain to add actions to unlock the door. The key point of planning is reusability, and planning research has yielded a number of formal languages, such as the well known *Stanford Research Institute Problem Solver* (STRIPS) language [Fikes and Nilsson 1971], and more recent the formalizations of the *Planning Domain Definition Language* (PDDL), which is the standard planning language [McDermott et al. 1998] for the ICAPS competition [Coles et al. 2012]. This is analogous to the notions of procedural and declarative goals known in the autonomous agents literature [Winikoff et al. 2002; Meneguzzi and De Silva 2013], which we borrow in this work. The usual approach is to use a procedural goal, where a predetermined procedure once executed successfully will achieve the goal. Our approach is the declarative goal, where the state desired is declared and the plan is not readily available as a procedure, being required a classical planner to find the sequence that achieve the goal-state.

3 Crowds

Several approaches have been used to model crowd behavior, the most influential models being based on flocking behavior [Reynolds 1987], sociological factors [Musse and Thalmann 1997], psychological effects [Pelechano et al. 2005], geographically-based direction [Sung et al. 2004] and social forces [Helbing and Molnar 1995]. The models are usually concerned with local problems while some global path planning is used to compute plans to reach an agent's goals. The flocking model [Reynolds 1987], which underlies all of these approaches, is a particle system with movement based on rules, called steering behaviors, use to create a consistent and fluid movement using local perception. All of these approach use the following three fundamental steering behavior rules:

- **Separation**: avoid crowding near particles, particles try to keep safe from collision in dense regions;
- **Alignment**: follow the average heading angle of near particles to maintain itself as part of the crowd going to the same place; and
- **Cohesion**: move towards the center of near particles, keep particles together.

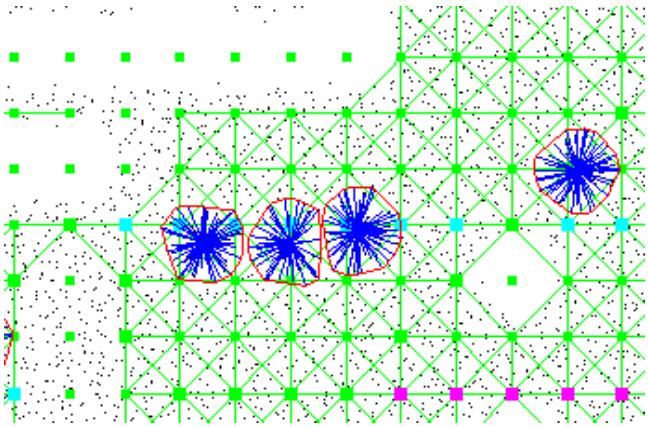

Figure 1: *BioCrowds*

This basic set of rules can be expanded to create realism for specific situations and recreates the idea of unlimited population for the crowd, as flocks of birds emerge naturally with each bird sensing only the birds immediately around itself.

3.1 BioCrowds

The BioCrowds model is based on a biological phenomenon that happens in leaves, where the leaf veins are attracted to the auxin hormone, resulting in particular growth patterns. Different parameters of this distribution and attraction make different veins possible, giving each plant a unique pattern. Since the veins compete for the auxins, the model resembles a crowd where each agent competes for more individual space, aiming to achieve a collision-avoidance system between agents in a dense crowd. To test the model, the behaviors of crowds being simulated must be checked at runtime, to ensure the following patterns are present:

- **lane formation**: as agents move, the free space behind them becomes attractive for the other agents that are going in the same direction, inducing several agents to use such spaces as a collision-free path, limited only by the speed of the agents ahead;
- **arch formation**: the flow of agents around an obstacle creates an arch around the obstacle as agents avoid colliding with it — arches can be seen as a special case of a lane;
- **speed based on density**: as more agents occupy the same region their speed is affected to avoid collision with other agents;
- **bottlenecks at small spaces**: as narrow paths are used by many agents, the competition for the passage causes agents to actually wait for the ones ahead of them to free the passage — bottlenecks may appear without walls around them if a resource is the destination of several agents; and
- **divergence**: after passing through a bottleneck, agents usually follow different paths, creating the inverse of bottlenecks, as no obstacles or agents are there to slow down their movement.

Those patterns did emerge in BioCrowds, proving that the model actually resembles the human interaction that occur in crowded spaces [Solmaz et al. 2012]. Agents compete for fixed markers (such as the auxins) distributed randomly across the free space, allocating and freeing these markers as they move through the space, allowing other agents to occupy their previous space. The amount of markers allocated around each agent is based on their requirement for personal space. Such a way of reasoning about space is called proxemics, which define different types of relation agents may have based on distance to other elements in the environment. Proxemics dictate the distance agents try to keep from each other, simplifying the obstacle reasoning the agents must do, as positions without free markers are seen as impossible to occupy.

4 Classical AI Planning

Classical planning is an area of AI concerned with creating algorithms to solve problems defined with a generic formal language that treat environment states as sets of discrete variables [Ghallab et al. 2004]. A classical planner is usually based on a search algorithm that tries to find a sequence of actions (formally defined as a plan) that, when executed, modifies the initial state of the world into the desired goal state. This plan is the sequence of intermediary points of the usual A* for path planning, but instead of points, the effects of each action yield the desired state when executed in order and successfully. Below, we summarize the key concepts concepts in AI planning required for our work:

- a **state** is a structure containing world properties true at a particular point in time;
- **free-variables** are values that can be any object from the problem, will be substituted to ground actions;
- a **predicate** is a named property of the world with any number of terms, each term can be a free-variable or an object;
- **objects** are explicitly or implicitly defined by the problem — once defined a problem's actions can be expanded into its possible instances;
- a **proposition** is a named property of the world with any number of terms, each term is an object;
- an **action** or **operator** is part of a domain's transition function that can be applied to the current state, it is specified in terms of preconditions and effects (expressed as logical formulas over predicates);
- the **preconditions** of an action is set of predicates (or a free-form logical formula) that must be true in the current state for an action to be applicable (executable) in that state;
- the **effects** of an action are a set of predicates that will be added or deleted from the current state, creating a new state;
- a **domain** describes the key elements of a planning domain, comprising the set of valid predicates (properties of the world) and the actions (transition function) available in the domain;
- a **problem** is a specific instance of the domain to be solved, with a set of objects, the entire initial state and the goal state that must be reached; and
- a **solution or plan** is a finite sequence of operators available to the domain that when applied to the initial state satisfy the goal state [Nebel 2011].

Some problems may have no solution (i.e. no plan exists that can transform the initial state into the goal state), while some problems may have multiple solutions. A plan is said to be optimal if it is the shortest plan that achieves the goal. To take advantage of the possibility of multiple plans for a given problem, some planners relax optimality constraints to speed-up search and find a suboptimal plan. Note that the sequence of actions may be represented in a tree-like structure to better describe dependency and order between actions, although the linear idea of sequence is used to simplify the relationship of preconditions and effects.

4.1 Planning Languages

The formal elements we describe in this section are used in the definition of planning languages, the first of which was implemented for *Stanford Research Institute Problem Solver* (STRIPS) system. STRIPS is a planning system created in 1971 [Fikes and Nilsson 1971] that became important due to its formalization of the description of the world, providing much of the structure for planning problem specifications we describe above. In order to specify more complex domains, later planners defined a series of extensions and planning languages. These languages were consolidated into the *Planning Domain Definition Language* (PDDL)[McDermott et al. 1998], created in 1998 to be the standard language for AI planning.

4.2 Classical Planning Algorithms

Different algorithms solve planning problems using different approaches on how to deal with the several combinations of action sequences that yield a solution. The most straightforward planning mechanism consists of a forward search in the state-space, checking which actions can be applied to the current state generating further states until the goal state is found. For much of the evolution of planning algorithms, forward search did not yield efficient results, due to the need for very good heuristics to avoid the large branching factor inherent to this type of planning. However, later research has shown that such heuristics are possible for very efficient planning [Bonet and Geffner 2001]. In this paper, we use a heuristic search-based planning algorithm.

5 BioPlan: A classical planning extension to BioCrowds

BioCrowds is both a model (see Section 3.1) and simulator that simulates interacting agents based on fixed markers in space. Agents move through the environment trying to reach a sequence of 'waypoints', which must be manually adjusted by a human designer. In order to reach their destinations, agents move by trying to occupy free markers around them, without differentiating whether the lack of free adjacent markers is due to walls, or the transient occupation of a marker by another agent. As each agent tries to occupy the markers in the direction of the next goal position, a problem arises when there is not enough intermediary destinations to guide them, which, as a result, become stuck not knowing whether to try alternative paths or to wait for a nearby marker to become free. Planning becomes interesting at this point, to find a path between the current position and one of many goal positions defined as declarative goals. The use of declarative goals frees a designer to simply specify what the goals are (be they positions in the map, or other, abstract states), and let the planning algorithm generate the intermediary positions in the map, as well as the positions that are to be avoided to prevent agents getting stuck. Classical planning on the other hand costs too much processing power for a group of agents and also removes freedom as the plan gives a description of each step, but has the flexibility of a declarative goal state. If we describe a goal state and how to perform actions to reach this state it is possible to remove some intermediary points to add freedom. Moving from a desired point to a desired state means we do not want to occupy a position, but what is there.

In order to plan reasonably fast, agents are grouped based on similar initial and desired states. All agents share the same map with the same action-points. Action-points are specific places where agents can achieve a certain abstract feature. Groups may consist of a single agent or hundreds of agents. Since multiple agents may start in the same initial state and share goals, we can group agents that share the same initial and goal configuration, and plan once for several agents. Agent groups have a set of shared attributes for path planning that are inherited by all agents in a group, but some attributes are randomized per agent, e.g. speed. Such different path planning attributes naturally lead to the actual paths taken by the agent being different even for agents in the same group, as speed and collision may create a different local scenarios for each agent. Although there are many more attributes, we summarize in Table 1 those attributes that are relevant for our model.

After the groups were described the next important element in a simulation is the map. The map is important for both planning and simulation, as obstacles define the environment and action-points define the different ways agents may achieve a certain feature. The map is considered static and described in a discrete form to the planner, making the goal of planning the change of the agents internal state instead of the world itself. Changing the map is usually a good idea for coordinated groups, as agents plan to act at specific places to reach the same goal, but becomes a problem for our crowd model. Imagine the goal of a group to be behind a fireproof door during an evacuation, the first agent to reach the door must open the door, everyone enters the safe room and the door is then closed by one them. This is a coordinated action, the problem can be broken into smaller pieces (open, enter, close) and not every agent needs to

Table 1: *Group attributes*

Attribute	Possible values	Description
agents	1 to ∞	Amount of agents from group
route	Array of nodes	Source and target destination
path mode	direct, A*, classical planning	Path generation method
freedom	1 to ∞	Amount of edges between nodes of path
start	Array of requirements	Initial state of group
goal	Array of requirements	Desire state of group
speed	Float	Agent maximum speed without obstacles
proxemics	Float	How near a marker must be to be used by the agent
color	RGB	The color used by agents, path and marker's connector

execute all those actions. If implemented by our planner all agents would have to close the door behind them as no agent knows if they are the last one to reach the door. An inconsistency happens if two agents reach the door at the same time, only one would be able to open it. This exemplifies very well the crowd planning problem, the agents do not recognize being part of a group and do what they need to satisfy their needs, even if all agents have the same goals. The second problem is the single entity approach, in order to make the last agent close the door, each agent would need to plan individually and one would need to be the designated at the agent responsible for closing the door. Thus, we concentrate on a static map as the simulator cannot display mutable features of the environment, such as doors being open or closed.

Our model is based on planning paths for all agent groups before the simulation starts. Instead of planning for the N groups at the same time, making the problem more complex for a large N, our model breaks the problem down for each group, making the simulation not only simpler but also more realistic. The simulation is more realistic because, in the real world, agents follow a path and react to other individuals as obstacles that alter their path locally. Therefore the behavior of the agent and not the plan is important locally. Our approach is also simpler, because taking into account all groups at the same time would yield a plan that represents the position of the groups in a discretized and deterministic way, which is not true at simulation time as collision avoidance and speed make the agents diverge locally from the optimal path to keep moving.

The desired state can involve several propositions beyond the position, with several being dependent and possible to achieve in different ways according to the initial state and actions available. Unlike path-planning, an agent using our planner may desire to explicitly avoid a certain property, such as starting dirty and desiring to be clean (not dirty). Most planners are tailored to deal with positive only literals, but to achieve full flexibility, both positive and negative literals must be supported. Once the positions inside the range of the bomb are identified the desired state is to stay away enough. Even if the agent tries to maximize this distance the effect is the same inside a game, and a plan long enough means a less processing than an extremely long plan with the agent running as far as possible, probably until reaching the border of the map.

As the simulation is executed, the agents occupy positions with different speeds (that are not represented by the planner) based on the random distribution of the markers and individual speed settings. Blindly following the plan to avoid other groups at the same time is not enough because of those non-deterministic attributes. If we used a planner in which those details were explicitly represented, the resulting plans would eliminate the realism of the situations created by this randomness, as agents would re-plan only if a large number of agents completely blocked their passage to their destination. Replanning would also create a processing bottleneck as only individuals require replanning based on their current situation. Therefore the simulator received few modifications to accept input from our planner.

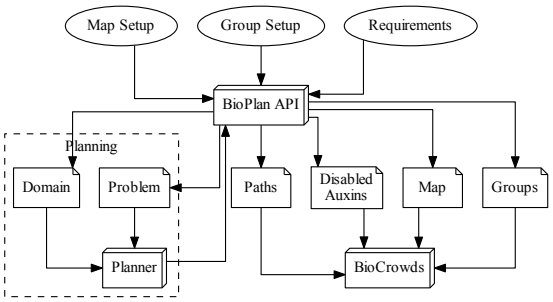

Figure 2: *Flow of the proposed model*

Adding the output of a classical planner as input for the simulator in a seamless way requires some constraints on the output of the planner to be interpreted correctly by the simulator, since the simulator expects a path and not a plan as input a conversion is required. The planner requires the map of the environment as much as the simulator and a conversion process was required to automate the generation of group attributes, adjacency information of the map, how actions interact with the world-state and how agents execute the plan or deal with a failure in planning. In other words a unique input was required to describe both planner and simulator inputs.

6 Implementation

In our implementation we modify a Ruby implementation of a search-based classical planner in such a way as to focus on a single group at a time, instead of generating several domain/problem inputs for each group, certain specific points of the planner were modified. Planning languages describe operators in terms of action schemata with variables that can be substituted for objects in the domain using unification, generating concrete (ground) operators. Thus, a single action schema may result in a very large number of operators, leading to a large branching factor. Instead of planning for every group at the same time the system focuses on each group, which means more plans, but each one with less steps. Search-based planners have to deal with problems with large branching factors due to the large number of possible concrete operators, by focusing on each group, we minimize the branching factor. Moreover, once concrete operators are generated and the planner knows the exact size of the state representation, we generate an internal binary-vector based representation that allows for very efficient planning both in terms of runtime and memory use. This approach speeds up the planning process and makes it easier to debug problem domains.

We integrate all of these systems in a processing pipeline, illustrated in Figure 2. Each subsystem (the classical planner and the simulator) requires its data to be converted before it can commnunicate with the other. To address that, we implemented the BioPlan API, which handles all translation processes. The designer must specify map, groups and transition rules to be used by the planner as in code 1. Actions for moving throughout the space are statically defined and reused in all simulation domains, and additional rules for specific simulation requirements are translated into additional planning operators. These rules are then used to create valid PDDL actions.

The groups are defined based on the number of agents, route (start/-goal), path mode (direct, A*, planning) and when planning is used, a degree of freedom and a set of requirements to fulfill along the path. This degree of freedom can be used to avoid an unnatural movement by the agents, removing intermediary movement-only actions among the plan. Once the map is consistent, it is possible to extract information about which nodes can be accessed. There are

three types of nodes so far and only clear nodes can be accessed, wall nodes are always closed and avoidance nodes are always considered closed for planning but clear for the simulator. This information is used to avoid creating markers at specific regions in the simulator. The map is also analyzed to find the adjacency between the nodes, creating the edges to be used by the planner. The requirements and some of the group attributes are part of the problem as initial or goal state while the domain is a static file with the movement and a generic action that can be performed based on the requirements previously defined. The planner is executed for each group defining planning as operating mode and if successful the plan is modified according to the freedom of the group in question.

```
require 'BioPlan'

group_0 = {
  :agents => 15,
  :route => [839, 666],
  :mode => BioPlan::PLANNING,
  :freedom => 3,
  :start => ['null'],
  :goal => ['receive_food']
}

group_1 = {
  :agents => 7,
  :route => [839, 540],
  :mode => Bioplan::PLANNING,
  :start => ['null','use_bathroom'],
  :goal => ['receive_food']
}

group_2 = {
  :agents => 5,
  :route => [839, 540],
  :mode => BioPlan::ASTAR
}
groups = [group_0, group_1]

requirements = [
  {
   :at => 968,
   :require => 'null',
   :able => 'use_bathroom'
  },
  {
   :at => 440,
   :require => 'use_bathroom',
   :able => 'pay_food'
  },
  {
   :at => 165,
   :require => 'pay_food',
   :able => 'receive_food'
  }
]
map = Image.load_bmp('map1.bmp')

BioPlan.setup(groups, map, requirements)
```

Code 1: *API*

6.1 Domain Knowledge

Domain knowledge is a series of small optimizations and tricks about a specific domain that can yield a better or faster solution when used. Several problems have to deal with the fact that the domain knowledge is incomplete or non-existent. The map with walls/obstacles and starting and goal points is not enough to have a good plan. Most problems faced by the BioCrowds model come from collision avoidance from narrow paths and agents being thrown away from their route by other agents. Adding more input to the map can yield a better path. Some of those problems are explained in this section with more detail.

Algorithm 1 BioPlan setup

```
1:  procedure BioPlan::setup(groups, map, requirements)
2:      clean map
3:      mark disabled map nodes
4:      save marked nodes to file
5:      add requirements to initial state
6:      for g in groups do
7:          add current state of g to initial state
8:          add goals of g to goal state
9:      end for
10:     extract graph from map
11:     add adjacencies to initial state
12:     save map to file
13:     save problem to file
14:     for g in groups do
15:         if mode g is PLANNING then
16:             plan ← planner(domain, problem, g)
17:             if plan found then
18:                 path ← []
19:                 for action in plan with index i do
20:                     if action = move and i % g.freedom = 0 then
21:                         path push position
22:                     else if action = do then
23:                         path push position
24:                     end if
25:                 end for
26:                 save path to file
27:             else
28:                 print warning for g
29:                 Downgrade g to path planning
30:             end if
31:         end if
32:     end for
33:     save groups to file
34: end procedure
```

Figure 3: *Regions to avoid during planning*

6.1.1 Map Contour

Without a predefined path, agents tend to use a movement pattern similar to a best-first search, thus approaching the goal position without taking into consideration the contour of the map. This usually results in failure (being stuck) if the distance is too long, as more walls may appear in the way. The failure persists even with a predefined path, with agents becoming trapped by an irregular wall. To actually avoid passing near those walls and becoming stuck in a local minima, additional knowledge of places to avoid is added to the map. This knowledge is used by the planner to both speed-up (eliminate nodes from the search) and leave those nodes as clear nodes during simulation and letting agents use them as a last resource when more individual space is needed. Several tricks can be used, such as: aiming to pass through a door targeting exactly its midpoint; avoiding walking too close to walls in passages to avoid being surprised by other agents; and avoiding traversing queues of waiting agents, but rather pass behind the queues. Some of those examples can be seen in Figure 3 in which gray squares represent the regions to avoid and black squares represent the walls.

The map is defined by nodes, nodes can be clear, walls and avoidance. Avoidance node certain are useful not only to remove certain paths from consideration, but also to provide much more control

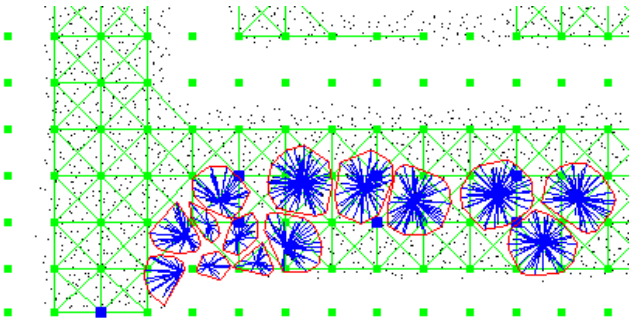

Figure 4: *Bathroom bottleneck*

over the resulting paths, something any designer want. Constraining where desired and letting the group free otherwise. The map walls represent places where no auxins exists, creating a lack of a possible path for agents. The terrain can be seen at Figure 11, with a zoom in the upper left corner, where green squares are the nodes and the black dots the auxins. Connected nodes are possible paths and auxins represent free space that agents use while moving to conquer more individual space in their goal direction. The avoidance system was inspired by the preference system of [Cassol et al. 2011], but instead of being part of the behavior it became part of the planning process.

6.1.2 Perception and choice

One of the main problems is how much freedom the agents can have. The idea of group here would be broken if some agents took a different path. In some situations this makes sense, like in a bathroom choice according to the size of the waiting line, but the problem persists not on how the agents would perceive this, but how effective is to let them free to explore based solely on perceptions or use some forks in the plan. Instead of using a reasoning system during run time the idea is to let each agent have more freedom without forgetting the path, removing some intermediary points. The simple removal of intermediary points can lead to failure in some cases, identify those cases is important to understand how free the agent can be while staying close to the path in narrow passages. One of the main problems that removing intermediary points is not enough is the bottleneck that happens when agents achieve one of their goals and suddenly start coming back. This can be seen in the bathroom situation, Figure 4, as the agents reach the bathroom they start coming back and compete for their previous space, breaking the lane formation created by the BioCrowds model. Some agents will try to go to the free side while others make the bad choice and end up stuck until most agents find their way to the bathroom.

6.1.3 Heuristic

Most problems can be solved faster using a relaxation strategy to focus in the relevant attributes. Working with this subset of the actual problem may lead to a sub-optimal solution with less resources used. Finding the heuristic to a specific problem is a problem by itself and since a classical planner have no idea which domains may be presented as input a specific heuristic for the problem cannot be used. Methods to relax the problem definition are used. One of the most basic heuristics is the Hamming distance [Hamming 1950], since the problem is completely observable it is possible to compare the current state with the desired state and count how many propositions (state variables) are different. Most propositions are modified by a single action, but several properties of the world may be affected by the same action, making this heuristic inadmissible for several problems and yielding longer plans. Problems without much information about the desired state cannot take any advantage too, for most states the heuristic return the same value while slowing the search to compute the difference between states and expanding like a breadth-first search. Figure 5 shows an example of computing the Hamming distance between states, with some propositions having a desired value (1 or 0) while others being *do not care* (X). To use this heuristic with the compression algorithm a

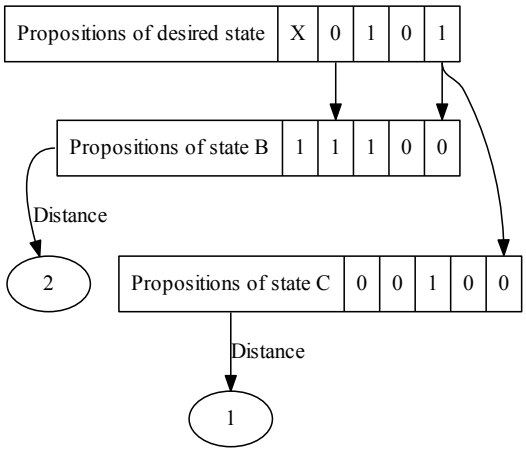

Figure 5: *Hamming distance of two states according to a desired state*

modification was required. The compression algorithm uses several bits for the same exclusive proposition, while the heuristic is trying to compare those bits without exclusivity, therefore a filter is required. The filter tests the compressed part first, if different it counts as one difference, the other bits are treated normally, each one counting as one. The little the difference, the promising the state looks. Incremental solutions, where each part of the desired state can be conquered without lost, will be achieved very fast by this heuristic. While trade solutions, where part of the goal must be lost for another part, will be achieved after other solutions failed. The heuristic can be much better used if different propositions are treated with different values of importance, creating a better evaluation system. The evaluation is the decision process that will help the search, giving priority to expand states that appear to be near the desired state. Although not a relaxation the Hamming distance gives a speed-up for most problems while not adding complexity.

7 Results

Three scenarios were created, one with several random walls and two more realistic ones. The path modes already supported by the BioCrowds simulator were exposed to the API, direct and A*. In every scenario the direct mode shows the problem of not having an entire path to follow, with some agents becoming stuck in the first perpendicular wall they encounter, while others bounce to reach freedom. Using the internal A* generated path is still a good choice when the group only wants to reach some point, but kills the entire freedom. The avoidance system, removing node connections without removing auxins, fits the A*, as both planners only use nodes. When there is a set of requirements to attend the planning is the tool of choice, making all the choices before the simulation starts and creating a robust path for the group. Both A* and planning can fail, that is one of the reasons to find a path before the simulation starts. If A* fails there is no path from the source point to the goal point, the group path can be modified to direct mode, this way the agents will keep trying to achieve their goal without success and their influence on other groups path remains. If planning fails the requirements may be unobtainable and after given-up can switch to A*. Switching path modes can only help if there is no path to meet the requirements, avoidance nodes can block some paths if not used carefully and need to be relaxed by the user or the planner. As A* is affected by the avoidance nodes there is no guarantee that giving up requirements will be enough. This way no group has to be destroyed because of path/requirements failure and the simulation will show what was predicted, total failure (agent can not reach destination with requirements) or partial failure (agent reach destination without requirements) by simplification of goal. The main problem with automatic relaxation of map, removing avoid-

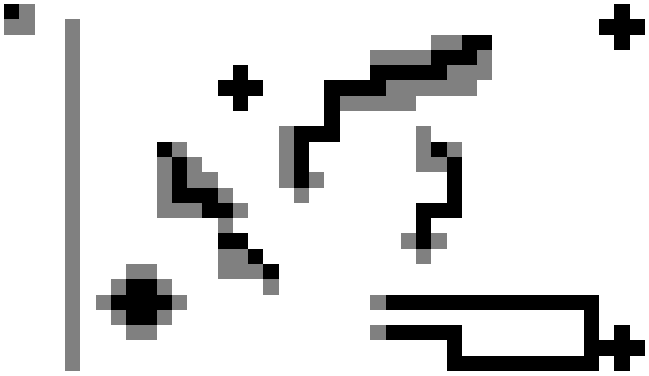

Figure 6: *Random map input as image file*

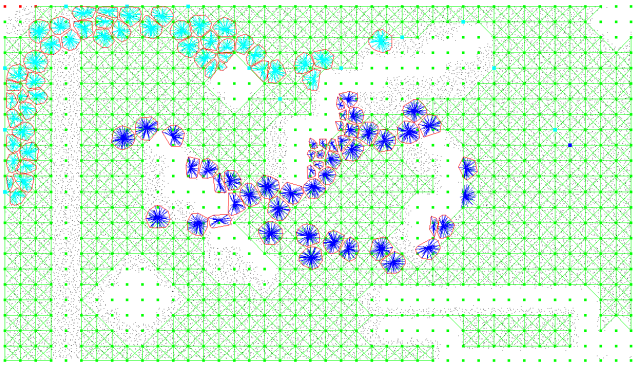

Figure 7: *Random scenario*

ance nodes as planner fails, is how a single group will affect the map used as input by the simulator and how to enforce avoidance of some parts to simulate a real event. Let it be clear that the planner does not know the cause of failure, it simply knows that there is no sequence of actions that can be executed to reach the desired state. The cause may be the map, therefore relaxation of map preferences may help, or the requirements being on completely disconnected regions of the map, being impossible for the agents to reach both requirements, and only a modification of the requirements would be enough to successfully find a plan. Real events being recreated may use more than avoidance nodes to raise the path quality, they may remove a path that people considered too dangerous or was not perceived at the time as a viable path. There are two ways a classical planner could deal with this without adding too much complexity, recreating the problem with less nodes and having several levels of avoidance nodes to remove as planning failed or letting the designer know that is failing and deciding what to do with the current map. We chose the designer as the first option to implement as creating different levels of avoidance is interesting while time consuming to define.

The maps being used here are 42x24 and can yield 5000 propositions of adjacency using a Moore neighborhood (8 connected cells around), this step is generated from a char-based map (an image can also be used, conversion is supported by the API). Although the maps used are rectangular grid-based the idea of connectivity based on Cartesian coordinates would become a constraint for different connectivity systems (like hexagonal tiles) and nodes are referenced by an identifier instead. A new map input method would be required to generate the connections for the planner and a few modifications in the simulator to actually display the map correctly. The Hamming heuristic helped the planner with nothing to a little speed-up for the proposed problems, would be much better for more complex scenarios as in simple scenarios most states are reached after a movement and the distance evaluation stays the same. The great planning optimization came from the compression, as the world can be represented by a single integer and operated through binary operations very fast instead of a sequence of integers. The scenarios are presented in the subsections and require less than 2 seconds to be solved.

7.1 Random Scenario

The random scenario was the first environment created and only illustrates the output of the tool. Can be seen at Fig. 6 and 7 with the two groups, a dark blue group being the control group (without path, direct mode) and the clear blue group being the path planned group. The planning group must avoid a long line to reach its goal, achieve part of its desires with an action in a specific point of the map and reach the other side. The control group just tries to reach the other side, but walls with varying angles make the goal much harder than previously thought. The control group was used to understand how much competition exists between the agents and was the motivator for the use of avoidance nodes, as many agents get stuck in this scenario and how to enforce planning to avoid those dangerous regions.

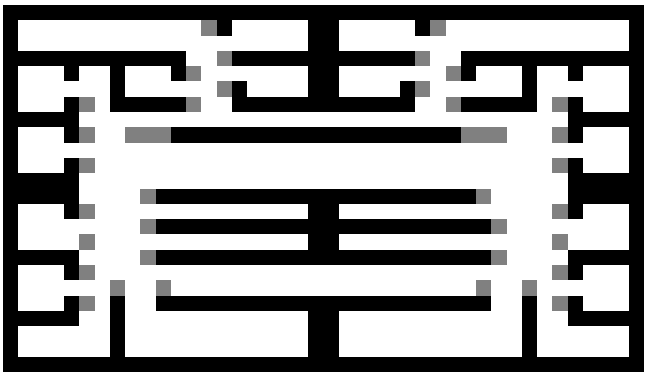

Figure 8: *Lab map input as image file*

7.2 Lab Scenario

In the Lab scenario, Fig. 8 and 9, the narrow paths become a problem that requires more than what the avoidance nodes may help, as other agents push others inside near rooms and the chance of groups collision being extremely high in the narrow corridors. Perhaps the problem is the perception of agents or the design of the space, a more accurate map is required to see if the environment is working against the agents. This environment have two bathrooms available and one printer room, two groups want to use the bathroom and go back to their respective rooms while one group wants to get the papers they printed and also use the bathroom. It is possible to see that each group tries to reach the nearest bathroom and the paper group goes around the lab in order to achieve both goals.

7.3 Snack Bar Scenario

In the snack bar scenario, Figures 10 and 11, the agents can go to a bathroom, pay for the food at the first counter and receive the food at the second counter. Some requirements were created to this scenario: each agent only can buy if they don't need to go to the

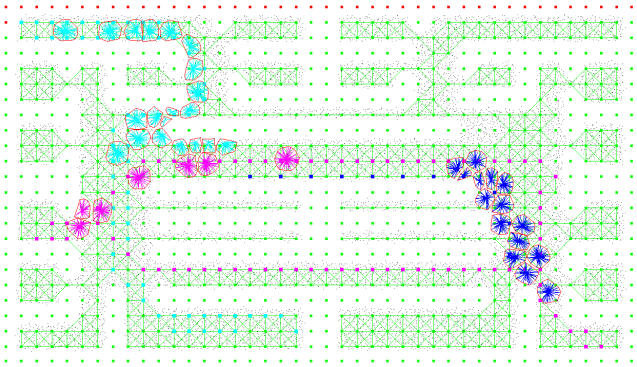

Figure 9: *Lab scenario*

bathroom first, once they bought they can receive the food. Some groups already start with no bathroom need while others only want to buy the food to later eat. The freedom here is that the bathroom can be easily reallocated and more than one bathroom can exist without problem, even more complex relations can be created, but the goal was to simplify the entire process, therefore there is no need to define the map and where the actions are possible at the same time, the map only holds the walls and the requirements can be obtained at specific nodes once their preconditions are satisfied.

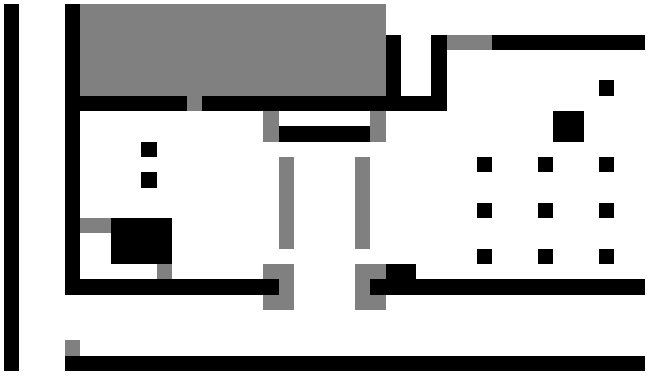

Figure 10: *Snack bar map input as image file*

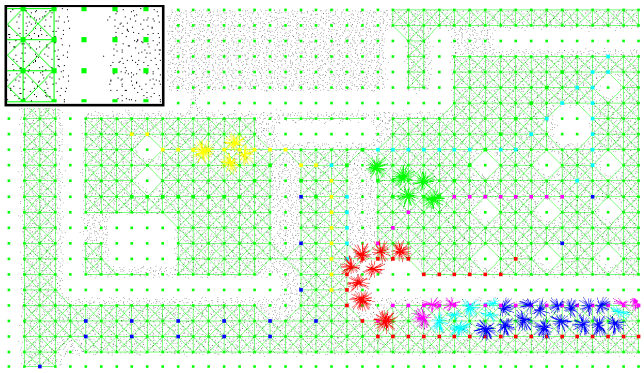

Figure 11: *Snack bar scenario*

8 Conclusion and Future Work

In this paper, we have described a novel approach based on classical planning for the generation of group behavior in crowd simulation. Classical planning and some domain knowledge can be applied to yield custom paths to BioCrowds, adding not just complexity to the paths but lowering the chance of failure (in the form of agents being unrealistically stuck in the map) as well as providing a more meaningful way to represent subgoals for the agents being simulated. Although our current implementation shows a relatively small set of subgoals being expressed by a single agent group, our approach can be easily employed to help develop more realistic scenarios without the need to explicitly define alternative paths to subgoals, ultimately allowing a simulation designer to focus on higher-level agent behavior. Although one may argue that the centralized and deterministic solution of the planner undermines the approach of crowd simulation with distributed behavior, two considerations can be used in favor of our implementation. The first is that knowledge of the markers is exactly the same for all BioCrowds agents. Every agent perceives the world in the same way, and no space is invaded because of different point of views, consequently centralized marker knowledge is already assumed by the BioCrowds model. The second point is the that the generated plans are not followed blindly by the agents, as each agent tries to solve the local problems based on their specific situation during simulation. Thus, our plans are merely guidelines for intermediary points that the agent follow as they try to reach their goal. Agents are grouped just for time-saving purposes as all would plan for the same goal from the same starting point. The only limits of planning being time and memory for the process, a thousand agents would plan the same way as a single agent, the difference would happen just in the simulation as less markers would be available per agent in the paths planned.

Our implementation was validated in a number of scenarios, however, there are some problems in our current setup. In some scenarios, maps with narrow choke points tend to require user intervention through the explicit specification of avoidance points. Although our focus so far has been on the integration of a planner into the crowd simulation tool, we aim to expand our work in a number of ways. First, we aim to investigate other mechanisms for node avoidance besides user-expressed avoidance nodes, and instead generate them automatically based on generic sets of rules (e.g. using cellular automata). Moreover, we aim to allow different preferences to be used for different subgroups within the same simulation (e.g. to avoid certain map regions as only authorized personnel may enter). Performance is not considered a problem, most problems are solved in less than 2 seconds by an interpreted planner with several type conversions and IO to the disk for debug and input to the simulator. A tailored version for a specific system could use low-level optimizations in the integration. Also most simulations and games do not require long plans every few seconds with immediate response. It is interesting to see the agents also taking time to think, perhaps leaving one core in a multicore machine for planning. The only problem is how to act once the plan fails, either by memory, time-out or impossibility to find a solution, as this may have to be handled different for each scenario the agents are in.

References

BONET, B., AND GEFFNER, H. 2001. Planning as heuristic search. *Artificial Intelligence 129*, 1, 5–33.

CASSOL, V. J., MARSON, F. P., VENDRAMINI, M., PARAVISI, M., BICHO, A., JUNG, C., AND MUSSE, S. 2011. Simulation of autonomous agents using terrain reasoning. In *Proc. the Twelfth IASTED International Conference on Computer Graphics and Imaging (CGIM 2011), Innsbruck, Austria. IASTED/ACTA Press*.

COLES, A., COLES, A., OLAYA, A. G., JIMÉNEZ, S., LÓPEZ, C. L., SANNER, S., AND YOON, S. 2012. A survey of the seventh international planning competition. *AI Magazine 33*, 1, 83–88.

DE LIMA BICHO, A. 2009. *Da modelagem de plantas à dinâmica de multidões: um modelo de animação comportamental bio-inspirado*. PhD thesis, Universidade Estadual de Campinas.

FIKES, R. E., AND NILSSON, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence 2*, 3, 189–208.

FLACH, L. M., CASSOL, V. J., MARSON, F. P., AND MUSSE, S. R. 2013. A procedural approach to simulate virtual agents behaviors in indoor environments. In *Intelligent Virtual Agents*, Springer, 448.

FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 29–38.

GHALLAB, M., NAU, D., AND TRAVERSO, P. 2004. *Automated Planning: Theory and Practice*. Elsevier.

HAMMING, R. W. 1950. Error detecting and error correcting codes. *Bell System technical journal 29*, 2, 147–160.

HELBING, D., AND MOLNAR, P. 1995. Social force model for pedestrian dynamics. *Physical review E 51*, 5, 4282.

LI, T.-Y., JENG, Y.-J., AND CHANG, S.-I. 2001. Simulating virtual human crowds with a leader-follower model. In *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*, IEEE, 93–102.

MCDERMOTT, D., GHALLAB, M., HOWE, A., KNOBLOCK, C., RAM, A., VELOSO, M., WELD, D., AND WILKINS, D. 1998. PDDL-the planning domain definition language.

MENEGUZZI, F., AND DE SILVA, L. 2013. Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review FirstView* (9), 1–44.

MUSSE, S. R., AND THALMANN, D. 1997. *A model of human crowd behavior: Group inter-relationship and collision detection analysis*. Springer.

NEBEL, B. 2011. On the compilability and expressive power of propositional planning formalisms. *arXiv preprint arXiv:1106.0247*.

ORKIN, J. 2006. Three states and a plan: the ai of fear. In *Game Developers Conference*, vol. 2006, 4.

PELECHANO, N., O'BRIEN, K., SILVERMAN, B., AND BADLER, N. 2005. Crowd simulation incorporating agent psychological models, roles and communication. Tech. rep., DTIC Document.

REYNOLDS, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics 21*, 4, 25–34.

SOLMAZ, B., MOORE, B. E., AND SHAH, M. 2012. Identifying behaviors in crowd scenes using stability analysis for dynamical systems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 34*, 10, 2064–2070.

SUD, A., GAYLE, R., ANDERSEN, E., GUY, S., LIN, M., AND MANOCHA, D. 2007. Real-time navigation of independent agents using adaptive roadmaps. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, ACM, 99–106.

SUNG, M., GLEICHER, M., AND CHENNEY, S. 2004. Scalable behaviors for crowd simulation. In *Computer Graphics Forum*, vol. 23, Wiley Online Library, 519–528.

TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum crowds. In *ACM Transactions on Graphics (TOG)*, vol. 25, ACM, 1160–1168.

WINIKOFF, M., PADGHAM, L., HARLAND, J., AND THANGARAJAH, J. 2002. Declarative & Procedural Goals in Intelligent Agent Systems. In *Proceedings of the Eighth International Conference on Principles and Knowledge Representation and Reasoning*, Morgan Kaufmann, D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, Eds., 470–481.