A Hybrid Reinforcement Learning and Cellular Automata Model for Crowd Simulation on the GPU

Sergio Ruiz[1(✉)] and Benjamín Hernández[2]

[1] Tecnológico de Monterrey, Mexico City, Mexico
sergio.ruiz.loza@itesm.mx
[2] Oak Ridge National Laboratory, Oak Ridge, TN, USA
hernandezarb@ornl.gov

Abstract. We present a GPU-based hybrid model for crowd simulations. The model uses reinforcement learning to guide groups of pedestrians towards a goal while adapting to environmental dynamics, and a cellular automaton to describe individual pedestrians' interactions. In contrast to traditional multi-agent reinforcement learning methods, our model encodes the learned navigation policy into a navigation map, which is used by the cellular automaton's update rule to calculate the next simulation step. As a result, reinforcement learning is independent of the number of agents, allowing the simulation of large crowds. Implementation of this model on the GPU allows interactive simulations of several hundreds of pedestrians.

Keywords: Reinforcement learning · Crowd simulation · Cellular automata · GPU

1 Introduction

Understanding the complexity of the metropolis at large scale has been made possible through simulations. The transportation community makes use of pedestrian simulations to plan evacuation routes and efficient commuting, or for city risk management and mitigation. In particular, microscopic modeling has taken on an increasingly important role in research and decision-making processes. Furthermore, close-to-real-time performance and the ability to model dozens of operational scenarios is important so that decision makers can choose the best course of action in a timely fashion.

There is a large body of work dedicated to microscopic modeling of crowds: rule-based [22], physics-based methods [12], and velocity-based [21]. However, we observe that pedestrians make a sequential decision process, constrained by—for example—physical traits, whether to reach their destinations in the least amount of time, by taking the quickest route, or any other goal which they seek to reach optimally. Specially, Reinforcement Learning (RL) provides a convenient framework for modeling pedestrian decision-making [8,17,18,29]. But the

E. Meneses et al. (Eds.): CARLA 2018, CCIS 979, pp. 59–74, 2019.
https://doi.org/10.1007/978-3-030-16205-4_5

problem with Multi-agent RL methods is that they are computationally expensive [16] and a RL problem needs to be solved for each pedestrian, thus reducing its application to small groups of agents.

We propose a new method to reduce the computational cost of multi-agent RL by encoding the learned policy into a navigation map, which in turn is used to guide the crowd. Local Collision Avoidance (LCA) is achieved by coupling our RL model with a Cellular Automaton (CA) model, using data structures based on two-dimensional grids to partition the navigable space: after the RL step, its resultant policy is refined into a local navigation map, which is the input for the CA update rule, in order to provide individual separation, control and velocities of agents toward goals. The contributions of this work are:

Embedding the Learned Policy into a Navigation Map. In multi-agent RL models for crowd simulations, the set of states grows exponentially according to the number of episodes and goals [29] or with the number of agents and actions [11]. To reduce the exponential growth of states with the number of agents and actions, we propose to encode states, and the learned navigation policy, into a coarse navigation map. The learned navigation policy is then used in a finer, local navigation map by the cellular automaton's update rule to displace agents while avoiding collisions.

On-Line and Interactive RL Training. Paired with our first contribution, a GPU (graphics processing unit) implementation of our RL model, reduces the training to only a few milliseconds which allows interactive steering of large crowds.

A Scatter and Gather Approach as a CA Update Rule. Scatter and gather data-parallel primitives allow an efficient implementation of our LCA approach on the GPU and provide individual pedestrian navigation and behavior control.

The rest of this paper is organized as follows. In Sect. 2 we present prior work, relevant to the areas of RL and CA in micro-scale crowd models. Later, in Sect. 3, a review of the RL-Navigation framework is followed by the description of the CA-LCA model. In Sect. 4, we present numerical measurements of our implementation in different scenarios. Finally, in Sect. 5 we present our conclusions and future work for this research.

2 Related Work

The main approaches to microscopic modeling of crowds[1] are: (1) Rule-based, which defines steering collision-free behaviors: cohesion, separation and alignment [22]; (2) Physics-based, which model agents, agents' behaviors and destinations as attractive or repulsive forces [12]; and (3) Velocity-based, which calculate a set of velocities that lead to a collision with an obstacle; to move

[1] A complete survey on crowd simulation can be found in [28].

on routes without collision, agents choose velocities out of this domain [21]. The main drawback of these techniques, is that all rules, parameters or input variables, require intensive tuning to model specific pedestrian behaviors [20]. Recently, RL and Markovian models[2] have attracted attention in the crowd simulation community because their formalism makes the specification of such rules easier, and eliminates the use of finely tuned variables.

2.1 Reinforcement Learning in Crowd Simulation

In general, RL has been applied to control theory, robotics, transportation engineering, logistics and multi-agent systems; a complete survey of multi-agent RL and its applications can be found in [7]. We summarize its applications to crowd simulation next.

Torrey [29] describes the challenges of multi-agent RL applied to a simplified school environment where agents move from one classroom to another, while staying in corridors to chat with other agents in between. She proposed that the reward function should be specified by an agent's internal motivations and found that the set of states, S, grew exponentially according to the number of episodes and distance to its goals. S growth was reduced by doing observations at intervals. Martinez-Gil et al. [17,18] proposed a multi-agent RL method to simulate a group of agents leaving a single-door scenario. They studied the scalability of their method by transferring the learned value function to larger scenarios with different numbers of agents. Later, they adopted a distributed memory model by using the Message Passing Interface [19]. Casadiego and Pelechano [8] proposed a similar approach, sharing a table of Q-values between different agents. Godoy et al. [11] proposed an online-RL method to improve the behavior of agents and reduce the congestion problem in a bottleneck scenario. They also noted that the state space grew exponentially with the number of agents and actions, which is computationally and memory demanding; thus, instead of learning a policy for the complete state-action space, agents learned from the recent history of action-reward pairs and feedback from the simulation.

Closely related to RL, Banerjee et al. [3] used a Markov Decision Process (MDP) to achieve adaptable navigation by analyzing navigable spaces. By means of a pre-calculated layered approach, the authors demonstrated that MDPs are a viable tool to produce agent paths dynamically. However, the implementation is limited by the number of dynamically introduced obstacles. Ruiz and Hernández [24] proposed a single layered MDP to calculate navigation routes free of collisions and "micro-scenarios" to dynamically adjust these trajectories in presence of new obstacles or other pedestrians. Later, they proposed two optimization techniques to solve a MDP interactively for crowd navigation (1) reduce the set of states by using an hexagonal grid and (2) a parallel implementation of the value iteration algorithm [23] and reported a technique to couple their MDP solver with an interactive 3D crowd visualization system [25].

[2] Deep Reinforcement Learning techniques are out of the scope of this research.

2.2 Cellular Automata for Pedestrian Behavior Modeling

CA pedestrian models have been researched vastly by the transportation community, and similar to RL and Markovian solutions, as stated by Blue and Adler [5] *"...the attractiveness of using CA is that the interactions of the entities are based on intuitively understandable behavioral rules"*. The idea of using CA for crowd modeling was inspired by its successful application to vehicular traffic models, by extending moves in one dimension defined by a car lane to a two dimensional space. General considerations for CA pedestrian models are:

Navigation space is discretized in cells and each cell should be big enough to fit a person, commonly a size of 0.4×0.4 meters is used.

Cells are marked as occupied if an agent's position matches that cell or free otherwise. Interacting range among cells defines "how far" an agent can see.

Agents move according to translation rules, which can be applied in parallel or sequentially. In this context, parallel means that all the cells are inspected first, then all the agents are displaced to free cells. Sequentially means that a given cell is inspected, and then its corresponding agents displaced, before proceeding with the next cell.

Blue and Adler [4] modeled walkways using multiple lanes allowing single directional pedestrian flow. Their CA rules were defined by a two-stage parallel update supporting lane changing and cell hopping, and later introduced bi-directional walkways [5] by modeling side-stepping, forward-movement and conflict-mitigation behaviors. The model included flows in directionally separated lanes, interspersed flow, and dynamic multi-lane flow. Weifeng et al. [30] also studied the bi-directional pedestrian flow, particularly the phase transition of pedestrian counter flow. Klupfel et al. [15] proposed the use of CA to model on-board passenger ships evacuations. They considered that agents can choose between different evacuation routes depending on their sight range, also modeling swaying and indecision behaviors. Burstedde et al. [6] introduced the idea of chemotaxis (floor field) to model individual intelligence and traces left by pedestrians. Each trace decayed to restrict the agents' interaction range. Kirchner et al. [14] extended this concept to support a probability factor in the diffusion and decay functions to model different behaviors as regular, panic or herding. Bandini et al. [2] used the floor field concept in the context of *situated cellular agents*, to model the action-at-a-distance behavior.

As mentioned before, CA cells should be small enough to fit a person; several researchers have studied how different cell sizes affect pedestrian models. For example, Kirchner et al. [13] studied the effects of reducing the cell size so pedestrians could occupy more than one cell. This modification allowed to represent finer and more accurate time scales, geometrical structures and pedestrian speeds non-multiple of 0.4 m. They also noted that finer discretizations could make CA comparable to continuous models. Later, Sarmady et al. [27] proposed a finer discretization at pedestrian level, i.e. each agent was represented by a

set of 0.05×0.05 m. cells being moved by a least-effort algorithm. Finally, Feliciani and Nishinari [10] suggested the use of a different method to discretize the navigable space to add more CA locations within the traditional grid approach; these locations were added at the edges and at the corners of each cell, allowing to simulate the *enter-crowd*, *move-in-crowd* and *leave-crowd* behaviors.

3 Problem Modeling

3.1 Reinforcement Learning for Navigation

Starting from the observation that a pedestrian—while moving through a navigable space—makes sequential decisions to find a path from its current position to a goal, we model this path by constructing a set of additive rewards using Reinforcement Learning. For a simulated group of pedestrians or crowd, multiple agents need to learn through RL, posing a computational challenge because the set of states grows exponentially due to the number of episodes, goals [29], agents and actions [11]. Moreover, close-to-real-time performance[3] is required to support decision-making processes during the simulation.

An approach to reduce the RL complexity in a multi-agent simulation is sharing the learned Q-values among different agents [8,17,18]. Our contribution is to build a *navigation map*: we use a coarse and discrete representation of the navigable space, where each cell represents a group of agents' state, i.e. its current position within the map, then after the RL process, its resultant policy as directions to follow, is refined into a *local navigation map*: an input to the CA in order to grant individual separation and control for agents. As a result, our approach keeps the number of states low and is independent of the number of agents, because only one RL solution is computed based on a navigation map. In other words, RL provides a navigation solution for pedestrian groups, while the CA provides individual navigation, control and LCA. Finally a GPU implementation of this algorithm allows simulations to run at interactive rates.

We use the MDP formalism to model pedestrian navigation as a RL problem. Based on our proposed solution, we define the MDP tuple $M =< S, A, T, R >$ as follows.

S (finite set of states) composed of every cell resulting from partitioning the navigable space.
A (finite set of actions) representing an agent's available movement directions, e.g. forward, left, right, and so on.
T (transition model) defined by the probabilities of choosing a given action from set A.
R (reward function) are cells marked as points of interest (high valued rewards), navigable space (medium valued rewards) and obstacles (low valued rewards).

[3] Thirty to forty five ms per simulation step.

From the previous setup, we calculate optimal navigation directions, the optimal policy (π^*) that achieves maximum reward from all states, using the Value Iteration algorithm as follows.

$$\begin{aligned} &\pi_t^*(s) = argmax_a Q_t(s,a) \\ &Q_t(s,a) = R(s,a) + \gamma \textstyle\sum_{j=0}^{|A|-1} T_{sj}^a V_{t-1}(j) \\ &V_t(s) = Q_t(s, \pi^*(s)) \\ &V_0(s) = 0 \end{aligned} \tag{1}$$

Such that $Q_t(s,a)$ is the value of performing action a—in this case moving towards direction a—from cell s; $V_t(s)$ represents the reward value of cell s at time t; $\gamma \in [0,1]$ is a future reward discount factor and, T_{sj}^a is the transition, function defined by the probability of an agent moving to state j from state s by action a. The defined MDP is fully observable, since the simulation's initial configuration is known and RL is episodic, i.e. it is solved for a number of iterations when the environment changes, allowing a gradual adaptation (learning) of the crowd flow in response to such changes. Then, the episode stops when π^* is achieved by convergence, and pedestrians can avoid the new obstacle, or walk towards the new goal through the optimal path.

	1	2	3	4
a	-3	-3	-3	+100
b	-3	-100	-3	-100
c	-3	-3	-3	-3

Fig. 1. Simple scenario where navigable space has a penalty of -3, obstacles a penalty of -100 and the exit a reward of 100. The orange cell represents an agent's position. (Color figure online)

We explain our RL-Navigation parallelization strategy by example, considering a discretized 3×4 map that results in twelve states. Its reward function is -3 for navigable space, -100 for obstacles and 100 for exits as shown in Fig. 1. For simplicity, our pedestrians can choose from three actions (1) moving West, W, (2) moving North, N, and (3) moving East, E; thus, $A = \{W, N, E\}$; a reward discount factor $\gamma = 1$; a probability for T_{sj}^a of $p = 0.8$ when choosing the current action a and $q = 0.1$ otherwise, to ensure that the sum of probabilities is 1. Using Eq. 1, we compute $\pi_t^*(s)$ after a RL episode for cell $a3$. Considering $\pi_t^*(s)$ and $Q_t(s,a)$ from Eq. 1 and replacing s and a terms by $a3$ and actions from set A, we have:

$$\pi_t^*(a3) = max\{Q(a3,E), Q(a3,W), Q(a3,N)\} \tag{2}$$

$$\begin{aligned} Q_t(a3,E) &= R(a3,E) + \gamma[pR(a3,E) + qR(a3,W) + qR(a3,N)] \\ Q_t(a3,W) &= R(a3,W) + \gamma[qR(a3,E) + pR(a3,W) + qR(a3,N)] \\ Q_t(a3,N) &= R(a3,N) + \gamma[qR(a3,E) + qR(a3,W) + pR(a3,N)] \end{aligned} \tag{3}$$

Replacing the corresponding variables with their numerical values, the action that maximizes the reward is E, moving east to reach the exit. From Eqs. 3, note that:

- For each cell or state, a similar set of equations is to be solved.
- The set of equations can be solved in parallel, if each cell queries rewards from neighboring cells.

– Probability variables p, q and reward values $R(s, a)$ can be stored in arrays for each cell.
– Expressions in brackets can be solved by parallel reductions. A second parallel reduction using conditionals will solve Eq. 2.

3.2 Cellular Automata for Local Collision Avoidance

In this section, we present a cellular automaton for LCA formulation, to be executed in parallel. We propose to solve the crowd flow and direction by introducing the learned policy—whether optimal or sub-optimal—as a local directional guide, granting separation and control for individual agents. We begin by observing a basic reference *RL for Navigation* example, in which a group of agents is to be directed towards a unique goal, while avoiding an obstacle, as shown in Fig. 2(a). Agents are likely to collide by merely following navigation paths, considering that each agent moves with unique velocity, risking agent-to-agent collision and, depending on its starting position within the cell, could even crash against the obstacle. We propose the use of CA to avoid these collisions. First, a sub-partition of the navigable space is computed, i.e. a partition of the RL states. Although there could be infinite partitions for the states set, an area equivalent to 0.4×0.4 m will serve to avoid pedestrian collisions against obstacles within the environment (Fig. 2(b)) as mentioned in Sect. 2.2. In this sub-partition, or *Local Navigation Map* (LNM), cells will be marked either as $OPENSPACE$, $OBSTACLE$, $GOAL$ or a $DIR[d] : 0 < d < 9, d \in \mathbb{N}$—one of eight directions—(Figure 2(c)).

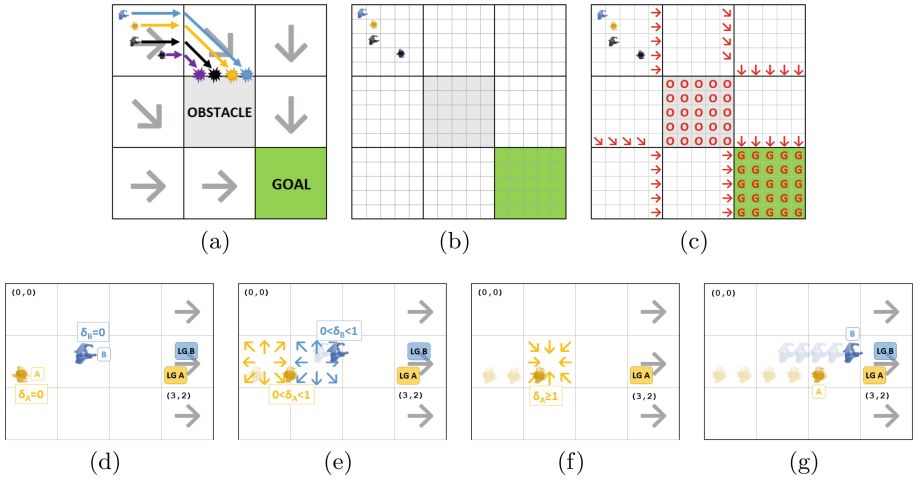

Fig. 2. TOP: LCA principles. (a) Agents will collide when only following the policy. (b) Partition for LCA. (c) Local Navigation Map from Π^* where arrows represent local goals. **BOTTOM**: Basic Scatter-Gather algorithm. (d) Local goal assignment. (e) Scatter step. (f) Gather step. (g) Final effect.

Then the *CA for LCA* is composed of:

A Set of Connected Sites represented by the LNM sub-partition.

State Variables as one of the following: $OPENSPACE$, $OBSTACLE$, $GOAL$ or $DIR[d]$ will determine the ability of agents to move or wait.

An Update Rule in two steps: a stage in which CA cells *scatter* the agents inside of them, and a stage in which CA cells *gather* nearby incoming agents.

The purpose of using scatter and gather operations as an update rule, is to translate agents toward the *nearest* CA cell marked as $DIR[d]$, or the *Local Goal* (LG) for an agent, defining its translation between CA cells as follows.

1. A starting relative position within the starting CA_s cell: $S = CA_s(x, z) + (\Delta x, \Delta z)$.
2. A vector to direct the agent and determine it's exit location: $E = LCA_e(x, z) + (\Delta x, \Delta z)$.
3. A weighted parameter considering the agent's predefined speed, terrain type at the current cell, as well as the elapsed time to generate a value $0 \leq \delta \leq 1$, $\delta \in \mathbb{R}$ that is incremented by an amount λ at each simulation step, assigning a unique speed to each agent within the simulated crowd.
4. Linear interpolation computes an agent's position as $P = S + \delta(E - S)$.

As an example, consider the bottom part of Fig. 2, where agent A is at CA cell $(0, 1)$ and agent B is at CA cell $(1, 1)$; agent A is 25% faster than agent B. The following algorithm performs the Scatter step, then the Gather step, for each CA cell.

1. *Scatter.*
   - For each agent at this CA cell, if its parameter is $\delta = 0$, then assign a LG as follows.
     - If this CA cell *is not* a LG, then assign the nearest LG within this RL cell as shown in Fig. 2(d).
     - If this CA cell *is* a LG, then assign the LG as the neighboring cell pointed to by Π^*, i.e. $DIR[d]$ in the LNM.
   - For each agent at this CA cell, if its parameter is $0 < \delta < 1$, then increment its parameter so that $\delta = \delta + \lambda$. Note that in Fig. 2(e), agent A will reach the end of its path first.
2. *Gather.* For agents moving towards this cell (i.e. query neighboring cells), if their parameter is $\delta \geq 1$, then reset their parameter to $\delta = 0$, and also set the agent's current cell to this CA cell as shown in Fig. 2(f).

Iteration of this algorithm, at each simulation frame, generates the cyclic phenomena shown in Fig. 2(g). Notice that translation between RL cells is guaranteed because the LNM preserves information from Π^* at each CA cell marked $DIR[d]$.

3.3 Improved Cellular Automata for Local Collision Avoidance

A problem with the previous Scatter-Gather algorithm is that, as the number of agents increases, a race condition will arise for agents competing to occupy the same CA cell causing an unpredictable system behavior. A second problem is the inability of faster agents to steer when they encounter a slower agent in their path, forming a single-lane queue even when open space is available around to pass (queuing problem). Furthermore, this results in the inability of agents to disperse around a congested area, waiting to occupy their local goal (waiting problem). We solve these issues by applying general semaphores [9] and a steering system to the basic Scatter-Gather algorithm, respectively, while keeping the parallel implementation.

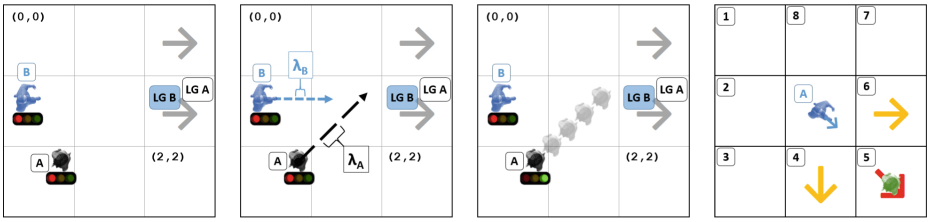

Fig. 3. Race condition, queuing and waiting solutions. (a) Flag implementation. (b) Distance increment measure step, where $\lambda_A > \lambda_B$. (c) $Agent_B$ gives way to $Agent_A$. (d) Similar directions: $DIR[4]$ and $DIR[6]$ are similar to $DIR[5]$ for $Agent_A$.

Race condition solution. In order to avoid the race condition problem, a `boolean` flag is implemented in the agent model, signaling the agent to stop moving when the flag's value is `false`. The following additional steps are scheduled in the cellular automaton, just before the scatter-gather step is performed.

1. For all agents incoming to destination CA cell (i.e. query neighboring cells), set all flags to `false` (Fig. 3(a)).
2. If destination CA cell is occupied and $0 < \delta_i < 1$, do nothing further.
3. For all agents incoming to destination CA cell, determine the agent with the greatest parameter increment λ (Fig. 3(b)).
4. Set the flag to `true` only for the agent with the greatest parameter increment λ (Fig. 3(c)).

Finally, we modify the Scatter-Gather algorithm to only scatter agents if their flag is set to `true`.

Queue and Waiting Solution. An agent may only move along adjacent cells, so for a given CA cell, only the eight neighboring cells are to be considered as alternatives when the next cell is occupied. We determine the optimal alternative with the aid of the following example: we suppose that $Agent_A$ is trying to move to the adjacent cell in direction $DIR[5]$, but that cell is currently occupied, as

shown in Fig. 3(d). Similar directions to $DIR[5]$: $DIR[4]$ and $DIR[6]$ are the best alternatives, as they point to cells adjacent to the original target. In general, for an N-directional set, $DIR[(i+1)\%N]$ and $DIR[(i+N-1)\%N]$ are the best alternatives to $DIR[i]$. Of the cells pointed to by these alternative directions, the closest to the LG will have priority. Now we can modify the Scatter-Gather algorithm as follows.

- Agent instances will maintain the index of the next CA cell in their path.
- Agent instances will maintain the index of the best CA cell alternatives to the next CA cell.
- If the next CA cell is occupied *and* the movement flag is set to `false` after the Race condition solution:
  - Check if the best alternative CA cell closest to the LG is occupied, set the next CA cell index to this alternative if it is not occupied.
  - If the best alternative CA cell closest to the LG is occupied, then check if the best alternative CA cell farthest to the LG is occupied, set the next CA cell index to this alternative if it is not occupied.
  - If the next CA cell index has changed:
    * If this CA cell *is not* a LG, then assign the nearest LG within this RL cell.
    * If this CA cell *is* a LG, then assign the LG as the neighboring cell pointed to by π^*, i.e. $DIR[d]$.
  - If both of the best alternative cells are occupied, do nothing further, and the agent is forced to wait, as otherwise it would steer away from its goal.

3.4 Coupling Navigation and Local Collision Avoidance

In the proposed model, the *RL for Navigation* and *CA for LCA* methods are linked by the LNM, since it is on this finer partition that the cellular automaton operates to coordinate the agents' movement. Once the Navigation policy—whether π^* or a sub-optimal policy—is activated, the LNM is computed by a parallel version of Algorithm 1. The *CA for LCA* method may be coupled to the *RL for Navigation* method because:

- Its set of connected sites is a partition of the RL states set, which in turn is a partition of the navigable space.
- The LNM preserves policy information and further, uses it as the input required to direct and control agents.
- The improved CA for LCA algorithm integrates a solution to the Navigation and Local Collision Avoidance problems.

input : RL Policy Π $POLICY$ of size $mdpWidth \times mdpDepth$
output: Local Navigation Map LNM of size $lcaWidth \times lcaDepth$

```
1  if lcaWidth % mdpWidth == 0 then
2      lcaWidthRatio ← lcaWidth / mdpWidth;
3      lcaDepthRatio ← lcaDepth / mdpDepth;
4      lcaRatio ← lcaWidthRatio × lcaDepthRatio;
5      for i ← 0 to lcaWidth × lcaDepth do
6          LNM[i] ← OPENSPACE;
7      end
8      for lx ← 0 to lcaWidth do
9          for lz ← 0 to lcaDepth do
10             mx ← lx / lcaWidthRatio;
11             mz ← lz / lcaDepthRatio;
12             mi ← mz × mdpWidth + mx;
13             if POLICY[mi] == OBSTACLE || GOAL then
14                 li ← lz × lcaWidth + lx;
15                 LNM[li] ← POLICY[mi];
16             end
17         end
18     end
19     foreach rlCell C do                                  // Π placement heuristic
20         foreach lcaCell L in perimeter of C do
21             direction ← POLICY[C];
22             if DoesNotPointToObstacle(direction,L) then
23                 if EdgeMatchesPolicyDir(direction,L) then
24                     LNM[L] ← direction;
25                 end
26             end
27         end
28     end
29 end
```

Algorithm 1. Local Navigation Map from its RL policy.

4 Implementation Details and Results

Our algorithm uses data parallel primitives (reductions, reductions using conditionals, transformations, gather and scatter) exposed in Thrust [1]. To achieve maximum performance between simulation and visualization, we tightly coupled the *RL for Navigation* and the *CA for LCA* stages with a crowd visualization engine implemented in C/C++ and OpenGL [26]. At run-time, a value iteration step is interleaved with frame rendering, with the objective of keeping an interactive simulation, as shown in Fig. 4. At a configurable iteration interval, the sub-optimal policy is downloaded and updated on the host, simulating the crowd's learning process adjustment to a change in the scenario, as new obstacles and goals are added or removed. Finally, the optimal policy is downloaded and updated on the host [25].

We designed two experiments to report the performance of our implementation. The first experiment was run in a 20-core Xeon CPU E5-2687W at 3.10 GHz, and a NVIDIA Pascal Titan X GPU using CUDA 8.0 and Thrust 1.8. It consisted in measuring the GPU performance of the *RL for Navigation* algorithm on different scenario sizes using as a baseline a parallel CPU implementation using 40 threads[4]. All scenarios in both experiments were specified similarly to

[4] Multi-threading was exposed by Thrust's TBB backend.

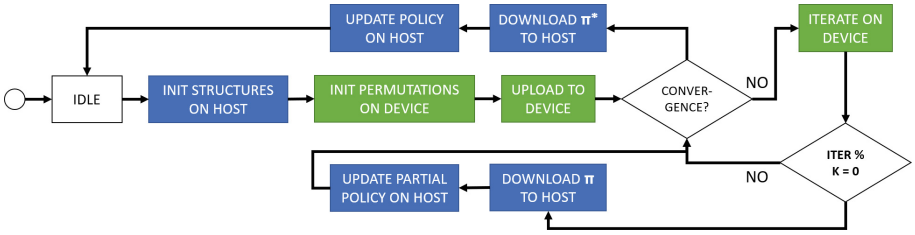

Fig. 4. Segmented GPU-based Value Iteration, where each process is interleaved with frame rendering. Configurable parameter K updates the current policy at a preset interval, integrating the learning process to the simulation.

Fig. 1 in a CSV file. For this experiment we started with a 16×16 cells area with no obstacles and a goal at the center. Then, we replicated this area to produce larger scenarios.

Figure 5 shows the performance of our algorithm (left) and GPU speedup (right). In small scenarios the parallel CPU version performed better; however, beyond 128×128 cells, the GPU outperformed the CPU due to its bulk processing capabilities. Also note GPU time for scenarios of 128×128 cells and smaller does not incur in a significant performance loss.

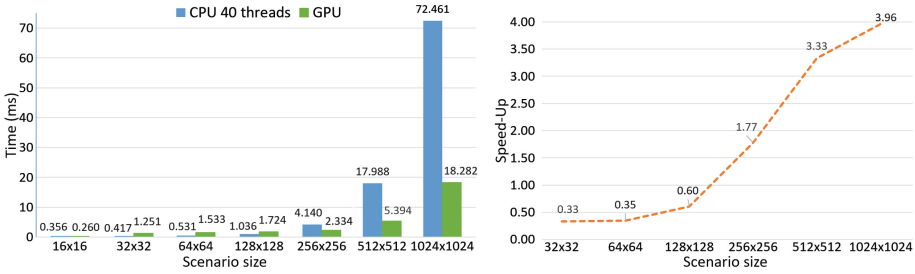

Fig. 5. Performance results and speed-up. *Left* parallel CPU vs. GPU performance, smaller values are better. *Right* GPU Speed-up.

The second experiment consisted in measuring the GPU performance of our fully integrated system including the hybrid *RL for Navigation - CA for LCA* model and 3D crowd rendering on four typical scenarios in crowd simulation: *bottleneck*, *route preference*, *shortest path* and *bi-directional walkways* and, in a more complex scenario with a large crowd, a *campus* scenario, modeled after actual facilities. The tests for these scenarios were run in an Intel Core i7-6700HQ @ 2.60GHz CPU, CUDA 8.0, Thrust 1.8 on a laptop PC connected to an external GPU (eGPU) graphics accelerator, hosting a NVIDIA GeForce GTX 1060 3GB GPU.

Table 1 shows scenario characteristics in three sections. The first one shows details of the *RL for Navigation* (Sect. 3.1) such as map size, number of iterations

to find an optimal policy after adding a new obstacle and total time to find an optimal policy. The second part shows details of the *CA for LCA* (Sect. 3.2 and Sect. 3.3), such as local navigation map sizes, and different times to solve collisions, racing conditions and, scatter and gather operations. The third section shows the total update time and time per frame of our hybrid model. Reported timings show that interactive simulation is feasible with our approach because the update cost for a change in the presented scenarios ranges from 1.89 ms (Bi-directional) to 7.88 ms (Campus) per frame. Visualization results for these experiments are available at https://youtu.be/dkx87F10x6k.

Table 1. Model execution results for the test scenarios.

	Bottleneck	Route preference	Shortest path	Bi-directional	Campus
Agents	256	256	256	256	4,096
RL layers	1	1	1	2	1
RL Width × Depth	20 × 20	32 × 32	64 × 64	16 × 16	200 × 200
RL iterations	29	61	186	162	207
RL avg. iteration (ms)	**1.758**	**1.88**	**2.287**	**0.457**	**3.511**
Total RL (ms)	78	156	562	110	1,218
CA cells per RL Cell	16	8	4	10	2
LNM Width × Depth	320 × 320	256 × 256	256 × 256	160 × 160	400 × 400
LNM update interval	10	10	10	10	10
CA Racing condition (ms)	0.256	0.236	0.186	0.085	0.353
CA Scatter-Gather (ms)	1.939	2.114	1.865	1.351	4.023
Total CA (ms)	**2.195**	**2.35**	**2.051**	**1.436**	**4.376**
Total update time (ms)	80.195	158.35	564.051	111.436	1,222.376
Time per frame (ms)	**3.953**	**4.23**	**4.338**	**1.89**	**7.887**

5 Conclusions and Future Work

We presented a model for crowd Navigation and Local Collision Avoidance in dynamic environments. In contrast to current multi-agent RL for Navigation algorithms, our approach can handle large crowds because (1) we encode states and the learned policy into a finer *local navigation map* that our algorithm uses to steer pedestrians, and (2) GPU implementation of the algorithms allows on-line and near-to-real-time calculation of Navigation policies and CA update rules.

Our model supports different behaviors through MDP layers, as shown in the bi-directional walkway scenario, where different groups moved towards different objectives. On the other hand, from the CA perspective, our approach could resemble the floor field approach; in this matter we are offering an alternative to [2,6,14] by introducing MDPs to model a similar phenomena to that which diffusion and decay functions produce in the floor field method. In particular, different reward values could model the diffusion effect and the discount value, γ, could be used to represent decay functions. Further analysis will help to illustrate these relationships, in addition to the effect produced by the inclusion of goals with different priorities.

In relation to [23], where a fully observable MDP was solved before the resultant policy could be used to steer crowds, in this paper we expose an online reinforcement learning approach by using partial solutions from the MDP, that allow the setting of dynamic goals and obstacles to which the crowd adapts while the simulation is running.

The framework presented in this paper can be extended to further applications, for example, to Geographic Information Systems, since the spatial analysis and mapping of evacuations usually requires the computation of shortest or safest routes, or even preferred routes according to groups of pedestrians. A similar application can be found in daily commuter activity analysis. However, practical applications of our model require calibration and validation, that are left as future work.

Acknowledgements. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. We thank NVIDIA for the donation of the Titan X GPU used in this research. Sergio Ruiz would like to thank the Tecnologico de Monterrey Computer Department for its support.

References

1. NVIDIA Thrust. https://thrust.github.io/. Accessed 14 May 2018
2. Bandini, S., Mauri, G., Vizzari, G.: Supporting action-at-a-distance in situated cellular agents. Fundamenta Informaticae **69**(3), 251–271 (2006)
3. Banerjee, B., Abukmail, A., Kraemer, L.: Advancing the layered approach to agent-based crowd simulation. In: Proceedings of the 22nd ACM/IEEE/SCS Workshop on the Principles of Advanced and Distributed Simulation (PADS), Rome, Italy, pp. 185–192 (2008)
4. Blue, V., Adler, J.: Emergent fundamental pedestrian flows from cellular automata microsimulation. Transp. Res. Rec. J. Transp. Res. Board **1644**(4), 29–36 (1998)
5. Blue, V.J., Adler, J.L.: Cellular automata microsimulation for modeling bi-directional pedestrian walkways. Transp. Res. Part B Methodol. **35**(3), 293–312 (2001)
6. Burstedde, C., Klauck, K., Schadschneider, A., Zittartz, J.: Simulation of pedestrian dynamics using a two-dimensional cellular automaton. Phys. A Stat. Mech. Appl. **295**(3), 507–525 (2001)
7. Buşoniu, L., Babuška, R., De Schutter, B.: A comprehensive survey of multi-agent reinforcement learning. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **38**(2), 156–172 (2008)
8. Casadiego, L., Pelechano, N.: From one to many: simulating groups of agents with reinforcement learning controllers. In: Brinkman, W.-P., Broekens, J., Heylen, D. (eds.) IVA 2015. LNCS (LNAI), vol. 9238, pp. 119–123. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21996-7_12
9. Dijkstra, E.W.: Cooperating sequential processes. In: Hansen, P.B. (ed.) The Origin of Concurrent Programming, pp. 65–138. Springer, New York (2002). https://doi.org/10.1007/978-1-4757-3472-0_2

10. Feliciani, C., Nishinari, K.: An enhanced cellular automata sub-mesh model to study high-density pedestrian crowds. In: El Yacoubi, S., Wąs, J., Bandini, S. (eds.) ACRI 2016. LNCS, vol. 9863, pp. 227–237. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44365-2_23
11. Godoy, J., Karamouzas, I., Guy, S.J., Gini, M.: Online learning for multi-agent local navigation. In: The AAMAS-2013 Workshop on Cognitive Agents for Virtual Environments, Saint Paul, Minnesota, USA (2013)
12. Helbing, D., Molnár, P.: Social force model for pedestrian dynamics. Phys. Rev. E **51**, 4282–4286 (1995)
13. Kirchner, A., Klüpfel, H., Nishinari, K., Schadschneider, A., Schreckenberg, M.: Discretization effects and the influence of walking speed in cellular automata models for pedestrian dynamics. J. Stat. Mech. Theor. Exp. **2004**(10), P10011 (2004)
14. Kirchner, A., Schadschneider, A.: Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics. Phys. A Stat. Mech. Appl. **312**(1), 260–276 (2002)
15. Klüpfel, H., Meyer-König, T., Wahle, J., Schreckenberg, M.: Microscopic simulation of evacuation processes on passenger ships. In: Bandini, S., Worsch, T. (eds.) Theory and practical issues on cellular automata, pp. 63–71. Springer, London (2001). https://doi.org/10.1007/978-1-4471-0709-5_8
16. Koenig, S., Simmons, R.G.: Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains. Carnegie Mellon University, Pittsburgh, PA, USA, Technical report (1992)
17. Martinez-Gil, F., Barber, F., Lozano, M., Grimaldo, F., Fernández, F.: A reinforcement learning approach for multiagent navigation. In: Proceedings of the International Conference on Agents and Artificial Intelligence, ICAART 2010, Artificial Intelligence, vol. 1, pp. 607–610. SciTePress (2010). https://doi.org/10.5220/0002727906070610. ISBN 978-989-674-021-4
18. Martinez-Gil, F., Lozano, M., Fernández, F.: Multi-agent reinforcement learning for simulating pedestrian navigation. In: Vrancx, P., Knudson, M., Grześ, M. (eds.) ALA 2011. LNCS (LNAI), vol. 7113, pp. 54–69. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28499-1_4
19. Martinez-Gil, F., Lozano, M., Fernández, F.: MARL-Ped: a multi-agent reinforcement learning based framework to simulate pedestrian groups. Simul. Model. Pract. Theor. **47**(Complete), 259–275 (2014)
20. Moussaïd, M., Helbing, D., Theraulaz, G.: How simple rules determine pedestrian behavior and crowd disasters. Proc. Nat. Acad. Sci. **108**(17), 6884–6888 (2011)
21. Paris, S., Pettre, J., Donikian, S.: Pedestrian Reactive Navigation for Crowd Simulation: a Predictive Approach. Computer Graphics Forum (2007)
22. Reynolds, C.W.: Flocks, herds and schools: a distributed behavioral model. SIGGRAPH Comput. Graph. **21**(4), 25–34 (1987)
23. Ruiz, S., Hernández, B.: A parallel solver for Markov decision process in crowd simulations. In: 2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI), pp. 107–116 (2015)
24. Ruiz, S., Hernández, B.: Procesos de decisión de Markov y microescenarios para navegación y evasión de colisiones para multitudes. Res. Comput. Sci. **74**, 103–116 (2014)
25. Ruiz, S., Hernández, B.: Real time markov decision processes for crowd simulation. In: Engel, W. (ed.) GPU Zen, pp. 323–341. Black Cat Publishing (2017)
26. Ruiz, S., Hernández, B., Alvarado, A., Rudomín, I.: Reducing memory requirements for diverse animated crowds. In: Proceedings of Motion on Games, MIG 2013, pp. 55:77–55:86. ACM, New York (2013)

27. Sarmady, S., Haron, F., Talib, A.Z.: Simulating crowd movements using fine grid cellular automata. In: 12th International Conference On Computer Modelling and Simulation (UKSim 2010), pp. 428–433. IEEE (2010)
28. Thalmann, D., Musse, S.R.: Crowd Simulation. Springer, London (2013). https://doi.org/10.1007/978-1-84628-825-8
29. Torrey, L.: Crowd simulation via multi-agent reinforcement learning. In: Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. The AAAI Press (2010)
30. Weifeng, F., Lizhong, Y., Weicheng, F.: Simulation of bi-direction pedestrian movement using a cellular automata model. Phys. A Stat. Mech. Appl. **321**(3), 633–640 (2003)