ICIC Express Letters
Volume **17**, Number **1**, January **2023**
 ISSN 1881-803X
pp. **13–24**

COLLISION AVOIDANCE AND PATH PLANNING IN CROWD SIMULATION

Panich Sudkhot[1,*], Kok Wai Wong[2] and Chattrakul Sombattheera[1,*]

[1]Faculty of Informatics
Mahasarakham University
Khamriang, Kantharawichai, Mahasarakham 44150, Thailand
*Corresponding authors: panich.sud@zyntelligent.com; chattrakul.s@msu.ac.th

[2]College of Science, Health, Engineering and Education
Murdoch University
Murdoch, Western Australia 6150, Australia
K.Wong@murdoch.edu.au

Received May 2022; accepted July 2022

Abstract. *Modern crowd simulation needs two important characteristics: i) capability to simulate how human beings plan their paths, and ii) capability to cope with a large number of agents, particularly, in congested areas. Most research in this area does not focus on both issues at the same time because it can be computationally expensive. This research proposes a framework for crowd simulation that embraces these two capabilities and offers low computational cost. We combine BDI (Belief-Desire-Intention) model for planning and making decision, and RVO (Reciprocal Velocity Obstacle) for coping with congested areas. We achieve satisfactory results as a large number of agents can be simulated smoothly. We found that by combining BDI with RVO, the simulation time, i.e., how fast agents can move, is at least 20% faster in all scenarios. The execution time is also reduced dramatically, 35% on average. We can simulate up to 40,000 agents in a large scale area in a typical computer. The framework allows for visualized simulation of 600 agents at 24 frames per second and can go up to 6,400 agents at 1.5 frames per second.*
Keywords: Path planning, BDI agent, Agent-based crowd simulation

1. **Introduction.** Crowd simulation, offering a vast diversity in real world applications, can be divided into two streams: i) macroscopic, models the behavior of the crowd based on mathematical models, and ii) microscopic, models the decision making of individual agents that collectively becomes the behavior of the crowd. The former offers low computation cost and is suitable for large scale simulation. The latter offers details about how individual agents make decision but its computational cost is relatively high. Because of these reasons, most research in crowd simulation tends to focus on either way. In recent years, microscopic has received more attention because the computer processors are cheaper and it offers more details and accuracy in decision making of agents.

In microscopic model, there are two important aspects: i) how we can simulate the decision making as similar as possible to human beings, and ii) how we can handle congested situation efficiently. To answer the first question, it has long been studied how human beings make decision and act, accordingly. In reality, we live in dynamic environments, which keep changing all the time. This means that we have to always plan, act, re-plan, act, repeatedly, in order to achieve our goal, reaching our destinations, etc. We refer to this as *path planning* of agents. The second question arises because agents make decision independently to achieve their goals. The worst case scenarios take place when a lot of agents are moving pass, or towards, a relatively small area, with respect to the number of

DOI: 10.24507/icicel.17.01.13

agents, in different directions. The area becomes it congested. From simulation perspective, agents must be properly navigated out of this. We do not allow agents to collide to each other. This is known as *collision avoidance* in crowd simulation. Handling these two requirements within one system is a challenge because both demand a lot of computation power.

In order to respond to both demands, a lot of simulation systems have been built up on existing techniques, causing more computational cost as an expense of these advances. This prevents these techniques from simulating a large number of agents. In our opinion, the most challenging and interesting research is the autonomous crowd simulation by handling both a large number of agents while capable of emulating high level decision making. We propose a crowd simulation framework based on BDI, a popular psychological principle for decision making at high level, and RVO, an underlying navigation mechanism for agents. The framework performs reasonably well with considerably large number of agents.

The paper is structured as the following: We review related works and research background in Section 2, introduce system overview in Section 3 and show how to implement agent movement in Section 4. We present three experiments in Section 5 and conclusion in Section 6.

2. **Background.**

Over the years, crowd simulation has been growing on various dimensions. In this section, we briefly review advances in crowd simulation and two principles that are basis of our work below.

2.1. **Crowd simulation advances.**

Crowd simulation offers rich applications in various domains, including movies and games [4], urban and architecture planning [5], disaster and crisis handling [6]. Due to variety of its applications, crowd simulation requires different techniques to focus on important aspects of those applications. The first and foremost aspect is visualization. During the early years, visual media or virtual cinematography are desperate for reality and efficiency of rendering of a crowd. This demands reduction of the complexity of the 3D scene and images [7-9]. The techniques on visualizing at algorithm levels have been developed over the years. In recent years, more computation power is provided by GPU [26].

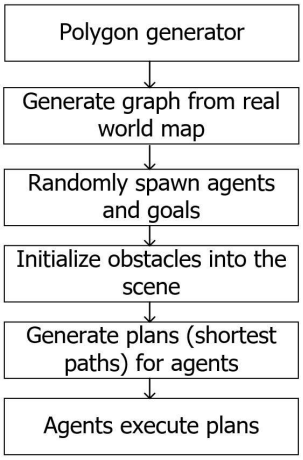
However, the overall performance of crowd simulation also depends heavily on the decision making part. Several techniques used for this purpose are based on a famous psychological model, BDI. BDI was extended to help model other agent platform [17]. BDI framework is extended to be used for planning and making decision in evacuation, based on general machine learning [16], reinforcement learning [14], deep reinforcement learning [18], integrate personality model and emotional model [10, 11]. With regards to low level navigation, many approaches have been proposed. Social force can be used to help simulate in high-density crowd [12, 13]. In addition to decision making, navigation at low level is also important. In agent-based crowd simulations, each agent merges many navigation algorithms for path-planning, oscillation prevention, collision avoidance, etc. Toll and Pettré [15] present topology-driven method for enhancing the navigation behavior. The method allows agents to synchronize their partial information to build global information for their individual plans. Gödel et al. [19] use variance-based sensitivity analysis using Sobol indices and then crosscheck the results by a derivative-based measure, the activity scores. Xing et al. [27] propose a method to improve the collision avoidance performance based on the existing crowd simulation and the example-based method chosen relies heavily on the data captured from the real world. Fraichard and Levesy [20] investigate for the extent the results obtained in the crowd simulation domain could be used to control a mobile robot navigating among people.

2.2. **Reciprocal Velocity Obstacle (RVO).** Navigation of agents has always been the first and foremost issue in crowd simulation. An outstanding requirement of simulating crowd is to avoid collision among agents. It is important that each agent remains occupying its own space that other agents must not eclipse into that particular space. Since we are interested in simulating moving agents simultaneously, taking account of moving information, i.e., the position, direction and speed of at least nearby agents, can be helpful. Berg et al. [1] propose RVO that considers other moving agents, or objects, as dynamic obstacles, and non-moving objects as static obstacles. Let $A = \{a_1, \ldots, a_n\}$ be a set of agents, residing in plane. An agent a_i is said to be at a reference position p_i, moving with a velocity v_i, preferred speed v_i^{pref}, towards the located goal g_i. Let $O = \{o_1, \ldots, o_p\}$ be a set of obstacles, each of which is stationed at position p. In order to move agents simultaneously, a simulation time step Δt must be decided. During each cycle of the simulation, the algorithm chooses the most appropriate values for each agent as its new status, i.e., speed, direction, position. The whole simulation process is repeated until each agent reaches its goal. Based on this foundation, more advanced simulation techniques [22-25] have been further developed. However, they are not considered of really high level decision making simulation.

2.3. **Belief-Desire-Intention (BDI).** Getting to know how our brain works, e.g., think, and decide, has long been an inspiring, important, mysterious, yet to be clarified, research issue. Among many models, BDI is a relatively new but influential principle. BDI was adopted to explain how intelligent agent, as a software governing body of robots, computer systems, etc., would work in forms of an agent language, AgentSpeak [2], e.g., Figure 1(a). Agents collect information about surrounding world in their belief, which could be a collection of data in forms of file, database, etc. Agents are given tasks to complete and are considered to be given goals for them to achieve. To achieve their goals, agents use data in their belief to generate plan accordingly. The dynamic and important parts of BDI agents are the ability to dynamically collect data and update belief, then update their plans accordingly. It is also very common that an overall plan can always be broken down recursively to sub-plan as long as it is needed.

```
1  initialize − state();
2  while until exit do
3  |    choices ←
   |   choice − generator(event − stack);
4  |    chosen − choices ← deliberate(choices);
5  |    update − intentions(chosen − choices);
6  |    execute();
7  |    get − new − external − events();
8  |    drop − successful − attitudes();
9  |    drop − impossible − attitudes();
10 end
```

(a) BDI interpreter



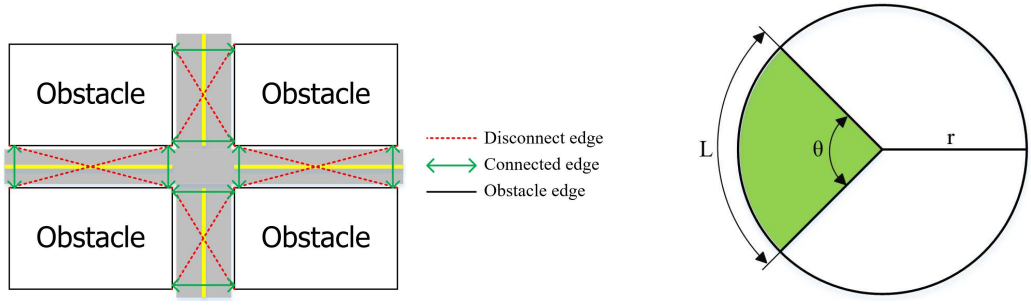
(b) Init-process

FIGURE 1. BDI interpreter and initialize process

3. **System Overview.** Figure 1(b) presents scene setting process before executing further simulation processes. Agents and initial plans are generated accordingly. During simulation, new plans and sub-plans will be generated.

3.1. **Process explanation.** Figure 1(b) presents six processes as follows. **Polygon generator** is the process to create static obstacles such as buildings, roads, fences, and walls, which are two-dimensional. We use our tools developed for drawing polygons. It provides a user interface to create obstacles by simply clicking a point on a satellite map image. In this step, the polygon data will be saved in JSON file for easy use in the simulator. **Generate graph from real world map** is the process to create a graph from the obstacle data in the previous process. Our graph algorithm repatedly considers each pair of obstacles, and creates a one-to-one connecting line from a corner of one obstacle, to a corner of the other obstacle. We check that these lines must not intersect other lines within the scene. Once corners are properly connected, we will achieve a new graph, that will be saved to a JSON file for an easy implementation in the next process. **Randomly spawn agents and goals** is the process to randomly spawn agents, and designated goals, into the scene. We use seed randomization to help create realistic scenes in the simulation. **Initialize obstacles into the scene** is the process to initialize settings for obstacles in the scene. We read the JSON polygon values in the first process. We transform the polygons in the previous process into two-dimensional coordination system. After we have done the transformation, obstacles are added to the simulator. **Generate plans (shortest paths) for agents** is an important process for assigning an initial path for each agent. We choose the path from the graph generated in the second process. **Agents execute plans** is a process of simulating agents movement. This process contains loops and recursive functions for computing agent plans, decision, collision and obstacle avoidance.

3.2. **Graph generation.** In this process, we connect appropriate points of polygons to create, within the scene, a large graph, on which agents can decide for their paths and route. In general, each polygon represents a static obstacle, e.g., a building, and a garden. The gap between these obstacles forms streets. Agents are not supposed to cross these obstacles but can only walk along their sides, on the footpaths or streets, if necessary. More importantly, we (human beings) typically cross streets at a corner or an assigned point, where we can get to the other side as quickly as possible. We hardly cross diagonally where the time it takes to get to the other side will be much longer and it will be very dangerous. Therefore, the algorithm, for each point of every polygon, will choose the nearest point of each of its neighboring polygons as the connecting path. This path must not cross any part of any polygon in the scene, e.g., Figure 2(a). Basically, it checks if any line segment intersects with others.



(a) Connect obstacle (b) Circular arc

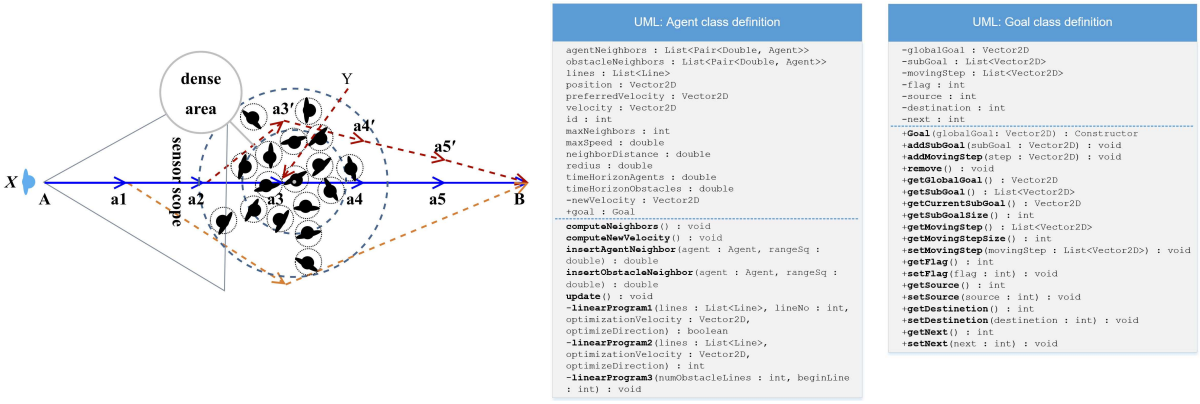
FIGURE 2. Connect obstacle (a) and circular arc sector (b)

4. **Agent Movement.** We bring about BDI model into our simulation to help increase decision making of our agents to be more similar to human beings because original RVO, although very efficient at guiding agents while avoiding collision, makes agents behaving like robots. In addition, BDI allows for agents to repeat their decision making according

to changing environment. Therefore, agents can nest their plans, having sub-plans when appropriate, in hierarchical manner, closer to how human beings behave.

4.1. **Minimizing scanning area with circular arcs.** As human beings, we plan our paths from our knowledge about available data. While we move on, the data we perceive from our eyes may trigger changes in our plans. We see the surrounding environment in limited area where our eyes are looking at. Typically, this area is a circular arc. In this research, we also take account of this manner, which also helps improve simulation efficiency by reducing scanning surrounding area, compared to RVO which scans all surrounding agents. As shown in Figure 2(b), given a circle, the area of a sector, bounded by an arc of length L and a pair of radii r separated by angle θ, is $A = \frac{r^2\theta}{2}$. Since the arc is on the angle θ, it is derived that the proportion of area A to the area of the circle and angle θ to the angle of the circle is $\frac{A}{\pi r^2} = \frac{\theta}{2\pi}$. Once canceling π on both sides, we have $\frac{A}{r^2} = \frac{\theta}{2}$. We multiply both sides by r^2, and we have the final result $A = \frac{1}{2}\pi r^2$. To make it work with 360 degree circle, we convert the proportion of the angle half θ with regards to 360 and we have $A = \frac{\alpha}{360}\pi r^2$. This area can help determine surrounding agents for making decision whether to change route or not.

4.2. **Agent planning.** Our agents make decision by integrating the scope sensor of agents with minimizing scanning area method. After the initialize process in (Section 3: System Overview), the agent begins execution plan process. The agent moves along a sub-path defined by the Dijkstra algorithm to reach a planned goal. In every simulation step, each agent has sensors to detect static obstacles and moving obstacles. We use the collision avoidance function based on the existing RVO with static obstacles. In this algorithm, our approach adds the agent's ability to move along a planned sub-path for the agent, allowing the agent to move quickly and similarly to real world movement. With regards to mobile obstacles, like other agents in the same scene, this process takes place when the agent's sensor detects other agents as shown in Figure 3(a). Therefore, when the agent can verify that from the original plan that the agent has to move through the density area, our agents use decision-making algorithms to avoid agent congestion, like Algorithm 1.



(a) Agent X (b) Agent class (c) Goal class

FIGURE 3. Simple scenario and agent, goal class data structure definition

Function **getAgentCondense** is used to detect the density ahead of the agent. Function 4.1 verifies if the area in the planned direction exceeds the specified threshold. If a point to avoid is achieved, the agent sub-plan is added to the stack. In the extreme case, on the condition that the agent cannot avoid the density area, e.g., Figure 3(a), we have a process of change path. When the speed of the agent per simulation step decreases as a result of calling the **getAgentVelocity** method, if the obtained value is less than

```
Function Decide(agentNo):
    Vector2D agentPos ← getAgentPosition(agentNo)
    Vector2D goalVector ← null
    if get(agentNo).getSubGoalSize() > 0 then
        goalVector ← getSubGoal(agentNo).get(0).subtract(agentPos)
    else
        goalVector ← getGlobalGoal(agentNo).subtract(agentPos)
    end
    double lengthSq ← goalVector.getNormSq()
    if lengthSq > 1.0 then
        goalVector ← goalVector.scalarMultiply(1/sqrt(lengthSq))
    end
    setAgentPreferredVelocity(agentNo, goalVector)
    condenseAgent ← getAgentCondense(agentNo)
    if condenseAgent > CONDENSE_THRESHOLD then
        if get(agentNo).getSubGoalSize() ⩽ 0 then
            angle ← getAngle(agentPos, getRandom(condenseAgent))
            distance ← getRandom(condenseAgent)
            goalX ← agentPos.getX() + distance ∗ cos(angle ∗ PI/180)
            goalY ← agentPos.getY() + distance ∗ sin(angle ∗ PI/180)
            Vector2D subGoal ← Vector2D(goalX, goalY)
            get(agentNo).add(subGoal)
            velocityAgent ← getAgentVelocity(agentNo)
            if velocityAgent < VELOCITY_THRESHOLD then
                status ← generatePlan(agentNo)
            end
        end
    end
End Function
```

Algorithm 1. Agent decides to avoid obstacle

TABLE 1. Threshold variable of each agent

Threshold variable	Description
CONDENSE_THRESHOLD	The number of other agents that are dense at the area detected by the agent scope.
VELOCITY_THRESHOLD	The speed of the agent at the simulation step.

velocity threshold, it immediately changes path to the goal. It calls the **generatePlan** method to create a new route for the agent. The agent decides to go to the left or to the right following argument: A_i argmin $= \sum_{n=1}^{detect}$; *left*, *right*.

4.3. **Planning strategy with sub-goal.** To understand our BDI agents better, consider the simple scenario shown in Figure 3(a) where agent X wants to walk-through from position A to B with crowded path. To illustrate how planning with sub-goal may work, Figure 3(a) provides an intuitive explanation. Agent X has a goal to move from position A, through a crowd of other agents, to position B. The original plan for X is to walk straight toward B. While moving closer to the crowded area, perceived information provides a couple of alternatives. The first one, while reaching $a1$, is to divert slightly to the right early on, avoiding the crowd completely, move towards $a3''$. Then X turns slightly to

the left and moves toward B. Another alternative, X keeps moving until it reaches $a2$, turns slightly to the left, and moves towards $a3'$, while going through parts of the crowd. The last choice is to keep moving straight as originally planned. The classes representing agent and goal are presented in Figures 3(b) and 3(c).

Figure 3(b) depicts the UML diagram for our agent class, modified from original RVO. It has typical agent setting properties, including id, position, maxNeighbors, maxSpeed, neighborDistance, radius, etc., and working functions for computing relative information for navigating agents. We add goal object variables for enhancing navigation ability. This helps navigate agents by containing more useful information including global plan, sub-plans, and other attributes. Figures 3(c) illustrates the goal class. It has a complex structure containing goal, sub-goals, data related to Dijkstra's shortest path algorithm. In each simulation round, the algorithm will update the plan in this class. When the agent detects that an obstacle has occurred and is affecting the originally planned movement, the agent decision-making and path planning process is activated. Once the route is obtained from the above process, the agent will update the plan and change its path.

5. **Experiment and Results.** We have implemented our agent based on both BDI and RVO, in order to improve multi-agent navigation. Our benchmark is an implementation of the traditional RVO, namely RVO2. We experiment our approach in three scenarios: **Simple scenario:** Figure 4 for simulating path of agent to see trajectory and dense position on Section 5.1. **Block scenario:** Figure 5 for blocks simulating result, in this experiment, we have four groups of 25 agents in each angle of the environment moving to the negate angle. In the middle, there are four static obstacles that form narrow passages. The groups with negate goal directions meet in the narrow passage on Section 5.2. **Large space scenario:** in Section 5.3, we simulate on 2D and 3D scenarios. In the 2D simulation, we have used satellite image to create graphs with our tools for simulating a large number of agents to get a good overview of the simulation. In the 3D simulations, we simulate our agents in Unity3D and build a virtual Bangkok city for simulating a large number of agents, which are very challenging to simulate.

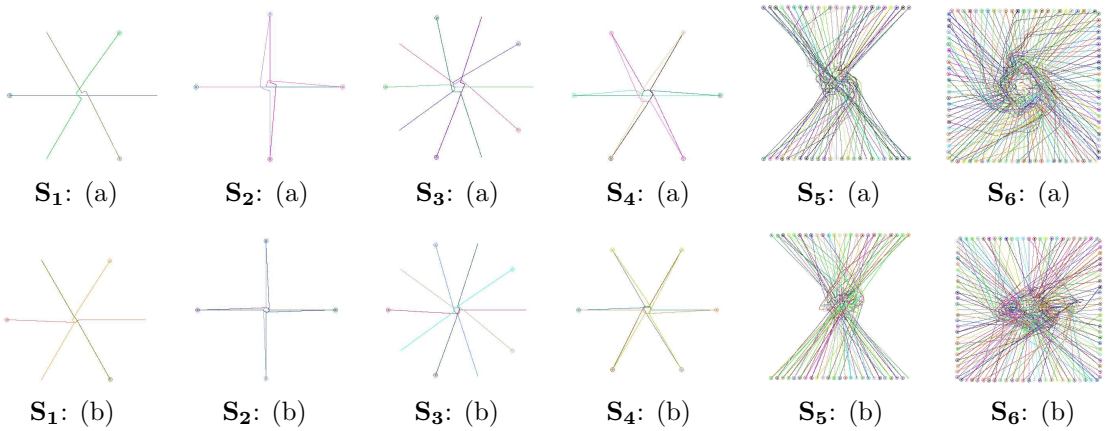


FIGURE 4. Settings: $\mathbf{S_1}$, $\mathbf{S_2}$, $\mathbf{S_3}$, $\mathbf{S_4}$, $\mathbf{S_5}$ and $\mathbf{S_6}$ scenarios to exchange negate position and resulting paths in the scenario using our approach (a) and RVO2 approach (b)

5.1. **Simple scenario.** For the first set of experiments, we use mini-scenes to evaluate moving paths and avoid high-density areas of agents. We set simulation step at 0.25 seconds in the simulator. We set each agent's properties, such as maximum speed, and radius, to the same values for all experiments. In this simple scenario, the agent is targeting the opposite position to create a density at the center of the scene. We have six mini scenes. Scenes S_1, S_2, S_3, S_4, S_5 and S_6 contain 3, 4, 5, 6, 50 and 100 agents.

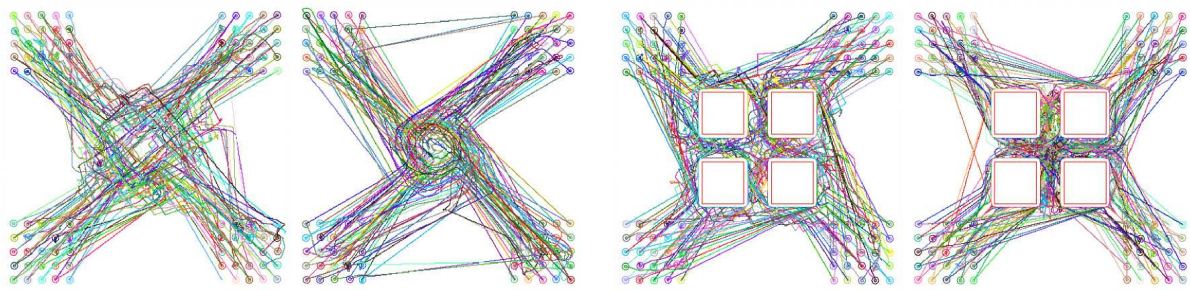
The results are shown in Figure 4. In each scene, we compare the result of our approach, appearing on the above figure, with that of traditional RVO2, appearing on the bottom figure. Since we enhance agent ability by adding BDI principles so that agents can be aware of their surroundings, as well as enhancing their route planning capabilities, the agent performance and overall performance of the simulation improve. In each scenario, our agents can avoid tight congested area (left figures), while traditional RVO tends to leading agents into tight congested area. Note that our approach allows for agents to plan with flexibility how early or far away the agent wants to avoid the congested area. In Figure 4, we show the trajectory of agents paths. Our method allows agents to avoid congestion before reaching it. We choose to show scenarios where the decision is to "just" avoid to tight congested area. For execution time, our approach was able to reach goal with less time to complete the scene than traditional approach. In the first scene, the agents were able to reach similar targets, while the second, third, fourth, fifth and sixth scenes clearly distinguished themselves. As the number of agents grows, the traditional approach is momentarily deadlocked as the agent moves towards the center of density. It is clear from the sixth scene that the agent group of our approach has a walking path in which the density avoidance occurs. Whereas the traditional approach agents are pushed, as shown in the simulation time in Table 2.

TABLE 2. Simple scenario: Execution time (sec) and simulation time (step)

Setting / Method	Execution time						Simulation time					
	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**
RVO2	452	1923	4482	516	1894	6443	362.5	14654.25	37176.5	441.75	816.25	1205.0
Our approach	457	474	486	507	1590	3983	362.5	371.0	377.25	376.0	655.0	710.25

Table 2 shows execution time of simple scenario. Our approach can navigate agents towards their goals with much less time than the traditional approach. In the first scene, the agents were able to reach similar targets, while the second, third, fourth, fifth and sixth scenes clearly distinguished themselves. As the number of agents grows, the traditional approach is momentarily deadlocked as the agent moves towards the center of density. It is clear from the sixth scene that the agent group of our approach has a walking path in which the density avoidance occurs. Whereas the traditional approach agents are pushed, as shown in the simulation time in Table 2. Based on Table 2 simulation time, our approach was able to reach goal with less time to complete the scene than traditional approach, and these results are similar to the results of the execution time.

5.2. **Block scenario.** For the second benchmark of this scene, we conduct our experiment by splitting agents into four groups, each of which is stationed at a corner of a simulation square. The agents then move diagonally towards the opposite corner as their located goals. As the agents move closer to the center, it becomes a tight congested area. We call this block scene. We begin our experiment with a set of 100 agents, i.e., 25 agents for each group. Figure 5(a) shows the trajectory paths of agents following our approach (a) and that of agents following traditional approach (b). As clearly shown, our approach can lead agents to avoid congestion area in the middle much better, i.e., the trajectory paths of our approach are spread out more evenly, while those of traditional approach are packed tightly in the middle. Figure 5(a)(1) shows that the majority of the agents avoid expected density points with noticeable deviation. Figure 5(a)(2), on the other hand, shows that there is a density of agents around the center because agents move towards it. The execution time in Table 3 shows that it takes 59.11 percent less time to run simulation results than the traditional approach. The simulation time of both scenes shows that our approach takes less time to simulate than the traditional approach. The difference was 633.25 in simulation step.



(1) (2) (1) (2)

(a) Clear (b) With obstacles

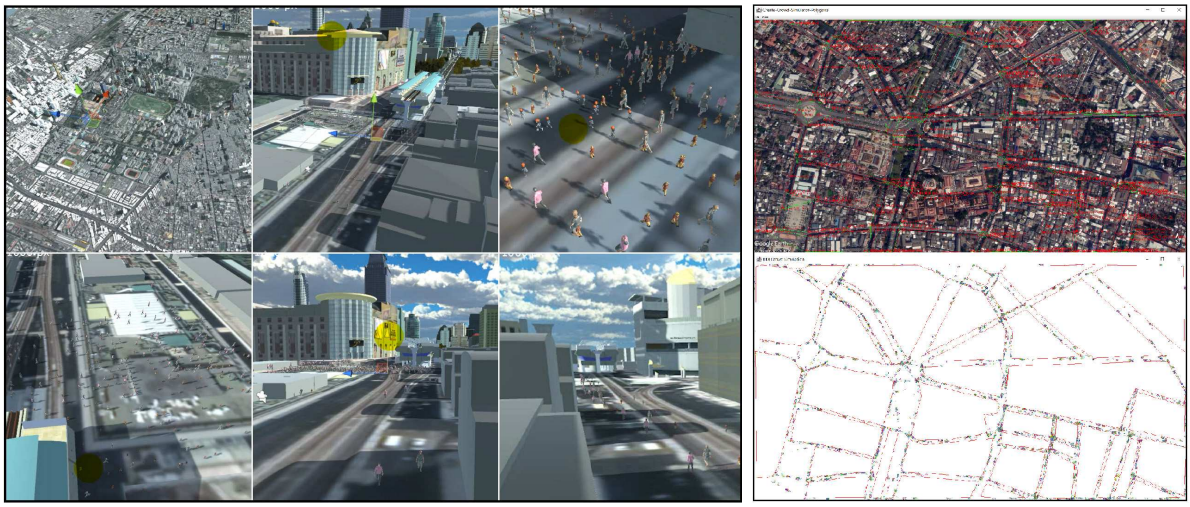
FIGURE 5. The resulting paths in the *block* scenario using our approach (1) and the RVO2 approach (2)

TABLE 3. Block scenario: Execution time (sec) and simulation time (step)

Setting / Method	Block		Block with obstacles	
	Execution time	**Simulation time**	**Execution time**	**Simulation time**
RVO2	7410	1449.5	11542	23122.25
Our approach	4380	816.25	9329	1677.75

To challenge our approach even further, we added some obstacles within the simulation scenes, while the number of agents, their origins and destinations are similar to the settings of previous scene. As shown in Figure 5(b), the scene consists of four obstacles. As done in previous experiment, we decrease the deviation points until the difference between our approach and the traditional one has minimal difference. As shown in Figure 5(b)(1) and 5(b)(2), we can see that the paths of the agents are not very different from each other. With respect to the execution time, Table 3 shows that our approach takes 80.83 percent less time to execute than the traditional approach. With respect to the simulation time, it shows that there was a significant difference between both approaches. When the simulations ended, our approach takes $23122.25 - 1677.75 = 21444.5$ seconds. Note that each simulation step takes 0.25 second. From this simulation in the obstacle-free scenes and the scenes with obstacles, we found that our approach performed much better in terms of execution time than traditional approach and in terms of simulation time as well.

5.3. **Large space scenario.** Next, we carry out experiments of large space urban in 2D space. As shown in Figure 6(b), we take a satellite image of the Democracy Monument, Bangkok, Thailand, as our scene. In recent history of Thailand, the area has been a remarkable scene for crowd simulation because there were so many real incidents took place there. We then use our tool and algorithm to create polygons of hundreds of building blocks. The algorithm for connecting points, using existing blocks, each of which consists of vertices and edges, to create graphs, from which agents can generate path for navigating themselves from origins to destinations, is applied. The created scenes can be saved to a file in JSON form, be reloaded, be modified and be saved back to (another) file again. Here, we randomly choose different positions as origins, from where agents will be spawned, and goals, to which agents will be destined. Figure 6(b) bottom shows a snapshot of a simulation results. In addition to 2D scene, we also carry out experiments on 3D scene. Instead of focusing on a particular area, we create a 3D world, covering almost three quarters of Bangkok area. Thousands of 3D buildings were originally created as polygon objected and 2D textures, looking very similar to real buildings, were then applied. The

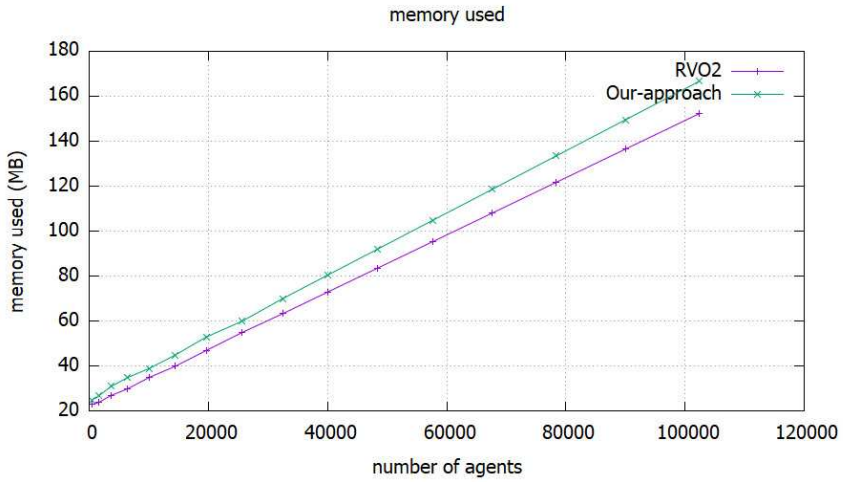


(a) 3D visualization of different scenes (b) 2D graph and result

FIGURE 6. Large space scenario with real-world map

simulation is carried out under Unity3D environment. Extra work was needed to convert Java source codes to C#. The creation of virtual 3D Bangkok is still going and improving. We manage to conduct our experiments on another historic area, namely, World Trade Center. As shown in Figure 6(a), ten thousands of agents are simulated in this scene.

5.4. **Performance and limit of algorithm.** We carried out our experiments on a typical desktop computer, featuring a 3.60 GHz Intel Core i7-4790 processor 4 core 8 thread of CPU, 16 GB 1600 MHz DD3 of RAM and AMD REDEON RX VEGA 64 of GPU, and the operating system is Windows 10 x64. We measure capabilities and limits of our framework in terms of average simulation speed (execution time) and visualization (Frames per Second: FPS). It is very important that visualization is realistic. As it is widely adopted in movies industry, the minimal acceptable rate is 24 FPS. The results are shown in Figure 7 (Table). Since the specification of our test box is relatively low and we also visualize the simulation in real time, the results are not very impressive. With thousands of agents being simulated, the visualization becomes very slow. The GPU manages to skip some parts of the simulation and visualize only some other parts of the simulation. Given this limit, we find that the bearable case, where the skipped part of simulation and the visualization are not disconnected, is with 40000 agents.



No. Agents	Exc.T (sec)	FPS (sec)
400	7.865	34.359822
1600	42.174	6.9504908
3600	121.266	2.7978576
6400	278.232	1.4934659
10000	495.469	0.9005407
14400	774.778	0.5994943
19600	1155.896	0.4339231
25600	1680.735	0.3218949
32400	2676.726	0.1989315
40000	**3877.317**	**0.1618606**
48400	5935.897	0.1171946
57600	8310.373	0.0976779
67600	11108.595	0.071176
78400	14121.182	0.0619336
90000	15099.398	0.0668023
102400	19959.877	0.0474412

FIGURE 7. Memory used: Block scenario without obstacle

Given above limits, there are two cases for considering limits of our framework, namely interactive and non-interactive. For the former, the execution time must be less than the simulation time so that the visualization can be done in timely fashion. For the latter, the execution time can be greater than the simulation time as long as it is acceptable. Acceptability for this problem varies depending on urgency of result needs. Since we do not demand immediate results but we cannot wait for too long, we accept delays within matters of hours. In many cases, the limit of algorithm is the hardware, i.e., the memory. In our experiments, we also check the limits on both visualization and execution. For visualization, the algorithm is 24 FPS for 600 agents. The number of frames per second drops rapidly when the number of agents rises from 19600 to 102400 agents. The rendering is very low, 0.4 to 0.04 FPS, when the number of agents is larger than or equal to 19600. In terms of memory usage, our algorithm consumes slightly more memory than RVO2. Both algorithms consume more memory when the number of agents is increased almost linearly. In our experiment, we have 1 GB of RAM and can carry out simulation up to 720000 agents.

6. **Conclusion.** We have developed a framework for crowd simulation based on BDI concept. Our framework allows for carrying out up to 40000 agents on a typical desktop computer. We embrace BDI, as a high-level decision-making mechanism, with RVO, as a low-level navigation mechanism, providing an efficient, from system performance point of view, and realistic, from simulation point of view, system. On one extreme end, we experiment our framework for collision avoidance in multiple extremely tight scenarios. The results show that the framework can simulate the crowds successfully. Simulation time is short enough, providing more time for realistic visualization. On the other end, we deploy our framework with a large scene allowing for tens of thousands of agents to be successfully simulated. In the future, our framework can be improved on more aspects. The first one is to handle more agents in the scene. The second one is to improve the visualization. Lastly, the internal algorithm for planning can also be improved.

REFERENCES

[1] J. V. D. Berg, L. Ming and D. Manocha, Reciprocal velocity obstacles for real-time multi-agent navigation, *2008 IEEE International Conference on Robotics and Automation*, pp.1928-1935, 2008.

[2] A. Rao and M. Georgeff, BDI agents: From theory to practice, *ICMAS*, 1995.

[3] D. Thalmann, Crowd simulation, *Encyclopedia of Computer Graphics and Games*, pp.1-8, 2016.

[4] J. Comptdaer, E. Chiva, S. Delorme, H. Morlaye, J. Volpoët and O. Balet, Multi-scale behavioral models for urban crisis training simulation, *The 16th Conference on Behavior Representation in Modeling and Simulation (BRIMS)*, Norfolk, Virginia, 2007.

[5] G. Drettakis, M. Roussou, A. Reche and N. Tsingos, Design and evaluation of a real-world virtual environment for architecture and urban planning, *Presence: Teleoperators and Virtual Environments*, vol.16, no.3, pp.318-332, 2007.

[6] S. Gwynne, E. Galea, M. Owen, P. Lawrence and L. Filippidis, A review of the methodologies used in the computer simulation of evacuation from the built environment, *Building and Environment*, vol.34, no.6, pp.741-749, 1999.

[7] F. Tecchia, C. Loscos and Y. Chrysanthou, Visualizing crowds in real-time, *Computer Graphics Forum*, vol.21, no.4, pp.753-765, 2002.

[8] D. Thalmann, H. Grillon, J. Maim and B. Yersin, Challenges in crowd simulation, *2009 International Conference on CyberWorlds*, pp.1-12, 2009.

[9] J. Maïm, B. Yersin and D. Thalmann, Unique character instances for crowds, *IEEE Computer Graphics and Applications*, vol.29, no.6, pp.82-90, 2009.

[10] C. Li, P. Lv, D. Manocha, H. Wang, Y. Li, B. Zhou and M. Xu, ACSEE: Antagonistic crowd simulation model with emotional contagion and evolutionary game theory, *IEEE Trans. Affective Computing*, 2019.

[11] Y. Mao, Z. Li, Y. Li and W. He, Emotion-based diversity crowd behavior simulation in public emergency, *The Visual Computer*, vol.35, no.12, pp.1725-1739, 2019.

[12] D. Helbing and P. Molnar, Social force model for pedestrian dynamics, *Physical Review E*, vol.51, no.5, 4282, 1995.
[13] H. Kolivand, M. Rahim, M. Sunar, A. Fata and C. Wren, An integration of enhanced social force and crowd control models for high-density crowd simulation, *Neural Computing and Applications*, vol.33, no.11, pp.6095-6117, 2021.
[14] Q. Wang, H. Liu, K. Gao and L. Zhang, Improved multi-agent reinforcement learning for path planning-based crowd simulation, *IEEE Access*, vol.7, pp.73841-73855, 2019.
[15] W. Toll and J. Pettré, Synchronizing navigation algorithms for crowd simulation via topological strategies, *Computers & Graphics*, vol.89, pp.24-37, 2020.
[16] S. Lee, Y. Son and J. Jin, An integrated human decision making model for evacuation scenarios under a BDI framework, *ACM Trans. Modeling and Computer Simulation (TOMACS)*, vol.20, no.4, 2010.
[17] D. Singh, L. Padgham and B. Logan, Integrating BDI agents with agent-based simulation platforms, *Autonomous Agents and Multi-Agent Systems*, vol.30, no.6, pp.1050-1071, 2016.
[18] S. Zheng and H. Liu, Improved multi-agent deep deterministic policy gradient for path planning-based crowd simulation, *IEEE Access*, vol.7, pp.147755-147770, 2019.
[19] M. Gödel, R. Fischer and G. Köster, Sensitivity analysis for microscopic crowd simulation, *Algorithms*, vol.13, no.7, p.162, 2020.
[20] T. Fraichard and V. Levesy, From crowd simulation to robot navigation in crowds, *IEEE Robotics and Automation Letters*, vol.5, no.2, pp.729-735, 2020.
[21] Z. Yao, G. Zhang, D. Lu and H. Liu, Learning crowd behavior from real data: A residual network method for crowd simulation, *Neurocomputing*, vol.404, pp.173-185, 2020.
[22] J. V. D. Berg, S. Patil, J. Sewall, D. Manocha and M. Lin, Interactive navigation of multiple agents in crowded environments, *Interactive 3D Graphics and Games (I3D)*, pp.139-147, 2008.
[23] J. Snape and D. Manocha, Navigating multiple simple-airplanes in 3D workspace, *2010 IEEE International Conference on Robotics and Automation*, pp.3974-3980, 2010.
[24] J. Snape, S. Guy, J. Van Den Berg and D. Manocha, Smooth coordination and navigation for multiple differential-drive robots, *Experimental Robotics*, pp.601-613, 2014.
[25] S. Kim, S. Guy, D. Manocha and M. Lin, Interactive simulation of dynamic crowd behaviors using general adaptation syndrome theory, *Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp.55-62, 2012.
[26] L. Diaz, I. Rivas, K. Rodriguez and I. Rudomin, Crowd data visualization and simulation, *Procedia Computer Science*, vol.139, pp.622-629, 2018.
[27] W. Xing, L. Zhu, X. Wei and P. Bao, Collision avoidance approach for example-based crowd simulation, *International Journal of Innovative Computing, Information and Control*, vol.14, no.1, pp.127-145, 2018.