Multi-Agent Continuous Motion Simulation using Discrete Events

Jack Rosenthal
Colorado School of Mines
Computer Science Department
jrosenth@mines.edu

Qin Yang
Colorado School of Mines
Computer Science Department
qinyang@mines.edu

***Abstract*—Multi-agent systems with continuous motion (such as swarm robotics systems) are often simulated in a time-step agent-based simulation system due to the relative ease of implementing an emergent behavior in agent-based simulations. The disadvantage of this approach is that simulation accuracy is proportional to the size of the time-step in the system, and as the size of the time-step decreases, the computational time required to run the simulation increases.**

Our paper proposes a modified discrete event simulation technique for simulating many agents operating in the same continuous space. We do so by adding two new techniques to a discrete event simulation we call *cross-cutting* and *cancellation*. Finally, we provide an evaluation of the computational complexity of our modified discrete event simulation technique when compared to time-step agent-based simulations through a simple swarm robotics scenario.

I. Introduction

Simulating continuous motion of many agents in the same space is non-trivial to implement, as the motions of one agent may depend on the motions of the surrounding agents. Often times, the system may be thought of using emergent behaviors; that is, each agent follows a simple set of rules in order to accomplish a behavior much greater than the sum of its parts.

Emergent behaviors are easy to implement in a time-step agent-based simulation, as the individuals behaviors can simply be encoded as a set of predicates and the resulting transition to the agent at each time step. While this model has the advantage of easy implementation and trivial verification, it is not very computationally efficient as the computational time required is proportional to the amount of steps in the simulation.

Certain multi-agent systems *need* to be encoded using a time-step agent-based simulation. For example, consider Conway's Game of Life [1], a cellular automaton: each agent fundamentally may only make a plan for the next step in the simulation by the very nature of the game. But in some systems, agents may plan for a continuous motion and only adjust their plan when they need to. For example, a driver of a vehicle may have their entire route planned ahead of time, but may have to make changes based on their simple rules for driving on the road (avoid accidents with other vehicles, don't drive on closed roads, etc.) For the latter kind of simulation, we can save computational time and improve simulation accuracy by implementing as a discrete event simulation rather than a time step model, at the cost of reduced implementation ease, and significantly reduced ease of verifying the computational model.

This paper extends the traditional discrete event simulation model by adding two new techniques to support continuous motion simulation with multiple agents:

1) Events may now be *cancelled*. In terms of a typical discrete event simulation software system, this means that previously scheduled events may be removed from the event queue (or are otherwise prevented from being processed) before they are processed.
2) Events must provide *cross-cutting*. In terms of a typical discrete event simulation software system, events which interfere with the completion of other events must trigger the cancellation and rescheduling to occur.

Our paper implements these techniques for modeling continuous motion of multiple agents under the following technique:

1) When a simulation agent A makes an initial plan, they schedule an event E_{A1} indicating the planned time of arrival at the desired position and create continuous functions which will represent their position and distance traveled with respect to the simulation clock as they move to that position.
2) When a simulation agent B executes an event E_{B1} which cross-cuts (prevents) A's plan from completing as intended, simulation agent A will devise a new plan, evaluating their previously derived continuous position and distance functions, updating their distance traveled accordingly, and derives new position and distance traveled functions from their previously evaluated position. A will then schedule the new event E_{A2} indicating their new plan for arrival.

Figure 1 shows a scenario derived from above example in a picture.

The remainder of the paper is organized as follows: Section II discusses related work in the field of both simulation, multi-agent systems, and robotics, section III formally defines the problem we are trying to solve, sec-

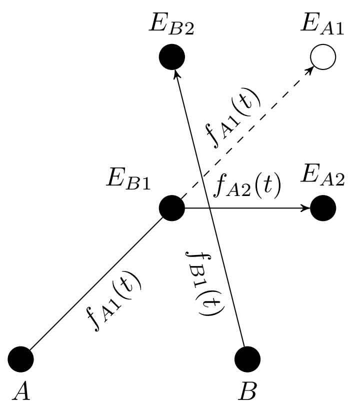

Fig. 1. Initially, agent A plans on arrival at the point indicated by E_{A1} with continuous travel function $f_{A1}(t)$. Then, agent B schedules travel to the point indicated by E_{B2}, but realizes this will cause collision with A, and A would notice this at time t_c. B then schedules E_{B1} for time t_c to indicate to A to change the plan of travel. When E_{B1} is executed, A then evaluates $f_{A1}(t_c)$ to determine its position, and creates a new travel function $f_{A2}(t)$ indicating the planned travel to the point indicated by E_{A2}. Presumably, at this point A may wish to schedule another event to reach its initial desired position.

tion IV defines a generalized simulation technique for implementing a discrete event simulation with multiple agents in motion through a continous space, and section V evaluates our simlulation technique using a scenario in swarm robotics. Finally, section VI draws conclusions from our work.

II. Related Work

Swarm robotics in a new and rising field, and many simulators for swarm robotics systems have been developed. For example, ARGoS [2], Gazebo [3], USARSim [4], Webots [5], and MuRoSimF [6] are all simulators for multi-robot systems. However, all of these simulators use a time-step agent-based model for their simulations. Our computational model is implemented using a discrete event simulation (see [7, p. 3] for the definition of a discrete event simulation), which is a fundamentally different simulation model than all of the previously mentioned simulators. To our knowledge, we are the first to simulate a multi-agent robot system using a discrete event simulation.

Cancellation is not a new technique in discrete event simulations. [8] includes canceling edges in their event graph theory, and even [9] and [10] both show how canceling events can be eliminated from a simulation model: cancellation is merely a convenience from a software standpoint. To our knowledge, we are the first to take advantage of cancellation in discrete event simulations to make multi-agent continuous motion possible.

III. Problem Statement

Our problem can be formally defined as followed:

> Given N ($N \geqslant 2$) agents in a shared k-dimensional ($k \geqslant 2$) coordinate system, for which the agents preform some amount of planning that may change, devise a generalized technique used to implement a discrete event simulation of the agents motion in the continuous space.

In addition, we will discuss technique to prevent collision of agents in the space, although this is not always needed (for example, a colony of ants often times will climb on top of each other, so the technique allows for simulation of agents directly on the same point).

IV. Simulation Technique

In order to discuss the techniques for writing a discrete event simulation of multiple agents in a continuous space, we must first define the common action of the agents: *travel.* While in the real world, travelling and planning algorithms may be implemented continuously, or in a time-step based simulation from step-to-step, we must extend the traditional definitions of travel to support usage in a discrete event simulation:

> **Travel:** travel is the act of *planning a complete path* from one position (the start) to another (the goal), and attempting to execute that plan at least partially, completing the plan unless another reason arises.

For an agent to preform travel, they must take advantage of our *cross-cutting* method. Our cross-cutting method is outlined in Algorithm 1.

Algorithm 1 Cross-cutting Travel Algorithm

```
procedure InitializeAgent($A, A_i$)
    ▷ The agent must have an initial cross-cutting function when it is created
    $p_A(t) \leftarrow t \mapsto A_i$
end procedure
procedure Travel($A, A_n, c, P$)
    ▷ Agent $A$ plans travel to point $A_n$ at time $c$ with planning function $P$
    ▷ Before updating the cross-cutting function, the simulation may want to make note of other intermediate variables, such as distance traveled
    $p_A(t) \leftarrow P(p_A(c), A_n)$
    for all $B$, where $B$ is an agent and not ourselves do
        Symbolically determine if $B_p(t)$ has an intersection with $A_p(t)$
        Schedule cancellation and notice events appropriately
    end for
end procedure
```

A. Planning Considerations

While Algorithm 1 is sufficient to prevent collision, it always considers the agent who scheduled travel after another to have the precedence in collision avoidance.

In order to avoid rescheduling the initial agent's travel, the planning algorithm used may wish to consider the travel of the other agents in the system and plan around some (or all) of their routes. Taking this into account, the precedence of the agents in the system can be considered in any order.

V. Evaluation: Using a Simple Swarm Robotics Scenario

To evaluate the effectiveness of our generalized simulation approach, we implement a simple scenario in swarm robotics to simulate using our techniques:

- **Robots** are the agents in the system. Each robot has a continuous 2-dimensional position, and maintains an internal state of the distance traveled.
- **Tasks** appear at specified times and locations. Each task has a defined **radius**, which specifies the distance at which the task may be circled in order to preform work on it.
- When a task appears, a selected set of robots goes to the task to preform work on it. When all robots selected are at the task, they begin circling around it. The selection function can be any partitioning function, but we implement as a simple *split into k partitions based on robot id* algorithm[1]

We define the following event types to handle this simulation:

- `RobotCreated`: A new robot has appeared in the system, and its initial cross-cutting function has been set.
- `TaskCreated`: A new task has been created. This event causes the robots to partition and spawns `BeginTravelToTask` events for the robots who travel to the task.
- `BeginTravelToTask`: The robot has begun travel to a task, and has scheduled an `ArrivalAtTask` for when it plans to arrive. The cross-cutting function is evaluated to determine current position, and the internal distance measure of the robot is updated accordingly. This event has the ability to cancel events, as the robot may not be travelling to where it was originally.
- `ArrivalAtTask`: The robot has arrived at the task, and is ready to begin work. If all robots have arrived at the task, a `TaskBegin` event is created. This event may be cancelled, as simulation agents may begin travel to another task once they are already on the way to the task.
- `TaskBegin`: All robots have arrived at the task, and work begins on the task. When this happens, the cross-cutting function of the robot has been updated to reflect their motion in a circle around the task. This event may be cancelled for the same reasons as `ArrivalAtTask`.

[1]This approach may seem too simple, but we are using this to test and evaluate the effectiveness of our simulation technique rather than of our planning technique. If one were using our simulation technique to implement a novel planning algorithm, presumably, this approach would be better.

Figures 2, 3, and 4 show a visualization of the various parts of the simulation.

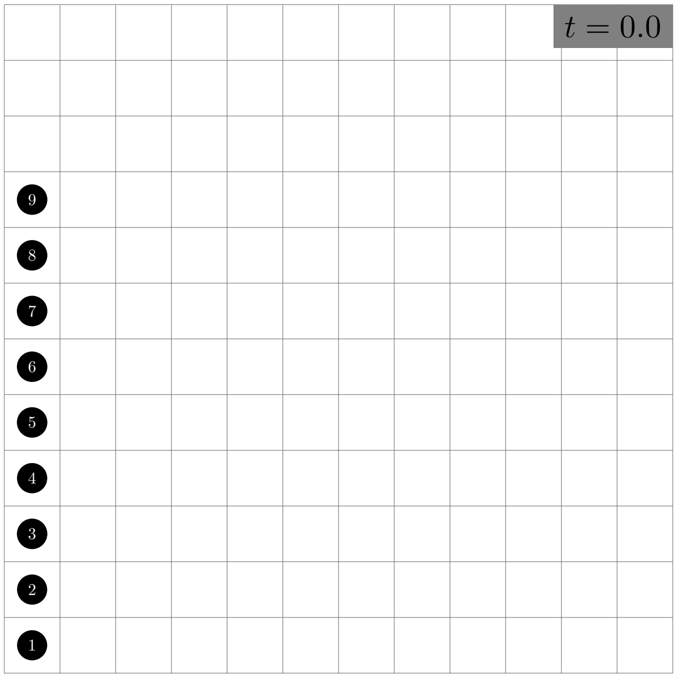

Fig. 2. At the beginning of the simulation with $N = 9$ robots, the robots start on the left hand side.

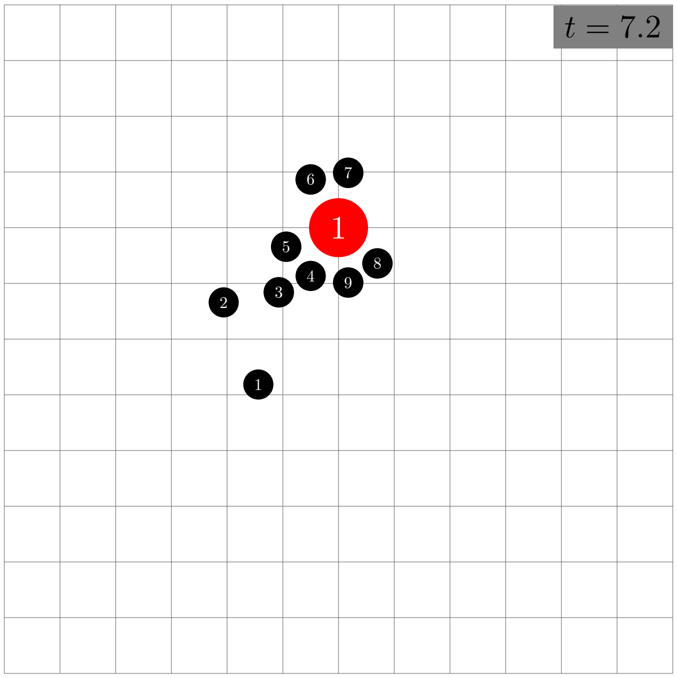

Fig. 3. At $t = 1$, task 1 appears, and the robots navigate to the task.

We implemented the simulation in Python 3.6 and used Python's `heapq` module for our underlying event queue data structure. The simulation software can be obtained at the following URL:

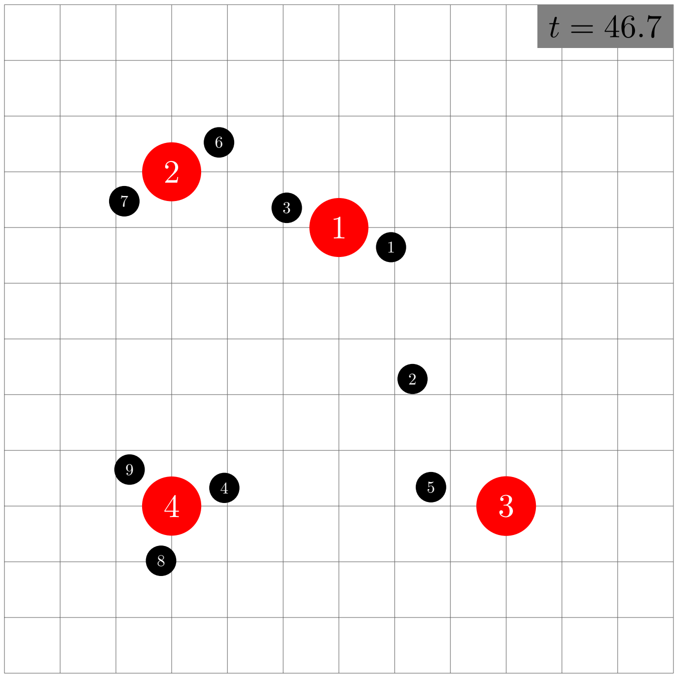

Fig. 4. As new tasks appear, the robots partition themselves amongst the tasks.

https://github.com/jackrosenthal/swarm-des

A. Performance Evaluation

The discrete event simulation technique will outperform a time-step based technique (arbitrarily) when large intervals are placed between different tasks, as the time-step simulation has to simulate the intermediate time intervals. But how does the simulation preform when given closely-spaced tasks (resulting in a large number of canceled events)?

To answer this question, we tested with 2 tasks placed at the same time. This causes the robots to initially schedule the first scheduled task, then half of the robots will have to cancel an event and go to the second task instead.

Tests were preformed on Intel® Core™ i7-6700K CPU at 4.00 GHz running Linux. Results for this selected set of parameters can be seen in Table I. Note that these selected set of parameters was an edge case designed to cause many cancellations and is not representative of the overall performance of the simulation.

TABLE I
Overlapping Tasks: Performance

Robots	**Execution Time** (seconds)
50	0.06
100	0.09
500	0.36
1000	1.13
2000	4.24
5000	18.61

The notable decay in performance is a result of the large heap event queue with cancelled events sitting in it. Performance for this edge case could be increased by switching to a more efficient data structure for event list management (such as *Henriksen's Algorithm* [11], a hybrid data structure overlaying a binary search tree on a linked list) or even eliminating cancellation from the simulation model [9].

VI. Conclusion

Our work introduces discrete event simulation to the simulation of continuous multi-agent systems, a field which previously only implemented simulations using time-step based models. By doing so, implementers of simulations are able to simulate their models for time scales which would be infusible in a time-step simulation.

Implementing in a discrete event simulation does have a number of drawbacks:

1) Developing the computational model requires significantly more effort: extra care must be taken to developing appropriate planning algorithms for the simulation rather than simply encoding the agent's logic for each time-step.
2) Verification of the computational model is no longer trivial as it is with time-step agent-based simulations. We verified our computational model using a visualization to ensure that the cross-cutting functions behaved as we thought we had made them. Developing better verification procedures for computational models of these systems could be further work in this area.

Simulation is important to the field of robot planning. By implementing motion planning algorithms in simulation (especially stochastic algorithms), the performance of the planning algorithm can be evaluated without the need for many expensive robots. By implementing in a discrete event simulation, time scales which would be infeasible in traditional time-step simulations become computable, at the cost of reduced ease of implementation and verification.

References

[1] M. Gardner, "The fantastic combinations of john conway's new solitaire game life," *Scientific American*, vol. 223, pp. 120–123, 1970.
[2] C. Pinciroli and V. Trianni, "Argos: A modular, multi-engine simulator for heterogeneous swarm robotics," *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.
[3] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 2149–2154, Apr. 2004.

[4] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, “Usarsim: A robot simulator for research and education,” pp. 1400–1405, May 2007.

[5] O. Michel, “Cyberbotics ltd. webots™: Professional mobile robot simulation,” *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004. DOI: 10.5772/5618. [Online]. Available: https://doi.org/10.5772/5618.

[6] M. Friedmann, K. Petersen, and O. Von Stryk, “Simulation of multi-robot teams with flexible level of detail,” *Transactions of The Society for Modeling and Simulation International - SIMULATION*, vol. 5325, pp. 29–40, Nov. 2008.

[7] L. M. Leemis and S. K. Park, *Discrete Event Simulation: A First Course.* 2006, ISBN: 9780131429178.

[8] L. Schruben, “Simulation modeling with event graphs,” *Commun. ACM*, vol. 26, no. 11, pp. 957–963, Nov. 1983, ISSN: 0001-0782. DOI: 10.1145/182.358460. [Online]. Available: http://doi.acm.org/10.1145/182.358460.

[9] E. L. Savage and L. W. Schruben, “Eliminating event cancellation in discrete event simulation,” in *Proceedings of the 27th Conference on Winter Simulation*, ser. WSC ’95, Arlington, Virginia, USA: IEEE Computer Society, 1995, pp. 744–750, ISBN: 0-7803-3018-8. DOI: 10.1145/224401.224722. [Online]. Available: http://dx.doi.org/10.1145/224401.224722.

[10] R. G. Ingalls, D. J. Morrice, and A. B. Whinston, “Eliminating canceling edges from the simulation graph model methodology,” in *Proceedings of the 28th Conference on Winter Simulation*, ser. WSC ’96, Coronado, California, USA: IEEE Computer Society, 1996, pp. 825–832, ISBN: 0-7803-3383-7. DOI: 10.1145/256562.256821. [Online]. Available: http://dx.doi.org/10.1145/256562.256821.

[11] J. O. Henriksen, “Event list management: A tutorial,” pp. 543–551, Jan. 1983.