Analysing the Performance of Optimistic Synchronisation Algorithms in Simulations of Multi-Agent Systems

Michael Lees, Brian Logan,
School of Computer Science and Information Technology
University of Nottingham, UK
{mhl,bsl}@cs.nott.ac.uk

Chen Dan, Ton Oguara and Georgios Theodoropoulos
School of Computer Science
University of Birmingham, UK
{cxd,txo,gkt}@cs.bham.ac.uk

Abstract

In this paper we present a detailed analysis of the performance of the Decision Theoretic Read Delay (DTRD) optimistic synchronisation algorithm for simulations of Multi-Agent Systems. We develop an abstract characterisation of the access patterns found in MAS simulations based on the simulation's degree of coupling and skew. Using this characterisation, we generated stereotypical test cases which we used to compare the performance of the DTRD algorithm with that of Time Warp and time windows. To determine if the test cases reliably predict performance in a real agent simulation, we compared the predictions made by the test cases with performance results from the Boids agent simulation benchmark for a range of simulation parameters. The results indicate that DTRD adapts to the mixtures of coupling cases found in real agent simulations and is capable of tracking changes in coupling during the simulation.

1. Introduction

The simulation of agent systems has traditionally played an important role in agent research and development. Simulation allows a degree of control over experimental conditions and facilitates the replication of results in a way that is difficult or impossible with a prototype or fielded system, freeing the agent developer or researcher to focus on the key aspects of a system. Simulation has been applied to a wide range of MAS research and design problems from models of complex individual agents employing sophisticated internal mechanisms to models of large scale societies of relatively simple agents which focus more on the interactions between agents, e.g., [1, 13, 4, 12].

However, despite its wide application, the most appropriate simulation technique for different kinds of MAS simulation problems is often unclear. There are many different degrees of variation in MAS, and an algorithm or simulator which works well in one case may not work well in others. The work reported in the MAS simulation literature has employed a wide range of benchmark problems, e.g., Tileworld in [13], a distributed workflow framework in [4] and a simple 'bouncing ball' benchmark in [12], making it difficult to compare performance across approaches. In addition, the particular properties of simulations of situated MAS means that the performance of algorithms on standard PDES benchmarks may not be a good indicator of their performance on MAS simulations.

In this paper we take a first step towards characterising the performance of optimistic synchronisation algorithms in simulations of situated MAS. We present a detailed analysis of the performance of the Decision Theoretic Read Delay (DTRD) algorithm presented in [6]. To better understand the performance of the algorithm, we developed an abstract characterisation of the access patterns found in MAS simulations based on the simulation's degree of coupling and skew. Using this characterisation, we generated stereotypical test cases which we used to compare the performance of the DTRD algorithm with that of Time Warp and time windows. To determine if the test cases reliably predict performance in a real agent simulation, we compared the predictions made by the test cases with performance results from the Boids [11] agent simulation benchmark for a range

of simulation parameters. The results indicate that DTRD is capable of adapting to the mixtures of coupling cases found in real agent simulations and tracking changes in coupling during the simulation.

The remainder of this paper is organised as follows. In section 2 we outline our model of MAS simulation and briefly describe the DTRD optimistic synchronisation algorithm. In section 3 we present a characterisation of access patterns in situated MAS simulations in terms of their coupling and skew. We show how an arbitrary MAS simulation can be decomposed into instances of three coupling cases and argue that performance on these three cases is indicative of performance on a real agent simulation. In section 4 we present a comparative evaluation of three optimistic synchronisation algorithms on two test cases generated using our characterisation of access patterns. In section 5 we extend our analysis to include comparisons on data from the Boids agent benchmark, before concluding in section 6

2. Distributed simulations of MAS

The simulation of situated agents (e.g., robots situated in a physical environment, or characters in a computer game situated in a virtual environment) presents particular challenges for standard parallel discrete event simulation (PDES) models and techniques as described in, e.g., [3, 2]. In a conventional decentralised event-driven distributed simulation the simulation model is divided into a network of Logical Process (LPs). Each LP maintains its own portion of the simulation state and LPs interact with each other in a small number of well defined ways. The topology of the simulation is determined by the topology of the simulated system and is largely static. In many cases we know the lower bound on the timestamp of an event generated by an LP in response to an input event.

In contrast, a defining characteristic of agents is their autonomy [19]. In a parallel discrete event simulation of a multi-agent system, agents may spontaneously generate an event at any point without there being a preceding input event. As a result, simulations of MAS typically have zero lookahead [18]. In addition, an agent's interaction with other agents and its environment is hard to predict in advance; indeed discovering how the agents interact with each other and their environment is often a primary goal of the simulation. For example, what a mobile agent can sense is a function of the actions it performed in the past which is in turn a function of what it sensed in the past. This makes it hard to determine an appropriate topology for a MAS simulation a priori, and simulations of MAS typically have a large shared state which is only loosely associated with any particular process [8].

To address these issues, we developed the PDES-MAS framework [8]. PDES-MAS adopts an optimistic approach to synchronisation in which agent logical processes (ALPs) interact with one or more shared state logical processes (SSLPs) which are responsible for maintaining the shared state of the simulation. ALPs interact with the shared state by reading and writing shared state variables (SSVs). We assume that agents execute a *sense–think–act* cycle, in which they obtain information from the environment (shared state) and compute an action which changes the environment. Sensing gives rise to read events, and acting gives rise to write events.

Within this read/write model, only certain patterns of access can cause a rollback [7]. Reads are the only inputs to an ALP and only the reading of incorrect values can cause a rollback. Agents which only write and never read can never be rolled back (though they can roll back other agents). For example, an ALP which simulates the weather within a virtual environment can never be rolled back by the actions of other agents (assuming there is no causal link between the agent's actions in the environment and the weather). Conversely, only agents which write can cause a rollback. For example, an agent which only reads and never writes, e.g., an LP which monitors the simulation, can never cause a rollback (though it can itself be rolled back by writes from other agents). More precisely, a rollback occurs when a SSLP receives a late or straggler write with timestamp t_w from an ALP a_i to a state variable which has previously been read with timestamp t_r by some ALP a_j, such that $t_w < t_r$ and $a_i \neq a_j$. We call a read *premature* if it is later rolled back by a straggler write.[1]

In [6] we presented DTRD, a synchronisation algorithm for PDES-MAS. DTRD attempts to avoid rollbacks by delaying the processing of read events which are likely to be premature, i.e., if there is reason to believe that the read is likely to be rolled back by a straggler write.[2] The algorithm uses a decision theoretic model to derive a optimal time to delay a read event so as to minimise the expected overall execution time of the simulation. The possible delay times are chosen from a finite set of delay times $\{0, 1, 2, \ldots\}$, where "delay for 0" means "commit this event immediately". Each delay time has an associated cost, namely the amount of real time the read is delayed and hence the ALP which generated the read must spend blocked waiting for the value returned by the read. If we prefer delay times with lower cost, we would therefore always choose to "delay for 0" which has zero cost. However, for each delay time we may have to pay an additional "rollback cost"—the real time spent rolling back if the read subsequently turns out to have been premature. We assume that the probability of paying this

[1] This scheme is similar to the query event tagging proposed in [15] and has similar advantages in reducing the frequency and depth of rollback and the state saving overhead.

[2] Write events can always be processed immediately: a write event by an ALP, a_i with a time stamp t_w can only be rolled back if a read by a_i with timestamp $t_r < t_w$ is rolled back.

rollback cost is lower for some delay times than others, i.e., the longer we delay, the lower the probability of a straggler write and hence of paying the rollback cost.

This gives us a simple trade-off which can be formulated in decision theoretic terms: delaying for less time costs less (in real time) but has a higher likelihood of incurring a rollback cost. The probability that the next write of a particular state variable will be a straggler, i.e., will have a virtual time earlier than that of the read being processed and will arrive after a given delay, is computed from the history of previous writes to the variable. The cost of rollback is computed by adding the cost of rolling back the ALP to the estimated cost of replaying the rolled back events by the ALP.

The DTRD algorithm is adaptive in the sense that it attempts to track changes in the access patterns by the ALPs during the simulation. Ideally it should reduce rollbacks and hence execution time across a wide range of MAS simulations (e.g., simulations in which agents interact frequently vs. simulations in which agents interact infrequently) and execution environments (e.g., environments in which all ALPs advance virtual time at the same rate vs. environments in which some ALPs run much faster than others). However assessing the performance of the algorithm in absolute terms is difficult: even if we know what the optimum performance (in terms of execution time) is for a particular simulation, the algorithm is unlikely to achieve it, given that it has only limited information. To better understand the performance of the algorithm we therefore developed an abstract characterisation of the access patterns found in MAS simulations and used this to investigate the performance of DTRD relative to that of two other optimistic synchronisation algorithms.

3. Characterising access patterns in MAS simulations

We can model the agents' interaction with the shared state at any given point in time in terms of an *access graph*. An access graph is a graph with two kinds of nodes, SSVs and ALPs, and two kinds of edges, corresponding to read and write events. A read edge from a variable v_i to an ALP a_j denotes a read of v_i by a_j, and a write edge denotes that v_i was written by a_j. Each edge has both a real and virtual timestamp, indicating the real and virtual time of the corresponding event. The access graph evolves in real time as edges are added (indicating read and write operations by agents on the shared state), and removed (as a result of rollback).

At any given point in the execution of a MAS simulation, a set of ALPs in the access graph can be characterised by their degree of coupling and their skew. *Coupling* refers to how the ALPs interact with the SSVs and the resulting potential for causality violations (rollback). By *skew* we mean the difference in the 'natural' rate of local virtual time (LVT) progression between the ALPs, in the absence of rollback or any throttling mechanism. Imbalances in the rate of LVT progression may be a result of different agent architectures requiring different amounts of real (CPU) time to advance by a single unit of simulation time, differing processor loads, differing network latencies between parts of a geographically distributed simulation etc. A given set of ALPs may vary between high and low degrees of coupling throughout the execution of the simulation. Similarly, depending on the execution environment (system loads etc) the degree of skew may vary throughout the simulation.

In an optimistic simulation of a highly coupled MAS, a high degree of skew will tend to result in frequent causality violations and rollbacks. In contrast, in a system with low coupling, a high degree of skew can be tolerated, at least from a correctness point of view, since the agents can't roll each other back. To avoid an excessive number of rollbacks, the skew of highly coupled ALPs should be minimised. This can be achieved either by balancing the load in the system to reduce the difference in the rate of LVT progression of the ALPs, or by constraining the optimism of the ALPs with positive skew, or both. While load balancing is desirable in general,[3] it is not always possible, for example, where the computational requirements of the ALPs are intrinsically very different, or where frequent process migration would result in unacceptable overhead, and in this paper we focus on constraining the optimism of the ALPs.

We can identify three stereotypical coupling cases characteristic of situated multi-agent simulations, and the degree of optimism appropriate for each:

uncoupled a set A of ALPs is uncoupled if for any variable v_i read (resp. written) by an ALP $a_i \in A$, for all $a_j \in A, i \neq j$, a_j does not write (resp. read) v_i. In an uncoupled set of ALPs, no matter how skew varies, there can be no causality violations.

fully-coupled a set A of ALPs is fully-coupled if for any ALP $a_i \in A$ there is a variable v_i written by a_i and read by an ALP $a_j \in A, i \neq j$, such that there is a sequence of ALPs $a_1 \ldots a_k$ and variables $v_1 \ldots v_{k-1}$, where a_1 writes v_1, a_2 reads v_1 and writes v_2 $\ldots a_k$ reads v_{k-1} and $a_j = a_1$ and $a_i = a_k$. With full coupling, the agent with the highest negative skew (slowest rate of LVT advance) constrains the rest.[4]

half-coupled a set A of ALPs is half-coupled if the ALPs are neither uncoupled or fully coupled. Informally, a

[3] We may also want to use load balancing in an uncoupled simulation, e.g., if the skew is due to an uneven distribution of processor loads.

[4] A fully-coupled set of ALPs is equivalent to a strongly connected component of the ALP nodes in the access graph, where reachability is defined in terms paths consisting write and read edges to and from SSV nodes. Full coupling is also similar to the notion of a *strong group* introduced in [17] in the context of load balancing in distributed optimistic simulations.

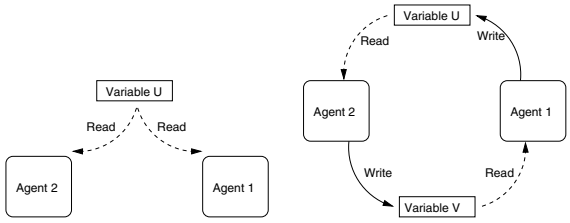
set of ALPs is half-coupled if there is a directed acyclic subgraph of the access graph, in which the leaves are linked to the subgraph by either read or write edges and the internal nodes are linked by both read and write edges. If the writing agents have negative skew (relative to the reading agents), then the optimism of the reading agents must be constrained, but not vice versa.

A fully-coupled set of ALPs may contain half-coupled or uncoupled subsets. Similarly, a half-coupled set of ALPs may contain uncoupled subsets. An ALP is constrained by the most constraining subgraph of which it is a member. For example, an ALP which is in both a half-coupled and fully-coupled subgraph of the access graph is constrained by the subgraph containing the writing agent with the highest negative skew.

An optimistic synchronisation algorithm for a MAS simulation should be able to cope with each of these three cases, and ideally should adapt as the degree of coupling and skew of the ALPs changes during execution. In the next section, we compare the performance of DTRD on the uncoupled and fully-coupled cases for differing degrees of skew, with that of two established optimistic synchronisation algorithms, Time Warp [5] and a windowing algorithm similar to Moving Time Windows [16].

4. Analysing the performance of optimistic synchronisation algorithms using access patterns

We created simple test cases consisting of instances of the uncoupled and fully coupled subgraph types in which we controlled the degree of skew of the ALPs.[5] In the uncoupled test case, two agents read the same state variable (see Figure 1(a)). In the fully-coupled test case, each agent reads a state variable which the other agent writes (see Figure 1(b)).



(a) Uncoupled (b) Fully coupled

Figure 1. Two coupling test cases

[5] In previous work [6] we have reported results for DTRD on the half-coupled case.

We compared the performance of the DTRD algorithm with that of Time Warp and time windows in these two test cases. All experiments were performed using the ASSK simulation kernel [6] developed for parallel discrete event simulation of multi-agent systems. An ASSK simulation consists of one or more agent ALPs and a single shared state LP (SSLP) which maintains the shared state of the simulation. Each ALP processes an event trace from an agent in the original agent simulation and asynchronously sends read and write events from the trace to the SSLP. The ALPs execute asynchronously and inject real and virtual time delays into the event sequences to model the real and virtual time spent by an agent in the 'think' part of its cycle. An ALP blocks while waiting for the response to a read event from the SSLP containing the requested value.[6] Upon receiving an event the SSLP applies the appropriate access to the relevant state variable and generates any necessary responses. If the SSLP receives a straggler event from an ALP, it triggers rollbacks on all the ALPs which accessed the variable at times after the timestamp of the straggler. Rollbacks cause the ALPs to delay for a time (representing the time required to rollback the ALP and here assumed to be 5 msecs) and then rewind and replay their event traces from the time of the rollback. ASSK uses a form of incremental state saving, and in the experiments reported here no fossil-collection mechanism was used. Although deterministic in that the events generated by the ALPs are entirely determined by the input traces, ASSK provides a flexible framework for synchronisation experiments.

All three simulations used the same ASSK infrastructure—only the implementation of the synchronisation algorithm was varied. The Time Warp implementation is essentially the ASSK kernel without the read/write optimisation (i.e., all straggler events cause rollback). Our implementation of time windows used a single window size, w, for the simulation. ALPs can only send events which have a timestamp t_e, such that $t_e < GVT + w$. When an ALP reaches the end of its window, it repeatedly issues requests for GVT computation until GVT advances, allowing it to process its next event. While this adds an extra computational overhead for the blocked ALP, it ensures that the ALP's LVT advances as fast as possible.[7] ASSK uses Mattern's GVT algorithm [10], which may require each ALP to process two GVT messages per GVT calculation.

We generated a synthetic access trace for each test case consisting of 100 simulation cycles. The pattern of vari-

[6] While read events from ALPs are serialised at the SSLP, only the ALPs are delayed by the DTRD algorithm, and the loss of potential concurrency is bounded by the time required to process the events in the SSLP's receive queue.

[7] This is essentially the same as the moving time windows implementation [15] with the polling parameter set such that LPs do not wait for inactivity before initiating GVT calculation.

able access was constant for all 100 cycles in both traces. For the first set of runs, the difference in virtual timestamps between successive agent cycles was taken from a normal distribution with mean 15 and standard deviation 4, and the mean real cycle times were 50 msec for one agent and 200 msec for the other. The standard deviation of real timestamps was 0.05 of the mean. The mean real cycle times are typical of many real agent systems (e.g., 5–20 fps video) and are similar to those used in other agent simulations; for example, the results reported in [12] are for agents with cycle times in the range 95–105 milliseconds. In the fully coupled trace, all read and write events at the same cycle are assumed to occur at the same virtual time. For the time windows implementation we used a window size, w, of 30; this allows on average 2 events to execute optimistically.

	Uncoupled		Fully Coupled	
Algorithm	Comp.	Replayed	Comp.	Replayed
Time Warp	27.78	0	42.51	224.90
Time Windows	42.13	0	43.14	192.90
DTRD	27.78	0	30.50	25.80

(a) Mean cycle times of 50, 200 msecs (skew 4)

	Uncoupled		Fully Coupled	
Algorithm	Comp.	Replayed	Comp.	Replayed
Time Warp	12.63	0	14.54	27.90
Time Windows	12.64	0	14.53	27.70
DTRD	12.63	0	13.54	12.80

(b) Mean cycle times of 50, 50 msecs (no skew)

	Uncoupled		Fully Coupled	
Algorithm	Comp.	Replayed	Comp.	Replayed
Time Warp	47.98	0	82.40	528.20
Time Windows	81.66	0	83.15	203.90
DTRD	47.98	0	52.46	42.40

(c) Mean cycle times of 50, 400 msecs (skew 8)

Table 1. Performance on uncoupled and fully-coupled access traces

Table 1(a) shows the raw performance data for the three algorithms. For each algorithm, we report the computation time and number of replayed cycles averaged over 10 runs of both the uncoupled and fully coupled trace files. The computation time is defined as the total amount of CPU time required for all ALPs. The number of replayed cycles is the total number of cycles repeated by all agents as a result of rollback. This value indicates the frequency and depth of rollbacks within the system.

The first two columns of table 1(a) show the computation time and replayed cycles for each algorithm for the uncoupled test case. As can be seen, the computation time required by Time Warp and DTRD is significantly less than that required for time windows. The number of replayed cycles is 0 for all three algorithms as there are no rollbacks in the uncoupled case. The uncoupled case tests the ability of the algorithms to exploit parallelism within the simulation. As might be expected, the unconstrained optimism of Time Warp outperforms time windows in such a case. With time windows the faster agent is constrained to run at the speed of the slower agent, and must perform frequent GVT computations to advance its window. The results show how the adaptive nature of the DTRD algorithm exploits the parallelism inherent in the uncoupled case—by executing optimistically, DTRD achieves the same computation time as Time Warp.

Table 1(a) also shows the computation time and replayed cycles for each algorithm for the fully-coupled test case. As can be seen, the computation time required by Time Warp is proportionately greater in this case, and is similar to that required by time windows. However the number of replayed events is less with time windows than with Time Warp. With Time Warp, the last event generated by the slower agent is guaranteed to roll back the faster agent, effectively constraining it to run at the speed of the slower agent. With time windows, although the window reduces the number of rollbacks and hence replayed cycles, the resulting reduction in computation time is offset by the frequent GVT computations necessary to advance the window of the faster agent. The DTRD algorithm outperforms both algorithms for replayed cycles (by 87%) and computation time (by 30%). DTRD trades off rollback time and/or GVT computation time for delay time, reducing the computation time required by the ALPs.

We also investigated the effect of skew on the performance of the algorithms. Tables 1(b) and 1(c) show the computation time and replayed cycles for both the uncoupled and fully coupled cases when the difference between the mean cycle time of the faster and slower agents is 0 (mean cycle time of 50 msecs for both agents) and 350 msecs (mean cycle times of 50 and 400 msecs). With no skew (Table 1(b)), all three algorithms have similar computation times in both the uncoupled and fully coupled cases. However, DTRD is more effective in reducing the number of replayed cycles in the fully coupled case, by about 50% compared to both Time Warp and time windows. With a skew of 0, the window is ineffective in reducing the number of rollbacks in the fully-coupled system, as nearly all premature events fall within the window. As a result time windows and Time Warp have a similar number of replayed cycles in this case. Increasing the amount of skew in the simulation to 8 (Table 1(c)) does not affect the pattern of results for computation time (compared to a skew of 4), with both Time Warp and DTRD requiring significantly less computation time than time windows in the uncoupled case and Time Warp and time windows having the same computation time with full coupling. However, the reduction in replayed cycles achieved by time windows relative to Time

Warp is significantly greater with increased skew. With full coupling, the increase in skew also highlights the ability of DTRD to convert time spent in replaying computation (Time Warp and time windows) and/or GVT computation (time windows) into delays, resulting in a reduction in computation time of 37%. DTRD also prevents rollbacks more effectively than a static window, reducing the number of replayed cycles by 92% compared to Time Warp and 79% compared to time windows.

The DTRD algorithm performs well in both test cases, and appears to adapt to varying differing degrees of coupling and skew. However real simulations contain mixtures of uncoupled and fully-coupled access patterns and the degree of coupling (and skew) changes during the simulation. To investigate whether DTRD is capable both of coping with mixtures of coupling cases and tracking changes in coupling during the simulation, we tested its performance on traces from a real agent simulation.

5. Analysing the performance of optimistic synchronisation in intermediate coupling cases

In this section we compare the performance of the DTRD algorithm with that of Time Warp and time windows on traces taken from a standard agent simulation benchmark, Boids [11]. Boids was originally developed as a model of coordinated animal motion such as flocking birds or schools of fish. Each boid has its own local viewpoint of the flock and computes its own motion based on information it collects about other boids within its sensor range. While the Boids testbed is very simple, it captures key characteristics of situated multi-agent systems.

The Boids simulation was implemented using the SIM_AGENT toolkit [14]. Each boid is represented as a single agent and executes a sense–think–act cycle. In the sensing phase the agent reads the x and y position of the other boids within its sensor range (assumed to be 50 units in these experiments). It then computes its motion for this cycle (think phase) and finally writes its own x and y positions (act phase). The simulation is parameterised by the number of agents and the size of the agent's environment.[8] For a given number of agents, by varying the size of the environment we are able to control the average degree of coupling of the simulation. Higher density environments (measured as the number of boids per unit area) increase the likelihood of flocking. With optimistic synchronisation, runs where the agents manage to form a flock should have a larger number of rollbacks on average, whereas runs where little or no flocking occurs should have fewer rollbacks.

[8]The environment wraps around, e.g., a boid moving off the right edge of the environment reappears on the left edge.

We tested the DTRD, Time Warp and time windows algorithms using traces from the Boids simulation. For the experiments we varied the size of the environment from 200 $\times$ 200 to 1000 $\times$ 1000 in 200 $\times$ 200 increments. For each environment size, we collected trace files of accesses to the shared state by each agent for five runs of the simulation. The trace files record access events, each of which represents a single operation on a shared state variable. Each trace consisted of 500 cycles for each boid. To allow comparison with the synthetic test cases, all environments contain 2 agents and the mean difference in agent cycle times is in the range 50–200 msecs. For the time windows implementation we used a window size of 30.

While these traces are very simple, they allow us to investigate performance in mixed coupling situations and in situations where the degree of coupling varies during the simulation.[9] For example, even in low density environments it is rare for the agents to remain completely uncoupled for 500 cycles. Similarly, even in a small environment, the agents will not necessarily be fully coupled for the whole run. However, in general, as the density and hence average degree of coupling increases, we would expect to see the computation time and number of replayed cycles increase for Time Warp. Similarly, we would expect to see the computation time for time windows become proportionally lower and the number of replayed events increase more slowly (relative to Time Warp). Finally, we would hope to see DTRD adapt both across simulations and within a simulation run as the degree of coupling varies.

	Environment size				
Algorithm	200	400	600	800	1000
Time Warp	159.00	149.27	140.98	134.08	132.37
Time Windows	158.63	156.30	156.62	155.85	155.63
DTRD	140.09	147.17	136.66	133.82	132.58

(a) Computation Time

	Environment size				
Algorithm	200	400	600	800	1000
Time Warp	331.46	229.52	141.48	61.48	36.62
Time Windows	229.90	52.22	76.74	15.92	0.26
DTRD	97.80	198.24	75.22	60.74	52.50

(b) No. of Replayed Cycles

Table 2. Performance on the Boids benchmark for varying environment sizes

Tables 2(a) and 2(b) show the computation time and number of replayed cycles as the environment size is increased in 200 $\times$ 200 steps from 200 $\times$ 200 to 1000 $\times$ 1000. The values reported represent an average over 10 runs

[9]The uncoupled and fully coupled test cases represent all the access patterns seen in the Boids simulation. Since the boids have omni-directional sensors, there are no half-coupled subgraphs in the Boids access graph.

of the 5 trace files for each environment size for each algorithm. As can be seen, for small environments, Time Warp and time windows require similar amounts of computation time. However, while the time required by time windows remains essentially constant, the time required by Time Warp drops in the larger, less coupled, environments, and in the largest environments Time Warp outperforms time windows by about 15%. In the 200 $\times$ 200 environment, DTRD requires 12% less computation time than both Time Warp and time windows and in the 1000 $\times$ 1000 environment its performance is equivalent to that of Time Warp. The reason for this becomes apparent when we consider the number of replayed cycles (Table 2(b)). In the smaller, more highly coupled, environments DTRD significantly reduces the number of replayed cycles (by about 70% compared to Time Warp and 57% compared to time windows in the 200 $\times$ 200 environment), and this reduction in replayed cycles is converted into a reduction in computation time for DTRD. In the larger environments (800 $\times$ 800 and 1000 $\times$ 1000) time windows is more effective than Time Warp and DTRD in reducing the number of replayed cycles. However in these cases, the agents rarely interact, and the reduction in replayed cycles is more than offset by the GVT computation overhead of time windows.

	Environment size				
Algorithm	200	400	600	800	1000
Time Warp	777.08	177.7	263.8	54.86	0.74
Time Windows	440.56	99.44	148.00	30.46	0.40
DTRD	92.68	32.48	33.44	9.78	1.68

(a) No. of Rollbacks

	Environment size				
Algorithm	200	400	600	800	1000
Time Warp	0.43	1.29	0.54	1.12	49.49
Time Windows	0.52	0.53	0.52	0.52	0.65
DTRD	1.06	6.10	2.25	6.21	31.25

(b) No. of Replayed Cycles per Rollback

Table 3. Rollback frequency and depth in the Boids benchmark

While the DTRD algorithm requires less computation time than both Time Warp and time windows for all environment sizes, it performs better in some cases than in others. For example, the computation time and number of replayed cycles for DTRD increases from the 200×200 environment to the 400×400 environment. To probe the reasons for this, we investigated the relationship between rollbacks and replayed cycles in more detail. Table 3(a) shows the average number of rollbacks for each environment size for each algorithm. As can be seen, DTRD is successful in reducing the number of rollbacks in the 400×400 environment. However, as shown in Table 3(b), the relationship between the depth of rollbacks and environment size is complex and different for each algorithm. For Time Warp and DTRD, in smaller, more tightly coupled environments, the average number of replayed cycles per rollback is smaller than in larger, less coupled environments. With frequent rollbacks the difference between the LVTs of the ALPs is bounded, resulting in shallower rollbacks and hence fewer replayed cycles per rollback. In medium size environments, rollbacks become less frequent and hence deeper. For example, the average depth of rollback for both Time Warp and DTRD for an environment of size 400 is greater than for an environment of size 200. As the environments become even larger, with less coupling, the agents can execute farther ahead of each other and so each rollback tends to be deeper still, resulting more replayed cycles per rollback. However this is more than offset by the reduced frequency of rollbacks in these environments. In contrast, with time windows, which constrains the depth of rollback by the window size, the average depth of rollback remains more or less constant for all environment sizes. These results are similar to the theoretical predictions of differences in LVT for highly coupled and uncoupled groups of LPs in [17]. Indeed, the relationship between frequency and depth of rollback has been exploited by some optimistic algorithms which force rollbacks to prevent over optimistic execution and reduce the overall cost of rollback (e.g., [9]).

We hypothesised that the performance on the test cases presented in section 4 is a good indicator of performance in real agent simulations. We should therefore expect to see good agreement between the relative performance of the algorithms on the uncoupled and fully-coupled synthetic traces and the Boids simulations with the lowest and highest degrees of coupling. In the Boids traces, the degree of coupling is lowest in those traces with large environments. As expected, DTRD and Time Warp outperform time windows in these traces (though to a lesser extent than in the uncoupled test case). In the smaller environments with higher coupling, we would expect time windows to outperform Time Warp, and in these cases Time Warp does result in more replayed cycles than time windows (even though the degree of coupling is still relatively low in these environments), though again to a lesser extent than in the fully-coupled test case. The performance of DTRD in terms of computation time is at least as good as Time Warp and time windows in all cases, and often significantly better.

6. Conclusions and further work

In this paper we have presented a more detailed characterisation of the DTRD optimistic synchronisation algorithm for simulations of situated MAS originally presented in [6] and compared its performance to that of Time Warp and time windows. To better understand the performance of

the algorithm, we characterised the problem of optimistic simulation of a situated MAS in terms of patterns of access in an access graph describing the shared state. By analysing the performance of the algorithms on stereotypical cases, we predicted where the algorithms should perform well and where they should perform badly in real agent simulations. We then extended our analysis by comparing the performance of the algorithms on a simple agent benchmark, which allowed us to investigate performance on time-varying combinations of the stereotypical test cases.

Our analysis gives us greater confidence about the performance of the decision theoretic approach to synchronisation in general and our specific algorithmic implementation in particular. More generally, we believe our characterisation of the problem is a first step towards a deeper understanding of the difficult problem of optimistic synchronisation for MAS simulation, which complements existing results from benchmark problems and testbeds. We hope that this characterisation of the problem may be useful to others investigating different algorithms for the simulation of situated MAS.

In future work we plan to investigate test cases for half-coupling and quantitative measures of the degree of coupling in an agent simulation based on the notion of critical accesses [7]. We also plan to test our predictions against results from large-scale agent simulations.

Acknowledgements

This work is part of the PDES-MAS project[10] and was partially supported by EPSRC research grant No. GR/R45338/01.

References

[1] J. Anderson. A generic distributed simulation system for intelligent agent design and evaluation. In *Proceedings of the Tenth Conference on AI, Simulation and Planning, AIS-2000*, pages 36–44, Tucson, 2000. Society for Computer Simulation International.

[2] A. Ferscha. *Parallel and Distributed Computing Handbook*, In A. Y. Zomaya, editor, Parallel and Distributed Simulation of Discrete Event Systems, pages 1003–1041. McGraw-Hill, 1996.

[3] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):31–53, 1990.

[4] L. Gasser and K. Kakugawa. MACE3J: Fast flexible distributed simulation of large, large-grain multi-agent systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pages 745–752, Bologna, 2002. ACM Press.

[5] D. R. Jefferson. Virtual time. In *ACM Transactions on Programming Languages and Systems*, volume 7, pages 404–425, 1985.

[6] M. Lees, B. Logan, C. Dan, T. Oguara, and G. Theodoropoulos. Decision-theoretic throttling for optimistic simulations of multi-agent systems. In *Proceedings of the Ninth IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2005)*, pages 171–178, Montreal, Canada, 2005. IEEE Press.

[7] M. Lees, B. Logan, and G. Theodoropoulos. Adaptive optimistic synchronisation for multi-agent simulation. In D. Al-Dabass, editor, *Proceedings of the 17th European Simulation Multiconference (ESM 2003)*, pages 77–82, Delft, 2003. Society for Modelling and Simulation International.

[8] B. Logan and G. Theodoropoulos. The distributed simulation of multi-agent systems. *Proceedings of the IEEE*, 89(2):174–186, 2001.

[9] V. K. Madisetti, D. A. Hardaker, and R. M. Fujimoto. The MIMDIX environment for parallel simulation. *Journal of Parallel and Distributed Computing*, 18(4):473–483, 1993.

[10] F. Mattern. Efficient algorithms for distributed snapshots and Global Virtual Time approximation. *Journal of Parallel and Distributed Computing*, 18(4):423–434, 1993.

[11] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[12] P. Riley. MPADES: Middleware for parallel agent discrete event simulation. In G. A. Kaminka, P. U. Lima, and R. Rojas, editors, *RoboCup-2002: The Fifth RoboCup Competitions and Conferences*, LNAI Vol. 2752, pages 162–178. Springer, Berlin. 2003.

[13] B. Schattenberg and A. M. Uhrmacher. Planning agents in JAMES. *Proceedings of the IEEE*, 89(2):158–173, 2001.

[14] A. Sloman and R. Poli. SIM_AGENT: A toolkit for exploring agent designs. In M. Wooldridge, J. Mueller, and M. Tambe, editors, *Intelligent Agents II: Agent Theories Architectures and Languages (ATAL-95)*, pages 392–407. Springer, Berlin. 1996.

[15] L. Sokol, J. Weissman, and P. Mutchler. MTW: An empirical performance study. In *Proceedings of the 1991 Winter Simulation Conference*, pages 557–563, 1991.

[16] L. M. Sokol, D. P. Briscoe, and A. P. Wieland. MTW: A strategy for scheduling discrete simulation events for concurrent simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 34–42, July 1988. Society for Computer Simulation.

[17] T. K. Som and R. G. Sargent. Model structure and load balancing in optimistic parallel discrete event simulation. In *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation (PADS'00)*, pages 147–154, Washington DC, 2000. IEEE Computer Society.

[18] A. M. Uhrmacher and K. Gugler. Distributed, parallel simulation of multiple, deliberative agents. In *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation (PADS'00)*, pages 101–110, Washington DC, 2000. IEEE Computer Society.

[19] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[10] `http://www.cs.bham.ac.uk/research/pdesmas`