

## USING GENETIC ALGORITHMS TO LIMIT THE OPTIMISM IN TIME WARP

Jun Wang  
Carl Tropper

School of Computer Science  
McGill University  
Montreal, Quebec, Canada H3A 2A7

### ABSTRACT

It is well known that controlling the optimism in Time Warp is central to its success. To date, this problem has been approached by constructing a heuristic model of Time Warp's behavior and optimizing the models' performance. The extent to which the model actually reflects reality is therefore central to its ability to control Time Warp's behavior. In contrast to those approaches, using genetic algorithms avoids the need to construct models of Time Warp's behavior. We demonstrate, in this paper, how the choice of a time window for Time Warp can be transformed into a search problem, and how a genetic algorithm can be utilized to search for the optimal value of the window. An important quality of genetic algorithms is that they can start a search with a random choice for the values of the parameter(s) which they are trying to optimize and produce high quality solutions.

### 1 INTRODUCTION

In parallel discrete event simulation (PDES), the simulation task is divided and assigned to logical processes (LPs) running on multiple CPUs (nodes). To ensure simulation correctness, synchronization algorithms have been developed (Fujimoto 2000). Time Warp (Jefferson 1985) is an optimistic algorithm, in which an LP proceeds under the assumption that it is safe to execute the next scheduled event. If an event arrives in the past of an LP's local simulation time the LP must *roll back* to a previous state and send out anti-messages to eliminate messages which might have been sent in error so that the causal relations between events are not violated. While Time Warp has the potential of making better use of the systems' parallelism, it is well known that over-optimism can lead to instability. In the worst case, the LPs spend most of their time rolling back, making it impossible for the simulation to progress (Lubachevsky, Shwartz, and Weiss 1991).

To address this over-optimism Time Warp protocol has been modified in order to regulate the LPs optimism (Das 2000). Adaptive protocols (Reynolds 1988) modify an LPs behavior dynamically during the course of a simulation in response to changes in the behavior of the simulation.

An adaptive protocol (Panesar and Fujimoto 1997, Lin et al. 1993, Ferscha 1995) relies upon a model of the system. The model typically employs several control parameter(s). There are two commonly used types of models, analytic and statistical. In an analytic model (Panesar and Fujimoto 1997, Lin et al. 1993), a control parameter is expressed as a closed-form function of the state of the system. As the state changes, the parameter is changed accordingly. A statistical model is based on knowledge of the distribution of the random variables in the model. When this knowledge is not available beforehand, the statistical model is built during the runtime of the simulation (Ferscha 1995).

Recently, machine learning techniques such as reinforcement learning (Sutton and Barto 1998, Gosavi 2003), have been applied to the optimization of Time Warp (Wang and Tropper 2007). In reinforcement learning there is no model of the system. The optimization problem is essentially treated as a search problem in which the goal is to find an optimal action to take in each system state. Each such action corresponds to a value of the control parameter.

Because the number of actions has a direct impact on the performance of the algorithm, the number of actions is necessarily kept small, thereby limiting the size of the search space.

A genetic algorithm (GA) (Goldberg 1989, Coley 1999) is a type of evolutionary algorithm (Eiben and Smith 2003) that is inspired by the evolutionary process and by genetics. Over the past few decades, GAs have been successfully used

to solve a wide variety of problems in science and engineering. A GA starts from a set of arbitrary candidate solutions and successively generates better solutions by mimicking the evolutionary process. Like reinforcement learning, GA is also a trial-and-error type of search algorithm. Its advantage is that it can potentially search through the entire search space, while reinforcement learning is limited to a set of candidate solutions.

The algorithm requires no knowledge of the simulation application, nor does it need to know anything about Time Warp itself. In fact, the whole simulation system is treated as a black box and all the algorithm relies upon is performance feedback from the system as it searches for the optimal solution. Although genetic algorithms have been widely used, to our knowledge they have not been used for the optimization of Time Warp.

The rest of the paper is organized as follows. Section 2 describes some adaptive methods which have been used in Time Warp. Section 3 gives an overview of genetic algorithms. In section 4 we discuss how a genetic algorithm is used in selecting a window in Time Warp. In section 5 we present experimental results for a parallel VLSI simulator on some benchmark circuits and our analysis of the results. Finally, section 6 contains our conclusions.

## 2 ADAPTIVE METHODS IN TIME WARP

It is well known that with uncontrolled optimism, Time Warp is prone to an excessive usage of memory and an explosive growth in the number of rollbacks. Excessive memory usage causes performance degradation, and uncontrolled rollbacks can become dominant in the simulation process and even cause system deadlock in the most extreme cases.

An obvious solution to this problem is to limit the optimism of Time Warp. An early protocol called Moving Time Window (MTW) (Sokol, Briscoe, and Wieland 1988) controlled optimism by only allowing events whose timestamps were within a certain time window to be executed optimistically. This is an example of static control, because the size of the window is determined in advance and remains fixed throughout the simulation. Obviously, the difficulty is to determine an appropriate window size before the simulation starts.

Adaptive Time Warp protocols go one step further. Instead of using a pre-determined constant value for the control parameter, they adjust a control parameter dynamically based on "knowledge of selected aspects of the state of the simulation" (Reynolds 1988). For example, in (Panesar and Fujimoto 1997) optimism is controlled by adjusting the size of a window that defines an upper bound on the number of uncommitted events which an LP can schedule; in (Palaniswamy and Wilsey 1993) a moving bounded time window is defined such that only events scheduled within the window can be optimistically executed. A survey of adaptive Time Warp protocols can be found in (Das 2000).

The operation of an adaptive protocol is usually based on a model of the simulation system that describes the relation between system state and the optimal control parameter value. In general, two types of models have been used: analytic and statistical. In an analytic model, the control parameter  $c$  is expressed as a closed-form function of a vector  $\vec{S}$  of some system state variables:  $c = f(\vec{S})$ . For example, the window in (Panesar and Fujimoto 1997) is based on a queuing model; the checkpoint interval in (Lin et al. 1993) is defined as a function of state variables including state-saving time and event execution time. The model is obtained through careful analysis of system behavior. At runtime, the system state is measured, and the result is used to determine the value of the control parameter. As the system state changes, the control parameter is adjusted automatically. The protocol is adaptive in the sense that the control parameter is a function of system state instead being a fixed constant.

Analytical models are usually established before the simulation starts. Statistical models can also be built in advance if the distributions of the controlling random variables of the system are known or can be approximated or, as in (Ferscha 1995), the model can be built at run-time.

While adaptive protocols have all been shown to improve Time Warp's performance in some applications, this improvement is dependent on the quality of the model. Because of the simplifications which are necessary to make in the course of developing such a model, the control strategy is not certain to be optimal. In other words, it leaves open the question whether there exists another function  $c = g(\vec{S})$  that could perform better than the model  $f(\vec{S})$ .

Recently it has been shown that methods from machine learning, in particular, reinforcement learning (RL), can be applied to Time Warp (Wang and Tropper 2007). RL is essentially a trial-and-error search algorithm. The system under control is known as the environment. The controlling entity is called an agent, which performs actions on the environment and receives feedback, or reinforcement signal, from the environment. No knowledge of the environment is required. The agent operates solely on the basis of the reinforcement signals. A set of actions is defined for the agent, which tries to find the best policy, i.e. the optimal mapping from system states to one of the actions. When applied to Time Warp, each of the action is statically mapped to a specific value of the control parameter. RL has been used to determine both the choice of a window size (Wang and Tropper 2007) and the choice of an interval between GVT executions (Wang and Tropper 2009). It was shown that RL could successfully identify the better ones among the given set of values. However, for RL to perform

```
Initialize population with a random set of candidate solutions;  
Evaluate each candidate in the population;  
REPEAT UNTIL termination condition is satisfied DO  
  Step 1: Select parents and put in mating pool;  
  Step 2: Recombine pairs of parents to produce children;  
  Step 3: Mutate the children;  
  Step 4: Evaluate children;  
  Step 5: Select individuals to form next generation of population;  
END DO
```

Figure 1: Genetic algorithm

well, it is necessary to limit the number of actions. Furthermore, this small set of actions is static and remains unchanged at run-time. Therefore, search is carried out in a small sub-set of the search space. For the search to be carried out in the entire search space, either the set of actions have to be enlarged, or multiple agents have to be implemented (Panait and Luke 2005). Neither is an attractive option.

In this paper we present a genetic algorithm for choosing the window size in Time Warp. Like RL, it is also a trial-and-error search algorithm and requires no knowledge of the system under control, only feedback about its actions. Unlike the RL algorithm, the set of candidate solutions can change dynamically, enabling the GA to search a realistic space.

### 3 GENETIC ALGORITHMS

This section gives an overview of genetic algorithms (GA). For an in-depth introduction to genetic algorithms the reader is referred to (Eiben and Smith 2003, Goldberg 1989, Coley 1999).

A GA starts with a set of candidate solutions known as the population, which can be randomly generated from the solution space. The fitness of each candidate in the population is evaluated. Based on the fitness, a number of candidates are selected and put in the *mating pool*. Pairs of candidates are then selected from the mating pool and carry out a recombination operation, which mimics the reproduction process in a biological population, to produce offsprings. Occasionally the newly generated children are subject to a mutation operation. Next, after the children have been evaluated, the survivor selection process is carried out and the next generation of the population is created. The process then repeats itself. Figure 1 outlines the major steps in a generic GA.

To create a GA, the following components must be taken into account.

#### 3.1 Candidate Representation

How to represent a candidate programmatically is a very basic problem. In GA, the candidates are represented by *chromosomes* consisting of *genes*. Two of the most common choices are binary representation and integer representation.

In a binary representation, a candidate solution is represented by a bit-string. Each bit or segment of bits is considered to be a gene. Obviously the length of the string  $n$  is determined by the number of possible solutions  $S$ . It is important to ensure that all possible bit strings represent valid solutions to the specific problem, and all solutions can be represented.

If each component of a candidate solution can take an integer value from a set of possible values, then integer representation may be a good choice. Each integer is considered a gene. As with the binary representation, it is important to ensure that there is a one to one mapping between representations and solutions.

#### 3.2 Evaluation Function

As the concept of fitness is central to the operation of GA, any GA must have a well-defined mechanism to evaluate the fitness of a candidate. This mechanism is known as the evaluation function, the fitness function or the objective function.

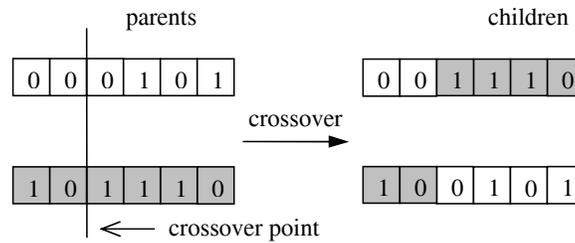


Figure 2: One-point crossover in genetic algorithm

### 3.3 Parent Selection

After all the members of the current population have been evaluated, the next step is to select, according to certain criteria, a number of candidates to serve as parents and pair them up in order to produce children solutions. We briefly describe just one of the many such selection methods.

Fitness proportional selection (FPS) is a selection mechanism based on the fitness level of the candidates. Specifically, if  $f_i$  is the fitness level of candidate  $i$ , then the probability of selecting  $i$  as a parent is  $p_i = f_i / \sum_{j=1}^c f_j$ , where  $c$  is the number of candidates in the population. Since the process is stochastic and is based on the fitness level, a candidate can be selected multiple times.

### 3.4 Recombination

After they have been selected, the parents are paired to produce children through a recombination operation, also known as crossover. One of the simplest recombination operations is the one-point crossover. A crossover point is randomly selected and the two children inherit all of the genes of one of the parents up to the crossover point and the genes of the other parent after the crossover point. This operation is demonstrated in Figure 2.

One-point crossover can be easily extended to n-point crossover, in which a child takes segments of the genes alternately from the two parents. The crossover operation is not limited to binary representation-integer representation can also use this operation for recombination.

### 3.5 Mutation

The purpose of mutation is to increase the diversity of the children. Mutation effectively introduces noise into the algorithm and helps the search to get out of local optima.

In a binary representation, the simplest mutation mechanism is bit-flipping. Based on a pre-defined probability threshold, a bit is randomly chosen to change from 0 to 1, or 1 to 0. This type of mutation can also be extended to integer representation using an operation called *random resetting*, in which the gene to be mutated is reset with a new value that is randomly selected from the set of permissible values.

### 3.6 Survivor Selection

The purpose of this step is to determine which candidates should survive. The most commonly used methods are either age-based or fitness-based or a combination of the two. In an age-based method each individual solution can exist in the population for a fixed number of iterations. In the most extreme case, the current population is entirely replaced by the children. In a fitness-based selection method, part or all of the current population is replaced by the children based on the fitness level of the individuals.

## 4 GENETIC ALGORITHMS IN TIME WARP

In this section we describe a genetic algorithm for selecting the time window size in Time Warp.

### 4.1 Control Parameter

The window size,  $w$ , together with the last GVT defines a limit for event execution. No events scheduled beyond  $GVT + w$  are allowed to be executed. If the next scheduled event of an LP is beyond the window, then the LP must block. When blocked an LP can still receive messages but it cannot send any messages except for messages involved in a GVT computation. When the GVT is updated, the window is moved forward, allowing blocked LPs to unblock if their next scheduled event falls inside the updated window. The window prevents an LP from going far ahead in virtual time, thereby reducing the risk of long rollbacks.

Denote by  $W$  the set of all possible values for the window size. If there are  $n$  nodes in the Time Warp system, then the window sizes used by all the nodes at any particular moment can be represented by the  $n$ -tuple  $(w_1, w_2, \dots, w_n)$ , where  $w_i \in W, 1 \leq i \leq n$ . We shall refer to this as a *vector window*. The problem of finding the optimal vector window can thus be formulated as a search problem within the  $n$ -space consisting of all  $|W|^n$  vectors.

### 4.2 The Algorithm

Having decided which parameter to control, we need to establish a mapping between genes and window sizes. We select a maximum value,  $g_{max}$ , for the gene so that the value of a gene is limited to the set  $X = \{1, 2, \dots, g_{max}\}$ . We then establish a mapping function  $y = f(x)$  to map each gene value  $x$  to a window size value  $y$ .

This is known as the mapping between the genotype and phenotype (Eiben and Smith 2003). In the mappings we adopt, we discretize the window size by defining an incremental unit,  $\omega$ , for the windows such that the actual window size adopted by an LP is always a multiple of  $\omega$ . The mapping function is defined as  $f(x) \equiv \omega x$ . In other words, a gene value  $x, x \in X$ , is mapped to the window size  $\omega x$ .

In our implementation, all LPs on the same node (CPU) use the same window size, but different nodes can have different windows. The size of the search space is thus  $|X|^n$ , where  $n$  is the number of nodes.

#### 4.2.1 Solution Representation

For the optimization problem we chose, it is natural to use an integer representation. A candidate solution is thus represented as an integer string  $x_1 x_2 \dots x_n$ . With the mapping function  $f(x)$ , node  $i$  uses window size  $f(x_i)$ . For example, if the number of nodes is four, then 1115 is a possible candidate solution, assuming  $g_{max} \geq 5$ .

If the number of possible values for each gene, i.e.  $g_{max}$ , is a power of 2, then a binary representation can also be used.

#### 4.2.2 Evaluation Function

In the absence of a formula for determining the fitness of a vector window, it must be determined empirically. To evaluate a vector window, we instantiate the window at the end of a GVT computation and measure the simulation progress at the end of the subsequent GVT interval. The progress is defined as

$$PI = \sum_{i=1}^n e_i / \Delta t,$$

where  $n$  is the number of nodes,  $e_i$  is the number of committed events (the events that will not be rolled back since their timestamps are smaller than the current GVT) in node  $i$  in the given GVT interval, and  $\Delta t$  is the elapsed real time in the GVT interval.

In order to obtain a good approximation for the fitness value of each vector window, the window is evaluated in 10 consecutive GVT intervals and the fitness is the average of the 10 evaluations.

### 4.2.3 Parent Selection

For parent selection we wish to ensure that the fitter candidates have a better chance to reproduce. The probability  $p_i$  of selecting the  $i$ -th candidate as a parent is proportional to its fitness and is given in Section 3. We use an array  $a$  such that the  $i$ -th element of the array is  $a[i] = \sum_{j=1}^i p_j$ ,  $1 \leq i \leq c$ , where  $c$  is the total number of candidates.

Two methods of selection can be defined (Eiben and Smith 2003). In the roulette method,  $c$  random numbers in the range  $[0, 1]$  are generated. For each random number  $r$ , if  $r \leq a[i]$ , and  $r > a[i-1]$ , then candidate  $i$  is selected. In the stochastic universal sampling (SUS) method, a single random number,  $r$ , in the range  $[0, 1/c]$  is generated. If  $r \leq a[i]$ , and  $r > a[i-1]$ , then candidate  $i$  is selected. Then  $r$  is incremented by  $1/c$ , until all parents have been selected.

### 4.2.4 Recombination

For recombination, we use the one-point crossover operation shown in Figure 2, except that it is not limited to binary representation. The crossover point is determined by generating a random number in the range  $[0, d-2]$ , where  $d$  is the length of the representation. For a string of length  $d$ , there are  $d-1$  crossover points between two elements of the string. For example, if the string length is 4 and the crossover point is 0, then the two candidates  $x_1x_2x_3x_4$  and  $x'_1x'_2x'_3x'_4$  will produce  $x_1x'_2x'_3x'_4$  and  $x'_1x_2x_3x_4$ .

### 4.2.5 Mutation

The probability of mutation is set to a small value  $p_m$ . For each gene, a random number  $r$  is produced, and if  $r \leq p_m$ , then that gene is mutated. Since we use integer representation, a random setting is used for mutation. A random number determines the new value for the mutated gene.

### 4.2.6 Survivor Selection

The survivor selection method we adopt is fitness-based. All of the individuals in the current population  $P$  and the  $|P|$  children are put together and sorted by the order of their fitness level, and the best  $|P|$  solutions form the population for the next iteration.

## 4.3 The GA Agent

A designated node that we shall refer to as the GA agent runs the genetic algorithm. It is responsible for disseminating the current solution, i.e., the vector window, to the other nodes. All of the other nodes are oblivious to the fact that a GA is being used.

As mentioned above, each candidate solution is used for 10 GVT intervals. At the end of a GVT computation, all of the nodes send their measurement of progress to the GA agent, which collects the information and updates the fitness of the current solution.

## 5 EXPERIMENTS

In this section we present experimental results making use of several sequential VLSI circuits.

We implemented a genetic algorithm to search for the optimal vector window. We used the clock period of the circuit as the incremental unit  $\omega$ . The value of  $g_{max}$  was chosen to be 8, so a gene could have a value between 1 and 8.

The population size was 12 and the initial population consisted of 12 random vector windows. The algorithm stopped when the population converges, i.e., when all of the candidates in the population are the same. The vector window found by the GA was then used in further simulation runs. We summarize the most important elements of the implementation in Table 1.

The test circuits we used include the two largest circuits from the ISCAS-89 suite, s38417 and s38584, and also, s417x5 and s584x5, which consist of five copies of s38417 and s38584, respectively. Table 2 shows the size of the circuits in terms of the number of primitive gates and D-type flip-flops. Each gate or flip-flop is represented by an LP, and all of the LPs on the same node share an event queue and the time window.

Table 1: Summary of GA implementation

Representation	Integer
Population size	12
Initial population	Random
Parent selection	FPS
Recombination	One-point crossover
Number of children	Same as population size
Mutation	Random resetting
Mutation probability	0.05
Survivor selection	Fitness-based
Termination condition	Population convergence

Table 2: Circuit size

Circuit	Gates	DFFs	Total
s38417	22179	1636	23815
s38584	19253	1426	20679
s417x5	110895	8180	119075
s584x5	96265	7130	103395

Table 3: Performance of GA

Circuits	Generations	Converg. Time
s38417	14.2	45.8
s38584	13.4	64.1
s417x5	15.0	162.1
s584x5	14.7	235.3

We conducted all our experiments on a LAN with 4 AMD Athlon 64 computers running the Linux operating system. The computers were connected by a fast Ethernet switch, and for the underlying messaging we used MPICH. Each circuit was given enough random test vectors for them to run for about 40 minutes.

In our experiments we are interested in examining both the performance of the GA and the effect which the vector windows computed by the GA has on the execution time and the stability of the simulation.

In Table 3 we show the average number of generations before the GA converges, and the average wall-clock time in seconds it takes for the GA to converge. We can see that the algorithm converges fairly quickly. Figure 3 shows the evolution of the average fitness of the population. These charts are typical of the results obtained by genetic algorithms. At the beginning, since we start with a random population the average fitness is rather low. However, in the first few generations, the average fitness grows quickly and then stabilizes.

We now turn to the performance of the vector windows computed by our algorithm. In Table 4 we list the wall clock running time of the simulations with two different algorithms. In Table 4 GA means that the vector window is set to the values found by the GA. NW stands for No Window, represents Time Warp with no time window. Also shown is the reduction in simulation time when using the vector time window found by the GA. For each circuit we ran the GA 10 times and obtained 10 (not necessarily distinct) vector windows. The results displayed in Table 4 are the average of using the 10 vector windows.

For comparison purposes, we also implemented and tested an algorithm which used random values for the size of the time window. The range for the random values is the same as the range used in GA. The running time of the algorithm is

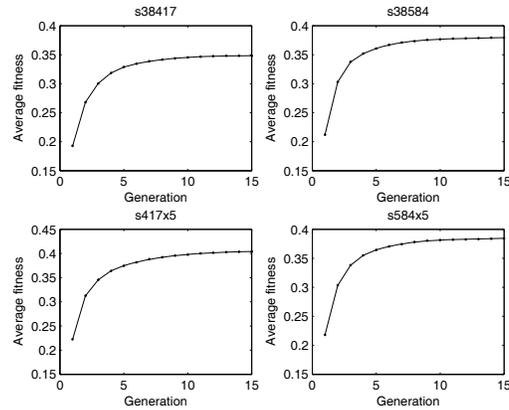


Figure 3: Average fitness of population

Table 4: Running time in seconds

Circuits	GA	NW	Reduction
s38417	1418.79	2204.24	35.6%
s38584	1589.11	2209.43	28.1%
s417x5	1950.87	2204.84	11.5%
s584x5	2075.86	2303.14	9.9%

Table 5: Running time: random window

Circuits	Random	Reduction
s38417	3104.02	54.3%
s38584	2521.61	37.0%
s417x5	4070.85	52.1%
s584x5	3653.08	43.2%

shown in Table 5 together with the reduction when GA is used. Each data item for the random window algorithm is the average of 12 simulation runs.

The data in Tables 4 and 5 demonstrate two important points. First, using good time windows does speed up the simulation, and GA does a good job in selecting appropriate vector windows. Using the GA to select the window size, we reduced the simulation time by 9.9% to 35.6% for the tested circuits.

Secondly, we can see that a bad vector window can have a significantly negative impact on simulation performance. In all the cases, choosing vector windows randomly performs much worse than using no window at all. The simulation time was 14% to 85% longer than using no window. Compared to random windows, using the windows found by GA reduces the simulation time a great deal.

## 6 CONCLUSIONS

In this paper we have demonstrated how the choice of the window size in Time Warp can be formulated as a search problem, and how a genetic algorithm can be applied to find the size of the window. Most approaches to Time Warp optimization problems are model-based, thereby making the quality of the approach highly dependent on the quality of the model. A

GA treats the underlying Time Warp simulation as a black box. All it requires from the system is the feedback on the performance of the simulation.

Used in our distributed circuit simulator, our GA implementation produced excellent results. The vector time window found with GA improved the simulation speed by 9.9% to 35.6% on the test circuits. It should be emphasized that the real power of genetic algorithms is that they can start with an arbitrary set of solutions and achieve good results. Applied to the problem of finding the optimal vector window in our Time Warp system, the use of a GA has proven successful. Encouraged by this success, we intend to apply GA to other optimization problems of Time Warp.

## REFERENCES

- Coley, D. 1999. *An introduction to genetic algorithms for scientists and engineers*. World Scientific.
- Das, S. 2000, April. Adaptive protocols for parallel discrete event simulation. *Journal of the operational research society (JORS)* 51 (4): 385–394.
- Eiben, A., and J. Smith. 2003. *Introduction to evolutionary computing*. Springer.
- Ferscha, A. 1995, July. Probabilistic adaptive direct optimism control in time warp. *Proceedings of the ninth workshop on Parallel and distributed simulation*:120–129.
- Fujimoto, R. 2000. *Parallel and distributed simulation systems*. Wiley-Interscience.
- Goldberg, D. 1989. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.
- Gosavi, A. 2003. *Simulation-based optimization: Parametric optimization techniques and reinforcement learning*. Kluwer Academic Publishers.
- Jefferson, D. 1985, July. Virtual time. *ACM Transactions on Programming Languages and Systems* 7 (3): 404–425.
- Lin, Y., B. Preiss, W. Loucks, and E. Lazowska. 1993, May. Selecting the checkpoint interval in time warp simulation. *Proceedings of the seventh workshop on parallel and distributed simulation*:3–10.
- Lubachevsky, B., A. Shwartz, and A. Weiss. 1991, April. An analysis of rollback-based simulation. *ACM transaction on modeling and computer simulation* 1 (2): 154–193.
- Palaniswamy, A., and P. Wilsey. 1993, March. Adaptive bounded time windows in an optimistically synchronized simulator. *Great lakes VLSI conference*:114–118.
- Panait, L., and S. Luke. 2005, November. Cooperative multi-agent learning: the state of the art. *Autonomous Agents and Multi-agent Systems* 11 (3): 387–434.
- Panesar, K., and R. Fujimoto. 1997. Adaptive flow control in time warp. *Proceedings of the 11th workshop on parallel and distributed simulation*:108–115.
- Reynolds, P. 1988. A spectrum of options for parallel simulation. In *Proceedings of the 1988 Winter Simulation Conference*, ed. M. Abrams, P. Haigh, and J. Comfort, 325–332. San Diego, CA, USA.
- Sokol, L., D. Briscoe, and A. Wieland. 1988, July. Mtw: a strategy for scheduling discrete simulation events for concurrent execution. *Proceedings of the SCS multiconference on distributed simulation* 19 (3): 34–42.
- Sutton, R., and A. G. Barto. 1998. *Reinforcement learning: an introduction*. The MIT Press.
- Wang, J., and C. Tropper. 2007. Optimizing time warp simulation with reinforcement learning techniques. In *Proceedings of the 2007 Winter Simulation Conference*, ed. S. G. Henderson, B. Biller, M.-H. Hsieh, J. D. Tew, and R. R. Barton, 577–584. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
- Wang, J., and C. Tropper. 2009. Selecting gvt interval for time-warp-based distributed simulation using reinforcement learning technique. *Proceedings of the 42nd Annual Simulation Symposium*.

## AUTHOR BIOGRAPHIES

**JUN WANG** is currently a PHD candidate in the School of Computer Science at McGill University, where he has been working in the area of distributed simulation of VLSI systems for the past few years. His other research interests include artificial intelligence and optimizing compilers. Jun Wang can be reached at [jwang90@cs.mcgill.ca](mailto:jwang90@cs.mcgill.ca).

**CARL TROPPER** is a Professor in the School of Computer Science at McGill University. His major research interest is in parallel and distributed computing. He has worked in the area of distributed discrete event simulation since the inception of the field. His focus over the past several years has been parallel VLSI simulation. His group has developed a distributed VLSI simulation environment which is being used for research in both the synchronization and performance issues associated with VLSI simulation. Another research direction is the integration of parallel continuous and discrete event simulation models. Carl Tropper can be reached at [carl@cs.mcgill.ca](mailto:carl@cs.mcgill.ca).