A Parallel Discrete Event Simulation Approach to Solving the Traveling Salesman Problem

Neil McGlohon
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th St, Troy, NY 12180
mcglon@rpi.edu

***Abstract*—The Traveling Salesman Problem (TSP) is easy to describe but challenging to solve. The problem can be stated in one sentence: given a list of cities and distances between them, find the shortest tour navigating to each city exactly once and ending in the initial city. Due to the combinatorial complexity of the total number of possible tours, finding a solution is known to be NP-Hard.**

Solving large city tours will take considerable time. New, neuromorphic, computer architectures have been designed that act more like biological brains. They have hardware neurons, synapses, and axons instead of traditional motherboards, processors, and memory. This different architecture, used by IBM's TrueNorth neuromorphic chip, is able to solve computing problems with very little power cost. Thus, designing a neuromorphic program to solve TSP could prove to be very power efficient.

Implementing a TSP solver on TrueNorth requires great understanding of the problem and neural network approaches to solving it. I have implemented a traveling salesman solver using Rensselear's Optimistic Simulation System (ROSS). The implemented model draws from neural network techniques, designating individual cities and their place in a proposed tour as entities in the simulation. These entities interact with each other to form complete, valid, tours. Performance of the model was measured through strong scaling experiments on an IBM Blue Gene/Q supercomputer. The main goal of this work is to act as a stepping stone toward understanding and implementing a TSP solver on neuromorphic hardware.

I. Introduction

Neuromorphic architectures are still relatively new and applications for them have been mainly focused around classification tasks, an objective that neural networks are very good at. IBMs TrueNorth chip is based of a spiking neural architecture as opposed to a continuous neural network found in traditional artificial neural networks (ANNs) but its capabilities are similar. When tasked with the objective of image classification, the TrueNorth chip is capable of doing real-time image processing with very low power consumption. [1].

Despite being proven to be in the same class as the most difficult problems in computer science, there is still reason to study the Traveling Salesman Problem. Exploring the capabilities of a neuromorphic program designed to solve TSP is an interesting avenue of research. It could, for instance, be able to solve the problem with an efficient use of energy rather than time.

Developing a neuromorphic program is a challenging process, for one there are many constraints and restrictions that come from the hardware perspective. Additionally, there's not an instruction set or anything that allows for a program to be written like a classical program. The chip's neurons must be flashed with already trained networks and weights are static throughout the runtime of the program. So to begin development on a TSP solver

Parallel Discrete Event Simulation (PDES) is a way of simulating interactions between simulation entities through events. These events are planned to occur at times scheduled during the simulation. Since times of events are known, the simulation doesn't have to rely on explicitly calculated incrementing ticks for synchronization. The simulation entities are known as logical processes (LPs). LPs can be spread across different processing environments (PEs) assigned to different cores or compute nodes in the computing system. The inter-process events are handled using a communication layer like MPI so the simulation can be computed in parallel. A simulation system that handles this overhead is Rensselaer's Optimistic Simulation System (ROSS) [2] .

ROSS is a Parallel Discrete Event Simulation platform that allows for LPs to operate with relative independence. When events are scheduled, they are done so optimistically. This optimistic scheduling can allow for events between two PEs to arrive out of order. When these out-of-order events are discovered, the simulation is rolled back according to the Time Warp algorithm [3]. This reverse computation algorithm ensures that there is a deterministic serialization of events in the simulation.

Main contributions of this work include designing a basic brute force PDES "algorithm" for solving the Traveling Salesman Problem and implementing it as a model on ROSS with optimizations and time/space saving improvements. The performance of this implementation was measured through a strong scaling experiment on the Center for Computing Innovation's IBM Blue Gene/Q supercomputer. Additional discussion of the challenges, process, and areas of improvement are also included.

II. Related Work

Spiking neural networks lend themselves well to PDES as their components can be imagined as LPs in a simulation

and spikes between them can be considered as events in the simulation. A spiking neural hardware simulator, NeMo [4], was developed to accurately simulate arbitrary spiking neuron hardware architectures, giving complete control of neuron, axon, and synapse behavior. I expect NeMo to be a crucial part of developing a TSP solver for use on neuromorphic architectures.

Focusing specifically on Traveling Salesman, the origin of the problem dates back to the 1800s but was popularized by Hassler Whitney and Merril Flood in 1934 [5].

There are many surveys [6] [7] [8] of TSP and various computations and heuristics to solve it. The authors in [9] sought to solve the euclidean traveling salesman problem in a parallelized divide and conquer approach that divided the total problem into several subproblems and devised a way to join subproblems while maintaining the optimality of the solution.

The authors Hopfield and Tank provide a good introduction on how neural approaches to optimization problems can yield good results as well as general approaches to constructing networks to solve them [10].

The goal of implementing a TSP solving model onto TrueNorth may require recurrent neural networks. These types of networks are not officially supported by IBM [1] but there is work being done to show how it may be possible to map recurrent neural networks to a spiking neuromorphic hardware [11]. The work shows specifically how Elman neural networks could be mapped to spiking neuromorphic hardware.

III. Problem Formulation

I consider a system model consisting of N cities arranged in a fully-connected undirected graph $G = \{V, E\}$. The edges of the graph are symmetric and defined such that each edge $e_{ij} = e_{ji}\ \forall\ i \neq j \in N$ represents the distance between cities i and j in the network. Self loops are not considered as a valid tour will never utilize an edge from one city to itself.

The formulation of the problem to an integer programming problem was notably made by Dantzig, Fulkerson, and Johnson (DFJ) [12] and modified in [7] [13].

Problem Statement 1. *The Traveling Salesman Problem can be stated as an integer programming problem based off of the DFJ formulation:*

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_i \sum_j d_{ij} x_{ij} \\ \text{subject to:} \quad & \\ & \textstyle\sum_j \ x_{ij} = 1, \quad i = 1, \ldots, N \\ & \textstyle\sum_i \ x_{ij} = 1, \quad j = 1, \ldots, N \\ & \\ & \textstyle\sum_{i,j \in S} \ x_{ij} \leq |S| - 1 \\ & S \subset V, 2 \leq |S| \leq N - 2 \\ & \\ & x_{ij} \in \{0, 1\} \end{aligned} \tag{1}$$

where d_{ij} is the distance between cities i and j. The constraints with S prevent a subtour from being allowed in the final solution. Assuring that any subset of vertices $S \subset V$ does not contain a tour itself. The binary value x_{ij} represents the the decision to travel from city i to city j as existing in the tour or not.

0	0	0	0
1	1	1	**1**
2	**2**	2	2
3	3	**3**	3

Fig. 1: The above array is an example of the valid tour, [0,2,3,1,0] selected from a 4 city network represented by this 4×4 array where each row represents a city and each column represents its place in the proposed tour.

IV. Complexity

The Traveling Salesman Problem is a difficult problem. It is of the class of NP-Hard problems, or at least as difficult as the hardest problems in NP. The problem can be phrased in a way that it is NP-Complete. That is that to find a solution is in NP-Hard but to verify a solution is still possible in polynomial time. The stated NP-Complete problem is as follows: Given a list of N cities and distances between them, determine if there is a valid tour, visiting each city in the problem exactly once and returning to the starting city, of weight less than L.

For example, a table showing numbers for how many possible tours there are given problems following the same system model from Section III:

$$\text{\# Valid Tours} = \frac{(N-1)!}{2}$$

Cities	Valid Tours
4	3
5	12
6	60
10	181440
20	6.0823×10^{16}

It follows then that the solution space for any noteworthy TSP input will be incredibly massive. This complexity will pose difficulties in the feasible implementation of a solver using PDES as discussed in this work.

V. Implementation

The computational model that I developed to solve the Traveling Salesman Problem is built on top of the ROSS framework. One can construct an array of size $N \times N$, where each element represents a city and a place in a potentially valid tour [10]. A valid tour can be constructed from picking a single element from each column, ensuring that no two elements also share a row. An example of this type of construction can be seen in Figure 1.

I constructed the simulation of logical processes (LPs) as elements from the representing matrix described in Figure 1. LPs are then created for each individual city and their all of its potential places in a proposed tour. Thus there are N^2 LPs in the simulation. To simplify the process of programming, I added in N additional LPs representing the final cities in the tour which are required to be the same cities that started in the tour. I consider an event in the system from city i to city j to be a decision x_{ij} in the described in Problem 1.

A. Parallel Algorithm

Seen in Algorithm 1 is a parallel brute force algorithm to be implemented in a PDES environment fitting the described computational model. This algorithm ensures that all possible tours are enumerated and finds the optimum weight tour once the simulation is complete.

Algorithm 1 PDES Algorithm for finding the optimal solution to the Traveling Salesman Problem in Problem Statement 1 as executed by LP α

$W(t) \leftarrow$ Weight of tour t

$\mathcal{L}_{x,y} \leftarrow$ LP representing city x in place y of a tour
$c \leftarrow$ city ID represented by α: $c \in C$
$p \leftarrow$ place ID represented by α: $p \in P$
$T \leftarrow$ Proposed tour so far
$T_{\text{best}} \leftarrow$ Best complete tour known by α

Initialization of Simulation:
$T_{\text{best}} \leftarrow \emptyset$
if $p = 0$ **then**
  $T \leftarrow \emptyset$
  Append c to T
  for $i \in C$ **do**
    if $i \notin T$ **then**
      Send `TOUR` message containing T to $\mathcal{L}_{i,p+1}$
    end if
  end for
end if

Receipt of `TOUR` Message:
$T \leftarrow$ proposed tour from message
Append c to T
if $p < |P| - 1$ **then**
  for $i \in C$ **do**
    if $i \notin T$ **then**
      Send `TOUR` message containing T to $\mathcal{L}_{i,p+1}$
    end if
  end for
else
  for $i \in C$ **do**
    for $j \in P$ **do**
      Send `COMPLETE` message containing T to $\mathcal{L}_{ij}$
    end for
  end for
end if

Receipt of `COMPLETE` Message:
$T \leftarrow$ proposed tour from message
if $W(T) < T_{\text{best}}$ **then**
  $T_{\text{best}} \leftarrow W(T)$
end if

B. ROSS Framework Functions

Creating a ROSS model means implementing the following ROSS features:

1) Initialization and Finalization of LPs
2) Event/Message Data definition
3) Prerun LP instructions
4) Forward Event Handler for LPs
5) Reverse Event Handler for LPs

as well as other necessary programmatic ROSS configuration details.

*1) **Initialization and Finalization**:* Each LP is initialized with some default base state that is defined based on the identity of the LP and is altered by incoming forward or reverse events in the simulation. The LPs have knowledge of the distances to their neighboring cities, they know their own identity: the city and place in the tour that they represent.

They keep track of the minimum *complete* tour that they are aware of. This is the minimum weight valid tour that they know of so far in the simulation. They also keep track of the weight of the minimum weight tour that they know of so as to compare future valid tours and gauge optimality. There are other metrics tracked used for potential model results and future generalization.

At the end of the simulation there is a finalization function that each LP performs. This is used for final sharing of information between LPs, outputting results, and deallocating any memory left allocated.

*2) **Event/Message Data Definition**:* Messages representing events are the only ways for LPs in the simulation to communicate. I will use messages and events interchangeably in this discussion. Thus any information that is to be used by LPs in the construction of valid, minimum tours must be encoded into the messages sent between them. ROSS works through the processing of events by LPs, these events when processed spawn new events that are to be received by LPs.

I have ensured that an event is able to carry with it information about the proposed tour up to the place that the receiving LP belongs to, including what other cities are in the tour, in what order they appear, and the total weight of the tour so far. There is also room for information necessary for the ability to roll back the state of LPs when out-of-order events are discovered but that will be explained later.

There are two types of events that can be sent between LPs. The reason for there being two event types are that there is different things that LPs must do with the information encoded in the message

A `TOUR` message represents the more common message in the simulation. This type of an event carries a proposed but incomplete tour and a weight so far.

The second type of event is a `COMPLETE` event. These contain a completed, valid tour and the total weight of said tour.

*3) **Prerun LP Instructions**:* This part of the program is how the system starts. After each LP has been initialized, each LP then follows their prerun instructions. In this case I have each LP that belongs to a city in place 0 of the tour (the first N LPs) send a message to itself. This message contains no tour information apart from an empty tour of weight 0.

This section is important to the simulation as it is what effectively determines how many tours are searched overall. I will go into further detail on that in Section VII.

4) ***Forward Event Handler****:* This is the most crucial part of the implementation. This is what defines the main behavior of LPs as they progress through the simulation. When events are received, this function is called to handle the information received, change the LP state accordingly, and propagate any additional messages warranted with this new state change.

When a `TOUR` message is received, the LP appends its city ID to the encoded tour, computes the new weight of the tour and sends this information along via new events to all LPs representing cities in the next possible place in the tour that aren't already in the tour. If there are no other possible cities left that aren't already in the tour, the message is sent to the LP representing the original city in the final position in the tour. These events are not propagated if, however, the weight of the proposed incomplete tour is worse than the weight of the best known complete tour. In a random graph this is not an incredibly common occurrence as the expected weight of a complete tour is greater than the expected weight of an incomplete tour but it does aid some in preventing time computing weak tours.

Once an LP from the final position in the tour receives a `TOUR` message, it knows that it has received a complete tour. It compares this proposed tour with its best known complete tour and if it is better it sends a `COMPLETE` event to all LPs in the system.

When a `COMPLETE` message is received, the LP compares the encoded tour with the best known tour and updates this information if it is a better tour.

5) ***Reverse Event Handler****:* A main feature of ROSS is its ability to allow for LPs to schedule events optimistically. This means that there is not anything preventing LPs from having events scheduled that violate happens-before relations. These out-of-order events cannot be left in that order as the determinism of the simulation relies on a consistent serialization of events. Any change of state that occurs due to conflicting events must be undone and any events spawned from said conflicting events must also be rolled back. Fortunately most of the legwork is handled by ROSS but the model implementation must still explicitly define how state is to be rolled back. In this case, certain pieces of LP state that is changed by events is stored in a message upon reception so that it can be restored if needed. Specifically this is the minimum complete tour weight amongst performance metrics. The LP must also keep track of how many random number generation calls were made so that they too can be rolled back - further ensuring the determinism of the simulation.

C. Event Scheduling

When scheduling an event in ROSS, an LP decides on a time greater than the current simulation time as viewed by that LP. This 'delay' allows for more fine grained control of how messages are propagated as the simulation progresses.

1) ***Event Explosion****:* The Traveling Salesman Problem is NP-Hard. This is due to the combinatorially increasing solution space as the number of cities increases. For a given TSP input of N fully connected cities, there are a total of $\frac{(N-1)!}{2}$ valid tours. To form each one of those tours, there are on order of N events scheduled. So for larger possible tours, the total number of events greatly increases. The limitation of computing this becomes the amount of memory on the computer running the program. The simulation must keep track of all events that could potentially be rolled back. Stored events before a time t in the simulation are purged after all LPs have progressed beyond t in the simulation but not sooner.

Due to the combinatorial nature of TSP this is simply not scalable. To make matters worse, the messages are not evenly distributed across the LPs in the simulation. The number of messages processed by any one LP exponentially increases as the place of the city that the LP represents increases. LPs in the second to last place (representing the penultimate city in a tour) receive the greatest number of messages. This poses a resource management issue that will be discussed in Section VII.

2) ***Initial Approach****:* In the first implementation of the TSP solver, I scheduled events to occur within time windows per each LP to help minimize the number of messages during any given time in the simulation and to help reduce the number of conflicting ordered messages. The performance of this scheduling method wasn't great and it was difficult to allocate enough memory to run a the program on an input network of greater than 10 cities.

3) ***Optimization****:* A more elegant solution to the handling of messages would be to utilize the in-situ configurable message delay to the advantage of the solver. If one wanted to determine which of two walkways home was shorter, they could simply send two people to walk both of them at the same speed and whichever path yielded a finished walker first would be the shorter path. Since shorter tours are preferable to longer tours, I give a a penalty for longer tours by setting the delay of the event to be the distance to the next city represented by the recipient LP. So the overall simulation time (not real time) spent on any given tour is equivalent to the weight of that tour.

The first tour completed in the simulation should, then, be the best tour. Additionally, if an LP receives an event scheduled for a time that is greater than the weight of its current best known tour, no processing is required and no further events are scheduled as a result. This means that there are a considerable number of messages that can be prevented from being created.

VI. Performance Results

To evaluate the performance of the model, I ran several experiments to measure the runtime of finding the optimum solution for a given TSP problem input. There was one major hiccup with that. The current implementation has a major issue with message complexity. There are so many messages generated that it fills up the maximum amount of memory that can be stored on any one compute node of the Blue Gene/Q system. That is, a single PE of the simulation for larger city tours can easily fill 16GB of memory, causing the program to crash.

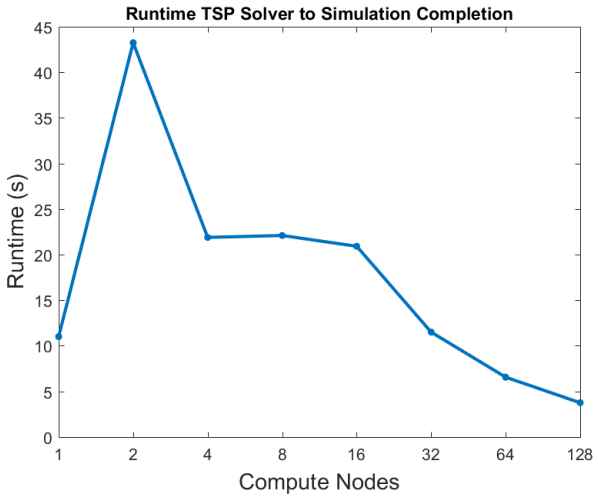

Fig. 2: Runtime analysis of the ROSS TSP solver on a 14 city network input.

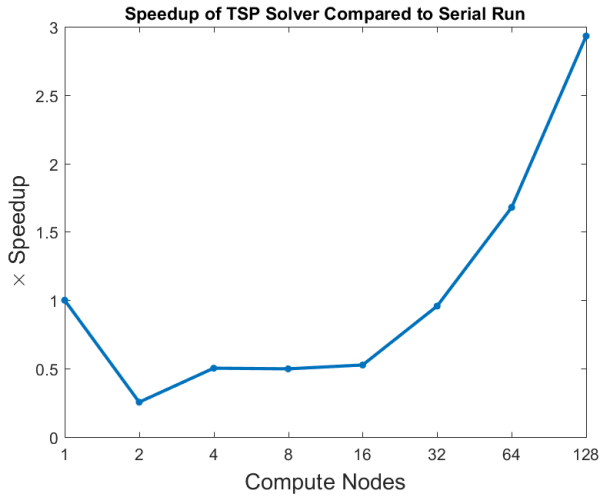

Fig. 3: ×Speedup analysis of the ROSS TSP solver compared to the runtime of the serial execution

So as long as this message complexity continues to be unbounded in the implementation, there will be considerable difficulty in getting solutions to larger city tours. The largest city network that I was able to reliably run with each number of compute nodes was 14 cities. So I did a runtime analysis of a 14 city optimum solution using 1, 2, 4, 8, 16, 32, 64, and 128 compute nodes of the Blue Gene/Q supercomputer.

In Figure 2 we see that the performance of the program is disappointing The main contributing factor for the under-performance appears to likely be due to poor resource allocation in the program. That is, there was a sub-optimal mapping of LPs to PEs. The default mapping scheme caused LPs that have high event populations to be grouped together, leading to some PEs having little to process while others are struggling to keep up.

Figure 3 shows that we do eventually get a good scaling line once we reach more than 16 compute nodes. With 14 cities there were a total of 210 LPs in the simulation. At 16 PEs, that makes almost 14 LPs per PE, which would mean that each PE was roughly responsible for LPs corresponding to all cities of a single place in the tour. It is after this point, when we move to 32 compute nodes and above that the LPs are pigeon holed such that two LPs from different cities but the same place in the tour wouldn't necessarily be in the same PE. This allows for some PEs to allow for the simulation to progress in parallel, but with too few PEs the simulation acts as a tiered sequential execution that can't effectively produce much speedup.

Further discussion of the mapping of LPs to PEs will be in Section VII-C.

VII. Considerations

A. First Place LPs

In my implementation of the TSP solver, I wanted to keep the solver as general as possible so that expanding its ability to work on different types of Traveling Salesman problem formulations would not require too much additional work. There are some LPs in the simulation that are not necessary if the problem is of the form of a symmetric all-to-all network. Specifically, there is only need for a single LP in the first place, meaning that only 1 out of the first LPs is required to acheive a solution. Since any solution starting with city 0 will be equivalent to a solution starting with city 1 just shifted by one city position, there is no reason to have city 1 initiate any possible tours. That is, let S_n be the set of all solutions found with city n as the starting point:

$$S_0 \subseteq S_1 \subseteq \ldots \subseteq S_{N-1}$$

Allowing only city 0 to be the starting city reduces the total number of events scheduled in the system by N by avoiding the searching of redundant proposed tours. A quick fix was made that prevented the other LPs in the first tour position from initiating any tours but this results in wasted and uneven resource allocation. Since the other $N-1$ starting position LPs are still defined but do little to no event processing, there is poor load balancing across allocated processors in the simulation.

B. Last Place LPs

In the simulation, there is no actual need for the LPs representing the cities in the final position. A big reason being that due to the optimization in Section VII-A, only one of the last place LPs will ever receive an event to process. Additionally, since determining if a tour is almost complete is done by the penultimate place LP and all LPs have a knowledge of the distance to the original city, the penultimate place LP can do all of the processing that the final LP would have done. It can also generate and send the resulting `COMPLETE` events. As it is now, there is considerable resource waste by allowing final place LPs to remain in the simulation.

C. Resource Assignment

The main pillar of PDES comes from the first letter: Parallel. If there is no benefit from having additional processors allocated to help run the simulation, then there is not much purpose to implementing a model on a PDES framework like ROSS. As noted by Section VI, there was an interesting result that showed that it was far better to run the simulation on a single processor than in parallel on multiple until significantly more compute nodes were included.

The default method of mapping LPs to processing environments, PEs, is a linear mapping. That is that LPs are mapped so that each PE has approximately equal number of LPs. Given P PEs and L LPs, the first $\frac{L}{P}$ LPs are placed in the first PE and so on. Due to the behavior of the model, LPs representing cities in the later places of the tour will have far more events received than other LPs. This means that the mapping can greatly affect the performance of the model as a whole. A poor mapping may yield a PE or two doing very little work while the others are bogged down with more events than they can effectively process. Designing a new mapping scheme of LPs to PEs so as to more equally balance the load of events in the simulation would likely show great improvement in the efficiency of the solver.

VIII. Future Work

A. Strict Bounding of Message Complexity

A big problem with the current implementation of the model is the message explosion that inevitably occurs. Things can easily get out of hand when each processed event spawns $O(N)$ other events with no limitation on how quickly these messages are generated. Work has been made on developing a recursive algorithm that allows for greater control over how many messages are allowed to persist in the simulation at any one time.

B. Neuromorphic Implementation

The main objective is still to implement a TSP solver on neuromorphic hardware. To do this, the implementation could be ported to run on NeMo, the spiking neuromorphic hardware simulator built on ROSS. This would help give insight into what main limitations of TrueNorth prevent TSP from being implemented - or act as a stepping stone to realizing a hardware neuron model.

IX. Conclusion

In summary, neuromorphic architectures have shown to have great performance-to-cost ratio. As such, implementing a solver of traditionally difficult problems such as the Traveling Salesman Problem is a lucrative idea. Accomplishing this goal is a challenging endeavor and small steps must be first made to understand the problem and what techniques may be utilized to solve it in a neuromorphic environment.

I developed a parallel discrete event simulation model that uses neural network techniques as inspiration to solve this classic problem. Using the combination of neural techniques and the PDES system ROSS, I implemented a model that exactly solved the TSP using a brute force algorithm with pruning techniques. Performance of this model was less than stellar but granted good insight into how future models should be developed.

Future work will be to focus on improving the performance of the model and to allow for larger tours to be solved and eventually implement the solver on neuromorphic hardware.

References

[1] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, and Others, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 2013, pp. 1–10.

[2] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: A high-performance, low-memory, modular time warp system," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648–1669, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731502000047

[3] D. Jefferson and H. Sowizral, "Fast Concurrent Simulation Using the Time Warp Mechanism. Part I. Local Control," Tech. Rep. ADA129431, 1982. [Online]. Available: http://www.rand.org/pubs/notes/N1906.html

[4] M. Plagge, C. D. Carothers, and E. Gonsiorowski, "NeMo: A Massively Parallel Discrete-Event Simulation Model for Neuromorphic Architectures," in *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*. ACM, 2016, pp. 233–244.

[5] M. M. Flood, "The traveling-salesman problem," *Operations Research*, vol. 4, no. 1, pp. 61–75, 1956.

[6] M. Bellmore and G. L. Nemhauser, "The Traveling Salesman Problem: A Survey," *Operations Research*, vol. 16, no. 3, pp. 538–558, 1968. [Online]. Available: http://www.jstor.org/stable/168581http://www.jstor.org/http://www.jstor.org/action/showPublisher?publisherCode=informs.

[7] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, jun 1992.

[8] G. Reinelt, *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, 1994.

[9] B. Avar and D. E. Aliabadi, "Parallelized neural network system for solving Euclidean traveling salesman problem," *Applied Soft Computing*, vol. 34, pp. 862–873, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494615003609

[10] J. J. Hopfield and D. W. Tank, ""Neural" computation of decisions in optimization problems." *Biological cybernetics*, vol. 52, no. 3, pp. 141–152, 1985. [Online]. Available: https://link.springer.com/article/10.1007/BF00339943

[11] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *Rebooting Computing (ICRC), IEEE International Conference on*. IEEE, 2016, pp. 1–8.

[12] G. B. Dantzig, D. R. Fulkerson, S. M. Johnson, V. Chvátal, and W. Cook, "Solution of a large-scale traveling-salesman problem," *Operations Research*, vol. 2, pp. 393–410, 1954. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.606.2752{&}rep=rep1{&}type=pdf

[13] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer Programming Formulation of Traveling Salesman Problems," *Journal of the ACM*, vol. 7, no. 4, pp. 326–329, oct 1960. [Online]. Available: http://portal.acm.org/citation.cfm?doid=321043.321046