Study of Cellular Automata to Solve Travelling Salesman Problem

Daniel Barry
School of Computer Engineering Science
University of Hertfordshire
Hatfield, Hertfordshire AL10 9AB
Email: danbarry16@googlemail.com

Abstract—**The Travelling Salesman Problem is a yet optimally unsolved, suitably complex problem in Computer Science that this project aims to test. Here we ask whether it can be benefited from biologically inspired systems, specifically cellular automata. To achieve this, an algorithm is designed to show that CA can potentially benefit NP complete problems with results close to the performance of brute force search for a low number of nodes.**

Keywords*—*Artificial Life, CA, TSP, Simulation.

I. Introduction

In nature it is not uncommon to see biological systems that solve mathematical problems, whether "intentionally" for some evolutionary gain or "unintentionally" as a side effect of solving another problem [1]. If it is rewarding in the evolutionary sense to solve a problem, there's a good chance a biological system will adapt to solve the problem in order to survive.

The Travelling Salesman (or Salesperson) Problem, or TSP for short, is generally considered a suitably difficult problem in mathematics, hence a good problem to test the computing power of Cellular Automata (CA). To clarify, any use of the word "solve" in this study is not suggesting that the infamous NP problem has been solved, merely that a solution that may or may not be optimal has been found in the large search space of TSP.

It has been shown that collections of agents are able to solve the TSP problem, including ant colonies, mosquitoes, bees and even bacteria [2] [3] [4] [5]. Another interesting method includes using quantum inspired methods, again using a natural system to solve a difficult computation problem [6]. In this study we look to use these biological systems as inspiration to the rules that drive cellular automata.

The following are the questions for the project:

- How can the performance of a TSP solver be measured, given that a CA may not end and may still be active when a solution is found?
- Is there cases where a CA may be created that can out-perform current methods?
- Is it possible to replicate methods found in current literature, including TSP solvers inspired by biological systems, general agents and CA?

It's worth noting that these are in the order they are aimed to be achieved. The last two items are much more difficult than the previous two and will just serve as talking points for the further research discussion on the possibilities these areas present.

II. Problem Description

The problem we are trying to solve is the travelling salesman problem (or TSP for short), where one must travel all of the nodes only once in the shortest path possible. This project aims to look at how this can be done with cellular automata and how good the results are that this method yields.

III. Review of Literature

Looking at the article on "Cellular Automata" from "The Computation Beauty of Nature" [7], we see that cellular automata are generally limited to local searching in terms of neighbours. Only looking at TSP from a local perspective per node is partially why classical algorithms appear to fail. Ideally, an algorithm would consider all viable options every pass and perhaps more importantly, the effect of each move.

Inspired by the hodgepodge machine [8], a uniform expansion from nodes may be interesting, but due to the nature of TSP intersection would likely have to be inferred. If individual information was to be kept about each node, then multiple dimensions would be needed to keep their expansion data separate. This would in turn imply a multi-dimensional search. In the solution described later, an expanding neighbour front is used to prevent the use of multiple dimensions to represent data - especially when dimensions can be shared, hence making a search much easier in terms of memory and time.

IV. Deliverables

The experiment should be able to show the following:

Simulation – A delivery of a suitable CA simulation environment for solving TSP problems with a checking system to test whether the problem has been solved. This will look at how the CA link the nodes and whether a connection has been made.

Performance Measurement – An automated measure of how well the system performs including how optimal a solution is, how long the solution took to develop and how the solution was reached. Of particular interest is whether the connections change throughout the running time.

Solver Demonstrations – At the very least this study should produce a way in which different methods may be compared for further research in this field where different CA based solvers may be tested on exactly the same problem.

Further Optimisation – This will consist of research into further optimisations on existing methods. While there is no certainty about the outcome, there will at very least be discussion as to where we may look to next to better optimise solvers.

V. Experiment Methodology

A. Measurements

Below are definitions for the concepts that will be measured from the experiment.

Path Length – This will be the measure of how long the path is for the CA solver, given as $\overline{F_S}$.

Solution Fitness – The solution fitness will be measured by comparing a brute forced solution with an optimal path length to the solution by the solver. Where the solution depends on randomness multiple F_{S_i} need to be collected, the average and ratio shall be taken as follows:

$$F_{RATIO} = F_O(\frac{1}{n}\sum_{n}^{i=1} F_{S_i})^{-1}$$

Where F is fitness, F_{S_i} is the CA Solver fitness given for experiment i, n is the number of experiments and F_O is the optimal solution for the given problem (Brute Force). F_{RATIO} tells us how close the algorithm being tested is to being optimal, represented as 1, or impossibly bad, tending towards zero.

Time Taken – This will simply be time, t, where full updates are measured as an iteration. NetLogo, the chosen simulator, refers to time in "ticks" of the simulator - the time taken to perform one update of an arbitrary length decided at runtime.

Information Entropy – This will be the measure of the stability of the CA and randomness of the CA and therefore whether a solution has been found, where we may be able to derive some measurement of whether a problem has been solved. This pull be done with the following:

$$H(X) = H_b(p) = -p\log_2(p) - (1-p)\log_2(1-p)$$

Where H is entropy, X is a discrete random variable, $H_b(p)$ is the binary entropy function and p is the uncertainty of a given position. Because we're concerned with how our system develops over time, we'll use a sliding window to encapsulate how the system evolves. The sum of entropy for each cell finally gives us the entire picture with the following equation:

$$\overline{H(X)_{p_t}} = \frac{1}{N}\sum_{N}^{x=0} -p_{t_x}\log_2(p_{t_x}) - (1-p_{t_x})\log_2(1-p_{t_x})$$

Here N is the total number of bits to be checked which in this case is the number of cells. We now look at the bits at time t, where x denotes a one dimensional position we iterate over. We then look at:

$$\Delta H(X)_p = \sqrt[2]{(\overline{H(X)_t} - \overline{H(X)_{t-1}})^2}$$

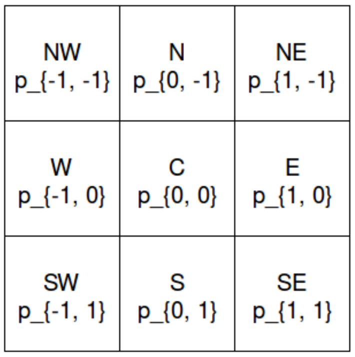
Where $\Delta H(X)_p \geq 0$ and $\Delta H(X)_p \leq 1$. Here we look at whether $\Delta H(X)_p$ is close to zero to detect whether the simulator is stable and therefore stoppped. Note how the entire world is treated as a 1-dimension string - we imagine a 2-dimension space filling curve where order doesn't matter.

Changes in Node Connections – This will be a simple measure of how many changes occur before a solution is found, where we use:

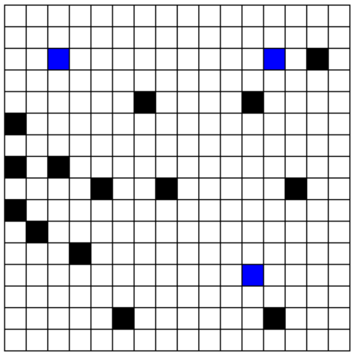
$$\frac{ChangeCount}{!(NodeCount)}$$

It is expected that only Brute Force should reach 1, any solvers over this limit are worse than checking every solution which indicates something is terribly wrong.

B. Data Representation



(a) Moore neighbourhood as seen in the Game of Life [9]



(b) An example of how the environment will be presented. Non-white cells indicate activity, white represent inactive cells and blue cells are the TSP nodes for the purpose of demonstration.

Fig. 1: Proposed Environment

1) Diagrams: The representation in *(a)* is the format the rules will be explained where the rules are formulated with respect to the cell being inspected, $p_{0,0}$ or px,y for clarification. Because of the complexity of the rules, the search will actually be done in increasing diameter, an extension to Moore's original eight neighbour concept.

The other representation in *(b)* is used for a more global explanation of how an environment has been set-up, including information about nodes and distribution of cells. In the cells will be numbers representing their individual summed value.

C. Algorithm

1) Inspiration for algorithm: Partially, the inspiration for the algorithm was the hodgepodge machine. It looked promising in the respect it expanded out and the centre of a swirl could be a node reaching out to another node.

2) Considered Algorithms: The following are ideas considered after a join has been detected to allow other joins to also be detected.

Rule continue – This was the first method tried and unfortunately runs into issue after not much time as nodes begin to interact in strange ways, removing the valleys the method relies on.

Detract from centre of node – Detracting from the centre of one of the nodes proved not be successful either, as this also made the future connections not clear.

Remove trace of joined pairs – This also didn't work, as again no clear connection could be made after the first connection pair.

Stop weakest node expanding – This almost looked successful, but after a short while this was also unsuccessful. The previous connection remained strong whilst no other connection could be made.

Stop weakest and retract environmental affects – This was the method finally decided settled on as it was the first to show results. Looking at it now, it's now the most obvious solution. It makes sense that removing the effects of the weakest, or at least one of the nodes, would make it behave as if it wasn't there and allow a new connection to be made.

3) Decided Algorithm: Step 1 – We increment the surrounding cells at a distance relative to the time passed, in this case ticks, where we sum with the ticks that have passed.

For example, For $t = 0$, $p_{(0,0)} += 0$.

For $t = 1$, $p_{(-1,-1)} += 1$, $p_{(0,-1)} += 1$, $p_{(1,-1)} += 1$, $p_{(-1,0)} += 1$, $p_{(0,0)} += 1$, $p_{(1,0)} += 1$, $p_{(-1,1)} += 1$, $p_{(0,1)} += 1$ and $p_{(1,1)} += 1$.

The solution for $t = 2$ is as follows:

$p_{(\{-2..2\},-2)} += 2$, $p_{(\{-2..2\},-1)} += 2$, $p_{(\{-2..2\},0)} += 2$, $p_{(\{-2..2\},1)} += 2$ and $p_{(\{-2..2\},2)} += 2$

Note that this is still relatively easy for a computer to perform even with very large numbers as loops can be implemented to make this process or looking up cells relatively trivial. Another method would be to look at the closest neighbour and implement their original value add one. This would require multiple dimensions to store the data and would make the algorithm much more complex, but in reality this is what we're simulating.

Step 2 – The next step is to test for a join, if no join is detected then we continue with *Step 1*. To detect a join between two nodes, we first consider whether they share the same value at $p_{(0,0)}$ from their perspective. If this is true then they share a common bridge as the following diagram demonstrates:

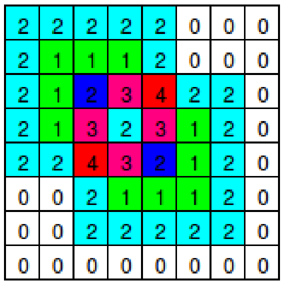


Fig. 2: An example of how two nodes may be joined.

As can be seen in the diagram, they clearly share a bridge of a common value which is 2 which is also the same value as the nodes. It could be possible to use a flood-fill to also find the optimal patch between two nodes but this project does not concern itself with this issue and looks only for the simple sharing of node values.

Step 3 – Given there is a join, we must work out which of the nodes is the "weakest" by summing the nearest neighbours. The node with the lowest sum is the weaker node as there are fewer potential local connections, therefore we do the following:

1) Undo the effects the weakest node had on it's environment.
2) Prevent the weakest node from doing any future expanding.
3) Draw a line between the two nodes to show a join has been made.

We then continue to *Step 4*.

Step 4 – Continue with *Step 1*, unless entropy is zero and the simulation has stabilised - in which case cease processing.

D. Experiment Set-up

Below is a brief description of the desired simulation set-up:

Nodes – Nodes will only take the form of just one cell so that connections may be directly made in a semi-efficient manner for future flood fill purposes.

CA – Each shall be represented by a single patch and evaluated independently of one another.

Representation of solution – This shall be done using lines across the cells.

CA Edge Scenario – The edge will not be updated in the environment so will remain at it's starting condition. This means that special rules are not required for the edge scenario.

CA Stop Condition – This will be done with a visual inspection although this is equivalent of comparing the change in entropy of the monitored area.

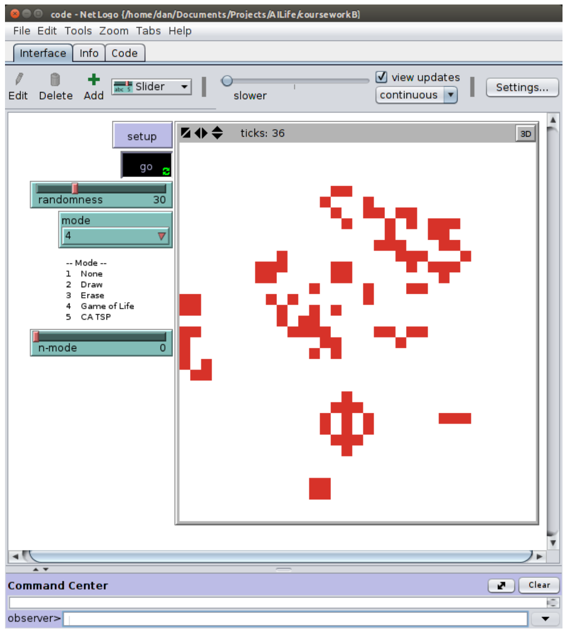
Visualisation of patch value – This is done with $((x\%13) * 10) + 15$, which loops through the bright colour space in NetLogo. This does not uniquely assign a value per patch but does provide a reliable way of seeing where potential "valleys" may reside in the map.

E. Requirements

1) Required Facilities: Development will be done in a Linux environment along with use of the NetLogo simulator. Code will be written in NetLogos own language.

2) Knowledge Areas: The following knowledge areas will be required in order to successfully deliver the project:

- Simulator - NetLogo, chosen for it's simplicity and reliability [10]
- General software engineering
- General knowledge in area of cellular automata



randomness 30

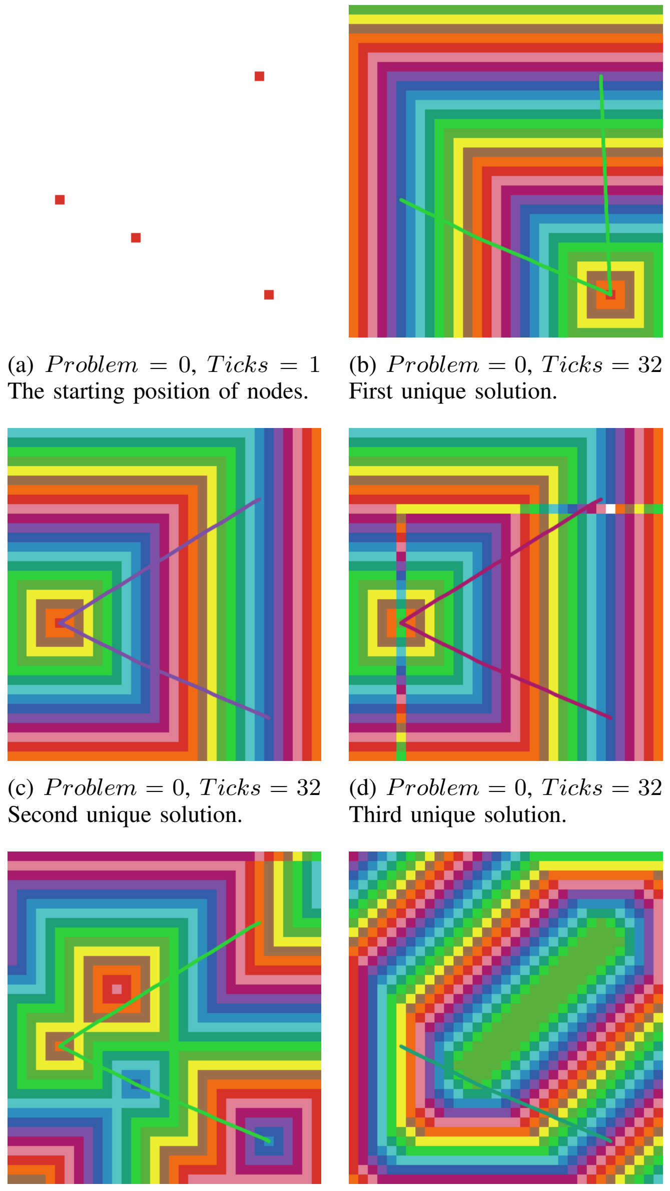
mode 4

-- Mode --
1 None
2 Draw
3 Erase
4 Game of Life
5 CA TSP

n-mode 0

Fig. 3: Capture of NetLogo simulation running a debug version of a simple implementation of Conway's *Game of Life* [11].

VI. Results



(a) $Problem = 0$, $Ticks = 1$ The starting position of nodes.

(b) $Problem = 0$, $Ticks = 32$ First unique solution.

(c) $Problem = 0$, $Ticks = 32$ Second unique solution.

(d) $Problem = 0$, $Ticks = 32$ Third unique solution.

(e) $Problem = 0$, $Ticks = 32$ Fourth unique solution.

(f) $Problem = 0$, $Ticks = 32$ Fifth unique solution.

Fig. 4: The many different ways in which a single solution may be solved.

Problem	**0**	**1**	**2**	**3**
Samples	40	40	40	40
Successful	27	33	14	26
Successful (%)	67.5	82.5	35	65
Average Solution	49.8	22.8	49.4	36.7
Brute Force	47.2	19.3	49.2	34.4
Ratio	0.95	0.85	1.00	0.94

Fig. 5: The results as calculated in the appendix.

VII. Analysis

As can be seen in figures 4b, 4c, 4d, 4e and 4f - there is plenty of variation in how a solution is reached. This is to do with the "asynchronous" nature of NetLogo, where no guarantee of execution order can be put on the code when referencing agents and patches. Interestingly, despite the CA operating differently each time because of this, the solution appears to be the same when looking at low numbers of nodes.

There is another peculiarity to be addressed which is seen in figure 4f, where a solution is not always found for the problem. This is because of the order in which the nodes are connected, where a certain order means that no optimal joins are left due to the method used for joining nodes together.

Now to the results of the equation in figure 5, we see that a 3 connection solution isn't always produced. In the results these incorrect solutions are removed, but we must bare in mind that the cellular automata rules doesn't always produce a solution due to the nature of NetLogo. Problem 2 is worrying with only a 35% success rate in finding a viable solution.

The solution ratio compared with the most optimal solution yielded by brute force search is above 85% in all cases. On inspection of the nodes that are connected during the simulations, if we make a Hamiltonian cycle a much higher percentage of the solutions match the brute force Hamiltonian cycle. Again, this is suspected to be related to the order in which NetLogo addresses patches as opposed to some other effect.

VIII. Limitations

Below are the limitations found whilst working on the project:

NetLogo Updating – NetLogo appears to use a method of asynchonous update as opposed to using synchronous updating of patches, which means the solution may not always be the same each time and changes depending on where is updated fast. It is unclear why this is done, but it may be to force non-synchronous solutions or as a performance optimization with fencing computer memory. Either way, this was not seen before hand as a potential issue with NetLogo and therefore was only found to be a concern later.

Visualisation – Currently it is hard to see the exact value of the patches in real time, or even paused, as the colours don't represent the differences clearly when multiple nodes interact.

Found Solution – The model isn't aware when a solution has been found as planned, therefore it's unclear how many steps to run the model for before stopping.

IX. Conclusion

From the results in figure 5, it's extremely clear that the cellular automata algorithm posed in this project is much better on average than random search. The algorithm is relatively close to the optimal solution given with the brute force search, where results are very close to 1.

It's clear that cellular automata certainly have potential in solving the travelling salesman problem, where this seems to come from nodes interacting with other nodes whilst connections are being made. Ideally this is the property that's wanted from an optimum solution to TSP, where each connection is weighted with the consequences of each other connection.

X. Critical Evaluation

There are a few aspects of the project that are much clearer now the project has been completed and these points will be iterated in the following paragraphs.

The use of NetLogo was something that was supposed to aid the project but instead has hindered it as it's a very comprehensive learning tool but doesn't appear to have the more traditional programming concepts natively supported by C type languages. Such things include arrays/lists/tables or the forced scope on how objects in the model can be referenced. A lot of these limitations have led to "hacks", i.e. very clear breaking of the intended use, making using NetLogo for this project more of a challenge than an aid. The visualisation it offers was perhaps the least important and easily replicated feature to program. Perhaps a more suitable language would have been Java, offering many of the features I desired from the language and ultimately being the engine NetLogo runs in.

The algorithm that was designed to be simulated wasn't designed with asynchronous updating in mind which is part the reason why the cellular automata offers different results each time it is run despite not using any randomness or different starting conditions. Again, it would have been much better to have been able to control that part of the simulation environment.

A number of features were not implemented due to a mixture of time constraints and NetLogos non-standard programming. Features that were not included as originally planned include how connections change over time for certain models of cellular automata, automated checking to see whether a solution has in fact been found.

There was another plan to learn from and optimise on existing algorithms and whilst this project did concern itself with looking at existing ideas, it was unable to develop on existing ideas directly as unfortunately there was no time given how difficult the features are to implement.

XI. Acknowledgements

I would like to acknowledge the *https://draw.io* website for providing a free and easy to use online tool for drawing diagrams seen in this project.

XII. Further Discussion

Given more time to work on this project, it would be good to solve many of the issues identified in the *Limitations* and *Critical Evaluation* sections, as well as increase the number of nodes and really challenges the algorithm. It is expected that performance will drop when number of nodes increases, but it's important to see how this compares to other algorithms whose performance also decreases given a large number of nodes.

A subject that is of particular interest to develop further would be to experiment with the other rules that most likely exist - some that are also capable of also producing solutions using cellular automata. Using larger numbers of nodes coupled with different rules would be a good way to test new solutions.

References

[1] Runarsson, T.P. "Evolutionary Problem Solving" *University of Iceland* 29 January 2001

[2] Dorigo, M., Gambardella, L. M. "Ant Colonies for the Traveling Salesman Problem" *Universit Libre de Bruxelles*

[3] Feng, X., Lau, F.C.M., Gao, D. "A New Bio-inspired Approach to the Traveling Salesman Problem" *East China University of Science and Technology*

[4] Agrawal, P., Kaur, H., Bhardwaj, D. "Enhanced Bee Colony Algorithm for Solving Travelling Salesperson Problem" *International Journal of Control Theory and Computer Modellin* Vol.2, July 2012

[5] Baumgardner, J., *et al*. "Solving a Hamiltonian Path Problem with a Bacterial Computer" *(*Journal of Biological Engineering) Vol.3, 24 July 2009

[6] Draa, A., Meshoul, S. "A Quantum Inspired Learning Cellular Automaton for Solving the Travelling Salesman Problem" *12th International Conference on Computer Modelling and Simulation* 2010

[7] Flake, G.W. "The Computational Beauty of Nature" *MIT Press* pp.231-259, July 1998

[8] Dewdney, A.K. "Computer Recreations - The hodgepodge machine makes waves" *Scientific American* pp.86-89, 1 August 1988

[9] Sharma, V., DEv, A., Ral, S. "A Comprehensive Study of Cellular Automata" *International Journal of Advanced Research in Computer Science and Software Engineering* Vol.2, No.10, October 2012

[10] Wilensky, U. "NetLogo" *Northwestern University* 1999. URL: `https://ccl.northwestern.edu/netlogo/`

[11] Hua, D., Pelikan, M. "Variations on Conways Game of Life and Other Cellular Automata" *University of Missouri-St. Louis* July 2012