Implementing Metropolis Hastings in CUDA: Simulated Annealing for the Traveling Salesman Problem

Steve Bronder
sab2287@columbia.edu

Haoyan Min[†]
hm2689@columbia.edu

December 17, 2016

Abstract

This article details smart computational techniques to implement Metropolis Hastings algorithms on a GPU. Maximizing throughput on a GPU allows the algorithm to propose and evaluate hundreds of thousands of samples simultaneously, massively decreasing downtime spent assessing failed proposals. The acceptance step employs a 'first past the post' style, using the small amount of divergence between threads in the GPU to accept the first seen accepted proposal. The only wait time for the acceptance step is the read-write of each first success between blocks of draws to a global variable. To showcase the method, a GPU based simulated annealing program for the traveling salesman problem is created and tested which uses this method. Results are fast, near the optimum, and the algorithm can handle millions of data points.

1 Introduction

The rise of graphics processing units (GPUs) as a general purpose computing device has created a world where computation of many algorithms has become considerably cheaper in both computational and hardware cost [11]. GPU's were first used for fast floating point calculations on blocks of pixels for an image, which makes it perfect for algorithms with parallelization at a very low level such as matrix multiplication and applying filters to images. It can be difficult to directly see how sequential algorithms, such as the Metropolis Hastings [7] style algorithms, can utilize the fast computation available with GPU computing. Because the value of parameters at time t are dependent on the values of time $t-1$ sequential style algorithms leave a tiny window open to obvious parallelization. The idea proposed in this paper is to treat the GPU as a giant 'sampling box' whereby where a large search of the parameter space happens at each step of the algorithm. For large parameter spaces, this allows for less downtime, by either moving to a spot with a lower loss or a higher loss with some probability at each step.

Section 2 reviews the Metropolis Hastings algorithm. Section 3 breaks down the tools needed to effectively sample and evaluate the proposal steps simultaneously. Section 4 gives an example of these methods by using simulated annealing [3] to solve the traveling salesman problem [15] while section 5 gives the experimental results of the implementation.

[†]Both co-authors contributed equally to this research

2 Metropolis Hastings

Metropolis Hastings (MH) methods exploit the fundamental theorem of Markov chains to sample from an intractable distribution. The MH approach is to design an irreducible, aperiodic and positive recurrent chain whose stationary distribution is the posterior distribution of interest [13]. For the experiment in this paper, sampling is done on the intractable distribution of possible combinations of routes in the traveling salesman problem. Each sample represents a possible state space of a Markov chain. The approach is to iteratively sample from the distribution one combination at a time conditioned on the current state of the salesman's trip.

There are two rules that are required to create an MH sampler, one to propose how to move from one position to another, and a second rule to accept that move as the new position. Suppose the stationary distribution $X = (X_1, \ldots, X_d)$ is

$$\pi(X_1, \ldots, X_d) = g(X_1, X_2, \ldots, X_d)/\kappa \tag{1}$$

where κ is a normalizing constant. Given the current value $\mathbf{x} = (x_1, x_2, \ldots, x_i)$, let $L(x_i)$ be the loss function evaluated with the realized value x_i from $\mathbf{x}$. Let $g(L(x_i), L(\tilde{x}_J)$ be the a function that, given the loss for the proposed value $\tilde{x}_J$ and current realized value x_i, returns an acceptance ratio with a value between $[0, 1]$. The next move follows the steps below.

1. Assign $J \sim U(1, 2, \ldots, n)$.
2. Sample X_j with the proposal distribution $\pi(\tilde{x}_J | x_1, \ldots, x_{J-1}, x_{J+1}, \ldots, x_d)$ where $\tilde{x}_J$ is the proposed move.
3. Evaluate whether $L(\tilde{x}_J) < L(x_i)$. If this is better, accept $\tilde{x}_J$ as the newest draw
4. If not, sample $A \sim U(0, 1)$ and if $A > g(L(x_i), L(\tilde{x}_J)$, accept $\tilde{x}_J$ as the newest draw

Note that, the Gibbs sampler's [6] acceptance function accepts this next move with probability one while a hill climbing algorithm's [14] acceptance function will always be zero. This means that both algorithms are special cases of the Metropolis Hastings algorithm. An algorithm such as simulated annealing has an acceptance function of

$$g(L(x_i), L(\tilde{x}_J) = e^{-\frac{L(\tilde{x}_J) - L(x_i)}{T_i}} \tag{2}$$

Where $T_i \in \mathbb{R} \cap [0, \infty]$ is a parameter known as temperature. At each step of the algorithm the temperature is decreased by some $\alpha \in \mathbb{R} \cap [0, 1]$ such that

$$T_i = T_{i-1} - \alpha T_{i-1} \tag{3}$$

At high values of temperature, simulated annealing will act like a Gibbs sampler by accepting every proposal step. As $T_i \to 0$ simulated annealing will start to work like a hill climbing algorithm and only accept proposals whose loss is less than the current values loss. The ability for Simulated Annealing to scale between these two algorithms is one reason that simulated annealing has become such a popular method for optimization. The beginning of the algorithm allows searches a large part of the parameter space while, as temperature goes down, simulated annealing will start to converge to a global optimum. Allowing temperature to go down over time allows the algorithm to jump out of local optimums by temporarily selecting a worse loss. Figure 1 shows this, whereby Gibbs sampling would continually jump around the entire space and hill climbing can end up in the local minimum, simulated annealing can jump out of the local minimum on its search towards the global minimum.

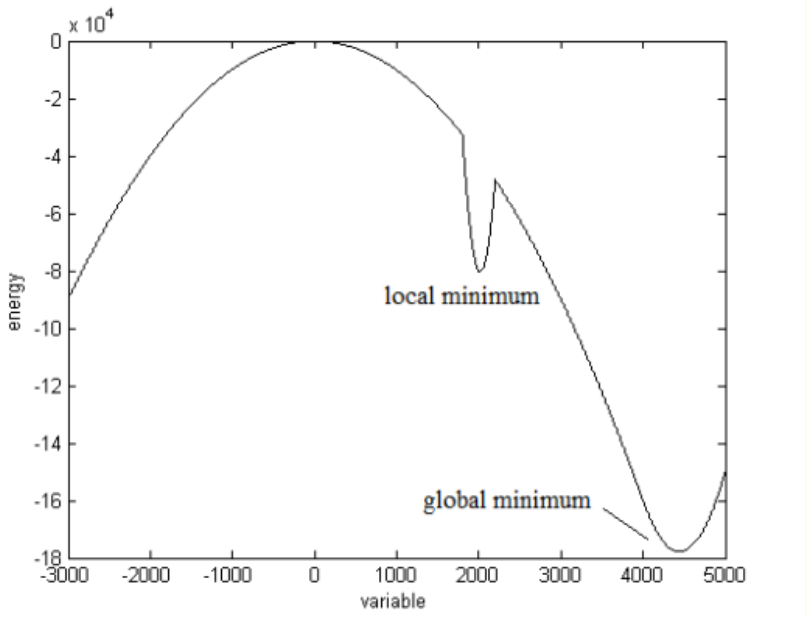


Figure 1: With the standard hill climbing algorithm, it's possible to end the optimization in the local minimum. The temperature decreasing over time allows simulated annealing to jump out of the local minimum and continue towards the global minimum.

3 Implementation of Metropolis Hastings for GPUs

Because MH is inherently sequential, the only obvious place to utilize the high amount of parallelization in a GPU is within each iteration. To do this, the GPU is used as a large box for sampling. The goal is to maximize throughput at each kernel initialization. For example, a GTX 780 has eight streaming multiprocessors that can take in sixteen blocks of size 1024 at a time. Maximizing throughput allows the MH algorithm to propose and evaluate 131,072 samples in one iteration. The process is simple enough, given each thread, draw a random proposal from the parameter space. Taking such a large amount of samples at each iteration allows the algorithm to 'go wide' and search a large amount of the parameter space relative to standard CPU based sequential methods.

For evaluation of each proposal, the algorithm uses the 'first past the post' [17] technique of acceptance. While Sohn implements this on multiple CPUs, the trick to doing this on a GPU is to create a `volatile unsigned int` variable that will capture the first winner. The parallel acceptance step uses the small amount of divergence between each thread to take in the 'first' acceptance. One slight problem in algorithm 1 is that it is possible for two acceptance threads to check the if statement one after another before one of them has the chance to write to `globalFlag`. Allowing multiple acceptance samples to go through the if statement causes multiple writes to `globalFlag`, which is only a problem if there are a lot of acceptance threads checking the if condition one after another without another acceptance thread changing `globalFlag`. This problem will most likely only happen for a few acceptance threads so it should not be a serious issue.

Algorithm 1 Parallel Acceptance Rejection Procedure

```
▷ Define Variables
unsigned int xid = threadIdx.x + blockIdx.x * blockDim.x;
volatile unsigned int globalFlag = -1;
Loss();                                              ▷ Function to calculate loss
g();                                  ▷ Function to calculate acceptance probability
float x̃_J;
float x_d;
procedure Parallel Acceptance Rejection Procedure

    if (Loss(x̃_J) < Loss(x_i) AND globalFlag == -1) then{
                                  ▷ Only take the loss if globalFlag has not been set yet
        globalFlag = xid;

    }
    __threadfence();

    if (globalFlag == -1) then{
        float u = randUnif(0,1);
        float accept = g(x̃_J, x_d)

        if (accept < u) then{
            globalFlag == xid;

        }
    }
```

In Algorithm 1 the global flag is used to store which thread had a winning proposal step. Then any update step can use the global flag in accessing the winning proposal, setting the global flag back to zero. This method allows the algorithm to quickly throw away all other samples besides the first winner it finds, meaning that it is unnecessary to check any other samples after the first acceptance. This method, in combination with a large number of samples, allows for very little downtime, where the algorithm has either found a better loss or accepted a worse loss with some probability. Continuously moving around the parameter space gives the algorithm an opportunity to converge in much fewer iterations.

4 Example: Traveling Salesman Problem

The traveling salesman problem (TSP) is the standard combinatorial optimization problem when exploring new optimization approaches. TSP is the standard optimization task because finding an optimal solution to TSP is NP-hard [5], though many researchers have created wonderful algorithms to find exact and approximate solutions. This sets the bar for new methods quite high. Given a complete, undirected, weighted graph $G(.)$, let V be the set of vertices, E the set of edges, and W is a set of weights, the undirected weighted graph is $G(V, E, W)$. The problem optimized in this paper is the Hamiltonian symmetric Euclidean traveling salesman problem [10]. In this problem, there is only one edge going between each vertice, traveling from v_i to v_j has the same edge and weight as going from v_j to v_i, and the

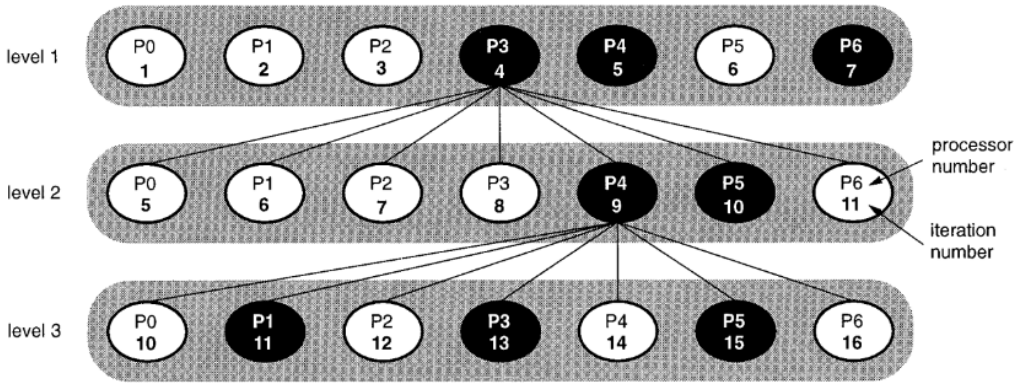


Figure 2: Figure from [17] showing the parallel method of acceptance and rejection. For each thread, the first found acceptance is taken as the acceptance of that step, with the next steps parameters being equal to this winner

goal is to find the shortest trip that starts and ends at the same point while going through each vertice once. The best solution to this TSP is the trip which minimizes the Euclidean distance of the Hamiltonian cycle, defined by the loss function $L(V, E, W)$

$$\min L(V, E, W) = \min \sum_{i=1}^{N} ||v_i - v_{i+1}||_2 \tag{4}$$

where $v_1 = v_N$ is the starting city of the trip.

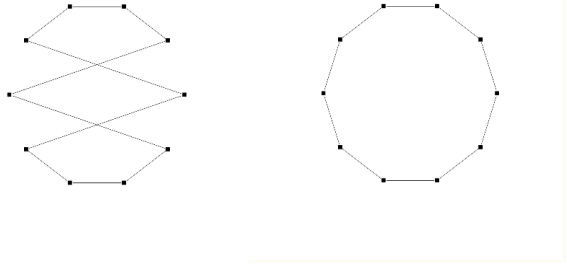
The two most well known solvers for TSP are LKH2 [8] and Concord [1]. As TSP is not the main goal of this article, but instead to use a GPU to implement MH algorithms, please see the references for papers explaining both programs.

4.1 Swap, Insertion, and 2-Opt Algorithms

Three simple algorithms are used in the new GPU based program Columbus to solve the TSP, swapping cities along the trip, inserting cities in the cycle, and reversing parts of the cycle with 2-opt. In linear form, given the cycle through each city 01234560, suppose the algorithm wants to swap city two with city six. Then the cycle becomes 01**6**345**2**0. Figure 3 gives an example of how the swap method can be used to untangle the trip and find a shorter tour. While the swap method is useful for when two cities create a tangle in the graph, insertions are useful when only one tour is out of place. Figure 6 examples this, with one point being out of place is inserted into the correct position and the graph becomes untangled[1]. The 2-opt algorithm [4] of figure 5 reverse an entire cycle of the chain in order to unwind the graph[2].

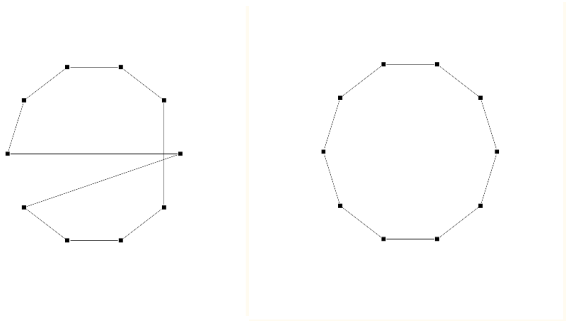
[1] In linear form an insertion would be, for city five to go in between two and three in the tour 01234**5**60, 012**5**3460

[2] In linear form a 2-opt would be, to reverse the chain between city five and one in the trip 0**12345**670, 0**15**432670



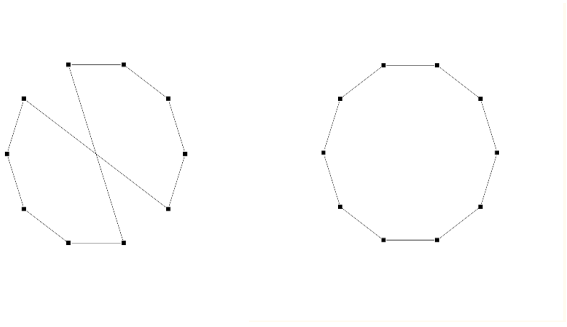
(a) pre-swap trip (b) post-swap trip

Figure 3: An example of how the swap method can find a shorter route



(a) pre-insert trip (b) post-insert trip

Figure 4: An example of how the insertion method can find a shorter route



(a) pre-2opt trip (b) post-2opt trip

Figure 5: An example of how the 2-opt method can find a shorter route

4.2 Implement Details and Edge Conditions

Figure 3 shows that swap has changed four segments of the route. Suppose that in the index $\mathbb{I}$ of the salesman's trip the algorithm chooses cities i and j to swap. Let $||i,j||_2$ be the Euclidean distance between the i^{th} and j^{th} city, then the change of the distance is defined as

$$\begin{aligned} \Delta L =& ||i-1,j||_2 + ||j,i+1||_2 + ||j-1,i||_2 + ||i,j+1||_2 && (5) \\ & -||i-1,i||_2 - ||i,i+1||_2 - ||j-1,j||_2 - ||j,j+1||_2 && (6) \end{aligned}$$

However there are edge conditions which break equation 6. There must be at least exist two cities between i and j, such that

$$|j-i| > 2$$

If this is false, there will be some overlap and make the calculation go wrong. For example, if

$$j = i+2$$

then as figure 6a shown, swapping i and $i+2$ only changes two segments of the total route, thus the equation 6 does not hold. The intrinsic reason is that swap deals with 4 segment changes, while the edge condition causes swap to degenerate to a 2-opt, which is designed for 2-segment-change condition. Insertion is designed for a 3-segment-change condition. Suppose city i is inserted between j and $j+1$ and the edge between city $i-1$ and $i+1$ is reconnected, then the change in loss is

$$\begin{aligned} \Delta L =& ||i-1,i+1||_2 + ||j,i||_2 + ||i,j+1||_2 && (7) \\ & -||i-1,i||_2 - ||i,i+1||_2 - ||j,j+1||_2 && (8) \end{aligned}$$

Similar to swap, there is an edge conditions. Because this is a 3-segment-change, the restriction is less strict than swap with the only assumption being

$$j > i+1 \text{ or } j < i-2 \tag{9}$$

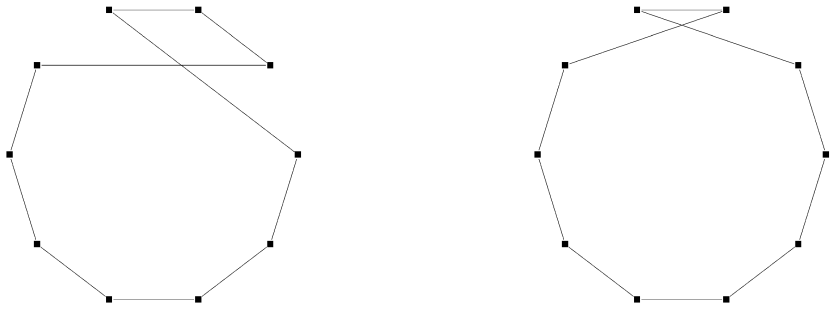
There is a slight difference in the insertion algorithm when $i > j$ and $i < j$ due to city i being inserted between the interval of j and $j+1$. When $i > j$, there is a one unit shift to the right in the index $\mathbb{I}$ of the trip. Alternatively, when $i < j$ there is a one unit shift to the left in the index $\mathbb{I}$ of the trip. Figure 6b shows the edge condition when the loss function in equation 8 would fail. It also degenerates to a 2-segment change condition. For 2-opt, because a 2-segment-change is the simplest change the algorithm can make[3] there is no edge condition. If the algorithm samples city i and city j and tries to remove segments $||i,i+1||_2$ and $||j,j+1||_2$, the change in loss is

$$\Delta L = ||i,j||_2 + ||i+1,j+1||_2 - ||i,i+1||_2 - ||j,j+1||_2 \tag{10}$$

4.3 Simulated Annealing optimizations

In addition to these algorithms, several smart simulated annealing methods are implemented. At each step, the algorithm runs 1,000 swap, insertion and 2-opt kernels. Running each algorithm multiple times allows the algorithm to search for much longer at a given temperature

[3]For a Hamiltonian cycle it is not possible for 2-opt to change only one segment without destroying the cycle



(a) Pre-swap (b) Post-swap

Figure 6: An example of how the insertion method can find a shorter route. While this graph would require one more swap to find the optimal route, a single insertion would allow the graph in (a) to untangle to the optimal solution.

such that it can meaningfully explore a large part of the parameter space as the temperature decreases, helping to reduce the chance of falling into a local minimum. The possible distance in the range of a swap, insertion, or 2-opt decays over time and is controlled by

$$\beta = \exp\left(-\frac{r}{T_i}\right) \times N + z \tag{11}$$

Where N is the size of the trip, r and z is are constants, and T_i is the starting and current temperatures, respectively. At the beginning of the simulated annealing process, it is useful to give the algorithms the opportunity to exchange very distant parts of the trip.

Over time, long range exchanges have a much lower probability of being useful, and so the algorithm automatically reduces the sampling distance. The first city in the range is chosen from a uniform distribution, while city two is selected by

$$\text{city\_two} = \text{city\_one} + (N(0,1) \times \beta) \tag{12}$$

Cities that are close by to the current selection are most likely to be candidates for a successful reduction in trip distance [9]. As the temperature decreases so does the variance, reducing the sample space a given step can utilize.

The generic loss function for the acceptance-rejection when using simulated annealing to solve the traveling salesman problem is

$$\exp\left(\frac{L(\tilde{x}_j) - L(x_i)}{T_i}\right) \tag{13}$$

A scale invariant version of the comparisons of the loss in the numerator can be written

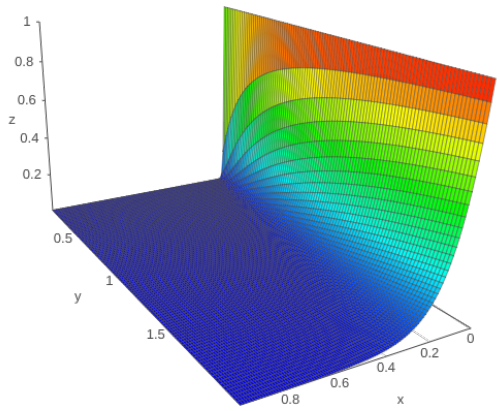


Figure 7: An example of the scale invariant loss function for numerator $\in [0, 1]$, $T_i \in [0, 2]$, and a constant equal to 18

such as[4]

$$\frac{L(\tilde{x}_j)}{L(x_i)} - 1 \in \mathbb{R} \cap [0, \infty] \tag{14}$$

This version of the acceptance probability makes the annealing process much easier to control. For a given set of problems such as TSP, placing (7) into (6) and adding a constant allows for a scale invariant version of the original TSP loss with a tuning parameter.

$$\exp\left(\frac{\left(\frac{L(\tilde{x}_j)}{L(x_i)} - 1\right)\gamma}{T_i}\right) \tag{15}$$

Using this tuning parameter is much simpler when attempting to find a good annealing schedule for multiple data sets as the numerator will normalize to values within $[0, n]$ for some small n. For example, if the proposal step was 1.5 times as large as the current step then the numerator become $.5\gamma$. Given a known range of temperature it's possible to graphically decide what the researcher would like the acceptance probability to be as the temperature decays.

5 Experimental Results

Experiments use a computer with an Intel Core i7-4790K CPU @ 4.00GHz and a Geforce GTX 780 GPU with 4 Synchronous DIMM DDR3 @ 1600MHz each holding 8GBs of memory for a total of 32 GBs. Code and data for these experiments can be found on github. While

[4]In the acceptance-rejection step the proposal's loss will always be worse than the current loss

not the primary focus of TSP, the art data found at the University of Waterloo's TSP page[12] give several famous pictures that through weighted Voronoi stippling [16] make a collection of points. The objective is to find the shortest continuous line through each point in the art [2]. For the columbus program, the first parameter is the tsp data set, while several flags control options for the simulated annealing algorithm

```
Inputs:

(Required)
input_file.tsp: [char()]
 - The name of the tsp file, excluding .tsp at the end, containing
 the cities to travel over.
 (Optional Flags)
-trip: [char()]
   The name of the csv file, excluding .csv, containing a
   previously found trip.
   If missing, a linear route is generated as the starting trip.
-temp: [float(1)]
   The initial starting temperature. Default is 10000
-decay: [float(1)]
   The decay rate for the annealing schedule. Default is .99
```

Five data sets were used to evaluate the algorithm, the 100,000 point Mona Lisa, 120,000 point Van Gogh, 140,000 point venus, 200,000 Earring, and the 744,710 point LRB744710 data sets. Table 1 gives the GPU implementations results, known optimal solutions, times[5], and percent difference from the optimal solution. The only TSP problem which had an optimal solution time posted was the Mona Lisa data set, taking up 11.4 CPU years[6]. The algorithm starts with a naive linear trip, going from $[v_1 \cdots v_N, v_1]$. The algorithm gives worse overall answers but is still within a reasonable bound for a heuristic solution. It also produces these solutions very quickly relative to current methods[7]. Figure 9 contains a graphical version of the problem. See the appendix for larger versions of the Mona Lisa, Van Gogh, Earring and LRB744710 continuous line drawings, where the continuous line is the shortest trip found by the GPU algorithm.

As seen in figure 8 it appears that the temperature and annealing schedule allows the process to move up and randomly sample the parameter space and then slowly anneal downwards towards the global minimum. Note that the results above are mostly preliminary. The algorithm does well with the Mona Lisa and Van Gogh data sets, but may need a week or more for a data set as large as LRB744710.

[5]Times are the amount of time the GPU algorithm was allowed to run

[6]A CPU year is defined as the time a 1 GHz processor would take to execute the instructions needed to find a solution.

[7]The results in table 1 are preliminary, given more time the algorithm's trip will become much closer to the optimal trip

Data Set	Size	GPU Optimal	Optimal	Time	Percent Difference
Mona Lisa	100,000	5,971,924	5,757,191	1.66 Hours	3%
Van Gogh	120,000	6,784,443	6,543,610	7.93 Hours	4%
Venus	140,000	7,041,669	6,810,665	2.32 Days	3%
Earring	200,000	8,901,054	8,171,677	3.93 Hours	9%
LRB744710	744,710	1,894,869	1,611,233	16.2 Hours	17%

Table 1: The data set, size, optimal solution found via the GPU algorithm, optimal known solution, time of GPU algorithm, and percent difference of GPU and best score.

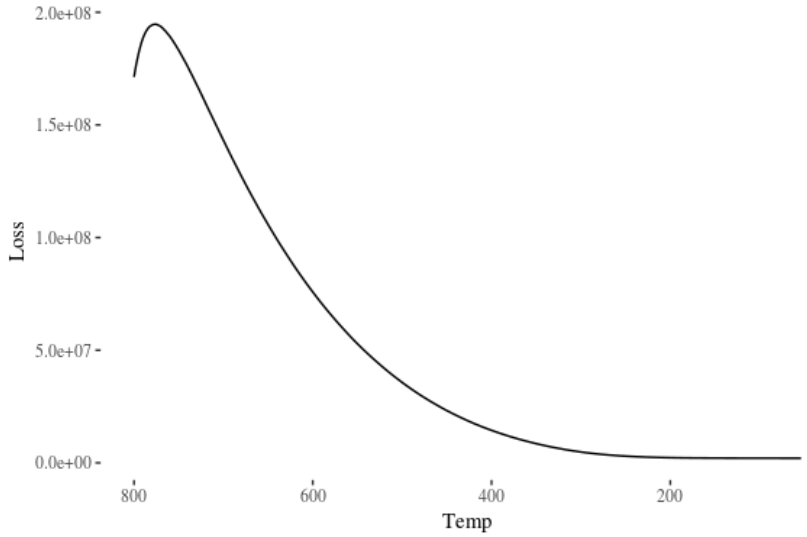


Figure 8: An example of the loss relative to the temperature for LRB744710 with decay rate .993%

(a) The stippled Mona Lisa

(b) The Mona Lisa drawn with one continuous line

Figure 9

6 Conclusion and Future Work

This paper proposes and illustrates a Metropolis Hastings style algorithm on a GPU. Using the 'first past the post' method for parallel acceptance of multiple proposal steps and maximizing throughput of the GPU reduces the downtime of having a rejected step and allows the algorithm to search a large area of the parameter space. The example in this paper uses simulated annealing on the traveling salesman problem. Solutions are found to be within acceptable values for an approximation of optimal solutions while being much faster than CPU implementations. Approximations could be made much better by implementing algorithms such as the Lin-Kernighan TSP heuristic [8]. In addition, the technology used in this experiment, the single GPU, is much cheaper than the 30 to 40 or upwards of multi-core processors current traveling salesman problem solvers need to solve data sets of the size used in this paper. When the loss functions starts to make less than 1% gains, it may be useful to parallelize at the CPU level by using multiple GPUs. As the loss functions decreases, the probability of the algorithm finding a better trip decreases at an exponential rate. Having multiple GPUs to search over the parameter space will allow the algorithm to converge to the global optimum much faster.

An improvement to this algorithm would be to start with a 'smart' first trip. Instead of a simple first trip, it is possible to use a simpler algorithm to design an intelligent starting trip. This would reduce the time necessary to compute the optimal trip. One possible future area of work would be to use this for video processing. Given a video, stipple each frame so it has the same amount of points, then run the algorithm over each frame. Each frame's optimal trip will be similar to the last frame's optimal trip. The first frame's end trip is then used for the second frames starting trip, etc. heavily reducing the time of the algorithm.

References

[1] David Applegate, William Cook, Sanjeeb Dash, and André Rohe. Solution of a min-max vehicle routing problem. *INFORMS Journal on Computing*, 14(2):132–143, 2002.

[2] Robert Bosch and Adrianne Herman. Continuous line drawings via the traveling salesman problem. *Operations Research Letters*, 32(4):302 – 303, 2004.

[3] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[4] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[6] Edward I. George George Casella. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[7] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[8] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106 – 130, 2000.

[9] L. Ingber. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29 – 57, 1993.

[10] David S. Johnson and Lyle A. Mcgeoch. *The Traveling Salesman Problem: A Case Study in Local Optimization*. 1997.

[11] Sparsh Mittal and Jeffrey S. Vetter. A survey of cpu-gpu heterogeneous computing techniques. *ACM Comput. Surv.*, 47(4):69:1–69:35, July 2015.

[12] G. Reinelt. TSPLIB- a traveling salesman problem library. *ORSA Journal of Computing*, 3(4):376–384, 1991.

[13] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[14] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[15] Alexander Schrijver. On the history of combinatorial optimization (till 1960). In G.L. Nemhauser K. Aardal and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 1 – 68. Elsevier, 2005.

[16] Adrian Secord. Weighted voronoi stippling. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering*, NPAR '02, pages 37–43, New York, NY, USA, 2002. ACM.

[17] Andrew Sohn. Parallel satisfiability test with synchronous simulated annealing on distributed-memory multiprocessor. *J. Parallel Distrib. Comput.*, 36(2):195–204, August 1996.

7 Appendix

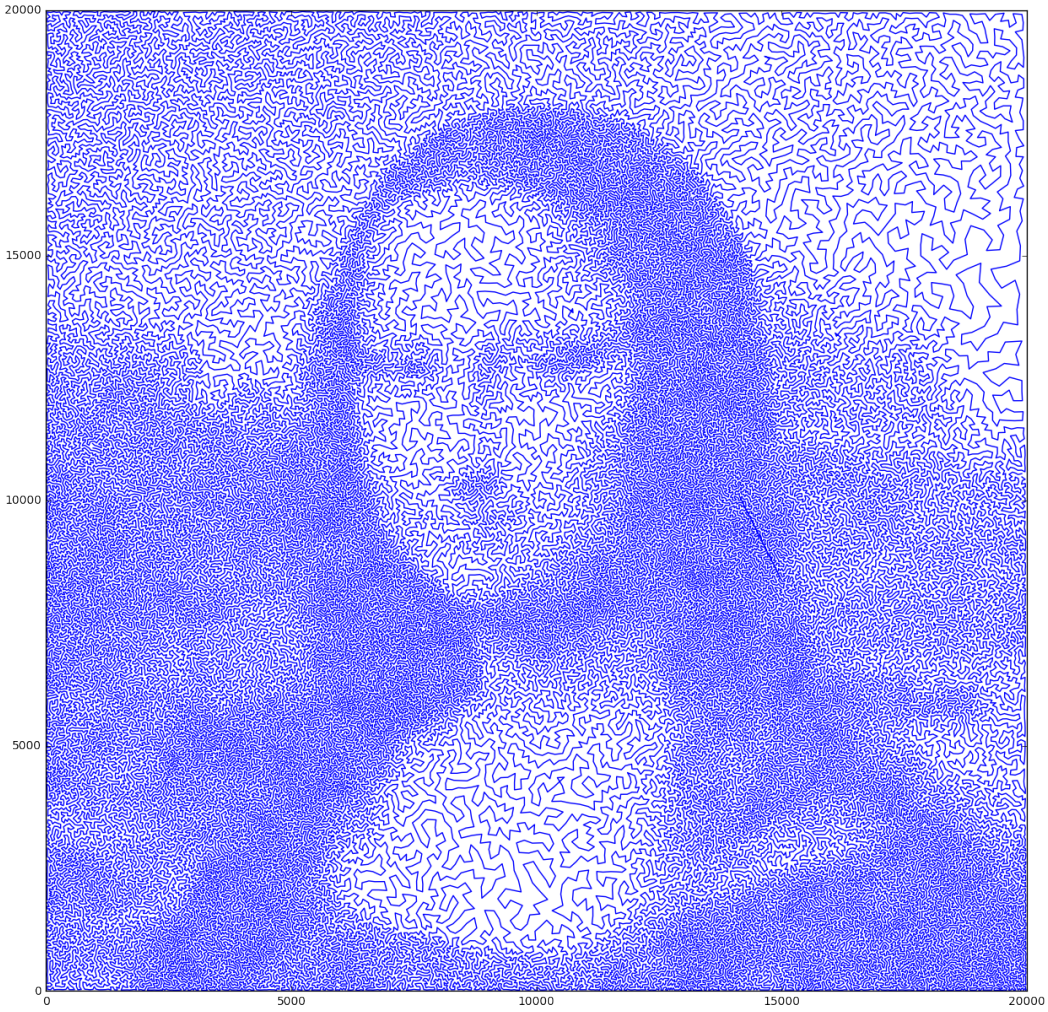


Figure 10: A closer view of the Mona Lisa

Figure 11: A closer view of the Van Gogh

Figure 12: A closer view of Venus

Figure 13: A closer view of the Earring

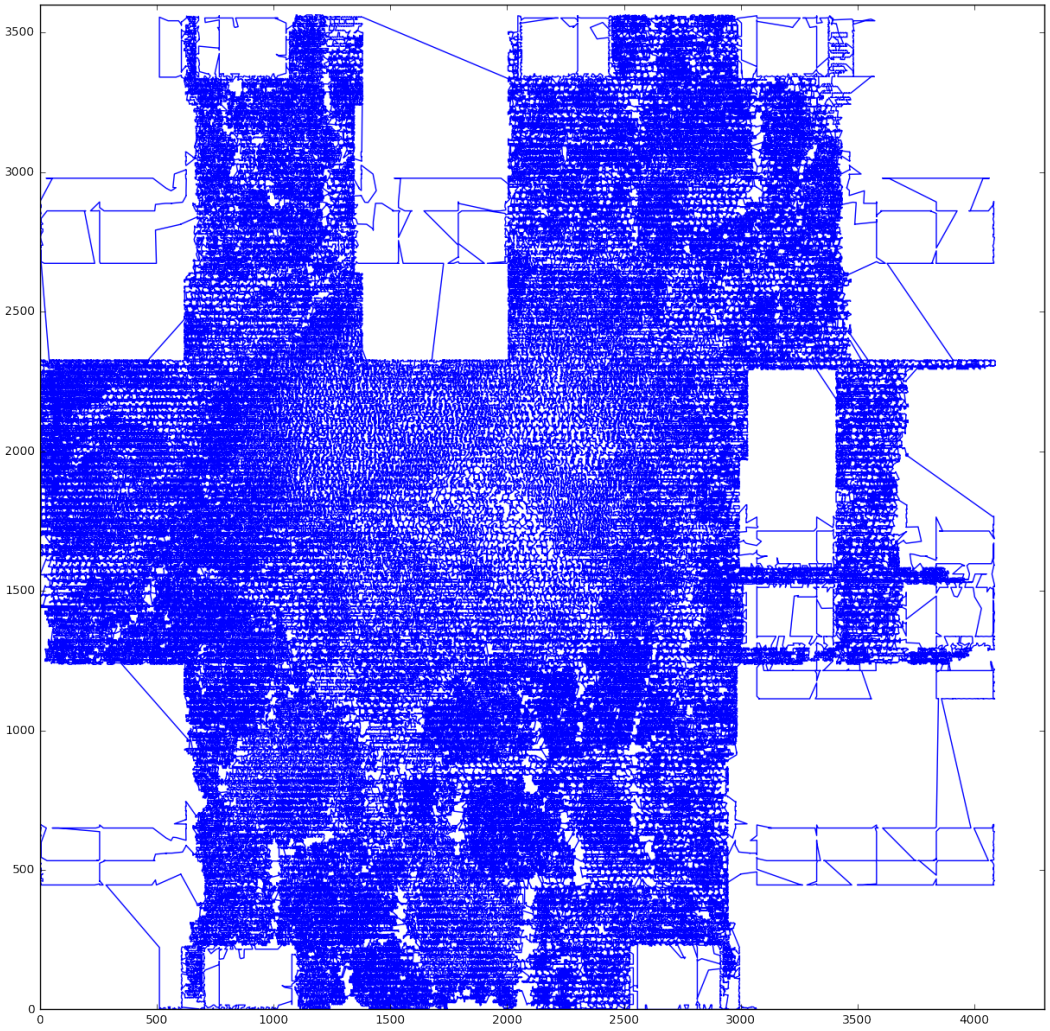


Figure 14: A closer view of lrb744710