

Hardware Accelerators and Optimization Algorithms for Unconventional Computing

Philip Canoza



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-3

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-3.html>

January 13, 2023

Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to thank Saavan Patel for his help and guidance throughout my time in Berkeley, both in undergrad and graduate school. I couldn't have become a grad student without him.

I also would like to acknowledge him and the rest of the PASSO team, Adihraj Datar and Steven Lu, for their hardwork to make the PASSO chip a reality.

I would also like to acknowledge Prof. Jiantao Jiao for being a reader for this report.

I would like to give many thanks to Prof. Sayeef Salahuddin for being my graduate school advisor. Thank you for mentoring me during good times and being supportive in tough times. I couldn't have asked for a better

advisor.

Finally, I would like to thank my friends and family for their support, without which I would not be where I am today.

Hardware Accelerators and Optimization Algorithms for Unconventional Computing

by Philip Canoza

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

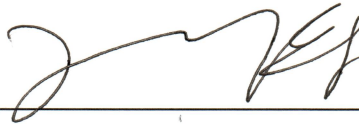
Committee:



Professor Sayeef Salahuddin
Research Advisor

1/12/2023

(Date)



Professor Jiantao Jiao
Second Reader

12/30/2022

(Date)

Hardware Accelerators and Optimization Algorithms for Unconventional Computing

Philip Canoza

Abstract

Ising machines have emerged as a new paradigm in unconventional computing that can solve NP-Hard problems. This report looks at both the hardware and algorithm design of these Ising machines. First we will take a look at the implementations of two different hardware accelerators. The first accelerator is a digitally-synthesized, Field Programmable Gate Array (FPGA) that performs synchronous Gibbs sampling of a Restricted Boltzmann Machine (RBM). The second accelerator is an asynchronous neural network that uses a mixed-analog signal neuron to drive stochastic local updates. On the algorithmic side, we explore mappings of the Travelling Salesman Problem. In particular, we introduce a Quadratic Unconstrained Binary Optimization (QUBO) mapping that uses the classical k -opt algorithm to perform local search.

Acknowledgements

I would like to thank Saavan Patel for his help and guidance throughout my time in Berkeley, both in undergrad and graduate school. I couldn't have become a grad student without him.

I also would like to acknowledge him and the rest of the PASSO team, Adihraj Datar and Steven Lu, for their hardwork to make the PASSO chip a reality.

I would also like to acknowledge Prof. Jiantao Jiao for being a reader for this report.

I would like to give many thanks to Prof. Sayeef Salahuddin for being my graduate school advisor. Thank you for mentoring me during good times and being supportive in tough times. I couldn't have asked for a better advisor.

Finally, I would like to thank my friends and family for their support, without which I would not be where I am today.

Contents

1	Introduction	4
1.1	Motivation	4
1.2	Outline	4
2	Background	4
2.1	Ising model	4
2.2	Sampling as Optimization	5
3	Hardware Accelerators	6
3.1	FPGA Accelerator	6
3.1.1	Overall Architecture	6
3.1.2	FPGA Programming	6
3.1.3	Matrix Multiplication with Mask	7
3.1.4	Sigmoid Approximation	7
3.1.5	Pseudo-Random Number Generation	7
3.2	PASSO	8
3.2.1	Overall Architecture	8
3.2.2	Stochastic Neuron Circuit	8
3.2.3	IO Ring Design	9
4	Travelling Salesman Problem	9
4.1	TSP as a QUBO	12
4.2	Stochastic Local Search and k -opt	14
4.3	QUBO Local Search	15
4.4	Benchmark comparison	16
5	Conclusion and Future Work	18

1 Introduction

1.1 Motivation

As the demand for big data increases and the speed of traditional CPUs cannot keep pace, a new processing paradigm is needed to tackle computing’s most difficult problems. There is burgeoning research examining these new paradigms, which uses the term “unconventional computing”. In particular, NP-Hard optimization problems are ones that both scale exponentially and have relevant practical applications in a variety of fields. A growing class of quantum-inspired classical computing solvers (sometimes called “Ising machines”) have taken up this challenge, using hardware platforms such as Quantum Annealing [1] and the Optical Pumping [2]. However, these platforms such as these are plagued by two issues:

1. infeasible hardware scaling to large-scale problems and
2. benchmarks for toy optimization problems such as Max Cut and infeasible algorithmic scaling for real-world problems.

My research aims to tackle these scaling issues by engaging in both hardware and algorithm design for Ising-based accelerators.

1.2 Outline

This technical report is organized as follows: first we give some background on the Ising model and its optimization in Section 2. Then in Section 3 we examine the implementation of two hardware accelerators that can solve the Ising model: a digital Field Programmable Gate Array (FPGA) neural network accelerator, previously documented in [3]; and an asynchronous, mixed-signal neural network, previously documented in reports by Datar [4] and Lu [5]. In this report, we will focus on parts of the hardware accelerators where I contributed in the implementation. To see how these accelerators may be applied, in Section 4 we look at relevant mappings and algorithms that can solve the Travelling Salesman Problem (TSP). Finally, we conclude with some final remarks and possible future work.

2 Background

2.1 Ising model

The Ising model has its roots in statistical physics as a model of interacting spin magnets. Finding the minimum energy of an Ising model is an example of a combinatorial optimization problem that is NP-Hard[6][7][8]. Thus, if there is a mapping of a NP-Hard problem onto an Ising formulation, minimizing the energy of the Ising model corresponds to solving the NP-hard problem. Ising problems can be formulated as the following: let n be the number of spins/states. We must minimize the energy function $E(s)$ over bipolar states $s \in \{-1, 1\}^n$. The energy can be written as

$$E(s) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n J_{ij} s_i s_j + \sum_{i=1}^n a_i s_i \tag{1}$$

where J_{ij} is usually referred to as the weight matrix and a_i is a bias vector.

Through out this paper, we will describe these problems using Ising, Quadratic Unconstrained Binary Optimization (QUBO), and Boltzmann Machine somewhat interchangeably, as they all have equivalent representations. The main difference is that often QUBO and Boltzmann machine formulations use binary-variables $\{0, 1\}^n$. The conversion from Ising to Boltzmann Machine is fairly straight-forward[9]. To convert to new weights W^* and biases a^* we have

$$W^* = 4W, \quad a^* = 2(a - W\mathbf{1}) \tag{2}$$

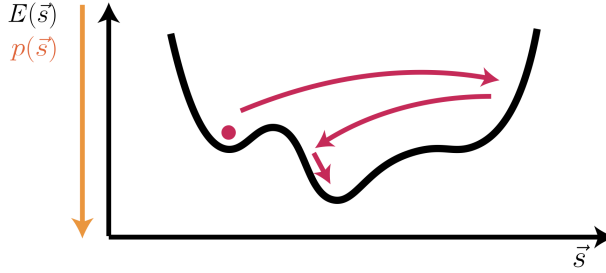


Figure 1: Example of energy landscape being explored by a stochastic sampling algorithm. Lowest energy state corresponds to the mode of the distribution.

To convert to a QUBO formulation, we simply need to absorb the bias vector into the weight matrix representation W' and write the minimization problem in the form

$$x^T W' x = \sum_{i=1}^n \sum_{j=1}^n W'_{ij} x_i x_j \quad (3)$$

One example of an Ising/QUBO problem that is prevalent in literature is Max Cut. It's formulation can be found in references [8] and [10]. Suppose we are given the graph (V, E) . Then we map each vertex $v_i \in V$ to a spin s_i . If $s_i = 0$, then v_i belongs to one side/set of the Max Cut. If $s_i = 1$, then v_i belongs to the other set. Then we simply must minimize the following energy

$$E(s) = - \sum_{(i,j) \in E} (s_i + s_j - 2s_i s_j)$$

The Max Cut problem is often used for benchmarking Ising Machines due to the straightforward mapping of the adjacency matrix with edges E to the Ising weight matrix.

2.2 Sampling as Optimization

Generally, our research group's approach to solving Ising problems can loosely be described as hardware-accelerated sampling techniques. This is done by mapping the Ising model to a Boltzmann Machine, what is a probabilistic, neural network model. A more structured intro to Boltzmann Machines can be found in this reference [11]. The probability of a state of the Boltzmann machine is related to the Ising energy in Equation 1 through the following:

$$p(s) = \frac{1}{Z} e^{-E(s)}$$

where Z is a normalizing constant. Thus, sampling high probability states of the Boltzmann machine corresponds to finding minimum energies of the Ising model, as seen in Figure 1.

This sampling can take many forms, with a basis in Markov Chain Monte Carlo (MCMC) algorithms. For example, if we restrict the Boltzmann Machine representation to a bipartite graph, we can perform updates in just two steps using Gibbs sampling. This is the basis of the sampling algorithm for the first hardware accelerator in Section 3. Alternatively, we can let the network locally update in continuous time, following Glauber Dynamics [12]. This is the basis of the analog mixed-signal hardware in the second part of Section 3.

There are many other ways to solve the Ising model. For a more complete literature review of Ising machines, we refer the reader to the review by Mohenshi et. al. [13].

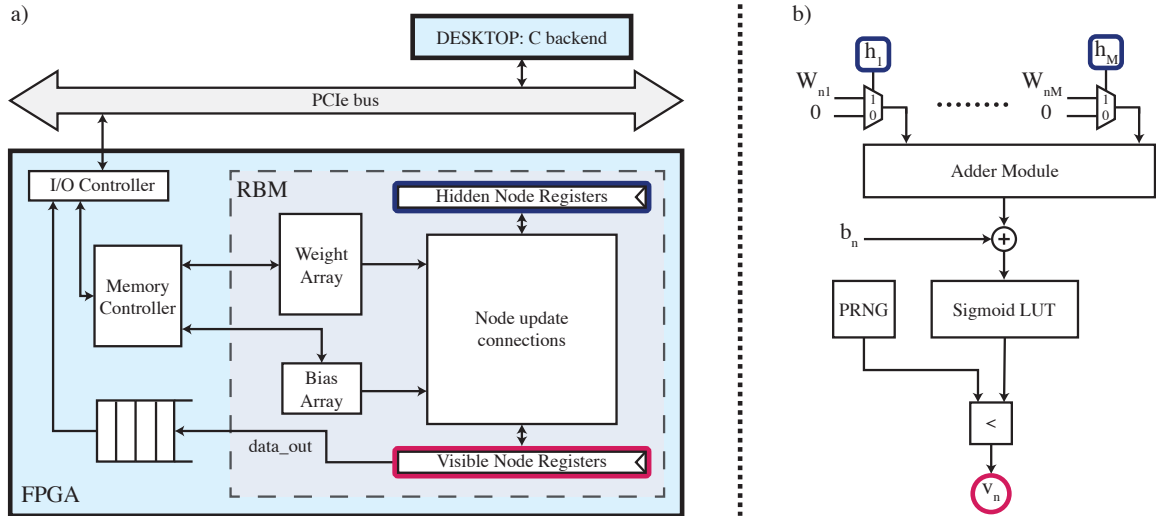


Figure 2: Memory and compute hierarchy of the FPGA accelerator (left), and an example of a node update connection (right)

3 Hardware Accelerators

In this section we will take a look at 2 different types of sampling-based hardware accelerators that can solve Ising problems.

3.1 FPGA Accelerator

First we will examine a digital, FPGA-based hardware accelerator for the Restricted Boltzman Machine (RBM). The majority of this section can be found in Patel et. al. [3]. This architecture has been used in optimization problems such as integer factorization [3] and Max-Cut [9].

3.1.1 Overall Architecture

Figure 2 shows the overall architecture used in the FPGA accelerator. On the left is a diagram of the memory and compute hierarchy. The RBM consists of memory to hold the weight, bias, and clamp values, registers to hold the node values, and circuitry to perform the node updates, which take up the bulk of the resources. The output is buffered to the IO controller that communicates results to the PCIe bus. A C backend reads in the data stream from PCIe, and can program the weights and biases from the memory controller.

The right part of Figure 2 shows an example of a node update connection. Given M hidden nodes, the figure depicts the circuitry to update the n^{th} visible node. The hidden nodes binary mask the n^{th} row of the weight matrix. The results are accumulated in the adder module and added to the n^{th} visible bias. It is then passed through a sigmoid look-up table (LUT) and compared to the output of a pseudo-random number generator (PRNG) to update the value of the visible node.

3.1.2 FPGA Programming

At the heart of the FPGA is the RBM computing core which performs the Gibbs sampling algorithm. All programming was done using the Xilinx Vivado suite on the on the Xilinx Virtex UltraScale+ XCVU9P-L2FLGA2104. The core was designed to output the most samples for the RBM sizes we had. The weights and biases are stored in on-chip SRAM to decrease access time. The values are broadcast to the node update modules each cycle, which performs the necessary operations for the sampling and take up the bulk of the computation resources. There are no pipeline or data hazards, removing the need for any complex timing schemes. Thus, if we instantiate a node update module for every node register, there is a new sample taken

from the visible node registers every clock cycle, taking full advantage of the RBM’s parallelism.

Each node update module contains the logic to perform a matrix-row multiplications, a sigmoid function, and a comparison with a random number. The matrix-row multiplication is performed via a binary mask and an adder module that accumulates the surviving weight values. Once each weight value is masked appropriately, they are passed into an adder tree which accumulates the results of each multiplication. The accumulators take the majority of LUT resources on the FPGA, and represent the bottleneck in the computation. In our implementation, we use single cycle accumulation, but multi-cycle accumulation is possible to save on hardware resources while scaling up for larger RBM sizes. A fixed point sigmoid function is implemented as a LUT, which allows for speed without expensive hardware operations. A Python script generates the LUT Verilog code in order to test different bit lengths and fixed point locations. Finally, a linear feedback shift register (LFSR) generates a pseudo-random number. The number is compared to the output of the sigmoid LUT and the relevant node is updated with the boolean result.

Results from the bank of visible node registers is buffered to an IO controller with a FIFO. The IO controller also uses a memory-mapped interface that can program the weights, clamps, and biases. The controller communicates to our desktop via PCIe. A simple PCIe link is provided through a Xillybus IP Core [14] which provides up to 800 MB/s data transfer rate. This is a sufficient speed to get all of the sample data off of the FPGA, but faster speeds are possible for future implementations. To handle the large data stream from the bus, a C backend was created to serve data to existing Python code for RBM analysis. This C backend is used for analysis of data and to judge the quality of the solution on the FPGA. This FPGA pipeline provides an efficient method for solving the problems of interest, where the limiting factor in computation speed can become the FPGA sampling speed.

Next we will highlight salient aspects of the FPGA design that made the RBM algorithm amenable to hardware acceleration: matrix multiplication, sigmoid approximation, and the random number generation.

3.1.3 Matrix Multiplication with Mask

The RBMs binary activations and fixed point weights allow for a very efficient matrix multiplication module. The binary activations convert multiplications into a binary mask, or a 2-to-1 mux using the activation value as the switch. This results in multiplications reducing to atomic operations on the FPGA, greatly reducing their area, power and latency. The usage of fixed point weights, instead of 32 bit floating point, is estimated to decrease the area of the accumulator circuits by 8x [15]. The smaller area of each component allows us to use a larger adder tree with less delay for accumulation as compared to 32 bit floating point operations, resulting in more computation which can be completed in one cycle.

3.1.4 Sigmoid Approximation

Exact calculation of the sigmoidal activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ is computationally expensive. To accomplish direct calculation, at least 3 extra hardware instructions are needed, exponentiation, addition, and division, which all incur a large hardware cost both in terms of latency and area. Instead, binary sigmoid values are precomputed and enumerated in a look up table (LUT) for use in the FPGA. This implementation allows for fast evaluation of the activation function without expensive hardware resources. After matrix multiplication and bias addition, the computed value is passed through the LUT based activation function to approximate the sigmoid. The Look Up Table values are hard-coded at synthesis time and thus do not use any LUT resources. Finally, the sigmoids are synthesized as 9 7-input Multiplexers (F7 MUX) within 6 Combinational Logic Blocks (CLBs) amounting to 1% of the total available F7 MUXs, and <1% of the total available CLBs. This is a common technique used in many FPGA and ASIC based neural network accelerators, which is further adapted for our particular FPGA implementation. [16, 17, 18, 19] .

3.1.5 Pseudo-Random Number Generation

To generate high quality samples, uncorrelated random numbers need to be produced every cycle. To accomplish this we use a 32 bit length Linear Feedback Shift Register (LFSR) pseudo random number

generator. The 32-bit LFSR size creates $2^{32} - 1$ random bits, or $2^{29} \approx 5 \times 10^8$ random 8 bit numbers. To determine the best LFSR size, we characterized performance for the sampling algorithm by varying the LFSR length and determining performance on the factorization problem. These results are presented in Supplementary Figure 6. As, the total cost of these LFSR based random number generators amounts to just 5% of the design flip flop usage, and 2% of the lookup table usage we chose a longer than necessary LFSR chain to minimize accuracy and performance problems from this element of the design. Each neuron has its own LFSR and is seeded with a different value to minimize the possibility of correlation. Other PRNG algorithms can produce higher quality random numbers [20, 21], but they require greater hardware resources and are generally more complex. Based on our experiments, we found the LFSR random number generator to be the simplest, as well as most performative pseudo-random number generator available on the FPGA.

3.2 PASSO

Next, let’s take a look at an asynchronous neural network made of stochastic, analog mixed-signal neurons. This architecture is referred to as the Parallel Asynchronous Stochastic Sampling Optimizer (PASSO). This accelerator was taped-out in Spring 2021 by Saavan Patel, Adhiraj Datar, Steven Lu, and myself. A more detailed overview of the digital system design can be found in Datar’s technical report[4], and an analysis of the analog circuit’s power, performance, and area can be found in Lu’s technical report [5].

3.2.1 Overall Architecture

Lu provides a good overall summary of the architecture [5]. The PASSO processor consists of a few main blocks: the main analog core consisting of the main 16x16 fabric of 256 neurons, a small cluster of 4 neurons for testing, SRAM, circuitry for streaming out outputs serially, and IO circuitry.

Before using the processor to solve a particular trained problem of interest, the processor first needs to be properly configured. The configuration bits must be shifted into the configuration shift registers, which are connected to form a long chain for the entire processor. There are 74 configuration bits for each neuron, 7 bits for each of the 32 trimmable current biases used for biasing the neurons, and 3 bits that encode the sampling frequency and number of neurons sampled for the streamout of the processor, totaling to 19171 configuration bits [4].

To move the outputs of the neurons in the analog core off-chip for data processing and analysis, the asynchronous neurons must be sampled with a sampling clock. The nominal sampling frequency is 300 MHz, but only 16 neurons can be sampled at once at this frequency due to IO limitations. The 3 bits used to configure the sampling of the neurons defines presets of sampling frequency and the number of neurons sampled, ranging from 16 neurons at 300MHz to all 256 neurons at 18.75 MHz, maintaining a constant throughput of 4.8 Gsamples/sec [4].

Since the IO circuitry limits the speed of the off-chip data transfer, the sampled neuron outputs are first written in burst mode into an SRAM buffer at the sampling clock frequency (maintaining the fixed 4.8 Gsamples/sec throughput) [4]. The SRAM buffer is then read out at a slower frequency to meet the IO speed specifications (20MHz IO clock) and serialized. In the case of a 300MHz sampling frequency producing 16-bit samples, the SRAM is read at a frequency of $20/16 = 1.25$ MHz [4].

3.2.2 Stochastic Neuron Circuit

We can see a schematic of the mixed-signal neuron in Figure 3. We use the inherent shot noise present in advanced CMOS nodes to create a binary, stochastic neuron circuit, which we couple with adjacent neurons through a digital fabric in an analog-mixed signal architecture. Each individual neuron amplifies noise to create a Poisson clock that is combined with the output of the neuron synapse, pushed through a sigmoid, and digitized. This is represented in Figure 3, where the green box demonstrates a high-level circuit for the neuron.

$$P(v_i = 1) = \sigma(W^T h + b_v)$$

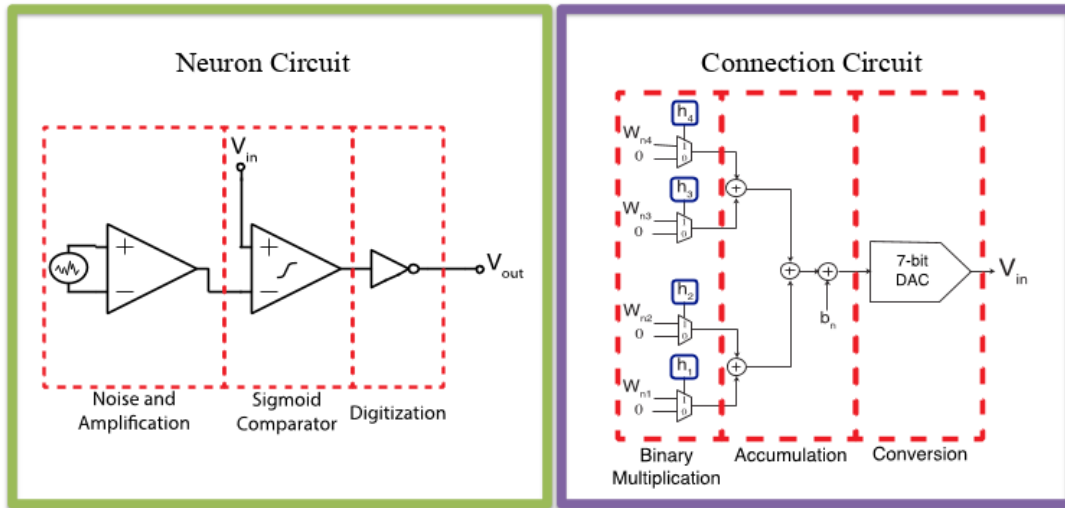


Figure 3: Schematic of mixed-signal neuron, both the stochastic analog portion (left) and the digital synapse circuitry (right)

The neuron synapse is composed of a digital multiply-accumulate to calculate the activation probabilities, followed by a digital-to-analog converter (DAC) to convert the sum to a bias voltage for the neuron. The binary activations allow the multiplications to be converted to MUX and accumulate, vastly decreasing the hardware cost. The neuron weights are stored in a distributed memory system which allows the architecture to overcome problems arising from the von-Neumann bottleneck, as weights are stationary throughout computation. This type of computation is shown in the purple box in Figure 3 in a fully combinational (clock-free) design.

We can see the behavior of the mixed-signal portion of the neuron in Figure 4. Here, we modulate the input to the analog neuron. At low voltages (red), the neuron output is clamped to logic zero. At middle voltages (green), the neuron has a stochastic output that's on average 0.4 V. At higher voltages (purple), the neuron still has a stochastic output but mostly outputs a logic one. A closer look at the neuron layout can be found in Figure 5, where you can see the digitally-synthesized synapse connect to the analog neuron via DAC.

3.2.3 IO Ring Design

To communicate with the outside world, the chip is connected to a multi-voltage IO ring. The IO ring consists of drivers from the ARM GPIO library for GlobalFoundries GF12LP. This IO ring is connected to C4 (flip chip) SNAG180 (tin-silver bumps with greater-than $180\mu\text{m}$ inter-bump spacing) IO Bumps through the redistribution layer routing (RDL) of the final metal layer. The signals associated with each bump can be found in Figure 6. In total, there are 64 IO bumps arranged in an 8×8 grid with a pitch of $230\mu\text{m}$. A total view of the final die layout can be seen in Figure 7. You can also see the 64 IO bumps and RDL routing in the chip arrival photo.

4 Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is perhaps one of the most widely studied problems in combinatorial optimization and serves as a testbed for new algorithmic ideas. It has many practical applications,

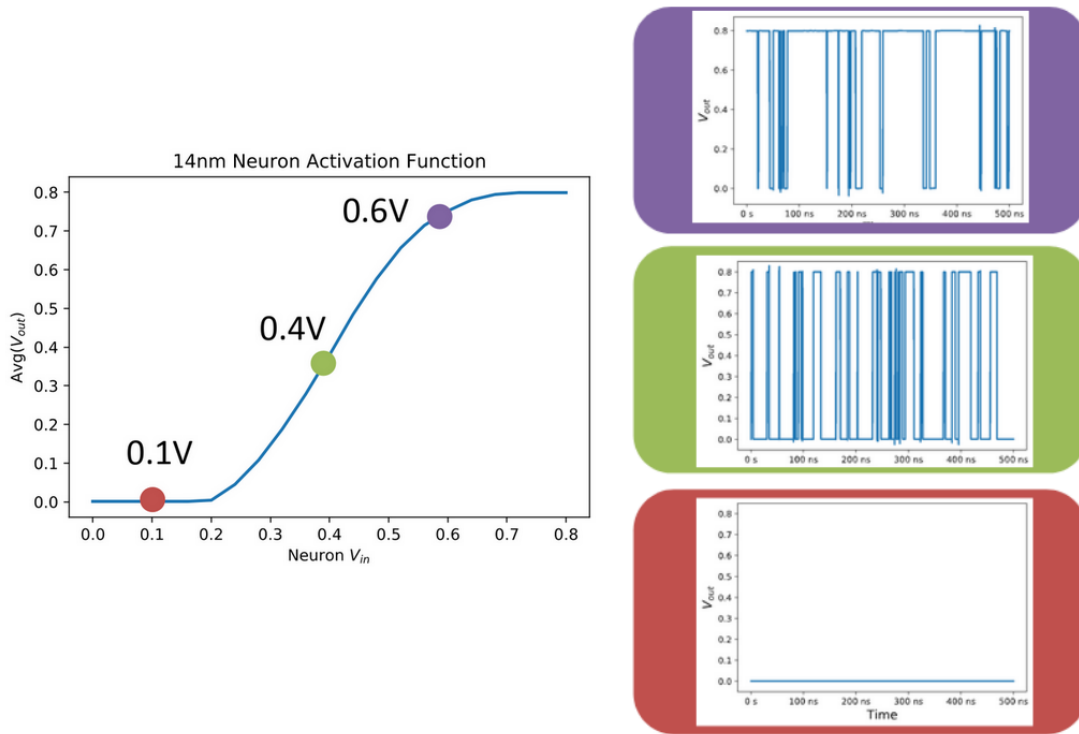


Figure 4: Time series of neuron switching activity with different synapse voltages

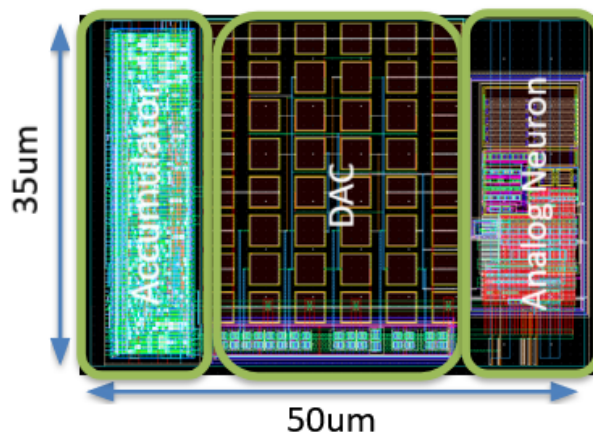


Figure 5: Neuron layout. Here you can see the digitally-synthesized accumulator synapse, the analog neuron, and the DAC that connects them

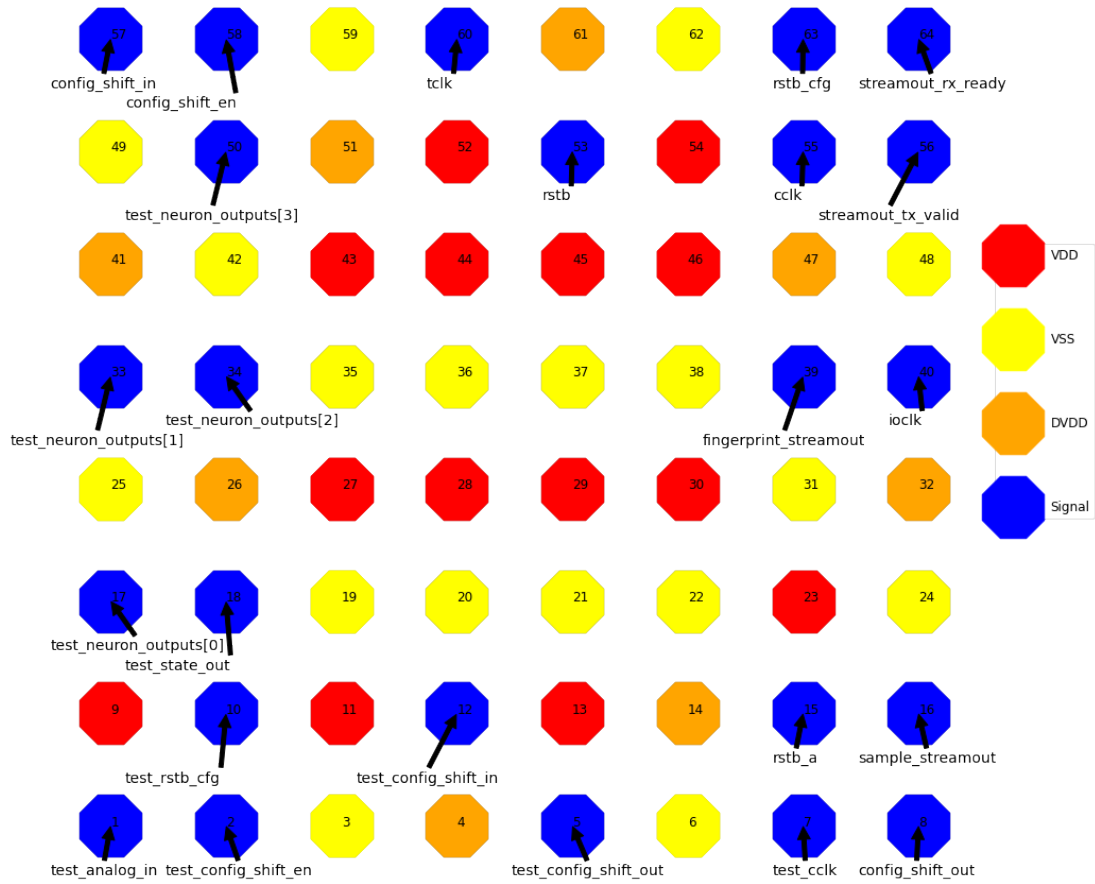


Figure 6: C4 Bump layout for chip IO

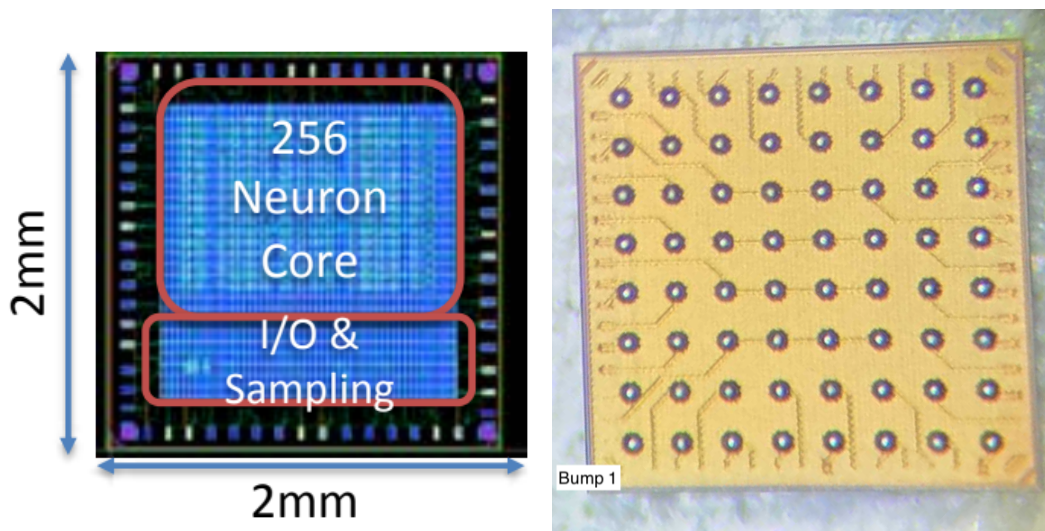


Figure 7: Final die layout (left) and chip arrival photo (right)

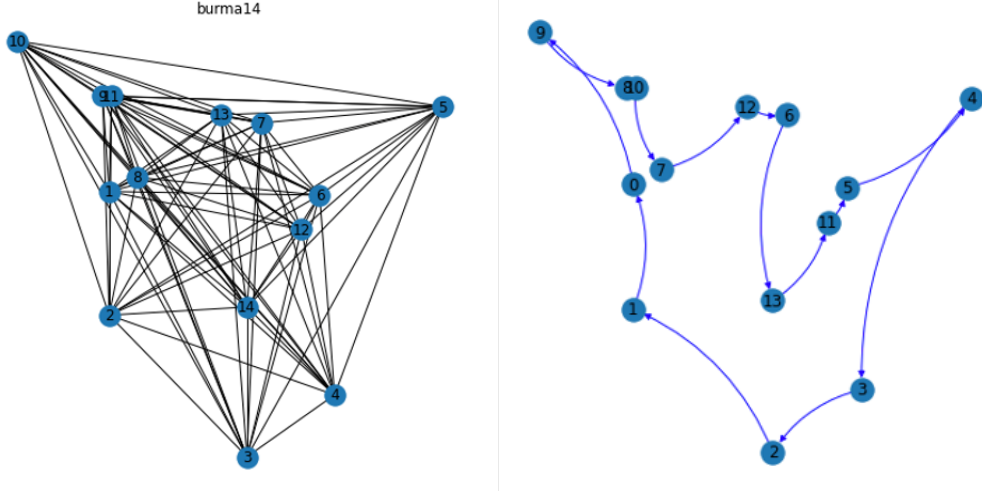


Figure 8: Example TSPLIB95 problem, burma14, as a fully connected graph (left), and an example solution to the TSP known as a “tour” (right)

from vehicle routing and logistics to minimizing drill paths in printed circuit board manufacturing. It is conceptually easy to understand yet NP-hard to solve, and thus makes a perfect application to explore various Ising-based optimization methods.

The TSP problem can be formally defined as follows [22]: given an edge-weighted, completely connected, directed graph $G := (V, E, w)$, where V is the set of $n := \#V$ vertices, E is the set of directed edges, and $w : E \mapsto \mathbb{R}^+$ a function assigning each edge $e \in E$ a weight $w(e)$, the TSP problem is to find a minimum weight Hamiltonian cycle in G ; that is, a cyclic path that contains each vertex exactly once and has minimal total weight.

Many Ising-based accelerators have explored the TSP problem. For instance, Feld et. al. has explored it’s use as part of a hybrid method to solve the Capacitated Vehicle Routing Problem (CVRP) [23]. However, in this paper, we will examine it’s simplest application: finding the optimal round trip through a number of geographical locations, where vertices are “cities”, edge weights are “distances” and the cyclic path is referred to as a “tour”. To evaluate performance of our optimization algorithms, we will benchmark against instances from TSPLIB95 [24]. An example TSPLIB95 instance can be seen in Figure 8

4.1 TSP as a QUBO

The TSP can be formulated as a Ising problem in the following way [8]: for a N city problem, we will use N^2 variables $x_{v,i}$ where v represents the vertex and i represents its order in the cycle. The Hamiltonian for the problem is given by $H = H_A + H_B$ where

$$H_A = A \sum_{v=1}^N \left(1 - \sum_{j=1}^N x_{v,j} \right)^2 + A \sum_{j=1}^N \left(1 - \sum_{v=1}^N x_{v,j} \right)^2 + A \sum_{(uv) \notin E} \sum_{j=1}^N x_{u,j} x_{v,j+1} \quad (4)$$

and

$$H_B = B \sum_{(uv) \in E} W_{uv} \sum_{j=1}^N x_{u,j} x_{v,j+1} \quad (5)$$

where A and B are large, positive constants. Equation 4 encodes the constraints for the Hamiltonian cycle, where the first term requires each city to occur only once, the second term enforces that each position in the

	A1	A2	A3	A4	B1	B2	B3	B4	C1	C2	C3	C4
A1	-1 -1	2	2	2	2	D_{ab}	0	D_{ba}	2	D_{ac}	0	D_{ca}
A2		-1 -1	2	2	D_{ba}	2	D_{ab}	0	D_{ca}	2	D_{ac}	0
A3			-1 -1	2	0	D_{ba}	2	D_{ab}	0	D_{ca}	2	D_{ac}
A4				-1 -1	D_{ab}	0	D_{ba}	2	D_{ac}	0	D_{ca}	2
B1					-1 -1	2	2	2	2	D_{bc}	0	D_{cb}
B2						-1 -1	2	2	D_{cb}	2	D_{bc}	0
B3							-1 -1	2	0	D_{cb}	2	D_{bc}
B4								-1 -1	D_{bc}	0	D_{cb}	2
C1									-1 -1	2	2	2
C2										-1 -1	2	2
C3											-1 -1	2
C4												-1 -1

Figure 9: Example TSP QUBO matrix from Feld et. al.[23]. A1 corresponds to city A in position 1 of the Hamiltonian cycle. The red terms correspond to the first term in Eq. 4, the green terms correspond to the second term in Eq.4, and the blue terms correspond to the terms from Eq. 5

FPGA Platform	Memory Element	Memory Size	Max Num. Cities
Xilinx Ultrascale	CLB FlipFlops	2,364 Kb	23
	Max Distributed RAM	6.1 Mb	29
	Total BRAM	75.9 Mb	55
	UltraRAM	270 Mb	76
Intel Stratix 10	Max Embedded Memory	239.5 Mb	73
	HBM	16 GB	211

Table 1: Scaling of the max number of TSP cities that can be solved given the FPGA memory element that is storing the QUBO matrix.

cycle must be assigned exactly one city, and the third term ensures that only valid edges are used. Equation 5 ensures that we are finding the Hamiltonian cycle of minimum length. When choosing penalty values A and B , a good choice is to have $0 < B \cdot \max(W_{uj}) < A$ [23]. An example of the resulting QUBO matrix of a 3 city problem can be seen in Figure 9.

This mapping gives a good starting point to solving the TSP with our hardware from Section 3. However, we are immediately plagued by two problems. Problem one is the quadratic scaling of the mapping. The N^2 scaling of the number of QUBO variables corresponds with a N^4 number of coefficients to store in hardware. Suppose we have 8 bit weights for our QUBO matrix. Table 1 shows how we are memory-limited given a FPGA platform. Even if we were to use all 16 GB of High Bandwidth Memory (HBM) available on a Intel Stratix 10, we could only solve a 200 city TSP problem. Thus, this mapping fails to solve problems of reasonable size given the hardware constraints.

Problem two is the quality of solutions to the QUBO formulation as we scale up. Experiments show that as the problem size increases, the solution quality significantly deteriorates, and the probability of obtaining an optimal solution is very low for TSPs of any useful size [25]. Figure 10 shows the performance of a

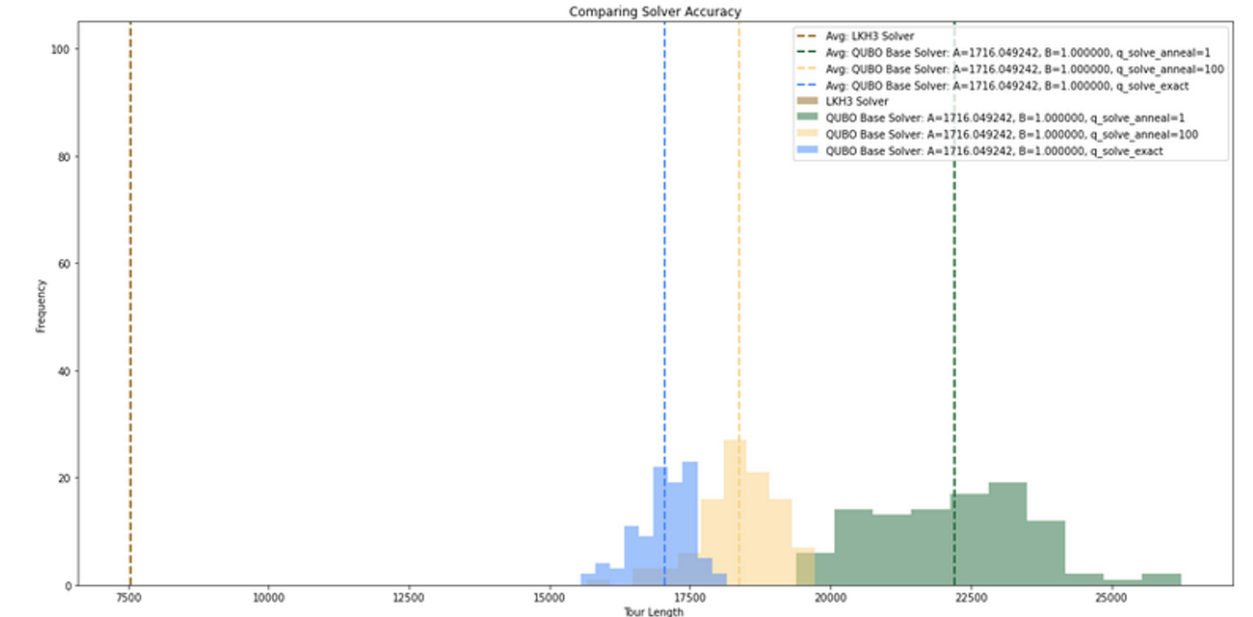


Figure 10: Performance of the TSP QUBO mapping on TSPLIB95 problem berlin52. The histograms show the distribution of tour lengths found by multiple runs of the QUBO solver. As the length of the annealing time increases, the proposed tour length decreases but fails to reach the optimum tour length

annealing-based QUBO solver on a 52 city problem (berlin52 from TSPLIB95). We see that as we increase the anneal time, the tour length decreases and thus the solution quality increases. However, it fails to come close to the optimal tour length.

4.2 Stochastic Local Search and k -opt

To address the issues of the TSP QUBO mapping, we will look to existing classical solving methods. In practice, even state-of-the-art TSP solvers do not solve an entire TSP problem at once. Many methods might start with a first guess as a solution, the repeatedly perform small changes with the goal of improving the solution quality. Such methods are known as stochastic local search (SLS) [22]. There is a wide variety of SLS algorithms, but here we will take a look at iterative improvement algorithms. In this scheme, the candidate solution is improved by searching a local neighborhood for a solution that decreases in the loss function. The candidate solution is iterated upon until we reach a local minima. This can be formally defined as seen in Algorithm 1.

Algorithm 1: SLS Iterative Improvement algorithm [22]

```

Starting from a feasible solution  $s$ 
while  $s$  is not a local optimum do
    search the neighborhood for a neighbor  $s'$ 
    if new solution  $s'$  is better then
        accept the new solution,  $s := s'$ 
    end
end

```

The performance of iterative improvement depends on how we define the neighborhood relation. Most iterative improvement algorithms for the TSP are based on the k -exchange neighborhood relation, colloquially also known as k -opt. Here, a candidate solution is in the neighborhood if it can be obtained by deleting k

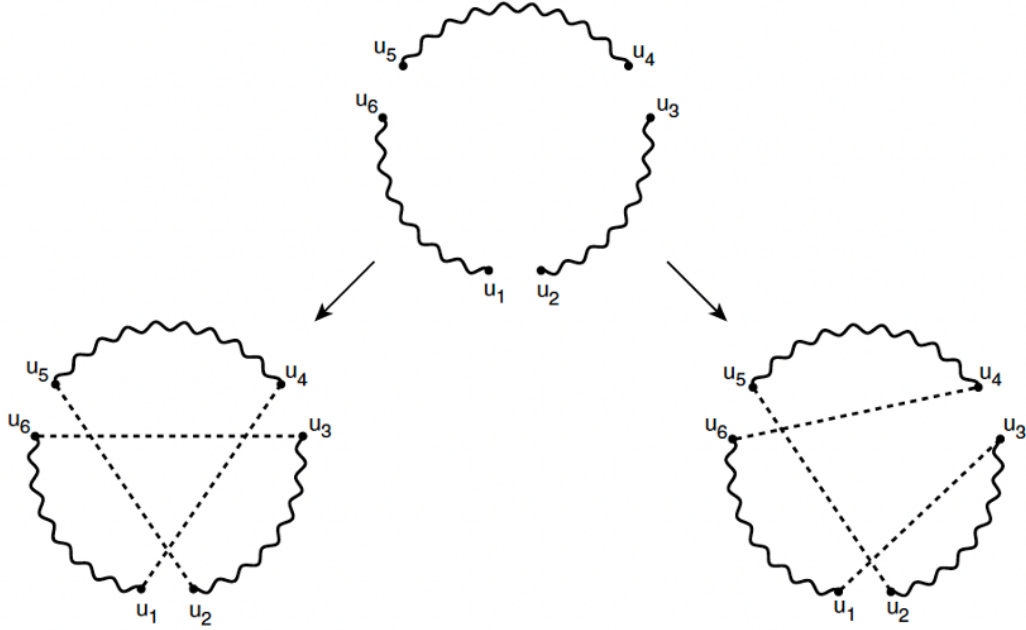


Figure 11: Two possible ways to perform a 3-opt move, where edges (u_1, u_2) , (u_3, u_4) and (u_4, u_5) are removed from the complete tour [22]. In total, there are 7 ways to reconnect the subtours.

edges and then rewiring the resulting fragments into a complete tour by inserting a different set of k edges [22]. That is, a k -opt move consists of removing k edges from a given tour and then reconnecting k segments to possibly get a shorter tour [26]. An example of 3-opt moves can be seen in Figure 11.

4.3 QUBO Local Search

We can solve larger TSP problems with a QUBO solver more accurately by incorporating ideas from the SLS algorithms of the previous section. To do this, we treat the neighborhood relation from iterative improvement as a subproblem, and then model this subproblem as a QUBO. Instead of checking each candidate solution in a given neighborhood, we let the QUBO solver propose a candidate solution by solving the subproblem. This is referred to "Quantum Local Search" (QLS) by Liu et. al. (see Algorithm 2). We see this is essentially an SLS iterative improvement algorithm where a QUBO solver is a subroutine.

Algorithm 2: QLS algorithm [26]

```

Starting from a feasible solution  $s$  of the original problem of size  $N$ 
for  $i = 0, 1, \dots$  do
    Choose a subset of the  $N$  variables
    Formulate the subproblem QUBO and use a QUBO solver to find a new feasible solution  $s'$ 
    if new solution  $s'$  is better then
        | accept the new solution,  $s := s'$ 
    end
end

```

For example of a subproblem to solve in this manner, let's model a k -opt move as a QUBO problem. We refer to the k tour fragments after deleting k edges as subtours. Let's label each subtour by its starting and ending city, where cities $2i$ and $2i + 1$ corresponds to the i^{th} subtour. In total we have $2k$ cities. We can define a new way to connect the subtours together by specifying the order of cities to visit in a length $2k$

Hamiltonian cycle. Thus, the QUBO variable $x_{v,j}$ corresponds to visiting city v as position j in the length $2k$ Hamiltonian cycle. To solve, we minimize $H = H_A + H_B$ where

$$H_A = A \sum_{v=1}^{2k} \left(1 - \sum_{j=1}^{2k} x_{v,j} \right)^2 + A \sum_{j=1}^{2k} \left(1 - \sum_{v=1}^{2k} x_{v,j} \right)^2 \quad (6)$$

and

$$H_B = \sum_{(uv) \in E} W_{uv} \sum_{j=1}^{2k} x_{u,j} x_{v,j+1} \quad (7)$$

$$W_{uv} = \begin{cases} -A & \text{if } u, v = 2i, 2i + 1 \text{ where } 0 < i < k \\ D_{uv} & \text{otherwise} \end{cases} \quad (8)$$

The terms in Equation 6 have the same exact interpretation as those in Equation 4 of the original TSP mapping. The modification occurs in Equation 7, where we now have two possible cases for weights between $x_{u,j}$ and $x_{v,j+1}$. The $-A$ case in ensures that cities of the same subtour always occur one after the other in the Hamiltonian cycle. Otherwise, the weight is just the distance between cities where $W_{uv} = D_{uv}$. Solving this QUBO thus solves the k -opt subproblem in a SLS iterative improvement algorithm. This is already an improvement over the original TSP mapping, as now the number of variables scales quadratically with the neighborhood size $4k^2$, rather than quadratically with the total TSP size N .

This is not the only way to define a QUBO subproblem that solves the TSP. Instead of a k -opt move, Liu et. al. proposes a k -reversal move, as seen in Figure 12, where the decision variable y_i corresponds to reversing the order of the i^{th} subtour [26]. In their paper, they minimize the following QUBO function:

$$q(y_k, y_1) + \sum_{i=1}^{k-1} q(y_i, y_{i+1}) \quad (9)$$

where

$$q(y_i, y_j) = w_{v_i, u_j} \bar{y}_i \bar{y}_j + w_{u_i, u_j} y_i \bar{y}_j + w_{v_i, v_j} \bar{y}_i y_j + w_{u_i, v_j} y_i y_j \quad (10)$$

and $\bar{y}_i = 1 - y_i$

This is actually equivalent to k -opt for $k = 2$ and 3 , but only represents a subset of k -opt moves for larger k . The advantage of this subproblem is that it can be formulated as a purely unconstrained problem, and so does not need to model ordering constraints like seen in Equations 4 and 6. In addition, the number of variables scales linearly with k , as opposed to the quadratic scaling of the previous mapping.

4.4 Benchmark comparison

In this section we examine the performance of the two QUBO Local Search algorithm mappings proposed in the previous section. To focus on the effects of the mappings rather than the QUBO solving algorithm, we use the existing PyQUBO [27, 28] Python library as opposed to our hardware from Section 3. We will also fix our neighborhood search size to $k = 2$. In Figure 13, we compare the performance of the original base TSP QUBO mapping vs QUBO 2-opt as a function of the problem size. Here we plot the optimality gap, which is defined as follows:

$$\text{Optimality Gap} = \frac{\text{Tour Length} - \text{Optimal Tour Length}}{\text{Optimal Tour Length}} \quad (11)$$

As such, the closer the optimality gap is to zero the better. We see in Figure 13 that as the number of cities increases, the TSP QUBO mapping solution quality suffers, while the QUBO Local Search method (represented by QUBO 2-opt) performs well.

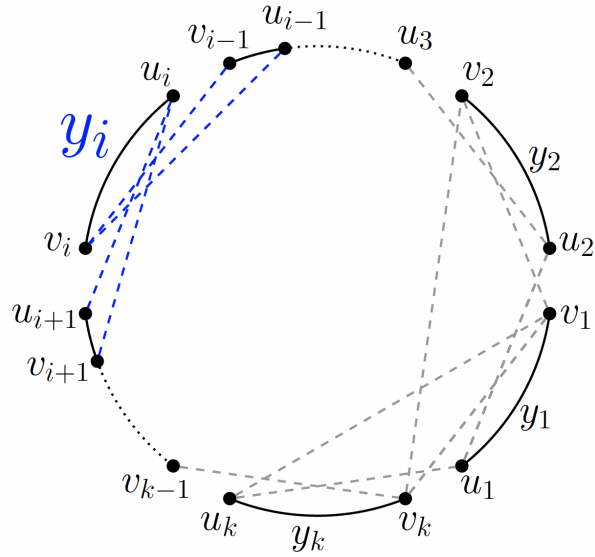


Figure 12: Subproblem for TSP that reverses at most k segments of a tour.

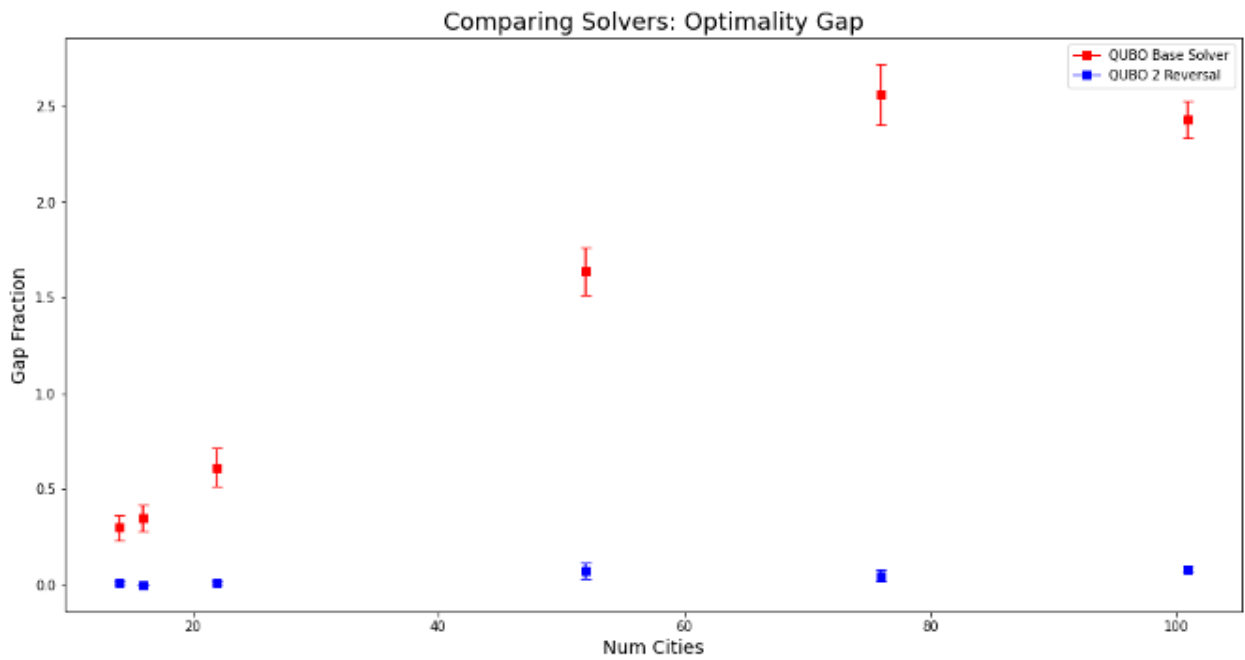


Figure 13: Performance of Base TSP QUBO mapping vs 2-opt QUBO algorithm

Problem	Num. Cities	QUBO Base	QUBO 2-opt	QUBO 2-reversal
burma14	14	0.30	0.01	0.01
berlin52	52	1.64	0.06	0.08
pr76	76	2.56	0.04	0.05
eil101	101	2.43	0.08	0.08

Table 2: Optimality gap showing performance over TSPLIB95 problems.

Problem	Num. Cities	QUBO Base	QUBO 2-opt	QUBO 2-reversal
burma14	14	0.08	1.68	0.44
berlin52	52	5.52	143.16	37.63
pr76	76	19.46	485.575	133.96
eil101	101	49.90	1172.54	294.06

Table 3: Runtime (seconds) of solvers over TSPLIB95 problems.

A closer look at the performance can be found in Table 2, where we compare the original TSP QUBO mapping, QUBO 2-opt, and QUBO 2-reversal. Again, the solution quality of the original base mapping suffers as the problem size increases. The performance of QUBO 2-opt and QUBO 2-reversal are similar. This is expected, as the problems are identical for $k = 2$. However, we see a stark difference in runtime in Table 3. This is due to fact that the subproblem for QUBO 2-reversal is only size $k = 2$, whereas the subproblem for QUBO 2-opt is size $4k^2 = 16$. That is, QUBO 2-opt is solving a larger QUBO problem per iteration. We also note that since we are solving many more QUBO problems overall, the QUBO Local Search algorithms take much longer than solving the base TSP mapping once.

5 Conclusion and Future Work

This report has been a brief look at unconventional computing methods centered around Ising machines. This came in two parts: we took a look at two different hardware accelerators, and we examined a possible application in the form of TSP. There are many veins of research that can follow from this work. For example, on the hardware-side, we may try to continue to scale up our implementations. This can take the form of multi-FPGA accelerators like those done by Ising machines based on Simulated Bifurcation [29]. Or we could take our asynchronous neural network and scale up the number of neurons and/or increase the connectivity of individual neurons. Both approaches could see larger problem sizes that can be mapped onto our hardware accelerators.

On the algorithm side, there are many ideas that can expand on our work on TSP. This work has only showed results of the k -opt algorithm for $k = 2$. We would expect solution quality to increase for larger k . However, keeping fixed neighborhood sizes scales poorly, as the inner iterative improvement loop scales with n^k [22]. This is remedied by algorithms such as the Lin-Kernighan algorithm (whose most popular implementation is via Helsgaun, thus it is often referred to as LKH), which uses a variable neighborhood search size to increase performance [30]. We may explore using an QUBO Local Search approach based off of LKH. We can also break up the TSP problem in different ways. Dan et. al. has used clustering to create smaller problems [25], and Sanyal et. al. has used graph neural networks [31]. Any one of these methods or their combinations can lead to exciting developments for TSP algorithms.

Finally, we marry the two sections of the report and engage in hardware-algorithm co-design of a specialized accelerator for TSP. Such research would culminate in an Ising machine that takes advantage of both our hardware scaling ideas as well as new algorithmic ideas based on local search.

References

- [1] M. W. Johnson, M. H. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose, “Quantum annealing with manufactured spins,” *Nature*, vol. 473, no. 7346, pp. 194–198, 5 2011. [Online]. Available: <https://www.nature.com/nature/journal/v473/n7346/full/nature10012.html>
- [2] T. Inagaki, Y. Haribara, K. Igarashi, T. Sonobe, S. Tamate, T. Honjo, A. Marandi, P. L. McMahon, T. Umeki, K. Enbutsu, O. Tadanaga, H. Takenouchi, K. Aihara, K. I. Kawarabayashi, K. Inoue, S. Utsunomiya, and H. Takesue, “A coherent Ising machine for 2000-node optimization problems,” *Science*, vol. 354, no. 6312, pp. 603–606, 11 2016. [Online]. Available: <https://science.sciencemag.org/content/354/6312/603> <https://science.sciencemag.org/content/354/6312/603.abstract>
- [3] S. Patel, P. Canozza, and S. S. Salahuddin, “Logically synthesized and hardware-accelerated restricted boltzmann machines for combinatorial optimization and integer factorization,” *Nat Electron*, vol. 5, 92–101, 2022.
- [4] A. Datar, “Digital system design and fullchip integration for asynchronous stochastic neural accelerator,” Master’s thesis, EECS Department, University of California, Berkeley, Jun 2021. [Online]. Available: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-161.html>
- [5] S. Lu, “Power, performance, and area analysis of asynchronous stochastic neural accelerator passov1,” Master’s thesis, EECS Department, University of California, Berkeley, May 2022. [Online]. Available: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-94.html>
- [6] F. Barahona, “On the computational complexity of ising spin glass models,” *Journal of Physics A: Mathematical and General*, vol. 15, no. 10, pp. 3241–3253, oct 1982. [Online]. Available: <https://doi.org/10.1088/0305-4470/15/10/028>
- [7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 5 1983. [Online]. Available: <https://www.sciencemag.org/lookup/doi/10.1126/science.220.4598.671> <http://science.sciencemag.org/content/220/4598/671>
- [8] A. Lucas, “Ising formulations of many np problems,” *Frontiers in Physics*, vol. 2, 2014. [Online]. Available: <https://www.frontiersin.org/article/10.3389/fphy.2014.00005>
- [9] S. Patel, L. Chen, P. Canozza, and S. S. Salahuddin, “Ising model optimization problems on a FPGA accelerated restricted boltzmann machine,” *CoRR*, vol. abs/2008.04436, 2020. [Online]. Available: <https://arxiv.org/abs/2008.04436>
- [10] F. Glover, G. Kochenberger, and Y. Du, “A tutorial on formulating and using qubo models,” 2018. [Online]. Available: <https://arxiv.org/abs/1811.11538>
- [11] A. Fischer and C. Igel, “An introduction to restricted boltzmann machines,” in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, L. Alvarez, M. Mejail, L. Gomez, and J. Jacobo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 14–36.
- [12] T. Hayes and A. Sinclair, “A general lower bound for mixing of single-site dynamics on graphs,” in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS’05)*, 2005, pp. 511–520.
- [13] N. Mohseni, P. L. McMahon, and T. Byrnes, “Ising machines as hardware solvers of combinatorial optimization problems,” *Nature Reviews Physics*, vol. 4, no. 6, pp. 363–379, 2022.
- [14] T. B. Preußer and R. G. Spallek, “Ready PCIe data streaming solutions for FPGAs,” in *Conference Digest - 24th International Conference on Field Programmable Logic and Applications, FPL 2014*. Institute of Electrical and Electronics Engineers Inc., 10 2014.

- [15] W. Dally, “High-Performance Hardware for Machine Learning,” in *Nips 2015*, 2015.
- [16] M. N. Bojnordi and E. Ipek, “Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning,” in *IEEE International Symposium on High Performance Computer Architecture*, 3 2016, pp. 1–13. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7446049/>
- [17] C. H. Tsai, Y. T. Chih, W. H. Wong, and C. Y. Lee, “A Hardware-Efficient Sigmoid Function with Adjustable Precision for a Neural Network System,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 11, pp. 1073–1077, 2015.
- [18] A. Z. Pervaiz, B. M. Sutton, L. A. Ghantasala, and K. Y. Camsari, “Weighted p-bits for FPGA implementation of probabilistic circuits,” *arXiv:1712.04166 [cs]*, 12 2017. [Online]. Available: <http://arxiv.org/abs/1712.04166>
- [19] M. T. Tommiska, “Efficient digital implementation of the sigmoid function for reprogrammable logic,” *IEE Proceedings: Computers and Digital Techniques*, vol. 150, no. 6, pp. 403–411, 2003.
- [20] G. Marsaglia, “Xorshift RNGs,” *Journal of Statistical Software*, vol. 8, pp. 1–6, 2003. [Online]. Available: <https://www.jstatsoft.org/article/view/v008i14/xorshift.pdf>
- [21] M. Matsumoto and T. Nishimura, “Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator,” in *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, 1 1998, pp. 3–30. [Online]. Available: <https://dl.acm.org/doi/10.1145/272991.272995>
- [22] H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*, ser. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science, 2005. [Online]. Available: <https://books.google.com/books?id=3HAedXnC49IC>
- [23] S. Feld, C. Roch, T. Gabor, C. Seidel, F. Neukart, I. Galter, W. Mauerer, and C. Linnhoff-Popien, “A hybrid solution method for the capacitated vehicle routing problem using a quantum annealer,” *Frontiers in ICT*, vol. 6, p. 13, 2019.
- [24] G. Reinelt, “Tsplib95,” *Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg*, vol. 338, pp. 1–16, 1995.
- [25] A. Dan, R. Shimizu, T. Nishikawa, S. Bian, and T. Sato, “Clustering approach for solving traveling salesman problems via ising model based solver,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.
- [26] X. Liu, H. Ushijima-Mwesigwa, A. Mandal, S. Upadhyay, I. Safro, and A. Roy, “On modeling local search with special-purpose combinatorial optimization hardware,” *arXiv preprint arXiv:1911.09810*, 2019.
- [27] K. Tanahashi, S. Takayanagi, T. Motohashi, and S. Tanaka, “Application of ising machines and a software development for ising machines,” *Journal of the Physical Society of Japan*, vol. 88, no. 6, p. 061010, 2019.
- [28] M. Zaman, K. Tanahashi, and S. Tanaka, “Pyqubo: Python library for qubo creation,” *IEEE Transactions on Computers*, 2021.
- [29] K. Tatsumura, M. Yamasaki, and H. Goto, “Scaling out ising machines using a multi-chip architecture for simulated bifurcation,” *Nature Electronics*, vol. 4, no. 3, pp. 208–217, 2021.
- [30] K. Helsgaun, “An effective implementation of the lin–kernighan traveling salesman heuristic,” *European journal of operational research*, vol. 126, no. 1, pp. 106–130, 2000.
- [31] S. Sanyal and K. Roy, “Neuro-ising: Accelerating large-scale traveling salesman problems via graph neural network guided localized ising solvers,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5408–5420, 2022.