An Approximate Parallel Annealing Ising Machine for Solving Traveling Salesman Problems

Qichao Tao, Tingting Zhang, *Graduate Student Member, IEEE*, and Jie Han, *Senior Member, IEEE*

***Abstract*—Annealing-based Ising machines have emerged as high-performance solvers for combinatorial optimization problems (COPs). As a typical COP with constraints imposed on the solution, traveling salesman problems (TSPs) are difficult to solve using conventional methods. To address this challenge, we design an approximate parallel annealing Ising machine (APAIM) based on an improved parallel annealing algorithm. In this design, adders are reused in the local field accumulator units (LAUs) with half-precision floating-point representation of the coefficients in the Ising model. The momentum scaling factor is approximated by a linear, incremental function to save hardware. To improve the solution quality, a buffer-based energy calculation unit selects the best solution among the found candidate results in multiple iterations. Finally, approximate adders are applied in the design for improving the speed of accumulation in the LAUs. The design and synthesis of a 64-spin APAIM show the potential of this methodology in efficiently solving complicated constrained COPs.**

***Index Terms*—Ising model, parallel annealing (PA), simulated annealing (SA), traveling salesman problem (TSP).**

I. Introduction

COMBINATORIAL optimization problems (COPs) exist in a wide spectrum of applications, including artificial intelligence, route planning, and scheduling [1]. For example, the optimizations in circuit layout design and routing algorithms are typical COPs in the semiconductor industry. Many COPs are nondeterministic polynomial-hard, so computationally intensive to solve. Recently, Ising model-based computers, or Ising machines, have emerged as efficient solutions for COPs.

The Ising model mathematically describes the ferromagnetism of a set of magnetic spins [2]. Solving a COP via an Ising machine is to find the ground state of the energy [3]. Ising machines function on the principles of physical or circuit oscillators [4], simulated bifurcation [5], and simulated annealing (SA) [2] in a Hamiltonian system.

Like thermal annealing in metallurgy, conventional SA is aimed to converge the energy to a minimum value [2]. However, it cannot simultaneously update the states of connected spins, thus resulting in an increased search time. To mitigate this issue, parallel annealing (PA) leverages a two-layer spin structure for parallel spin update [6], [7]. Recently, a PA-based Ising machine, named STATICA [6], achieved a fast annealing for solving unconstrained max-cut problems. However, it is difficult for this system to escape from local minimum states when solving constrained COPs due to the limited precision in coefficients and fluctuations in energy. As a constrained COP, a traveling salesman problem (TSP) is to find the shortest route that visits every city exactly once and returns to the origin. An improved PA (IPA) exploits an exponential temperature function with a dynamic offset for efficiently solving a TSP [7]. For a higher solution quality, however, it requires a clustering approach to decompose the TSP to smaller problems, thus incurring additional overhead.

This letter presents the design of an approximate PA Ising machine (APAIM) using the IPA to efficiently solve constrained COPs such as the TSP. Inspired by the STATICA machine in [6], a general circuit architecture is developed to implement a 64-spin prototype. Half-precision floating-point (FP) coefficients are used to obtain an extended range of numbers for solving complex COPs.

The contributions lie in the following novelties to achieve a tradeoff between solution quality and hardware efficiency: 1) a new buffer-based energy calculation unit is designed to improve the solution quality in lieu of the clustering approach; 2) a linear function approximately implements the momentum scaling factor in the IPA to reduce the computational complexity; and 3) approximation is applied to less significant bits (LSBs) in the addition to further save hardware.

The remainder of this letter is organized as follows. Section II introduces the IPA algorithm for solving TSPs. The circuit design of the APAIM is discussed in Section III. Section IV reports the experimental results. Section V concludes this letter.

II. Preliminaries

The Ising model mathematically simulates the formation of magnetic domains in ferromagnets. The interactions between the ith and jth spins (J_{ij}) and the external magnetic field (h_i) determine the ith spin state (σ_i). The Hamiltonian of an Ising model is given by $H = -\sum_{i,j} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i$ [1].

An n-city TSP can be formulated as an n^2-spin Ising problem. The Hamiltonian of solving a TSP using the IPA based on a two-layer structure (H_{IPA}) is given by [7]

$$H_{\text{IPA}} = -\sum_{i,k,j,l} J_{ikjl}\sigma_{ik}^{L}\sigma_{jl}^{R} - \frac{1}{2}\sum_{i,k} h_{ik}(\sigma_{ik}^{L} + \sigma_{ik}^{R}) + \omega_{ik}\sum_{i,k}(1 - \sigma_{ik}^{L}\sigma_{ik}^{R}) \quad (1)$$

Manuscript received 13 July 2023; accepted 22 July 2023. This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Project RES0051374, and in part by the University of Alberta under Project RES0049590. The work of Tingting Zhang was supported by the China Scholarship Council (CSC). This manuscript was recommended for publication by P. R. Panda. *(Corresponding author: Tingting Zhang.)*

The authors are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada (e-mail: qichao@ualberta.ca; ttzhang@ualberta.ca; jhan8@ualberta.ca).

Digital Object Identifier 10.1109/LES.2023.3298739

1943-0671 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Algorithm 1 IPA for TSPs [7]

```
1:  Initialize spin configurations, $T$, $\Delta T$
2:  for $s = 1$ to $s_{\max}$ do
3:      if $s$ is odd then
            $A \Leftarrow L, B \Leftarrow R$
4:      else
            $A \Leftarrow R, B \Leftarrow L$
5:      end if
6:      Update $p$ and $c$, $T \Leftarrow (T + \Delta T) \cdot r^{s-1}$
7:      for $i = 1$ to $n$ do
8:          for $k = 1$ to $n$ do
9:              $\omega_{ik} \Leftarrow 0$ with $p$, or $\omega_{ik} \Leftarrow c \cdot \omega_{ik}$ with $1 - p$
10:             $lf_{ik} \Leftarrow (\frac{h_{ik}}{2} + \sum_{j,l} J_{ikjl}\sigma_{jl}^{B})$
11:             $\Delta E_{ik} \Leftarrow (2lf_{ik}\sigma_{ik}^{A} + 2\omega_{ik}\sigma_{ik}^{A}\sigma_{ik}^{B})$
12:             $P_{ik} \Leftarrow \min\{1, \exp(-\Delta E_{ik}/T_s)\}$
13:             if $P_{ik} > rand$ then
14:                 $\sigma_{ik}^{A} \Leftarrow -\sigma_{ik}^{A}$
15:             end if
16:         end for
17:     end for
18:     if no spin is flipped then
            $\Delta T \Leftarrow \Delta T + T_{inc}$
18:     else
            $\Delta T \Leftarrow 0$
19:     end if
20: end for
```

where σ_{ik}^{L} (or σ_{jl}^{R}) is the state of a spin with the index (i, k) [or (j, l)] in a lattice on the left (or right) layer, J_{ikjl} is the coupling coefficient between σ_{ik}^{L} and σ_{jl}^{R}, h_{ik} is the external magnetic field for σ_{ik}^{L} and σ_{ik}^{R}, and ω_{ik} is the self-interaction factor to give the coupling strength between σ_{ik}^{L} and σ_{ik}^{R}. J_{ikjl} and h_{ik} are related to the route distance between cities and parameters to balance the weights between the objective function and constraints.

As shown in Algorithm 1, the state of a spin is randomly initialized to "-1" or "$+1$," the temperature (T) for annealing is initialized to a relatively large value, and the dynamic offset (ΔT) for increasing the temperature is initialized to "0" (line 1). Spins in the left layer are updated when the current iteration (s) is odd; otherwise, the spins in the right layer are updated (lines 3–5). First, in each iteration, T and ω_{ik} are recalculated. ω_{ik} is set to "0" with the probability p or decreased to $c \cdot \omega_{ik}$ with the probability $1 - p$, where p is the dropout rate and c is the momentum scaling factor (line 9). The local field (lf_{ik}) for obtaining the variation (ΔE_{ik}) depends on J_{ikjl} and h_{ik} (line 10). ΔE_{ik} indicates the change in energy when σ_{ik} is flipped (line 11). Subsequently, the spin-flip probability (P_{ik}) is calculated using the Metropolis algorithm [8] (line 12). After each iteration, for introducing some randomness to help the system jump out of the local minima, ΔT will increase by an increment value (T_{inc}) when no spin is flipped; otherwise, ΔT will be reset to "0" (lines 17–19). At the end of a search (i.e., reaching the predefined number of iterations $s_{\max}$), the spin configuration provides the final solution.

III. Design of the APAIM

A. Architecture

Fig. 1 shows the architecture of the APAIM with $N(= n^2)$ spins, inspired by the STATICA design in [6]. It consists of N local field accumulator units (LAUs), N spin update units

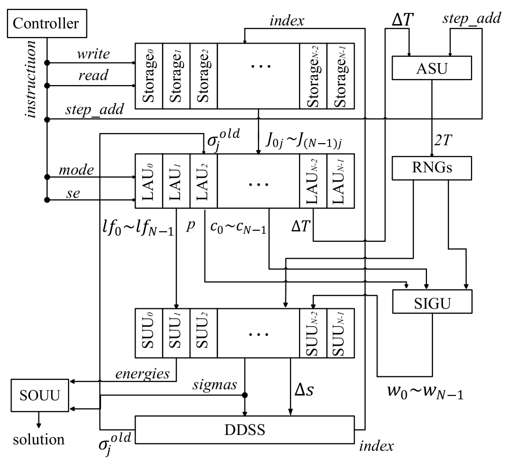

Fig. 1. Architecture of the APAIM (adapted from [6]).

(SUUs), a delta-driven simultaneous SUU (DDSS), a controller, an annealing schedule unit (ASU), N self-interaction generating units (SIGUs), $N/2$ random number generators (RNGs), a solution update unit (SOUU), and a memory block. The 2-D model in (1) is implemented as a 1-D Ising model for efficiency by converting σ_{ik} to $\sigma_{(i-1)\cdot n+k}$.

The local field (lf_i), dropout rate (p), momentum scaling factor times self-interaction ($c \cdot \omega_i$) for computing ΔE_i (line 11 in Algorithm 1), and dynamic offset (ΔT) (line 17) are calculated in the LAUs. The SIGUs temporarily set ω_i to "0" with probability p (line 9). The SUUs use random numbers from the RNGs, lf_i from the LAUs, and ω_i from the SIGUs to compute the new spin states (lines 11–14). It outputs Δ_i to indicate whether the ith spin is flipped. Then, $lf_i \cdot \sigma_i$ values are prepared for the total energy calculation. The states of spins on the left and right layers, denoted by σ_i^{L} and σ_i^{R}, respectively, can be represented by σ_i^{new} and σ_i^{old}, where σ_i^{new} denotes σ_i^{L} (or σ_i^{R}) and σ_i^{old} denotes σ_i^{R} (or σ_i^{L}) for updating σ_i^{L} (or σ_i^{R}). The *sigmas* and Δs are sent to the DDSS to obtain *index* and σ_j^{old}, while the *sigmas* and *energies* are used for the solution update. The *index* is sent to the memory block to select one from $\{2 \cdot J_{0,j}, 2 \cdot J_{1,j}, \ldots, 2 \cdot J_{N-1,j}\}$. These J values and σ_j^{old} are used in the LAUs for new lf_i calculation (line 10). Finally, the controller determines the system operation by coordinating the circuit timing with an *instruction* signal.

B. Local Field Accumulator Unit

An LAU is newly designed for reusing its adder. ΔE_i is computed as $\Delta E_i = 2lf_i\sigma_i^{\text{new}} + 2\omega_i$ for simplicity in hardware (line 11) [6]. The LAU calculates the local field lf_i as $(h_i/2) + \sum_j J_{ij}\sigma_j^{\text{old}}$. This unit is idle when the other units are working. Hence, it can be utilized to accumulate other values in the system, including the dropout rate $p(= 0, 0.5 - [s/2s_{\max}])$, momentum scaling factor times self-interaction $c \cdot \omega_i$, where $c = \sqrt{(s/s_{\max})}$, and dynamic offset ΔT. To save the area of the circuit, c is approximated by a linear function $(s/s_{\max})$.

As shown in Fig. 2(a), the LAU consists of a multiplier, four registers, two multiplexers, and a demultiplexer. Only one LAU requires four registers in a system and others just need two, as p and ΔT are two variables shared among all spins. The demultiplexer and two multiplexers share one select signal (se), while four different signals control four registers. When

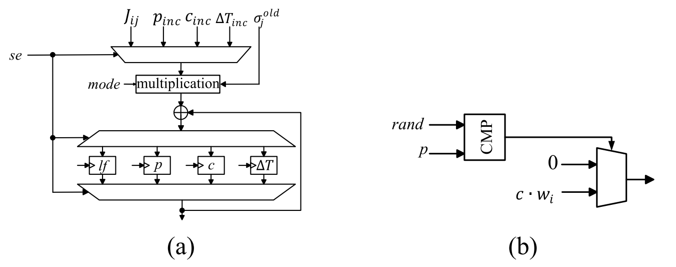
J_{ij} p_{inc} c_{inc} ΔT_{inc} σ_j^{old}
se
mode
multiplication
lf
p
c
ΔT
(a)
rand
p
CMP
0
$c \cdot w_i$
(b)

Fig. 2. (a) LAU and (b) SIGU. In the LAU, the multiplication only performs $\times(+1)$ or $\times(-1)$, where $+1$ and -1 are represented by 1 and 0, respectively. The multiplication with -1 is implemented by an inverter that flips the sign bit of input. Then, a multiplexer determines whether $\times(+1)$ or $\times(-1)$ is performed by taking the value of σ_j^{old} as the selection signal.

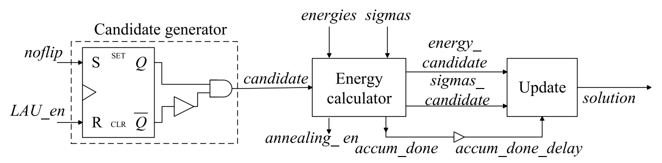

Fig. 3. SOUU.

the LAU works for the local field accumulation, *mode* = 1. se selects $2 \cdot J_{ij}$ and σ_j^{old} is multiplied with $2 \cdot J_{ij}$. Then, the adder accumulates the product to the value in the register of *lf*. Furthermore, the multiplication unit has a *mode* input; the input from the multiplexer is the output without any change when *mode* = 0. The spin states are initialized to "−1." Thus, the initial accumulation result of the local field and external field (with all spin states being "−1") is stored in the register. The product from the multiplier is multiplied by 2 as energy is increased or decreased by $2 \cdot J_{ij} \cdot \sigma_j^{\text{old}}$ when σ_j^{old} is changed. Therefore, $2 \cdot J_{ij}$ is stored in the memory block.

C. Self-Interaction Generating Unit

The newly designed SIGU generates the new self-interaction factor (line 9), as shown in Fig. 2(b). It consists of a comparator and a two-to-one multiplexer. ω_i is multiplied with the value of c. Then, the output of an SIGU is zero with a probability p, or $c \cdot \omega_i$ with the probability $(1 - p)$.

D. Solution Update Unit

An SOUU is newly designed to improve the solution quality. It selects a solution with the lowest total energy of the Ising model among the found results. Calculating the energy at every iteration is inefficient. Moreover, feasible solutions only show up at local minima or the ground state for TSPs. Therefore, the SOUU only calculates the energy when the Ising machine is stuck in a local minimum. It consists of a candidate generator, an energy calculator, and an update unit, as shown in Fig. 3.

In the candidate generator, the *noflip* signal pulses every clock cycle if there is no change in the spins' configuration. Then, an RS flip-flop and an AND gate are used to generate a *candidate* signal that only pulses once. The buffer is for making the *candidate* hold for a delay period.

Fig. 4 shows a structure of the energy calculator. A 64-spin Ising machine requires 64 storage blocks and each of them uses 16 16-bit registers. It stores up to 16 candidates for

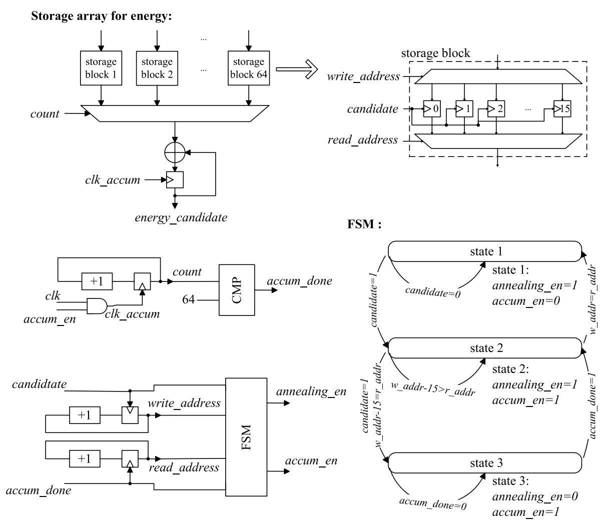

Fig. 4. Energy calculator unit in the SOUU.

accumulation and all registers are controlled by the *candidate* signal. In each storage block, a *read_address* signal determines which candidate is output to the accumulator, and a *write_address* signal determines which new candidate data are written into which registers. The *accum_done* signal becomes 1 when *count* = 64. An *annealing_en* signal changes from 1 to 0 when all data in the 16 registers have not been accumulated but a new *candidate* signal arrives. The Ising machine waits until storage space is released and then continues the annealing process. An *accum_en* signal changes from 1 to 0 when all data in the 16 registers have been accumulated but no new *candidate* signal arrives. The Ising machine waits for the next rising edge of *candidate* to continue the accumulation process.

A finite state machine (FSM) is developed to realize the change of *annealing_en* and *accum_en*. In the initial state 1, the FSM waits for *candidate* = 1, and then moves to state2. In state 2, annealing and accumulation continue working until $w\_addr - 15 = r\_addr$ (*w_addr* represents *write_address* and *r_addr* represents *read_address*) and *candidate* = 1. In state 3, the FSM waits for the accumulation to be finished. When an accumulation is finished (*accum_done* = 1), the FSM moves back to state 2. When all data in the registers have been accumulated (i.e., $w\_addr = r\_addr$), the FSM moves back to state 1. Another similar storage array for energies is used to store *sigmas_candidate*.

In the update unit, the new energy *energy_candidate* is compared with the lowest energy the Ising machine found so far. If the new energy is lower, the lowest energy is updated by *energy_candidate* and the solution is updated by the corresponding sigmas' states (*sigmas_candidate*). Otherwise, the lowest energy and the solution do not change.

IV. Experimental Results and Discussion

A. Approximation of the Momentum Scaling Factor

The momentum scaling factor, $c = \sqrt{(s/s_{\max})}$, is approximately realized by a linear function $c = (s/s_{\max})$ for hardware efficiency. In this way, the adders in LAUs are reused to calculate $c \cdot \omega_i$ in each iteration because the increment for

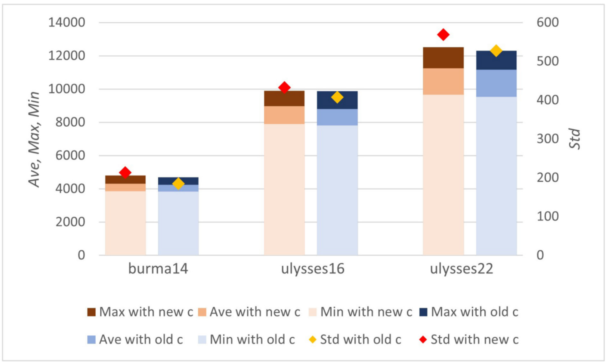

Fig. 5. Comparison of travel distances between applying a nonlinear (old) or linear (new) momentum scaling factor (*burma*14: a 14-city TSP, *ulysses*16: a 16-city TSP, and *ulysses*22: a 22-city TSP [9]).

state update is constant. Fig. 5 shows the effect of applying a linear and nonlinear momentum scaling factor on the solution quality, evaluated by the average (Ave), the maximum (Max), the minimum (Min), and the standard deviation (Std) of the obtained route distances. Using linear approximation for implementing c results in increases of 1.6%, 1.9%, and 0.7% on Ave, 15.2%, 13.3%, and 9.7% on Std, respectively, for the benchmark datasets *burma*14, *ulysses*16, and *ulysses*22.

B. Approximation for Addition

To solve complicated COPs, we consider a more extensive range for the coefficients using a 16-bit FP number representation. However, the FP arithmetic computation is expensive to implement in hardware. Therefore, the approximation technique in the lower-part-OR adder (LOA) [10] and the truncation technique are considered in the mantissa adder to simplify and accelerate the accumulation. The k LSBs in the mantissa adder are approximated by truncating l LSBs and using OR gates to process the remaining $(k-l)$ bits, resulting in a lower-part-OR and truncated adder (LOTA). The circuit area for 64 LAUs without approximation is 78674.4 μm^2. When $l = 0$ and k increases by 1, this circuit area is decreased by approximately 166.5 μm^2. When $l = k$, the area for 64 LAUs diminishes by approximately 675.1 μm^2 with every increment of 1 in the value of l. As a first prototype, a 64-spin APAIM is implemented to solve an 8-city TSP with distances between each pair of cities scaled to $[0, 1]$. We further introduce a metric, the violation rate (VR), to indicate the probability of getting a result that does not conform to constraints. Although not shown, due to space limitation, our experiments indicate that with the increase of k, it is more likely to find an inferior solution. When l increases from 0 to 4 (with $k = l$ here), the Ave and Std increase, but still with an extremely low VR. $l = 3$ is selected to guarantee a high probability that the obtained solution meets the constraints. For $l = 3$, the VR significantly increases when $k > 5$. Thus, $3 < k \leq 5$ and $l = 3$ are further considered in the APAIM. Note that no other solutions of TSPs are provided by a PA machine or available for comparison.

C. Circuit Evaluation

The 64-spin APAIM with 16-bit FP coefficients is implemented with or without approximate adders. Simulation results are obtained by synthesizing circuits using the Synopsys

TABLE I
CIRCUIT MEASUREMENTS OF THE APAIM

Ising Machines		Spin	Precision	Area (mm^2)	Power (mW)	Delay (ns)
APAIM	$k, l = 0$	64	16-bit	6.300	41.289	3.97
	$k = 4,\ l = 3$	64	16-bit	6.269	41.130	3.84
	$k = 5,\ l = 3$	64	16-bit	6.262	41.108	3.82
STATICA [6]		512	5-bit	12	629	-

-: not reported. The STATICA was designed to solve the max-cut problem and synthesized using a 65-nm technology, but not for constrained COPs such as the TSP.

Design Compiler. A CMOS 28-nm technology is applied with a supply voltage of 1.0 V, a temperature of 25°C, and a clock frequency of 200 MHz. The circuit measurements are presented in Table I (for STATICA too). The use of approximate adders, i.e., the LOTAs, results in a reduction in area, power, and delay. It indicates that the proposed APAIM performs computation with a high precision and maintains hardware efficiency. Thus, it has the potential to achieve a tradeoff between solution quality and hardware efficiency in solving complicated constrained COPs.

V. CONCLUSION

In this letter, a new Ising machine, named an APAIM, is designed for solving TSPs as a typical class of COPs. Various computation units are designed and approximately implemented to reduce circuit complexity with only a marginal reduction in solution quality. A prototype of the APAIM is developed and synthesized for a system of 64 spins with 16-bit FP coefficients. As a first PA machine able to solve the TSP, the APAIM achieves a relatively high precision in the model coefficients and is potentially scalable for solving more complex problems. A multichip architecture might be useful on this regard and will be investigated in future work.

REFERENCES

[1] A. Lucas, “Ising formulations of many NP problems,” *Front. Phys.*, vol. 2, p. 5, Feb. 2014.

[2] R. A. Rutenbar, “Simulated annealing algorithms: An overview,” *IEEE Circuits Syst. Mag.*, vol. 5, no. 1, pp. 19–26, Jan. 1989.

[3] N. Mohseni, P. L. McMahon, and T. Byrnes, “Ising machines as hardware solvers of combinatorial optimization problems,” *Nat. Rev. Phys.*, vol. 4, no. 6, pp. 363–379, 2022.

[4] T. Wang and J. Roychowdhury, “OIM: Oscillator-based Ising machines for solving combinatorial optimisation problems,” in *Proc. UCNC*, 2019, pp. 232–256.

[5] K. Tatsumura, A. R. Dixon, and H. Goto, “FPGA-based simulated bifurcation machine,” in *Proc. FPL*, 2019, pp. 59–66.

[6] K. Yamamoto et al., “STATICA: A 512-spin 0.25M-weight annealing processor with an all-spin-updates-at-once architecture for combinatorial optimization with complete spin–spin interactions,” *IEEE J. Solid-State Circuits*, vol. 56, no. 1, pp. 165–178, Jan. 2021.

[7] Q. Tao and J. Han, “Solving traveling salesman problems via a parallel fully connected Ising machine,” in *Proc. DAC*, 2022, pp. 1123–1128.

[8] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953.

[9] G. Reinelt, 1997. [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[10] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, “Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010.