博 士 学 位 論 文
Doctoral Thesis

論文題目

Thesis Title

スピン状態の差分を用いた

大規模完全グラフ向け

高性能アニーリングプロセッサに関する研究

東 北 大 学 大 学 院 工 学 研 究 科

Graduate School of Engineering,

TOHOKU UNIVERSITY

専攻/Department: 通信工学専攻

学籍番号/ ID No: C1TD2303

氏 名 /Ｎａｍｅ： シントッキュ

指 導 教 員	羽生 貴弘 教授
研究指導教員	鬼沢 直哉 准教授
審 査 委 員 （○印は主査）	○ 羽生 貴弘 教授 1 佐藤 茂雄 教授　2 張山 昌論 教授 3 鬼沢 直哉 准教授　4 教授 5 教授　6 教授

提 出 者 略 歴	
ふりがな しんとっきゅ 氏 名 シントッキュ	昭和 (平成) 5 年 7 月 3 日生
本籍 都・道 府・県	国籍 韓国
履 歴 事 項	
【学歴】	
平成 25 年 4 月 1 日	東北大学工学部 入学
令和 1 年 3 月 27 日	同 卒業
令和 1 年 4 月 1 日	東北大学大学院工学研究科博士課程前期 入学
令和 3 年 3 月 25 日	同 修了
令和 3 年 4 月 1 日	東北大学大学院工学研究科博士課程後期 入学
令和 6 年 3 月 25 日	同 修了
【職歴】	
年 月 日	
年 月 日	
年 月 日	

備考(1)外国人留学生は，国籍を記入すること。

(2)履歴事項は，大学入学から年次にしたがって記入すること。

Study on High-Performance Annealing Processor for Large Fully-Connected Ising Model Based on Differential Stochastic Simulated Annealing

ABSTRACT : The combinatorial optimization problem is expected to find applications in various real-world scenarios. For instance, it is applied in the design of very-large-scale-integrated (VLSI) circuits, robot path finding, decision problems in high-frequency trading (HFT) for stocks, and object recognition in autonomous driving systems. Such COPs belong to the class of NP-hard problems, where finding their optimal solutions in linear time can be extremely challenging. To efficiently explore solutions for these NP-hard combinatorial optimization problems, probabilistic algorithms are commonly employed. Simulated Annealing (SA) is a well-known algorithm capable of efficiently optimizing NP-hard COPs. SA originates from the annealing process used in metallurgy, where materials are heated and gradually cooled to grow crystals. The gradual cooling process helps reduce defects while allowing crystals to grow. SA randomly explores solutions in a search space to find the optimal solution satisfying the global minimum of a cost function. The newly selected solution is chosen to meet a lower cost than the previous solution. However, as the algorithm finds a solution satisfying a local optimum, it is difficult to escape from the local optimum. Therefore, SA introduces the acceptance of changes in the direction of increasing costs probabilistically. The stochastic acceptance of worsened solutions is controlled by a hyperparameter known as temperature (T). Similar to the annealing process, the temperature T gradually decreases from high temperatures to low temperatures. At high temperatures, the algorithm accepts worsened solutions with a high probability, allowing for more active exploration in the search space. Subsequently, by gradually cooling the temperature, the algorithm reduces the probability of accepting worsened solutions, ultimately converging to the optimal solution satisfying the global minimum. However, as the size of the COP grows, the time required for the simulated annealing algorithm to converge to an optimal or near-optimal solution also increases exponentially.

To address the issue of increasing computational time, new annealing algorithm based on new computational approaches have been proposed, such as Quantum Annealing (QA), simulated annealing using magnetic tunnel junction (MTJ) devices known as pSA, and Stochastic Simulated Annealing (SSA) based on stochastic computing. These algorithms are applied to Ising models converted from COPs. The Ising model represents a network model comprising binary states ('-1' or '+1') known as Ising spins and their interactions. In the Ising model, the energy is referred to as the Hamiltonian, and when all spins are in their optimal states, the Hamiltonian exhibits a global minimum. For the Ising model, the SA algorithm converges the spin states to achieve the global minimum of the Hamiltonian, exploring the optimal solution for the COP. QA uses qubits, a quantum device, and pSA uses MTJ-based p-bits to implement Ising spins. However, qubits require extremely low temperatures for operation, consuming significant electrical power for cooling. Moreover, p-bits are MTJ devices, large-scale implementation of the Ising model is still challenging. On the other hand, SSA based on stochastic computing provides a faster and more accurate solution to COPs compared to traditional SA. Unlike QA or pSA, which requires dedicated devices, SSA is a hardware-friendly algorithm that can be designed using conventional CMOS technology. SSA has been evaluated using various COPs, such as traveling sales man problem (TSP), graph isomorphism problem (GI), and maximum cut problem (MAXCUT), consistently achieving near-optimal solutions. For instance, SSA successfully solved the 2K-spin COP, achieving the best-known solution for a K2000 MAXCUT problem.

In this paper, we proposed a differential stochastic simulated annealing (DSSA) algorithm for realizing an annealing processor for large and fully-connected Ising models. When implementing an annealing processor for the fully-connected Ising model, the spin-to-spin connections requires enormous hardware area. In general, serialization is a commonly employed technique to reduce hardware area. DSSA effectively reduces the hardware area, resulting from the fully-connected topology, by serializing spin connections. While serialization generally poses a trade-off between hardware area and computation time. For instance, in a fully connected Ising model with 2,000 spins, where each spin is connected to 1,999 other spins, serially calculating

these connections results in a 1,999 times increase in computation time. The proposed DSSA method addresses this challenge by using spin state differentials when serially calculating spin connections, mitigating the increase in computation time. DSSA calculates the spin state only using spin state differentials, thus, the connections with upflipped spins can be ignored. Furthermore, DSSA, like SSA, relies on stochastic computing, making it easily implementable with CMOS circuits. Thus, implementation using FPGA or ASIC is straightforward, and additional speedup through hardware implementation is anticipated. The annealing performance of the proposed DSSA method is compared with serialized conventional SSA. Several 2,000-spin maximum cut problems, including fully-connected maximum cut problem, K2000, are solved using DSSA. DSSA achieves near optimal solutions for G22, G23, G24, G27, G35, G39, which are the benchmark problems from G-set, and K2000, and it accelerates annealing times approximately 2.8-42.1 times faster than that of serialized SSA. Furthermore, DSSA achieves the best known solution for K2000 and G27 with 15.4 times and 42.1 times faster annealing time, respectively.

The high-performance annealing processor for the large and fully-connected Ising model is designed based on the proposed DSSA method using TSMC 28 nm process. The power consumption of the synthesized DSSA processor is 316 mW with the clock frequency of 500 MHz. For evaluation, the DSSA processor is compared with the state-of-the-art ASIC based annealing processors. The proposed DSSA processor contains fully-connected 2,048 Ising spins in a single chip. Each spin in the proposed DSSA processor consumes 0.15 mW, resulting in a 6.2 times improvement in power efficiency than that of the state-of-the-art annealing processor. Furthermore, the DSSA processor achieves the near optimal solution of the K2000 problem with an annealing time of 2.7 ms with an energy consumption of 0.86 mJ. This resulted in approximately 3.5 times higher energy efficiency compared to the energy consumption of conventional annealing processors for the same problem.

The major contributions of this paper are: 1) We introduce DSSA algorithm for cost-effective hardware implementation. Based on the proposed DSSA, 2) the fully-connected 2,048-spin 316 mW 500 MHz DSSA processor is synthesized using TSMC 28 nm process. The performance of the DSSA processor is assessed across various COPs, 3) achieving near-optimal solutions for K2000 with 3.5 times greater energy efficiency compared to state-of-the-art approaches.

目次

第 1 章

緒言

組合せ最適化問題 [1] は実世界の様々なアプリケーションへの応用が期待されている．例えば，very-large-scale-integrated (VLSI) 回路の設計 [2]，ロボットの経路探索 [3, 4]，株の高速取引の決定問題 [5–7]，自動運転での物体認識 [8] などで応用されている．このような組合せ最適化問題は NP 困難問題に属しており，線形時間におけるその解の探索が非常に困難な場合がある [9]．そのような NP 困難である組合せ最適化問題の解探索に効率的なアルゴリズムとして確率的アルゴリズムが用いられる．Simulated annealing (SA) は NP 困難な組合せ最適化問題を効率的に最適化できるアルゴリズムとして知られている [10]．SA は金属材料の焼きなまし法から由来したアルゴリズムであり，焼きなまし法は材料を加熱し，徐々に冷却しながら材料の結晶を成長させる方法である．徐々に材料を冷却することで欠陥を減らしながら結晶を成長させる．SA はある探索空間に対して，コスト関数の全域的極小値を満たす最適解をランダムに探索するアルゴリズムである．新しく選択される解は以前の解より低いコストを満たす解を選択することになる．このとうなアルゴリズムは局所的最適値を満たす解を見つかった時からは解の変化ができなくなってしまう．従って，SA ではコストを高める方向の解の変化も確率的に受け入れることで局所解から脱出する．改悪な解の確率的受け入れはハイパーパラメータである温度 (T) によって調整される．焼きなまし法の同様に，温度 T は高い温度から徐々に低い温度に変化していく．高い温度の際は高い確率で改悪の解を受け入れることでより活発に探索空間での解探索を行う．その後，温度を徐々に冷却することで，改悪の解を受け入れる確率を減らし，最終的には全域的極小値を満たす最適解に収束する．

一方，SA 法は問題規模の増加に応じてその計算時間が指数関数的に増加する．計算時間増加の問題を解決するため，量子アニーリング [11, 12] や magnetic tunnel junction (MTJ) デバイス [13] を用いる pSA [14]，ストカスティック演算 [15] に基づいたストカスティックシミュレーテッドアニーリング [16] などの新しい計算手法が提案されている．このような新しいアルゴリズムは組合せ問題から変換されるイジングモデルに対して適用される [17]．イジングモデルは ‘-1’ または ‘+1’ のバイナリ状態を持つイジングスピンと，それらの相互作用からなるネットワークモデルである [18]．イジングモデルのエネルギーはハミルトニアンといい，全てのスピンが最適な状態である場合，ハミルトニアンは全域的極小値を持つ．すなわち，イジングモデルにおける SA 法はスピン状態を変化させながらハミルトニアンを全域的極小値に収束することで組合せ問題の最適解を探索する．量子アニーリングでは qubit という量子デバイスを，pSA では MTJ ベースの p-bit を用いてイジングスピンを実装する．しかし，qubit を動作させる

ためには極低温が必要であり，冷却に多くの電力を消費する．また，p-bit は MTJ デバイスなので大規模なイジングモデルの実装がまだ困難である．一方，ストカスティック演算に基づいたストカスティックシミュレーテッドアニーリング (SSA) は従来の SA 法より高速かつ高精度で組合せ最適化問題を解決できる [16]．また，専用デバイスを必要とする量子アニーリングや MTJ デバイスを必要とする pSA とは異なり，従来の CMOS を用いて設計が可能であるハードウェアフレンドリーなアルゴリズムである．SSA 法でのイジングスピンの動作は p-bit の動作をストカスティック演算より近似することで実現する．ストカスティック演算は情報をストカスティックビットストリームでの ‘1’ の出現確立で表す．ストカスティック演算を用いることで，ハードウェア実装に多くのリソースを必要とする $tanh$ ような複雑な関数を小面積で実装できる [15,19]．このようなストカスティック演算より MTJ デバイスを必要とする p-bit の動作を近似することで従来の CMOS 回路よりイジングスピンを実現できる．イジングモデルにおける SA 法では一つのスピンのみの状態を反転させながら最適解を探索する．一方，SSA 法でのイジングスピンの状態変化はボルツマンマシン [20] をその基板としており，同時に複数のスピンの状態を変化させる．従って，SSA 法は従来の SA 法より高速に与えられた組合せ最適化問題の最適解を探索できる．SSA の有効性は traveling salesman problem (TSP) [21]，graph isomorphism (GI) [22]，maximum cut problem (MAX-CUT) [23] などの組合せ問題に対して示されている [16]．SSA の MAX-CUT 問題のベンチマークデータセットである G-set [24] に含まれている 800 イジングスピンからなる問題である G11 に対しての計算時間は従来の SA 法の計算時間より約 45 倍高速である．また，504-qubit を搭載している D-Wave Two machine より 16 倍大規模な GI 問題を解決できる．このように，SSA は従来の SA 法における問題サイズによる計算時間の指数関数的な増加問題を解決できる新しいアニーリング手法として期待されている．

もう一つの SA 法の高速化の手法として，FPGA や application-specific integrated circuit (ASIC) を用いたハードウェア化が研究されている．例えば，2021 年 ISSCC では複数の ASIC チップを用いた 144k イジングスピン搭載のアニーリングプロセッサ [25] 及び，512 スピン搭載の [26] アニーリングプロセッサなどが発表されている．しかし，[25] で示されたプロセッサは多くのイジングスピンを搭載しているが，そのスピン間の相互作用が隣接する 8 個のスピンのみと接続されるスパースな構造である．また，[26] で示された STATICA プロセッサは完全グラフ構造のイジングモデルに対応可能であるが，搭載のスピン数が 512 個であり，対応可能な問題サイズに限界がある．前述した SSA 法はストカスティック演算に基づき，従来の CMOS 回路で実装が可能であり，field-programmable gate array (FPGA) を用いた 800 個のイジングスピンを搭載の SSA ハードウェア実装が示されている [27]．800 スピンの SSA ハードウェアが Xilinx 社の Kintex-7 を搭載している Digilent 社の Genesys 2 FPGA ボードを用いて実装されり，動作周波数は 100 MHz である．800 スピンからなる MAX-CUT 問題 G11 に対して，SSA ハードウェアは従来の SA 法より 114 倍高速で近似解を探索できる．一方，FPGA を用いた SSA ハードウェアのスピン隣接構造はスパースな構造である king’s graph [28] である，一つのスピンは隣接の 8 個のスピンのみと接続されている．また，搭載スピン数も 800 個であるため，実世界で応用されるような大規模な組合せ問題にはまだ対応が困難である．このように問題サイズによる SA 法の計算時間増加を抑え SA 法の実世界への応用を実現するためには: 1) 新しい計算方式に基づいたアルゴリズムの高速化，2) 大規模かつ完全グラフに対応可能な高性能のハードウェア実装が大きいな課題である．

そこで本論文では SA 法を実世界へ応用するための高速化及び大規模化という課題を解決するために，

スピン状態の差分を用いてアルゴリズムの高速化を実現する differential stochastic simulated annealing 法 (DSSA) を提案する．大規模のハードウェア実装の際，計算のシリアル化による小面積化は一般的な手法である．多数のイジングスピンの搭載かつ全てのスピンが相互作用を持つ大規模なイジングモデルに対応可能な SSA ハードウェアを実現するためにはシリアル化が必須不可欠である．SSA ハードウェアにおいてはスピン間の接続をシリアル化することで，単一スピンを小面積で実装するとが可能である．しかし，スピン接続をシリアルに計算することによって，その計算時間は爆発的に増加してしまう．例えば，完全接続している 2,000 スピンのイジングモデルで，一つのスピンは 1,999 個のスピンと接続しており，これらの接続を単純にシリアルで計算すると，その計算時間は 1,999 倍増加してしまう．このようなスピン接続のシリアル実装による計算時間の増加は，シリアル化による小面積化の大きな課題である．提案の DSSA 法ではシリアルにスピン接続を計算する際，スピン状態の差分を用いて計算されるスピンを選択することで，スピン接続計算のシリアル化による計算時間の増加を抑える．このように DSSA 法という新しいアニーリングアルゴリズムを通して，実世界への SA 法の応用するために必要な，対応可能なイジングモデルの大規模化を目指す．また，提案の DSSA 法は SSA 法と同様に，ストカスティック演算に基づいているため，CMOS 回路による実装が容易である．従って，FPGA や ASIC を用いた実装がとても容易であり，ハードウェア実装によるさらなる高速化も期待でいる．

本論文では提案の DSSA 法の性能を従来の SSA 法と, シミュレーションより比較し，その有効性を検証する．また，DSSA 法に基づいた，互いに完全接続している 2,048 スピンを搭載した DSSA ハードウェアを設計し，その動作と性能を関連研究 [25, 26, 29] 及び graphics processing unit (GPU) に実装された SSA 法と比較する．2,000 スピンからなる完全グラフの MAX-CUT 問題である K2000 において [30]，提案 DSSA 法は単純にシリアル化された従来の SSA 法より約 15.4 倍高速で近似解を探索できる．また，TSMC 28nm プロセスを用いて設計された 2,048 スピンの DSSA プロセッサの論理合成時の消費電力は 316 mW であり，その際の動作周波数は 500 MHz である．K2000 問題において設計された DSSA プロセッサは近年発表されたアニーリングプロセッサより 3.5 倍低電力で同等な近似解を探索できる．

本論文の成果は: 1) 大規模かつ完全グラフのイジングモデルに対応可能な高性能なアニーリングプロセッサを実現するために，スピン状態の差分を用いて高速化を行う DSSA アルゴリズム提案である．また，提案の DSSA 法に基づいて，2) 完全グラフの 2,048 イジングスピンを搭載した DSSA プロセッサを設計し，ハードウェア化によるさらなう高速化かつ低電力化を示す．設計した DSSA ハードウェアは，3) 近年発表されたアニーリングプロセッサより 3.5 倍のエネルギー効率である，その有効性を示す．

本論文の構成は以下のようである．第 1 章は序論であり，本論文の研究背景及び目的について述べた．第 2 章は組合せ最適化問題よ従来の SA 法及び SSA 法の基礎について考察する．組合せ最適化問題のイジングモデルによる表現とイジングモデルにおける SA 法及び SSA 法の適用について概説し，従来手法の課題について述べる．第 3 章では SSA 法に基づいた SSA ハードウェアの FPGA 実装及びその性能評価について述べる．FPGA を用いて実装された 800 スピンのスパースな SSA ハードウェアの設計と MAX-CUT 問題を用いた SSA ハードウェアの検証及び関連研究との性能比較について述べる．第 4 章ではスピン状態の差分を用いた提案アルゴリズムである DSSA について述べる．イジングスピンの相互作用のシリアル化による小面積化とそれによる計算時間増加の問題について紹介し，スピン差分を用いた高速化についてのべる．加えて，シミュレーションによる DSSA 法の計算時間などの性能を検証及び

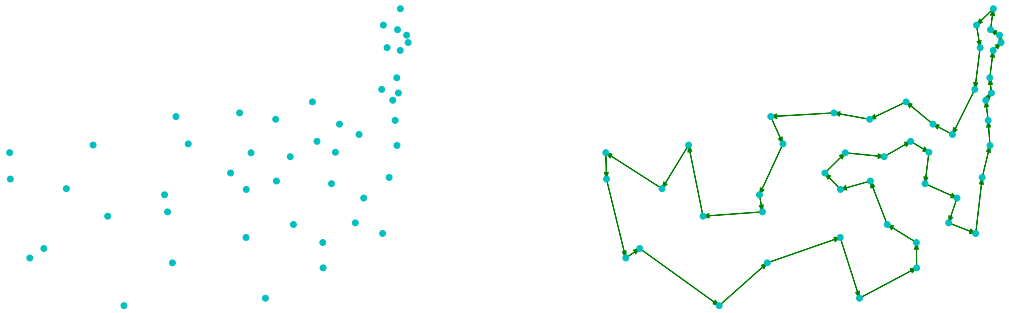
従来の SSA 法との比較を行う．第 5 章では提案 DSSA 法に基づいた大規模かつ完全グラフ向け 2,048 スピン DSSA プロセッサについて述べる．提案プロセッサのアーキテクチャ及びその動作について説明し，配置配線されたレイアウト設計を示す．また，配置配線後のプロセッサの性能を近年発表された AISC によるアニーリングプロセッサと比較する．最後に第 6 章は結論であり，本論文の成果をまとめ，DSSA 法の応用など今後の展望を述べる．以上，本論文の企図するとこを概説した．

第 2 章

アニーリング法とその原理

2.1 まえかき

本章では本論文の背景になるストカスティックシミュレーテッドアニーリングについて概説する．初めに組合せ最適化問題からイジングモデルにおける従来のシミュレーテッドアニーリング (SA) アルゴリズムにいて述べ，SA の計算時間の問題について考察する．その後，SA 法の計算時間問題を解決するために提案されたストカスティックシミュレーテッドアニーリング (SSA) アルゴリズムについて概説する．SSA 法でのストカスティック演算を用いたイジングスピンの近似やアルゴリズムの動作などを示し，従来の SA 法との計算速度やアニーリング精度などの性能比較を紹介する．



(a)　　(b)

図 2.1: Traveling salesman problem (TSP) の例．(a) TSP のベンチマークデータセットである TSPLIB のインスタンス att48 問題の都市の配置．(b) att48 問題の最適な回路 (最適解).

2.2　シミュレーテッドアニーリングの原理

組合せ最適化問題 (COP) はある解空間で定義されるコスト関数が与えられたとき，そのコスト関数を最小化とする解空間内の変数の組合せを探索する最適化問題である [1]．実世界での組合せ最適化問題の応用は very-large-scale-integrated (VLSI) 回路の設計 [2]，ロボットの経路探索問題 [3, 4]，ドローンを用いた配送の動的経路探索 [31]，分散組込みシステムの動的タスク割り当て問題 [32]，株の高速取引決定問題 [5] などの様々な領域で行われている．組合せ最適化問題の数学的な定義は以下のように表現できる．ある探索空間 F に対して，コスト関数 c は

$$c : F \longrightarrow R^1, \tag{2.1}$$

のように定義される．ここで，

$$c(f) \leq c(y) \text{ for all } y \in F \tag{2.2}$$

を満足する組合せ $f \in F$ を探索する問題が組合せ最適化問題である．代表的な COP としては traveling salesman problem (TSP) がある [21]．TSP は都市などのある地点とそれらの間の距離 (移動コスト) が与えられた時，全ての地点を 1 回ずつ巡り，出発地に戻る最短経路を探索する問題である．Fig. 2.1a. は TSP のベンチマークの一つである TSPLIB [33] のインスタンスである att48 の各都市の配置を示す．att48 問題は 48 都市とそれらの座標からなる TSP 問題であり，ある都市からには他の全ての都市まで移動が可能である．また，Fig. 2.1b は att48 問題の全ての地点を 1 回ずつ巡る最短経路を示す．TSP のコスト関数，c，は移動する距離の総和であり，コスト関数の定義域，F，は各都市を訪問する順番である．この TSP は NP 困難問題に属する問題であり，その最適解の厳密に求めることは非常に難しい．例えば，ある地点から最も距離が違い地点だけを選ぶことだけでは局所解に収束し，最適解を探索するころはできない．ここで，COP の最適解を効率的に探索可能な確率的アルゴリズムがシミュレーテッドア

ニーリング (SA) である [10]．SA 法では，現在の解をランダムに変化させながらコストを削減することで最適解を探索する．この際，コストが削減される解の変化だけではなく，コストが増加する改悪の変化も受け入れることで局所解から脱出できる．SA 法で更新された解の受け入れは温度パラメータ，T によって制御される．現在の解を x，更新された解を x' とする．ここで，それぞれの解のコスト関数の値の差は

$$\Delta c = c(x') - c(x), \tag{2.3}$$

である．新しい解 x' の受け入れの確率は以下のように定義される．

$$p(x') = \exp(-\frac{\Delta c}{T}) = \exp(-\frac{c(x') - c(x)}{T}). \tag{2.4}$$

Eq. (2.4) からわかるように Δc が 0 より小さければ (改善の更新であれば)，更新の受け入れ確率は 1 以上になり，必ず解が更新される．一方，Δc が 0 より大きい場合 (改悪の更新場合)，式 Eq. (2.4) から求められる確率と乱数を比較して，確率が高い時のみ更新された解を受け入れる．このように SA 法では改悪の解更新を確率的に受け入れることで局所的な解から脱出でき，最適解への探索を続けることができる．

Eq. (2.4) で定義された更新の受け入れ確率は温度パラメータ，T を含めている．SA 法はこの温度パラメータを調節することで，探索を行う．温度，T が高い場合 (高温状態)，Eq. (2.4) は 1 に近い値を持つため，高い確率で改悪の更新を受け入れる．一方，T が小さい場合 (低温状態) には Eq. (2.4) の値は小さくなり，改悪の更新を受け入れにくくなる．SA 法を用いて COP を解く際，初期温度は高い値になっている．従って，アニーリング処理の初期には改悪の更新を多く受け入れながら，活発に広い範囲の探索空間を探索する．アニーリング処理が進めることに連れて温度 T は徐々に低減していく．従って，改善の更新を多く受け入れ，最適解へ収束することができる． Fig. 2.2 は att48 問題に対して，アニーリングステップとコスト関数の値を表すグラフである．図からわかるようにアニーリング処理を反復するとこで，コストである移動距離が収束していくことが確認できる．この際の温度パラメータ，T は以下の式より更新される．

$$T(t+1) = T(t) \cdot \alpha, \tag{2.5}$$

ここで，t はアニーリングステップ，α は温度パラメータの変化率である．アニーリング処理中の Eq. (2.5) による温度パラメータの変化は Fig. 2.3 のようであり，線形的に変化する．SA 法は Fig. 2.3 のように線形関数の変化だけではなく，問題に応じて適切に温度パラメータの変化関数を選択する必要がある．

例として Fig. 2.1b で示した TSPLIB の att48 の最良解は 10,628 である．SA 法は確率的なアルゴリズムであるため，処理後に得られた解が必ずしも最適解であることは保障できないため，att48 の最良解が知られている．よって，Fig. 2.2 で示された att48 のアニーリング結果は最良解へ到達していないため，さらなるアニーリングによる解探索が必要である．ここで，問題のサイズが増加すると，探索空間はより拡大するため，その解探索に必要な計算時間はより増加していまう．SA 法の計算時間は問題のサイズの増加に連れて指数関数的に増加していまう [34]．従って，実世界の問題に応用されるような大規模な COP を解決するためにはより高速な新たなアルゴリズムが必要である．

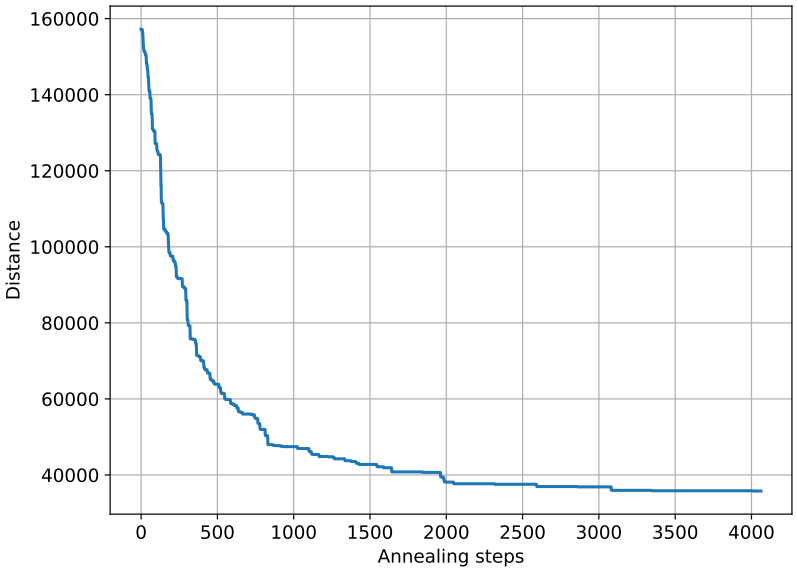


図 2.2: アニーリングステップと TSP 問題のコスト関数．TSPLIB の att48 問題に対して，アニーリングステップによる移動距離 (コスト) のグラフ．アニーリング処理を反復することでコストは最適解へ収束する．

2.3　イジングモデルにおけるシミュレーテッドアニーリング

第 2.2 章で述べた大規模 COP に対する SA 法の計算時間増加の課題を解決するために，従来のコンピューティング方式ではない，新しいコンピューティング方式に基づいたアニーリングアルゴリズムが提案されている．D-Wave machine [12] で代表される量子アニーリング (QA) [11] は量子コンピュータの基本単位である qubit [35] を用いてアニーリング処理を行う．また，qubit の動作を模倣した p-bit [13] を用いてアニーリング処理を行う pSA [14] などが近年の研究より発表されている．これらのアニーリングアルゴリズムは COP から変換されたイジングモデル [18, 36] を用いて与えらた COP の最適解を探索する．QA ではそれぞれのイジングスピンを qubit を用いて実装しており，pSA では p-bit を用いて実装する．

イジングモデルは ‘-1’ と ‘+1’ のバイナリな状態を持つイジングスピンと，各イジングスピンのバイアス，重みを持つイジングスピン間の相互作用より構成されつネットワークモデルである．Fig. 2.4 は三つのイジングスピンからなるイジングモデルを表す．ここで，$\sigma_i \in \{-1. + 1\}$ は i 番目のスピンの状態，$h_i \in R$ は i 番目のスピンのバイアス，$J_{ij} \in R$ は i 番目と j 番目のスピン間の相互作用の重みである．h_i 及び J_{ij} は COP からイジングモデルに変換される際，その値が決まる．Fig. 2.5 のようなイジングモデルはそのモデルのエネルギー，ハミルトニアンを定義するできる．ハミルトニアン，H，は以下のよう

に定義される．

$$H = -\sum_{i}^{n} \sigma_i h_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} \sigma_i \sigma_j J_{ij}, \tag{2.6}$$

ここで，n はイジングモデルに含まれているイジングスピンの数を表す．イジングモデルに変換される COP の最適解は Eq. (2.6) で定義されるイジングエネルギーを全域的極小値にするイジングスピン状態の組合せで表現される．つまり，COP からイジングモデルに変換される際，イジングモデルのパラメータ，h_i と J_{ij} は，全てのイジングスピンが最適な場合にイジングエネルギーが全域的極小値になるようにその値が決まる．Fig. 2.5 はイジングモデルのエネルギーとイジングスピンの状態の組合せの関係を表している．イジングモデルにおける SA 法は直接に COP のコスト関数とその解空間から探索を行う代わりに，イジングスピンの状態を更新させながらイジングモデルのエネルギー (ハミルトニア) を最小値に収束させることで，解を探索する [17]．すなわち，Fig. 2.5 で表されているようにあるスピン状態の組合せから，ハミルトニアンを最小値に収束させる方向でスピンの状態を更新していく．

COP をイジングモデルに変換するためには，まず，COP を quadratic unconstrained binary optimization (QUBO) モデル [37] に変換する必要がある．QUBO モデルは [37] 量子コンピューティング分野で幅広く使われる最適化モデルである．イジングモデルのように，バイナリのベクトル x と，ベクトル x の要素間の相互作用を表す行列 Q から構成される．QUBO の定義を式で表すと，

$$\text{QUBO} : \min y = x^T Q x = \sum_{i} \sum_{j} Q_i j x_i x_j, \tag{2.7}$$

である．ここで，QUBO モデルの変数 x はイジングモデルのようにバイナリの状態を持つが，‘-1’ と

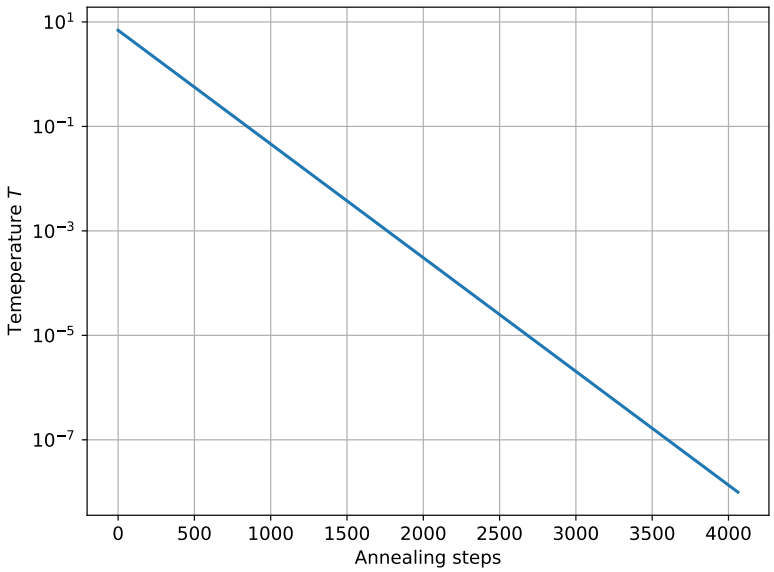


図 2.3: アニーリングステップに対する SA 法の温度パラメータの変化．SA 法では初期の高温から徐々に低温まで温度パラメータを調節する．

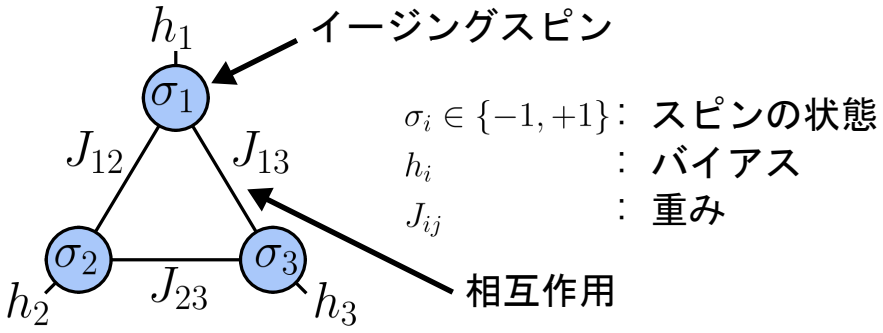


図 2.4: 三つのイジングスピンからなるイジングモデルの構造．イジングスピンはバイナリの状態を持つ．また，それぞれのスピンはバイアスを持ち，周辺のスピンと重みを持つ相互作用で接続されている．

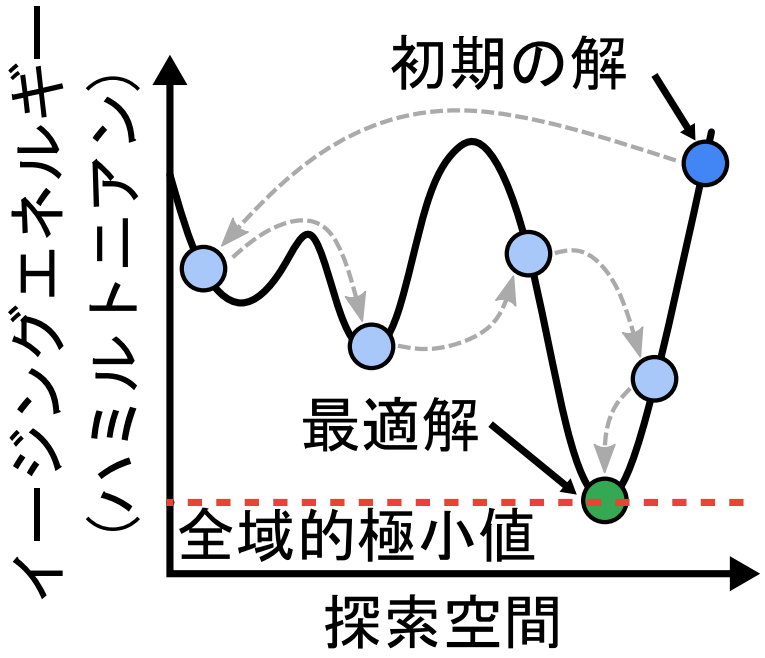


図 2.5: イジングモデルのイジングエネルギーとスピン状態との関係．与えられた組合せ最適化問題の最適解はイジングエネルギーを全域的極小値とするスピン状態で表現される．

‘+1’ ではなく，‘0’ と ‘1’ の状態を持つ．また，イジングモデルの J_{ij} の対角成分は 0 であるが，QUBO モデルの Q は対角成分が存在する上三角行列である．一方，QUBO モデルもイジングモデルと同じく，バイナリ状態の変数の組合せを探索するモデルでるため，QUBO モデルとイジングモデルは等価であり，互いに変換することができる．QUBO モデルの x_i をイジングモデルの σ_i で表すと，

$$x_i = \frac{\sigma_i + 1}{2}, \tag{2.8}$$

であり，Eq. (2.8) を Eq. (2.7) に代入すると，

$$\begin{aligned} y &= \frac{1}{4}\sum_i\sum_j Q_{ij}(1+\sigma_i+\sigma_j+\sigma_i\sigma_j), \\ &= \frac{1}{4}(\sum_i\sum_j Q_{ij}+\sum_i\sum_k (Q_{ik}+Q_{ki})\sigma_i+\sum_i\sum_j Q_{ij}\sigma_i\sigma_j+\sum_i Q_{ii}), \end{aligned} \tag{2.9}$$

である．ここで，Eq. (2.9) の第 2 項だけを考えと，

$$\begin{aligned} \frac{1}{4}\sum_i\sum_j (Q_{ik}+Q_{ki})\sigma_i &= \frac{1}{4}\sum_i \sigma_i(\sum_k (Q_{ik}+Q_{ki})), \\ &= \sum_i \sigma_i(\frac{1}{2}Q_{ii}+\frac{1}{4}\sum_{k\in\partial_i} Q_{ik}), \end{aligned} \tag{2.10}$$

のように展開することができる．ここで，∂_i は σ_i と接続している全ての重みを意味する．また，第 1 項及び第 4 項は σ_i に関係ない定数であるため，無視するできる．Eq. (2.10) を Eq. (2.9) に代入し，Eq. (2.9) の第 1，4 項を無視して式を書き直すと，

$$y = \sum_i \sigma_i(\frac{1}{2}Q_{ii}+\frac{1}{4}\sum_{k\in\partial_i} Q_{ik}) + \sum_i\sum_j \sigma_i\sigma_j\frac{1}{4}Q_{ij}, \tag{2.11}$$

のようである．従って，イジングモデルのパラメータ，h_i と J_{ij}，は，QUBO モデルの Q_{ij} より以下のように求められる．

$$h_i = -\frac{1}{2}Q_{ii} - \frac{1}{4}\sum_{k\in\partial_i} Q_{ik}, \tag{2.12}$$

$$J_{ij} = -\frac{1}{4}Q_{ij}. \tag{2.13}$$

イジングモデルにおける従来 SA 法では，一回のアニーリング処理で，一つのスピンの状態を更新するため，問題サイズに応じてその計算時間が指数関数的に増加する．計算時間の増加を解決するために，前述したように，QA [11] た pSA [14] などの新たなアニーリング手法が研究されている．これらのアニーリング手法では，一回のアニーリングステップで，複数のスピンを更新することができるため，従来 SA 法より高速に最適解を探索できる．しかし，これらの手法を大規模のイジングモデルに適用するためには，更なる研究が必要である．QA はイジングスピンを量子デバイスである qubit で実装しており，その動作には数十 mK という極低温が必要である [38]．また，pSA でのスピンの実装が p-bit [13] という特殊デバイスであり，その実装に MTJ [39] が必要である．加えて，d-wave advantage の qubit の構造は pegasus グラフ [40] であり，スピン間の接続に限界が存在する [41]．COP から変換されるイジングモデルの構造は，スピン間の接続がスパースな物に限らなく，全てのスピンが互いに接続される完全グラフになる場合もある．問題のイジングモデルを，pegasus や chimera グラフのような，イジングソルバーの構造に合わせた構造に変換する手法を minor embedding という [42–44]．Minor embedding は元のモデルを対象モデルの部分グラフに埋め込むアルゴリズムである．しかし，minor embedding はヒューリスティックなアルゴリズムであり，最適なエンベディンググラフの探索は NP 困難問題である．また，完全グラフのモデルをスパースな構造の対処モデルに埋め込む際，必要なスピン数は増加していまう．

例えば，n 個のスピンからなる完全グラフのイジングモデルを，隣接の 8 個のスピンと接続する king‘s graph にエンベディングするためには，少なくとも，$(n-1)^2$ 個のスピンが必要である [28]．すなわち，問題に関係なく，大規模な完全接続のイジングモデルに対応可能なアニーリングハードウェアを実現するためには: 1) 並列なスピンの更新が可能なアニーリングアルゴリズム，2) ハードウェア実装が容易かつ小面積な実装が可能，3) 完全接続のイジングモデルに対応可能なスピン接続が必須不可欠である．

2.4　ストカスティックシミュレーテッドアニーリングの原理

第 2.4 章で述べた，ハードウェア実装が容易かつ並列なスピン更新が可能なアニーリングアルゴリズムがストカスティックシミュレーテッドアニーリング (SSA) である [16]．SSA 法は MTJ デバイスで実装される p-bit の確率的な動作を，確率的コンピューティング方式であるストカスティックコンピューティング [15, 45] を用いて近似したアニーリングアルゴリズムである．SSA 法は巡回セールスマン問題 (traveling salseman problem，TSP) [21]，グラフ同型判定問題 (graph isomorphism，GI) [22]，最大カット問題 (maximum cut problem, MAXCUT) [23] などの COP に対してその有効性を示した [16]．SSA 法について述べる前に，その基盤となるストカスティックコンピューティングについて概説する．ストカスティックコンピューティング (SC) はある情報をストカスティックビット列の “1” の出現確立で表現する確率的な演算方式である [46]．例えば，一般的なデジタル演算であは，符号なし整数のビット列 $(1000)_2$，$(0001)_2$ はそれぞれ “8” と “1” を意味する．一方，二つのビット列での “1” の出現確率は両方とも 0.25 であり，SC では同じ情報を表現するビット列である．SC の表現方法にはユニポーラよバイポーラの二種類が存在する．ビット列 $x(t)$ での “1” の出現確立を P_x とすると，そのビット列がユニポーラ表現で表す数 X_u は:

$$X_u = P_x \ (0 \leq X_u \geq 1), \tag{2.14}$$

で定義される．一方，バイポーラ表現で表される X_b は:

$$X_b = 2P_x - 1 = 2X_u - 1 \ (-1 \leq X_b \geq 1), \tag{2.15}$$

で定義される．例えば，ストカスティックビット列 $(1000)_2$ は，ユニポーラ表現では ‘0.25’ を，バイポーラ表んでは ‘-0.5’ を意味する．

このような SC を用いることで，ハードウェア実装に多くの面積を必要とする計算を小面積で実現することが可能である．例えば，SC での乗算はユニポーラ表現では AND ゲートで，バイポーラ表現では XNOR ゲートで計算できる [19]．Fig. 2.6a はユニポーラ表現で AND ゲートによる乗算器を表す．AND ゲート入力のビット列をそれぞれ A，B とし，出力のビット列を C とすれば，

$$\mathrm{C} = (\mathrm{AND}(\mathrm{A}, \mathrm{B})), \tag{2.16}$$

であり，それぞれのビット列が表す実数を a, b, c とすると，c は

$$c = P_c = P_a \cdot P_b = a \cdot b, \tag{2.17}$$

となる．Fig. 2.6a の例のように，$a = 0.25$，$b = 0.5$ から，$c = 0.125$ となり，AND ゲートより a と b の乗算が正しく計算される．同様に，バイポーラ表現表現では，

$$\mathrm{C} = \mathrm{XNOR}(\mathrm{A}, \mathrm{B}) = \mathrm{OR}(\mathrm{A} \cdot \mathrm{B}, (1 - \mathrm{A}) \cdot (1 - \mathrm{B})), \tag{2.18}$$

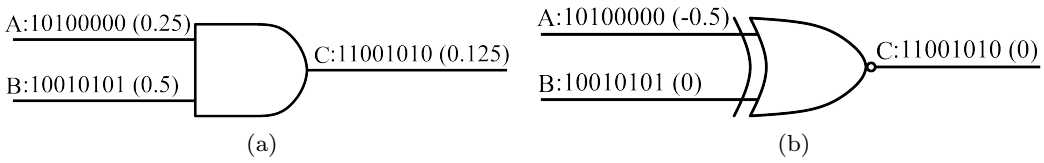


図 2.6: ストカスティックコンピューティングにおける乗算器．(a) ユニポーラ表現での AND ゲートを用いた乗算．(b) バイポーラ表現での XNOR ゲートを用いた乗算.

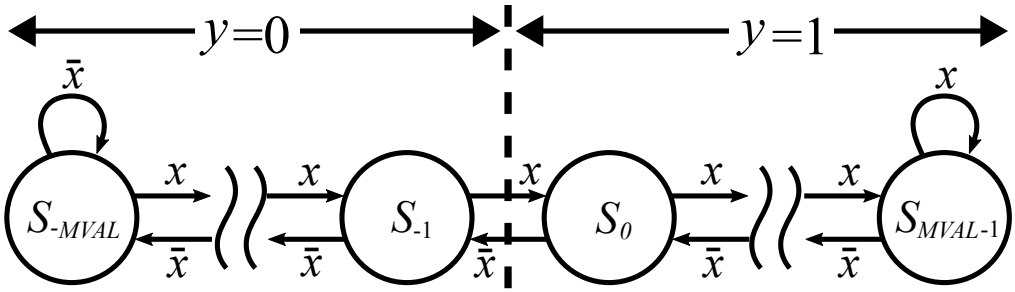


図 2.7: ストカスティックコンピューティングを用いた tanh の近似．ストカスティックコンピューティングにより tanh は有限状態マシンに近似することが可能である.

であり，c は,

$$c = (2P_a - 1)(2P_b - 1) + (2P_{1-a} - 1)(2P_{1-b} - 1), \tag{2.19}$$

となるため，Fig. 2.6b のように XNOR ゲートより乗算が計算できる.

SC を用いた複雑な関数の低コストのハードウェア実装の例として，tanh の実装がある [47]．一般的に tanh は事前に関数の結果をメモリなどに保持して入力に応じて結果を出力する look up tables (LUTs) を用いてハードウェア実装されるため，多くの面積が必要である．このように実装が困難な tanh は SC を用いて有限状態マシン (finite state machine, FSM) で近似することが可能であるため，小面積な実装が可能になる．Fig. 2.7 は SC を用いて近似された tanh の FSM を表す．ここで，x は入力ビット列のビットを表しており，FSM の状態数は $2 \cdot MVAL$ である．Fig. 2.7 が表す近似された $y = \tanh$ を Stanh とすると，Stanh は以下のように定義される:

$$y = \tanh(x \cdot MVAL) \approx \mathrm{Stanh}(2 \cdot MVAL, x). \tag{2.20}$$

Stanh の FSM の計算は，$S(t)$ を時間 t における FSM での現在の状態とすると，

$$S(t) = \begin{cases} MVAL - 1 & \text{if } S(t) + X(t) \geq MVAL, \\ -MVAL & \text{if } S(t) + X(t) < -MVAL, \\ S(t) + X(t) & \text{otherwise,} \end{cases} \tag{2.21}$$

であり，ここで $X(t)$ は時間 t における入力ビットを表す．SC ではバイポーラ表現でも-1 から +1 の実数，ユニポーラ表現では 0 から 1 の実装のように，その表現可能な範囲が制限されている．表現可能な情報を範囲を拡張するために提案されたものがインテグラルストカスティックコンピューティングであ

る [47]．インテグラル SC では，一つの実装を複数のビット列を用いて表現することで，表現可能な数の範囲を 1 以上 (-1 以下) の実数に拡張する．

SSA 法では，インテグラル SC を用いて pSA の p-bit の動作を近似する．P-bit の動作はボルツマンマシン [20] を基盤として，その状態が決まる．ボルツマンマシンはイジングモデルのように，ユニットとそれらの接続 (エッジ) からなるネットワークモデルであり，ネットワークのエネルギーは Eq. (2.6) の同様に定義される．ボルツマンマシンで，一つのユニットが 0 から 1 べ，または 1 から 0 へ状態変位した場合に及ばすグローバルエネルギーの差，ΔE_i は以下のように定義さる．

$$\Delta E_i = \theta_i + \sum_j w_{ij} s_j, \tag{2.22}$$

ここで，s_j はユニット j の状態，θ_i はユニット i のバイアス，w_{ij} はユニット i と j 間のエッジの重みである．ユニット i の状態が 0 から 1 に変位した時のグローバルエネルギーの差は:

$$\Delta E_i = E_{i=0} - E_{i=1}, \tag{2.23}$$

であり，ボルツマン分布に従ったそれぞれのエネルギーを満たす確率を用いると，Eq. (2.23) は:

$$\Delta E_i = -k_B T \ln(p_{i=0}) + k_B T \ln(p_{i=1}), \tag{2.24}$$

ここで，k_B はボルツマン因子，T は温度である．式を簡単にするため，定数であるボルツマン因子を省略し，Eq. (2.24) をユニット i が 1 である確率，$p_{i=1}$，について展開すると以下のようである．

$$\begin{aligned}
\frac{\Delta E_i}{T} &= \ln(p_{i=1}) - \ln(p_{i=0}), \\
&= \ln(\frac{p_{i=1}}{1-p_{i=1}}), \\
&= -\ln(\frac{1-p_{i=1}}{p_{i=1}}), \\
&= -\ln(\frac{1}{p_{i=1}} - 1),
\end{aligned} \tag{2.25a}$$

$$\exp(-\frac{\Delta E_i}{T}) = \frac{1}{p_{i=1}} - 1, \tag{2.25b}$$

$$p_{i=1} = \frac{1}{1+\exp(-\frac{\Delta E_i}{T})}. \tag{2.25c}$$

Eq. (2.25c) のように，ボルツマンマシンでユニット i が 1 になる確率はユニットが及ばすエネルギーの差と温度 T より決まる．温度 T を $\frac{1}{I_0}$ に置き換え，$I_i = I_0 \Delta E_i$ とすると Eq. (2.25c) は:

$$p_{i=1} = \frac{1}{1+\exp(-I_i)}, \tag{2.26}$$

であり，I_i によりユニットの状態の確率が決まる．I_i は Eq. (2.22) より以下のように定義される．

$$I_i = I_0(\theta_i + \sum_j w_{ij} s_j). \tag{2.27}$$

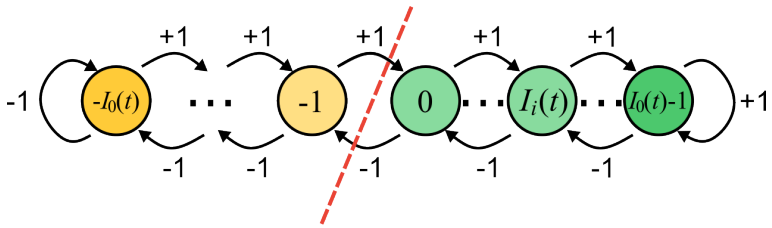


図 2.8: ストカスティックコンピューティングにより近似された Itanh 関数を計算する有限状態マシン．FSM の状態数は SSA の逆温度パラメータである I_0 により決まる．

また，SA 法のようにユニット i の状態，s_i，はユニット状態の確率と乱数の比較より決まるため，Eq. (2.26) を tanh 関数を用いると，s_i は

$$s_i = \mathrm{sgn}(\mathrm{rnd} + \tanh(I_i)), \tag{2.28}$$

と定義できる．Eq. (2.27) 及び Eq. (2.28) をイジングモデルのパラメータ (h_i，J_{ij}) を用いて表すと，pSA での時間 t におけるスピン状態式は以下のように定義される [14].

$$I_i(t+1) = I_0(h_i + \sum_j J_{ij}\sigma_i(t)), \tag{2.29}$$

$$\sigma_i(t+1) = \mathrm{sgn}(\mathrm{rand} + \tanh(I_i(t+1))). \tag{2.30}$$

pSA では Eqs. (2.29) and (2.30) の動作を行う MTJ デバイスである p-bit を用いてイジングスピンを実装する．

SSA 法では p-bit の動作をインテグラル SC を用いて近似することで，イジングスピンの確率的動作を CMOS 回路を用いて実現する [48]．すなわち，SSA 法では，Eqs. (2.29) and (2.30) のように定義されるスピン動作式を以下のように近似する．

$$I_i(t+1) \approx h_i + \sum_j J_{ij}\sigma_j + r_i(t) \cdot n_{rnd}, \tag{2.31}$$

$$\sigma_i(t+1) = \begin{cases} 1 & \text{if } Itanh_i(t+1) \geq 0, \\ -1 & \text{otherwise}, \end{cases} \tag{2.32}$$

ここで，n_{rnd} は乱数の大きさであり，SSA 法のハイパーパラメータの一つである．また，Eq. (2.31) で，Itanh は SC より近似された tanh 関数であり，以下のように定義される．

$$\mathrm{Itanh}_i(t+1) = \begin{cases} I_0(t) - 1 & \text{if } \mathrm{Itanh}_i(t) + I_i(t+1) \geq I_0(t), \\ -I_0(t) & \text{else if } \mathrm{Itanh}_i(t) + I_i(t+1) < -I_0(t), \\ I_i(t+1) & \text{otherwise}. \end{cases} \tag{2.33}$$

SSA 法では，I_0 より I_i をスケーリングすることではなく，Itanh_i を計算する FSM の状態数として温度を利用する．Fig. 2.8 は Eq. (2.33) を計算する FSM を表す．また，乱数をスピン更新の確率として利用することでなく，Eq. (2.31) のように直接に与えることでスピンの確率的動作を実現する．SSA 法と

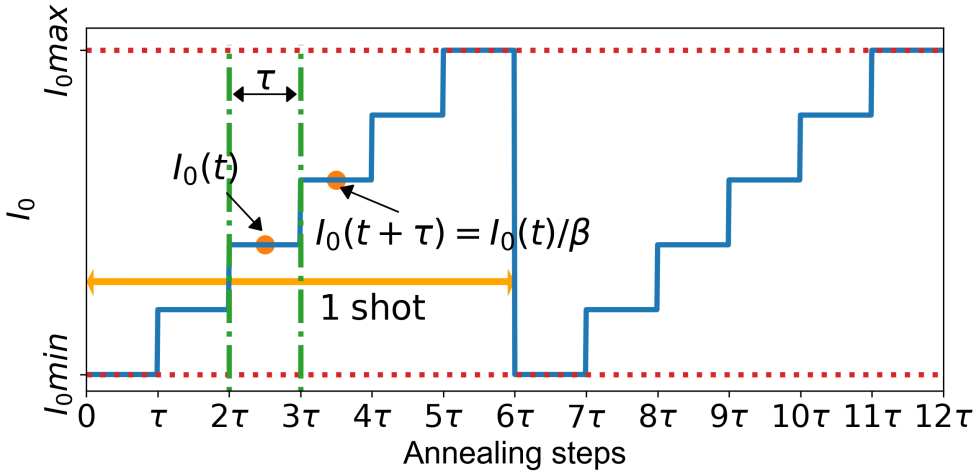


図 2.9: SSA 法のアニーリングステップに対する逆温度パラメータ，$I_0(t)$，の動作.

pSA 法の大きな違いは，今までのスピン更新の影響を累積で考慮することである．Eq. (2.29) のように，pSA 法では，現在の I_i のみでスピン状態を決める．一方，SSA 法のスピン動作は Eq. (2.31) の第 4 項のように，今までの計算結果を累積してスピン状態を決める．また，SSA 方が SC を基盤としているため，Eqs. (2.31) and (2.33) で定義されるスピンの動作を全て CMOS 回路で実装可能である．

SSA 法も従来 SA 法のように温度によってスピン状態の更新の確率を制御する [16]．SA 法では高い温度から低い温度に温度を冷却しながら，改悪の更新の発生頻度を減らしてスピンが最適な状態へ収束するように処理を行う．一方，SSA 法は I_0 で定義される逆温度を用いてスピンの状態更新の確率を制御する．Fig. 2.9 は SSA 法のアニーリングステップ，t，に対する逆温度パラメータ，$I_0(t)$，を表す．$I_0(t)$ は以下の式に基づき，t に応じて逆温度の最小値，I_{0min}，から最大値，I_{0max}，まで増加する．

$$I_0(t+\tau) = \frac{I_0(t)}{\beta}, \tag{2.34}$$

ここで，τ は逆温度が一定に維持されるアニーリングステップ数，β は逆温度の増加率である．逆温度パラメータ，$I_0(t)$，は SC より近似された Itanh の FSM の状態数として用いられる．$I_0(t)$ が小さい場合はスピン状態，σ_i，が反転するまでに必要な FSM の状態数が少ないため，スピン状態の更新が活発に行われる．従って，低い逆温度の場合にはスピン状態を活発に変化させながら広い範囲の解空間での最適化を行う．一方，高い逆温度の場合には，スピン状態の反転が起きにくくなり，スピンは互いの接続に応じて最適な状態へ収束する．

また，SA の温度はアニーリング処理が進むことに連れて一方的に冷却されるように制御される．一方，SSA の逆温度は比較的に短いステップ数で最小値から最大値まで増加し，最大値に達した後，再び最小値に戻り同じ動作を反復する．Fig. 2.9 の例では，6τ ステップで逆温度は最大値である I_{0max} に到達し，$6\tau + 1$ ステップからは I_{0min} から増加する．SSA のアニーリング処理では逆温度が最小値から最大値まで増加する一連の処理を ‘shot’ という処理の単位として，この ‘shot’ を繰り返す．すなわち，SSA の全体のアニーリング処理は shot を何回繰り返すかによってその計算時間が決まり，1 shot 当た

表 2.1: SSA 法のハイパーパラメータの一覧.

ハイパーパラメータの名前	ハイパーパラメータの説明
I_{0min}	逆温度パラメータの最小値
I_{0max}	逆温度パラメータの最大値
n_{rnd}	乱数の大きさ
τ	逆温度パラメータが一定な値を維持するアニーリングステップ数
β	逆温度パラメータの増加率
$mshot$	Shot の反復回数
$trial$	SSA の全体の処理の反復回数

りのアニーリングステップは:

$$(\text{1 shot 当たりのアニーリングステップ数}) = \log_\beta \frac{I_{0min}}{I_{0max}} + 1 \cdot \tau, \tag{2.35}$$

である．Table 2.1 は SSA のハイパーパラメータの一覧とその説明である．SSA 法も従来 SA 法のように確率的アルゴリズムであるため，一つの問題に対して複数回試行を反復してその結果を確認することが一般的である．$trial$ は SSA 法の全体のアニーリング処理の反復回数であり，$mshot$ は一回のアニーリング処理 (1 trial) での shot の反復回数である．SSA 法の動作に直接的に関与するハイパーパラメータは，I_{0min}，I_{0max}，n_{rnd}，τ，β の 5 種類であり，与えられた COP の最適解を探索するためには適切なハイパーパラメータの組合せを決める必要がある．

Fig. 2.2 及び Fig. 2.3 のように，イジングモデルにおける SA 法では温度の減少に連れて，イジングエネルギーが減少する．同様に SSA 法でも逆温度の増加に連れてイジングエネルギーが減少していく．ただ，SSA の逆温度は一回の shot が終わる時，逆温度が減少する (最小値に初期化) ため，新しい shot の

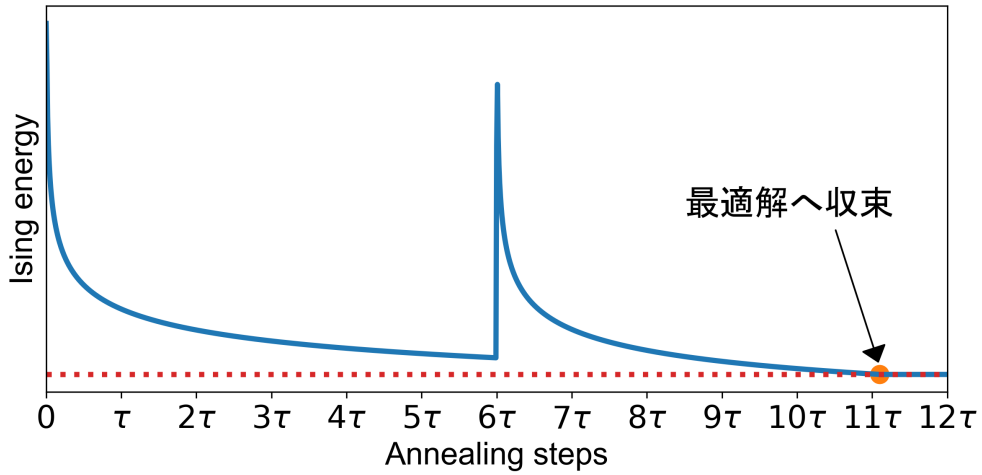


図 2.10: SSA 法のアニーリング処理中のイジングエネルギーの変化．SSA の逆温度パラメータの変化に応じて，エネルギーが減少する．

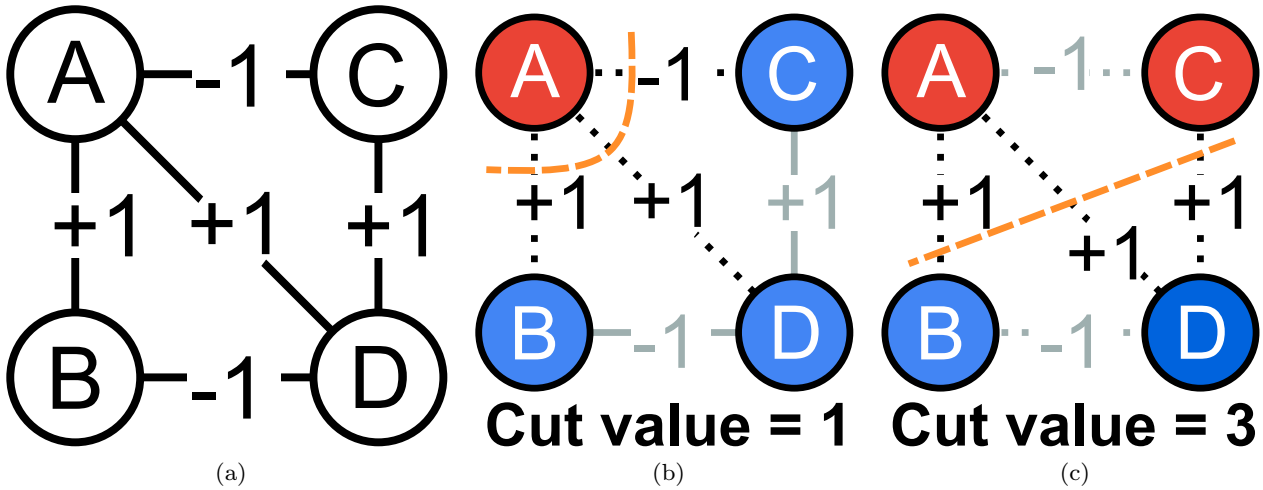


図 2.11: MAXCUT 問題の例．(a) 四つのノードからなる MAXCUT 問題．例題の最適であるカット値は 3 である． (b) 例題の間違った解の例． (c) 例題の最適解の例.

始まりでは一時的にエネルギーが増加する．Fig. 2.10 は Fig. 2.9 に対応した，SSA のアニーリングステップに対するイジングエネルギーの変化を簡略に表したものである．SA 法は改悪のスピン状態更新を確率的に受け入れることで，局所的最適解から脱出して全域的最適解を探索する．SSA 法ではスピン状態の変化に乱数を用いた確率的な動作を取り組むことで局所的最適解から脱出できる．それに加えて，短い時間で逆温度の増減を繰り返すことで，少ない数のスピンの更新では脱出できない，深い局所的最適解からも脱出が可能になる．

それでは，SSA 法の動作を確認し，アニーリング精度や計算速度など，その性能を従来 SA 法 [10] と比較する．性能評価には代表的な COP の一つである MAXCUT 問題を用いる．MAXCUT 問題はあるグラフを二つの部分グラフに分割する際，二つの部分グラフの間のエッジの重みの総和が最大になるグラフの分割のやり方を探索する COP である [23]．Fig. 2.11 は簡単な MAXCUT 問題を例であり，Fig. 2.11a のように四つのノード (A，B，C，D) とそれらを繋ぐ五つのエッジからなるグラフである．Fig. 2.11a のグラフを G とすると，$V(G) = \{A, B, C, D\}$ である．G を Fig. 2.11b のように G_1, $(V(G_1) = \{A\})$ と G_2, $(V(G_2) = \{B, C, D\})$ に分割すると，グラフ 1 と 2 の間のエッジの重みの総和は 1 である．一方，Fig. 2.11c のように G_1, $(V(G_1) = \{A, C\})$ と G_2, $(V(G_2) = \{B, D\})$ に分割するとカット値は 3 である．Fig. 2.11a の問題の最適解 (最大のカット値) は 3 であるため，Fig. 2.11b は不正解，Fig. 2.11c は正解 (最適解) となる．SSA 法の動作確認には，MAXCUT 問題のベンチマークデータセットである ‘G-set’ を用いる [24]．G-set は，様々なノード数及びグラフ構造を持つグラフのデータセットであり，組合せ最適化アルゴリズムの評価などに多く使われており，G-set に含まれる問題のインスタンスの最良解がすでに知られている．実験では G-set の一つのインスタンスである，G11 問題を用いる．G11 問題は 800 個のノードからなる MAXCUT 問題であり，グラフの構造はトロイダルグラフでエッジの数は 1,600 個である．すなわち，G11 問題の一つのノードは隣接の四つのノードと-1 または +1 の重みのエッジでつながっている．また，G11 問題の最良解が 564 で知られている．

従来 SA 法及び SSA 法は Python 3.8 を用いて実装され，実験に用いた central processing unit (CPU)

は Intel Xeon Gold 6252 であり，動作周波数は 2.1 GHz である．G11 問題に対して SSA 法の動作を確認する前に，与えられた問題に応じて最適なハイパーパラメータの組合せを探索する必要がある．Table 2.2 は G11 問題に対して最適なハイパーパラメータの組合せをまとめたものでる．*Mshot* は 150 であり，Eq. (2.35) から計算した 1 shot 当たりのアニーリングステップは 600 であるため，全体の処理は 90,000 ステップである．従って，従来 SA 法も 90,000 ステップのアニーリング処理を行い，その結果を SSA 法の結果と比較する．また，*trial* は 100 であり，SSA 法及び SA 法は 90,000 ステップのアニーリング処理を 100 回繰り返す．

表 2.2: G11 問題における SSA 法の最適なハイパーパラメータの組合せ．

ハイパーパラメータ	最適な値
I_{0min}	1
I_{0max}	32
n_{rnd}	2
τ	100
β	0.5
mshot	150
trial	100

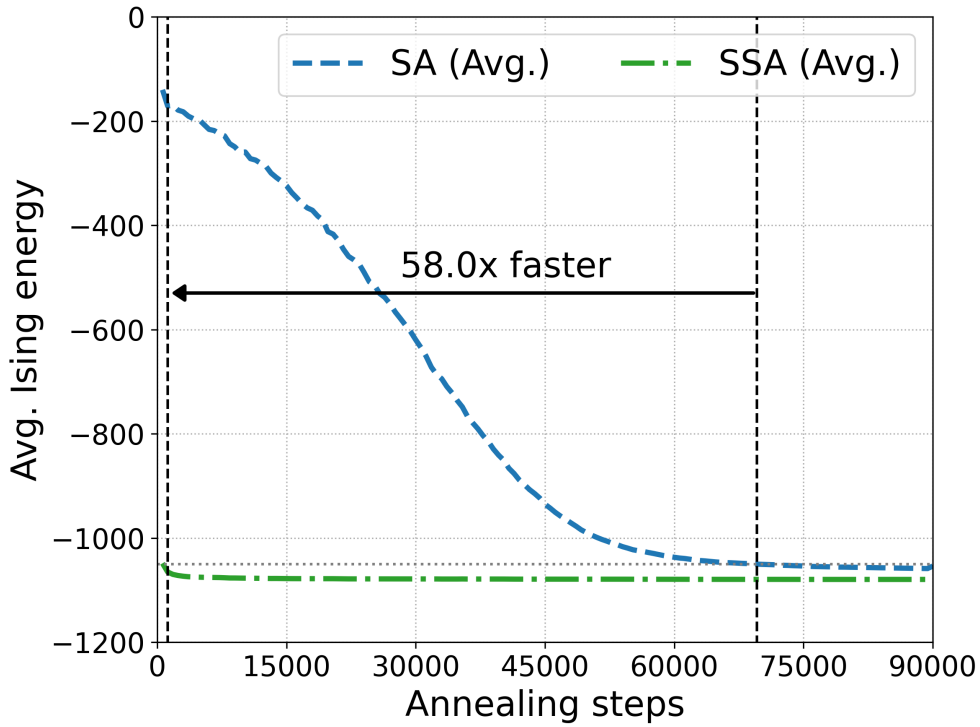
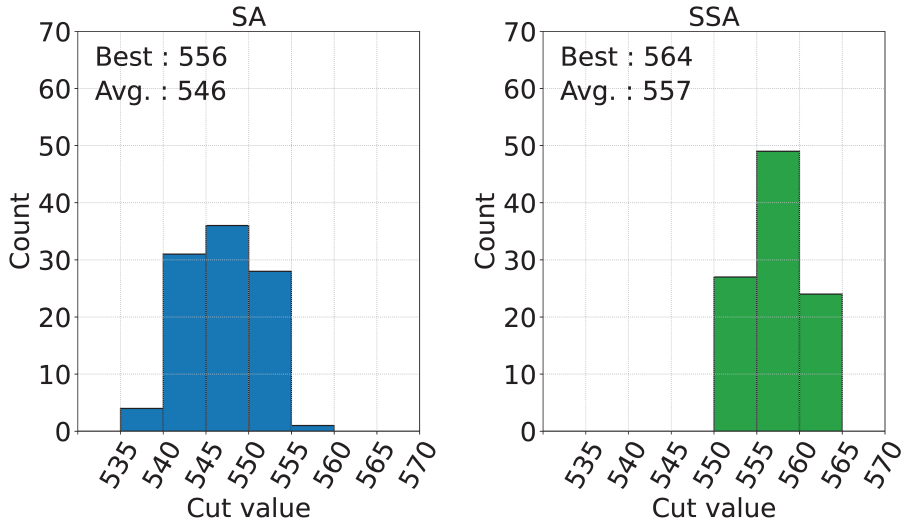


図 2.12: G11 問題における，アニーリングステップに対する従来 SA 法及び SSA 法のイジングエネルギーのグラフ．SSA 法は従来 SA 法より約 58 倍高速で G11 問題の最良解の 99% に到達した．



(a)　(b)

図 2.13: G11 問題における従来 SA 法及び SSA 法の 100 trial のカット値のヒストグラム．(a) 従来 SA 法のヒストグラム．(b) SSA 法のヒストグラム．

Fig. 2.12 は G11 問題における，SA 法及び SSA 法のアニーリングステップに対する平均イジングエネルギーを表しており，平均エネルギーは 100trial の平均である．従来 SA 法は G11 の最良解の 96% に 69,600 ステップで到達した．一方，SSA 法は従来 SA 法の約 58 倍高速である 1,200 ステップで最良解の 96% に到達しており，2 shot だけで近似解を得ることができる．Fig. 2.13a 及び Fig. 2.13b はそれぞれ従来 SA 法及び SSA 法から得られたカット値をヒストグラムで表したものである．従来 SA 法から得られた 100 trial の平均カット値は 546 であり，最大値は 556 である．一方，SSA 法の平均値は 557 であり，最大値は G11 問題の最良解である 564 を得ることができた．これらの結果のように，SSA 法は従来 SA 法に比べ，最適解 (または近似解) への収束が高速であり，得られた解もより高精度である．

2.5　むすび

本章では COP からイジングモデルの変換を含め，イジングモデルにおける SA 法の基礎，また従来 SA 法の課題を解決するために提案された新たなアニーリングアルゴリズムについて概説した．そのようなアルゴリズムの一つとして確率的演算方式である SC に基づいた SSA 法について述べ，その動作を COP の一つである MAXCUT 問題を用いて確認した．また，SSA 法の計算速度やアニーリング精度などの性能を従来 SA 法の性能と比較し，SSA 法の有効性確認した．

第 3 章

ストカスティックシミュレーテッドアニーリング (SSA) のハードウェア実現

3.1 まえかき

本章では SC に基づいた SSA 法のハードウェア実装について述べる．提案 SSA ハードウェアは一つのイジングスピンが隣接の 8 個のスピンと接続するスパースな構造を持つ 800 個のスピンを搭載する．初めに提案 SSA ハードウェアのアーキテクチャについて述べ，ハードウェア実装に向けたメモリ管理や改善などについて述べる．また，SSA のスピン動作に基づいて設計されたハードウェアの基本コンポネントであるスピンゲートについて説明する．提案 SSA ハードウェアの性能検証としては MAXCUT 問題をハードウェアより解き，その結果を従来 SA 法の性能と比較する．

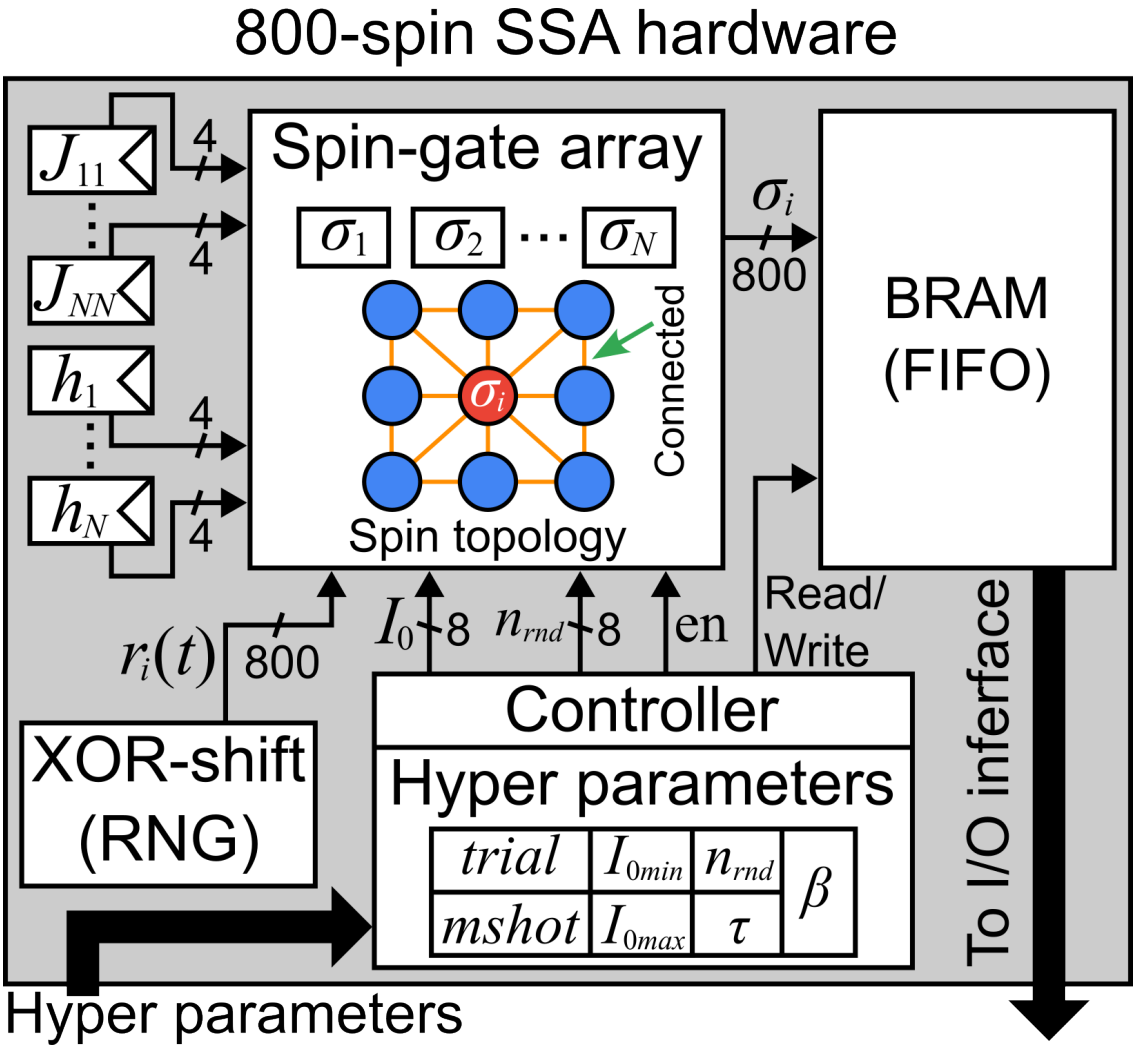


図 3.1: SSA ハードウェアのアーキテクチャ．提案ハードウェアは 800 個のスピンを搭載しており，一つのスピンは隣接の 8 個のスピンと接続する．また，ハードウェアを制御するためのコントロールや結果を保存するためのメモリを含む．

3.2　SSA ハードウェアのアーキテクチャ

前章で述べたように SSA 法は同時に複数のイジングスピンの更新が可能であるため，イジングモデルにおいて従来 SA 法より高速に最適解を探索することができる．加えて，QA や pSA のように特殊デバイスを必要とするアニーリング法とは異なり，SC を用いた近似より CMOS 回路でイジングスピンの確率的な動作を実装可能である．一方，SSA 法の有効性はソフトウェアを用いた実装より確認されたが，ハードウェア実装による有効性はまだ検証が必要である．従って，SSA 法に基づいたアニーリングハードウェアを設計し，ハードウェア実装による性能向上を検証する．

Fig. 3.1 は提案 SSA ハードウェアのアーキテクチャを表す．提案 SSA ハードウェアはイジングモデルのスピンに対応するスピンゲートのアレーである ‘spin-gate array’ をメインコンポネントとして，ハー

ドウェアを制御するための ‘controller’，スピンの確率的な動作のための乱数信号を生成する ‘random number generator (RNG)’，イジングモデルのバイアスと重みを保持するレジスタ，アニーリング結果を保存するためのメモリを含める．Spin-gate array は 800 個のスピンゲートからなるスピンのアレーである．一つのスピンゲートは隣接の 8 個のスピンゲートと接続している，king‘s graph の構造を持つスパースなグラフ構造である．従って，提案 SSA ハードウェアは一つのスピンが最大に 8 個のスピンと接続するイジングモデルまで対応可能である．スピンゲートは Eqs. (2.31) to (2.33) に基づいて設計された CMOS 回路であり，その詳細は Section 3.3 で述べる．Spin-gate array は controller から現在の逆温度パラメータ，$I_0(t)$，及び乱数信号の大きさ n_{rnd} を入力として受け取る．SSA でのスピン動作には乱数信号を用いて確率的動作を行うため，スピンゲートも乱数信号を必要とする．それぞれのスピンゲートは 1 ビットの乱数信号を用いるため，spin-gate array は 800 ビットの乱数信号を入力として受け取る．また，イジングモデルのパラメータをレジスタから受け取り，対応するスピンにつながる．800 スピンを搭載しており，スピン当たりの接続数は 8 であるため，重み，J_{ij}，の数は 3,200 であり，バイアスの数は 800 である．また，リセットやスピンゲートの動作開始信号などの制御信号も controller から入力される．Spin-gate array の出力は各アニーリングステップでのスピンの状態を出力する．従って，1 アニーリングステップで 800 ビットのビット列を出力し，メモリに格納する．

イジングモデルの重みとバイアスはレジスタに格納される．一般的にこのようなパラメータはメモリを用いて格納することが一般的であるが，提案 SSA ハードウェアはアニーリングステップ当たり 800 ビットのアニーリング結果を格納するため，メモリの代わりにレジスタを用いてパラメータを保持する．提案 SSA ハードウェアの重みの数は 3,200 個，バイアスは 800 個であるため，それぞれ 3,200 個，800 個のレジスタが搭載されている．また，提案ハードウェアが対応可能なパラメータの 4 ビットの符号付き整数であり，それぞれのレジスタも 4 ビット幅のレジスタである．

RNG は Eq. (2.31) の $r_i(t)$ 信号を生成する．前述のように各スピンゲートは 1 ビットの乱数信号を用いるため RNG は 800 ビットの乱数信号毎アニーリングステップで生成する．RNG としては RNG のコストを削減するために xorshift [49] を用いており，シードは 1,024 ビットである．RNG の制御信号はリセット信号，乱数生成信号の二種類であり，それらの信号は controller から生成される．

Spin-gate array，RNG，メモリなど，提案ハードウェアの動作は全て controller より制御される．Contorller はハードウェア外部から Table 2.1 に示されたハイパーパラメータを受け取り，それらのハイパーパラメータに応じてハードウェアを制御する．また，リセット信号などの制御信号の生成以外に逆温度パラメータ，$I_0(t)$，及び乱数信号の大きさ n_{rnd} 信号を生成して spin-gate array に入力する．SSA 法の逆温度パラメータの制御は Eq. (2.34) に従って行われる．しかし，Eq. (2.34) に基づいた $I_0(t)$ の増加には乗算器または除算器が必要であるため，controller のコストが大きくなってしまう．提案 SSA ハードウェアでは controller のコストを削減するために，よりハードウェアフレンドリーな逆温度パラメータの制御を行う．逆温度パラメータは I_{0min} から I_{0max} まで増加するため，Eq. (2.34) での除数である β の範囲は 0 から 1 の間の実数である．つまり，β を 1 以上にすることで，Eq. (2.34) の除算を乗算に置き換えるとこができる．加えて，$I_0(t)$ を整数に限定すると，提案 SSA ハードウェアの逆温度パラメータの制御は以下のように変更可能である．

$$I_o(t+\tau) = I_0 \cdot 2^{\beta} \tag{3.1}$$

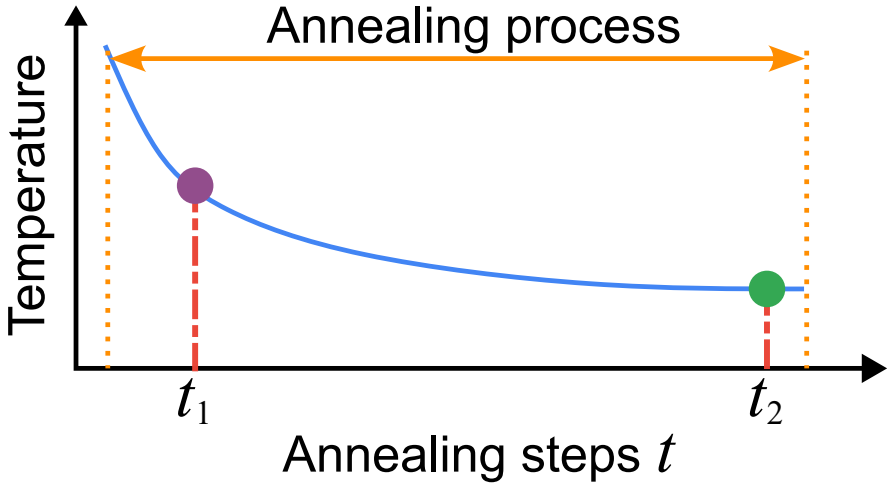


図 3.2: 従来 SA 法において，異なるアニーリングステップの時のエネルギーの比較．SA 法の温度は一方的に冷却されるため，エネルギーも一方的に減少する．

Eq. (3.1) に基づいた逆温度パラメータの精度が整数のみになるが，SSA 法は整数のみのハイパーパラメータだけでも高いアニーリング精度を達成敵ることが既に知られている [16]．また，β を 2 の指数として用いることで，乗算ではなくシフト演算より逆温度パラメータの増加を計算可能になるため，controller のコストを削減できる．逆温度パラメータ制御の変更より，Eq. (2.35) で表される 1 shot 当たりのアニーリングステップ数も以下のように変更できる．

$$(\text{SSA ハードウェアでの 1 shot 当たりのアニーリングステップ数}) = \log_{2^\beta} \frac{I_{0min}}{I_{0max}} + 1 \cdot \tau, \qquad (3.2)$$

Spin-gate array から出力されたアニーリング結果 (スピンの状態は)Block-RAM (BRAM) に保存される．提案 SSA ハードウェアのメモリは搭載スピン数に応じて入出力ビット幅が 800 ビットの first in，first out (FIFO) BRAM である．メモリの制御も他のコンポネントと同様に controller から制御される．メモリに保存されたアニーリング結果は通信インタフェースより外部に伝送される．提案ハードウェアの通信インタフェースはシリアル通信インタフェースである universal asynchronous receiver transmitter (UART) [50] を用いるため，800 ビットの結果はバッファを用いて，8 ビットのデータパケットとして外部に伝送される．

イジングモデルにおける従来 SA 法 [10,17] はアニーリング処理が進むことに連れて温度を冷却し，イジングエネルギーを最小値に収束させる．この際，SA 法の温度は Fig. 3.2 のように高い温度から低い温度に徐々に冷却される．低い温度では改悪のスピン更新を受け入れる確率が低くなり，エネルギーを減少させる更新のみを受け入れる．Fig. 3.2 でのアニーリングステップ，t_2 $(> t_1)$，の時のイジングエネルギー，$H(t_2)$，は $H(t_1)$ より低い確率が高い．従って，SA 法のアニーリング結果は与えられたアニーリングステップの処理が全て終了し，最後のスピン状態を確認することで最良解を得ることができる．一方，SSA 法の逆温度パラメータは Fig. 3.3 のように I_{0min} から I_{0max} までを増減する動作を複数繰り返す．従って，Fig. 3.3 の二点 t_1 と t_2 において，従来 SA 法のように各点でのエネルギー $H(t_2)$ が $H(t_1)$ より低いとは限らない．従って，SSA 法は与えられたアニーリングステップの処理が全て終わる前にも最良解を得る可能性がある．このように SSA 法は処理中の結果を確認ることで，与えられたアニーリン

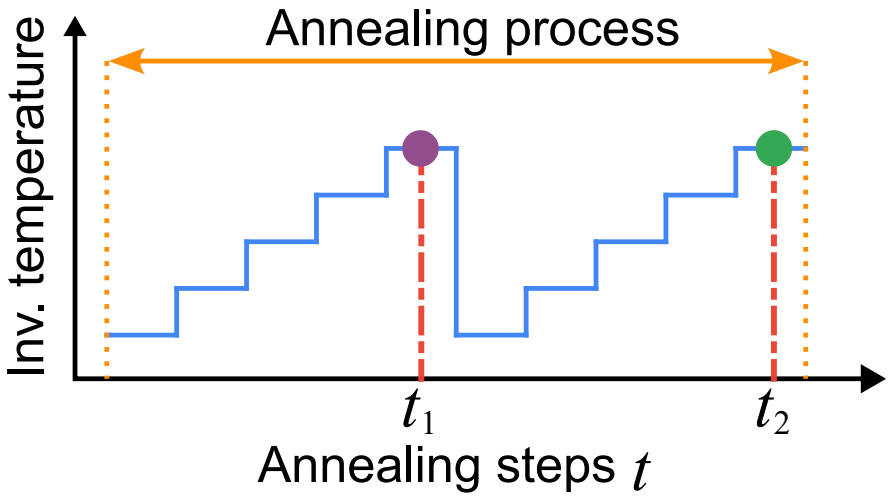


図 3.3: SSA 法において，異なるアニーリングステップの時のエネルギーの比較．SSA 法の逆温度は増減を反復するため，以前のエネルギーが現在のエネルギーより低い可能性がある．

グステップ数より早めに処理を終了させたり，複数の結果からより精度が高い解を得ることができる．このような SSA 法の特徴を生かすためには処理途中の結果を保持または出力する必要がある．提案 SSA ハードウェアは 800 スピンが搭載されているため，各アニーリングステップで 800 ビットのスピン状態をアニーリング結果として生成する．1 アニーリングステップを計算するために必要なクロックサイクル数は 1 サイクルであり，ハードウェアの動作中には 1 クロックで 800 ビットの結果が生成されるため，それらをリアルタイムで外部に出力することは困難である．従って，提案ハードウェアはそれらの結果を FIFO メモリに保存して外部に出力する．しかし，一般的に SSA 法のアニーリング処理が数万以上のアニーリングステップの計算を行うため，全ての結果を保存するためには膨大な容量のメモリが必要である．SSA ハードウェアで 1 trial で生成される全ての結果を保存するためのメモリ使用量を Mem_u とすと，Mem_u は:

$$Mem_u = mshot \cdot N \cdot (\log_{2^\beta} \frac{I_{0min}}{I_{0max}} + 1) \cdot \tau \text{ bits} \tag{3.3}$$

であり，ここで，N はハードウェアの搭載スピン数である．

提案 SSA ハードウェアは途中結果を保存するためのメモリ使用量を削減するために，アニーリング処理中の逆温度パラメータに応じて保存する結果を選択する．Fig. 3.4 は SSA 法を用いて G11 問題を解く場合，アニーリングステップに対する逆温度パラメータ及びイジングエネルギーの変化を表す．図からわかるように，SSA 法も SA 法と同様に，逆温度パラメータが最大値の時により最適解に近い解に収束する．すなわち，途中結果を全て保存するより，逆温度パラメータが最大である時のみの結果を保存するだけで，最良解を得ることができる．逆温度パラメータが最大の時の結果のみを保存する場合のメモリ使用量，Mem'_u は:

$$Mem'_u = mshot \cdot N \cdot \tau \quad \text{bits} \tag{3.4}$$

であり，全ての結果を保存する時より，$(\log_{2^\beta}(I_{0min}/I_{0max}) + 1)$ 倍メモリ使用量を削減できる．

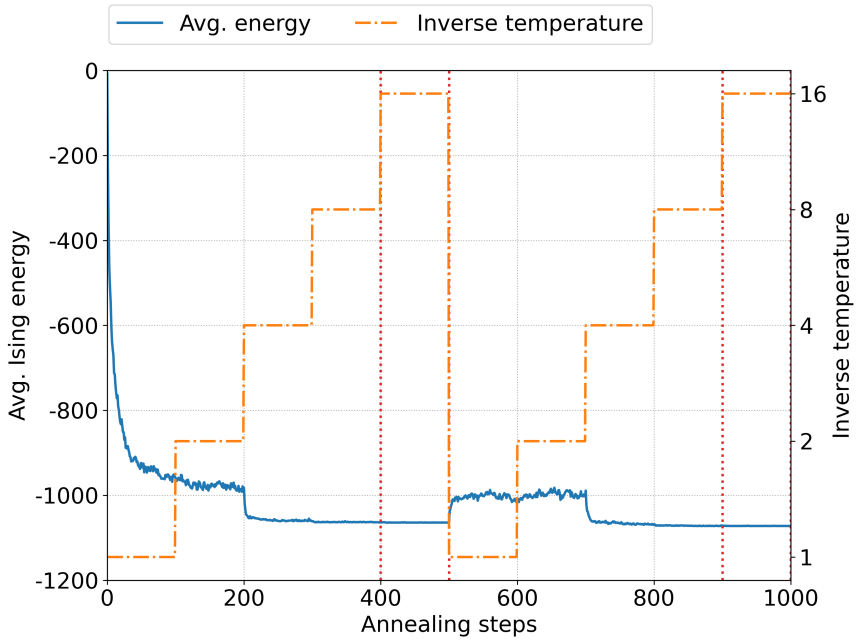


図 3.4: SSA 法のアニーリングステップに対する逆温度パラメータ及びイジングエネルギーの変化．SA 法のように SSA 法も逆温度パラメータが最大値である場合，最良解に収束する．

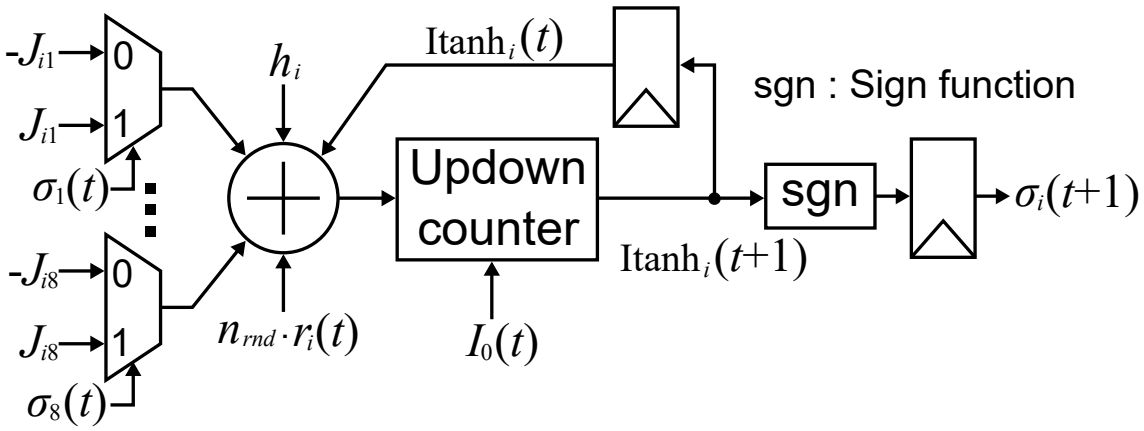


図 3.5: SSA ハードウェアでイジングスピンに対応する spin-gate の回路図.

3.3　SSA ハードウェアにおけるイジングスピンの回路構成

提案 SSA ハードウェアでは Eqs. (2.31) to (2.33) に基づいて設計された ‘spin-gate’ よりイジングスピンを実装する．SSA 法のスピン動作は SC に基づくため，MTJ や量子ビットのような特殊デバイスではなく，CMOS 回路を用いてスピンを実装することが可能である．従って，既存の CMOS 回路の設計技術や製造技術をそのまま活用することも可能であり，回路の集積化が容易である．Fig. 3.5 は spin-gate の回路図であり，spin-gate は一つのイジングスピンに対応する．従って，提案ハードウェアのメインコ

ンポネントである spin-gate array は 800 個の spin-gate から構成される．Spin-gate は接続されているスピンとの相互作用 ($J_{ij} \cdot \sigma_j$) をマルチプレクサを用いて計算する．提案ハードウェアは一つのスピンが隣接の 8 個のスピンと接続する king‘s graph に基づいて設計されているため，spin-gate は 8 個のマルチプレクサを持つ．スピン間の相互作用はバイナリアダーよりバイアス，h_i，及び乱数信号，$n_{rnd} \cdot r_i(t)$，と加算される．また，以前のアニーリングステップでの Itanh_i も一緒に加算される．

Section 2.4 で SC を用いて近似された Itanh は Fig. 2.8 で示された FSM より計算が可能であり，その FSM は Fig. 3.5 の saturated updown counter より実装される．この際，FSM の状態数は逆温度パラメータ，$I_0(t)$，によって決まり，$I_0(t)$ が小さい場合はスピンの状態変化が活発に行われる．一方，$I_0(t)$ が高い場合はスピン状態が反転するために必要な FSM の状態遷移が多くなるため，スピン状態の変化が穏やかになる．前述したように，$I_0(t)$ 信号は提案ハードウェアの controller から生成される．Saturated updown counter の計算結果は次のアニーリングステップに用いられるため，レジスタに保持される．

イジングスピンの状態は-1 または +1 であり，Eq. (2.32) のように Itanh によってその状態が決まる．Spin-gate では saturated updown counter の出力を sgn 関数 (sign 関数) を用いてスピン状態を決める．この際，spin-gate はディジタル回路であるため，スピン状態 ‘-1’ を論理値 “0” で，スピン状態 ‘+1’ を論理値 “1” で表現することで，1 ビットだけでスピン状態を表現する．Spin-gate の出力は対応するイジングスピンの状態であり，アニーリング結果として提案ハードウェアのメモリに保存される．また，隣接のスピンとの相互作用の計算にスピン状態が使われるため，spin-gate array の内部では，提案 SSA ハードウェアのグラフ構造に応じて spin-gate の入出力がつながっている．このように spin-gate は確率的に動作するイジングスピンを小面積な CMOS 回路より実装可能であるため，SSA 法に基づくアニーリングハードウェアは大規模化が容易である．

3.4　SSA ハードウェアの実験環境

800 スピン SSA ハードウェアの性能評価のために，FPGA を用いて実装を行い性能を確認する．実装に用いた FPGA ボードは Xilinx 社の Kintex 7 (XC7K325T-2FFG900C) が搭載されている Digilent 社の Genesys 2 である．Genesys 2 の BRAM 容量は 16 Mbit であり，Section 3.2 で述べたように，800 ビットのアニーリング結果を保存する．提案 SSA ハードウェアの RTL 設計は SystemVerlig より記述され，FPGA への論理合成及び配置配線は Xilinx Vivado 2022.1 を用いて行われる．Fig. 3.6 は FPGA に実装された SSA ハードウェアの実験環境を表す．SSA 法の動作に必要なハイパーパラメータは FPGA 外部から UART インタフェースを用いてハードウェアに入力される．ハイパーパラメータの入力後，FPGA ボードに搭載されているボタンより，ハードウェアはアニーリング処理を開始する．また，ハードウェアのリセット信号もボタンより生成される．BRAM に保存された 800 ビットのアニーリング結果も UART インタフェースよりハードウェア外部に伝送され，SSA ハードウェアのアニーリング結果を確認できる．ハードウェアへのハイパーパラメータ入力及びハードウェアからのアニーリング結果の受け取りは UART で接続されている PC より行われる．使用する PC の CPU は Intel Core i9-12900KF であり，CPU の動作周波数は 3.20 GHz，メモリの容量は 64.0 GB である．PC 上には Python 3.8 より記述されたアプリケーションからハードウェアとの通信を行う．

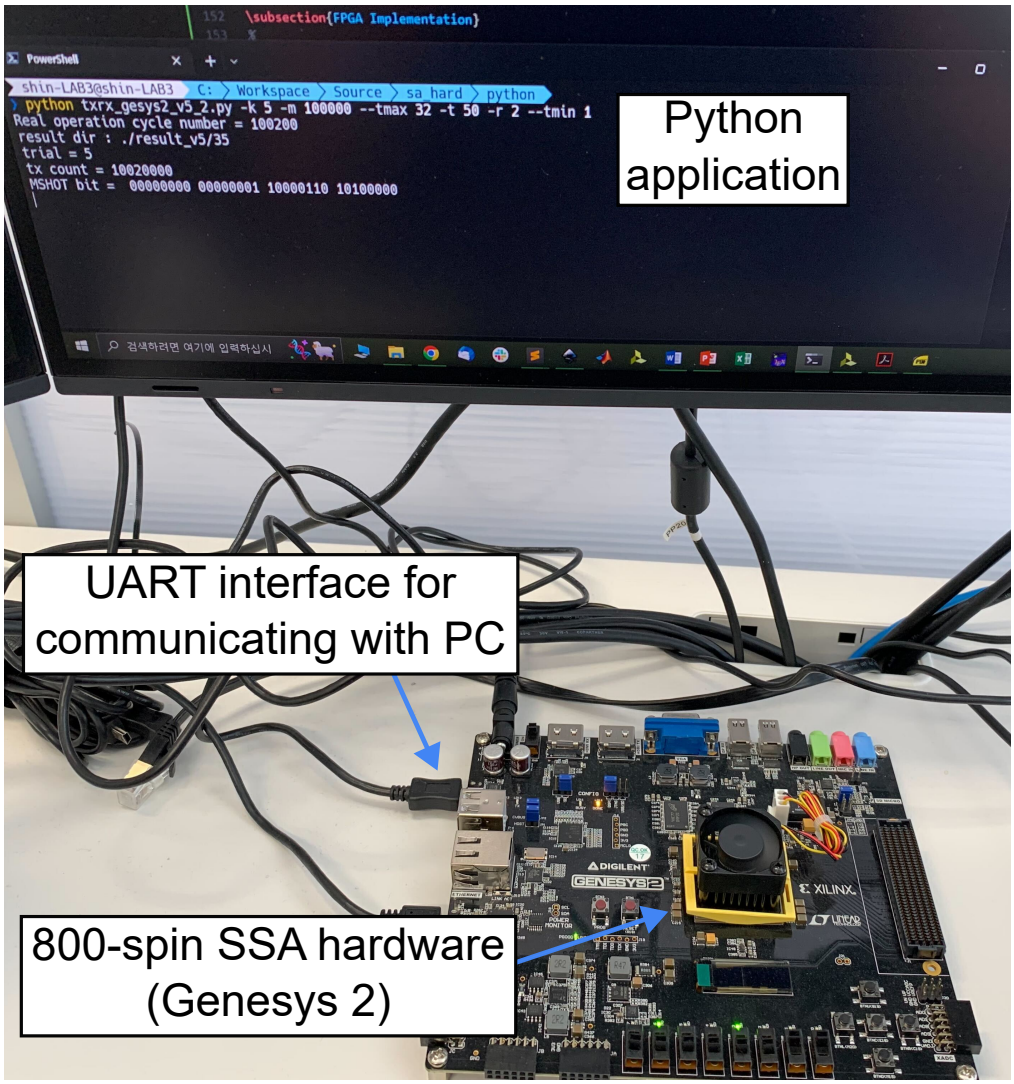


図 3.6: FPGA を用いて実装された 800 スピン SSA ハードウェアの実験環境．実装されたハードウェアは UART インタフェースより PC 上の Python アプリケーションと通信する．

SSA ハードウェアの性能比較として，計算時間やアニーリング精度を従来 SA [10] と比較する．従来 SA 法は Python 3.8 より記述されたソフトウェア実装であり，プログラムを動作させる PC はの CPU は動作周波数 2.1 GHz の Intel Xeon Gold 6252 である．また，従来 SA 法の温度パラメータの制御法は線形関数より線形的に減少させる．従来 SA 法はソフトウェアによる実装であり，提案 SSA ハードウェアは FPGA より実装されているため，実際の計算時間を比較すると，提案ハードウェアが明らかに高速な計算時間を示す．従って，計算時間の比較は実際の計算時間だけではなく，同じアニーリングステップ数におけるイジングエネルギーの収束の傾向を比較する．

提案 SSA ハードウェア及び従来 SA 法の検証には MAXCUT 問題を用いており，SSA 法の動作確認にも使われた G-set を用いる．Table 3.1 は提案 SSA ハードウェアの性能評価に用いる MAXCUT 問題について述べた物である．提案ハードウェアは 800 スピンを搭載しているため，800 ノードからなる G11 問題を提案ハードウェア及び従来 SA 法を用いて結果を確認する．一方，G11 問題は一つのノード

表 3.1: 提案 SSA ハードウェアの性能評価に用いる MAXCUT 問題．King1 問題はランダムに生成された問題であるため，その最良解が未知である．

	G11	King1
# of nodes	800	
# of edges	1,600	3,200
最良解	564	N/A

表 3.2: 800 スピン SSA ハードウェアのハードウェア性能.

動作周波数 [MHz]		100
FPGA リソース使用量	Look up tables (LUTs)	170,035 (83.43%)
	Flip flops (FFs)	14,738 (3.62 %)
	BRAM	356 (80.00 %)
消費電力 [W]		5.532

が隣接の 4 個のノードと接続されている構造を持つ．提案ハードウェアのスピンは 8 個の隣接スピンと接続しているため，ハードウェアのスピン接続構造と同様な構造を持つ MAXCUT 問題として ‘king1’ を生成し，その結果を確認する．King1 問題のエッジの数は 3,200 個であり，エッジの重みは G11 と同様に ‘-1’ または ‘+1’ の値を持つ．また，G11 問題はその最良解が 564 で知られているが，king1 問題はランダムに生成されたものであるため，その最良解が未知である．

3.5　800 スピン SSA ハードウェアの性能評価

FPGA に実装された提案ハードウェアのハードウェア性能を示す．Table 3.2 は 800 スピン 8 接続の SSA ハードウェアの動作周波数，FPGA リソース使用量，消費電力をまとめたものである．提案 SSA ハードウェアの動作周波数は 100 MHz であり，消費電力は 5.532 W である．LUT は Genesys 2 の 83.43%，FF は 3.62% を使用しており，結果を保存するための BRAM は全体の 80% を使用する．

Section 2.4 の SSA 法の動作確認では，G11 問題における最適なハイパーパラメータの組合せを探索した．SSA ハードウェアも同様なハイパーパラメータを使うために，ハイパーパラメータの組合せを探索する必要がある．この際，SSA 法の動作確認で用いた Table 2.2 のハイパーパラメータの組合せは，全て整数であるため，提案 SSA ハードウェアでもそのまま用いることができる．また，Eq. (2.34) から変更さらた提案ハードウェアの逆温度パラメータ制御は Eq. (3.1) のようであり，β のみが変化する．Table 2.2 での $\beta = 0.5$ と同等な動作を行うために $\beta = 1$ とすることで，G11 に対して最適なハイパーパラメータの組合せを決めることができる．

まず，G11 問題を用いて提案 SSA ハードウェアの性能を従来 SA 法と比較する．提案 SSA ハードウェアより G11 問題を 150 shot 分のアニーリング処理で解いた時のイジングエネルギーの変化は Fig. 3.7 のようである．ハイパーパラメータの組合せから 1 shot 当たりのアニーリングステップ数は 600 ステッ

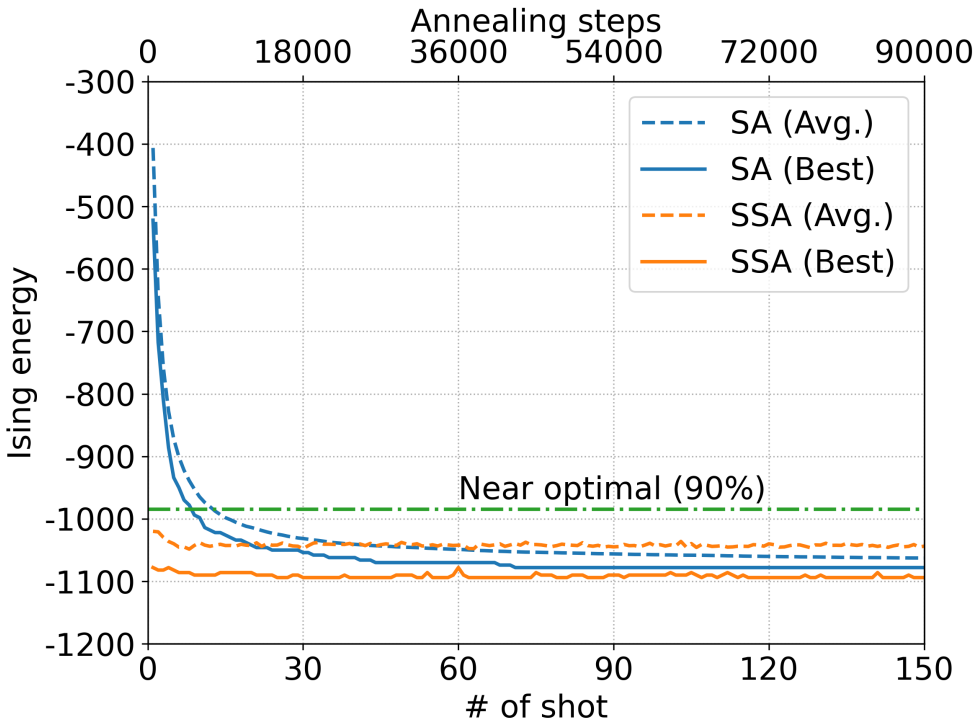
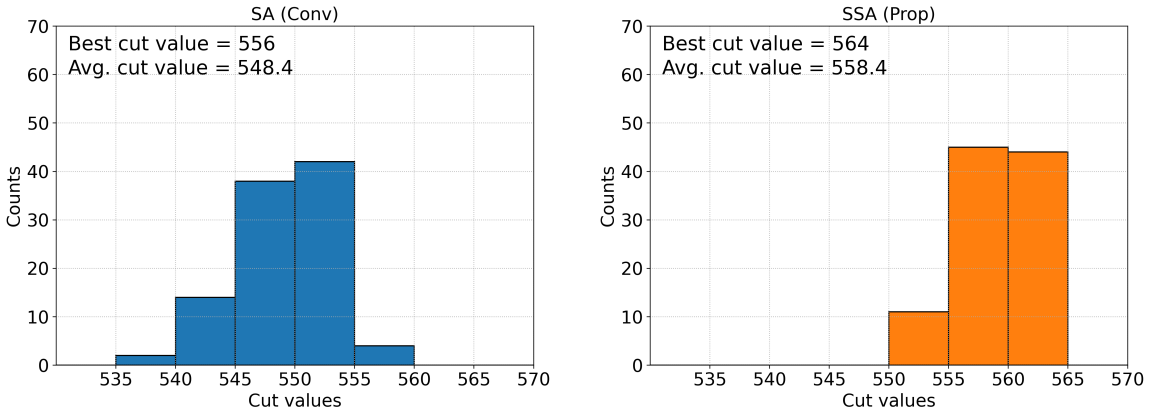


図 3.7: G11 問題におけるアニーリングステップに対する提案 SSA ハードウェア及び従来 SA 法のイジングエネルギーの変化．提案 SSA ハードウェアは逆温度パラメータが最大の時の結果のみを保存するため，shot 当たりの結果を用いる．

プである，全体処理は 90,000 ステップである．また，90,000 ステップの処理を 100 回繰り返して結果を確認した．従来 SA 法も同様に 90,000 ステップのアニーリング処理を 100 回反復した．Fig. 3.7 の平均エネルギー (Avg.) が 100 trial の平均であり，最小値 (Best) は各 shot 及びステップで，100 trial 中の最良解を表す．提案 SSA ハードウェアは最初の shot で既に G11 の最良解の 90% に到達しており，従来 SA 法より高速に近似解に収束することができた．また，90,000 ステップ及びハードウェアの動作周波数から計算した SSA ハードウェアのアニーリング処理の計算時間は 0.9 ms であり，同じステップ数の処理を行う際の従来 SA 法の計算時間は 342.29 s である．

Fig. 3.8a と Fig. 3.8b はそれぞれ従来 SA 法及び提案 SSA ハードウェアの 100 tiral のアニーリング結果をヒストグラムで表したものである．従来 SA 法の 100 trial の最大値は 556 であるが，提案 SSA ハードウェアは G11 の最良解である 564 を見つけることができた．また，100 tiral の平均も提案 SSA ハードウェアは 558.4 であり，548.4 の従来 SA 法より高い精度の解を得ることができた．

続いて，king‘s graph の構造を持つ king1 問題を提案 SSA ハードウェアより解いた結果を確認する．Fig. 3.9 は SSA 法の shot 及びアニーリングステップに対する平均及び最小エネルギーを表す．King1 問題は G11 と同様に 800 ノードの MAXCUT 問題であるため，ハイパーパラメータは G11 の時の組合せと同様なハイパーパラメータを用いた．また，G11 問題とは異なり，king1 問題の最良解は知られていないため，Fig. 3.9 では提案 SSA ハードウェアより得られた最良解を表示する．G11 問題と同様に，



(a)　(b)

図 3.8: G11 問題における 100 trial のアニーリング結果のヒストグラム．(a) 従来 SA 法によるアニーリング結果．(b) 提案 SSA ハードウェアによるアニーリング結果．

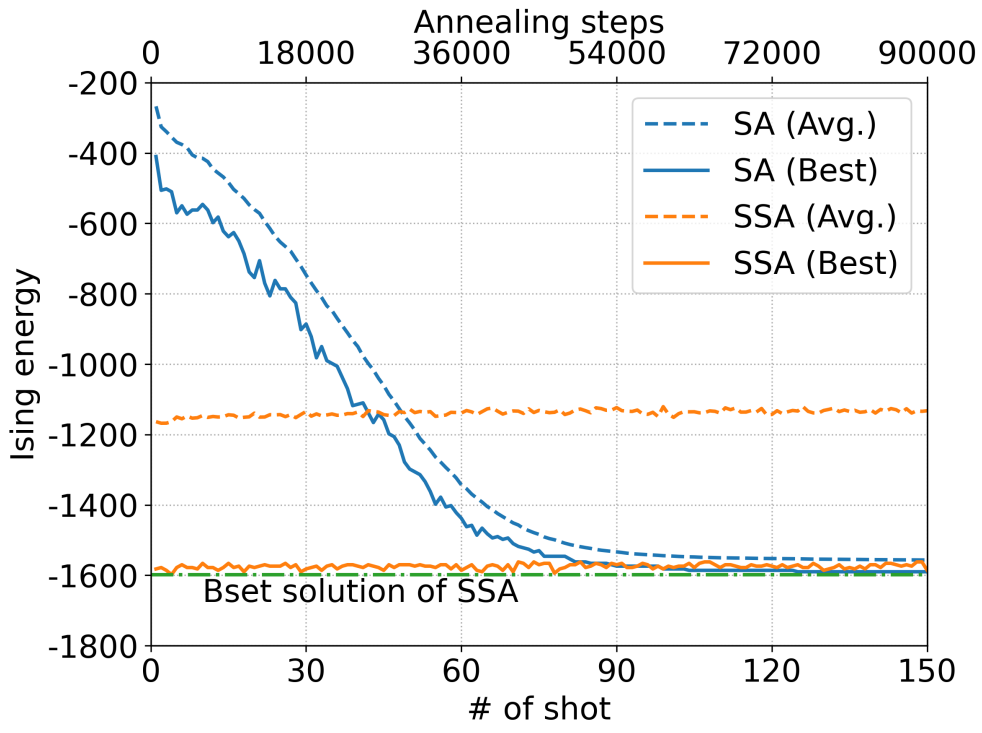
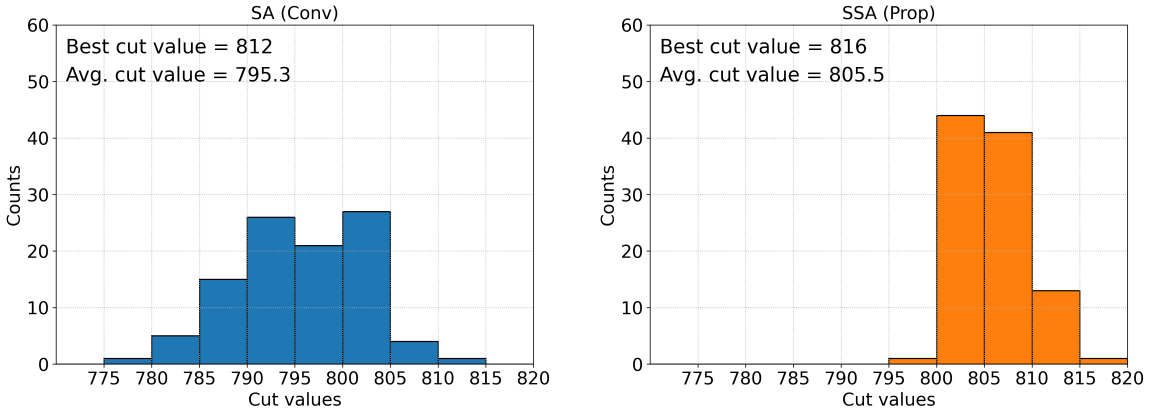


図 3.9: King1 問題におけるアニーリングステップに対する提案 SSA ハードウェア及び従来 SA 法のイジングエネルギーの変化．提案 SSA ハードウェアは逆温度パラメータが最大の時の結果のみを保存するため，shot 当たりの結果を用いる．



(a)　(b)

図 3.10: King1 問題における 100 trial のアニーリング結果のヒストグラム．(a) 従来 SA 法によるアニーリング結果．(b) 提案 SSA ハードウェアによるアニーリング結果．

SSA ハードウェアは 1 shot 目で既に最良解の付近の解を探索できた．King1 問題を解く際のアニーリングステップ数は G11 の場合と同様に 90,000 ステップであるため，提案 SSA ハードウェアの計算時間は 0.9 ms である．同じステップ数のアニーリング処理で，king1 における従来 SA 法の計算時間は 219.14 s であり，提案ハードウェアは二つの問題に対して従来 SA より高速である．一方，従来 SA 法は提案 SSA ハードウェアから得られた最良解に到達するでに，より長いアニーリング処理を必要とする．提案 SSA ハードウェアの 100 trial の平均エネルギーは従来 SA 法よりその精度が低い．この原因んとしては，ハイパーパラメータの組合せが G11 問題に最適なものであり，king1 に足しては最適ではないからである．従って，提案 SSA ハードウェアより高い精度の結果を得るためには適切なハイパーパラメータの組合せを探索する必要がある．

Fig. 3.10a 及び Fig. 3.10b は king1 問題における従来 SA 法及び提案 SSA ハードウェアの結果を表すヒストグラムである．King1 問題においても，提案 SSA ハードウェアは平均で 805.5，最大で 816 のカット値を得られており，平均 795.3，最大 812 の従来 SA 法より高い精度の結果を得られた．

Table 3.3 は提案 SSA ハードウェアの性能を，FPGA ベースの従来アニーリングハードウェア [51] と比較したものである．IPAPT はアニーリングアルゴリズムの一種であるパラレルテンパリング [52, 53] に基づいたアニーリングハードウェアであり，FPGA を用いて実装される [51]．[51] では Xilinx Vertex 5 を用いて，提案 SSA ハードウェアと同じく 800 スピンの G11 問題を解くアニーリングハードウェアを示したため，その結果を提案ハードウェアの結果と比較する．IPAPT の動作周波数は 150 MHz であり，LUT の使用量は 46,753 で提案 SSA ハードウェアより少ない FPGA リソースを使用する．一方，IPAPT のスピン接続は G11 の構造と同様であり，一つのスピンが隣接の四つのスピンと接続する．加えて，スピンの相互作用の重みのビット幅は 2 ビットであり，対応可能な重みの範囲は-1，0，1 のみである．提案 SSA ハードウェアのスピン接続構造は king‘s graph に基づいているため，一つのスピンが隣接つの八つのスピンと接続し，その重みのビット幅は 4 ビットである．従って，提案 SSA ハードウェアは IPAPT より対応可能な問題の範囲がより広い．

表 3.3: 提案 SSA ハードウェアと FPGA に実装された関連研究 [51] との性能比較.

	IPAPT [51] (Conv)	SSA hardware (Prop)
アニーリングアルゴリズム	IPAPT	SSA
FPGA board	Vertex 5	Kintex 7
動作周波数 [MHz]	150	100
LUT 使用率	46,753	170,035
FF 使用率	19,797	14,738
パラメータの表現範囲	$\{-1, 0, 1\}$	$\{-8, -7, ..., 6, 7\}$
スピンの接続数	4	8
G11 問題における平均結果	561	558
G11 問題における最大結果	564	564
100,000 ステップの計算時間 [ms]	2.64	1.00

提案 SSA ハードウェア及び IPAPT を用いて 800 ノードの MAXCUT 問題である G11 問題を解いた結果を比較する．[51] では G11 問題を 100,000 アニーリングステップのアニーリング処理を 1,000 回 (1,000 trial) 反復して，カット値の平均及び最大値を求める．この際の平均カット値は 561，最大カット値は G11 の最良解である 564 であり，この際の通信などの処理時間を除いたアニーリング処理の計算時間は 2.64 ms である．提案 SSA ハードウェアの 100,000 ステップの計算時間は動作周波数及びアニーリングステップ数から 1.00 ms であり，IPAPT より高速にアニーリング処理ができる．また，提案 SSA ハードウェアより得られた平均カット値は 558，最大カット値は IPAPT と同じく 564 であり，提案ハードウェアは 2.64 倍高速で，ほぼ同等な精度を達成できる．

3.6　むすび

本章では SC を基盤とするた SSA アルゴリズムに基づいたアニーリングハードウェアの設計及び実装について述べた．SC より近似されたイジングスピンは CMOS 回路である spin-gate として実装され，一つの spin-gate は隣接の 8 個の spin-gate と接続する構造を持つ．提案 SSA ハードウェアはこのような spin-gate を 800 個搭載しており，800 スピンの king‘s graph の構造を持つイジングモデルに対応可能である．FPGA に実装された提案 SSA ハードウェアを用いて 800 ノードからなる MAXCUT 問題，G11 また king1 問題を解き，その結果を従来 SA 法及び関連研究と比較した．G11 及び king1 問題おいて，提案 SSA ハードウェアは従来 SA 法より高速でより高精度の結果を達成できた．また，同じ数のスピンを搭載した関連研究での FPGA 基盤アニーリングハードウェアと比べ，G11 問題に対して提案ハードウェアは 2.64 倍高速でほぼ同等な精度の結果を得ることができた．

第 4 章

スピン状態の差分を用いた SSA (DSSA) アルゴリズム

4.1 まえかき

本章であ大規模かつ完全接続アニーリングハードウェアを実現されるため，スピン状態の差分を用いた differential stochastic simulated annealing (DSSA) について述べる．まず，Chapter 3 で示したスパースなスピン接続構造を持つ 800 スピン SSA ハードウェアから大規模かつ完全接続アニーリングハードウェア設計の際に生じるハードウェア面積増加の課題から，その課題を克服するためのスピン接続のシリアル化及びパイプライン化について述べる．スピン接続のシリアル化による計算時間増加を抑えるため，スピン状態の差分を用いてスピンの更新を行う DSSA アルゴリズムの動作について詳しく説明する．DSSA アルゴリズムの動作を MAXCUT 問題を用いて確認し，単純シリアル化された場合の SSA 法とその計算時間及びアニーリング精度を比較して DSSA アルゴリズムの有効性を示す．

表 4.1: スピン当たりの接続数に対するハードウェア面積 (LUT 使用量) の変化．用いられた FPGA は Xilinx Kintex 7 である．

# of connection per spin	4	8	799 (Est.)
# of used LUTs	105,294 (51.67%)	170,035 (83.43%)	16,847,040 (8,267.19%)

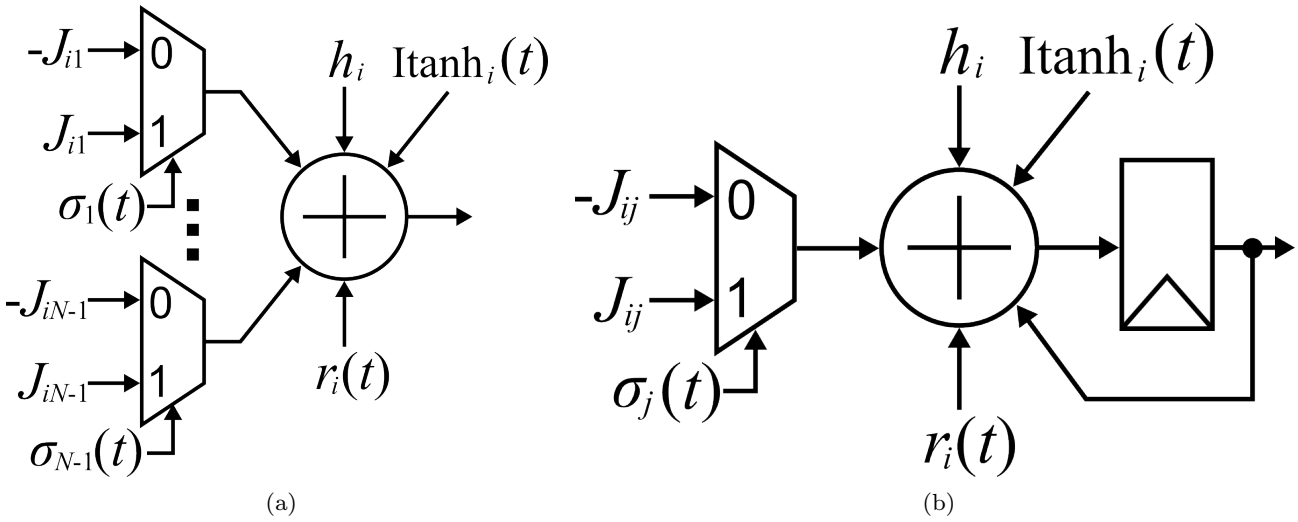


図 4.1: パラレル及びシリアルのスピン接続計算方式における Spin-gate のマルチプレクサの設計．(a) パラレルスピン接続計算のマルチプレクサ設計．(b) シリアルスピン接続計算のマルチプレクサ設計．

4.2　シリアル化による小面積化

Chapter 3 で示した 8 接続 800 スピン SSA ハードウェアのアーキテクチャの基盤として，一つのスピンが隣接の四つのスピンと接続する 800 スピン SSA ハードウェアの LUT 使用量を確認する．4 接続の SSA ハードウェアは一つの spin-gate がスピン相互作用を計算するためのマルチプレクサが四つ含まれている．4 接続及び 8 接続の SSA ハードウェアの LUT 使用量及び 800 スピン完全接続のハードウェアの予想 LUT 使用量を Table 4.1 にまとめる．4 接続から 8 接続に接続数が 2 倍増えるとき，LUT 使用率は 51.67% から 83.43% に約 1.6 倍増加する．この接続数及び LUT 使用量増加から 800 スピン完全接続ハードウェアの面積を推定すると，接続数が 4 接続から約 200 倍増加するため，LUT 使用量は 160 倍増加することになる．4 接続の SSA ハードウェアの LUT 使用率から推定した 800 スピン完全接続ハードウェアの LUT 使用率は 8,267.19% であり，膨大な LUT を必要とする．例え複数の FPGA を用いて設計を行うことにしても，このような面積のハードウェアを実装することは現実的ではない．加えて，推定した完全接続のハードウェアが含むスピン数は 800 個であり，まだ，アプリケーションの側面でも，ハードウェア設計の側面でも大規模とは言えない．このように，大規模かつ完全グラフに対応可能なアニーリングハードウェアは搭載スピン数及び，スピン間の接続の配線または演算回路の面積より，必要とする回路面積が膨大に増加する．

このような回路面積増加の問題を乗り越えるために，一般的に用いられる手法が演算のシリアル化・パラレル化である．Section 4.2 は完全グラフ対応のスピン接続計算実装方式による spin-gate のマルチプレクサの設計を表す回路図である．Fig. 4.1a は Chapter 3 で示した 800 スピン SSA アニーリングハードウェアのスピン接続計算方式であるパラレル実装に対応する spin-gate の設計を表す．ハードウェアに搭載されたスピンの数を N とすると，一つのスピンは他の全てのスピンと相互作用を持つため，スピン当たらいの接続数は $N-1$ である．パラレル実装では，全てのスピン接続の計算を同時に行う必要がらうため，spin-gate でスピン接続の計算を行うマルチプレクサを $N-1$ 個が必要となる．例えば，2,000 スピンの完全グラフ対応可能な SSA ハードウェアを Fig. 4.1a のように実装すると，一つの spin-gate は 1,999 個の spin-gate と接続しており，それぞれの接続に対応するマルチプレクサを 1,999 個含む．さらに，そのハードウェアは 2,000 スピンのイジングモデルに対応するものであるため，1,999 個のマルチプレクサを含むスピンゲートを 2,000 個搭載しなければならない．このように，スピンの完全接続構造のパラレル実装は Table 4.1 のように，スピン数の増加に応じてハードウェア面積が爆発的に増加するため，大規模なハードウェア設計には適していない．このようなハードウェア面積増加を抑えることができる実装方法が，スピン接続の計算を時間分割するシリアル実装である．Fig. 4.1b はスピン接続のシリアル実装に基づいた spin-gate の回路図の一部である．一つの spin-gate にはその接続数に関係なく一つ (または複数) のマルチプレクサを持つ．この際，スピン i におけるスピン相互作用の計算は時間分割され，一つのマルチプレクサを使いまわすことで計算される．例えば，2,000 スピン完全グラフのイジングモデルに対応する SSA ハードウェアの spin-gate は，最初は 0 番目のスピとの相互作用の計算，$\sigma_0 J_{i0}$ を行い，その後 1 番目のスピンとの相互作用を計算する．つまり，自分自身との接続はないため，2,000 スピン完全接続の SSA ハードウェアの spin-gate は一つのマルチプレクサを 1,999 回使いまわすことでスピンの完全接続を実現する．しかし，シリアルに相互作用を計算するため，途中の計算結果を保持しなければならない．シリアル実装の spin-gate では，加算器の後にレジスタを配置し，シリアル化されたスピン相互作用計算の途中結果を保持する．シリアル実装はパラレル実装に比べ，対応可能なイジングモデルのグラフ構造とは関係なく小面積な spin-gate の実装が可能な設計方式である．

スピン相互作用計算のシリアル実装は spin-gate のハードウェア面積を削減できるが，その計算が時間分割されるために，計算時間とのトレードオフになる．Section 4.2 はパラレル実装及びシリアル実装のスピン相互作用計算をハードウェアのクロック信号に対して表すタイミングチャートである．SSA ハードウェアの任意のスピン i において，パラレル実装はスピン i が接続する全てのスピン相互作用に対応するマルチプレクサが spin-gate に含まれるため，全ての相互作用が同時に計算できる．すなわち，Fig. 4.2a のようにスピン i に対する全ての相互作用の計算，$\sigma_j \cdot J_{ij}$，は 1 クロックサイクルで計算できる．一方，spin-gate に一つのマルチプレクサだけ持つシリアル実装の場合，1 クロックサイクルにスピン i に対する相互作用の中で一つの接続のみを計算できる．全てのスピン相互作用を計算するためには，Section 4.2 のように N クロックサイクルが必要である．完全グラフイジングモデルでのスピン当たりの正確な接続数は自分自身との接続はないため $N-1$ である．しかし，大規模 SSA ハードウェアにおいて，不要の計算を除くためのオーバーヘッドが大きくなるため，自分自身との相互作用を 0 とみなして処理を行う．従って，完全グラフイジングモデルに対応するスピン相互作用の計算に必要なクロックサイクルは N である．シリアル実装に基づいた完全接続 2,000 スピン搭載の SSA ハードウェアを例として考える．この場合，1 回スピンの状態を更新 (1 アニーリングステップの計算) するためにはパラレル

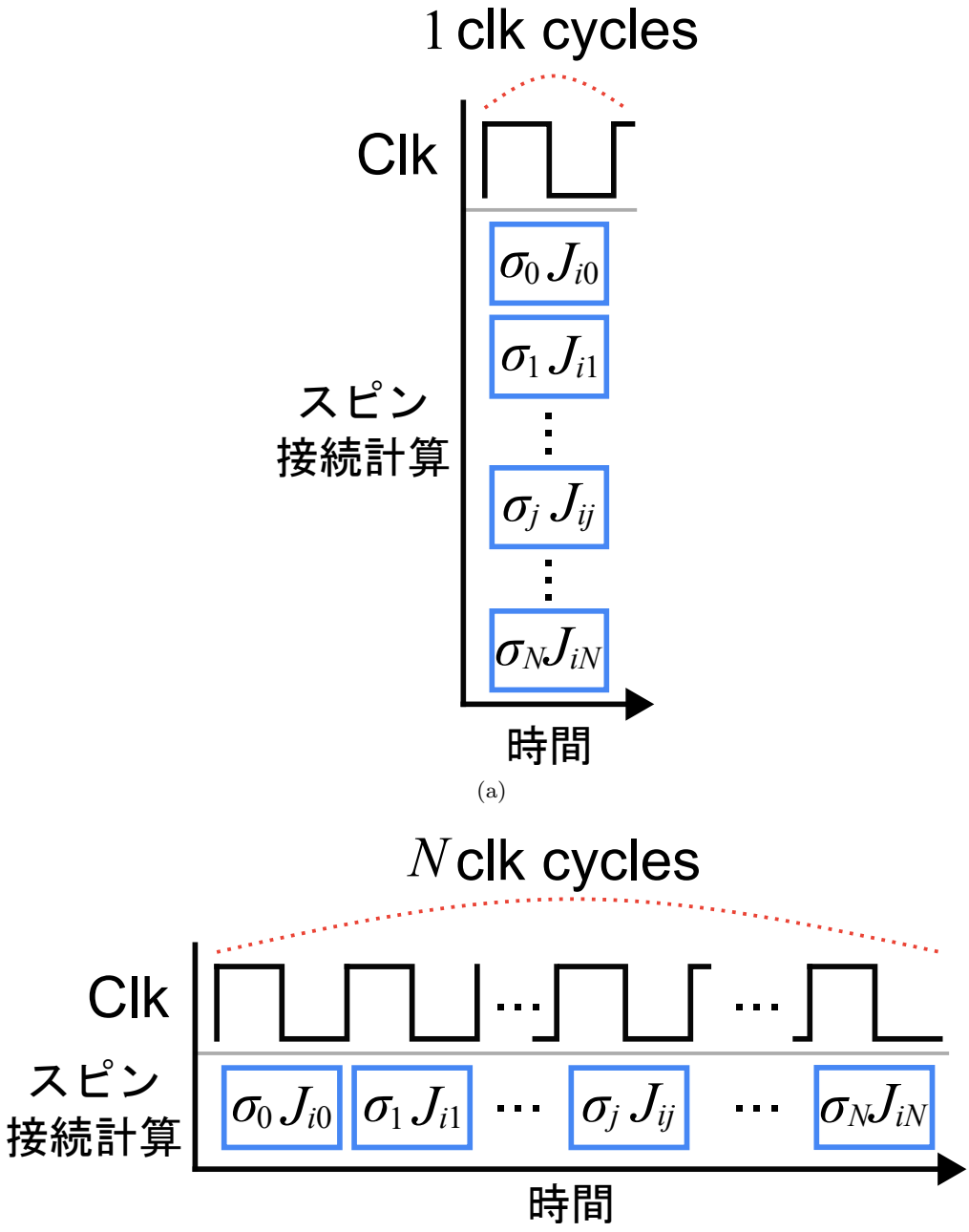


図 4.2: パラレル及びシリアルのスピン接続計算方式における相互作用計算のタイミングチャート．(a) パラレルスピン接続計算のクロック信号に対するタイミングチャート．(b) シリアルスピン接続計算のクロック信号に対するタイミングチャート．

表 4.2: スピン数を N とした時，従来 SSA 法のスピン相互作用のパラレル実装とシリアル実装，DSSA 法の spin-gate のマルチプレクサ個数及び 1 アニーリングステップの計算クロックサイクルの比較.

	パラレル実装	シリアル実装	DSSA
Spin-gate のマルチプレクサの数	N	1	1
クロックサイクル数	1	N	M $(0 \leq M \leq N)$

実装に比べて 2,000 倍のクロックサイクルが必要となる．より多くのスピンを持つ SSA ハードウェアの場合，この計算クロックサイクル数の増加はさらに大きくなる．このように，完全グラフイジングモデルのスピン相互作用計算のシリアル実装は，spin-gate のハードウェア面積を多く削減できるが，その計算が時間分割されているため，アニーリング処理の計算時間が爆発的に増加してしまう．

4.3　DSSA アルゴリズムの動作

Section 4.2 で述べたように，大規模な完全グラフイジングモデル向け SSA ハードウェアの設計において，スピン相互作用計算のパラレル実装はハードウェア面積がスピン数の増加に応じて爆発的に増加するために現実的ではない．また，シリアル実装の場合はスピン数増加に応じたハードウェア面積の増加を抑えることができるため，大規模なアニーリングハードウェアを実装することが可能である．しかし，シリアル実装はスピン相互作用の計算が時間分割されているため，スピン数の増加に応じて計算時間が増加してしまう．大規模な完全グラフイジングモデル向け SSA ハードウェアの実装に向けて，シリアル化されたスピン相互作用の計算から生じる計算時間増加の問題を克服するハードウェアアルゴリズムが differential stochastic simulated annealing (DSSA) 法である．DSSA はスピンの相互作用の計算をシリアル化し，ハードウェアの面積を削減する．さらに，スピン相互作用の計算において，従来 SSA 法のように毎回全てのスピンの相互作用を計算することではなく，スピン状態の差分を用いて相互作用の計算に必要なクロックサイクルを削減する．Table 4.2 は Section 4.2 で述べたスピン相互作用計算のパラレル実装及びシリアル実装，また，提案の DSSA 法の spin-gate に含まれるマルチプレクサの数 (ハードウェア面積) と 1 アニーリングステップにかかるクロックサイクル数の比較である．スピンの数を N とすると，パラレル実装の SSA 法では spin-gate に N 個のマルチプレクサを持つため，1 アニーリングステップの計算を 1 クロックサイクルで行う．シリアル実装はマルチプレクサの数をパラレル実装の $1/N$ に削減できるが，1 アニーリングステップに N クロックサイクルが必要である．提案の DSSA はスピン相互作用の計算をシリアル化しているため，spin-gate のマルチプレクサはシリアル実装と同様にパラレル実装の $1/N$ 倍にできる．ここで，DSSA 法の 1 アニーリングステップの計算に必要なクロックサイクル数は，以前のアニーリングステップで状態が変化したスピン数，M，に比例する．1 アニーリングステップで状態が変化したスピン数は，全くスピンが変化していない場合は 0 であり，最大でも全体のスピン数，N，以下である．Fig. 4.3 は SSA 法の 1shot 間のアニーリングステップに対する SSA 法の逆温度パラメータ及びイジングエネルギーの変化を表す．逆温度パラメータの制御に関わるハイパーパラメータは，$I_{0min} = 1$，$I_{0max} = 16$，$\beta = 1$，$\tau = 50$ である．図からわかるように，逆温度タが低い場合はスピン状態の変化の頻度が高く，イジングエネルギーは急激に減少する．一方，逆温度の増加に連れて

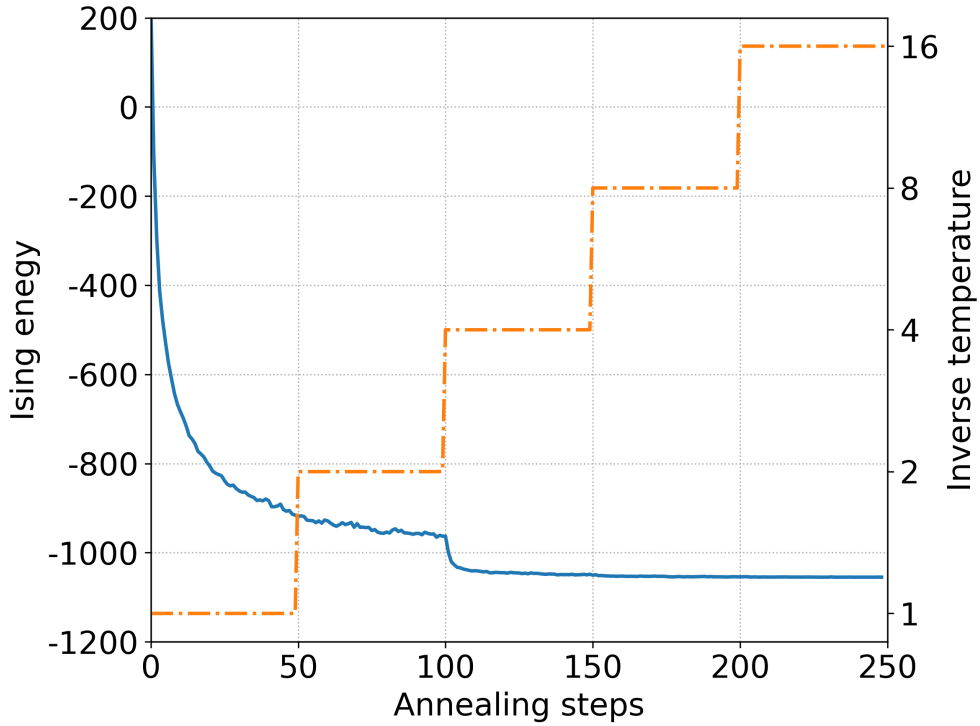


図 4.3: アニーリングステップに対する SSA 法の逆温度パラメータ及びイジングエネルギーの変化．高い逆温度では，スピン状態の変化の頻度が少ないためイジングエネルギーの変化も少ない．

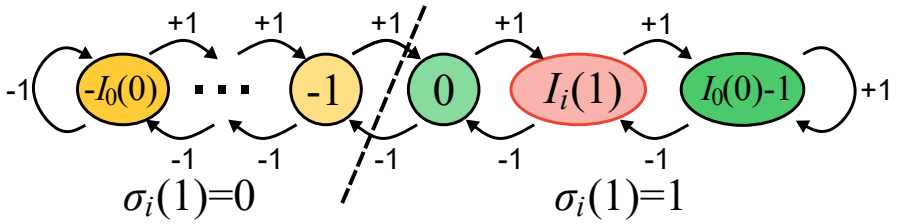


図 4.4: 最初のアニーリングステップの時，DSSA 法の Itanh_i の FSM の状態．

はイジングエネルギー変化は穏やかになり，逆温度が最大に達するとエネルギーはほぼ変化しない．つまり，高い逆温度の場合は状態が変化するスピンが少なくなり，M は比較的に小さい値になる．すなわち，DSSA 法はシリアル実装のようにハードウェア面積を削減しながら，1 アニーリングステップの計算に必要なクロックサイクル数を削減できる．

それでは，スピン状態の差分を用いた提案 DSSA 法の動作について述べる．DSSA の動作は，アニーリングステップに対して，最初のステップとその後のステップでの動作に分けられる．まず，アニーリングステップと t とし，$t = 0$ の時 (最初のステップ) の動作について述べる．以前のアニーリングステップ $(t-1)$ の時に状態が変化したスピンを差分スピンだとすると，DSSA は差分スピンを用いてスピンの状態を更新する．しかし，最初のステップでは，以前の状態がないため，差分スピンを求めることができない．従って，まず最初のスピンの状態を決める必要があり，この際のスピン動作は Eqs. (2.31) to (2.33)

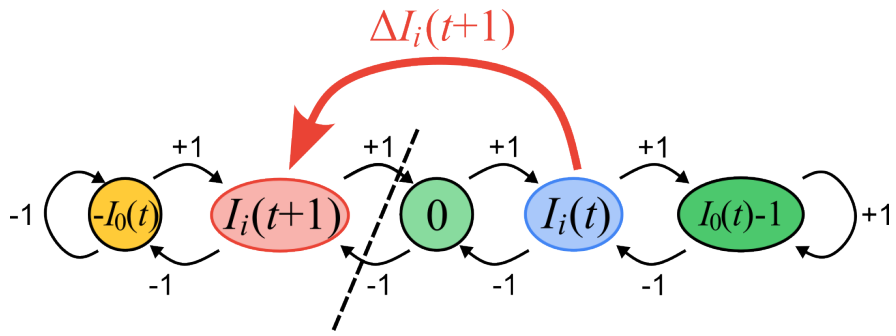


図 4.5: $t > 0$ のアニーリングステップにおける DSSA 法の Itanh_i の FSM の状態変化の動作．以前の状態，$I_i(t)$，から差分スピンのみを用いた FSM の状態の変化量のみを用いて FSM の状態遷移が行われる．

で定義された従来 SSA 法のスピン動作と同様である．つまり，$t = 0$ の時の DSSA 法のスピン動作は:

$$I_i(1) = h_i + \sum_j J_{ij}\sigma_j(0) + n_{rnd} \cdot r_i(0), \tag{4.1a}$$

$$\mathrm{Itanh}_i(1) = \begin{cases} I_0(0) - 1 & \text{if } I_i(1) \geq I_0(0), \\ -I_0(0) & \text{else if } I_i(1) < -I_0(0), \\ I_i(1) & \text{otherwise}, \end{cases} \tag{4.1b}$$

$$\sigma_i(1) = \mathrm{sgn}(\mathrm{Itanh}_i(1)) = \begin{cases} +1 & \text{if } \mathrm{Itanh}_i(1) \geq 0, \\ -1 & \text{otherwise}, \end{cases} \tag{4.1c}$$

のように定義され，ここで，$I_i(0) = 0$ であり，$\sigma_i(0)$ は各スピンの初期値である．ニューラルネットワーク [54] において重みなどのパラメータの初期値は学習速度やモデルの性能に大きな影響を与える [55]．同様にイジングモデルにおける SA 法でもスピンの初期値は最適解への収束時間や解の精度に影響を与える．DSSA 法でもスピンの初期値，$\sigma_i(0)$，はアニーリング精度に影響を与えるはずであり，適切な初期値の選択が更なるアニーリング精度向上につながる可能性がある．一方，適切なスピンの初期値を決めるためにはイジングモデルの重みなどのパラメータを考慮する必要があるため，ハードウェア実装において初期値を決めるためのコストがかかる．従って，提案の DSSA 法の実装においてスピンの初期値は全て '-1' とする．Eq. (4.1b) は Section 2.4 で述べたように，FSM を用いて表すことができる．Fig. 4.4 は $t = 0$ の時の Eq. (4.1b) の動作を FSM より表す．最初のアニーリングステップでは FSM の最初の状態を直接計算する．最初の FSM の状態を決めるためには，スピンが接続している全ての相互作用を計算する必要があるため，スピン数 N に対してスピン相互作用の計算には N クロックサイクルが必要である．

最初のアニーリングステップ ($t = 0$) で求めた Itanh_i 及びスピン状態を用いて，その後のアニーリングステップ ($t > 0$) での差分スピンを用いた DSSA のスピン動作が可能になる．DSSA 法及び SSA 法で，スピンの状態の確率は Itanh_i で表されており，FSM よりその確率とスピンの状態を計算する．従って，スピンの動作を式で定義する前に，差分スピンを用いた DSSA のスピン動作を FSM より考察する．Fig. 4.5 は差分スピンを用いた Itanh_i を計算する際の FSM の動作を表す．従来 SSA 法でが，スピンの状態を更新するたびに，Fig. 4.4 のように FSM の状態を求める．従って，スピンが接続している全て

のスピンとの相互作用を計算しなければならない．一方，DSSA では，FSM の以前の状態 ($I_i(t)$) から，差分スピンによる状態の遷移の変化量 ($\Delta I_i(t+1)$) のみを計算し，新しい FSM の状態 ($I_i(t+1)$) を決める．新しい FSM の状態が決まると，$\mathrm{Itanh}_i(t+1)$，の値を計算できるため，従来の SSA 法と同様に，Eq. (2.32) に従って，スピンの状態を求めることができる．すなわち，DSSA 法に基づいた $t > 0$ のアニーリングステップでは，差分スピンを用いて FSM の状態遷移の量，$\Delta I_i(t+1)$ を求めることで，状態が変化していないスピンとの相互作用の計算を無視できる．差分スピンの個数，M，は全体のスピン数 N より必ず少ないため，$M \leq N$ であり，シリアル化された従来 SSA 法より少ない M クロックサイクルでスピンの動作が可能である．

Fig. 4.5 での差分スピンによる FSM の状態遷移，$\Delta I_i(t+1)$ は以下のように定義できる:

$$\Delta I_i(t+1) = \mathrm{Itanh}_i(t+1) - \mathrm{Itanh}_i(t). \tag{4.2}$$

ここで，$\mathrm{Itanh}_i(t)$ は Eq. (2.33) に従って I_i によりその値が決まるため，$\mathrm{Itanh}_i(t) \approx I_i(t)$ と見なして式を簡単に表すと，$\Delta I_i(t+1)$ は:

$$\begin{aligned}\Delta I_i(t+1) &= I_i(t+1) - I_i(t) \\ &= h_i + \sum_j J_{ij}\sigma_j(t) + n_{rnd}\cdot r_i(t) - (h_i + \sum_j J_{ij}\sigma_j(t-1) + n_{rnd}\cdot r_i(t-1)) \\ &= \sum_j J_{ij}(\sigma_j(t) - \sigma_j(t-1)) + n_{rnd}(r_i(t) - r_i(t-1)),\end{aligned} \tag{4.3}$$

である．ここで, Eq. (4.3) の $r_i(t) - r_i(t-1)$ は-1 か +1 の乱数であり，この項を $r'_i(t) = r_i(t) - r_i(t-1)$ と置き換えれば Eq. (4.3) は:

$$\Delta I_i(t+1) = \sum_j J_{ij}(\sigma_j(t) - \sigma_j(t-1)) + n_{rnd}\cdot r'_i(t), \tag{4.4}$$

のように表すことができる．ここで，$\Delta_j(t)$ を $\sigma_j(t) \neq \sigma_j(t-1)$ の場合には 1，$\sigma_j(t) = \sigma_j(t-1)$ の場合には 0 とする差分スピンのフラグとすると，$\Delta_j(t)$ は:

$$\Delta_j(t) = \begin{cases} 1 & \text{if } \sigma_j(t) \neq \sigma_j(t-1), \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

である．$\Delta_j(t)$ は以前のステップからスピン状態が変化していなければ，0 になるため，状態が変化していないスピンは $\Delta I_i(t+1)$ に影響しない．すなわち，スピンの状態が変化した差分スピンとの相互作用を計算するだけで，Eq. (4.3) の計算が可能になる．また，$\sigma_j(t) \neq \sigma_j(t-1)$ の場合，$\sigma_j(t) - \sigma_j(t-1)$ は:

$$\sigma_j(t) - \sigma_j(t-1) = \begin{cases} 2 & \text{if } \sigma_j(t) = 1, \\ -2 & \text{elseif } \sigma_j(t) = -1, \end{cases} (\text{When } \sigma_j(t) \neq \sigma_j(t-1)), \tag{4.6}$$

のようであり，$\sigma_j(t) - \sigma_j(t-1)$ の符号は $\sigma_j(t)$ より求めることができため，Eq. (4.5) 及び Eq. (4.6) から $\sigma_j(t) - \sigma_j(t-1)$ は:

$$\sigma_j(t) - \sigma_j(t-1) = 2\Delta_j(t)\cdot\sigma_j(t), \tag{4.7}$$

であり，Eq. (4.7) を Eq. (4.4) に代入すると:

$$\Delta I_i(t+1) = 2\sum_j J_{ij}\Delta_j(t)\cdot\sigma_j(t) + +n_{rnd}\cdot r'_i(t), \tag{4.8}$$

となる．Eq. (4.8) を用いて，Eq. (4.3) をアニーリングステップ t の時，求めたい $I_i(t+1)$ に対して整理すると:

$$\begin{aligned} I_i(t+1) &= I_i(t) + \Delta I_i(t+1), \\ &= I_i(t) + 2\sum_j J_{ij}\Delta_j(t)\cdot\sigma_j(t) + +n_{rnd}\cdot r'_i(t), \end{aligned} \tag{4.9}$$

であり，Eq. (4.9) の $I_i(t)$ を，元の $\mathrm{Itanh}_i(t)$ に戻すと，Eq. (4.9) は:

$$I_i(t+1) = \mathrm{Itanh}_i(t) + 2\sum_j J_{ij}\Delta_j(t)\cdot\sigma_j(t) + n_{rnd}\cdot r'_i(t), \tag{4.10}$$

であるため，SSA 法のスピン状態の確率 $I_i(t+1)$ を差分スピンのみで計算できる．すなわち，$t>0$ のアニーリングステップにおいて，DSSA 法のスピン動作は:

$$\Delta I_i(t+1) = 2\sum_j J_{ij}\Delta_j(t)\cdot\sigma_j(t) + n_{rnd}\cdot r_i(t), \tag{4.11a}$$

$$\mathrm{Itanh}_i(t+1) = \begin{cases} I_0(t)-1 & \text{if } \mathrm{Itanh}_i(t)+\Delta I_i(t+1) \geq I_0(t), \\ -I_0(t) & \text{else if } \mathrm{Itanh}_i(t)+\Delta I_i(t+1) < -I_0(t), \\ \mathrm{Itanh}_i(t)+\Delta I_i(t+1) & \text{otherwise}, \end{cases} \tag{4.11b}$$

$$\sigma_i(t+1) = \begin{cases} 1 & \text{if } \mathrm{Itanh}_i(t+1) \geq 0, \\ -1 & \text{otherwise}, \end{cases} \tag{4.11c}$$

でで意義できる．このように DSSA 法は従来 SSA 法のスピン動作と同様な動作を，状態が変化した差分スピンとの相互作用のみを用いることでスピンの状態を更新でき，より高速にシリアル化されたスピン相互作用の計算を可能とする．

DSSA 法は SSA 法を基盤としているため，逆温度パラメータに応じてスピンの状態変化が制御される．SSA 法と同様に，逆温度パラメータ，$I_0(t)$ は Table 2.1 のハイパーパラメータより制御され，逆温度の制御はハードウェア実装に向けて Eq. (3.1) に基づいて行われる．また，逆温度パラメータが I_{0min} から I_{0max} までに増加する間を 1 shot というアニーリング処理を構成する単位とすることも従来 SSA 法と同様である．

4.4　DSSA アルゴリズムの性能評価

DSSA 法のハードウェア実装に向けて，アルゴリズムの動作を検証し，その性能を確認する．DSSA 法の動作検証はソフトウェアに基づいたシミュレーションより行われる．シミュレーションに用いられる CPU は Intel Xeon Gold 5318Y で CPU の動作周波数は 2.1 GHz であり，使用 OS は Rocky Linux 8.6 である．DSSA のシミュレーションは Python 3.8 より実装されたソフトウェアである．

DSSA の動作検証には COP の一種である MAXCUT 問題を用いる [23]．MAXCUT 問題のデータセットは SSA 法の動作確認でも用いた G-set [24] であり，G-set のいくつかのベンチマークインスタンスを用いて DSSA の動作を確認する．DSSA 法は大規な模完全グラフイジングモデル向けアニーリングプロセッサの実装のために提案された物であるため，アニーリングハードウェアに関する関連研究 [26, 29] から，ハードウェア実装を考慮した大規模なイジングモデルのスピン数は 2,000 としている．

表 4.3: DSSA 法の動作検証に用いら MAXCUT 問題の詳細.

問題	ノードの数	エッジの数	グラフ構造	重み	最良解
G22	2,000	19,990	ランダム	{+1}	13,359
G23					13,344
G24					13,337
G27				{−1, +1}	3,341
G35		11,778	平面	{+1}	7,687
G39				{−1, +1}	2,408
K2000		1,999,000	完全	{−1, +1}	33,337

表 4.4: DSSA の動作検証におけるハイパーパラメータ探索の探索範囲.

I_{0min}	$2^0,\ 2^1,\ 2^2$
I_{0max}	$2^5,\ 2^6,\ 2^7,\ 2^8,\ 2^9,\ 2^{10}$
n_{rnd}	$1,\ 2,\ 3,\ \ldots, 31,\ 32$
τ	$100,\ 200,\ 300,\ \ldots, 500,\ 600$

従って，DSSA の検証に用いる G-set のインスタンスは 2,000 ノードの問題である，G22，G23，G24，G27，G35，G39 である．それぞれの問題インスタンスはあ全て 2,000 ノードの MAXCUT 問題であり，G22，G23，G24，G27 の構造はノード間の接続がランダムに決まったランダムグラフでエッジの数は 19,990 である．また，G35 及び G39 は平面グラフであり，エッジの数は 11,778 である．G22，G23，G24，G35 のエッジの重みは 1 のみであり，G27，G39 の重みは-1 また +1 である．G-set の問題は様々な構造を持つが，全てのノードが互いに接続する完全グラフの問題は含まれていない．完全グラフのイジングモデルにおける DSSA の性能を検証するために，G-set に加えて 2,000 ノードから完全グラフ MAXCUT 問題である K2000 問題を用いる [30]．K2000 問題は 2,000 ノードからなる完全グラフ MAXCUT 問題であり，-1 または +1 の重みを持つエッジの数は 1,999,000 である．検証に用いられる問題の最良解は全てに知られており，それぞれの問題の最良解を含めた詳細は Table 4.3 のようである．

DSSA 法を用いて Table 4.3 の問題を解く前に，高精度の解を得るためには，それぞれの問題において最適なハイパーパラメータの組合せを探索する必要がある．前述したように，DSSA のハイパーパラメータは従来 SSA 法と同様であり，Table 2.1 のようである．ハイパーパラメータの探索は，それぞれのハイパーパラメータをある範囲で変えながら短いアニーリング処理を行い，そのアニーリング結果より探索するグリッドサーチを行う．Table 4.4 はハイパーパラメータのグリッドサーチにおける各ハイパーパラメータの探索範囲を表す．I_{0min} は 1 から 4，I_{0max} は 2^5 から 2^10 の範囲で 2 のべき乗を探索範囲とする．n_{rnd} は 1 から 32 の整数を範囲とし，τ は 100 から 600 まで 100 ずつ増やしなら探索を行う．逆温度，I_0 の増加率である β は DSSA 法及び SSA 法の解の精度や収束速度に影響を与えるハイパーパラメータでが，ハイパーパラメータの探索時間を削減するために，本実験では 1 で固定する．ハイパー

表 4.5: G22 問題に対するハイパーパラメータの探索結果．平均解を基準で top5 の結果をまとめた．

順位	n_{rnd}	I_{0min}	I_{0max}	τ	β	Best	Best[1][%]	Avg.	Avg.[2][%]
Top 1	7	2	512	600	1	13,347	99.91	13,331.2	99.79
Top 2	7	1	512	500	1	13,349	99.93	13,330.0	99.78
Top 3	7	4	256	600	1	13,347	99.91	13,329.4	99.78
Top 4	7	2	64	600	1	13,347	99.91	13,329.2	99.78
Top 5	7	2	128	500	1	13,347	99.91	13,329.0	99.78

[1] 最良解に対する最大解の比

[2] 最良解に対する解の平均の比

表 4.6: G23 問題に対するハイパーパラメータの探索結果．平均解を基準で top5 の結果をまとめた．

順位	n_{rnd}	I_{0min}	I_{0max}	τ	β	Best	Best[1][%]	Avg.	Avg.[2][%]
Top 1	4	4	2,048	600	1	13,323	99.84	13,318.2	99.81
Top 2	4	4	64	600	1	13,337	99.95	13,318.0	99.81
Top 3	7	1	128	600	1	13,323	99.84	13,316.8	99.80
Top 4	4	4	256	600	1	13,327	99.87	13,316.2	99.79
Top 5	7	4	128	600	1	13,324	99.85	13,315.8	99.69

[1] 最良解に対する最大解の比

[2] 最良解に対する解の平均の比

パラメータの探索は Table 4.4 の範囲のハイパーパラメータの組合せにおいて，短いアニーリング処理を行って得られた解の精度を比較する．短いアニーリング処理は DSSA の処理の単位である shot を基準として，それぞれの組合せに対して 5 shot のアニーリング処理を行う．5 shot のアニーリング処理の実際のアニーリングステップ数は，ハイパーパラメータの組合せに応じて異なるが，逆温度パラメータが最初から最大に達することを DSSA 法においての完全なアニーリング処理と考え，shot を単位として探索を行う．また，DSSA は確率的アルゴリズムであるため，1 回の試行だけではそのハイパーパラメータの組合せの良さを正しく評価できない．従って，それぞれのハイパーパラメータの組合せに対して 5 shot のアニーリング処理を 5 回繰り返し (5 trials)，その際の平均及び最大の結果を比較する．

まず，G22 問題に対するハイパーパラメータの探索結果を Table 4.5 にまとめる．Table 4.5 は G22 問題に対して Table 4.4 の範囲で行ったハイパーパラメータ探索結果から，5 trials の平均解を基準で top 5 の組合せとその組合せにおけるアニーリング結果を示す．Table 4.5 からわたるように，G22 問題に対して平均解が最大になるハイパーパラメータの組合せは top 1 のようである．しかし，最大解に関しては top 2 の組合せがより高い解を探索でき平均解の差は 1.2 でありその差は非常に少ない．また，Eq. (3.2)

表 4.7: G24 問題に対するハイパーパラメータの探索結果．平均解を基準で top5 の結果をまとめた．

順位	n_{rnd}	I_{0min}	I_{0max}	τ	β	Best	Best[1][%]	Avg.	Avg.[2][%]
Top 1	7	2	512	600	1	13,316	99.84	13,304.8	99.76
Top 2	7	1	256	600	1	13,316	99.84	13,304.6	99.76
Top 3	7	1	512	600	1	13,312	99.81	13,304.2	99.75
Top 4	4	4	2,048	500	1	13,318	99.86	13,303.8	99.75
Top 5	7	2	2,048	500	1	13,309	99.79	13,303.2	99.75

[1] 最良解に対する最大解の比

[2] 最良解に対する解の平均の比

表 4.8: G27 問題に対するハイパーパラメータの探索結果．平均解を基準で top5 の結果をまとめた．

順位	n_{rnd}	I_{0min}	I_{0max}	τ	β	Best	Best[1][%]	Avg.	Avg.[2][%]
Top 1	3	1	64	600	1	3,334	99.79	3,329.0	99.64
Top 2	3	1	32	300	1	3,334	99.79	3,328.8	99.64
Top 3	3	1	32	500	1	3,332	99.73	3,327.2	99.59
Top 4	4	1	64	400	1	3,333	99.76	3,326.2	99.56
Top 5	4	1	512	600	1	3,332	99.73	3,326.0	99.55

[1] 最良解に対する最大解の比

[2] 最良解に対する解の平均の比

表 4.9: G35 問題に対するハイパーパラメータの探索結果．平均解を基準で top5 の結果をまとめた．

順位	n_{rnd}	I_{0min}	I_{0max}	τ	β	Best	Best[1][%]	Avg.	Avg.[2][%]
Top 1	3	4	32	500	1	7,651	99.53	7,640.4	99.39
Top 2	3	4	512	600	1	7,645	99.45	7,639.0	99.38
Top 3	3	4	128	500	1	7,655	99.58	7,638.6	99.37
Top 4	3	4	64	600	1	7,648	99.49	7,637.2	99.35
Top 5	3	4	512	500	1	7,645	99.45	7,636.6	99.34

[1] 最良解に対する最大解の比

[2] 最良解に対する解の平均の比

表 4.10: G39 問題に対するハイパーパラメータの探索結果．平均解を基準で top5 の結果をまとめた．

順位	n_{rnd}	I_{0min}	I_{0max}	τ	β	Best	Best[1][%]	Avg.	Avg.[2][%]
Top 1	3	2	256	600	1	2,394	99.42	2,387.2	99.14
Top 2	3	2	64	500	1	2,393	99.38	2,386.6	99.11
Top 3	3	1	256	400	1	2,393	99.38	2,386.0	99.09
Top 4	2	1	128	500	1	2,392	99.34	2,385.6	99.07
Top 5	5	2	32	300	1	2,392	99.34	2,385.4	99.06

[1] 最良解に対する最大解の比

[2] 最良解に対する解の平均の比

表 4.11: K2000 問題に対するハイパーパラメータの探索結果．平均解を基準で top5 の結果をまとめた．

順位	n_{rnd}	I_{0min}	I_{0max}	τ	β	Best	Best[1][%]	Avg.	Avg.[2][%]
Top 1	31	1	512	600	1	33,323	99.96	33,296.6	99.80
Top 2	32	1	2,048	600	1	33,294	99.87	33,264.6	99.78
Top 3	31	1	256	600	1	33,296	99.88	33,264.4	99.78
Top 4	31	2	512	500	1	33,283	99.84	33,264.2	99.78
Top 5	26	1	128	500	1	33,307	99.91	33,261.8	99.77

[1] 最良解に対する最大解の比

[2] 最良解に対する解の平均の比

より求めた DSSA 法の処理の単位である shot 当たりのアニーリングステップ数は top 1 は 5,400 ステップ，top 2 は 5,000 ステップであり，top 2 がより高速に近似解を探索できた．従って，G22 問題に対して最も最適な DSSA 法のハイパーパラメータの組合せは top 1 の組合せではなく，top 2 だと考えられる．同様に，他のベンチマーク問題に対するハイパーパラメータの探索結果の平均解を基準とした top 5 の結果をそれぞれ Tables 4.6 to 4.11 にまとめる．G22 問題と同様に，G23 と G24 問題に対しては top 1 と top 2 の組合せがほぼ同等な最大解及び平均解を達成した．一方，shot 当たりのアニーリングステップ数は G23 問題の場合，top 1 が 6,000 ステップ，top 2 が 3,000 ステップであり，top 2 の組合せがより高速であるため，top 2 を最適な組合せとする．G24 の場合は top 1 と top 2 両方とも shot 当たり 5,400 ステップであるが，逆温度パラメータの最初値及び最大値がより低い top 2 を最適な組合せとする．同様に G27 問題に対しても，top 1 と top 2 の最大解及び平均解の差は少ないが，top 2 の shot 当たりステップ数が 3,600 ステップであるため，top 2 を最適な組合せとする．G35 問題に対しては top 1 が最も高い精度の解を得た上で，shot 当たりステップ数もより少ないため，top 1 を最適な組合せとする．一方，G39 問題は top 2 の組合せが top 1 に比べ，shot 当たりのステップ数が小さいが，最大解が

表 4.12: K2000 問題に対するハイパーパラメータの探索結果．最大解を基準で top5 の結果をまとめた．

順位	n_{rnd}	I_{0min}	I_{0max}	τ	β	Best	Best[1][%]	Avg.	Avg.[2][%]
Top 1	24	1	1,024	600	1	33,337	100.0	33,240.8	99.71
Top 2	21	1	64	600	1	33,337	100.0	33,186.0	99.55
Top 3	16	1	128	400	1	33,337	100.0	33,090.0	99.26
Top 4	32	1	512	600	1	33,335	99.99	33,259.2	99.77
Top 5	25	2	2,048	600	1	33,335	99.99	33,188.0	99.55

[1] 最良解に対する最大解の比

[2] 最良解に対する解の平均の比

表 4.13: ベンチマーク問題に対する DSSA の最適なハイパーパラメータの組合せとそのアニーリング結果．

問題	n_{rnd}	I_{0min}	I_{0max}	τ	β	Best	Best[1] [%]	Avg.	Avg.[2] [%]
G22	7	1	512	500	1	13,349	99.93	13,330.0	99.78
G23	4	4	64	600	1	13,337	99.95	13,318.0	99.81
G24	7	1	256	600	1	13,316	99.84	13,304.6	99.76
G27	3	1	32	300	1	3,334	99.79	3,328.8	99.64
	3	1	2,048	300	1	3,341	100.0	3,321.8	99.42
G35	3	4	32	500	1	7,651	99.53	7,640.4	99.39
G39	3	2	256	600	1	2,394	99.42	2387.2	99.14
K2000	24	1	1,024	600	1	33,337	100.0	33,240.8	99.71

[1] 最良解に対する最大解の比

[2] 最良解に対する解の平均の比

より高精度であため，最適なハイパーパラメータの組合せは top 1 とする．

Table 4.11 にまとめている K2000 における五つの組合せはどれでも再了解の 99% を超える最大解及び平均解を得られた．しかし，K2000 問題は従来 SSA 法において，その最良解を既に得られている [30]．従って，ハイパーパラメータの探索結果を平均解ではなく，最大解を基準として top 5 の組合せをまとめると，Table 4.12 のようである．5 shots の短いアニーリング処理でも DSSA 法は K2000 問題の最良解を達成でき，Table 4.12 での top 1 の組合せは高い精度の平均解も達成できたため，K2000 においての最適なハイパーパラメータの組合せは Table 4.12 の top 1 の組合せとする．加えて，G27 問題も最大解を基準として，ハイパーパラメータの探索結果をまとめると，$n_{rnd} = 3$，$I_{0min} = 1$，$I_{0max} = 2,048$，$\tau = 300$ の組合せで得られた最適解が G27 の最良解であった．従って，G27 に関しては Table 4.8 の top 2 に加えて，最良解を達成した組合せも含めて動作確認を行う．

以上の DSSA 法のハイパーパラメータ探索の結果から，それぞれのベンチマーク問題における最適なハイパーパラメータの組合せを Table 4.13 でまとめる．Section 2.4 及び Chapter 3 では性能評価として 800 ノードの MAXCUT 問題である G11 を用いた．加えて，問題のグラフ構造などが異なる 800 ノードの問題も用いたが，その際のハイパーパラメータの組合せは，ノード数が 800 で G11 と等しいため，G11 に対して最適なハイパーパラメータの組合せをそのまま用いた．しかし，Table 4.13 のように，問題のノード数が等しくても，そのノード間の重みや問題のグラフ構造によって，最適なハイパーパラメータの組合せは異なるため，DSSA 法を用いて高い精度のアニーリング結果を得るためには最適なハイパーパラメータの組合せの探索が必須不可欠である．以降の DSSA 法の動作検証及びシリアル実装された従来 SSA 法とのスピン相互作用計算のクロックサイクル数の比較では Table 4.13 のハイパーパラメータの組合せを用いる．

探索さいたハイパーパラメータの組合せを用いて DSSA 法のアニーリング動作を確認する．動作検証は，5 shots を 1 trial としてそれぞれのベンチマークに対して 100 trials を行い，そのアニーリング結果を評価する．まず，G22 問題において DSSA 法を用いた場合，アニーリングステップに対しての 100 trials の平均カット値は Fig. 4.6 のようである．Fig. 4.6 からわかるように，DSSA 法は従来 SSA 法のように逆温度パラメータに応じて近似解へ収束する．また，1 shot の処理が終了し，逆温度が初期化されるときに，再び低い解の状態に戻り解の探索を行う．すなわち，差分スピンのみを用いてスピンの状態を更新する DSSA 法は従来 SSA 法のような動作が可能であり，正しくアニーリング処理を行うと考えられる．加えて，DSSA 法は G22 問題において 1 shot 目のアニーリング処理より，最良解の 99% の近

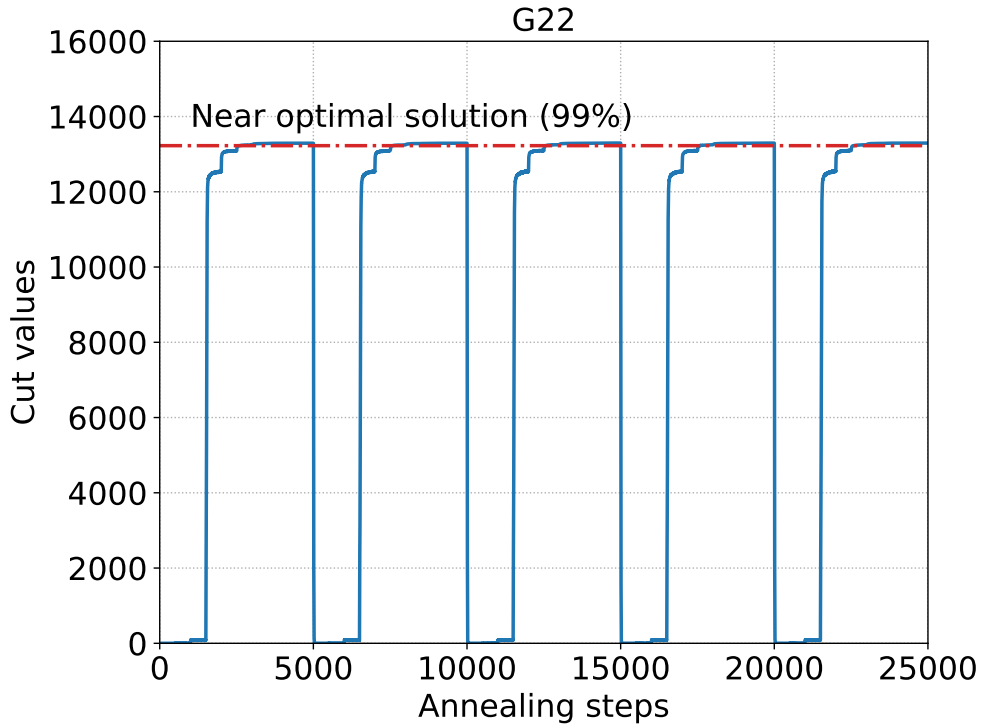


図 4.6: G22 問題において DSSA 法によるアニーリングステップに対しての平均カット値.

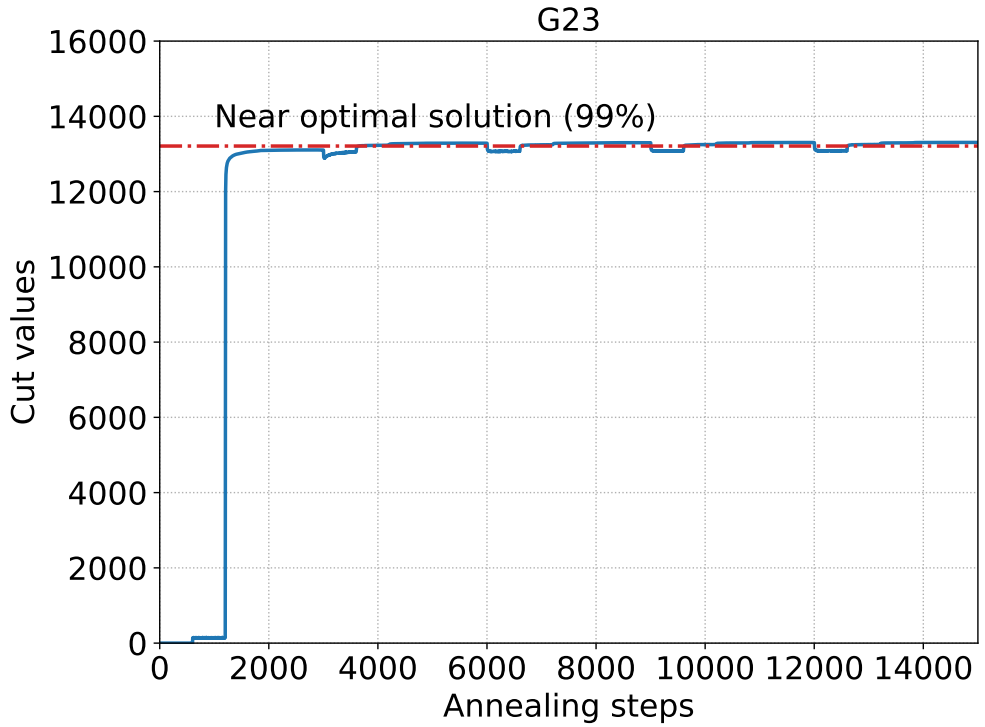


図 4.7: G23 問題において DSSA 法によるアニーリングステップに対しての平均カット値.

似解を得られ，高速に近似解を得ることが可能である．

同様にそれぞれのベンチマーク問題において，アニーリングステップに対する 100 trials の平均カット値を Figs. 4.7 to 4.12 で示す．どの問題に対しても DSSA 法は最良解の 99% の平均近似解を達成することができた．

続いて，それぞれのベンチマーク問題に対して，各 trial で得られた最大解をヒストグラムで表すと，Figs. 4.13 to 4.19 のようである．また，それぞれの図には対応する問題の最良解の 99% の近似解も表しており，G22，G23，G24，K2000 問題では全ての trials で 99% の近似解より高い解を得ることができた．G27 問題の 100 trials の結果では最良解の 99% の近似解より高い解を得られる確率は 78%，G35 問題は 99%，G39 問題は 42% であり，G39 問題以外では 100 trials の半分以上の試行で最適解に近い近似解を達成した．それぞれの問題に対して 100 trials の最小，平均，最大解を Table 4.14 でまとめる．G22，G23，G24 問題に対いて DSSA 法は最大で最良解の 99.9% の近似解を達成でき，G35 と G39 問題では最良解の 99.5% の近似解を達成した．また，K2000 問題及び G27 問題ではそれぞれの問題の最良解を得ることができた．G39 問題以外の全ての問題においては平均でも 99% 以上の近似解に到達し，G22，G23，G24，K2000 問題では最小解でも 99% 以上の解を得た．このように DSSA 法は MAXCUT 問題に対して，短いアニーリング処理で高精度の近似解を達成でき，2,000 ノードからなる大規模なイジングモデルにおいて高速かつ高精度なアニーリングアルゴリズムである．

それでは，スピン相互作用の計算がシリアル化された従来 SSA 法と，差分スピンのみを用いてスピン相互作用を計算する DSSA 法において，スピン相互作用の計算にかかるハードウェアのクロックサイク

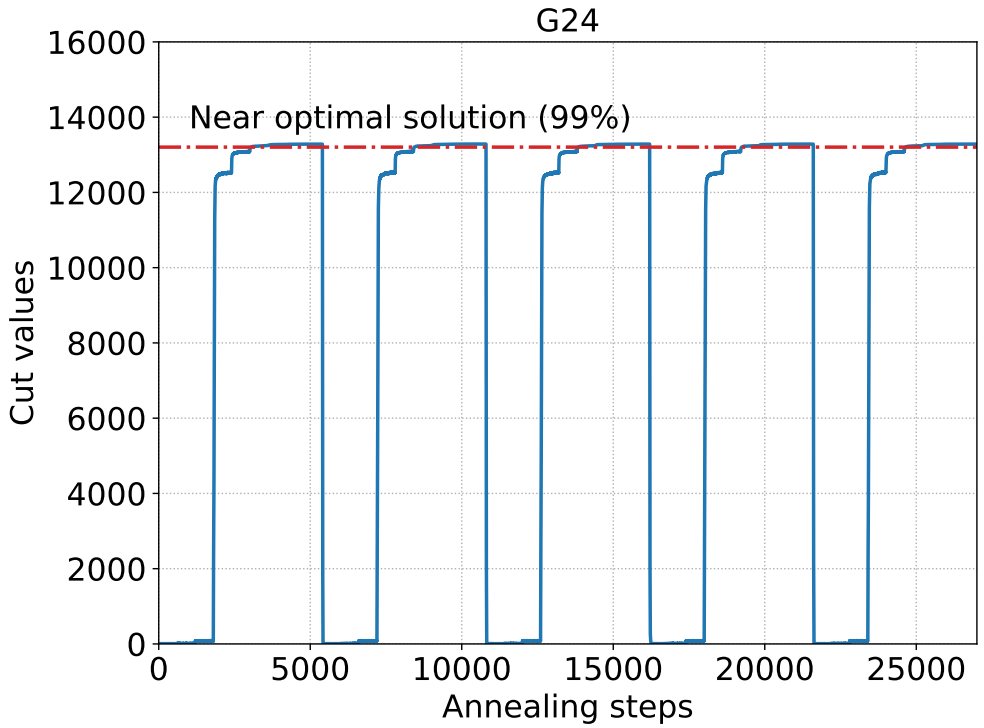


図 4.8: G24 問題において DSSA 法によるアニーリングステップに対しての平均カット値.

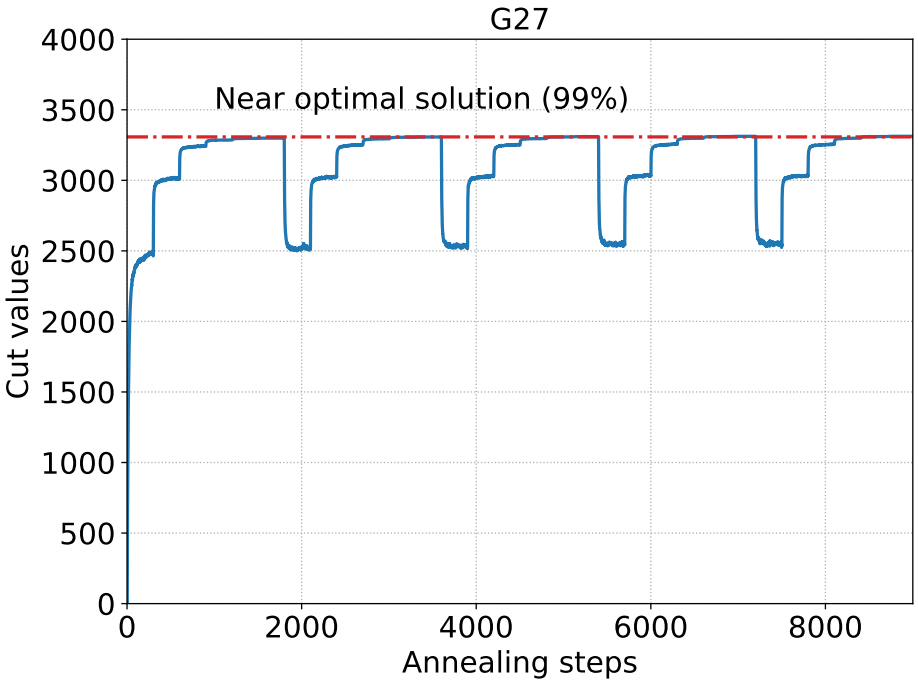
表 4.14: それぞれのベンチマーク問題における DSSA 法から最小，平均，最大解.

問題	Min	Min* [%]	Avg.	Avg.* [%]	Max	Max* [%]
G22	13,280	99.4	13,318.0	99.7	13,349	99.9
G23	13,280	99.5	13,309.5	99.7	13,337	99.9
G24	13,283	99.6	13,300.3	99.7	13,318	99.9
G27[1]	3,289	98.4	3,317.1	99.3	3,338	99.9
G27[2]	3,285	98.3	3,316.8	99.3	3,341	100.0
G35	7,606	98.9	7,634.1	99.3	7,651	99.5
G39	2,345	97.4	2,380.8	98.9	2,397	99.5
K2000	33,073	99.2	33,229.1	99.7	33,337	100.0

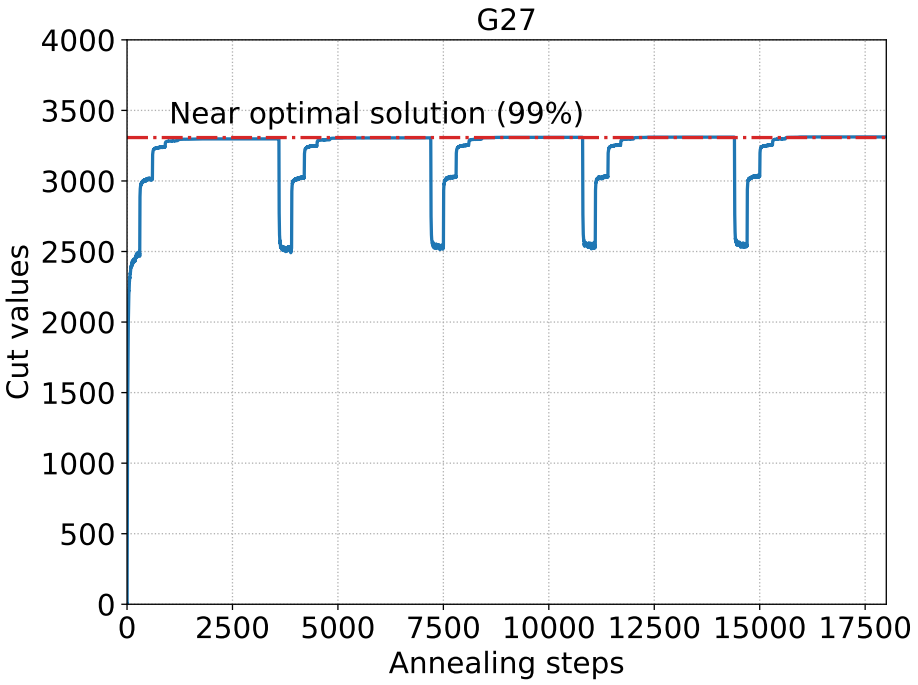
* 最良解との比

[1] 平均解基準のハイパーパラメータの組合せ

[2] 最大解基準のハイパーパラメータの組合せ



(a)



(b)

図 4.9: G22 問題において DSSA 法によるアニーリングステップに対しての平均カット値．(a) 平均解基準のハイパーパラメータの組合せを用いた場合．(b) 最大解基準のハイパーパラメータの組合せを用いた場合．

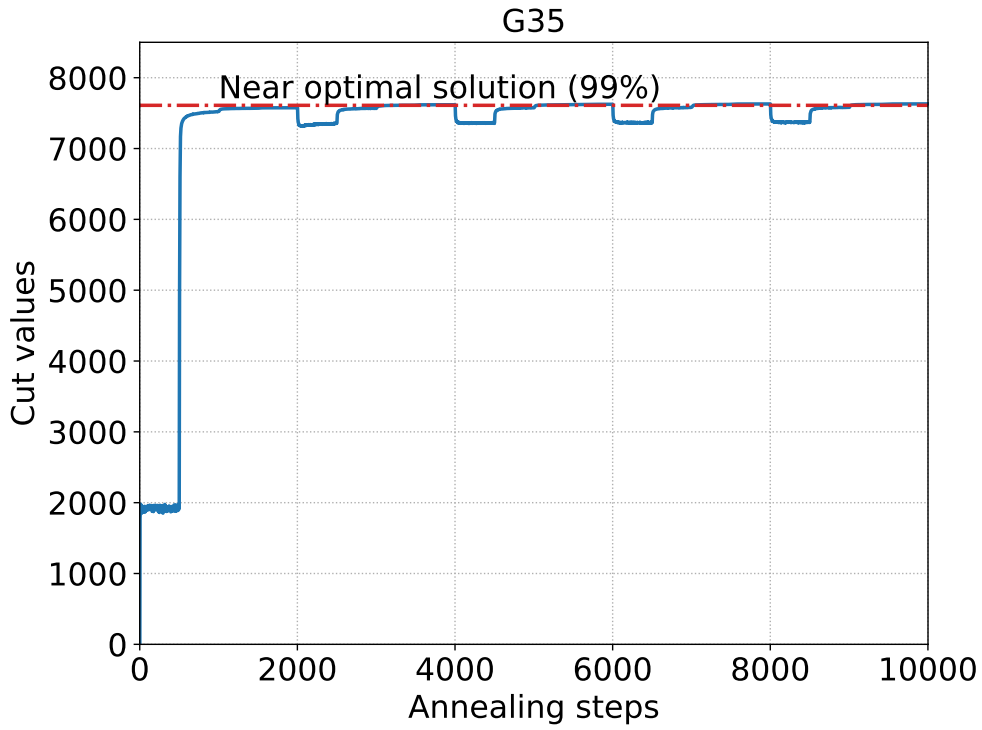


図 4.10: G22 問題において DSSA 法によるアニーリングステップに対しての平均カット値.

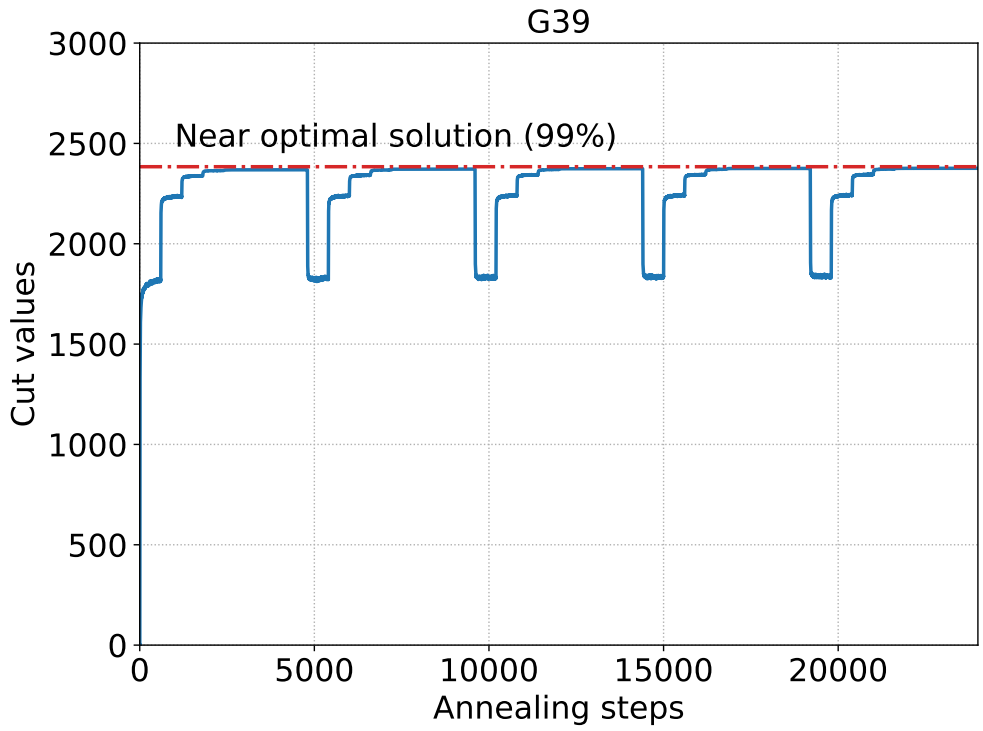


図 4.11: G22 問題において DSSA 法によるアニーリングステップに対しての平均カット値.

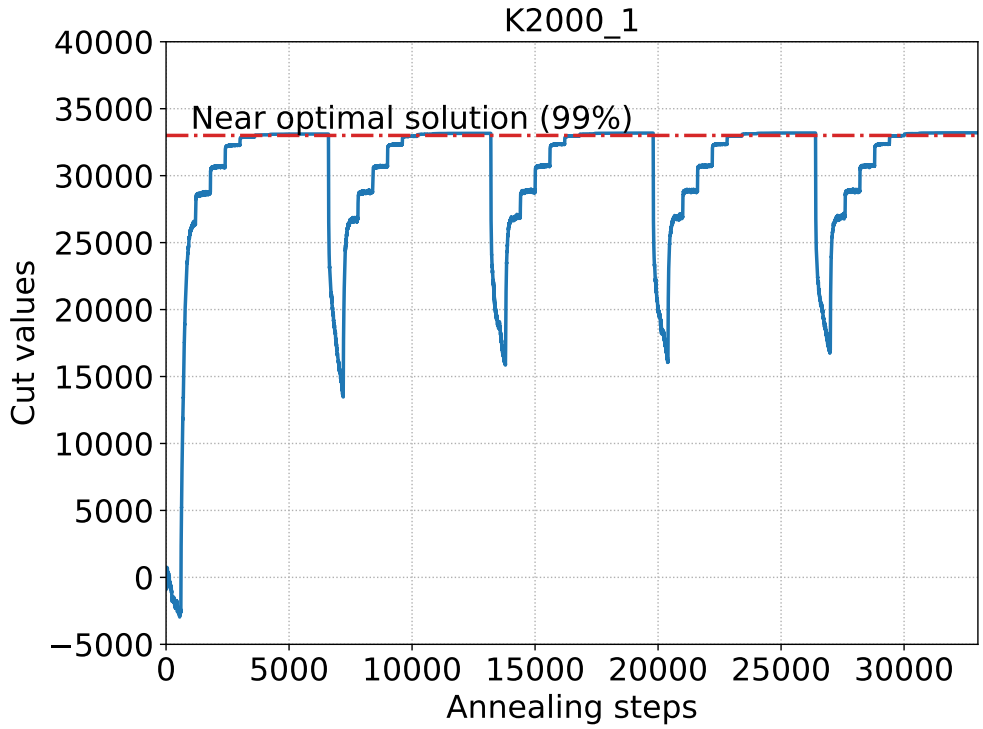


図 4.12: K2000 問題において DSSA 法によるアニーリングステップに対しての平均カット値.

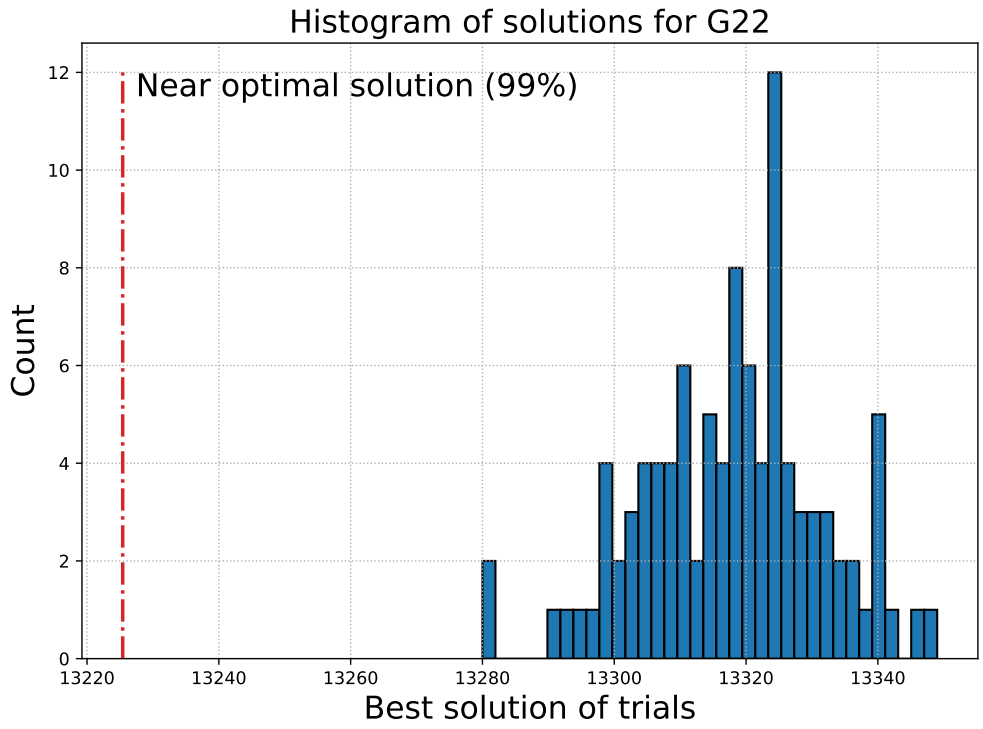


図 4.13: G22 問題においての 100 trials の最大解のヒストグラム.

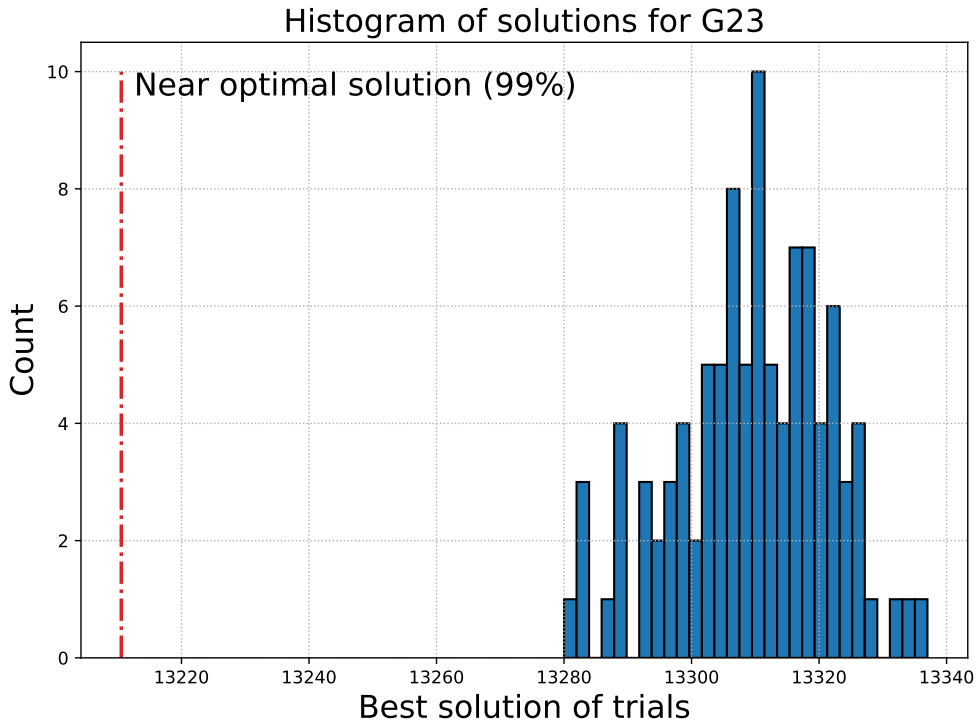


図 4.14: G23 問題においての 100 trials の最大解のヒストグラム.

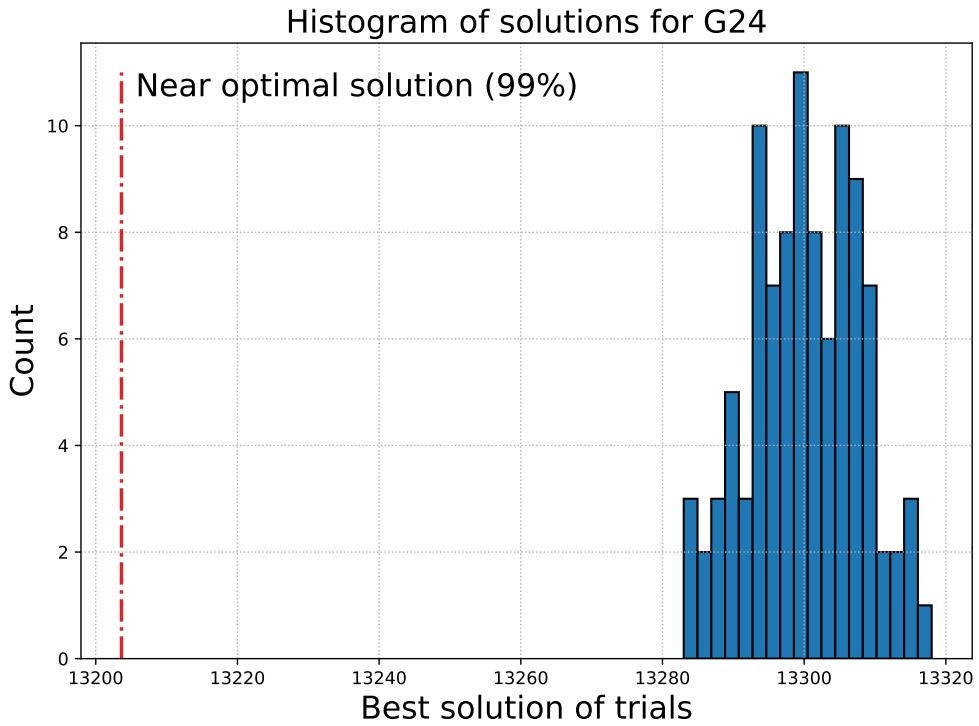
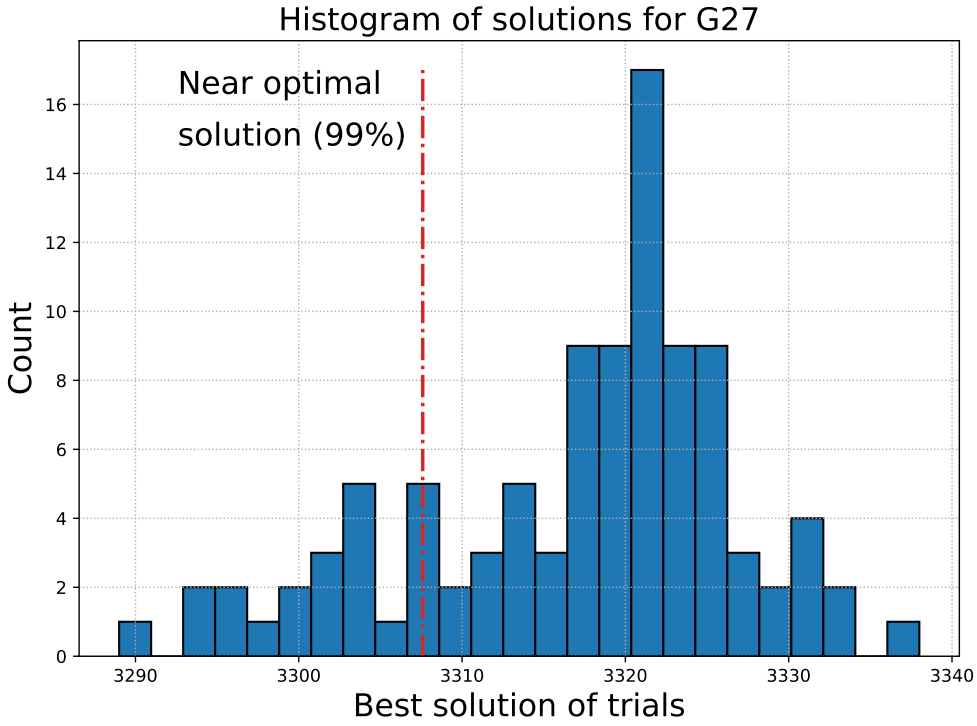
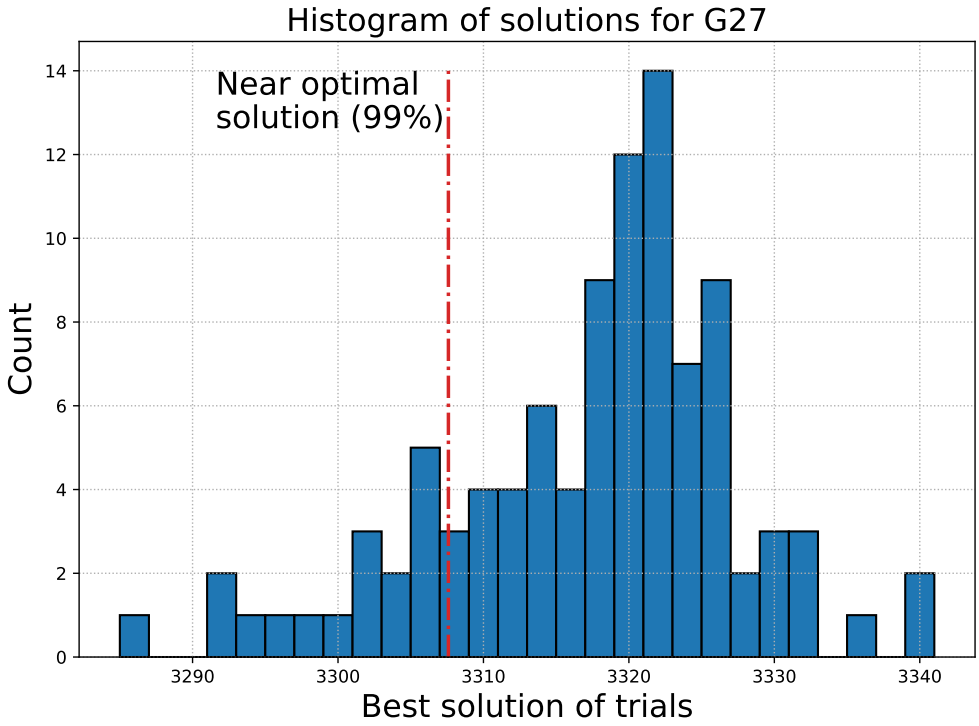


図 4.15: G24 問題においての 100 trials の最大解のヒストグラム.



(a)



(b)

図 4.16: G27 問題においての 100 trials の最大解のヒストグラム．(a) 平均解基準のハイパーパラメータの組合せを用いた場合．(b) 最大解基準のハイパーパラメータの組合せを用いた場合．

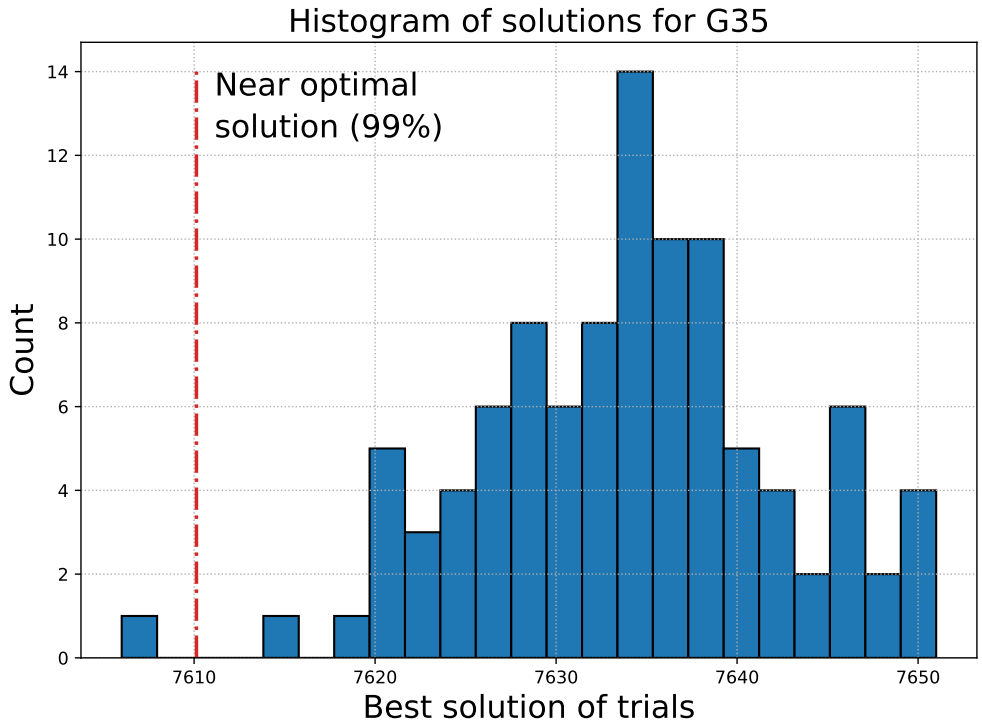


図 4.17: G35 問題においての 100 trials の最大解のヒストグラム.

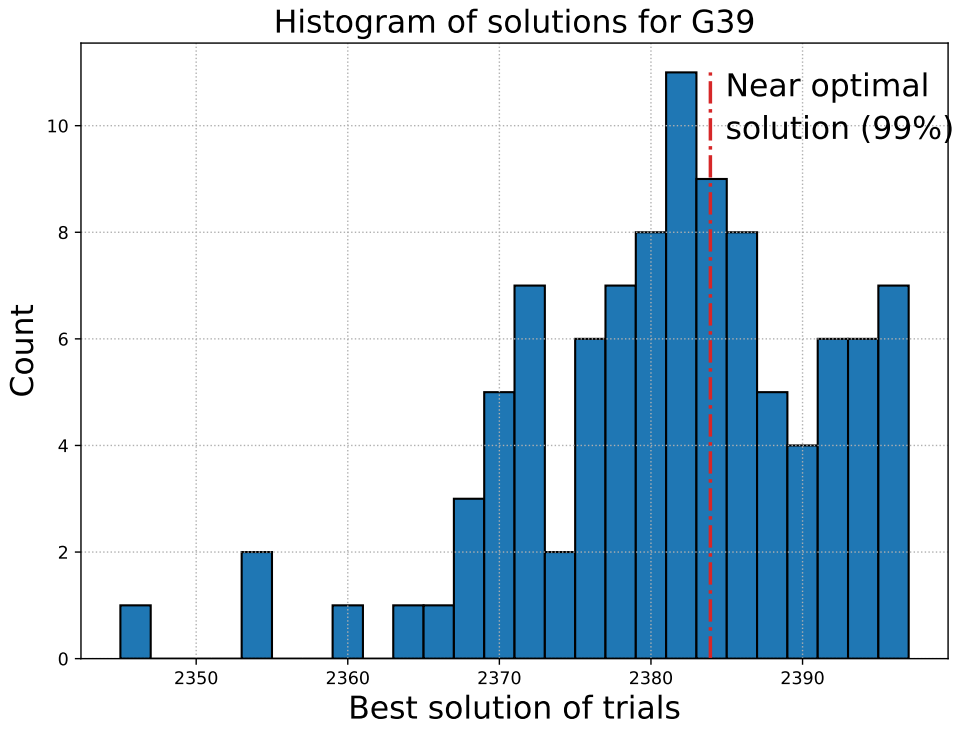


図 4.18: G39 問題においての 100 trials の最大解のヒストグラム.

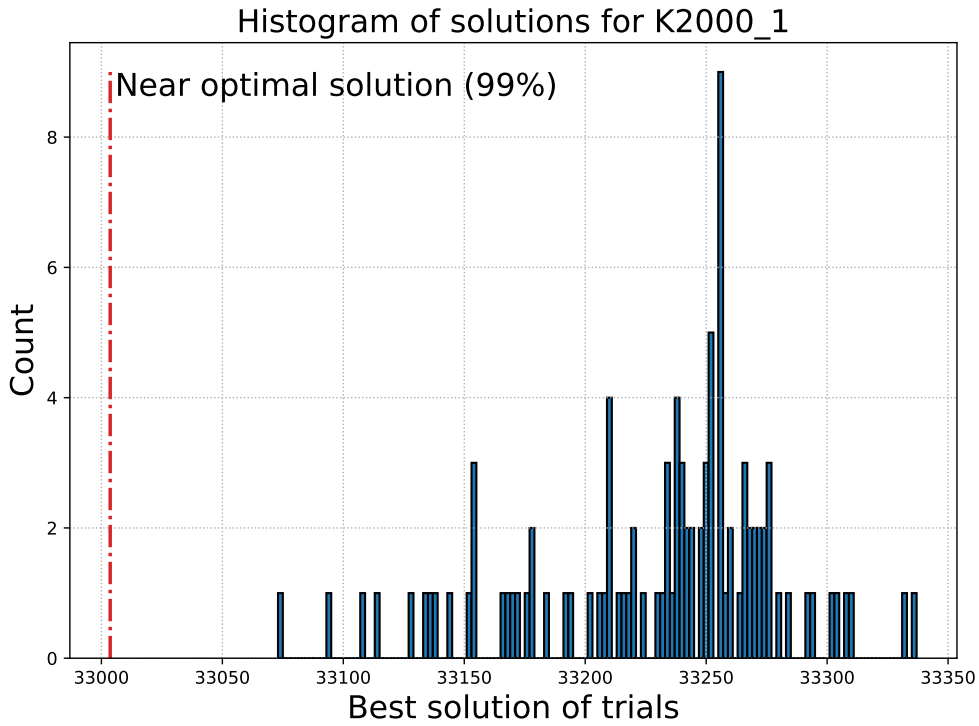


図 4.19: K2000 問題においての 100 trials の最大解のヒストグラム.

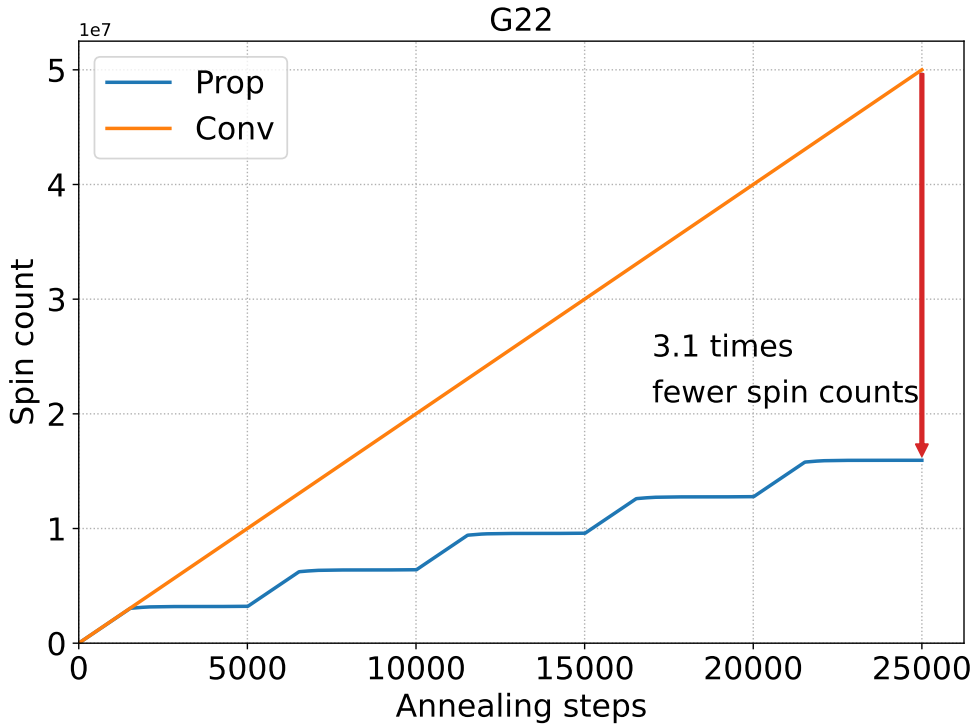


図 4.20: G22 問題において，シリアル化された従来 SSA 法及び DSSA 法のアニーリングステップに対する計算クロックサイクル数の累積.

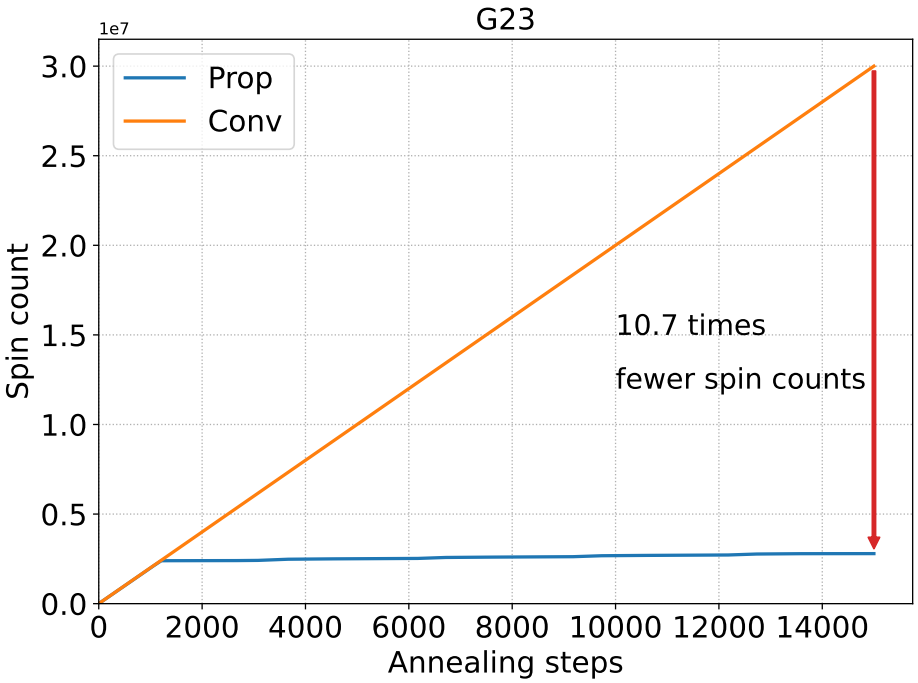


図 4.21: G23 問題において，シリアル化された従来 SSA 法及び DSSA 法のアニーリングステップに対する計算クロックサイクル数の累積.

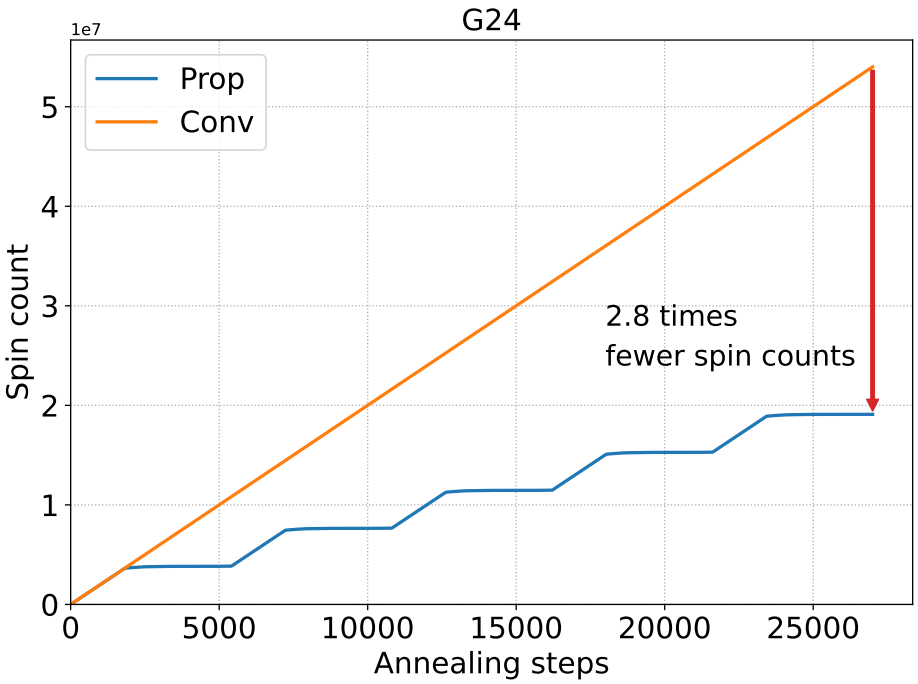
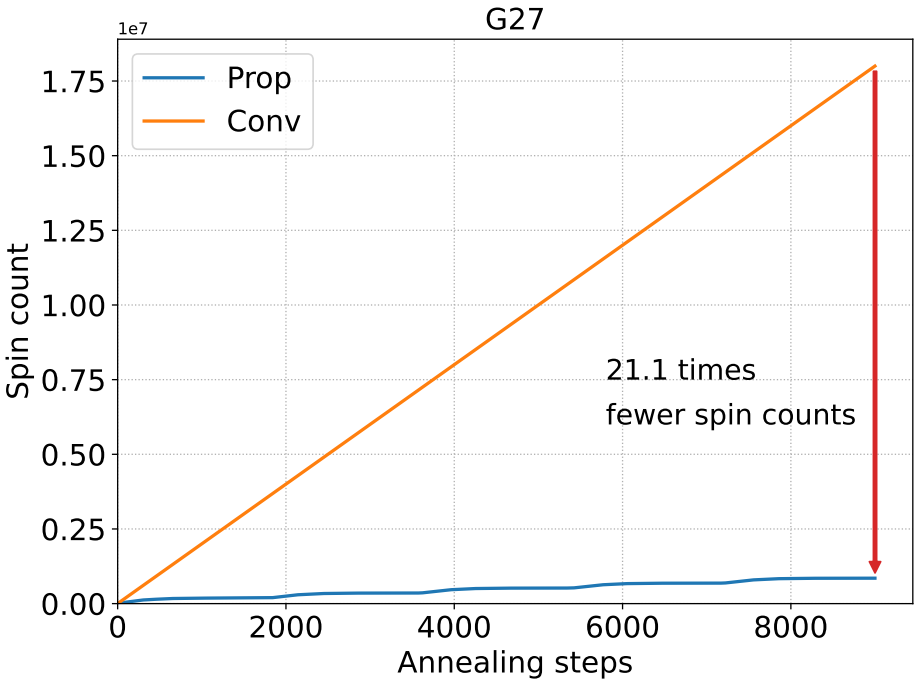
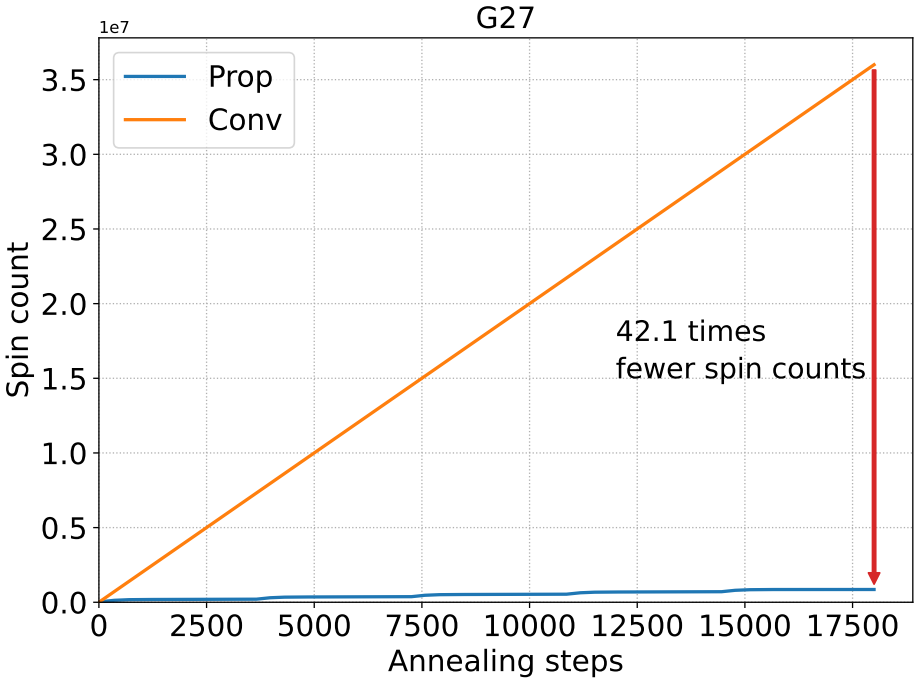


図 4.22: G24 問題において，シリアル化された従来 SSA 法及び DSSA 法のアニーリングステップに対する計算クロックサイクル数の累積.



(a)



(b)

図 4.23: G27 問題において，シリアル化された従来 SSA 法及び DSSA 法のアニーリングステップに対する計算クロックサイクル数の累積．(a) 平均解基準のハイパーパラメータの組合せを用いた場合．(b) 最大解基準のハイパーパラメータの組合せを用いた場合．

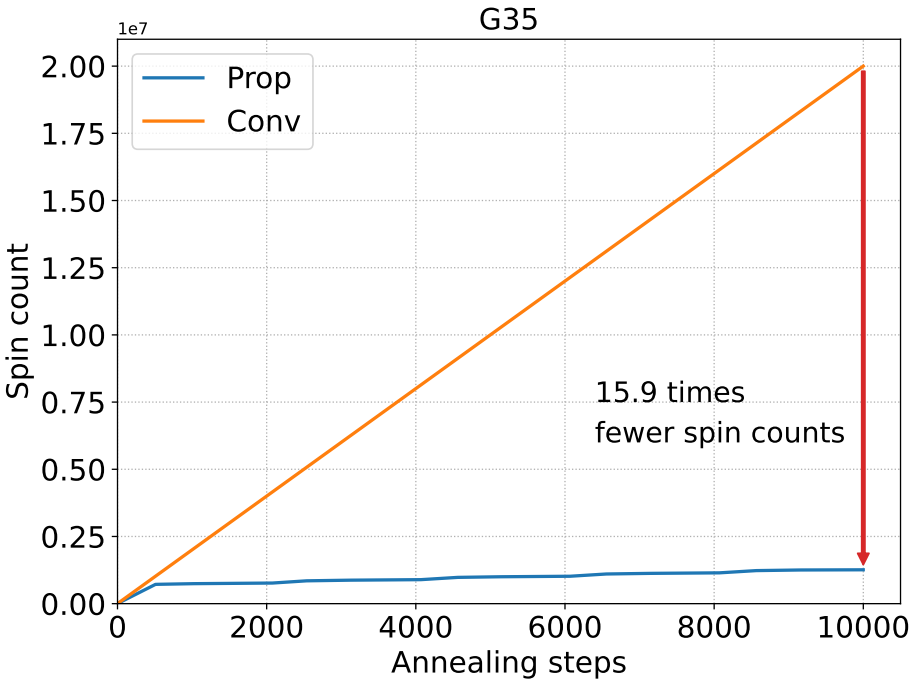


図 4.24: G35 問題において，シリアル化された従来 SSA 法及び DSSA 法のアニーリングステップに対する計算クロックサイクル数の累積.

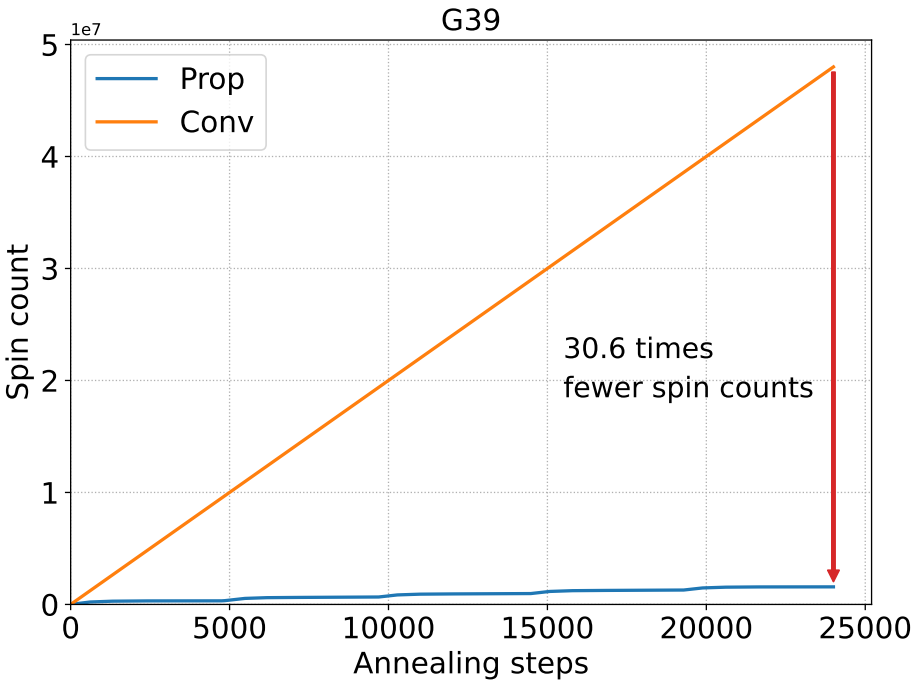


図 4.25: G39 問題において，シリアル化された従来 SSA 法及び DSSA 法のアニーリングステップに対する計算クロックサイクル数の累積.

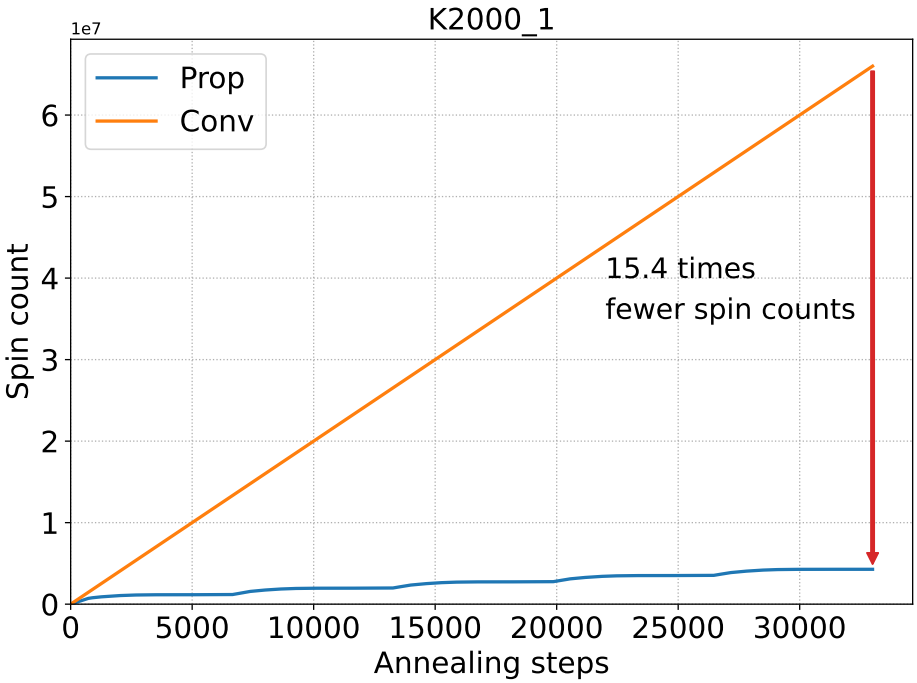


図 4.26: K2000 問題において，シリアル化された従来 SSA 法及び DSSA 法のアニーリングステップに対する計算クロックサイクル数の累積.

ル数を比較する．本論文では 2,048 スピンのイジングモデルを，ハードウェア実装の側面での大規模なイジングモデルとしており，そのモデルのグラフ構造は完全グラフのものである．従って，単純シリアル化された従来 SSA 法も用いられてる問題の実際のスピン当たり接続数ではなく，2,000 個のスピンが互いに完全に接続しているハードウェア構造を想定する．従って，単純シリアル化された従来 SSA 法のスピン相互作用計算のクロックサイクル数は，一つのスピンが 2,000 個の接続を持つ 2,000 スピンが搭載されていると仮定し，1 アニーリングステップの計算には 2,000 クロックサイクルが必要だとする．すなわち，シリアル化された従来 SSA 法の全体アニーリング処理の累積クロックサイクル数は，(1 trials のアニーリングステップ数 × 2,000) で求められる．一方，DSSA 法は，最初のアニーリングステップでは全てのスピン相互作用を計算して最初のスピン状態を確定する．その後，スピン状態が変化した差分スピンとの相互作用のみを計算するため，最初アニーリングステップ以降では差分スピン数に比例して計算クロックサイクル数が累積される．Fig. 4.20 は G22 問題において，Table 4.13 の最適なハイパーパラメータの組合せを用いた際のアニーリング処理を 100 trials 反復した時，単純シリアル化された従来 SSA 法及び DSSA 法のアニーリングステップに対した平均クロックサイクル数の累積を表す．従来 SSA 法は，スピンの動作に関係なく，毎回 2,000 回のスピン相互作用計算を行うため，1 アニーリングステップを処理するたびに 2,000 クロックサイクルずつ線形的に増加する．一方，DSSA 法は各アニーリングステップで状態が変化した差分スピンのみを用いて相互作用を計算するため，各アニーリングステップに必要なクロックサイクル数が差分スピンの数に比例する．Fig. 4.20 のように，差分スピン数は逆温度パラメータに応じて変化し，低い逆温度ではスピン状態の変化が激しいため 1 アニーリングス

表 4.15: 単純シリアル化された従来 SSA 法と DSSA 法における各アニーリングステップに必要なクロックサイクル数と累積クロックサイクルの比較．DSSA 法は各アニーリングステップでの平均差分スピン数に比例して従来 SSA 法より高速化できる．

問題	ステップ当たりのクロックサイクル数		累積クロックサイクル数の比	DSSA 法の全体スピン数と平均差分スピン数の比
	単純シリアル	DSSA	(Conv. SSA/DSSA)	
G22	2,000	638.0	3.1	31.9%
G23		186.2	10.7	9.3%
G24		707.3	2.8	35.4%
G27[*1]		94.6	21.1	4.7%
G27[*2]		47.5	42.1	2.4%
G35		126.0	15.9	6.3%
G39		65.4	30.6	3.3%
K2000		129.6	15.4	6.5%

[*1] 平均解基準のハイパーパラメータの組合せ

[*2] 最大解基準のハイパーパラメータの組合せ

テップの処理に比較的多くのクロックサイクル数が必要となる．スピン状態の変化は逆温度パラメータの増加と共に穏やかになり，低い逆温度の時のアニーリング処理より少ないクロックサイクルで処理ができる．全てのアニーリングステップの処理が終わった後，累積のクロックサイクル数を比較すると，DSSA 法は単純シリアル化された従来 SSA 法より約 3.1 倍少ないクロックサイクルで等しいステップ数のアニーリング処理ができた．続いて，G22 以外のベンチマーク問題において，単純シリアル化された従来 SSA 法と DSSA 法のアニーリングステップに対する累積クロックサイクル数を Figs. 4.21, 4.22 and 4.24 to 4.26 and Section 4.4 にて表す．どの問題においても DSSA 法は従来 SSA 法より少ないクロックサイクルで等しいステップ数のアニーリング処理ができた．

Table 4.15 はそれぞれのベンチマーク問題対して，単純シリアル化された従来 SSA 法と DSSA 法の 1 アニーリングステップでかかる平均クロックサイクル数及び全体アニーリング処理にかかった累積クロックサイクルの比を表す．加えて，DSSA 法での問題の全体スピン数とステップ当たりの平均差分スピン数の比も示す．DSSA 法のアニーリングステップ当たりの平均クロックサイクル数は問題によって様々ではあるが，どの問題でも従来 SSA 法より少ない．また，アニーリングステップでの差分スピンの数の平均，M，は G27 問題で最小である 47.5 個であり，最大でも G24 問題で 707.3 個である．ベンチマーク問題は全て 2,000 ノード問題であり，全体のスピン数，2,000，に対した差分スピン数，M の比 $(M/2,000)$ は，最小の場合が G27 問題の 2.4% から，最大の場合でも G24 問題の 35.4% の範囲であり，全体スピン数に比べて比較的に少ない数のスピンだけが状態を変化する．毎回のアニーリングステップで 2,000 クロックサイクルがかかる従来 SSA 法と DSSA 法を比較すると，最悪の場合でも G24 問題のように約 2.8 高速で同等なアニーリング処理ができる．最善の結果であれば，G39 問題の約 30.6 倍高速

化でき，DSSA 法によるシリアル処理の高速化が有効であることを確認できる．特に，2,000 スピンの完全接続問題である K2000 に関しては，DSSA 法のアニーリングステップ当たりの平均差分スピン数は 129.6 であり，全体スピンのわずか 6.5% のみが相互作用の計算に用いられる．全体のアニーリング処理の累積クロックサイクル数も，単純シリアル化された従来 SSA 法より約 15.4 倍高速で同等な処理ができる．

このように，DSSA 法は大規模な完全グラフ構造のイジングモデルに対応可能な SSA ハードウェア実装という課題を達成するためのスピン相互作用の計算を，シリアル化による小面積化を用いて実現する．また，シリアル化されたスピン相互作用の計算を，状態が変化した差分スピンのみを用いて行うことで，シリアル化による計算時間のトレードオフを効果的に削減できる．DSSA 法は COP の一種である MAXCUT 問題において，それぞれの問題の最良解の約 99% の近似解を高い確率で達成でき，高精度のアニーリングアルゴリズムであることを示した．また，2,000 ノードの MAXCUT 問題において，問題に応じて約 2.8 倍から約 47.5 倍高速で単純シリアル化された従来 SSA 法と同等な処理を可能とする．特に，K2000 という 2,000 ノード完全接続 MAXCUT 問題の最良解である 33,337 のカット値を達成でき，100 trials の平均でも最良解の約 99.7% 以上の近似解を達成した．単純シリアル化された従来 SSA 法と比べ，DSSA 法は K2000 問題を約 15.4 倍高速で解くことができた．

4.5　むすび

本書では SSA 法に基づいたアニーリングハードウェアが対応可能なイジングモデルの構造に応じて，ハードウェアに搭載されたスピンの接続数の増加が及ぼすハードウェア面積の増加について考察し，大規模な完全グラフイジングモデル向けアニーリングハードウェアを実現するためのハードウェア面積の課題について述べた．完全接続のスピンを小面積に実装するための，スピン相互作用の計算のシリアル化の面積削減の有効性と共に，時間分割されたスピン相互作用の計算から生じる計算時間増加のトレードオフ関係について考察した．これらの課題から，シリアル化されたスピン相互作用計算を高速に行うために，状態が変化した差分スピンのみを用いてスピン状態を更新する DSSA 法について詳しく述べた．DSSA 法は，スピン状態の確率の変化量のみを差分スピンより求めることで，状態が変化していないスピンとの接続を無視できるアルゴリズムであり，COP の一種である MAXCUT 問題を用いてその有効性を確認した．完全グラフ構造を含む様々な構造を持つ 2,000 ノードの MAXCUT 問題に対して，DSSA 法は 100 回の反復試行の平均解が，それぞれの問題の最良解の最悪でも約 98.3% (G39 問題)，最善では約 99.7% (G22，G23，G24，K2000 問題) の近似解を達成でき，G27 問題はその最良解を達成できた．加えて，どのベンチマーク問題でも指定のアニーリング処理の初期にすべに近似解に到達でき，2,000 スピンのイジングモデルに対して高速な収束時間を示した．

2,000 スピンの完全接続のアニーリングハードウェアを考えた場合，スピン相互作用の計算を単純にシリアル化させた従来 SSA 法と同等なアニーリン処理を，最小でも 2.8 倍 (G24)，最大で 47.5 倍 (G27) 倍高速で行った．特に，2,000 ノードからなる完全グラフの MAXCUT 問題である K2000 問題に関しては，100 回の反復試行の平均解が K2000 の最良解の約 99.7% である 33,229.1 を達成し，最大解としては最良解を得ることができた．このように，DSSA 法は COP を高速かつ高精度で解くことができ，大規模な完全接続を小面積かつ効率的に実装可能とする．すなわち，大規模な完全グラフのイジングモデル

向けの SSA ハードウェア実装に向けて，DSSA 法はその基盤となるハードウェアアルゴリズムである．

第 5 章

DSSA に基づく大規模・完全グラフ向け高性能アニーリングプロセッサの実現

5.1 まえかき

本章では差分スピンを用いてシリアル化されたスピン相互作用の計算を高速化するハードウェアアルゴリズムである DSSA 法に基づいた DSSA アニーリングプロセッサについて述べる．DSSA 法は大規模な完全グラフ楮のイジングモデル向けのアニーリングハードウェアを実装するために提案され，スピン相互作用の計算を高速化することで，計算の時間分割によるシリアル化から生じる面積と計算時間のトレードオフの関係を乗り越えることを可能とする．このような DSSA 法に基づいて，従来の ASIC で実装された state-of-the-art のアニーリングプロセッサより大規模のイジングモデルかつ完全接続に対応可能な DSSA プロセッサを実装する．提案 DSSA プロセッサは単一チップに，stete-of-the-art のアニーリングプロセッサより多くの，2,048 スピンを搭載し，その接続も完全グラフ構造を持つ．初めに，完全接続の 2,048 スピン DSSA プロセッサのアーキテクチャについて述べ，差分スピンを用いた DSSA 法のスピン相互作用計算を行う DSSA の spin-gate 設計を含むプロセッサのコンポーネントについて説明する．その後，DSSA 法のスピン動作に応じたプロセッサのデータフロー及びタイミングチャートなどプロセッサの動作について述べる．設計した DSSA プロセッサを ASIC 実装するための実装環境について述べ，FPGA を用いたプロセッサのサンプルデザインのシミュレーションを含め，プロセッサの ASIC 実装について述べる．最後に，設計された DSSA プロセッサの性能を評価し，state-of-the-art の研究を含む従来の ASIC ベースのアニーリングプロセッサとその性能を比較する．

5.2　DSSA プロセッサのアーキテクチャ

DSSA プロセッサのアーキテクチャについて述べる前に，イジングモデルで表現される COP 向けの ASIC ベースのアニーリングプロセッサの先行研究について概説する．ASIC ベースのアニーリングプロセッサの先行研究の一つは単一チップに 16k のスピンを搭載した CMOS-AP が研究されている [25]. CMOS-AP は単一チップで 16k スピンを搭載しており，最大 9 個のマルチチップを用いて 144k のスピンを扱うことができる．一方，CMOS-AP のスピン接続構造は完全グラフではなく，Chapter 3 で示した 800 スピン SSA ハードウェアのように，一つのスピンが隣接の 8 個のスピンと接続する king‘s graph 構造である．STATICA は単一チップに 512 個の完全接続のスピンを搭載した ASIC ベースのアニーリングプロセッサである [26]．アニーリングプロセッサの state-of-the-art の先行研究は Amorphica であり，単一チップに完全接続の 512 スピンを搭載している [29]．Amorphica は単一チップ上の四つのレプリカ，または，四つのマルチチップを活用して 2,048 スピンを単一チップで扱うことができる．しかし，チップ上のメモリには 512 スピンのイジングモデルのパラメータのみを格納できるため，2,048 スピンに対応するためには，パラメータの通信やチップ間通信が必要である．このように，state-of-the-art のアニーリングプロセッサでも単一チップで完全接続の 2,048 スピンに対応することは更なる研究が必要である．本論文で，state-of-the-art の研究も含めたアニーリングプロセッサに関する先行研究から，ハードウェア実装の側面での大規模な完全グラフのイジングモデルは 2,048 スピンからなる完全グラフのイジングモデルとし，提案 DSSA プロセッサもマルチチップやレプリカなどを使わずに 2,048 スピンの完全グラフモデルに対応することを想定する．

それでは，シリアル化より生じる面積と計算時間のトレードオフ関係を差分スピンを用いて乗り越える DSSA 法に基づく提案アニーリングプロセッサの全体のアーキテクチャを Fig. 5.1 に示す．前述のように提案 DSSA プロセッサは完全接続の 2,048 スピンに対応可能であり，800 スピン SSA ハードウェアのように，2,048 スピンは spin-gate array としてプロセッサに搭載されている．2,048 スピンの完全グラフイジングモデルの重みは全てプロセッサ内部のメモリ (WM, weight memory) に格納される．WM の出力は spin-gate array (SGA) モジュールに含まれているそれぞれの spin-gate に接続され，スピン相互作用の計算に用いられる．従来 SSA 法の spin-gate の出力は spin-gate が対応するイジングスピンの状態のみである．一方，DSSA 法の spin-gate は対応するイジングスピンの状態が変化した差分スピンかを検出する必要があり，提案 DSSA プロセッサの spin-gate はスピンの状態と共に差分スピンかを示す差分フラグ信号，Δ_i，も出力する．Spin-gate から生成されたスピン状態，σ_i，及び差分フラグ信号，Δ_i，はスピン相互作用計算に用いられる差分スピンを逐次的に選択する spin select circuit (SSC) モジュールに入力される．SSC は差分スピンフラグより計算に用いられるスピンの状態を選択して SGA に入力し，入力されたスピンの状態はスピン相互作用計算に使われる．また，逐次的なスピン相互作用途中，既に計算された差分スピンのフラグをリセットするさ，差分フラグリセット信号，rstΔ_j，も SSC で生成される．DSSA 法のスピン動作にも従来 SSA ハードウェアと同様に，逆温度パラメータ，乱数信号の大きさと乱数信号が必要であり，DSSA プロセッサ内部から全ての信号が生成される．乱数信号は乱数発生器 (RNG) から生成され，DSSA プロセッサでは xorshift [49] を RNG として用いる．逆温度信号，$I_0(t)$，及び乱数の大きさ，n_{rnd} は，scheduler モジュールから生成される．Scheduler モジュー

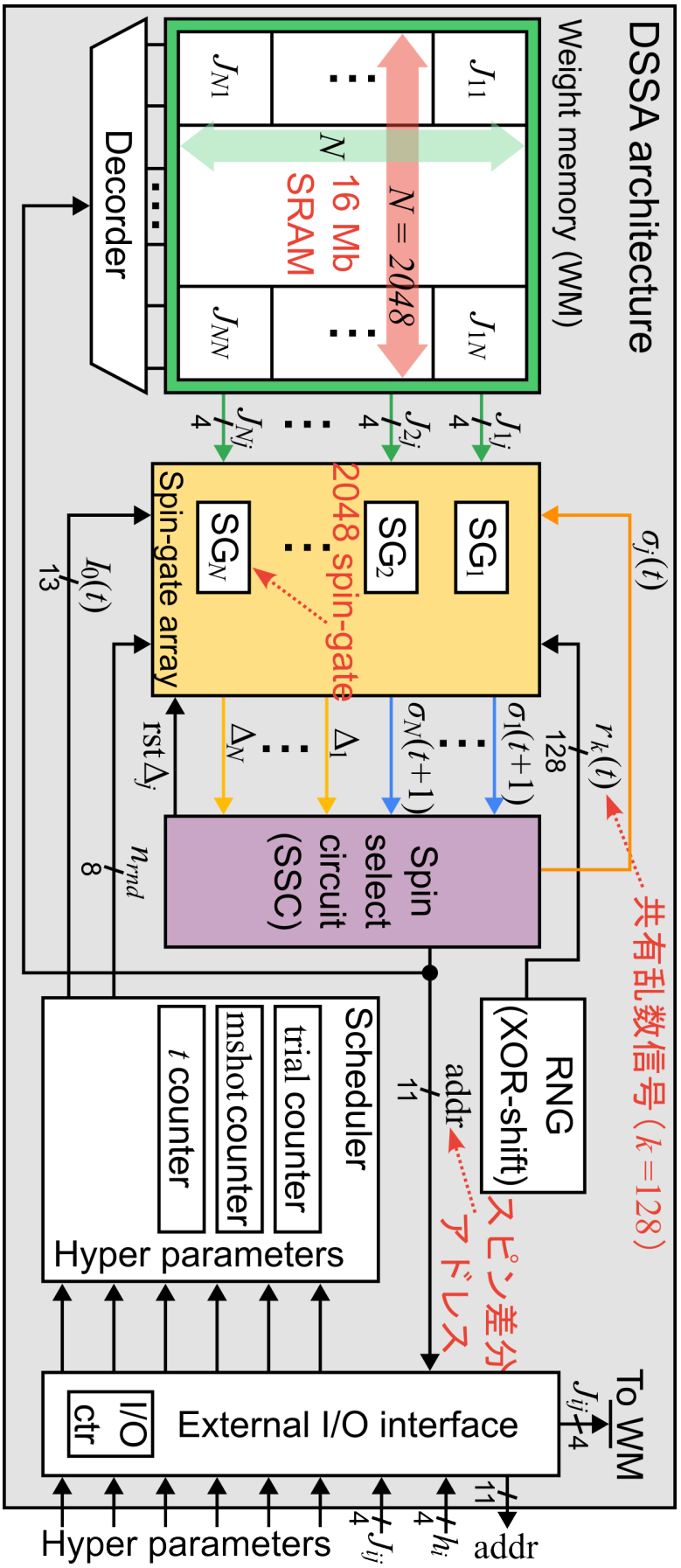


図 5.1: 2,048 スピンが搭載された大規模な完全グラフアイジングモデル向け DSSA プロセッサのアーキテクチャ.

ルは逆温度よ乱数の大きさの生成だけではなく，WM の入出力の制御，enable や reset などの SGA と SSC の制御信号なども生成する．逆温度パラメータや DSSA プロセッサのアニーリング動作を制御はハイパーパラメータに従って行われるため，scheduler はチップ外部から通信インタフェースを通して必要なハイパーパラメータを受け取る．同じく，イジングモデルの重み，J_{ij}，とバイアス，h_i，のパラメータは通信インタフェースを通じてチップ外部からプロセッサに入力され，J_{ij} は WM に格納され，h_i は spin-gate に直接入力される．

通信インタフェースから入力されたイジングモデルの重み，J_{ij}，は WM に保存される．DSSA プロセッサの J_{ij} は 800 スピン SSA ハードウェアと同様に 4 ビットの符号付整数であり，表現可能な J_{ij} の範囲は $\{-8,-7,...,6,7\}$ である．2,048 スピンの完全グラフイジングモデルのエッジの数 (重みの数) は $(2,048 \times 2,047)/2 = 2,094,128$ であるが，スピン相互作用の計算は Eq. (4.1a) や Eq. (4.11a) に従って，行列とベクトルの演算であるため，メモリに保存される重みは $(2,048 \times 2,048) = 2^{22}$ 個である．それぞれの重みのビット幅は 4 ビットであるため，DSSA プロセッサの WM の容量は $(2^{22} \times 2^2) = 2^{24} = 16 \cdot 2^{20} = 16$Mb である．DSSA プロセッサのスピン相互作用は逐次的に計算されるが，搭載スピン数は 2,048 であるため，1 回の相互作用計算処理では 2,048 個の相互作用を計算する．例えば，スピン i と他のスピンとの相互作用を計算する際，スピン i はプロセッサに搭載されてる全てのスピンと接続しているため，1 クロックサイクルで計算されるスピン相互作用は 2,048 接続である．つまり，スピン相互作用計算の際，1 クロックサイクルで用いられる相互作用の重みの総ビット幅は 2.048×4bits であり，WM のワードの幅は $2^1$3 ビットである．WM は static random access memory (SRAM) であり，当たり前に 2^{13} ビットのワード幅の SRAM は現実的ではないため複数の SRAM マクロを用いて WM を構成する．WM から差分スピンの相互作用の重みを読み出すためには，それらの重みが格納されているメモリのアドレスが必要となる．WM に格納される重み行列の次元は $2,048 \times 2,048$ であり，WM の深さは 2^{11} であり，アドレスのビット幅は 13 ビットである．その差分スピンの重みは差分フラグ，Δ_j，より SSC から生成され，WM に入力される．差分スピンのアドレスはスピンの番号であり，例えば，1 番目のスピンのアドレスは '$00000000000_{(2)}$' である．

WM から読み出された重み，J_{ij}，は SGA に入力され，スピン相互作用の計算が行われる．Fig. 5.2 は 2,048 個の spin-gate から構成される SGA のアーキテクチャを示す．SGA は 2,048 個の spin-gate から構成される DSSA プロセッサのメインコンポネントである．800 スピン SSA ハードウェアは SGA の内部で king's graph 構造を基盤として spin-gate が隣接の spin-gate と配線より接続されている．一方，DSSA プロセッサの SGA は内部の spin-gate 同士は接続していない．全ての spin-gate の出力，$\sigma_i(t+1)$，は SGA モジュールの外部に出力され，SSC に入力される．従来 SSA ハードウェアと同様に，$\sigma_i(t+1)$ はイジングスピンの状態が '-1' であれば論理値 "0"，'+1' であれば論理値 "1" となる．スピン相互作用に用いられる差分スピンの状態，$\sigma_j(t)$，は SSC から SGA に入力される．Scheduler から生成された逆温度パラメータ，$I_0(t)$，及び乱数信号の大きさ，n_{rnd}，は，それぞれ 13 ビット，8 ビットの整数であり，スピンの動作に影響する．従来 SSA ハードウェアと異なる部分は，あるスピンが差分スピンなのかを表す差分フラグ信号，$\Delta_j(t)$，である．差分フラグ信号は対応するスピンの状態が変化した差分スピンである場合 1 になり，状態変化がないスピンは 0 である．これらの差分フラグ信号はそれぞれの spin-gate から生成され，スピン状態，$\sigma_i(t+1)$，と共に SSC に入力され，差分スピンの選択に用いられる．提案 DSSA プロセッサの搭載スピン数は 2,048 個であるため，SGA が出力するスピン状態信号，

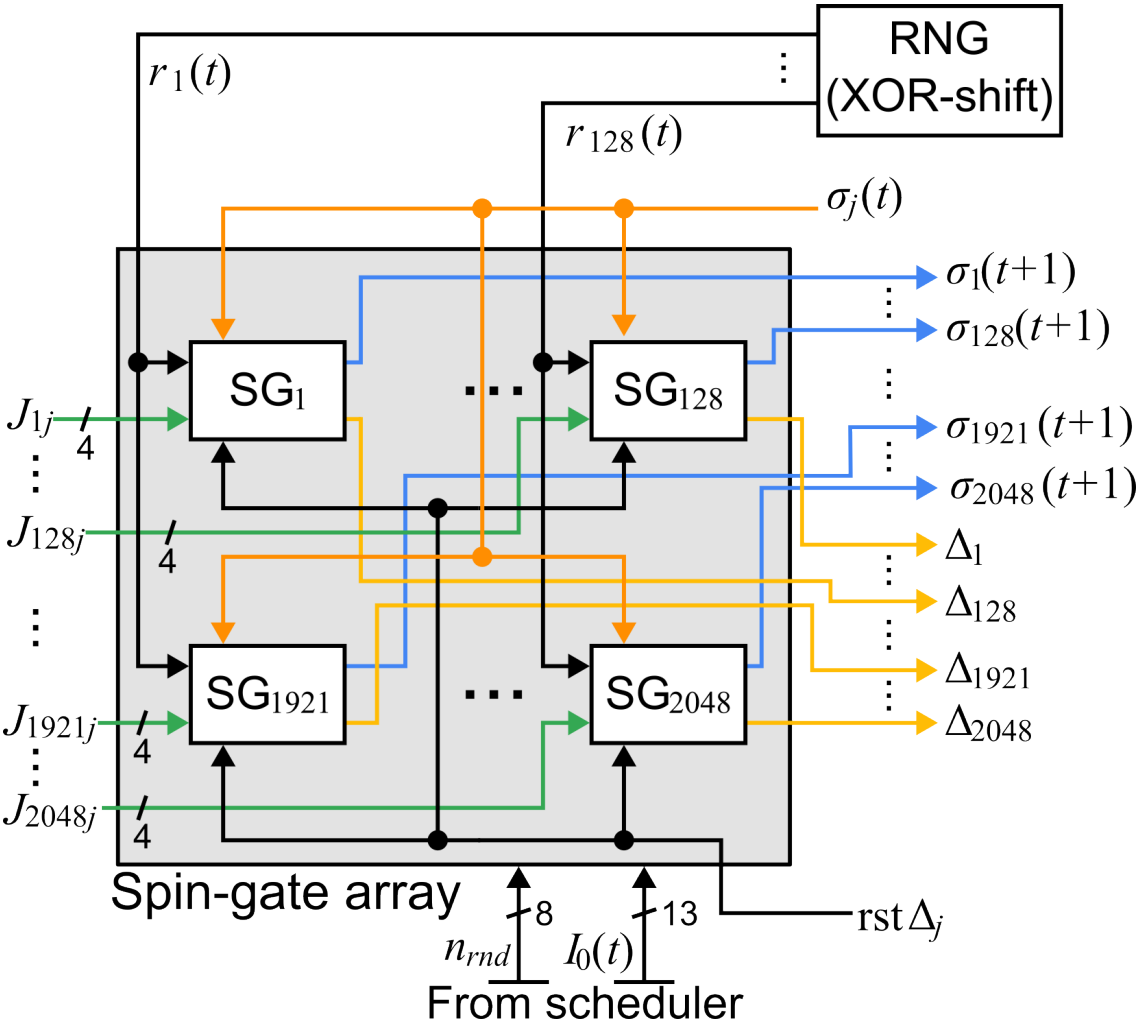


図 5.2: 2,048 個の spin-gate からなる spin-gate array (SGA) の構造．SGA はスピン接続のシリアル化よりその配線の面積を削減する．

$\sigma_i(t+1)$，と差分フラグ信号，$\Delta_j(t)$，のビット幅は 2,048 ビットである．

従来 SSA 法と同様に，DSSA 法もスピンの確率的な動作を実現するために，乱数信号が必要であり，提案プロセッサでは xorshift を RNG として用いる．800 スピン SSA ハードウェアでは各スピンに 1 ビットの乱数信号を用いるため，800 ビットの乱数信号を RNG から生成する．同様に，2,048 スピンの spin-gate が搭載された提案プロセッサには，Fig. 5.3 のように搭載スピン数と等しい 2,048 ビットの乱数信号が必要であり，RNG のコストが増加する．提案 DSSA プロセッサでは，RNG のコストを削減するために，複数の spin-gate で乱数信号を共有することで，乱数信号のビット幅を削減する．Fig. 5.4 は提案プロセッサの RNG と spin-gate 間の乱数信号の配線を表す．1 ビットの乱数信号が 16 個の spin-gate に共有され，RNG から生成される乱数信号のビット幅は $2.048/16 = 128$ ビットのみである．RNG の生成ビット幅に対するハードウェア面積の比較を Table 5.1 に表す．RNG のハードウェア面積は TSMC 28nm プロセスを用いて，論理合成後の面積である．RTL (register-transfer level) 設計からの論理合成にが Synposys Design compiler 22.12 より行われた．2,048 ビットを生成する xorshift

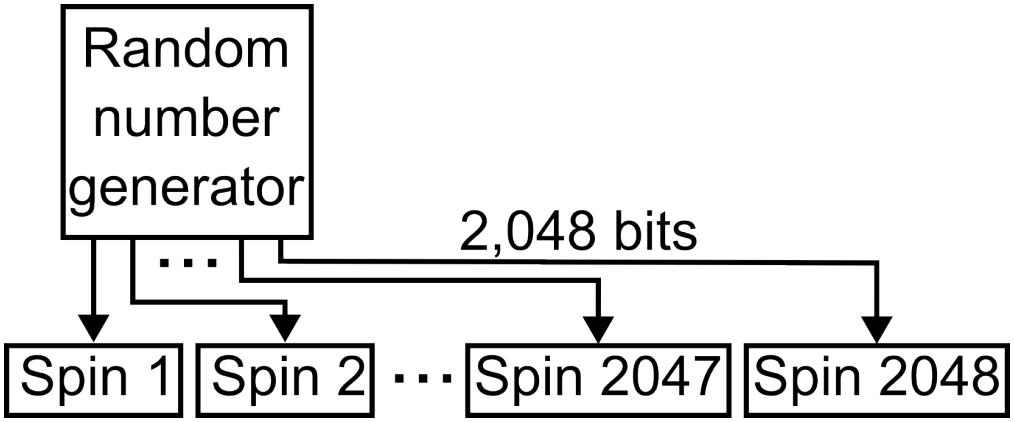


図 5.3: それぞれの spin-gate が 1 ビットの乱数信号を用いる時の乱数発生器と乱数信号の配線．一つの spin-gate に 1 ビットの乱数信号を用いて，必要な乱数信号のビット幅は 2,048 ビットである．

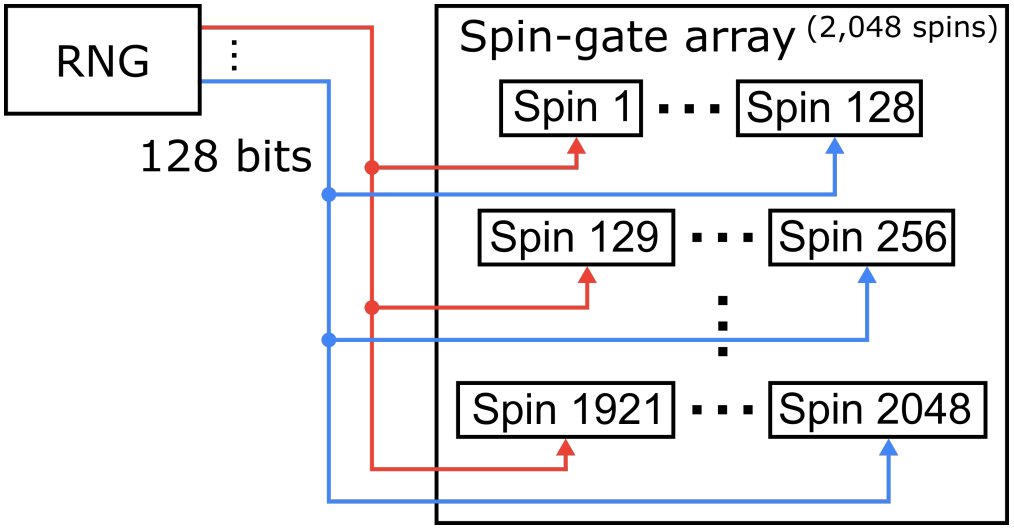


図 5.4: 提案 DSSA プロセッサの乱数発生器と乱数新語の配線．1 ビットの乱数信号を 16 個の spin-gate が共有することで，乱数信号のビット幅を 128 ビットに削減する．

表 5.1: 乱数発生器の生成乱数信号のビット幅に対するハードウェア面積の比較．ハードウェア面積は TSMC 28nm プロセスを用いて論理合成後の結果である．

乱数信号のビット幅	2,048 ビット (Conv)	128 ビット (Prop)
ハードウェア面積 [μm^2]	112,298.6	3,563.8

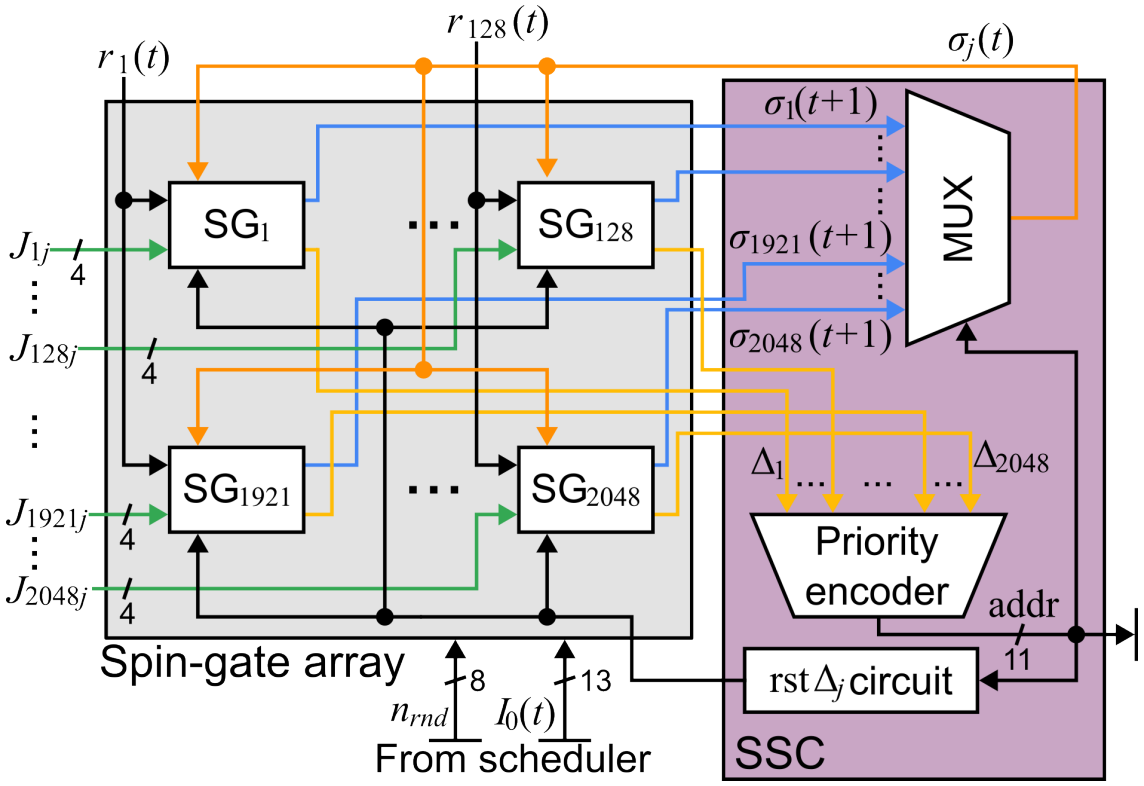


図 5.5: SSC モジュールの構造と SGA と SSC の配線.

の面積は 112,298.6 μm^2 であるが，提案プロセッサの 128 ビットの RNG の面積は 3,563.8 μm^2 であり，約 31.5 倍小面積化できる．提案 DSSA プロセッサの全体の面積で RNG の面積の割合が少ないかもしれないが，乱数信号の共有化によるビット幅の削減は更なる大規模化に有効な手法である．

スピン相互作用に用いられるスピンは SGA から生成されたスピン状態，$\sigma_i(t+1)$，及び差分フラグ信号，$\Delta_j(t)$，より SSC で選択される．Fig. 5.5 は SSC の構造及び SGA と SSC の接続を示す．SSC では $\Delta_j(t)$ 信号から priority encoder を用いて，$\Delta_j(t) = 1$ のスピンの番号を逐次的に出力する．Priority encoder から生成された差分スピンのアドレスは WM から J_{ij} を読み出すためのアドレス，及び SSC 内部で差分スピンの状態を選び SGA へ入力するためのマルチプレクサの入力として用いられる．また，アニーリング処理途中，既にスピン相互作用の計算に用いられた差分スピンが再び選択されることを防ぐために，使われた差分スピンのフラグを初期化する必要がある．差分フラグ信号を初期化する差分フラグリセット信号，rstΔ_j，も SSC から生成される．提案 DSSA プロセッサの搭載スピン数は 2,048 スピンであり，プロセッサ外部でアニーリング結果を確認するためにはスピンの状態を外部に出力しなければならない．しかし，高速で動作するプロセッサから 2,048 ビットのスピン状態を外部に出力するためには，通信のためのコストが必要となり，通信のインタフェースも考慮する必要がある．提案 DSSA プロセッサでは，2,048 ビットのスピン状態を出力する代わりに，差分スピンのアドレス (状態が変化したスピンの番号) のみを外部に出力する．Section 4.3 で述べたように，プロセッサの spin-gate の初期値は全て 0 であるため，プロセッサ外部でもアニーリング処理が始まる時のスピン状態は既知である．プロセッサからは状態が変化したスピンの番号のみを出力することで，全体のスピン状態を出力することな

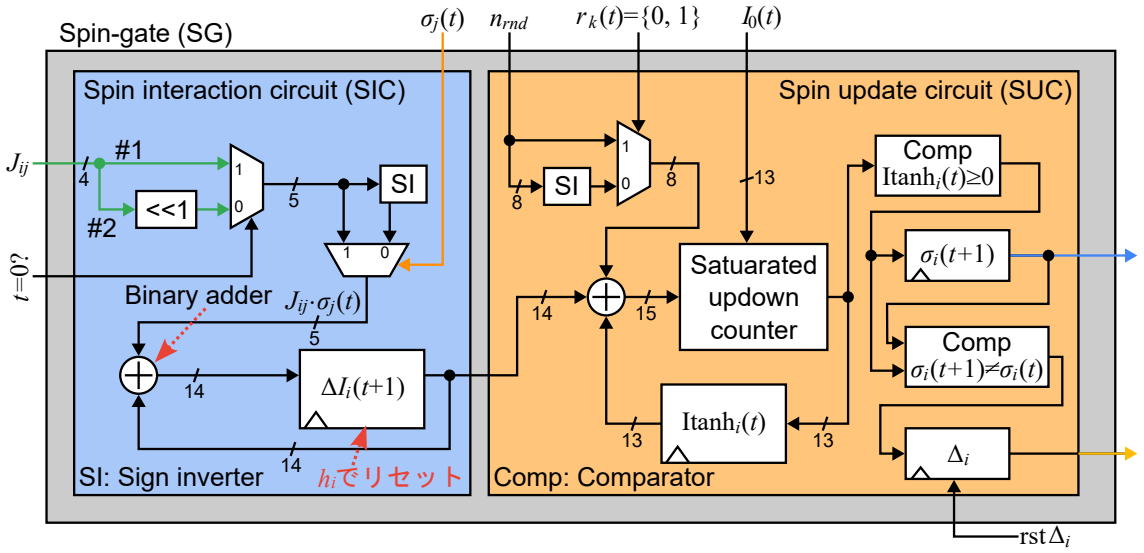


図 5.6: DSSA プロセッサの spin-gate の構造．DSSA 法の spin-gate はスピン相互作用を計算する SIC，スピン状態を更新する SUC より構成される．

く，外部にアニーリング結果を伝送できる．

Fig. 5.6 は DSSA プロセッサに搭載された spin-gate の構造を示す．一つの spin-gate は一つのイジングスピンに対応し，提案プロセッサには 2,048 個の spin-gate が SGA モジュールを構成する．Spin-gate はスピンの相互作用を計算する spin interation circuit (SIC) とスピン状態を更新する spin update circuit (SUC) から構成される．SIC は WM から読み出された相互作用の重み，J_{ij}，SSC から選択された差分スピンの状態，$\sigma_j(t)$，を用いて相互作用を逐次的に計算する．DSSA 法の動作はアニーリングステップ $t = 0$ の場合に Eq. (4.1a) に基づいき，J_{ij} の元の値をそのまま利用してスピンの相互作用を計算する．一方，$t > 0$ のアニーリングステップでのスピン相互作用の計算は Eq. (4.11a) に基づいくため，2 倍された重みである $2J_{ij}$ を用いて差分スピンによる $\Delta I_i(t+1)$ を計算する．従って，SIC に入力された J_{ij} はアニーリングステップ，t，によって，$t = 0$ の場合に#1 のパスを通り最初のスピン状態計算に用いられる．$t \neq 0$ の場合は，入力された J_{ij} はシフト演算され，#2 のパスを通り，差分スピンを用いたスピン動作に用いられる．$\sigma_j(t)$ と J_{ij} (または $2J_{ij}$) の乗算はマルチプレクサより演算さる．DSSA 法はスピン相互作用の計算がシリアル化されているため，計算途中の結果を保存する必要があり，SIC は計算途中の $\Delta I_i(t+1)$ を保持するレジスタが含まれている．イジングモデルの重み，J_{ij}，は DSSA プロセッサの WM に格納されるが，バイアス，h_i，のメモリは存在しない．提案プロセッサでは h_i をメモリを用いて格納する代わりに，$\Delta I_i(t+1)$ のレジスタを h_i に初期化することで，バイアスの計算を行う．$\Delta I_i(t+1)$ のレジスタをアニーリング処理が始まる前に h_i で初期化することで，バイナリアダーよりバイアスを加算できる．

SIC で計算された $\Delta I_i(t+1)$ ($t = 0$ の時は $I_i(1)$) は SUC に入力され，乱数信号，$n_{rnd} \cdot r_i(t)$，以前のアニーリングステップでの Itanh$_i(t)$ と加算される．ここで，Itanh$_i(t)$ はレジスタに保持され，そのレジスタはアニーリング処理が始まる前に 0 で初期化される．加算の結果は saturated updown counter に入力され，Eq. (4.11b) に従って Itanh$_i(t+1)$ を計算する．Eq. (4.11b) の計算は SC の近似より FSM

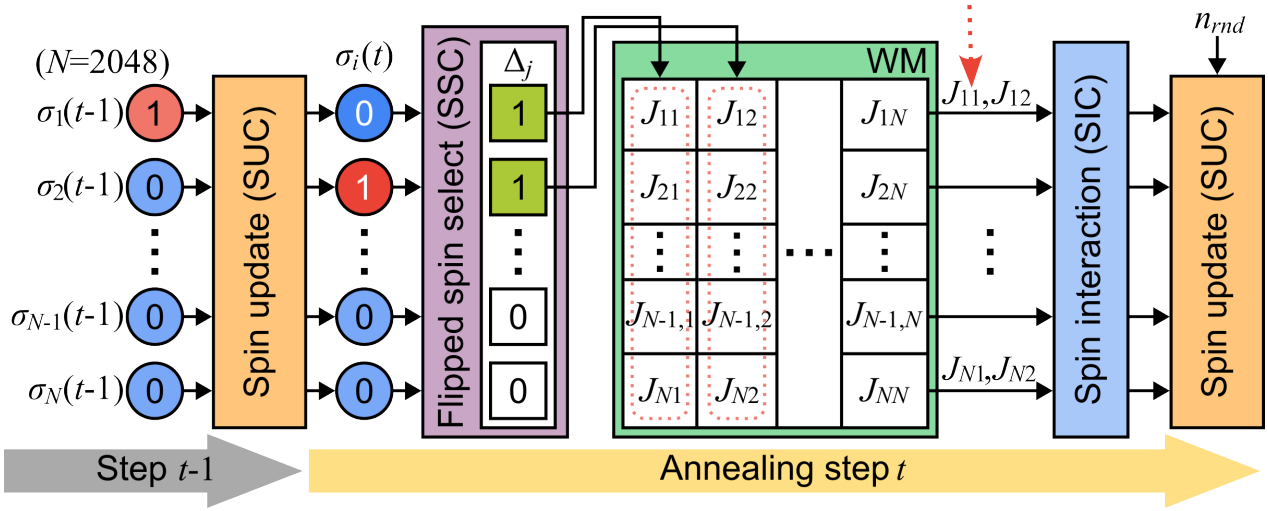


図 5.7: 差分スピンが存在する場合の DSSA プロセッサの動作及び内部信号のデータパス．

で計算され，saturated updown counter は FSM の計算を行う．この際，FSM の状態数は逆温度パラメータ，$I_0(t)$，によって決まり，$I_0(t)$ は DSSA プロセッサの scheduler から制御される．Saturated updown counter より計算された $\mathrm{Itanh}_i(t+1)$ を用いてスピン状態，$\sigma_i(t+1)$，及び差分フラグ，$\Delta_i(t)$ を計算する．Eq. (4.11c) に従って，比較器で $\mathrm{Itanh}_i(t+1)$ を 0 と比較して $\sigma_i(t+1)$ を決めてレジスタにその値を保持すると同時に，差分フラグを決めるための比較器に入力される．差分フラグを計算する比較器は，現在のスピンの状態とレジスタに保持されている以前のスピン状態を比較し，スピンの状態が変化していれば $\Delta_i(t)=1$ に，変化していなければ $\Delta_i(t)=0$ とする．スピン相互作用計算に用いられるスピンの選択は差分フラグより左右される．DSSA プロセッサの最初のアニーリングステップ $(t=0)$ では全てのスピン相互作用を計算しなければならない．提案プロセッサでは，$\Delta_i(t)$ のレジスタをアニーリング処理が開始する前に 1 に初期化することで，最初のアニーリングステップで全てのスピン相互作用の計算を実行する．

5.3　プロセッサの動作

続いては提案 DSSA プロセッサのアニーリング処理の動作についてプロセッサを構成するモジュール間のデータパスと動作のタイミングチャートを用いて述べる．Section 4.3 ではアニーリングステップ t によって DSSA 法のスピン動作を分類した．本章では差分スピンの有無を基準で DSSA プロセッサの動作を分類する．

Fig. 5.7 はアニーリングステップ t において，差分スピンが存在する場合の DSSA プロセッサ内部信号のデータパスを表す．Fig. 5.7 の例では，以前のアニーリングステップ $t-1$ で，スピン 1 の状態，σ_1，が 1 から 0 へ，スピン 2 の状態，σ_2，が 0 から 1 へ状態が変化し，差分スピンが二つの場合を示す．σ_1 と σ_2 に対応する spin-gate ではそれぞれの差分フラグ $\Delta_1(t)$ と $\Delta_2(t)$ が 1 になり，他のスピンの差分フラグは 0 になる．SSC では $\Delta_1(t)$ のスピンアドレス，$(00000000000)_2$，を priority encoder から生成し，WM のアドレスとして入力する．同時の SSC 内部のマルチプレクサにもスピン

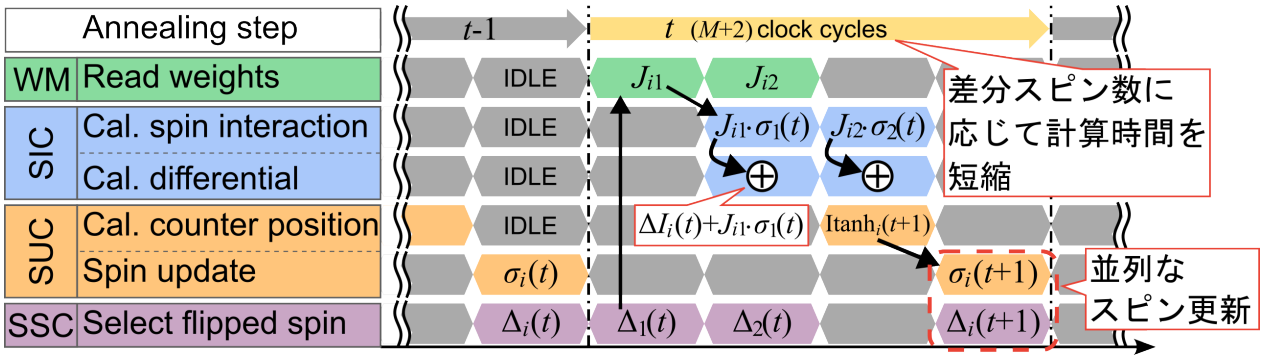


図 5.8: 差分スピンが存在する場合の DSSA プロセッサのクロックサイクルに対するタイミングチャート．

アドレスを入力し，σ_1 によるスピン相互作用を計算するあためにに $\sigma_1(t)$ を選択して SGA に入力する．SSC から WM に送られた σ_1 のスピンアドレスに従って，WM では各スピンと σ_1 の相互作用の重み，$\{J_{11}, J_{21}, J_{31}, ..., J_{N-1\,1}, J_{N1}\}$，を読み出す．読み出された σ_1 の重み，J_{i1}，は差分スピンの状態，$\sigma_1(t)$，と SGA に入力され，spin-gate の SIC にてスピン相互作用を計算する．同様に，σ_2 に関しても，SSC からスピンアドレスの生成後，WM から σ_2 の重み，J_{i2}，を読み出し，SGA に入力することで，差分スピン，σ_2，から生じるスピン相互作用を計算する．差分スピン，σ_1 及び σ_2，によるスピン相互作用の計算が終了すると，SIC から SUC へ $\mathrm{Itanh}_i(t+1)$ が入力され，SUC でスピンの状態更新する．DSSA の逐次的なスピン相互作用計算での各モジュールの動作はパイプライン化されていて，効率的に逐次処理が可能である．Fig. 5.8 は Fig. 5.7 で示した二つの差分スピンが存在する場合の DSSA プロセッサのクロックサイクルに対する各モジュールのタイミングチャートである．t アニーリングステップの処理が始まる際，SSC からは $\Delta_1(t)$ に応じて σ_1 のスピンアドレスを生成して WM に入力する．WM は SRAM であり，読み出しの遅延は 1 クロックサイクルであるため，アドレス生成から 1 クロックサイクル後に対応する重み，J_{i1}，を SIC に送る．SIC のスピン相互作用計算の回路は組合せ回路であるため，J_{i1} の入力と同時にスピン相互作用，$2J_{ij} \cdot \sigma_1(t)$，を計算できる．SIC での $2J_{ij} \cdot \sigma_1(t)$ の計算と同時に，SSC では $\Delta_2(t)$ に応じて σ_2 のスピンアドレスを WM に送り，WM から J_{i2} を読み出す．σ_1 の相互作用計算終了後，$\Delta I_i(t) + 2J_{ij} \cdot \sigma_1(t)$ の計算結果はレジスタに保持され，σ_2 の相互作用計算に用いれる．SIC で $\Delta I_i(t+1)$ の計算が終わると同時に，SUC では $\mathrm{Itanh}_i(t)$ と $\Delta I_i(t+1)$ の加算を行われ，加算の際には乱数信号も同時に加算される．加算の結果は saturated updown counter を通して，比較器に入力され，新しいスピン状態，$\sigma_i(t+1)$，及び $\Delta_i(t+1)$ を求める．SUC の入力から比較器までは全て組合せ回路であり，スピン相互作用計算の終了と同時に動作する．最後に，全てのスピンの情報を並列に 1 クロックサイクルで更新することで，t アニーリングステップの処理が完了する．このように，差分スピンが存在する場合，DSSA プロセッサはそれぞれのモジュールの動作をパイプライン化することで，メモリの読み出しやスピン状態更新の遅延を最小化する．差分スピンが存在する場合の 1 アニーリングステップの計算に必要なクロックサイクル数は，差分スピンの数を M とすると，アニーリングステップ開始時のメモリ読み出しの 1 クロック，スピン状態更新の 1 クロック，スピン相互作用計算に M クロックサイクルが必要であるため，$M+2$ クロックサイクルである．

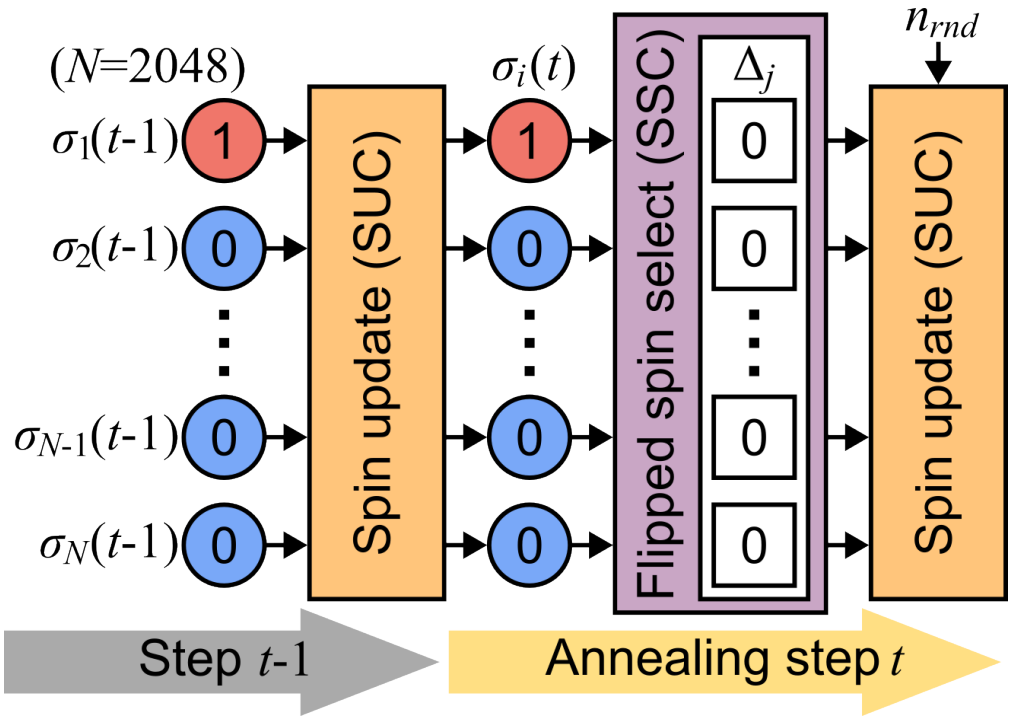


図 5.9: 差分スピンが存在しない場合の DSSA プロセッサの動作及び内部信号のデータパス.

$t = 0$ の最初のアニーリングステップでは，全ての $\Delta_i(0)$ は 1 に初期化されている．従って，上述のような動作を全てのスピンに対して行うため，差分スピン数，M，は提案プロセッサの搭載スピン数，$N = 2,048$，と等しい．すなわち，$t = 0$ のアニーリングステップの計算クロックサイクル数は $(2,048 + 2) = 2,050$ クロックサイクルである．

次は差分スピンが存在しない場合のプロセッサの内部信号のデータパス及びタイミングチャートを Fig. 5.9 及び Fig. 5.10 に示す．差分スピンがないと全ての差分フラグは 0 であり，SSC は動作しない．SSC からアドレスが出力されないため，WM も動作しないため，メモリの読み出しは発生しない．しかし，DSSA 法のスピンはスピンの相互作用だけではなく，乱数信号からも影響を受けて状態は変化する．従って，差分スピンがない場合でも乱数信号は変化するため，乱数によるスピン状態の更新が必要であり，乱数信号と $\mathrm{Itanh}_i(t)$ の加算が行われる．この乱数信号の加算と saturated updown counter の演算，SUC の比較器の演算に 1 クロックサイクルがかかり，演算の結果から各スピンの状態の更新が 1 クロックがかかる．すなわち，差分スピンがない場合でも，1 アニーリングステップの計算には 2 クロックが必要である．差分スピン数，M，は 0 とすると，差分スピンがある場合のステップ当たりのクロックサイクル数の計算式，$M + 2$，から差分スピンがない場合のクロックサイクル数を計算できる．このように，提案 DSSA プロセッサの 1 アニーリングステップの計算に必要なクロックサイクル数は，各ステップでの差分スピン数，M，に応じて変化し，M に応じて計算時間が短縮される．

5.4　ASIC 実装環境

提案 DSSA プロセッサは ASIC 実装を想定して設計であり，プロセッサの動作検証及び性能評価

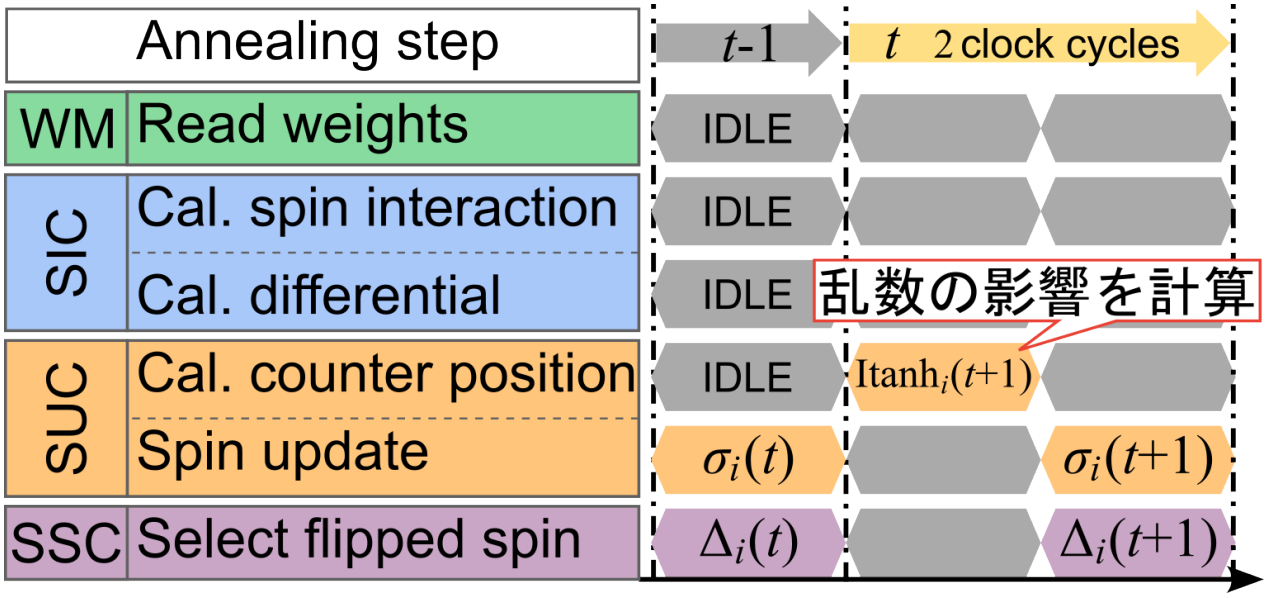


図 5.10: 差分スピンが存在しない場合の DSSA プロセッサのクロックサイクルに対するタイミングチャート.

は ASIC 実装された DSSA プロセッサを用いる予定である．ASIC 設計には様々な electonic design automation (EDA) ツールが必要であり，提案プロセッサの ASIC 設計の環境について述べる．提案 DSSA プロセッサの RTL 設計は System Verilog を用いて設計される．設計された RTL デザインの論理合成は Synopsys 社の Design Compiler U-2022.12 を用いる．提案プロセッサの実装に使用されるプロセスノードは TSMC 28 nm であり，完全ディジタル回路であるため，TSMC 28 nm プロセスの standard cell や I/O のライブラリを用いて論理合成される．論理合成されたゲートレベルデザインの配置配線には Cadence 社の自動配置配線ツールである Innovus v21.12 を使用する．Innovus を用いて，SRAM マクロの配置，I/O cell の配置，ゲートレベルの自動配置配線を行う．ASIC チップのチップ外部との通信は I/O cell と pad が必要である．Pad の配置には Cadence 社の Virtuoso 6.1.8 を用いて行う．Pad などを含んだ最終デザインは Virtuoso より GDS フォーマットで出力される．ASIC 設計において，プロセスノードの design rule を違反をなくすことはとても重要であり，design rule check (DRC) や layout versus schematic (LVS) よりルール違反を検出する．設計されたレイアウトデザインの DRC 及び LVS などの検証には Mentor 社の Calibre v2022.2_38.20 を用いる．DRC に検出したルール違反箇所の修正は Innovus 及び Virtuoso を用いて行う．上述の EDA ツールの実行マシーンの CPU は Intel Xeon Gold 6252，CPU の動作周波数は 2.1 GHz であり，マシーンの OS は CentOS 7 である．

5.5　DSSA プロセッサの ASIC 実装

Section 5.4 で述べた ASIC 設計環境を用いて行った大規模完全グラフイジングモデル向け DSSA プロセッサの ASIC 実装について述べる．しかし，ASIC 実装結果について述べる前に，System Verilog で設計した RTL デザインの検証を FPGA を用いて行う．DSSA プロセッサの FPGA 実装に用いられる

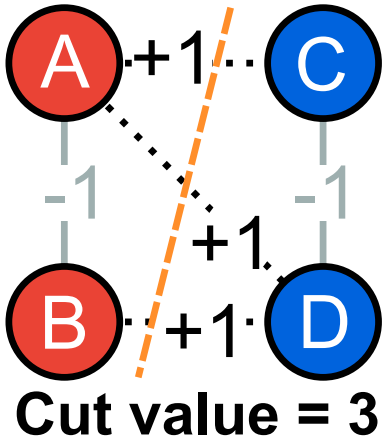
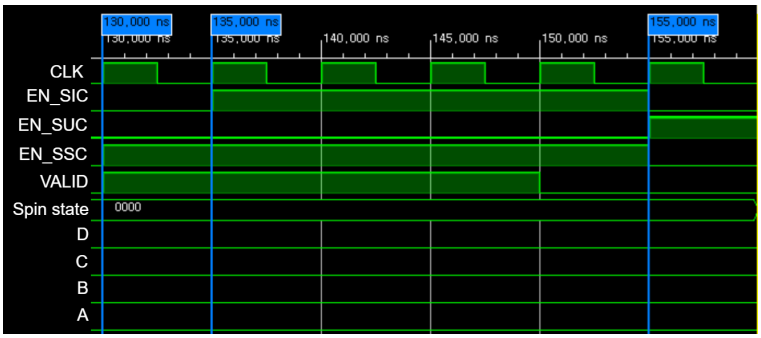


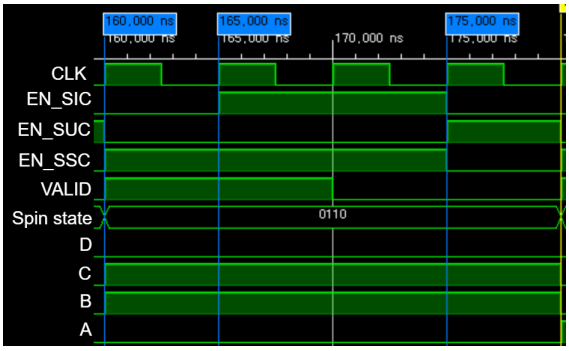
図 5.11: FPGA 実装の 4 スピン DSSA プロセッサのベンチマーク問題．4 ノードの MAXCUT 問題を用いて動作検証を行う．

FPGA ボードは Xilinx Kintex 7 が搭載された Digilent Genesys 2 を用いる．RTL デザインの FPGA 向け論理合成及び配置配線には Xlinx Vivado 2021.2 を使用する．FPGA を用いてプロセッサの設計を検証するにあたって，提案プロセッサの設計からいくつか変更が必要となる．提案プロセッサでイジングモデルの重みは SRAM を用いて格納されるが，FPGA では SRAM を使用することができない．一般的に，FPGA 上でメモリを使用する際には BRAM を用いるが，Kintex 7 の BRAM は読み出しの遅延が 2 クロックサイクルであり，SRAM の 1 クロックサイクルより遅い．従って，SRAM からなる WM を flip flops を用いて，SRAM の動作を模擬することで，重みを格納する．また，提案 DSSA プロセッサは 2,048 個の完全接続のスピンを搭載していが，FPGA 実装の目的はあくまでも RTL 設計の検証であるため，搭載スピン数は 4 個の小規模の DSSA プロセッサの動作を検証する．また，四つの spin-gate のスピン状態の出力，$\sigma_i(t+1)$，を FPGA ボード上の LED に直接繋ぐことで，スピンが正しい状態へ収束するかを確認する．

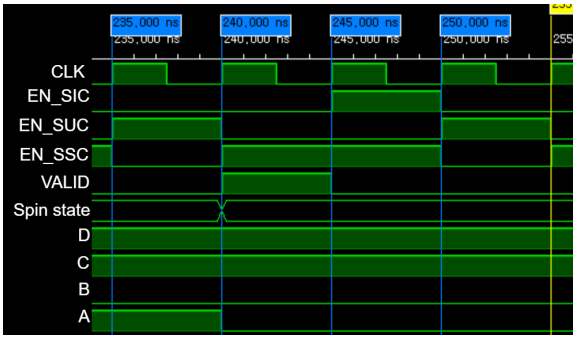
FPGA に実装された 4 スピン DSSA プロセッサの動作確認には Fig. 5.11 のような 4 ノードの MAXCUT 問題を用いる．ベンチマーク問題の最適解はノード A と B の部分グラフ，ノード C と D の部分グラフに分割された場合であり，この場合のカット値は 3 である．Figs. 5.12a to 5.12c はそれぞれ 4 スピン DSSA プロセッサの RTL シミュレーションの最初のアニーリングステップ ($t = 0$)，$t = 1$ のアニーリングステップ，最後のアニーリングステップでの信号の波形を表す．Fig. 5.12 で表す信号はプロセッサのクロック信号 (CLK)，spin-gate の SIC の動作信号 (EN_SIC)，SUC の動作信号 (EN_SUC)，SSC の動作信号 (EN_SSC)，差分スピンの検出信号 (VALID)，Fig. 5.11 の各ノードに対応するスピン状態 (A，B，C，D) である．$t = 0$ のアニーリングステップでは，全てのスピン相互作用を逐次的に計算しており，メモリからの重みの読み出しのクロックを含めて 5 クロック間スピン相互作用を計算する．スピン相互作用計算の終了後，1 クロックで 4 つのスピンの状態を更新し，1 アニーリングステップの計算に 6 クロックサイクルがかかった．続いて，$t = 1$ のステップではスピン C と D が差分スピンであり，Fig. 5.12b のように計算クロックサイクルは $2 + 2 = 4$ クロックである．最後のステップでは，スピン A が差分スピンであり，3 クロックサイクルの計算が行われる．また，Fig. 5.12c からわかるように，ス



(a)



(b)



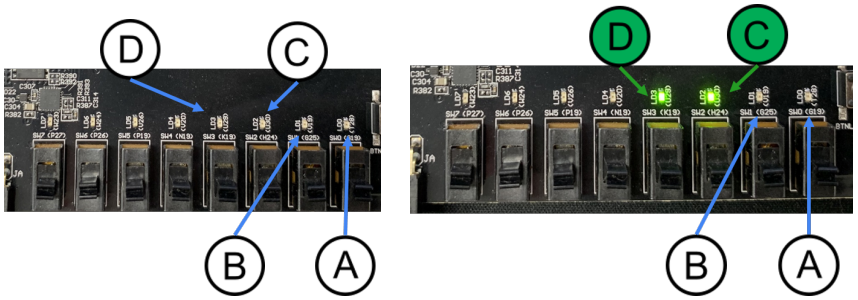
(c)

図 5.12: FPGA ベース 4 スピン DSSA プロセッサの RTL シミュレーションの動作波形．(a) $t = 0$ の最初のアニーリングステップの動作波形．(b) $t = 1$ のアニーリングステップの動作波形．(c) アニーリング処理最後のステップの動作波形．

ピン A と B の状態が 0，スピン C と D の状態が 1 であり，与えられたベンチマーク問題の最適を正しく探索できた．

Fig. 5.13 は四つの spin-gate から出力されるスピンの状態，$\sigma_i(t+1)$，を表示する FPGA ボード上の LED を表す．DSSA プロセッサのスピンの初期値は全て 0 であるため，Fig. 5.13a のように全ての LED が消灯されている．その後，与えられたベンチマーク問題に対してアニーリング処理を行うと，それぞれのスピンの状態は最適な状態へ収束し，問題の最適解を探索する．Fig. 5.13b はアニーリング処理終了後の各スピンの状態を表示している LED を表す．ベンチマーク問題の最適解は (A，B) と (C，D) の二つの部分グラフに分割される分割が最適解である．アニーリング結果，スピン A と B の LED は消灯され，スピン C と D の LED が点灯されており，全てのスピンが正しい状態へ収束したことを確認できる．このように，FPGA を用いた小規模の提案 DSSA プロセッサの RTL デザインが正しく動作することを確認できる．

TSMC 28 nm プロセスを用いて実装された提案 DSSA プロセッサのレイアウトデザインは Fig. 5.14 のようである．提案プロセッサのダイサイズは 3 mm × 4 mm であり，チップ面積は 12 µm^2 である．論理合成後の gate count は 8,498,667 であり，配置配線後の total utilization は 35.7% である．提案 DSSA プロセッサは，4 ビットの $2,048 \times 2,048$ 個をメモリモジュールである WM に格納し，WM の容量は 16 Mb である．TSMC 28 nm プロセスに含まれる SRAM マクロの中で，2,048 ワードの 128 ビッ



(a)　(b)

図 5.13: FPGA に実装された 4 スピン DSSA プロセッサの動作検証の結果を表す LED．(a) アニーリング処理開始以前の初期のスピン状態を表す LED．(b) アニーリング処理終了後のスピン状態を表す LED．

ト幅の物を使う．従って，WM モジュールは 2,048 ワードの 128 ビット幅のマクロを 64 個を組合せて構成される．チップの外部との通信のための IO の個数は，四つの電源ピン，四つの接地ピン，四つの IO 電源ピン，四つの IO 接地ピンを含めて 118 個である．プロセッサの電源電圧は 0.9V であり，IO の電源電圧は 1.8V である．論理合成及び配置配線時の動作周波数の制限は 500 MHz であるが，配置配線で生じる遅延の増加に備えて，論理合成時には 600 MHz の周波数制約を用いた．500 MHz の制約での論理合成後の消費電力は 316.1 mW である．

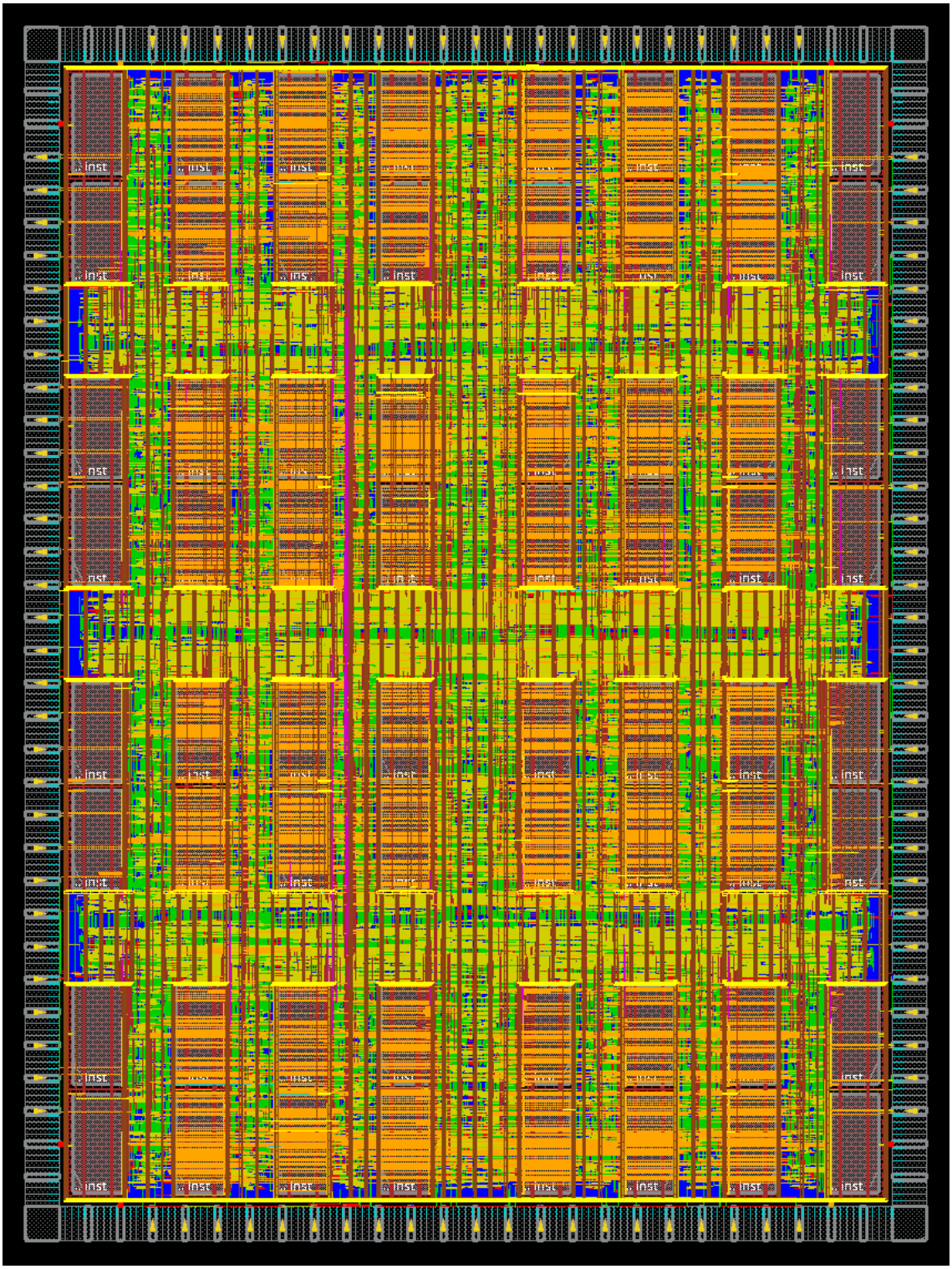


図 5.14: 完全接続の 2,048 スピンが搭載された DSSA プロセッサのレイアウトデザイン.

表 5.2: 従来のアニーリングプロセッサと DSSA プロセッサの性能比較.

	ISSCC‘21 [25]	JSSC‘21 [26]	ISSCC‘23 [29]	DSSA
プロセスノード	TSMC 40 nm	TSMC 65 nm	TSMC 40 nm	TSMC 28 nm
アルゴリズム	SA	SA (SCA)	Metamorphic	DSSA
チップ面積 [mm^2]	24	12	9	12
搭載スピン数	16k (144k w/ 9-multi chips)	512	512 (2,048 w/ 4-multi chips)	2,048
スピン接続構造	King‘s graph	完全グラフ	完全グラフ	完全グラフ
スピンの接続数	8	511	511	2,047
動作周波数 [MHz]	100	320	134 - 336	500
電源電圧 [V]	1.1	0.75 - 1.1	0.8 - 1.1	0.9
消費電力 [mW]	N/A	649 @ 320 MHz	151.6 - 474.9 @ 320 MHz	316.1 @ 500 MHz
スピン当たりの消費電力 [mW]	N/A	1.27	0.29 - 0.93	0.15

5.6　性能比較

提案 DSSA プロセッサの性能を state-of-the-art の研究を含めた ASIC ベースアニーリングプロセッサに関する先行研究 [25, 26, 29] と比較する．Table 5.2 は ASIC ベースの従来アニーリングプロセッサと提案 DSSA プロセッサの面積，搭載スピン数及びスピン接続構造，動作周波数，消費電力などの比較を示す．ISSCC‘21 [25] は SA アルゴリズムに基づいたイジングモデル向けアニーリングプロセッサに関する研究であり，単一チップに 16k のスピンを搭載し大規模なプロセッサである．また，最大 9 個のマルチチップを用いて 144k という膨大な数のスピンを扱うことも可能である．しかし，スピンの接続構造が隣接の 8 個のスピンと接続する king‘s graph で，スピン当たりの接続数は 8 であり，対応できるイジングモデルの構造に制限がある．JSSC‘21 [26] は完全グラフイジングモデルに対応可能なアニーリングプロセッサであり，DSSA 法のようにスピン更新を並列に行うアニーリングアルゴリズムである stochasitc cellular automata annealing (SCA) に基づく．SCA はイジングモデルの並列サンプリング法である probabilistic cellular automata (PCA) sampling [56] に基づいて，従来 SA 法でのできないスピン状態更新を可能とする．JSSC‘21 のアニーリングプロセッサは STATICA といい，TSMC 65 nm を用いた完全接続の 512 スピンアニーリングプロセッサである．STATICA のスピン当たりの接続数は 511 で，完全グラフ構造のイジングモデルに対応可能であるが，搭載スピン数は 512 個であるため対応可能なイジングモデルの問題サイズに限界がある．最後に ISSCC‘23 で発表された Amorphica は完全接続の 2,048 スピンイジングモデルにも対応可能なアニーリングプロセッサであり，ASIC ベースの state-of-the-art のプロセッサである [29]．しかし，Amorphica は単一チップに 512 スピンだけ搭載しており，スピン当

たりの接続数は 511 である．Amorphica は単一チップでは四つのレプリカ，または四つのマルチチップを活用することで完全接続の 2,048 スピンを実現する．Amorphica のアニーリングアルゴリズムは従来 SA や STATICA の SCA，また，digital annealing (DA)，ratio-controlled parallel annealing (RCA) の四つのアルゴリズムから選択できる．DA はイジングモデルの全てのスピンを確認し，更新可能なスピンからランダムに一つを選んで更新するアニーリングアルゴリズムであり [57]，RCA は DSSA のように並列にスピン状態の更新の際，実際に更新されるスピンの比率を制御するアニーリングアルゴリズムである [29]．また，イジングモデルの重みは，提案 DSSA プロセッサのように，チップ内部の SRAM に格納する．Amorphica の重みのビット幅は 8 ビットであり，2,048 スピンの重みを全て格納するためには 32 Mb の容量が必要である．しかし，重みを格納する SRAM の容量は 8 Mb であり，全ての重みを格納することはできない．従って，Amorphica を用いて完全グラフの 2,048 スピンイジングモデルを解くためにはメモリの書き換えやマルチチップ間の通信のコストが必要であり，単一チップだけでは 2,048 スピンのモデルを扱うことはできない．一方，提案 DSSA プロセッサは単一チップに完全接続の 2,048 スピンを全て搭載しており，スピン当たりの接続数も 2,047 で全てのスピンが互いに接続するためマルチチップやレプリカなしでも state-of-the-art より大規模のイジングモデルに対応可能である．また，完全接続 2,048 スピンモデルの重みを全てチップ内部のメモリに格納できるため，アニーリング処理途中のメモリの書き換えなども不要である．

それぞれのアニーリングプロセッサの動作周波数は ISSCC‘21 が 100 MHz，STATICA が 320 MHz であり，Amorphica の動作周波数は電源電圧と用いるアルゴリズムによって代わり，最小 134 MHz から最大 336 MHz である．一方，提案 DSSA プロセッサの動作周波数は，配置配線後の結果であはあるが，500 MHz の動作周波数で動作可能であり，プロセスノードを考慮しても従来のアニーリングプロセッサより高速な動作速度を示す．動作電力に関しては STATICA が 320 MHz の動作周波数で 649 mW，Amorphica は 320 MHz で用いるアルゴリズムによって最小 151.6 mW から最大 474.9 mW であり，ISSCC‘21 はプロセッサの消費電力は記載していない．提案 DSSA プロセッサの論理合成後の動作電力は 500 MHz の動作周波数で 316.1 mW であり，最も少ない電力を消費する．加えて，スピン以外の要素の電力を考慮せずスピン当たりの消費電力を計算すると，STATICA が 1.27 mW，Amorphica が最小 0.29 mW から最大 0.93 mW である．一方，提案 DSSA プロセッサのスピンの消費電力は 0.15 mW であり，STATICA より約 8.5 倍，Amorphica より約 1.9 から 6.2 倍の電力効率を達成した．

次は DSSA プロセッサのアニーリング性能を STATICA 及び GPU 実装した従来 SSA 法と比較する．GPU ベース従来 SSA 法は Python より記述され，Python の GPU をい用いた行列演算ライブラリである CuPy [58] を使い，従来 SSA 法のアニーリング処理を GPU で加速する．従来 SSA 法の Python プログラムの実行に用いる GPU は 10,496 CUDA コアーと 24 GB のメモリを持つ NVIDIA GeForce RTX 3090 である．JSSC‘21 [26] では 512 スピン STATICA の性能から想定した 2,000 スピン STATICA の予想性能と，そのシミュレーション結果について述べている．2,000 スピン STATICA のシミュレーションには 2,000 スピン完全グラフ MAXCUT 問題である K2000 を用いる．Table 5.3 は従来 SSA 法の GPU 実装，2,000 スピンの STATICA，提案 DSSA プロセッサの K2000 問題における性能比較である．アニーリング性能の比較の際，それぞれ手法のアニーリングアルゴリズムが異なり，アニーリング処理の計算時間やアニーリングステップ数なども一定にするとは難しい．従って，それぞれの手法の計算速度やアニーリング精度などのアニーリング性能を比較することは困難である．このような，アニーリン

表 5.3: 2,000 ノード完全グラフ MAXCUT 問題である K2000 問題におけるアニーリングプロセッサの及び GPU ベース従来 SSA 法の性能比較.

	従来 SSA 法 (GPU)	STATICA[1]	DSSA[2]
消費電力 [mW]	52.8×10^3	2.0×10^3	316.1
アニーリング処理時間 T [ms]	499.88	0.48	2.30
$P_a(T)$	0.45	0.77	0.98
$tts(0.99)$ [ms]	3850.6	1.5	2.7
$tts(0.99)$ の消費エネルギー [mJ]	203.3×10^3	3.0	0.86

[1] 512 スピン STATICA から推定した 2,000 スピン STATICA による結果

[2] 提案プロセッサ及びシミュレーションより推定した結果

グ精度を比較するため，一般的に用いられる評価指標が time-to-solution (TTS) である [59, 60]．TTS は QA [11] のアニーリング性能を従来 SA 法と比較するために提案された評価指標であり，ある問題の近似解をある確率，P，で得られるために必要な計算時間を意味である．例えば，$P = 0.99$ である場合，$tts(0.99)$ は近似解を 99% の確率で得るために必要なアニーリング処理の計算時間である．近似解を見つける目標確率，P，に対しる TTS，$tts(P)$ は:

$$tts(P) = T \cdot \frac{\ln(1-P)}{\ln(1-P_a(T))}, \tag{5.1}$$

であり，ここで，T は 1 回のアニーリング処理の計算時間，$P_a(T)$ は T 時間のアニーリング処理から近似解を得る確率である．また，Table 5.3 では $tts(0.99)$ の消費エネルギーも比較している．$tts(0.99)$ の消費エネルギーは $tts(0.99)$ の動作時間で消費するエネルギーであり，以下のように求められる．

$$(tts(0.99) \text{ の消費エネルギー}) = tts(0.99) \times E, \tag{5.2}$$

ここで，E はアニーリングプロセッサの消費電力である．

JSSC‘21 では K2000 問題の近似解を 30,000 として評価を行い，0.48 ms の計算時間で，目標の近似解を得る確率，$P_a(T)$ は 0.77 であった．目標価格率を 0.99 とすると，2,000 スピン STATICA の $tts(0.99)$ は Eq. (5.1) より 1.5 ms である．また，512 スピン STATICA から推定した 2,000 スピン STATICA の消費電力は 2.0 W である．つまり，K2000 問題の近似解を 99% の確率で達成するための STATICA の消費エネルギーは 3.0 mJ である．従来 SSA 法の GPU 実装は，499.88 ms のアニーリング処理で 30,000 の近似解を 0.45 の確率で得ることができ，この時の $tts(0.99)$ は 3850.6 ms である．この際の従来 SSA 法の GPU 実装の消費エネルギーは 203.3×10^3 mJ であり，最も多くのエネルギーを消費する．提案 DSSA プロセッサの K2000 に対するアニーリング処理の計算時間及び精度はチップの動作周波数とシミュレーションより求めたものである．K2000 問題に対して，2.3 ms のアニーリング処理を行った時，DSSA プロセッサは 98% の確率での近似解に収束した．この結果から求めた提案 DSSA プロセッサの $tts(0.99)$ は 2.7 ms であり，$tts(0.99)$ における消費エネルギーは 0.86 mJ である．すなわち，K2000 問題の近似解を 99% の確率で見つけるために，提案 DSSA プロセッサは GPU 実装より約 236,395 倍，2,000 スピン STATICA より約 3.5 倍のエネルギー効率を示した．

5.7　むすび

本章では大規模な完全グラフイジングモデル向けアニーリングプロセッサ実装するためのハードウェアアルゴリズムである DSSA 法に基づいた，DSSA プロセッサのについて述べた．提案 DSSA プロセッサは，スピン接続のシリアル化による小面積化及び差分スピンを用いたスピン相互作用の高速化より，完全接続の 2,048 スピンを単一チップに搭載した．DSSA プロセッサは DSSA の spin-gate からなる SGA をメインコンポネントとして，イジングモデルの重みのためのメモリを含めて全てのコンポネントを含めている．また，時間分割されたスピン相互作用の計算を行うモジュールの動作をパイプライン化することで，効率的に DSSA 法のアニーリング動作を行う．提案 DSSA プロセッサは ASIC 実装を目的として設計されており，ASIC 設計に向けて FPGA を用いた RTL デザインの検証を行った．プロセッサのダイ面積は 12 mm^2 であり，動作周波数は 500 MHz，消費電力は 316.6 mW である．提案 DSSA の搭載スピン数は 2,048 スピンであり，state-of-the-art のアニーリングプロセッサと比較しても，最も多くのスピンを搭載しており，スピン当たりの動作電力も約 6.2 倍低電力であった．2,000 スピンの MAXCUT 問題において，提案 DSSA プロセッサは従来のアニーリングプロセッサより省エネルギーで与えられた問題の近似解を高い確率で達成できる．

第 6 章

結言

本論文ではイジングスピンの差分を用いた differential stochastic simulated annealing (DSSA) 法及び，その DSSA 法に基づいた大規模な完全グラフイジングモデル向けアニーリングプロセッサの実装について述べた．組合せ最適化問題 (COP) の表す数理モデルの一つであるイジングモデルはバイナリの状態を持つイジングスピンとそれらの相互作用からなるネットワークモデルである．イジングモデル向けのアニーリングプロセッサの実装において，モデルの全てのスピンが互いに接続する完全グラフイジングモデルに対応可能なアーキテクチャは，COP の社会応用においてもとても重要な課題である．DSSA 法は，stochastic computing (SC) に基づいたスピン動作の近似及び，スピン相互作用の計算を時間分割することでシリアル化し，イジングスピンの小面積な実装を達成した．また，スピン状態の差分を用いた相互作用の計算よりシリアル化から生じる計算時間増加を抑え，大規模完全グラフイジングモデルにおけるアニーリング処理を高速化した．このように DSSA 法は大規模完全グラフイジングモデルに効果的なハードウェアアルゴリズムであり，大規模完全グラフ向けアニーリングプロセッサの実装の基盤を構築した．DSSA 法に基づいた，DSSA アニーリングプロセッサは互いに完全接続する 2,048 スピンを一つのチップに搭載しており，ASIC ベースの従来アニーリングプロセッサより大規模なイジングモデルにも対応可能である．DSSA プロセッサは 2,000 ノードからなる MAXCUT 問題に対して，従来アニーリングプロセッサより約 3.5 倍のエネルギー効率を示し，大規模な完全グラフイジングモデルにおいてその有効性を示した．

第 2 章では，NP 困難問題に属する COP の最適解を効率的に探索可能な確率的アルゴリズムであるシミュレーテッドアニーリング (SA) の原理及びイジングモデルにおける SA の動作について概説した．SA 法は解の組合せをランダムに変化させながら，与えられた問題のコストを最小にする解の組合せを探索する．探索の際に，改悪の解を確率的に受け入れることで局所解から脱出できるため，効率的に COP を解くことができる．一方，イジングモデルに SA 法を適用する場合，同時に更新できるスピン数は一つだけであるため，イジングモデルのサイズの増加に応じて SA 法の計算時間は指数関数的に増加する．このような問題のサイズに対する計算時間増加問題を解決するために，量子アニーリングなど，新しいコンピューティング方式に基づいたアニーリングアルゴリズムが研究されており，第 2 章では，確率的演算方式である SC に基づいた stochastic simulated annealing (SSA) について概説した．SSA 法はボルツマンマシンを基盤として，同時に複数のスピン状態することで，高速にイジングモデルの最適化を探索できる．また，SC に基づいた SSA 法のスピン動作及び，SSA 法の逆温度パラメータの制御などにつ

いて詳しく述べた．最後に，800 ノードからなる COP に対して SSA 法の動作を検証し，計算時間やアニーリング精度などの性能を従来 SA 法と比較した．

第 3 章では SSA 法に基づいたアニーリングハードウェアの FPGA 実装及びその動作について述べた．SSA 法のスピン動作に基づいた CMOS 回路である spin-gate の構造から，800 スピンを搭載した SSA ハードウェアのスピン接続後続を含めた，ハードウェアのアーキテクチャについて述べた．SSA ハードウェアのスピン接続構造は一つのスピンが隣接の八つのスピンと接続するスパースな構造を持つ．加えて，SSA 法の特性に基づいた逆温度パラメータの制御及びアニーリング結果を保存するためのメモリ制御について考察し，SSA ハードウェアのメモリ使用量の削減を示した．800 スピン SSA ハードウェアを FPGA を用いて実装し，800 ノードの COP に対するアニーリング性能を従来 SA 法と比較した．さらに，同様なスピン数を持つ FPGA ベースの従来アニーリングハードウェアとその性能を比較した．提案 SSA ハードウェアは従来アニーリングハードウェアより 800 ノード COP の近似解を 2.64 倍高速で見つけた．このように，800 スピン SSA ハードウェアを実装し，その性能について考察することで，大規模な完全グラフイジングモデル向けアニーリングプロセッサ設計の基盤を構築した．

第 4 章では，大規模完全グラフイジングモデル向けの DSSA アルゴリズムの詳細について述べた．スパースなスピン接続構造を持つ 800 スピン SSA ハードウェアから完全接続構造のハードウェア面積問題について考察し，シリアル化による小面積化と，シリアル化の面積と計算時間のトレードオフ関係について考察した．そこで，状態が変化した差分スピンのみを用いてシリアル化による計算時間増加を抑える DSSA アルゴリズムを提案した．DSSA 法はスピン状態の確率の変化量だけを求めてスピン状態を更新することでシリアル化されたスピン相互作用の計算を高速化するアニーリングアルゴリズムである．DSSA 法の動作検証は 2,000 ノードの COP を用いて，アニーリング精度を確認し，単純シリアル化された従来 SSA 法と計算時間を比較して，大規模完全グラフイジングモデルにおける DSSA 法の有効性を確認した．2,000 ノード COP に対して，DSSA 法は問題の最良解の約 99% の近似解を得ることができた．加えて，2,000 ノード完全グラフの問題に対しては，その問題の最良解を得ることができ，DSSA 法が大規模完なイジングモデルに高いアニーリング精度を持つことを示した．単純シリアル化された従来 SSA 法に比べ，DSSA 法は問題に応じて最大 42.1 倍高速で同様な動作が可能であることを確認した．

第 5 章では DSSA 法に基づいた大規模完全グラフイジングモデル向けアニーリングプロセッサのアーキテクチャ及びその動作について述べた．提案 DSSA プロセッサは互いに完全接続する 2,048 スピンを単一チップに搭載ており，マルチチップなどを使用せずに，2.048 スピンという大規模な完全グラフイジングモデルに対応可能である．シリアル化されたスピン相互作用に従って，プロセッサに含まれてるモジュールの動作をパイプライン化することで，アニーリング処理の計算クロックサイクル数を状態が変化したスピンを意味する差分スピン数に応じて短縮する．また，スピンの確率的動作に必要な乱数信号を複数のスピンに共有することで，プロセッサに含まれる乱数発生器の面積の削減など，大規模な DSSA プロセッサの設計を示した．2,048 スピン DSSA プロセッサの ASIC 実装に向けて，小規模の DSSA プロセッサのシミュレーション及び FPGA 実装を用いてその動作を検証した．提案 DSSA プロセッサは TSMC 28 nm を用いて ASIC 実装向けに設計され，論理合成後の性能は動作周波数は 500 MHz，消費電力は 316.1 mW であった．提案 DSSA プロセッサは state-of-the-art を含めた ASIC ベース従来アニーリングプロセッサに比べ，最も大規模な 2,048 スピンを搭載し，その接続構造も完全グラフである．また，state-of-the-art のアニーリングプロセッサより約 6.2 倍省電力化を達成した．アニーリング精度の

比較としては K2000 という 2,000 ノードの完全グラフ COP において，従来アニーリングプロセッサより約 3.5 倍のエネルギー効率を示した．このように，本論文では COP の社会応用のために要求されている大規模な完全グラフイジングモデル向けアニーリングプロセッサを実現するために，スピン状態の差分を利用する高速な DSSA アルゴリズムについて述べた．また，その DSSA アルゴリズムに基づいた 2,048 スピン DSSA プロセッサを設計することで，実世界に応用される大規模な COP に対応可能なアニーリングプロセッサ実現の基盤を構築した．

本論文で示した 2,048 スピン DSSA プロセッサはシミュレーションと論理合成及び配置配線後の結果を用いて性能評価を行った．提案プロセッサが ASIC に実装された時の性能を評価し，大規模な COP における DSSA プロセッサの有効性を確認することが今後の課題である．イジングモデルより表現された COP は最短経路決定や決定問題など社会全般の様々な分野に適用可能な数理モデルであり，その実世界への応用は社会へ大きい影響を与える可能性が高い．本論文では COP の一種である最大カット問題を用いてアニーリング精度など，DSSA 法とそれに基づいたプロセッサの性能を評価した．大規模な完全グラフイジングモデル向けの提案 DSSA プロセッサを，最大カット問題だけではなく，より実世界の問題に密接に関係する最適化問題に提供することで，提案プロセッサの有効性及び社会的影響を確かめることが，今後の展望である．

以上を持ち，スピン状態の差分を用いた大規模完全グラフ向け高性能アニーリングプロセッサに関する研究の全容を示した．

付録

Chapter 6 では本論文で用いた従来 SSA 法及び DSSA 法の動作確認で用いた Python プログラムのソースコードを示す．ただし，Python プログラムの実行環境の OS は Rocky Linux 8.6 であり，Python 3.8 を使用する．用いる Python のライブラリは，Numpy 1.24.3，networkx 3.1，matplotlib 3.7.1，である．

1. SSA 法のシミュレーションを実装した Python class のソースコード．

```python
import numpy as np
import math
import networkx as networkx


def convert(vertex, gset):
    # vertex : Number of vertexs in G-set
    # edge : Number of edges in G-set
    # gset : Name of G-set text file (ex) G11)
    lg = np.zeros((vertex, vertex)).astype(int)
    with open('./graph/' + gset + '.txt', 'r') as fp:
        lines = fp.readlines()
        edge = len(lines)
        for line in lines:
            l_tmp = line.strip().split(sep=' ')
            [s, t, w] = list(map(int, l_tmp))
            lg[s-1, t-1] = w

    lg += lg.T
    J = lg * -1
    h = np.zeros(vertex)
    G = nx.from_numpy_array(lg)

    return G, h, J


class annealier:
    def __init__(self, tau=10, I0_ini=1, I0_max=16, beta=0.5, nrnd=1,
                 trial=10, Mcycle=100):
```

```
30          self.tau = tau
31          self.I0_ini = I0_ini
32          self.I0_max = I0_max
33          self.beta = beta
34          self.nrnd = nrnd
35          self.trial = trial
36          self.Mcycle = Mcycle
37
38          self.cut = np.zeros((trial, Mcycle+1))
39          self.energy = np.zeros((trial, Mcycle+1))
40
41          self.NI = int(math.log10(self.I0_ini / self.I0_max) / math.log10(self.beta))
42          self.NI = 1 + float(self.NI)
43
44      def hamiltonian_set(self, h, J):
45          self.I0 = np.zeros(self.Mcycle)
46          self.h = h
47          self.J = J
48
49          self.length = len(h)
50
51          self.mo = np.zeros((self.trial, self.length, self.Mcycle+1))
52          self.Ii = np.zeros((self.length, self.Mcycle+1))
53
54      def process(self, verbose=False):
55          self.temperture_control()
56          for k in range(self.trial):
57              print('#= Trial', k, 'start!')
58              mi = 2 * np.random.randint(2, size=self.length) - 1
59              rnd = self.nrnd * (2 * np.random.randint(2, size=(self.length, self.Mcycle)) - 1)
60              self.energy[k, 0] = self.energy_calculation(mi)
61              self.cut[k, 0] = self.cut_calculation(mi)
62
63              for t in range(self.Mcycle):
64                  mi = self.node_calculation(k, t, mi, rnd[:, t])
65
66                  self.energy[k, t+1] = self.energy_calculation(mi)
67                  self.cut[k, t+1] = self.cut_calculation(mi)
68                  # if verbose:
69                  #     print('#= Trial', k, t, 'cycle : E =', self.energy[k, t+1], '| Cut val =',
                    ↪  self.cut[k, t+1])
70              if verbose:
71                  print('#= Trial', k, ' : Min E =', self.energy[k].min(), '| Cut val =',
                    ↪  self.cut[k].max())
72
73      def node_calculation(self, k, t, mi, rnd):
```

```
74          Jm_tmp = np.dot(self.J, mi)
75          self.Ii[:, t+1] = self.Ii[:, t] + self.h + Jm_tmp + rnd
76          self.Ii[:, t+1] = np.where(self.Ii[:, t+1] >= self.I0[t], self.I0[t]-1, self.Ii[:,
            ↪  t+1])
77          self.Ii[:, t+1] = np.where(self.Ii[:, t+1] < -self.I0[t], -self.I0[t], self.Ii[:, t+1])
78          self.mo[k, :, t+1] = np.where(self.Ii[:, t+1] >= 0, 1, -1)
79
80          return self.mo[k, :, t+1]
81
82      def temperture_control(self):
83          for t in range(self.Mcycle):
84              if t % (self.NI * self.tau) == 0:
85                  self.I0[t] = self.I0_ini
86              elif t % self.tau == 0:
87                  self.I0[t] = self.I0[t-1] / self.beta
88              else:
89                  self.I0[t] = self.I0[t-1]
90
91      def energy_calculation(self, mi):
92          Jm_tmp = np.dot(self.J, mi)
93          hm_tmp = np.dot(self.h, mi.T)
94
95          return -np.sum(Jm_tmp * mi) / 2 - hm_tmp
96
97      def cut_calculation(self, mi):
98          cut_tmp = 0
99          for i in range(self.length):
100             for j in range(i+1, self.length):
101                 if mi[i] != mi[j]:
102                     cut_tmp -= self.J[i, j]
103
104         return cut_tmp
105
106     def find_minimum(self):
107         min_energy = self.energy.min(axis=1)
108         amin_energy = self.energy.argmin(axis=1, keepdims=True)
109
110         max_cut = np.take_along_axis(self.cut, amin_energy, axis=1)
111
112         node_state = np.zeros((self.trial, self.length))
113         for k in range(self.trial):
114             node_state[k, :] = self.mo[k, :, amin_energy[k][0]]
115
116         return (min_energy, amin_energy, max_cut, node_state)
```

SSA 法のプログラムは class として定義され，Python のカスタムモジュールである pysa に含ま

れる.

2. SSA 法シミュレーションの実行プログラム

```
import numpy as np
import pickle as pkl
import matplotlib.pyplot as plt
import networkx as nx
import time
import pysa
import os


# ------ G set info ------ #
vertex = 800
gset = 'G11'
en_cut = False

# G set Info
# G1 - G21 : vertex = 800
# G22 - G42 : vertex = 2000

# ------ SA parameter ------ #
trial  = 100
Mcycle  = 90000
nrnd   = 2
I0_ini = 1
I0_max = 32
beta   = 0.5
tau    = 100

# Convert txt to numpy.array
G, h, J = pysa.convert(vertex, gset)

# Annealing process
anlier = pysa.annealier(tau, I0_ini, I0_max, beta, nrnd, trial, Mcycle)
anlier.hamiltonian_set(h, J)
anlier.process(verbose=True)

# Make directory for saving result
path = './doc_results/'
if not os.path.exists(path):
    os.mkdir(path)
path += gset
if not os.path.exists(path):
    os.mkdir(path)
path += '/'
```

```
44
45  flist = os.listdir(path)
46  didx = 0
47  dname = 'tri=' + str(trial) + '_MS=' + str(Mcycle)
48  dname += '_RW=' + str(nrnd) + '_tmin=' + str(I0_ini)
49  dname += '_tmax=' + str(I0_max) + '_beta=' + str(beta)
50  dname += '_tau=' + str(tau) + '_'
51  for tdname in flist:
52      if tdname.startswith(dname):
53          didx += 1
54  dname += str(didx)
55  os.mkdir(path + dname)
56  path += dname + '/'
57
58  # # Read the result from the annealier
59  result_out = {}
60  result_out['eng'] = anlier.energy
61  print(anlier.energy.shape)
62  result_out['cut'] = anlier.cut
63
64  # Save the result
65  # 'scsa_sim_result.pkl' : Energy and cut value for every cycles
66  # 'noise_param.pkl' : Hyper parameters
67  # 'node_state.pkl' : State of nodes for every cycles
68  with open(path + 'ssa_sim_result.pkl', 'wb') as fp:
69      pkl.dump(result_out, fp)
70
71  noise_param = {}
72  noise_param['trial'] = trial
73  noise_param['mcycle'] = Mcycle
74  noise_param['rnd_weight'] = nrnd
75  noise_param['tem_min'] = I0_ini
76  noise_param['tem_max'] = I0_max
77  noise_param['beta'] = beta
78  noise_param['tau'] = tau
79  with open(path + 'noise_param.pkl', 'wb') as fp:
80      pkl.dump(noise_param, fp)
```

SSA 法の class を含めた pysa を用いて SSA 法のシミュレーションを実行する.

3. DSSA 法のシミュレーションを実装した Pyton class のソースコード.

```
import numpy as np
import wandb
from .func import random_generator, cal_mcycle, cal_mcycle_fp, time_stamp
import os
import pickle as pkl


class dissa:
    def __init__(self, h, J, noise_params,
                 rng_method='xor',
                 noise_share=16,
                 spin_init='h',
                 fp=True,
                 wandb_proj=None,
                 wandb_run_name=None,
                 gset='K2000_1'
                 ):
        self.h = h
        self.J = J
        self.num_spin = len(h)

        self.trial  = noise_params['trial']
        self.mshot  = noise_params['mshot']
        self.nrnd   = noise_params['rnd_weight']
        self.tem_min = noise_params['tem_min']
        self.tem_max = noise_params['tem_max']
        self.beta   = noise_params['beta']
        self.tau    = noise_params['tau']

        self.st_time = time_stamp()

        self.ns = noise_share
        self.rng = random_generator(rng_method, (self.num_spin // self.ns))

        if fp:
            self.NI, self.mcycle = cal_mcycle_fp(self.mshot, self.tem_min, self.tem_max,
            ↪  self.beta, self.tau)
        else:
            self.NI, self.mcycle = cal_mcycle(self.mshot, self.tem_min, self.tem_max,
            ↪  self.beta, self.tau)
        self.cyc_p_shot = self.NI * self.tau
        if fp:
            self.dtyp = float
            self.temperture_control_fp()
```

```
43          else:
44              self.dtyp = int
45              self.temperture_control()
46
47          self.mo = np.zeros((self.trial, self.mcycle+1, self.num_spin), dtype=np.int8)
48          self.Ii = np.zeros((self.mcycle+1, self.num_spin), dtype=self.dtyp)
49
50          for k in range(self.trial):
51              self.mo[k, 0] = self.spin_initialize(spin_init)
52
53          self.cut = np.zeros((self.trial, self.mcycle+1), dtype=int)
54          self.energy = np.zeros((self.trial, self.mcycle+1), dtype=int)
55          self.update_count = np.zeros((self.trial, self.mcycle), dtype=int)
56          self.actual_clock = np.zeros(self.trial, dtype=int)
57
58          if wandb_proj is not None:
59              self.en_wandb = True
60              self.wandb_set(wandb_proj, wandb_run_name, noise_params, rng_method, self.ns,
                ↪  spin_init)
61          else:
62              self.en_wandb = False
63
64          print('Process cycles = ' + str(self.mcycle))
65
66      def make_result_dir(self, path):
67          if self.en_wandb:
68              self.st_time = self.wandb_run_name + '_' + self.st_time
69          path += '/' + self.st_time
70          if not os.path.isdir(path):
71              os.mkdir(path)
72          return path + '/'
73
74      def spin_initialize(self, spin_init):
75          if spin_init == 'h':
76              return np.ones(self.num_spin, dtype=np.int8) * -1
77          else:
78              return np.random.randint(0, 2, self.num_spin) * 2 - 1
79
80      def temperture_control(self):
81          self.I0 = np.zeros(self.cyc_p_shot, dtype=self.dtyp)
82          for t in range(self.cyc_p_shot):
83              if t // self.tau == 0:
84                  self.I0[t] = self.tem_min
85              elif t % self.tau == 0:
86                  self.I0[t] = self.I0[t-1] * pow(2, self.beta)
87              else:
```

```
88                 self.I0[t] = self.I0[t-1]
89
90         def temperture_control_fp(self):
91             self.I0 = np.zeros(self.cyc_p_shot, dtype=self.dtyp)
92             for t in range(self.cyc_p_shot):
93                 if t // self.tau == 0:
94                     self.I0[t] = self.tem_min
95                 elif t % self.tau == 0:
96                     self.I0[t] = self.I0[t-1] / self.beta
97                 else:
98                     self.I0[t] = self.I0[t-1]
99
100        def wandb_set(self, project, run_name, noise_params, rng_method, noise_share, spin_init):
101            # self.en_wandb = True
102            self.wandb_project = project
103            self.wandb_run_name = run_name
104            self.wandb_conf = noise_params
105            self.wandb_conf['noise_share'] = noise_share
106            self.wandb_conf['rng_method'] = rng_method
107            self.wandb_conf['spin_init'] = spin_init
108
109        def energy_calculation(self, mi):
110            Jm_tmp = np.dot(self.J, mi)
111            hm_tmp = np.dot(self.h, mi.T)
112
113            return -np.sum(Jm_tmp * mi) / 2 - hm_tmp
114
115        def cut_calculation(self, mi):
116            cut_tmp = 0
117            for i in range(self.num_spin):
118                diff_mask = np.where(mi[i:] != mi[i], True, False)
119                J_mask = self.J[i, i:][diff_mask]
120                cut_tmp -= np.sum(J_mask, dtype=int)
121
122            return cut_tmp
123
124        def process(self, verbose=True):
125            # Record annealing start time
126            self.st_time = time_stamp()
127            for k in range(self.trial):
128                # Run wandb
129                if self.en_wandb:
130                    self.wandb_conf['trial'] = k+1
131                    if self.wandb_run_name is not None:
132                        wandb.init(
133                            project=self.wandb_project,
```

```
134                         name=self.wandb_run_name + '-{0:03d}'.format(k+1),
135                         config=self.wandb_conf
136                     )
137                 else:
138                     wandb.init(
139                         project=self.wandb_project,
140                         config=self.wandb_conf
141                     )
142 
143             # Initialize spins, counters, diff flags
144             mi = self.mo[k, 0]
145             self.energy[k, 0] = self.energy_calculation(mi)
146             if verbose:
147                 self.cut[k, 0] = self.cut_calculation(mi)
148             delta_mi = np.arange(self.num_spin)
149             self.delta_Ii = np.copy(self.h)
150 
151             # Annealing process
152             for t in range(self.mcycle):
153                 # Make noise signal
154                 rnd_tmp = self.rng.run()
155                 # Duplicate noise signal
156                 rnd = np.copy(rnd_tmp)
157                 for ii in range(self.ns-1):
158                     rnd = np.append(rnd, rnd_tmp)
159 
160                 mi_nxt = self.node_calculation(k, t, mi, delta_mi, rnd)
161                 # Find flipped spins
162                 delta_mi = np.argwhere(mi_nxt != mi).reshape(-1)
163                 # Save the new state of spins
164                 mi = np.copy(mi_nxt)
165 
166                 # Calculate energy and cut value
167                 self.energy[k, t+1] = self.energy_calculation(mi)
168                 if verbose:
169                     self.cut[k, t+1] = self.cut_calculation(mi)
170                     self.actual_clock[k] += self.update_count[k, t] + 2
171 
172                 # Log data at wandb project
173                 if self.en_wandb:
174                     wandb.log({'Energy': self.energy[k, t+1], 'Cut': self.cut[k, t+1]}, step=t)
175                     wandb.log({'Update count': self.update_count[k, t]}, step=t)
176                     if verbose:
177                         wandb.log({'Actual clock': self.actual_clock[k]}, step=t)
178             # Summary the run
179             if self.en_wandb:
```

```
180                 wandb.run.summary['Max Cut'] = np.max(self.cut[k])
181                 wandb.run.summary['Min Energy'] = np.min(self.energy[k])
182                 if verbose:
183                     wandb.run.summary['Total clock'] = self.actual_clock[k]
184                 wandb.finish()
185 
186     # DISSA node calculation
187     def node_calculation(self, k, t, mi, delta_mi, rnd):
188         tt = t % self.cyc_p_shot
189         # Accumualtion of spin interactions from flipped spins
190         for sidx in delta_mi:
191             if t == 0:
192                 # At the beginning of annealing, differences are initialized as biases
193                 self.delta_Ii += self.J[sidx] * mi[sidx]
194             else:
195                 self.delta_Ii += (self.J[sidx] * 2) * mi[sidx]
196             self.update_count[k, t] += 1
197 
198         # Up-down counter position accumulation
199         self.Ii[t+1] = self.delta_Ii + self.Ii[t] + (self.nrnd * rnd)
200 
201         # Inverse temperature limit
202         self.Ii[t+1] = np.where(self.Ii[t+1] >= self.I0[tt], self.I0[tt]-1, self.Ii[t+1])
203         self.Ii[t+1] = np.where(self.Ii[t+1] < -self.I0[tt], -self.I0[tt], self.Ii[t+1])
204 
205         # Spin update
206         mi_nxt = np.where(self.Ii[t+1] >= 0, 1, -1)
207         self.mo[k, t+1, :] = np.copy(mi_nxt)
208 
209         return mi_nxt
210 
211     # Save results as pickle
212     def save_result(self, path):
213         result_dict = {}
214         result_dict['energy'] = self.energy
215         result_dict['cut'] = self.cut
216         result_dict['update_count'] = self.update_count
217         result_dict['actual_clock'] = self.actual_clock
218 
219         with open(path + 'exp_data.pkl', 'wb') as file:
220             pkl.dump(result_dict, file)
```

SSA 法と同様に DSSA 法のシミュレーションも Python の class として実装され，カスタムモジュールである pyssa に含まれており，実行プログラムから呼び出して使われる．また，全体アニーリング処理のステップ数などは pyssa に含まれた関数より計算される．

4. カスタムの Python モジュールである pyssa に含まれた，DSSA 法のシミュレーションで用いられる関数のソースコード．

```
import os
import pickle as pkl
import math
import networkx as nx
import numpy as np
from .xorshift import xor_shift
import time


def convert(graph):
    with open('./graph/' + graph + '.txt', 'r') as fp:
        lines = fp.readlines()
        sl = []
        tl = []
        wl = []
        for line in lines:
            l_tmp = line.strip().split(sep=' ')
            [s, t, w] = list(map(int, l_tmp))
            sl.append(s - 1)
            tl.append(t - 1)
            wl.append(w)
        num_spin = max(max(sl), max(tl)) + 1
        weights = np.zeros((num_spin, num_spin), dtype='int')

        for i in range(len(sl)):
            weights[sl[i], tl[i]] = wl[i]
        weights += weights.T

        G = nx.from_numpy_array(weights)
        J = weights * -1

        return G, J


def noise_container(trial, mshot, tem_min, tem_max, nrnd, beta, tau):
    noise_param = {}
    noise_param['trial'] = trial
    noise_param['mshot'] = mshot
    noise_param['rnd_weight'] = nrnd
    noise_param['tem_min'] = tem_min
    noise_param['tem_max'] = tem_max
    noise_param['beta'] = beta
    noise_param['tau'] = tau
```

```
44
45      return noise_param
46
47  def cal_mcycle(mshot, I0_ini, I0_max, beta, tau):
48      NI = math.log10(I0_max / I0_ini)
49      NI = int(NI / math.log10(pow(2, beta))) + 1
50      mcycle = int(NI * tau * mshot)
51
52      return NI, mcycle
53
54
55  def cal_mcycle_fp(mshot, I0_ini, I0_max, beta, tau):
56      NI = int(math.log10(I0_ini / I0_max) / math.log10(beta))
57      NI = int(1 + float(NI))
58      mcycle = int(NI * tau * mshot)
59
60      return NI, mcycle
61
62
63  class random_generator:
64      def __init__(self, rng_method='xor', length=100):
65          self.rng_method = rng_method
66          self.length = length
67          if rng_method == 'xor':
68              self.rng = xor_shift(self.length)
69              self.rng.reset()
70          elif rng_method == 'random':
71              self.rng = np.random.default_rng()
72
73      def run(self):
74          if self.rng_method == 'xor':
75              return self.rng.run()
76          elif self.rng_method == 'random':
77              return (self.rng.integers(low=0, high=2, size=self.length) * 2) - 1
78
79
80  def time_stamp():
81      tm = time.localtime(time.time())
82      str_tm = time.strftime('%y-%m-%d_%H-%M-%S', tm)
83      return(str_tm)
```

DSSA 法のアニーリング処理のステップ数などを計算する，シミュレーションのための関数は pyssa モジュールに含まれる．

5. DSSA 法のシミュレーションの実行プログラム.

```
import pyssa
import numpy as np
import os
import pickle as pkl


gset = 'K2000_1'
trial = 5
mshot = 5
nrnd = 3
I0_ini = 1
I0_max = 2048
tau = 300
beta = 1

print('Problem :', gset)
path = './rand_test/' + gset

if not os.path.isdir(path):
    os.mkdir(path)

dnum = len(os.listdir(path))

wandb_proj = 'DOC_K2000'
wandb_run_name = 'run-' + '{:02d}'.format(dnum) + '-' + gset

with open('./graph/graph_dict.pkl', 'rb') as fp:
    gdict = pkl.load(fp)
bst_cut = gdict[gset]
rng_method = 'xor'
noise_share = 50
spin_init = 'h'
fp = False

_, J = pyssa.convert(gset)
h = np.zeros(J.shape[0], dtype=int)

noise_params = pyssa.noise_container(trial, mshot, I0_ini,
                                     I0_max, nrnd, beta, tau)
NI, mcycle = pyssa.cal_mcycle(mshot, I0_ini, I0_max, beta, tau)

print('T Inc in shot :', NI)
print('Total annealing step :', mcycle)

```

```
annil = pyssa.dissa(h, J, noise_params,
                    rng_method=rng_method,
                    noise_share=noise_share,
                    spin_init=spin_init,
                    fp=fp,
                    wandb_proj=wandb_proj,
                    wandb_run_name=wandb_run_name
                    )
path = annil.make_result_dir(path)
with open(path + 'hp_param.pkl', 'wb') as file:
    pkl.dump(noise_params, file)

# Annealing process
annil.process(verbose=True)

# Save annealing results
annil.save_result(path)

near_opt_cut = bst_cut * 0.99
eng = annil.energy
cut = annil.cut
cnt = annil.update_count

clk_step = np.zeros_like(cut)
for i in range(clk_step.shape[1]-1):
    clk_step[:, i+1] = clk_step[:, i] + cnt[:, i] + 2
atime = clk_step * 0.000002

fin_time = atime[:, -1]

avg_cut = np.mean(cut, axis=0)
avg_atime = np.mean(atime, axis=0)
near_opt_step = 0
for i in range(avg_cut.shape[0]):
    if avg_cut[i] >= near_opt_cut:
        near_opt_step = i
        break

max_cut = np.max(cut[:, :i], axis=1)
d_P = np.sum(np.where(max_cut >= near_opt_cut, 1, 0)) / max_cut.shape[0]
d_tts = pyssa.time_to_sol(d_P, atime[:, i])
```

DSSA 法のシミュレーションでは，各アニーリングステップ当たりのスピンの状態，イジングエネルギー，与えられた COP に対するコストなどを結果として記録する．

参考文献

[1] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., USA, 1982.

[2] Jianli Chen, Wenxing Zhu, and M. M. Ali. A hybrid simulated annealing algorithm for non-slicing vlsi floorplanning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 41, No. 4, pp. 544–553, 2011.

[3] Hui Miao and Yu-Chu Tian. Dynamic robot path planning using an enhanced simulated annealing approach. *Applied Mathematics and Computation*, Vol. 222, pp. 420–437, 2013.

[4] Kun Shi, Zhengtian Wu, Baoping Jiang, and Hamid Reza Karimi. Dynamic path planning of mobile robot based on improved simulated annealing algorithm. *Journal of the Franklin Institute*, Vol. 360, No. 6, pp. 4378–4398, 2023.

[5] Kosuke Tatsumura, Ryo Hidaka, Jun Nakayama, Tomoya Kashimata, and Masaya Yamasaki. Real-time trading system based on selections of potentially profitable, uncorrelated, and balanced stocks by np-hard combinatorial optimization. *IEEE Access*, Vol. 11, pp. 120023–120033, 2023.

[6] Kosuke Tatsumura, Ryo Hidaka, Jun Nakayama, Tomoya Kashimata, and Masaya Yamasaki. Pairs-trading system using quantum-inspired combinatorial optimization accelerator for optimal path search in market graphs. *IEEE Access*, Vol. 11, pp. 104406–104416, 2023.

[7] Ryo Hidaka, Yohei Hamakawa, Jun Nakayama, and Kosuke Tatsumura. Correlation-diversified portfolio construction by finding maximum independent set in large-scale market graph. *IEEE Access*, Vol. 11, pp. 142979–142991, 2023.

[8] 大矢晃示, 藤本裕, 濱川洋平, 山崎雅也, 辰村光介. 量子インスパイアード車載プラットフォームの提案と試作. 自動車技術会論文集, Vol. 54, No. 6, pp. 1216–1221, 2023.

[9] Richard M. Karp. *Reducibility among Combinatorial Problems*, pp. 85–103. Springer US, Boston, MA, 1972.

[10] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, Vol. 220, No. 4598, pp. 671–680, 1983.

[11] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Phys. Rev. E*, Vol. 58, pp. 5355–5363, Nov 1998.

[12] Catherine McGeoch and Pau Farré. Advantage processor overview. Technical Report D-Wave Technical Report Series 14-1058A-A, D-Wave Quantum Systems Inc., 01 2022.

[13] Rafatul Faria, Kerem Y. Camsari, and Supriyo Datta. Low-barrier nanomagnets as p-bits for spin logic. *IEEE Magnetics Letters*, Vol. 8, pp. 1–5, 2017.

[14] Kerem Y. Camsari, Brian M. Sutton, and Supriyo Datta. p-bits for probabilistic spin logic. *Applied Physics Reviews*, Vol. 6, No. 1, p. 011305, 03 2019.

[15] Warren J. Gross and Vincent C. Gaudet. *Stochastic Computing: Techniques and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2019.

[16] Naoya Onizawa, Kota Katsuki, Duckgyu Shin, Warren J. Gross, and Takahiro Hanyu. Fast-converging simulated annealing for ising models based on integral stochastic computing. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–7, 2022.

[17] Sergei V. Isakov, I.N. Zintchenko, T.F. Rønnow, and M. Troyer. Optimised simulated annealing for ising spin glasses. *Computer Physics Communications*, Vol. 192, pp. 265–271, 2015.

[18] Roy J. Glauber. Time ‐ dependent statistics of the ising model. *Journal of Mathematical Physics*, Vol. 4, No. 2, pp. 294–307, 1963.

[19] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel. Computation on stochastic bit streams digital image processing case studies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 22, No. 3, pp. 449–462, March 2014.

[20] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, Vol. 9, No. 1, pp. 147–169, 1985.

[21] David S. Johnson and Lyle A. McGeoch. *8. The traveling salesman problem: a case study*, pp. 215–310. Princeton University Press, Princeton, 2003.

[22] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, Vol. 60, pp. 94–112, 2014.

[23] Clayton W. Commander. *Maximum cut problem, MAX-CUT*, pp. 1991–1999. Springer US, Boston, MA, 2009.

[24] Y. Ye. Computational optimization laboratory, 1999.

[25] Takashi Takemoto and et al. 4.6 a 144kb annealing system composed of 9 × 16kb annealing processor chips with scalable chip-to-chip connections for large-scale combinatorial optimization problems. In *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, Vol. 64, pp. 64–66, 2021.

[26] Kasho Yamamoto, Kazushi Kawamura, Kota Ando, Normann Mertig, Takashi Takemoto, Masanao Yamaoka, Hiroshi Teramoto, Akira Sakai, Shinya Takamaeda-Yamazaki, and Masato Motomura. 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 Journal of Solid-State Circuits*, Vol. 56, No. 1, pp. 165–178, 2021.

[27] Duckgyu Shin, Naoya Onizawa, Warren J. Gross, and Takahiro Hanyu. Memory-efficient FPGA implementation of stochastic simulated annealing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Vol. 13, No. 1, pp. 108–118, 2023.

[28] Yuya Sugie, Yuki Yoshida, Normann Mertig, Takashi Takemoto, Hiroshi Teramoto, Atsuyoshi

Nakamura, Ichigaku Takigawa, Shin-ichi Minato, Masanao Yamaoka, and Tamiki Komatsuzaki. Minor-embedding heuristics for large-scale annealing processors with sparse hardware graphs of up to 102,400 nodes. *Soft Computing*, Vol. 25, No. 3, pp. 1731–1749, Feb 2021.

[29] Kazushi Kawamura, Jaehoon Yu, Daiki Okonogi, Satoru Jimbo, Genta Inoue, Akira Hyodo, Ángel López García-Arias, Kota Ando, Bruno Hideki Fukushima-Kimura, Ryota Yasudo, Thiem Van Chu, and Masato Motomura. Amorphica: 4-replica 512 fully connected spin 336mhz metamorphic annealer with programmable optimization strategy and compressed-spin-transfer multi-chip extension. In *2023 IEEE International Solid- State Circuits Conference (ISSCC)*, pp. 42–44, 2023.

[30] Kota Katsuki, Duckgyu Shin, Naoya Onizawa, and Takahiro Hanyu. Fast solving complete 2000-node optimization using stochastic-computing simulated annealing. In *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4, 2022.

[31] Cong Zhang, Yaoxin Wu, Yining Ma, Wen Song, Zhang Le, Zhiguang Cao, and Jie Zhang. A review on learning to solve combinatorial optimisation problems in manufacturing. *IET Collaborative Intelligent Manufacturing*, Vol. 5, No. 1, p. e12072, 2023.

[32] Ramón Rotaeche, Alberto Ballesteros, and Julián Proenza. Speeding task allocation search for reconfigurations in adaptive distributed embedded systems using deep reinforcement learning. *Sensors*, Vol. 23, No. 1, 2023.

[33] Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA Journal on Computing*, Vol. 3, No. 4, pp. 376–384, 1991.

[34] Vasil S. Denchev, Sergio Boixo, Sergei V. Isakov, Nan Ding, Ryan Babbush, Vadim Smelyanskiy, John Martinis, and Hartmut Neven. What is the computational value of finite-range tunneling? *Phys. Rev. X*, Vol. 6, p. 031015, Aug 2016.

[35] M. Veldhorst, C. H. Yang, J. C. C. Hwang, W. Huang, J. P. Dehollain, J. T. Muhonen, S. Simmons, A. Laucht, F. E. Hudson, K. M. Itoh, A. Morello, and A. S. Dzurak. A two-qubit logic gate in silicon. *Nature*, Vol. 526, No. 7573, pp. 410–414, Oct 2015.

[36] STEPHEN G. BRUSH. History of the lenz-ising model. *Rev. Mod. Phys.*, Vol. 39, pp. 883–893, Oct 1967.

[37] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models, 2018.

[38] Andrew D. King, Jack Raymond, Trevor Lanting, Sergei V. Isakov, Masoud Mohseni, Gabriel Poulin-Lamarre, Sara Ejtemaee, William Bernoudy, Isil Ozfidan, Anatoly Yu. Smirnov, Mauricio Reis, Fabio Altomare, Michael Babcock, Catia Baron, Andrew J. Berkley, Kelly Boothby, Paul I. Bunyk, Holly Christiani, Colin Enderud, Bram Evert, Richard Harris, Emile Hoskinson, Shuiyuan Huang, Kais Jooya, Ali Khodabandelou, Nicolas Ladizinsky, Ryan Li, P. Aaron Lott, Allison J. R. MacDonald, Danica Marsden, Gaelen Marsden, Teresa Medina, Reza Molavi, Richard Neufeld, Mana Norouzpour, Travis Oh, Igor Pavlov, Ilya Perminov, Thomas Prescott, Chris Rich, Yuki Sato, Benjamin Sheldan, George Sterling, Loren J. Swenson, Nicholas

Tsai, Mark H. Volkmann, Jed D. Whittaker, Warren Wilkinson, Jason Yao, Hartmut Neven, Jeremy P. Hilton, Eric Ladizinsky, Mark W. Johnson, and Mohammad H. Amin. Scaling advantage over path-integral monte carlo in quantum simulation of geometrically frustrated magnets. *Nature Communications*, Vol. 12, No. 1, p. 1113, Feb 2021.

[39] Takahiro Hanyu, Tetsuo Endoh, Daisuke Suzuki, Hiroki Koike, Yitao Ma, Naoya Onizawa, Masanori Natsui, Shoji Ikeda, and Hideo Ohno. Standby-power-free integrated circuits using mtj-based vlsi computing. *Proceedings of the IEEE*, Vol. 104, No. 10, pp. 1844–1863, 2016.

[40] Nike Dattani, Szilard Szalay, and Nick Chancellor. Pegasus: The second connectivity graph for large-scale quantum annealing hardware, 2019.

[41] Kelly Boothby, Paul Bunyk, Jack Raymond, and Aidan Roy. Next-generation topology of d-wave quantum processors, 2020.

[42] Stefanie Zbinden, Andreas Bärtschi, Hristo Djidjev, and Stephan Eidenbenz. Embedding algorithms for quantum annealers with chimera and pegasus connection topologies. In Ponnuswamy Sadayappan, Bradford L. Chamberlain, Guido Juckeland, and Hatem Ltaief, editors, *High Performance Computing*, pp. 187–206, Cham, 2020. Springer International Publishing.

[43] Tomas Boothby, Andrew D. King, and Aidan Roy. Fast clique minor generation in chimera qubit connectivity graphs. *Quantum Information Processing*, Vol. 15, No. 1, pp. 495–508, Jan 2016.

[44] Prasanna Date, Robert Patton, Catherine Schuman, and Thomas Potok. Efficiently embedding qubo problems on adiabatic quantum computers. *Quantum Information Processing*, Vol. 18, No. 4, p. 117, Mar 2019.

[45] B. R. Gaines. *Stochastic Computing Systems*, pp. 37–172. Springer US, Boston, MA, 1969.

[46] B. R. Gaines. Stochastic computing. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pp. 149–156, New York, NY, USA, 1967. ACM.

[47] Arash Ardakani, Francois Leduc-Primeau, Naoya Onizawa, Takahiro Hanyu, and Warren J. Gross. VLSI implementation of deep neural network using integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 25, No. 10, pp. 2688–2699, Oct 2017.

[48] Sean C Smithson, Naoya Onizawa, Brett H. Meyer, Warren J. Gross, and Takahiro Hanyu. Efficient CMOS invertible logic using stochastic computing. *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 66, No. 6, pp. 2263–2274, June 2019.

[49] Sebastiano Vigna. Further scramblings of marsaglia' s xorshift generators. *Journal of Computational and Applied Mathematics*, Vol. 315, pp. 175 – 181, 2017.

[50] Yi-yuan Fang and Xue-jun Chen. Design and simulation of uart serial communication module based on vhdl. In *2011 3rd International Workshop on Intelligent Systems and Applications*, pp. 1–4, 2011.

[51] Hidenori Gyoten, Masayuki Hiromoto, and Takashi Sato. Enhancing the solution quality of hardware ising-model solver via parallel tempering. In *2018 IEEE/ACM International Confer-*

ence on Computer-Aided Design (ICCAD), p. 1 – 8. IEEE Press, 2018.

[52] David J. Earl and Michael W. Deem. Parallel tempering: Theory, applications, and new perspectives. *Phys. Chem. Chem. Phys.*, Vol. 7, pp. 3910–3916, 2005.

[53] Koji Hukushima and Koji Nemoto. Exchange monte carlo method and application to spin glass simulations. *Journal of the Physical Society of Japan*, Vol. 65, No. 6, pp. 1604–1608, 1996.

[54] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, Vol. 521, No. 7553, pp. 436–444, 2015.

[55] Meenal V. Narkhede, Prashant P. Bartakke, and Mukul S. Sutaone. A review on weight initialization strategies for neural networks. *Artificial Intelligence Review*, Vol. 55, No. 1, pp. 291–322, Jan 2022.

[56] Paolo Dai Pra, Benedetto Scoppola, and Elisabetta Scoppola. Sampling from a gibbs measure with pair interaction by means of pca. *Journal of Statistical Physics*, Vol. 149, No. 4, pp. 722–737, Nov 2012.

[57] Maliheh Aramon, Gili Rosenberg, Elisabetta Valiante, Toshiyuki Miyazawa, Hirotaka Tamura, and Helmut G. Katzgraber. Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Frontiers in Physics*, Vol. 7, , 2019.

[58] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[59] Troels F. Rønnow, Zhihui Wang, Joshua Job, Sergio Boixo, Sergei V. Isakov, David Wecker, John M. Martinis, Daniel A. Lidar, and Matthias Troyer. Defining and detecting quantum speedup. *Science*, Vol. 345, No. 6195, pp. 420–424, 2014.

[60] Sergio Boixo, Troels F. Rønnow, Sergei V. Isakov, Zhihui Wang, David Wecker, Daniel A. Lidar, John M. Martinis, and Matthias Troyer. Evidence for quantum annealing with more than one hundred qubits. *Nature Physics*, Vol. 10, No. 3, pp. 218–224, Mar 2014.

謝辞

本論文は，東北大学大学院 工学研究科 通信工学専攻 博士後期課程在学中に東北大学電気通信研究所 新概念 VLSI システム研究室で起こった研究を取りまとめたものです．恩師の東北大学 電気通信研究所 教授 羽生 貴弘 先生には，日々多忙な中で進捗に合わせて熱心なご指導とご鞭撻頂きました．国内・国際学会での発表や論文執筆など，研究者において貴重な経験を数多くさせていただきました．単に研究成果を出すだけではなく，その先の本質を見極め，探求していくことから研究者としての発展につながることを本研究を通して学ぶごとができました．学部 4 年生から博士課程前期及び博士課程後期を通して研究に向けての姿勢を指導くださるとともに，新しい価値を作り上げる喜びを教えてくださりました．ここに改めて深く感謝の意を申し上げます．

本学 電気通信研究所 教授 佐藤 茂雄 先生，並びに本学 大学院 情報科学研究科 教授 張山 昌論 先生には，それぞれの御専門の立場から多角的な御指導を頂き，本研究をより充実させることができました．改めて深く感謝申し上げます．

本学 電気通信研究所 准教授 鬼沢 直哉 先生には，本研究に対して非常に多くの技術的・学術的意見を頂きました．学部 4 年生から今までの研究に対して終始御助言を頂いたことは感謝しても感謝しきれません．研究を通して先生の識見は今後研究者として進ことに対して大きな指標になると思います．ここに改めて厚く御礼を申し上げます．

なお，本学 電気通信研究所 准教授 夏井 雅典 先生にはセミナー発表などを通してより広い視点から数々の御指導頂きました．博士課程後期の研究に向けた考え方をご指導ご鞭撻を賜りました．深く感謝申し上げんす．

また，研究室での研究及び生活において，様々な面から親身に御助言・御協力を頂きました，平塚 愛 秘書を初めとする新概念 VLSI システム研究室の皆様に深く感謝いたします．

最後に，これまでの学生生活を物心両面に支えてくれた両親，妻 熊谷 侑華，親友たちに感謝の意を表して，本論文を結びます．

令和 6 年 1 月 6 日

シントッキュ