Bandit-based Variable Fixing for Binary Optimization on GPU Parallel Computing

Ryota Yasudo
Graduate School of Informatics, Kyoto University
Yoshida-honmachi, Sakyo-ku, Kyoto 606–8501, Japan
Email: yasudo@i.kyoto-u.ac.jp

***Abstract*—This paper explores whether reinforcement learning is capable of enhancing metaheuristics for the quadratic unconstrained binary optimization (QUBO), which have recently attracted attention as a solver for a wide range of combinatorial optimization problems. In particular, we introduce a novel approach called the bandit-based variable fixing (BVF). The key idea behind BVF is to regard an execution of an arbitrary metaheuristic with a variable fixed as a play of a slot machine. Thus, BVF explores variables to fix with the maximum expected reward, and executes a metaheuristic at the same time. The bandit-based approach is then extended to fix multiple variables. To accelerate solving multi-armed bandit problem, we implement a parallel algorithm for BVF on a GPU. Our results suggest that our proposed BVF enhances original metaheuristics.**

***Index Terms*—quadratic unconstrained binary optimization, GPGPU, multi-armed bandit problem, decision making.**

I. Introduction

Triggered by the rise of quantum annealing [1], the quadratic unconstrained binary optimization (QUBO) has been gathering attention because it is mathematically equivalent to finding the ground state of an Ising model. In particular, QUBO solvers with FPGAs [2]–[4], GPUs [5], [6], and ASICs [7], [8] are developed and reported in the literature as a potential universal solver of a wide range of combinatorial optimization problems. QUBO is an NP-hard problem in which the variables are restricted to 0 and 1, and a solution is represented by a bit vector. Thus, most of the solvers rely on metaheuristics that *flip* a bit repeatedly based on the values of Δ_i, a change of the objective function when bit i is flipped, as illustrated in Fig. 1 (b). A design of a metaheuristic hence reduced to a policy of selecting a bit to flip.

An interesting question here is whether reinforcement learning can enhance existing metaheuristics. In particular, we focus on the multi-armed (or k-armed) bandit problem, a fundamental and simple reinforcement learning problem [9]. Suppose that you faced repeatedly with a choice among k slot machines, each of which returns a numerical reward based on a stationary probability distribution that depends on each machine. The multi-armed bandit problem aims to maximize the expected total reward over some plays of slot machines. This problem is simple yet practical; it has many applications. Recently it is applied to sophisticated reinforcement learning methods such as Monte Carlo tree search, which is notably employed in software that plays board games [10], [11].

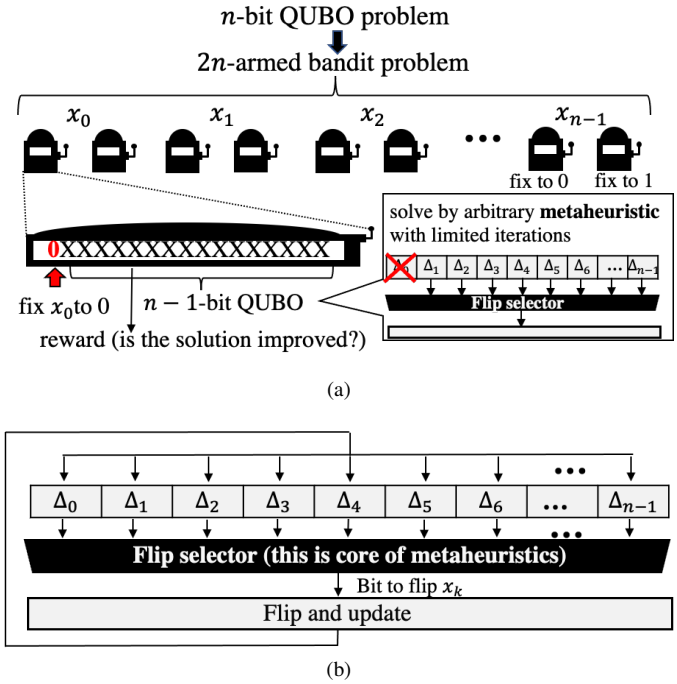


Fig. 1. Basic concept of our method. (a) The bandit-based variable fixing for an n-bit QUBO. It solves a $2n$-armed bandit problem and $n-1$-bit QUBO simultaneously. (b) Existing metaheuristics for quadratic unconstrained binary optimization (QUBO), which is used in our method as a play of a slot machine.

One straightforward idea to apply the multi-armed bandit problem to QUBO is to regard each flip as a machine. However, this formulated problem is hard to solve because the probability distribution becomes non-stationary; the reward is directly determined by the value of Δ, which changes every time a variable is flipped. Another possible idea to apply the multi-armed bandit problem is to determine the best metaheuristic from multiple candidates, each of which corresponds to a machine. In principle, however, this approach cannot outperform a system that executes the best metaheuristic. Our method proposed herein is more aggressive, general, and capable of outperforming existing metaheuristic.

Specifically, we propose to apply the multi-armed bandit problem to a *variable fixing* method as a preliminary step toward metaheuristics. Fixing a bit to 0 or 1 divides a search space into bisection, and we can select a half search space so that the expected solution is better. For an n-bit QUBO, the

number of possible 1-bit fixing is $2n$, and we select one of them. To estimate the best search space, we are required to explore many possible search spaces and obtain the results by executing a metaheuristic. At the same time, we must exploit the knowledge of which search space is better. Here we face the so-called *exploration-exploitation dilemma*, and hence we propose to resolve this dilemma by solving the multi-armed bandit problem.

Our key idea is to regard a variable fixing (or a search space bisection) as a machine. We call this approach, whose basic concept is depicted in Fig. 1, the *bandit-based variable fixing* (*BVF*). Our method can directly use such existing flip-based metaheuristics as a play of a slot machine in a multi-armed bandit problem, and further improve the solution quality. In this sense, BVF is a general method and a higher-level method than metaheuristics. A main issue of BVF is that the number of machines is large, i.e., $2n$ for n-bit QUBO. It takes much time to play all of the machines. To address this issue, this paper accelerates BVF by exploiting GPU parallel computing for the multi-armed bandit problem.

Section II describes quadratic unconstrained binary optimization and multi-armed bandit problem. Section III introduces the bandit-based variable fixing (BVF) and its GPU implementation. In Section IV, BVF is evaluated in terms of the solution quality, the execution time. Finally, Section V concludes the paper.

II. Preliminaries

A. Quadratic Unconstrained Binary Optimization

The quadratic unconstrained binary optimization (QUBO) is binary optimization where the objective function is quadratic and there are no constraints. Let X be an n-bit vector $X = x_0x_1\cdots x_{n-1}$ ($x_i \in \{0,1\}$); then QUBO aims to find X such that $f(X) = X^\top QX = \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} q_{i,j}x_ix_j$ is minimum, where a matrix $Q = (q_{i,j})$ determines the quadratic function $f(X)$. QUBO is NP-hard, and a wide range of combinatorial optimization problems, including all of Karp's 21 NP-complete problems [12] and training machine learning models [13], can be reduced to QUBO by setting the values of $q_{i,j}$.

Most of the metaheuristics for solving QUBO relies on Δ-based flip policy (Fig. 1 (b)). This policy retains Δ_i for all $i \in \{0,1,\cdots,n-1\}$, which denotes the difference in the objective function after bit i is flipped, and determine which bit to flip based on the values. For example, a hill climbing selects bit $i = \arg\max \Delta_i$, and a simulated annealing selects a bit on the basis of Metropolis–Hastings algorithm. We may regard $X = 00\cdots0$ as an initial solution because $\Delta_i = q_{i,i}$ holds in this solution, and Δ can be updated in $O(1)$ computational cost [6]. This update computation can be parallelized by GPU and FPGA. Some literature suggests that classical (non-quantum) QUBO solvers outperform quantum annealing [14].

B. Multi-Armed Bandit Problem

Throughout the paper we focus on the *stochastic* multi-armed bandit problem, where the rewards for each slot machine are provided from a probability distribution. Suppose that we have k slot machines (or arms). Let μ_i be the expected reward obtained from machine i ($i \in \{0,1,\cdots,k-1\}$). If the value of μ_i for all i is known, then the best policy continues to play machine i^* such that $\mu_{i^*} = \mu^* = \max_{i\in\{0,1,\cdots,k-1\}}\mu_i$. When the number of total plays is T, the best cumulative reward is μ^*T. However, the bandit problem assumes that μ_i is unknown, and actual cumulative reward should be less than μ^*T. This difference is called *regret* and denoted by $\mathrm{regret}(T)$. The objective of the bandit problem is to minimize the expected value of regret, denoted by $\mathbb{E}[\mathrm{regret}(T)]$.

Many methods to solve the multi-armed bandit problem have been proposed. The *UCB* policy relies on the upper confidence bound (UCB) score based on Hoeffding's inequality. In particular, UCB1 algorithm [15] assumes that the reward is in $[0,1]$ and the UCB score is computed by $\hat{\mu}_i + \sqrt{\frac{2\ln N}{N_i}}$, where $\hat{\mu}_i$, N, and N_i denote a sample mean of the reward of machine i, the number of total plays, and the number of plays of machine i, respectively. The UCB policy selects a machine with the maximum UCB score. Intuitively, term $\sqrt{\frac{2\ln N}{N_i}}$ indicating uncertainty is large if machine i is not played so far. This follows the principle of *optimism in the fact of uncertainty*. We adopt the UCB1 algorithm in this paper.

III. Methods

A. Basic Bandit-based Variable Fixing

First, we describe how to determine a variable fixing for n-bit QUBO based on the multi-armed bandit problem. For any $i \in \{0,1,\cdots,n-1\}$, let machine $2i$ and machine $2i+1$ correspond to fixing bit i to 0 and 1, respectively. This assumption results in $2n$-armed bandit problem. A play of each machine corresponds to an execution of a metaheuristic with bit i fixed. A reward is determined by whether the solution is improved or not. If the solution is improved, then the reward is 1. If the solution is the same, the reward is 0.5. If the solution becomes worse, the reward is 0.

Algorithm 1 shows a main function of the basic BVF. At first, machines are initialized based on the policy for bandit problem. Since we adopt UCB1 algorithm, the UCB score requires at least one play trial N_i for each machine, and hence the number of trials is initialized to 1. The algorithm then starts from an initial solution $X = 00\cdots0$ (i.e., $x_i = 0$ and $\Delta_i = q_{i,i}$ for all i), and repeatedly executes PLAY_MACHINE(X), where an arbitrary metaheuristic are executed with a variable fixed and returns an updated solution.

Algorithm 2 shows pseudocodes of INIT_MACHINE() and PLAY_MACHINE(X) based on the UCB1 algorithm. Since the UCB score is computed by the number of play trials and rewards, UCB structure consists of an integer *trials* and a floating point variable *reward*. As an initialization in INIT_MACHINE(), the number of play trials and reward of every machine is set to 1 and 0.5, respectively. Subsequently PLAY_MACHINE(X) fixes a variable such that the UCB score is maximum, executes a metaheuristic, and updates the values of trials and reward. A pair of UCB(i,j) and UCB($i,(j-1)^2$)

Algorithm 1 Basic bandit-based variable fixing

```
1: procedure BVF
2:     INIT_MACHINE()
3:     X ← 00···0                                  ▷ initial solution
4:     loop
5:         X ← PLAY_MACHINE(X)
```

Algorithm 2 UCB-based selection

```
1: function INIT_MACHINE
2:     UCB(i, j).trial ← 1                  ▷ 0 ≤ i < n and j ∈ {0, 1}
3:     UCB(i, j).reward ← 0.5
4: function PLAY_MACHINE(X)
5:     select pair of i, j such that max UCB = UCB(i, j).
6:     fix x_i to j (j ∈ {0, 1}).
7:     E ← f(X)
8:     X' ← ARBITRARY_METAHEURISTIC(X)
9:     E' ← f(X')
10:    UCB(i, j).trial ← UCB(i, j).trial + 1
11:    UCB(i, (j − 1)^2).trial ← UCB(i, (j − 1)^2).trial + 1
12:    if E' < E then
13:        UCB(i, j).reward ← UCB(i, j).reward + 1
14:    else if E' > E then
15:        UCB(i, (j − 1)^2).reward ← UCB(i, (j − 1)^2).reward + 1
16:    else
17:        UCB(i, j).reward ← UCB(i, j).reward + 0.5
18:        UCB(i, (j−1)^2).reward ← UCB(i, (j−1)^2).reward+0.5
19:    return X
```

$(j \in \{0, 1\})$ are updated at the same time because fixing x_i to 0 and fixing x_i to 1 are complementary.

B. Multi-bit Bandit-based Variable Fixing

The basic BVF fixes only one variable in each slot machine, and hence a search space is just bisected. We extend the basic concept of BVF so that it can fix multiple variables for dividing a search space further. Algorithm 3 shows the extended multi-bit algorithm. In addition to INIT_MACHINE() and PLAY_MACHINE(X), this algorithm includes SEMIPERMANENT_FIX(Y, Z). Once PLAY_MACHINE(X) with bit i fixed is executed, SEMIPERMANENT_FIX(Y) fixes bit i semi-permanently by updating $Y = y_0y_1 \cdots y_{n-1}$ so that $y_i = 1$. Hence, in the next iteration of the loop, bit j ($j \neq i$) is fixed and consequently two variables are fixed. In general, k bits are fixed in the k-th iteration.

Algorithm 3 Multi-bit bandit-based variable fixing on a graph

```
1: procedure BVF
2:     INIT_MACHINE()
3:     X ← 00···0                                  ▷ initial solution
4:     Y ← 00···0                                  ▷ y_i = 1 if x_i is fixed
5:     loop
6:         X ← PLAY_MACHINE(X, Y, Z)
7:         Y ← SEMIPERMANENT_FIX(Y)
8: function SEMIPERMANENT_FIX(Y)
9:     if x_i is fixed in the last play of a slot machine then
10:        y_i ← 1
11:    if it takes a certain iterations after x_i is fixed then
12:        y_i ← 0
13:    return Y
```

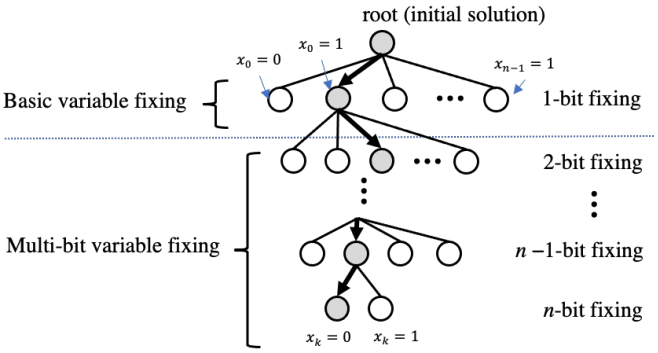


Fig. 2. Concept of a multi-bit BVF on a graph. In our proposed method, the graph is not actually a tree because it constitutes cycles when unfixing variables.

We can regard the multi-bit BVF as a walk on a graph as depicted in Fig. 2. While a basic BVF just repeats a walk from a root node to one of the adjacent $2n$ nodes, a multi-bit BVF further goes to distant nodes. For 2-bit fixing, we can reach $2(n+1)$ possible nodes. In general, the number of nodes corresponding to k-bit fixing is $2(n + k - 1)$. A trivial idea for solving n-bit QUBO by such a permanent variable fixing is to repeat the loop (lines 5–7 in Algorithm 3) n times as depicted in the figure. However, this prevents us to obtain good solutions because the value of x_i is improperly determined by the result of a limited duration of a metaheuristic. To address this issue, we introduce *semi-permanent* variable fixing with a certain duration (100 iterations in our evaluation). More specifically, we fix bit i in a certain number of iterations after bit i to fix (i.e., machine $2i$ or machine $2i + 1$) is selected. After a certain number of iterations end, the algorithm can select bit i to fix again. The value of y_i is reset to 0 at this point (line 12 in Algorithm 3).

C. Acceleration and Parallelization with GPU

We accelerate and parallelize BVF with NVIDIA RTX A6000 GPU and CUDA C++. CUDA programming model consists of multiple blocks, each of which has multiple threads. The blocks are assigned to a streaming multiprocessor (SM) and executed. In the case of RTX A6000 GPU, 84 SMs can execute up to $84 \times 1536 = 129024$ threads at the same time.

Algorithm 4 shows the parallel algorithm of our multi-bit BVF. Each block has one solution X and is responsible for a part of machines, and all of the blocks share information of fixed variables Y in a global memory. Thus, a host code executed in a CPU has an array $\boldsymbol{X}$ for all of the solutions and a shared variable Y. Furthermore, an additional variable $Z = z_0z_1 \cdots z_{n-1}$ is introduced for retaining the values of fixed bits. For example, if bit i is fixed to 0, we set $y_i = 1$ and $z_i = 0$. This information is necessary because every solution in the blocks must be updated so that it follows the variable fixing before a metaheuristic is executed.

In PLAY_MACHINE($\boldsymbol{X}, Y, Z$), which is a device code for GPU, each block independently executes a metaheuristic. Since each block is responsible for a part of variables to fix,

Algorithm 4 Parallel version of multi-bit BVF

procedure BVF ▷ host code
  INIT_MACHINE()
  for $i \leftarrow 0, \#$ of blocks $- 1$ **do**
    $\boldsymbol{X}[i] \leftarrow 00\cdots0$ ▷ initial solutions for the blocks
  $Y \leftarrow 00\cdots0$ ▷ $y_i = 1$ if x_i is fixed
  $Z \leftarrow 00\cdots0$ ▷ x_i is fixed to z_i if $y_i = 1$
  loop
    $\boldsymbol{X} \leftarrow$ PLAY_MACHINE($\boldsymbol{X}, Y, Z$) ▷ $\boldsymbol{X}$ is an array.
    $Y, Z \leftarrow$ SEMIPERMANENT_FIX(Y, Z)
function PLAY_MACHINE($\boldsymbol{X}, Y, Z$) ▷ device code
  update $\boldsymbol{X}$[blockID] if $y_i = 1$ and $z_i \neq x_i$ for any bit i assigned to this block
  select bit to fix from the bits assigned to this block based on UCB policy
  execute an arbitrary metaheuristic
  update UCB
  return $\boldsymbol{X}$[blockID]
function SEMIPERMANENT_FIX(Y, Z) ▷ device code
  select the best bit to fix, say x_i^{best}, based on the results of metaheuristics
  if this block is responsible for x_i **then**
    $y_i \leftarrow 1$
    $z_i \leftarrow x_i^{\text{best}}$
  if this block is responsible for x_i **then**
    $y_i \leftarrow 0$
  return Y, Z

the search is entirely different from those of the other blocks. This parallelization makes two advantages. First, it improves the solution quality because multiple blocks output different solutions. Second, updating UCB is accelerated.

SEMIPERMANENT_FIX(Y) starts with a selection of the variable to fix because there are multiple candidates given by the blocks. This selection policy is also important. We can use UCB also for this selection policy, but it is better to consider the quality of the solutions generated by each block. Thus, we select the best variable $x_i = x_i^{\text{best}}$ to fix from the block that obtains the best solution. The best solution is simply computed by a reduction operation using shared memory. Then it updates Y and Z so that they force to fix bit i to x_i^{best}. Also, it unfixes bit j if a certain number of iterations end since bit j is fixed.

IV. RESULTS

All of our evaluations are carried out in an environment with AMD EPYC 7502P CPU and NVIDIA RTX A6000 GPU. We compare original metaheuristics without variable fixing, basic BVF, and multi-bit BVF in terms of the solution quality and the execution time. Benchmark problems include the maximum cut (max-cut) problem and the traveling salesman problem (TSP), all of which are reduced to QUBO. The max-cut instances include G1 derived from G-set and K2000. The TSP instance gr24 is derived from TSPLIB [16].

First, we evaluate the solution quality of QUBO with existing metaheuristics: hill climbing and the cyclic-min algorithm [17]. We repeat the loop in Algorithm 4 n times. Figs. 3–5 show the transition of the objective function during the algorithm.

TABLE I
EXECUTION TIME (HILL CLIMBING).

Problem	Original	Basic BVF	Multi-bit BVF
TSP (gr24)	0.3337 s	0.4086 s (+21%)	0.4030 s (+19%)
Max-cut (G1)	0.7502 s	0.9573 s (+28%)	0.9630 s (+28%)
Max-cut (k2000)	6.410 s	7.326 s (+14%)	7.441 s (+16%)

Fig. 3 shows the solution quality of hill climbing. In all of the benchmark problems, an original hill climbing cannot find good solutions. The basic BVF (Algorithm 1) improves an original hill climbing because of the variable fixing, but it also cannot escape from a local optimal solution as suggested by the convergence. On the other hand, the multi-bit BVF perturbs the solution and consequently improves the solution quality. The improvement ratio over the original hill climbing is 1.72%, 6.82%, and 3.03%, respectively. This suggests that BVF is effective for a naive greedy local search.

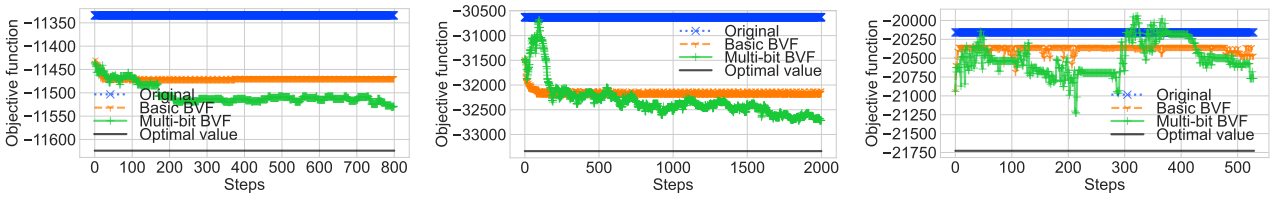
Fig. 4 shows the solution quality of tabu search, which adopts a tabu (forbidden) list of the bits. Once a bit is flipped, it is inserted the tabu list and it is not flipped during a certain number of iterations. As shown in Fig. 4, an original tabu search significantly outperforms hill climbing, but it still cannot find the optimal solution. For G1 max-cut problem, both basic and multi-bit BVF algorithms quickly find the optimal solution ($f(X) = -11624$). For K2000 max-cut problem that is fully-connected and larger problems than G1, the multi-bit BVF provides the best solution. Interestingly, the multi-bit BVF provides extremely bad solutions in the first 250 steps. We assume that this is because a UCB policy makes wrong decisions in the first some steps, but it gradually learns to make right decisions. For TSP problem, whose QUBO formulation is generally hard to solve, both BVF algorithms improve the solution quality. In particular, the multi-bit BVF successfully finds the optimal solution.

Fig. 5 shows the solution quality of the cyclic-min algorithm. This algorithm provides better solutions than hill climbing and tabu search. G1 max-cut problem is too easy for the cyclic-min algorithm, so BVF is unnecessary for obtaining the optimal solution. However, BVF improves the solution quality for K2000 max-cut problem and TSP, albeit only slightly. These results suggest that BVF potentially improves the solution quality also for sophisticated metaheuristics.

Next, we measure the execution time for evaluating overhead of BVF. Tab. I summarizes the representative results for hill climbing. As shown in the table, the overhead of the execution time is only 14–28 %, and there is no clear difference between basic and multi-bit BVF methods. The overhead occurs because of the additional computational costs regarding UCB computation. Note that it takes unacceptably long time if GPU is not used for BVF.

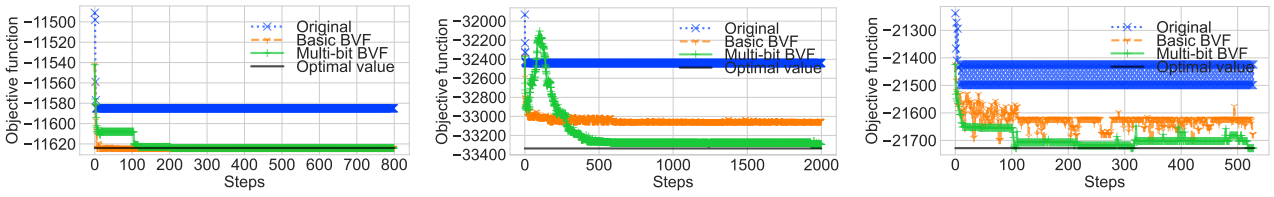
V. CONCLUSIONS

We have demonstrated that the multi-armed bandit problem is capable of enhancing metaheuristics for binary optimization. In particular, we propose a novel approach called a bandit-based variable fixing (BVF). Our results have shown that both



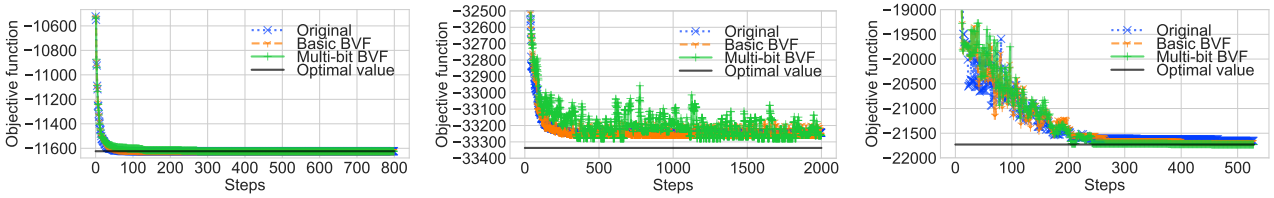
(a) Max-cut (G1) (b) Max-cut (K2000) (c) TSP (gr24)

Fig. 3. Solution quality (transition of the objective function): Hill climbing and its BVF extensions



(a) Max-cut (G1) (b) Max-cut (K2000) (c) TSP (gr24)

Fig. 4. Solution quality (transition of the objective function): Tabu search and its BVF extensions



(a) Max-cut (G1) (b) Max-cut (K2000) (c) TSP (gr24)

Fig. 5. Solution quality (transition of the objective function): Cyclic-Min algorithm and its BVF extensions

basic and multi-bit BVF improves the solution quality for hill climbing, tabu search, and the cyclic-min algorithm. The improvement ratio over original metaheuristics is up to 6.82%, and BVF allows a simple tabu search to find the optimal solution for G1 max-cut problem and gr24 TSP. The results suggest that the multi-bit BVF outperforms the basic BVF.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP22H05193.

References

[1] M. W. Johnson *et al.*, "Quantum annealing with manufactured spins," *Nature*, vol. 473, no. 7346, pp. 194–198, 2011.

[2] K. Tatsumura, A. R. Dixon, and H. Goto, "FPGA-based simulated bifurcation machine," in *International Conference on Field Programmable Logic and Applications (FPL)*, pp. 59–66, 2019.

[3] A. Mondal and A. Srivastava, "Ising-FPGA: A spintronics-based reconfigurable Ising model solver," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 1, pp. 1–27, 2020.

[4] H. Kagawa *et al.*, "Fully-pipelined architecture for simulated annealing-based QUBO solver on the FPGA," in *International Symposium on Computing and Networking*, pp. 39–48, 2020.

[5] T. Okuyama *et al.*, "Binary optimization by momentum annealing," *Physical Review E*, vol. 100, no. 1, 2019.

[6] R. Yasudo *et al.*, "Adaptive bulk search: Solving quadratic unconstrained binary optimization problems on multiple GPUs," in *49th International Conference on Parallel Processing-ICPP*, pp. 1–11, 2020.

[7] M. Yamaoka *et al.*, "20k-spin Ising chip for combinational optimization problem with CMOS annealing," in *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 1–3, 2015.

[8] K. Kawamura *et al.*, "Amorphica: 4-replica 512 fully connected spin 336mhz metamorphic annealer with programmable optimization strategy and compressed-spin-transfer multi-chip extension," in *International Solid-State Circuits Conference (ISSCC)*, 2023.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[10] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.

[11] C. B. Browne *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[12] A. Lucas, "Ising formulations of many NP problems," *Frontiers in Physics*, vol. 2, p. 5, 2014.

[13] D. Arthur *et al.*, "QUBO formulations for training machine learning models," *Scientific reports*, vol. 11, no. 1, pp. 1–10, 2021.

[14] T. Imanaga *et al.*, "Solving the sparse qubo on multiple gpus for simulating a quantum annealer," in *International Symposium on Computing and Networking (CANDAR)*, pp. 19–28, 2021.

[15] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

[16] G. Reinelt, "TSPLIB: A traveling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.

[17] R. Yasudo *et al.*, "GPU-accelerated scalable solver with bit permutated cyclic-min algorithm for quadratic unconstrained binary optimization," *Journal of Parallel and Distributed Computing*, vol. 167, pp. 109–122, 2022.