THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU) | RESEARCH DEPARTMENT

2024 HKUST(GZ) Undergraduate Research Program (UGRP)

Research Report

Tropical neural network for reasoning

BY

Li Xiaofeng
50013382

SUPERVISED BY

Jinguo Liu

By submitting the Research Report you confirm you have read and understood the Hong Kong University of Science and Technology (Guangzhou) Regulations for Student Conduct and Academic Integrity and Policy on Research Conduct and Integrity.

SOLVING THE FACTORING PROBLEM WITH AN ISING MACHINE

Abstract

This project investigates the use of Ising machines to tackle the prime factorization problem, essential for the security of RSA encryption systems. We develop the `ProblemReductions.jl` package to reduce complex problems into Ising models, enabling efficient solutions through Ising machines. We demonstrate the process of solving the factoring problem using an Ising machine with the assistance of `ProblemReductions.jl`. This work provides a practical way to leverage the power of Ising machines for factoring problems and builds a reduction framework for NP problems.

1. Introduction: Ising machine and the factoring problem

Ising machines are powerful hardware solvers that are used to find out the optimal solutions for Spin Glass problems. People have employed multiple kinds of algorithm, like Simulated Annealing and Tensor Networks, to realize Ising machine [1]. For example, Figure 1 (a) shows a Boltzmann machine and Figure 1 (b) shows a commercial quantum annealing Ising machine from D-Wave Systems Inc. With an Ising machine, people could solve Spin Glass problems much faster than traditional computers with von-Neuman architecture.

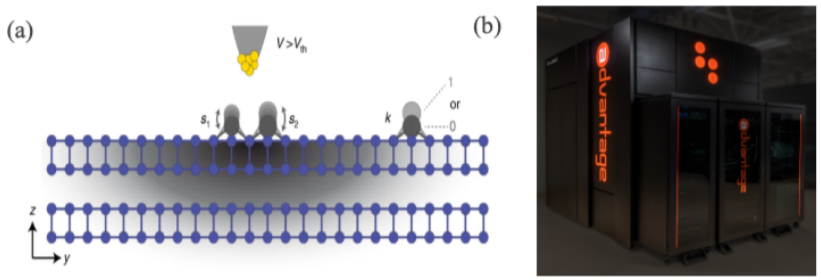


Figure 1. (a) A Boltzmann machine[2]. (b)Quantum Annealing Ising machine from D-Wave Systems Inc[3].

The ability of Ising machines to solve Spin Glass problems is not that powerful in the context that many algorithms could only solve a single type of problem and when nuanced differences are introduced, the algorithm does not work well. However, in the context of computational complexity, people only care about complexity class of problems such as P, NP, and NP-complete problems class shown in Figure 2. The P problems are the problems that could be solved in polynomial time, the NP problems are the problems that could be verified in polynomial time, and the NP-complete problems, where Spin Glass problem belongs to, are the hardest problems in NP. The basic principle of problem reduction is that if a problem A could be reduced to problem B, then problem B is at least as hard as problem A so that problems of same complexity class can be reduced to one another in the time polynomial to the problem size [4]. Therefore, since many optimization problemms are in NP-complete, it is possible to formulate any NP problem into a Spin Glass problem [5] and that is where the Ising machine comes in handy.

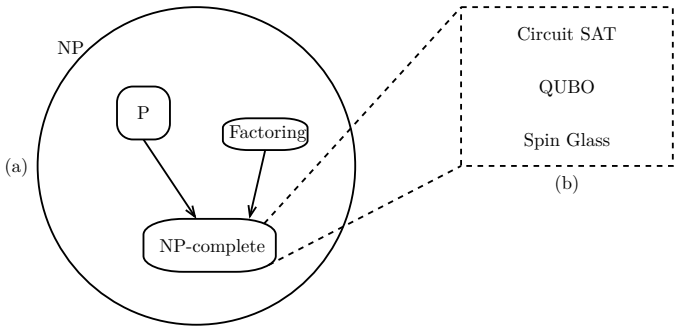


FIGURE 2. (a) A possibe diagram of P, NP, and NP-complete problems [6] and their reduction relations. The circles represent the classes of problems. The arrow represents the reduction from one kind of problem to another. (b) Several Problems in the package `ProblemReductions.jl`.

Among these problems, the factoring problem is one of the most important problems in number theory and cryptography such as RSA system [7, 8]. It is in NP-intermediate as shown in Figure 2, which means it is neither in P nor in NP-complete. Solving the factoring problem could lead to breakthroughs in analyzing the vulnerabilities of the RSA system. And up to now, people have developed many ways to solve the factoring problem, for example, reducing it to the closest lattice vector problem then use *Shortest Vector Problem*(SVP) algorithm to solve it [9], and using a primitive method — encoding it into an array multiplier [10], both of which could be reduced to the Ising model.

Unfortunately, although there are already many well-established reductions for NP problems [5], we don't have a clear and usable way to reduce factoring to Ising model. This motivates the development of our package, `ProblemReductions.jl`, which not only provides a practical way to bridge this gap, enabling us to leverage the power of Ising machines for factoring problems, but also build up the reduction framework of NP problem. In this article, we aim to demonstrate the process of solving a factoring problem using an Ising machine with the assistance of ProblemReductions.jl, as depicted in Figure 3. Since we do not have access to actual hardware, we employ the `GenericTensorNetworks` package instead, which provides a software-based Ising machine solver.

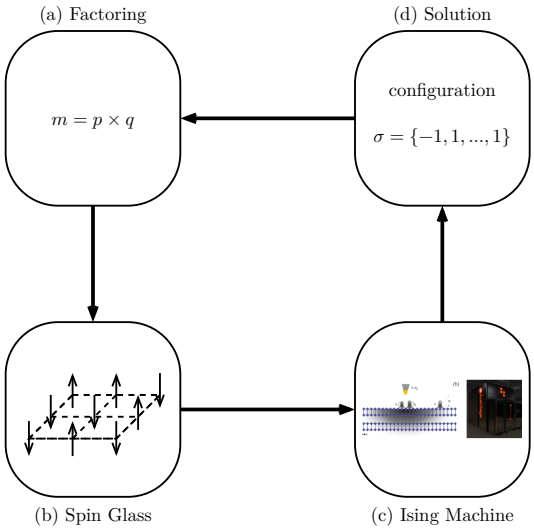


FIGURE 3. Process of solving a factoring problem by Ising machine using `ProblemReductions.jl`. (a) Factoring problem needed to solve. (b) Through `ProblemReductions.jl`, we reduce the factoring problem to a corresponding Spin Glass problem. (c) The Ising machine is used to solve the Spin Glass problem. (d) Extract the solution to the factoring problem through `ProblemReductions.jl`.

2. From factoring to Ising machine

2.1. **Factoring problem.**

Factoring, a problem of decomposing an n-bit composite integer m into its prime factors p and q, is denoted as $m = p \times q$. To specify it, we use the binary representation for the integer $m = \sum_{i=0}^{n-1} 2^i m_i$, with $m_i \in \{0, 1\}$, $p = \sum_{i=0}^{k-1} 2^i p_i$ for the k-bit integer, and $q = \sum_{i=0}^{n-k-1} 2^i q_i$ for the $(n-k)$-bit integer. The factoring problem thus amounts to finding the unknown bits p_i and q_i such that

$$\sum_{i=0}^{n-1} 2^i m_i = \sum_{i=0}^{k-1} \sum_{j=0}^{n-k-1} 2^{i+j} p_i q_j. \tag{1}$$

Note that k is a priori unknown since we just want to consider this specific problem. However, one could consider this problem for any $k = 1, 2, ..., \frac{n}{2}$ [10].

2.2. **Factoring → Circuit Satisfaction.**

The factoring problem can be reduced to the *Circuit Satisfaction*(Circuit SAT) problem. Circuit SAT is a problem of determining whether a given boolean circuit has a satisfying assignment. Hence, factoring is the problem of finding the satisfying assignment of the integer multiplication circuit. In computer science, the multiplication of two integers is often implemented using an *array multiplier* as shown in Figure 4. The basic idea of the array multiplier is to multiply each bit of the multiplicand with every bit of the multiplier and then add the partial products to obtain the final product [11]. The building block of the array multiplier is composed of an AND gate and a full adder as shown in Figure 4. To clarify it, we consider a simple multiplication of 3-bits binary numbers

$$5 \times 7 = 35 \rightarrow 111 \times 101 = 100011, \tag{2}$$

the graphical representation of its vertical calculation and array multiplier is shown in Figure 4.

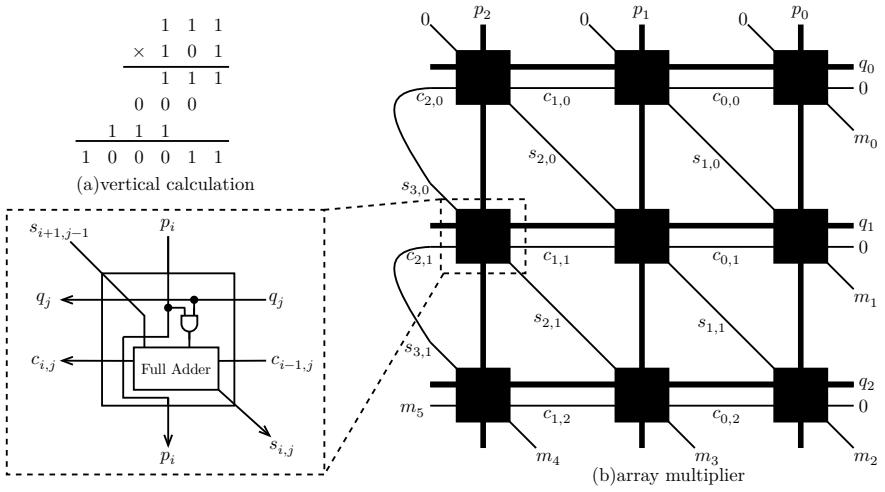


FIGURE 4. (a) The multiplication of 3-bits binary numbers. The vertical calculation of 7 times 5 is shown. (b) The array multiplier [10]. It multiplies each bit of the multiplicand with the multiplier bits to generate partial products, which are then summed for the final product. (c) The building block of array multiplier (Equation 3). Each building block is composed of an AND gate and a full adder.

Observing the array multiplier and blackbox in Figure 4, it is obvious that each blackbox contains several constraints to its input. Here, for such a $n \times n$ multiplier, we could define the constraint for each blackbox as:

$$\begin{aligned} s_{i,j} + 2c_{i,j} &= p_i q_j + c_{i-1,j} + s_{i+1,j-1} \\ c_{-1,j} &= s_{i,-1} = 0 \end{aligned} \tag{3}$$

for $i, j \in \{0, 1, ..., n\}$, let c_{ij} and s_{ij} represent the carry-out and sum-out, p_i and q_j denote the ith bit of the multiplicand and the jth bit of the multiplier, and $c_{i-1,j}$ and $s_{i,j-1}$ refer to carry-in and sum-in, where $i-1$ and $j-1$ simply indicates that each blackbox receives carry-out from the last blackbox in the same row, and sum-out from the upper-left blackbox(which, in vertical calculation corresponds to the previous column) [10].

A boolean circuit is a directed graph with input nodes and one or more output node. The internal nodes, known as "gate", produce logical function of inputs. One could devide a circuit into a series of layers of gates and the gates from each layer receive inputs from the layer above them [4]. Based on this "layer" design, we could reduce factoring to Circuit SAT simply by pushing all the constraints in the blackbox to certain layers in the circuit.

2.3. Circuit Satisfaction → QUBO.

Firstly, we give a formal definition of the *Quadratic Unconstrained Binary Optimization* (QUBO) model. Definition: The QUBO model is expressed by the optimization problem:

$$\text{QUBO: minimize/maximize } y = x^t Q x \tag{4}$$

where x is a vector of binary decision variables and Q is a square matrix of constants. It is worth noticing that the there are distinct differences between QUBO and Circuit SAT since, firstly, the former is a optimization problem and the latter is a decision problem and secondly, QUBO is unconstrained and Circuit SAT is constrained. However, by constructing Q properly, one could implicitly introduce the constraints into the QUBO model.

For constrained optimization problems, quadratic penalties are introduced to simulate the constraints so that the constraints problems could be re-formulated into QUBO problems effectively [12]. As an example, we consider the logical operations that are common in Circuit SAT and their corresponding QUBO penalties in Table 1.

Logical Operation	**QUBO Penalty**
$z = \neg x$	$2xz - x - z + 1$
$z = x_1 \vee x_2$	$x_1 x_2 + (x_1 + x_2)(1 - 2z) + z$
$z = x_1 \wedge x_2$	$x_1 x_2 - 2(x_1 + x_2)z + 3z$

Logical Operation	**QUBO Penalty**
$z = x_1 \oplus x_2$	$2x_1x_2 - 2(x_1 + x_2)z - 4(x_1 + x_2)a + 4az + x_1 + x_2 + z + 4a$

TABLE 1. QUBO Penalties for Logical Operations [13]

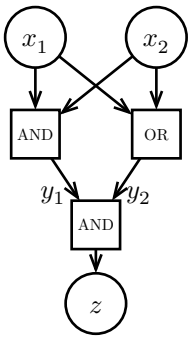
In Table 1, all the variables are intended to be binary value and note that in that case, we have

$$x_i^2 = x_i \tag{5}$$

and thereby we could transform the linear part into quadratic one [12]. For example, for the NOT operation, we could obtain its QUBO expression as follows:

$$z = \neg x \to y = 2xz - x^2 - z^2 + 1 \to y = [x \;\; z]\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} x \\ z \end{bmatrix} \tag{6}$$

For each truth assignment of the variables, the penalty would be 0 if the logical operation is satisfied and be larger than 0 otherwise. By checking whether the penalty is 0, we could determine whether the logical operation is satisfied. Given this penalties gadgets, we process by considering a simple conjunction of two gedgets. Given a Circuit SAT example



(a)

In (a), there is a simple conjuction of an AND gate and an OR gate. The output of them, y_1 and y_2, is then connected to the AND gate in the next layer as inputs. The conjuction of the two gates in the first layer is given by:

$$z = y_1y_2 - 2(y_1 + y_2)z + 3z \tag{7}$$

So by simply change the literals in the QUBO penalty, we could simulate the conjuction of two gadgets and therby the Circuit SAT problem. The whole reduction of (a), from Circuit SAT to QUBO, is given by:

$$y_1 = x_1x_2 - 2(x_1 + x_2)y_1 + 3y_1 \tag{8}$$

$$y_2 = x_1x_2 + (x_1 + x_2)(1 - 2y_2) + y_2 \tag{9}$$

and Equation 7

2.4. **QUBO → Spin Glass and Ising machine.**

2.4.1. *Reduction from QUBO to Spin Glass.*

Spin Glass problem is of great interest both in solid state physics and in statistical physics. Simply speaking, in a Spin Glass system, there are many spins that interact with each other and the onsite energy, such as magnetic field. In spin glss, there is an energy interaction between spins: $H_{12} = J_{12}\sigma_1\sigma_2$, where σ_1 and σ_2 are

the spins and J_{12} is the interaction [14]. For a configuration $\sigma = \{\sigma_1, \sigma_2, ..., \sigma_n\}$, where $\sigma_i = \{+1, -1\}$, the energy of the configuration is given by the Hamiltonian

$$H = \sum_{i,j} J_{ij}\sigma_i\sigma_j + \sum_i h_i\sigma_i. \tag{10}$$

The parameters J and h correspond to the energies associated with spins' interactions with other spins and the external field, respectively. Note that in Equation 10 we set the sign to be positive to keep consistent with QUBO's penalty rules. For a ferromagnet, J is positive and a configuration with most interacting spins having parallel moments ($\sigma_i = \sigma_j$) has lower energy and vice versa. Here we notice that the Hamiltonian is composed of quadratic sum term and linear sum term, which is close to a QUBO form except that spin's value is not binary. A simple way to convert it is by setting $x_i = \frac{1-2\sigma_i}{2}$ and substitute it into the QUBO penalty, after which we could obtain the Hamiltonian.

Consider a simple example of QUBO: $\min y = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, which is the QUBO penalty of $x_2 = \neg x_1$. We substitute $x_i = \frac{1-2\sigma_i}{2}$ into the penalty and we have:

$$y = \begin{bmatrix} \frac{1-\sigma_1}{2} & \frac{1-\sigma_2}{2} \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \frac{1-\sigma_1}{2} \\ \frac{1-\sigma_2}{2} \end{bmatrix} \rightarrow y = -\frac{1}{4}\sigma_1^2 - \frac{1}{4}\sigma_2^2 + \frac{1}{2}\sigma_1\sigma_2. \tag{11}$$

Note that $\sigma_i \in \{-1, 1\}$, so the square terms are actually constants. Therefore, we could simplify it and obtain the Hamiltonian of the target Spin Glass problem as:

$$H = \frac{1}{2}\sigma_1\sigma_2. \tag{12}$$

We could verify its correctness by comparing the solution of both problems. For QUBO, the solution is $x_1 = 1, x_2 = 0$ and for the Spin Glass, the solution is $\sigma_1 = -1, \sigma_2 = 1$. Through the conversion, we could see that the two solutions are consistent.

2.4.2. *Ising machine.*

Ising model is actually a simplified version of Spin Glass model. The spins in Ising model would assumes one of the two values, $+1$ or -1, to settle themselves in the lowest energy state with numerous alternatives in the process.The Ising machine is designed to mimic this process and find the optimal solutions for the model. Many algorithms are used to realize the Ising machine, such as simulated annealing, quantum annealing, and tensor network.

In a nutshell, Ising machine is a powerful tool for Ising model. After the whole reduction process, Ising machine could help us obtain a solution of target problem and through extracting the solution, we could get the solution for factoring.

3. Code Implementation

3.1. **Julia programming language.**

Julia is a modern, open-source, high performance programming language for technical computing. It was born in 2012 in MIT. Though Julia is new, it has a large number of packages and a strong and fresh community(JuliaHub Inc).

Julia is fast. Its feature of Just-In-Time compilation and its types system keep it from typical compilation and execution uncertainties [15]. At the same time, multiple dispatch feature allows the package based on Julia to be more flexible and

extensible, which is not only important for the open-source community but also fit for requirements of the field of optimization problems.

For more information, refer to following links:

- Official website: JuliaLang
- Julia Introduction: ThinkJulia.jl, Scientific Computing for Physicists

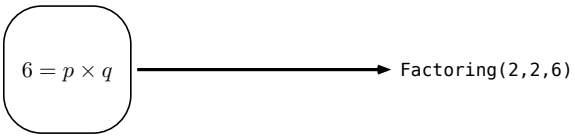
3.2. **ProblemReductions.jl.**

In this section, we will introduce how to use `ProblemReductions.jl`. The main function of the package is problem reduction. It defines a set of computational hard problem(`models`) and provides feasible interface(`reduction_graph` and `reduceto`) to reduce one to another. Here is an example of reduce a factoring problem to a Spin Glass problem through the package.

Consider factoring problem: $6 = p \times q$, note that in the package, the parameters for factoring problem is `m`, `n` and `input` where `m` and `n` is the number of bits for the factors and `input` is the number to be factored. Open a Julia REPL and run the following code:

```julia
julia> using ProblemReductions  #import the package

julia> factoring = Factoring(2, 2, 6) # 3 bits factors and 6 as input
Factoring(2, 2, 6)
```



When we initialize an instance, not only `Factoring`, we need to offer some information about the problem. For `Factoring`, we need to provide the number of bits for the factors and the number to be factored. And the outcome would be a `Factoring` instance with these information.

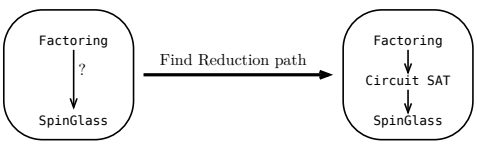
The next thing is to find out how to reduce the factoring to Spin Glass.

```julia
julia> g = reduction_graph(); # get the reduction graph

julia> path = reduction_paths(Factoring, SpinGlass)
1-element Vector{Vector{Type}}:
 [Factoring, CircuitSAT, SpinGlass]

julia> reduction_result = implement_reduction_path(path[1], factoring);

julia> target = target_problem(reduction_result)
SpinGlass{HyperGraph, Vector{Int64}}(HyperGraph(90, [[1, 2], [1, 3], [2, 3], [1], [2], [3], [4], [3, 4], [3, 5], [3, 6]  …  [84, 88], [82, 88], [88], [83, 89], [63, 89], [89], [88, 89], [88, 90], [89, 90], [90]]), [1, -2, -2, 3, 3, 0, 0, 1, -1, -2  …  -2, -2, -3, -2, -2, -3, 1, -2, -2, 1])
```



The `reduction_graph` function returns a graph, where each vertex represent a `model` and each edge represent a reduction from one model to another. The `reduction_paths` function returns a list of paths from the source model to the target

model. Here we get a vector of problems: `[Factoring, CircuitSAT, SpinGlass]`. That means we need to reduce the factoring problem to a circuit satisfaction problem and then to a Spin Glass problem. The `implement_reduction_path` function is used for problem reductions and it would return the result of the reduction. It's worth noticing that the reduction result is an instance of `AbstractReductionResult` class, which contains the information of the reduction process, not just single problem instance. The `target_problem` function is used to get the target problem from the reduction result. In this case, the target problem is a `SpinGlass` instance.

So the basic programs to implement the problem reduction using `ProblemReductions.jl` are as follows:

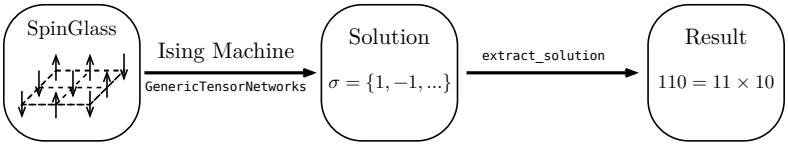
- Initialize the source problem and offer the priori known information.
- Initialize the reduction graph using `reduction_graph()`.
- Get the reduction paths using `reduction_paths(source, target)`, note that the source and target should be the type of the source and target problem.
- Implement the reduction path using `implement_reduction_path(path, source)`.
- Get the target problem using `target_problem(reduction_result)`.

```
julia> import GenericTensorNetworks  # solver                                    Julia

julia> gtn_problem = GenericTensorNetworks.SpinGlass(
                       target.graph.n,
                       target.graph.edges,
                       target.weights
                     )

julia> result = GenericTensorNetworks.solve(
                       GenericTensorNetwork(gtn_problem),
                       SingleConfigMin()
                     )[]
(-92.0, ConfigSampler{44, 1, 1}(10000000000000110101000001010010000011010000))ₜ

julia> extract_solution(reduction_result, 1 .- 2 .* Int.(read_config(result)))
4-element Vector{Int64}:
 1
 1
 0
 1
```



The result is $p = 3$ and $q = 2$ which is the correct factors of 6. The code above shows how to reduce a factoring problem to a Spin Glass problem and solve it using the `GenericTensorNetworks` package. The `GenericTensorNetworks` package is a solver for the Ising machine and it provides a set of solvers for the Ising machine. The `SingleConfigMin` is a solver that finds the minimum energy configuration of the Ising machine. The `extract_solution` function is used to extract the solution from the result of the solver. The solution is then used to find the factors of the input number.

4. CONCLUSIONS AND OUTLOOK

In this work, we have presented an efficient reduction framework to solve the factoring problem using Ising machines, with ProblemReductions.jl serving as the core tool for problem transformation. The process leverages the power of tensor networks and other optimization techniques to tackle Spin Glass problems, which can be embedded into hardware solutions. Although the reductions demonstrated are practical and adaptable, there are still limitations. For instance, the scalability of the reduction graph, leave room for improvement.

Looking forward, several directions could be explored. Firstly, for the package, the `reduction graph` could be extended to increase the robustness of the reduction system.In terms of the reduction complexity, the reduction paths and rules could be optimized to reduce the size of intermediate elements. What's more, better Ising machine solver could be employed to improve the efficiency of the solution.

5. Acknowledgements

Thank the authors of the package `ProblemReductions.jl`: Jin-guo Liu, Chen-guang Guan, and Huaiming Yu for their contributions to the package. Thank Huanhai Zhou for advices on the report writing. We acknowledge the funding support provided by Research Department of HKUST(GZ) through UGRP program.

6. Appendix

6.1. My Contributions to the `ProblemReductions.jl`.

6.1.1. *GitHub repository.* ProblemReductions.jl

6.1.2. *Activities.* My contribution the the package as *c-allergic* during the summer vacation:

- Code Contributions

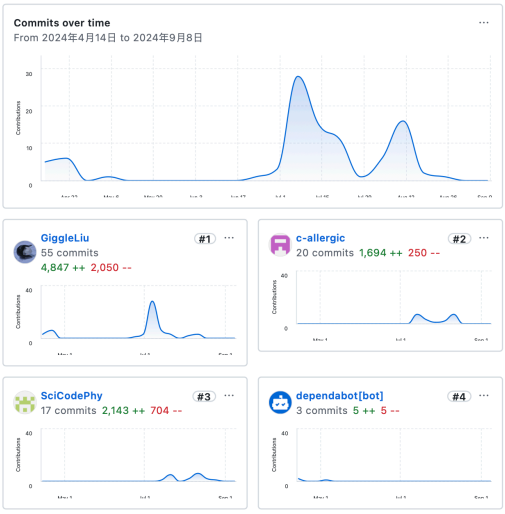


Figure 5. Commits of the package ProblemReductions.jl: totally added 1694 lines of code and deleted 250 lines of code.

6.1.3. *Pull Requests.*
13 merged pull requests in total.

- New: PaintShop model

- fix: parameters and set_parameters in Matching.jl and add tests
- New: matching model
- New: Maximal Independent Set
- Fix: xor symbol and others bugs
- Add: Reduction from SpinGlass{HyperGraph} -> MaxCut
- Clean up: unnecessary type deduction
- New: reduction from Sat to Coloring
- New: Reduction from vertex covering to set covering
- New: Vertex Covering model
- New: Reduction between simple kinds of SpinGlass and MaxCut problems
- New: MaxCut model
- Set covering

References

1. Mohseni, N., McMahon, P. L., Byrnes, T.: Ising machines as hardware solvers of combinatorial optimization problems. Nature Reviews Physics. 4, 363–379 (2022)
2. Kiraly, B., Knol, E. J., Weerdenburg, W. M. van, Kappen, H. J., Khajetoorians, A. A.: An atomic Boltzmann machine capable of self-adaption. Nature Nanotechnology. 16, 414–420 (2021)
3. D-Wave Systems Inc.: Media Resources, https://www.dwavesys.com/company/media-resources/
4. Moore, C., Mertens, S.: The nature of computation. Oxford University Press (2011)
5. Lucas, A.: Ising formulations of many NP problems. Frontiers in physics. 2, 5–6 (2014)
6. Garey, M. R., Johnson, D. S.: Computers and intractability. freeman San Francisco (1979)
7. Rivest, R. L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM. 21, 120–126 (1978)
8. Aggarwal, D., Maurer, U.: Breaking RSA generically is equivalent to factoring. In: Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings 28. pp. 36–53 (2009)
9. Schnorr, C. P.: Fast factoring integers by SVP algorithms, corrected. Cryptology ePrint Archive. (2021)
10. Nguyen, M.-T., Liu, J.-G., Wurtz, J., Lukin, M. D., Wang, S.-T., Pichler, H.: Quantum optimization with arbitrary connectivity using Rydberg atom arrays. PRX Quantum. 4, 10316–10317 (2023)
11. Asha, K., Shinde, K. D.: Performance analysis and implementation of array multiplier using various full adder designs for DSP applications: A VLSI based approach. In: Intelligent Systems Technologies and Applications 2016. pp. 731–742 (2016)
12. Glover, F., Kochenberger, G., Hennig, R., Du, Y.: Quantum bridge analytics I: a tutorial on formulating and using QUBO models. Annals of Operations Research. 314, 141–183 (2022)
13. Reformulating a Problem — D-Wave System Documentation documentation, https://docs.dwavesys.com/docs/latest/handbook_reformulating.html
14. Barahona, F.: On the computational complexity of Ising spin glass models. Journal of Physics A: Mathematical and General. 15, 3241–3242 (1982)
15. Bezanson, J., Edelman, A., Karpinski, S., Shah, V. B.: Julia: A fresh approach to numerical computing. SIAM review. 59, 65–98 (2017)