Ising Machines With Feasibility Guarantee

Max Bannach*[1]

[1]*European Space Agency, Advanced Concepts Team, Noordwijk,The Netherlands*

Ising machines are promising new hardware accelerators whose use is studied in many space applications, such as satellite scheduling and trajectory optimization. These accelerators can only solve unconstrained problems, which makes programming them challenging and unnatural. Previous studies have revealed that common tricks to encode constraints into these unconstrained problems lead to numeric instabilities and even infeasible solutions.

We propose a solution to both issues by relying on the well-established paradigm from symbolic artificial intelligence to decouple reasoning and optimization. To that end, we use an Ising machine (for optimization) in conjunction with a SAT solver (for reasoning) and develop an algorithm for the maximum satisfiability problem based on the implicit hitting set approach. We argue that it is more natural to use maximum satisfiability as general-purpose language, prove that our algorithm is guaranteed to output a feasible solution, and provide a prototype implementation that experimentally shows the advantages and disadvantages of the approach. In this sense, the proposed algorithm can be seen as a new interface to Ising machines that avoids the direct use of the quadratic unconstrained binary optimization problem.

1 Introduction

Many problems in early-stage mission design, the maintenance of communication infrastructure, and space logistics are computationally intractable. For instance, designing active derbies removal missions or missions to the asteroid belt to visit multiple asteroids requires solutions to variants of the famous Traveling Salesperson problem (TSP) [1, 2]; tracking space crafts with the Deep Space Network (DSN) involves various scheduling problems [3]; and space logistics involves many classical combinatorial problems such as flow problems (to model the flow of materiel, services, and information) and facility location problems (such as the vertex cover and dominating set problem) [4].

Example 1. *As a running example, let us consider (a simplified version of) the* satellite scheduling problem. *For a detailed description of real data, see [3]. We are given a set S of* satellites, *a set G of* ground stations, *a set T of discrete* time windows, *and a set $R \subseteq S \times G \times T$ of communication options, e.g., $(s, g, t) \in R$ means that satellite s could communicate with ground station g at time t. The goal is to find a schedule $X \subseteq R$ of* maximum *size such that no satellite is scheduled twice and such that no ground station is used twice at the same time, i.e.:*

$$(s, g, t), (s', g', t') \in X \Rightarrow (s \neq s') \wedge (g \neq g' \vee t \neq t').$$

Due to the increasing number of parties entering the new space sector, solving problems such as satellite scheduling becomes increasingly important on ever larger instances. Exact methods often fail to scale to instances of industrial size and, thus, many researchers have focused on heuristics [5–7].
An alternative are *hardware accelerators* based on *non-classical hardware*. A promising line of research are *Ising machines* or *Ising processing units* (IPUs) [8], specialized chips that compute the *ground state* of an *Ising model*, i.e., a vector $(x_1, \ldots, x_n) \in \{0,1\}^n$ that minimizes

$$\psi(x_1, \ldots, x_n) := \sum_{i=1}^{n} w_{i,i} x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{i,j} x_i x_j + c$$

for *weights* $w_{i,j} \in \mathbb{R}$ and a $c \in \mathbb{R}$. The problem of finding the ground state is called *quadratic unconstrained binary optimization* (QUBO). Ising machines are explored in many space domains, e.g., active debris removal [9], trajectory optimization [10], near-Earth object classification [11], and satellite scheduling [3, 12].

The perhaps best-known IPU is the quantum annealer by D-Wave, which claims to find the ground state using quantum mechanical effects [13]. But an Ising machine is not necessarily quantum. For instance, Fujitsu and Toshiba are developing *digital annealers* based on CMOS annealing and the Markov chain Monte Carlo method [14, 15]. *Neuromorphic hardware* such as Intel's Loihi [16] or IBM's TrueNorth system [17] can also be used as an Ising machine [18].

Example 2. *In order to solve a problem such as the satellite scheduling problem with an* IPU, *we have to* encode *it as a* QUBO. *Following the notation of Example 1, we can encode the satellite scheduling problem by building a*

*Corresponding author. E-Mail: max.bannach@esa.int

conflict graph G *with vertex set* $V(G) = R$ *and edge set:*

$$E(G) = \{(s,g,t),(s',g',t') \mid (s = s') \vee (g = g' \wedge t = t')\}.$$

An independent set, *i.e., a set* $S \subseteq V(G)$ *of pairwise non-adjacent vertices, of* G *corresponds to a schedule* X*. The problem can be encoded as* QUBO *by introducing for every* $v \in V(G)$ *a variable* x_v *with the semantic* $x_v = 1 \Leftrightarrow v \in S$*. For a large* penalty term ρ*, the* QUBO *encoding is:*

$$\psi_{is} := \underbrace{-\sum_{v \in V(G)} x_v}_{\text{maximize the size of } S} + \underbrace{\rho \sum_{uv \in E(G)} x_u x_v}_{\text{do not select both endpoints of an edge}}.$$

The example illustrates some issues with this new technology. First, the encoding into QUBO is somewhat artificial as we are used to encode problems using *constraints*. In the example, we were not allowed to select two adjacent vertices, which we encoded using the *penalty method*.

I1 Encodings into QUBO require penalty terms.

Penalties lead to numerical errors and reduce the convergence speed. This reinforces the second issue of Ising machines: They are *not* optimal, i.e., an IPU is not guaranteed to output the global minimum.

I2 Ising machines can output suboptimal solutions.

Both issues together imply a third and more severe issue: In order to encode constraints, the penalty method reasons along the lines of "Every solution that does not satisfy the constraint sacrifices so much in the objective function that an *optimal* solution cannot afford to falsify the constraint." However, this reasoning does not hold for *suboptimal* solutions, i.e., an Ising machine can output *infeasible solutions* [19]:

I3 Ising machines can output infeasible solutions.

In an application such as satellite scheduling, **I2** means a waste of resources, e.g., we could downstream more data. In contrast, **I3** means a *loss of data* since multiple satellites would try to communicate with the same ground station simultaneously. To overcome this issue, one must apply post-processing to fix unsatisfied constraints [19, 20]. This, however, must be done on a problem-to-problem basis and, hence, Ising machines are not full *general-purpose solvers* as one does not only need to encode the problem but also prepare dedicated algorithms to fix the solution:

I4 Ising machines are not general-purpose solvers.

1.1 Our Contribution

While issue **I2** is strictly bound to the properties of Ising machines, we tackle issues **I1**, **I3**, and **I4** by integrating an Ising machine into established automated reasoning algorithms. In particular, we develop the *Implicit Ising Hitting Set* algorithm (I^{2}HS) by integrating an Ising machine into the well-known *implicit hitting set* algorithm for *maximum satisfiability*. The matured formalism of MAX-SAT does not suffer from **I1**, and we propose a QUBO encoding for the hitting set problem that, with a simple post-processing routine, is guaranteed to produce *feasible hitting sets*. Hence, our hybrid algorithm always produces feasible MAX-SAT solutions and does not suffer from **I3**.

We supplement our theoretical findings with a *prototype* implementation using PySAT [21] and the Fixstars Amplify Annealing Engine [22]. Experiments show that this approach, indeed, always outputs feasible solutions and, furthermore, often produces optimal solutions.

2 Implicit Hitting Sets With IPUs

Instead of using QUBO as formalism, we use the well-established language of *maximum satisfiability*. The reader does not need to be familiar with the logical background (which we briefly introduce) to follow the main idea of the proposed algorithm. We refer to the *Handbook of Satisfiability* for an introduction [23]. The input of the maximum satisfiability problem (MAX-SAT) is a propositional formula in conjunctive normal form. Each clause has a *weight* $w \in \mathbb{R}^+ \cup \{\infty\}$, where ∞ indicates a *hard* constraint, while the other clauses are *soft*. The goal is to find an assignment of the variables to $\{0,1\}$ such that all hard clauses are satisfied and such that the sum of the weights of the satisfied soft clauses is maximized.

Example 3. *We denote weighted formulas by notating the corresponding weights as the exponent of every clause:*

$$\phi = (x_1 \vee x_2 \vee x_3)^2 \wedge (\neg x_1 \vee x_2)^\infty \wedge (x_1 \vee \neg x_2)^\infty \wedge (\neg x_1 \vee \neg x_2)^5 \wedge (x_1 \vee x_2 \vee \neg x_3)^{120} \wedge (x_3)^{100}.$$

The maximum is achieved by setting all variables to 1.

Since MAX-SAT provides soft and hard constraints, it is easy and natural to encode problems into it. In particular, it does *not* suffer from **I1**.

Example 4. *The encoding for the satellite scheduling problem from Example 2 can be expressed in* MAX-SAT*:*

$$\phi_{is} := \bigwedge_{v \in V(G)} (x_v)^1 \wedge \bigwedge_{uv \in E(G)} (\neg x_u \vee \neg x_v)^\infty.$$

We utilize the *implicit hitting set* algorithm [24], which famously "decouples" logical reasoning and optimization by letting a SAT solver extract so-called

cores (the reasoning part) and by encoding the optimization part into a *hitting set* instance [25].

We briefly review the main idea: In order to solve MAX-SAT with the implicit hitting set approach, we assume that the given formula has the special form in which all soft clauses are positive unit clauses, i.e., they contain exactly one positive literal. This normal form can easily be achieved by introducing relaxation variables, e.g., if c is a soft clause $c = (\neg x)$ or with $|c| > 1$, we replace c with the *hard* clause $(c \vee \neg r)$ and the soft clause (r) of weight $w(c)$ for a fresh variable r. Let then R be the set of all relaxation variables. Modern SAT solvers can be executed with a set A of literals as *assumptions*, which just means that the solver sets these literals to true. Crucially, if the formula under some assumptions is *not* satisfiable, the solver will return a core $e \subseteq A$ of literals, of which at least one must be false in any satisfying assignment, i.e., a *reason* why the formula is not satisfiable under assumption A.

The implicit hitting set algorithm always assumes that as many relaxation variables as possible are set to true under the current knowledge (recall that all weights are positive, so setting everything to 1 is the best-case scenario). To that end, the algorithm maintains a hypergraph H with vertex set $V(H) = R$ and the edge set $E(H)$ being the set of extracted cores. Initially, there are no cores (i.e., $E(H) = \emptyset$), and we assume the best-case scenario that all relaxation variables are set to true (i.e., $A = R$). If the SAT solver confirms that the formula is satisfiable under the current assumptions, we found the optimum and terminate. Otherwise, we obtain a core e and add it as an edge to H. Since we want to *maximize* the number of satisfied clauses, we want to *minimize* the number of falsified relaxation variables. Hence, we need the cheapest way to falsify at least one element from each core, i.e., we need a set $X \subseteq V(H)$ with $X \cap e \neq \emptyset$ for all $e \in E(H)$ that minimizes $\sum_{r \in X} w(r)$, whereby $w(r)$ denotes in slight abuse of notation the weight of the soft clause containing r. Such a set X is called a *hitting set* of H. We now assume $A = R \setminus X$ and repeat the process until we found an assumption under which the formula is satisfiable. It is well-known that this procedure solves MAX-SAT exactly if an exact hitting set solver is used [25]. It is also easy to see that we will obtain a heuristic, but *feasible*, solution if we compute heuristic hitting sets – and that is precisely what we will do with an IPU.

3 The I^{2}HS Algorithm

The main ingredient of the I^{2}HS algorithm is an Ising algorithm to compute *hitting sets* of small cardinality. Our QUBO-encoding for the hitting set problem consists of two parts, an *optimization* part ψ_1 and a *penalty* part ψ_2 to enforce the constraint that every edge gets hit. The first part is easy, as the objective of the problem is already in the form of a quadratic unconstrained binary optimization problem, i.e., we introduce for every vertex $v \in V(H)$ an indicator variable x_v with the semantic $x_v = 1 \Leftrightarrow v \in X$. We obtain:

$$\psi_1 := \sum_{v \in V(H)} w(v) \cdot x_v.$$

Encoding the constraints that $X \cap e \neq \emptyset$ for every $e \in E(H)$ is a bit more complicated. We can rephrase the set intersection as a cardinality constraint:

$$X \cap e \neq \emptyset \Longleftrightarrow \sum_{v \in e} x_v \geq 1.$$

Using a *slack variable* s_e with domain $\{0, \ldots, |e| - 1\}$ we can rewrite the right side as $\sum_{v \in e} x_v - s_e = 1$. Following Djidjev, we use $k := \lfloor \log_2(|e| - 1) + 1 \rfloor$ binary variables $s_{e,i}$ to encode s_e. We then rewrite the equality constraint as the following quadratic objective [26]:

$$\Big(\sum_{v \in e} x_v - \sum_{i=0}^{k} 2^i s_{e,i} - 1 \Big)^2.$$

Observe that this expression can evaluate to zero if, and only if, at least one x_v is set to 1. That X is a hitting set can, thus, be encoded using:

$$\psi_2 := \sum_{e \in E(H)} \Big(\sum_{v \in e} x_v - \sum_{i=0}^{k} 2^i s_{e,i} - 1 \Big)^2.$$

To ensure that we optimize the weighted size of the hitting set while satisfying all constraints (i.e., hitting all edges), we define a penalty $\rho := 1 + \sum_{v \in V(H)} |w(v)|$ and collect the insights of this section as:

Theorem 1. *Let H be a hypergraph and ψ_1, ψ_2, ρ be defined as above. Then the ground state of $\psi_{hs} := \psi_1 + \rho\psi_2$ corresponds to a minimum weight hitting set of H.*

3.1 Ensuring Feasible Hitting Sets

Solving the hitting set problem using ψ_{hs} suffers from the same issues discussed in the introduction. Due to **I3**, an IPU may output a set $X \subseteq V(H)$ that does *not* hit all the edges. Fortunately, it is relatively easy to obtain a set $X' \supset X$ that is a feasible hitting set using:

Lemma 1. *Let H be a hypergraph and $X \subseteq V(H)$ be an arbitrary vertex set. Let $\tilde{X}$ be a hitting set of the hypergraph $\tilde{H}$ with $V(\tilde{H}) = V(H) \setminus X$ and $E(\tilde{H}) = \{e \in E(H) \mid X \cap e = \emptyset\}$. Then $X \cup \tilde{X}$ is a hitting set of H.*

Proof. We need to argue that $(X \cup \tilde{X}) \cap e \neq \emptyset$ for all edges $e \in E(H)$. This is trivial if $X \cap e \neq \emptyset$, so assume otherwise. Then $e \in \{e' \in E(H) \mid X \cap e' = \emptyset\} = E(\tilde{H})$ and, since $\tilde{X}$ is an hitting set of $\tilde{H}$, $\tilde{X} \cap e \neq \emptyset$. □

The lemma suggests a recursive algorithm that computes a potential hitting set using an IPU (X in the lemma) and recursively a hitting set of the remaining hypergraph (X'). Figure 1 contains the details.

```
1  INPUT  A hypergraph H = (V(H), E(H)) with weights w: V(H) → ℝ.
2  OUTPUT A hitting set X ⊆ V(H).
3
4  algorithm ISINGHS(H = (V(H), E(H)))
5    if E(H) = ∅ then
6      return ∅
7    X ← hitting set of H using IPU with ψ_hs
8    if X = ∅ then
9      e ← arbitrary element of E(H)
10     v ← arbitrary element of e
11     X ← {v}
12   return X ∪ ISINGHS((V(H) \ X, {e ∈ E(H) | X ∩ e = ∅}))
```

Figure 1: A recursive Ising algorithm that computes a hitting set of H using an IPU and ψ_{hs}.

Theorem 2. *Algorithm* ISINGHS *always outputs a feasible hitting set using at most $O(|V(H)|)$ calls to the* IPU.

Proof. We prove by an induction over $|E(H)|$ that ISINGHS always outputs a feasible solution. The base case is given by $|E(H)| = 0$ and Line 6. For the induction step, we observe that otherwise $X \neq \emptyset$ due to lines 8–11. Then, by Lemma 1 and the induction hypothesis, Line 12 outputs a hitting set of H.

Since X is non-empty in every recursive step, the graph contains no vertex (and thus no edge) after at most $O(|V(H)|$ recursive calls. □

The description of the implicit Ising hitting set algorithm (I^{2}HS) can now be made very short:

> *Compute the hitting sets in the implicit hitting set algorithm using* ISINGHS.

4 Results

We implemented a prototype [27] of I^{2}HS based on PySAT [21] and the Fixstars Amplify Annealing Engine [22], and performed experiments on a MacBook Pro 2016 with an Apple M1 Max processor and 64 GB of RAM running macOS Sonoma 14.5. Domain-specific experiments are postponed to the long version of this article, but we briefly report on benchmarks from the MAX-SAT evaluation [28]. We assembled a subset of the instances on which we never produced a hitting set instance that was too large for the free plan of the Annealing Engine. The annealer was allowed to run for 10 seconds per hitting set instance, which is the maximum possible in the free plan. I^{2}HS found the optimal solution on 31.25% of the instances, and an error of at most ten on 52.82% of the instances. Table 1 presents an excerpt from the results.

Table 1: Some instances of the MAX-SAT evaluation [28]. For each instance, we report the *gap* to the optimal solution (0 if we found it), the time (in seconds) spent preparing the QUBOS, solving the QUBOS, the time used by the SAT solver, and the total number of calls to the IPU.

Gap	*Encoding*	*Annealing*	*SAT*	*#IPU*
CSG_wt-CSG40-40-95 1				
0	0.0021	25.1218	0.0030	2
max-realizability_wt-power-distribution_9_4				
0	0.0044	37.5707	0.0445	3
planning_wt-depot01c				
0	0.0989	379.9619	0.0116	38
preference_planning_wt-WCNF_storage_p02				
0	0.2588	754.5158	0.6816	56
pseudoBoolean_wt-normalized-factor-size=9-P=397-Q=449				
0	0.3756	552.2546	0.0429	45
qcp_wt-file_qc_wcnf_N10_H60_2				
0	0.0342	13.2721	0.0142	1
preference_planning_wt-WCNF_pathways_p11				
3	0.1256	283.5582	0.0559	18
planning_wt-driverlog01bc				
10	0.0108	24.6299	0.0005	1
Security-CriticalCyber-PhysicalComponents_wt-test52–n-15000				
29	0.0179	36.8779	0.0128	3
quantum-circuit-portfoliovqe_4_18_rigetti-agave_8				
117	0.6591	19.5794	0.1129	1

5 Discussion and Outlook

We presented the I^{2}HS algorithm to solve MAX-SAT problems heuristically with the help of an Ising machine. The approach solves various issues associated with IPUS: It is well understood and natural to encode combinatorial problems into the maximum satisfiability problem (**I1**); the user does not need to care about restoring and post-processing, i.e., it is a true general-purpose algorithm (**I4**); and the algorithm is guaranteed to output *feasible solutions* (**I3**).

We provide a prototype and showed in first experiments that the approach can solve many MAX-SAT instances optimal or almost optimal. The experiments also revealed a limitation of the approach (observe the difference between the time used for annealing and ten times the number of IPU calls in Table 1):

L1 The communication time with the IPU.

As with other hardware accelerators (e.g., GPUS), communication bandwidth with the IPU is a bottleneck, which is exacerbated as Ising machines can currently almost solely be accessed via cloud services. We think, however, that this is a temporary issue. Once Ising machines get cheaper and installed directly into the user's device (e.g., Intel Loihi chips [16]), the situation will improve automatically.

A further research direction is a *dynamic scheme* for the annealing time: In our prototype, we always let the annealer search for 10 seconds. However, this may be a waste of time on easy hitting set instances at the beginning of the search and potentially insufficient to find good hitting sets in large hypergraphs that emerge later. Thus, changing the annealing time with the changing hypergraph over time seems promising.

References

1. Izzo, D., Getzner, I., Hennes, D. & Simões, L. F. *Evolving Solutions to TSP Variants for Active Space Debris Removal* in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015* (2015), 1207–1214.
2. Dorrington, S. & Olsen, J. A Location-Routing Problem for the Design of an Asteroid Mining Supply Chain network. *Acta Astronautica* **157,** 350–373. ISSN: 0094-5765 (2019).
3. Guillaume, A. *et al.* Deep Space Network Scheduling Using Quantum Annealing. *IEEE Transactions on Quantum Engineering* **3,** 1–13 (2022).
4. Ho, K. *Modeling and Optimization for Space Logistics Operations: Review of State of the Art* in *AIAA SCITECH 2024 Forum* (2024), 1275.
5. Chien, S., Doyle, R., Davies, A. G., Jonsson, A. & Lorenz, R. The Future of AI in Space. *IEEE Intelligent Systems* **21,** 64–69 (2006).
6. Johnston, M. D. *Spike: AI scheduling for NASA's hubble space telescope* in *Sixth Conference on Artificial Intelligence for Applications* (1990), 184–185.
7. Cesta, A. *et al.* Mexar2: AI Solves Mission Planner Problems. *IEEE Intelligent Systems* **22,** 12–19 (2007).
8. Coffrin, C., Nagarajan, H. & Bent, R. *Evaluating Ising Processing Units with Integer Programming* in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings* (2019), 163–181.
9. Snelling, D. *et al.* Innovation in Planning Space Debris Removal Missions Using Artificial Intelligence and Quantum-Inspired Computing (2021).
10. Carbone, A., De Grossi, F. & Spiller, D. Cutting-Edge Trajectory Optimization through Quantum Annealing. *Applied Sciences* **13,** 12853 (2023).
11. Bhogal, A. S., Sinha, M. & Meshram, P. *NASA Nearest Earth Object Classification Using Quantum Machine Learning: A Survey* in *International Conference on Electrical and Electronics Engineering* (2024), 439–456.
12. Johnston, M. D. Conflict-and Fairness-Directed Heuristic Search Scheduling for NASA's Oversubscribed Deep Space Network (2022).
13. Johnson, M. W. *et al.* Quantum Annealing with Manufactured Spins. *Nature* **473,** 194–198 (2011).
14. Matsubara, S. *et al. Digital Annealer for High-Speed Solving of Combinatorial optimization Problems and Its Applications* in *25th Asia and South Pacific Design Automation Conference, ASP-DAC 2020, Beijing, China, January 13-16, 2020* (2020), 667–672.
15. Kashimata, T., Yamasaki, M., Hidaka, R. & Tatsumura, K. Efficient and Scalable Architecture for Multiple-Chip Implementation of Simulated Bifurcation Machines. *IEEE Access* **12,** 36606–36621 (2024).
16. Davies, M. *et al.* Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook. *Proc. IEEE* **109,** 911–934 (2021).
17. Akopyan, F. *et al.* TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **34,** 1537–1557 (2015).
18. Alom, M. Z., Essen, B. V., Moody, A. T., Widemann, D. P. & Taha, T. M. *Quadratic Unconstrained Binary Optimization (QUBO) on Neuromorphic Computing System* in *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017* (2017), 3922–3929.
19. Ohno, K., Shirai, T. & Togawa, N. Toward Practical Benchmarks of Ising Machines: A Case Study on the Quadratic Knapsack Problem. *CoRR* **abs/2403.19175** (2024).
20. Zhang, T., Tao, Q., Liu, B. & Han, J. *A Review of Simulation Algorithms of Classical Ising Machines for Combinatorial Optimization* in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)* (2022), 1877–1881.
21. Ignatiev, A., Morgado, A. & Marques-Silva, J. *PySAT: A Python Toolkit for Prototyping with SAT Oracles* in *SAT* (2018), 428–437. https://doi.org/10.1007/978-3-319-94144-8_26.
22. *Fixstars Amplify* https://amplify.fixstars.com/.
23. *Handbook of Satisfiability - Second Edition* (eds Biere, A., Heule, M., van Maaren, H. & Walsh, T.) ISBN: 978-1-64368-160-3 (IOS Press, 2021).
24. Chandrasekaran, K., Karp, R. M., Moreno-Centeno, E. & Vempala, S. S. *Algorithms for Implicit Hitting Set Problems* in *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011* (2011), 614–629.
25. Davies, J. *Solving MAXSAT by Decoupling Optimization and Satisfaction* PhD thesis (University of Toronto, Canada, 2014).
26. Djidjev, H. N. Quantum Annealing with Inequality Constraints: The Set Cover Problem. *CoRR* **abs/2302.11185** (2023).
27. Bannach, M. *i2hs* github.com/maxbannach/i2hs.
28. *MaxSAT Evaluation 2023: Solver and Benchmark Descriptions* (eds Berg, J., Järvisalo, M., Martins, R. & Niskanen, A.) (University of Helsinki, 2023).