Research Article

An Approximated QUBO Formulation for Solving Practical SAT Problems

Tomohiro Sonobe[1]

1. National Institute of Informatics, Tokyo, Japan

In this paper, we introduce an approximated quadratic unconstrained binary optimization (QUBO) formulation on satisfiability (SAT) problems derived from real-world applications. The proposed method is an inexact (approximated) formalization of a SAT instance to a QUBO, where the ground state of the QUBO does not correspond to the solution of the SAT instance. The method leverages the fact that practical SAT instances often contain many binary (size 2) clauses, allowing us to directly use a MAX2SAT formalization. In comparative experiments using existing formulations, the proposed method exhibits superior performance on the pigeonhole principle, graph coloring problems, and a subset of instances from the SAT Competition 2023. Notably, we were able to find the exact solutions of PHP instances using our method, despite its approximate formalization.

Correspondence: papers@team.qeios.com — Qeios will forward to the authors

1. Introduction

Due to recent advancements, many Ising model solvers (Ising solvers) have been developed[1][2][3][4] and are used to address combinatorial optimization problems[5][6]. The ground state search of an Ising model can be converted into Quadratic Unconstrained Binary Optimization (QUBO). QUBO is a combinatorial optimization problem consisting of n-dimensional binary variable x $(\in \{0,1\}^n)$ and a $n \times n$ square matrix Q. The goal is to find the variable assignment that minimizes the following Hamiltonian $H(x)$:

$$H(x) = \sum_i \sum_j Q_{i,j} x_i x_j \tag{1}$$

Satisfiability problems (SAT) are the most basic NP-complete problems[7]. Given a formula, SAT is to determine whether the given formula has a satisfying variable assignment or not. Many problems, such as theorem proving[8], neural network verification[9], and circuit verification[10], are translated into SAT

 doi.org/10.32388/JD6K4G.2

problems and solved efficiently by SAT solvers[11]. In general, SAT formulas are given in Conjunctive Normal Form (CNF), where Boolean variables appear in clauses connected by logical-OR, and all clauses are connected by logical-AND. More formally, a SAT instance $\phi = (V, C)$ comprises a set of Boolean variables $V = \{x_1, x_2, \ldots, x_n\}$ and a set of clauses C. A literal is a positive or negative form of a variable. A clause $c_i \in C$ of size k is a disjunction (logical-OR) of k literals, expressed as $c_i = (c_{i,1} \vee c_{i,2} \vee \ldots c_{i,k})$ where $c_{i,j}$ represents a literal of a variable. A SAT instance with fixed clause size k is called kSAT. As a variant of kSAT, MAXkSAT is an optimization problem where the objective is to find a variable assignment that maximizes the number of satisfied clauses. Not-All-Equal kSAT (NAEkSAT) is also a variation of kSAT with a different satisfaction criterion: a clause is satisfied only when at least one literal is true and at least one is false. Consequently, any solution to NAEkSAT is also a valid solution to kSAT on the same formula. Existing research[12][13][14][15] focuses on solving 3SAT, MAX2SAT, and NAE3SAT using the Ising model and QUBO formulations. Especially, our preliminary experiments demonstrated that solving QUBO instances derived from practical MAX2SAT instances[16] using simulated annealing exhibited high performance.

In this paper, we introduce a simple formalization method to convert (general) SAT instances into QUBO instances. Unlike existing QUBO formalization methods for SAT instances[12][13][14], our proposed method is designed for practical SAT instances containing a large number of binary clauses. As demonstrated in our experiments, practical SAT instances frequently exhibit a high proportion of binary clauses due to the prevalence of relationships between pairs of variables, often represented by at-most-one constraints[17]. Our approach employs MAX2SAT formalization for binary clauses and NAE3SAT formalization for ternary clauses, resulting in an approximate QUBO. The QUBO formulations of MAX2SAT and NAE3SAT require no additional variables, meaning that a Boolean variable in a MAX2SAT/NAE3SAT instance directly corresponds to a QUBO variable. In contrast, existing QUBO formulations for 3SAT instances often necessitate the conversion of binary clauses into ternary clauses by introducing auxiliary variables for general (non-3SAT) SAT instances. Our method directly leverages the binary clauses through MAX2SAT formalization. For ternary clauses, the method employs NAE3SAT formalization, resulting in an approximate QUBO where the ground state does not directly correspond to a solution that satisfies all three literals in a specific ternary clause. Note that clauses containing more than three literals are decomposed into ternary clauses by introducing additional variables. Additionally, we assume that SAT instances are free of unit clauses through pre-processing (unit propagation[18]). Our experimental results demonstrate the effectiveness of the proposed method on practical instances, including the pigeonhole principle, graph coloring, and a subset of instances from the SAT Competition 2023.

2. Proposed Method

Existing formalization methods[12][13] are limited to CNFs with only ternary clauses. To accommodate clauses of different sizes, auxiliary variables must be introduced, potentially increasing the size of the QUBO. Our proposed method leverages the prevalence of binary clauses in real-world SAT instances, often arising from at-most-one constraints[17]. By directly translating these binary clauses to a QUBO via MAX2SAT formalization, we aim to mitigate this issue and reduce the overall problem size. Note that our formalization method generates an approximated QUBO where the ground state may not correspond to the solution of the SAT instance, unlike exact formalization methods such as Chancellor[12] and Nüßlein[13].

We define a given SAT instance as $\phi = (V, C = C_2 \cup C_3)$ where $V = \{x_1, x_2, \ldots, x_n\}$ is a set of Boolean variables, C_2 is a set of binary clauses, and C_3 is a set of ternary clauses. We assume that the given instance has been preprocessed by applying unit propagation[18] to eliminate unit clauses and introducing auxiliary variables to convert longer clauses (greater than size 3) into ternary clauses. For the construction of the QUBO, we adopt the same notation for QUBO variables as for Boolean variables, i.e., a Boolean variable x_i directly corresponds to a QUBO variable x_i.

First, for binary clauses C_2, we simply apply MAX2SAT formalization[19] to them. The binary clauses are classified into the following three types, and corresponding QUBO formalizations are shown.

- $(x_i \vee x_j) : 1 - x_i - x_j + x_i x_j$
- $(\neg x_i \vee x_j) : x_i - x_i x_j$
- $(\neg x_i \vee \neg x_j) : x_i x_j$

The objective QUBO Hamiltonian can be constructed by summing up all the formulas.

Second, for ternary clauses C_3, our method simply uses NAE3SAT formalization as follows:

$$H = \sum_{(x_1 \vee x_2 \vee x_3) \in C^3} \zeta_{x_1}\zeta_{x_2}(2x_1 - 1)(2x_2 - 1) + \zeta_{x_1}\zeta_{x_3}(2x_1 - 1)(2x_3 - 1) + \zeta_{x_2}\zeta_{x_3}(2x_2 - 1)(2x_3 - 1) \quad (2)$$

doi.org/10.32388/JD6K4G.2

where ζ_x is the sign function for a literal x ($+1$ for a positive literal and -1 for a negative literal). Since a clause in which all its literals have truth assignments is not allowed in NAE3SAT, such a solution of the SAT cannot be found. However, solutions of NAE3SAT are valid for SAT and used in practice[15].

The whole process is illustrated in Figure 1. As a concrete example, consider a (general) SAT instance $\phi_1 = (V_1 = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\},\ C_1 = \{(x_1), (\neg x_1 \lor x_2 \lor x_3), (x_3 \lor x_4 \lor x_5 \lor x_6 \lor x_7)\})$. By unit propagation, the unit clause (x_1) is forced to be true (assigning x_1 as true) and removed from the clause set. Consequently, the clause $(\neg x_1 \lor x_2 \lor x_3)$ is shortened to $(x_2 \lor x_3)$ since the literal $\neg x_1$ is false. Subsequently, the clause $(x_3 \lor x_4 \lor x_5 \lor x_6 \lor x_7)$ is divided into three ternary clauses by introducing auxiliary variables: $(x_3 \lor x_4 \lor y_1)$, $(\neg y_1 \lor x_5 \lor y_2)$, and $(\neg y_2 \lor x_6 \lor x_7)$. At this point, Our method is applied to the re-formalized instance $\phi_2 = (V_2 = \{x_2, x_3, x_4, x_5, x_6, x_7, y_1, y_2\},$ $C_2 = \{(x_2 \lor x_3), (x_3 \lor x_4 \lor y_1), (\neg y_1 \lor x_5 \lor y_2), (\neg y_2 \lor x_6 \lor x_7)\})$. For existing methods applied to 3SAT instances, the binary clauses must be transformed. The binary clause $(x_2 \lor x_3)$ is converted to $(x_2 \lor x_3 \lor z_1)$ by introducing another padding variables $\{z_1, z_2, z_3\}$. To ensure all the three variables to be false, seven ternary clauses must be added: $C_p = \{(\neg z_1 \lor z_2 \lor z_3), (z_1 \lor \neg z_2 \lor z_3), (z_1 \lor z_2 \lor \neg z_3), (\neg z_1 \lor \neg z_2 \lor z_3), (\neg z_1 \lor z_2 \lor \neg z_3), (z_1 \lor \neg z_2 \lor \neg z_3), (\neg z_1 \lor \neg z_2 \lor \neg z_3)\}$. Note that these three padding variables can be reused for other binary clauses, i.e., we only need to add z_1 to all the binary clauses (no additional padding variables are required). The final 3SAT instance $\phi_3 = (V_3 = V_2 \cup \{z_1, z_2, z_3\},$ $C_3 = \{(x_3 \lor x_4 \lor y_1), (\neg y_1 \lor x_5 \lor y_2), (\neg y_2 \lor x_6 \lor x_7), (x_2 \lor x_3 \lor z_1)\} \cup C_p)$ is given to the existing methods.

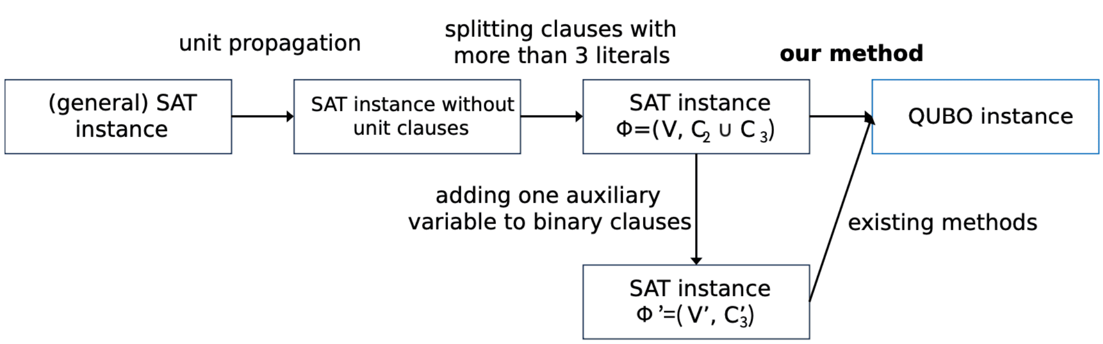

Figure 1. Data flow of the proposed/existing methods.

doi.org/10.32388/JD6K4G.2

3. Experiments

We conducted comparative experiments on practical SAT instances using a machine equipped with an Intel Core i9-10980X CPU, 256GB RAM, and Ubuntu 20.04. Our code[1] was implemented in Python 3.10, and we utilized PyQUBO[20][21] for QUBO construction. For ground state search of a QUBO, we used the "SimulatedAnnealingSampler" (SAS) from the dwave-neal package[2] version 0.6.0. We invoked the "sample_qubo" function of the SAS with the following parameters: "num_reads" $= 10$, "num_sweeps" $= 10000$, and "seed" $= 1$."

As a comparison, we used existing methods: Chancellor[12], Nüßlein[13], and FullApprox[14]. Chancellor and Nüßlein are exact formalization methods that generate a QUBO with $n + m$ variables from a 3SAT instance with n Boolean variables and m clauses. FullApprox is an approximated formalization method that constructs a QUBO with the same number of variables as the given 3SAT instance. Additionally, we employed NAE3SAT formalization as a naive baseline.

As benchmark instances, we used three types of SAT instances: the pigeonhole principle (PHP), graph coloring (GC), and a subset of instances from the SAT Competition 2023 (COMP23). PHP instances were generated with CNFgen[22] by setting n pigeons and n holes ($n = \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$); hence, all 10 instances are satisfiable. GC instances were also generated with CNFGen, from the instances of the DIMACS graph coloring challenge[3]. We selected instances whose chromatic number was known and encoded each SAT instance with its chromatic number, thus resulting in a satisfiable instance. The subset of the SAT Competition 2023 instances[4] was created by selecting instances with less than 100 KB and at least one binary clause, resulting in 40 instances From these, we excluded instances with non-sequential variable indices, leaving 28 instances. It includes 1 satisfiable instance and 25 unsatisfiable instances (and 2 solution-unknown instances). For each instance and formalization method, we measured the number of unsatisfied clauses in the solution output by the SAS. An exact solution for a satisfiable instance will exhibit 0 unsatisfied clauses.

PHP states whether it is possible to assign each pigeon to exactly one hole such that no two pigeons are assigned to the same hole. A PHP instance is formalized in SAT as follows. Given an integer n indicating the number of pigeons and holes (note that we consider an equal number of pigeons and holes in this paper), a Boolean variable $x_{i,j} (1 \leq i, j \leq n)$ is true if i-th pigeon enters j-th hole and false otherwise. The following two constraints are encoded as a set of clauses.

 doi.org/10.32388/JD6K4G.2

- each pigeon must enter at least one hole (at-leas-one constraint): $\bigvee_{j=1}^{n} x_{i,j}$ for $1 \leq i \leq n$.
- there is at most one pigeon for each hole (at-most-one constraint): $\neg x_{i,k} \vee \neg x_{j,k}$ for $1 \leq i < j \leq n$ and for $1 \leq k \leq n$.

GC determines the vertices of a graph can be colored with at most r different colors such that no two adjacent vertices have the same color. GC instances are similar to PHP ones. Given a graph $G = (V, E)$, where V is a set of vertices and E is a set of edges, and an integer r, a Boolean variable $x_{i,j}(1 \leq i \leq |V|, 1 \leq j \leq r)$ is true if vertex i is colored j. The following three constraints are encoded as a set of clauses.

- each vertex must be colored at least one color (at-leas-one constraint): $\bigvee_{j=1}^{r} x_{i,j}$ for $1 \leq i \leq |V|$.
- each vertex must be colored at most one color (at-most-one constraint): $\neg x_{i,j} \vee \neg x_{i,k}$ for $1 \leq i \leq |V|$ and for $1 \leq j < k \leq r$.
- adjacent vertices must have different colors: $\neg x_{u,k} \vee \neg x_{v,k}$ for $(u, v) \in E$ and for $1 \leq k \leq r$

Hence, at-most-one constraints generate binary clauses in proportional to n^2 in PHP and r^2 in GC. Consequently, the percentage of binary clauses in both PHP and GC instances can be high. Additionally, we emphasize that our method can find an exact solution for satisfiable PHP/GC instances. This is because for non-binary clauses (i.e., the at-least-one constraint), there always exists a solution where only one literal is true and the others are false. For example, in a PHP instance, we can easily find an exact solution where $x_{i,i} =$ true for $1 \leq i \leq n$. In this solution, each clause of the at-least-one constraint is satisfied by a single true literal (with the others being false). The same holds true for GC instances: only one literal is true in the clauses of the at-least-one constraint for a valid solution. These solutions can be found through the NAE3SAT formalization.

n	vars	clauses	b-clauses	b-ratio	N3M2	Chancellor	Nüßlein	FullApprox	NAE3SAT
5	25	55	50	90.91%	0	19	0	30	34
10	100	460	450	97.83%	0	121	6	360	3
15	225	1590	1575	99.06%	0	328	13	0	1379
20	400	3820	3800	99.48%	0	629	20	0	3420
25	625	7525	7500	99.67%	0	1255	25	6900	3
30	900	13080	13050	99.77%	0	1555	4116	0	12180
35	1225	20860	20825	99.83%	0	2391	35	19635	19635
40	1600	31240	31200	99.87%	0	3465	40	0	29640
45	2025	44595	44550	99.90%	0	4156	15618	42570	2
50	2500	61300	61250	99.92%	0	5475	50	0	3
total		184525			0	19394	19923	69495	66299
unsat-ratio		-			0.00%	10.51%	10.80%	37.66%	35.93%

Table 1. Results for PHP instances. The column “n” indicates the number of pigeons and holes, “vars” and “clauses” represent the number of variables and clauses in the original CNF (before adding auxiliary variables), “b-clauses” is the number of binary clauses, “b-ratio” is the ratio of binary clauses, and the remaining columns show the number of unsatisfied clauses (lower is better) for each method. The bottom two rows indicate the total number of unsatisfied clauses for all the instances and the ratio for the total number of clauses.

Table 1 presents the results of PHP. PHP instances contain a significant number of binary clauses, particularly for larger values of “n”. The proposed method (named N3M2) successfully found exact solutions for all PHP instances. FullApprox could find exact solutions for half of the instances, but its solutions often failed to satisfy the clauses for certain instances (e.g., instances with $n = \{25, 35, 45\}$) due to its approximate nature. While the NAE3SAT method could also find near-exact solutions for some instances, N3M2 leveraged the benefits of MAX2SAT formalization to achieve superior performance. In

doi.org/10.32388/JD6K4G.2

terms of total performance, the Chancellor and Nüßlein methods achieved approximately 10%, better than both the FullApprox and NAE3SAT methods (over 35%). As for the processing time of SAS, N3M2 generated smaller QUBOs, leading to faster search times (as shown in Table 4). We also show the processing time of CaDiCaL[23], one of the state-of-the-art SAT Solvers. CaDiCaL could solve all the instances within 0.1 seconds. As expected, PHP instances with an equal number of pigeons and holes, considered in this paper, proved to be easily solvable.

instance	vars	clauses	b-clauses	b-ratio	N3M2	Chancellor	Nüßlein	FullApprox	NAE3SAT
anna.col_k11	1518	13151	13013	98.95%	55	2267	134	10728	10631
david.col_k11	957	9338	9251	99.07%	41	1541	85	7569	68
fpsol2.i.1.col_k65	32240	1789686	1789190	99.97%	238	111914	496	1736419	244
fpsol2.i.2.col_k30	13530	457366	456915	99.90%	352	36485	134270	19	387
fpsol2.i.3.col_k30	12750	445940	445515	99.90%	338	34109	425	20	375
games120.col_k9	1080	10182	10062	98.82%	81	2012	120	0	7922
huck.col_k11	814	7455	7381	99.01%	32	1339	72	0	6084
inithx.i.1.col_k54	46656	2247426	2246562	99.96%	416	149564	789756	22	2164646
inithx.i.2.col_k31	19995	733919	733274	99.91%	563	55572	201812	697495	622
inithx.i.3.col_k31	19251	722425	721804	99.91%	585	54416	197918	688212	639
jean.col_k10	800	6220	6140	98.71%	38	1398	74	0	69
le450_15b.col_k15	6750	170235	169785	99.74%	538	14824	450	18	148945
le450_15c.col_k15	6750	297900	297450	99.85%	959	19176	450	267178	265935
le450_15d.col_k15	6750	298950	298500	99.85%	1032	19324	450	268378	816
le450_25a.col_k25	11250	341950	341500	99.87%	305	27026	450	314665	396
le450_25b.col_k25	11250	342025	341575	99.87%	334	27257	450	3	426
le450_25c.col_k25	11250	569025	568575	99.92%	672	35850	450	29	526748
le450_25d.col_k25	11250	571075	570625	99.92%	678	34872	450	529449	528336
le450_5a.col_k5	2250	33520	33070	98.66%	591	4377	449	0	20401
le450_5b.col_k5	2250	33620	33170	98.66%	530	4472	450	20027	20764
le450_5c.col_k5	2250	53965	53515	99.17%	296	3526	450	1	659
le450_5d.col_k5	2250	53735	53285	99.16%	33	3484	450	2	507
miles1000.col_k42	5376	245408	245280	99.95%	63	15674	128	0	233802
miles1500.col_k73	9344	715966	715838	99.98%	46	37810	128	696226	696220
miles250.col_k8	1024	6808	6680	98.12%	59	1513	110	5010	5101

doi.org/10.32388/JD6K4G.2

instance	vars	clauses	b-clauses	b-ratio	N3M2	Chancellor	Nüßlein	FullApprox	NAE3SAT
miles500.col_k20	2560	47848	47720	99.73%	64	5312	128	42948	43045
miles750.col_k31	3968	125151	125023	99.90%	62	9939	128	1	117005
mulsol.i.1.col_k49	9653	424194	423997	99.95%	91	27530	197	406691	406840
mulsol.i.2.col_k31	5828	208043	207855	99.91%	168	14207	74639	195499	195059
mulsol.i.3.col_k31	5704	207140	206956	99.91%	196	14566	184	8	194113
mulsol.i.4.col_k31	5735	208536	208351	99.91%	143	14334	185	8	195505
mulsol.i.5.col_k31	5766	209839	209653	99.91%	144	14446	75499	8	196505
myciel3.col_k4	44	157	146	92.99%	9	45	3	73	12
myciel4.col_k5	115	608	585	96.22%	13	135	17	351	21
myciel5.col_k6	282	2168	2121	97.83%	38	407	39	1414	58
myciel6.col_k7	665	7375	7280	98.71%	120	1114	2256	5200	153
myciel7.col_k8	1528	24419	24228	99.22%	314	2367	191	3	18514
queen11_11.col_k11	1331	28556	28435	99.58%	34	2659	121	9	66
queen13_13.col_k13	2197	56615	56446	99.70%	46	4853	169	48402	69
queen5_5.col_k5	125	1075	1050	97.67%	5	200	24	630	659
queen6_6.col_k7	252	2822	2786	98.72%	9	455	36	1990	27
queen7_7.col_k7	343	4410	4361	98.89%	12	657	49	3115	51
queen8_12.col_k12	1152	22848	22752	99.58%	13	2198	96	18960	42
queen8_8.col_k9	576	8920	8856	99.28%	12	1156	64	6921	6940
queen9_9.col_k10	810	14286	14205	99.43%	11	1639	81	11397	47
zeroin.i.1.col_k49	10339	449247	449036	99.95%	80	29747	211	430989	127
zeroin.i.2.col_k30	6330	198226	198015	99.89%	109	15226	211	7	181
zeroin.i.3.col_k30	6180	196016	195810	99.89%	117	14463	206	7	183181
total		12625789			10685	877457	1485211	6416101	6198963
unsat-ratio		-			0.08%	6.95%	11.76%	50.82%	49.10%

doi.org/10.32388/JD6K4G.2

Table 2. Results for GC instances.

Table 2 presents the results of GC. All the instances were generated by CNFGen for each graph and chromatic number; hence, all of them are satisfiable. Similar to PHP, the majority of clauses of GC instances are binary clauses. While N3M2 did not find exact solutions, it performed well on all instances, leaving only 0.08% of clauses unsatisfied. FullApprox found exact solutions for 5 instances but satisfied only half of the total clauses, comparable to NAE3SAT performance. Contrary to PHP, N3M2 failed to reach the ground state of any instances. One possible reason for this is that GC has one more constraint than PHP, namely, the adjacency constraint. This constraint increases the difficulty of solving GC. Additionally, approximately half of the GC instances are larger than the largest PHP instance ($n = 50$). In fact, the average processing time of CaDiCaL for GC instances was longer than that for PHP instances.

doi.org/10.32388/JD6K4G.2

ID	sol	vars	clauses	b-clauses	b-ratio	N3M2	Chancellor	Nüßlein	FullApprox	NAE3SAT
1	unsat	354	2938	2922	99.46%	32	458	619	1	25
2	unsat	270	6663	1362	20.44%	649	89	14	1116	312
3	unsat	351	4291	4264	99.37%	23	650	1256	9	22
4	unsat	176	1149	1133	98.61%	11	282	13	933	11
5	unsat	398	741	5	0.67%	51	8	3	1	57
6	sat	260	624	520	83.33%	104	216	171	306	137
7	unsat	228	2023	2004	99.06%	12	382	19	1709	14
8	unsat	247	2073	2054	99.08%	17	380	575	5	16
9	unsat	90	415	405	97.59%	85	117	94	324	109
10	unsat	132	1527	168	11.00%	147	31	22	146	162
11	unsat	187	1348	1331	98.74%	22	269	402	4	14
12	unsat	348	4817	4788	99.40%	35	679	29	15	4225
13	unsat	518	7611	7574	99.51%	54	1091	36	6580	6503
14	unknown	526	5477	5458	99.65%	44	662	20	1	38
15	unsat	288	3240	3216	99.26%	33	602	1020	2790	2863
16	unknown	460	4494	4476	99.60%	33	594	1036	3707	34
17	unsat	205	4549	975	21.43%	575	151	12	828	552
18	unsat	228	1915	1896	99.01%	12	308	557	1665	1625
19	unsat	190	632	2	0.32%	85	22	7	63	79
20	unsat	460	8655	8637	99.79%	48	646	1881	1	48
21	unsat	448	5562	5530	99.42%	34	806	31	8	4860
22	unsat	392	4116	4088	99.32%	31	651	25	6	32
23	unsat	231	1880	1859	98.88%	16	443	559	4	19
24	unsat	462	6403	6370	99.48%	53	864	31	5633	5486
25	unsat	412	3654	3637	99.53%	38	572	17	1	2987

ID	sol	vars	clauses	b-clauses	b-ratio	N3M2	Chancellor	Nüßlein	FullApprox	NAE3SAT
26	unsat	252	2049	2028	98.98%	17	427	617	3	21
27	unsat	192	1252	1236	98.72%	8	373	363	1028	1021
28	unsat	299	3182	3159	99.28%	26	564	23	8	2804
total			93280			2295	12337	9452	26895	34076
unsat-ratio			-			2.46%	13.23%	10.13%	28.83%	36.53%

Table 3. Results for COMP23 instances. The column "sol" indicates solutions (satisfiable/unsatisfiable or unknown). Due to space constraints, instance names are abbreviated (see Appendix for full names).

Table 3 presents the results of COMP23 instances. Due to space constraints, instance names are abbreviated (see Appendix for full names). N3M2 showed stable performance for all the instances, leaving only 2.46% of total clauses unsatisfied. Some instances contain fewer binary clauses, and N3M2 was ineffective for them. FullApprox could output near-optimal solutions, only 1 unsatisfied clause, for 5 instances, and less than 10 unsatisfiable clauses for 13 instances. Note that, as shown in the Table 6, the large part of instances could not solved by CaDiCaL within 10000 seconds. Compared to PHP and GC, COMP23 instances are more difficult to solve.

4. Summary

We introduce a simple, approximate formalization method to transform SAT instances into QUBOs, leveraging NAE3SAT and MAX2SAT formalization. This approach benefits from the high prevalence of binary clauses in practical SAT instances. Our experimental results demonstrate the superior performance of the proposed method on PHP, GC, and small instances from the SAT Competition 2023. Notably, the method successfully finds all exact solutions for PHP instances. Future work will focus on the theoretical analysis of our method.

Appendix A. Processing time and instance names of the SAT Competition 2023

Tables 4, 5, and 6 show processing times of each method for PHP, GC, and COMP23, respectively. We also report the processing time of CaDiCaL[23] version 2.1.2, one of the state-of-the-art SAT solvers, by setting the time limit as 10000 seconds. Table 7 presents the correspondence between IDs and instance names of COMP23 instances.

n	vars	clauses	b-ratio	N3M2	Chancellor	Nüßlein	FullApprox	NAE3SAT	CaDiCaL
5	25	55	90.91%	0.05	0.17	0.14	0.06	0.06	0.00
10	100	460	97.83%	0.25	1.02	0.78	0.30	0.29	0.00
15	225	1590	99.06%	0.61	3.22	2.37	0.74	0.76	0.00
20	400	3820	99.48%	1.17	7.42	5.36	1.44	1.84	0.00
25	625	7525	99.67%	1.93	14.93	11.25	2.46	2.91	0.00
30	900	13080	99.77%	3.24	24.36	19.53	3.83	3.87	0.01
35	1225	20860	99.83%	4.73	42.71	32.77	6.74	7.67	0.01
40	1600	31240	99.87%	7.05	66.08	53.36	9.95	10.19	0.02
45	2025	44595	99.90%	9.14	104.79	81.64	12.85	13.59	0.02
50	2500	61300	99.92%	11.22	146.39	113.20	16.47	17.43	0.03

Table 4. Processing time for PHP instances. Each value of each method stands for time (seconds) of SimulatedAnnealingSampler. The right-most column shows the processing time of CaDiCaL.

doi.org/10.32388/JD6K4G.2

instance	vars	clauses	b-ratio	N3M2	Chancellor	Nüßlein	FullApprox	NAE3SAT	CaDiCaL
anna.col_k11	1518	13151	98.95%	5.25	28.63	21.54	6.00	6.40	0.01
david.col_k11	957	9338	99.07%	3.36	19.20	14.40	3.54	3.79	0.01
fpsol2.i.1.col_k65	32240	1789686	99.97%	800.00	6326.45	5906.63	859.76	1352.33	0.39
fpsol2.i.2.col_k30	13530	457366	99.90%	267.44	1707.06	1594.83	236.42	400.95	0.12
fpsol2.i.3.col_k30	12750	445940	99.90%	248.48	1647.17	1510.63	222.63	373.20	0.12
games120.col_k9	1080	10182	98.82%	3.32	21.51	16.26	3.84	4.42	0.00
huck.col_k11	814	7455	99.01%	2.52	15.46	11.81	2.96	3.11	0.00
inithx.i.1.col_k54	46656	2247426	99.96%	1141.75	7715.49	6961.00	1009.75	1800.78	0.51
inithx.i.2.col_k31	19995	733919	99.91%	477.43	2792.77	2725.76	427.45	702.44	0.20
inithx.i.3.col_k31	19251	722425	99.91%	459.95	2746.58	2657.01	416.78	679.69	0.18
jean.col_k10	800	6220	98.71%	2.73	13.34	10.03	2.79	2.82	0.00
le450_15b.col_k15	6750	170235	99.74%	43.48	610.75	555.32	56.60	56.97	0.27
le450_15c.col_k15	6750	297900	99.85%	79.73	1093.39	979.41	138.50	116.74	10000.00
le450_15d.col_k15	6750	298950	99.85%	76.78	1081.64	937.43	135.35	116.50	10000.00
le450_25a.col_k25	11250	341950	99.87%	172.54	1338.32	1220.98	202.06	298.00	0.12
le450_25b.col_k25	11250	342025	99.87%	163.06	1353.05	1221.12	203.31	294.87	0.10
le450_25c.col_k25	11250	569025	99.92%	222.09	2184.04	1975.19	313.90	408.84	10000.00
le450_25d.col_k25	11250	571075	99.92%	212.97	2189.96	1981.19	309.72	405.79	10000.00
le450_5a.col_k5	2250	33520	98.66%	7.12	65.88	53.78	10.65	11.34	0.04
le450_5b.col_k5	2250	33620	98.66%	7.15	66.32	53.88	10.67	11.37	0.15
le450_5c.col_k5	2250	53965	99.17%	8.56	115.16	91.64	13.26	14.19	0.03
le450_5d.col_k5	2250	53735	99.16%	8.63	110.56	92.01	13.69	14.57	0.03
miles1000.col_k42	5376	245408	99.95%	50.29	818.42	724.57	67.66	65.19	0.24
miles1500.col_k73	9344	715966	99.98%	186.89	2408.22	2219.19	288.37	403.13	0.91
miles250.col_k8	1024	6808	98.12%	2.67	15.10	11.65	3.33	3.40	0.00

instance	vars	clauses	b-ratio	N3M2	Chancellor	Nüßlein	FullApprox	NAE3SAT	CaDiCaL
miles500.col_k20	2560	47848	99.73%	11.38	96.92	79.88	15.10	16.83	0.03
miles750.col_k31	3968	125151	99.90%	23.71	364.56	334.57	31.97	36.22	0.19
mulsol.i.1.col_k49	9653	424194	99.95%	151.47	1519.02	1402.65	189.45	290.78	0.12
mulsol.i.2.col_k31	5828	208043	99.91%	55.09	664.32	649.67	62.39	62.38	0.06
mulsol.i.3.col_k31	5704	207140	99.91%	61.84	675.76	650.16	62.04	89.33	0.06
mulsol.i.4.col_k31	5735	208536	99.91%	74.08	703.15	673.31	58.63	115.76	0.06
mulsol.i.5.col_k31	5766	209839	99.91%	56.84	698.48	674.06	65.20	87.70	0.06
myciel3.col_k4	44	157	92.99%	0.08	0.33	0.28	0.10	0.09	0.00
myciel4.col_k5	115	608	96.22%	0.25	1.20	0.91	0.30	0.30	0.00
myciel5.col_k6	282	2168	97.83%	0.70	4.01	2.92	0.86	0.87	0.00
myciel6.col_k7	665	7375	98.71%	1.94	13.89	10.52	2.39	2.42	0.00
myciel7.col_k8	1528	24419	99.22%	5.50	46.04	34.07	7.28	8.33	0.01
queen11_11.col_k11	1331	28556	99.58%	5.17	53.22	37.73	7.34	8.52	10000.00
queen13_13.col_k13	2197	56615	99.70%	10.65	119.51	92.78	13.41	17.29	10000.00
queen5_5.col_k5	125	1075	97.67%	0.28	1.91	1.35	0.36	0.36	0.00
queen6_6.col_k7	252	2822	98.72%	0.67	4.91	3.57	0.88	0.88	0.01
queen7_7.col_k7	343	4410	98.89%	0.93	7.62	5.34	1.23	1.24	0.00
queen8_12.col_k12	1152	22848	99.58%	4.25	42.56	32.08	6.04	6.65	0.04
queen8_8.col_k9	576	8920	99.28%	1.86	16.12	11.93	2.35	3.11	0.15
queen9_9.col_k10	810	14286	99.43%	2.76	26.48	20.10	3.67	4.15	0.97
zeroin.i.1.col_k49	10339	449247	99.95%	106.30	1448.36	1265.31	167.54	218.66	0.11
zeroin.i.2.col_k30	6330	198226	99.89%	46.40	620.13	565.62	68.84	100.92	0.05
zeroin.i.3.col_k30	6180	196016	99.89%	46.96	579.77	505.11	57.02	67.67	0.06

Table 5. Processing time for GC instances.

doi.org/10.32388/JD6K4G.2

ID	sol	vars	clauses	b-ratio	N3M2	Chancellor	Nüßlein	FullApprox	NAE3SAT	CaDiCaL
1	unsat	354	2938	99.46%	0.94	5.82	4.42	1.13	1.11	10000.00
2	unsat	270	6663	20.44%	0.96	11.16	8.00	1.35	1.44	10000.00
3	unsat	351	4291	99.37%	1.04	8.01	5.81	1.32	1.50	10000.00
4	unsat	176	1149	98.61%	0.47	2.32	1.79	0.57	0.57	132.77
5	unsat	398	741	0.67%	1.63	3.59	3.33	1.59	1.80	0.02
6	sat	260	624	83.33%	0.66	1.99	1.78	0.76	0.80	10000.00
7	unsat	228	2023	99.06%	0.63	3.98	2.71	0.77	0.78	7574.74
8	unsat	247	2073	99.08%	0.69	4.24	3.05	0.85	0.85	10000.00
9	unsat	90	415	97.59%	0.22	0.94	0.73	0.26	0.26	4.14
10	unsat	132	1527	11.00%	0.36	2.62	2.06	0.45	0.45	4.35
11	unsat	187	1348	98.74%	0.50	2.72	2.06	0.61	0.61	715.66
12	unsat	348	4817	99.40%	1.04	8.87	6.32	1.35	1.44	10000.00
13	unsat	518	7611	99.51%	1.81	14.36	10.28	2.22	2.25	10000.00
14	unknown	526	5477	99.65%	1.44	11.03	7.65	1.87	1.82	10000.00
15	unsat	288	3240	99.26%	0.83	6.22	4.32	1.05	1.06	10000.00
16	unknown	460	4494	99.60%	1.26	8.92	6.32	1.48	1.58	10000.00
17	unsat	205	4549	21.43%	0.72	7.81	6.20	0.96	0.96	3154.73
18	unsat	228	1915	99.01%	0.64	3.77	2.80	0.77	0.79	10000.00
19	unsat	190	632	0.32%	0.27	1.07	1.09	0.25	0.27	10000.00
20	unsat	460	8655	99.79%	1.45	15.81	11.27	1.92	1.93	420.15
21	unsat	448	5562	99.42%	1.40	10.66	8.08	1.85	1.86	10000.00
22	unsat	392	4116	99.32%	1.19	8.13	6.01	1.46	1.59	10000.00
23	unsat	231	1880	98.88%	0.63	3.68	2.86	0.77	0.80	152.61
24	unsat	462	6403	99.48%	1.47	11.95	8.51	1.95	2.03	10000.00
25	unsat	412	3654	99.53%	1.09	7.40	5.32	1.30	1.28	10000.00

ID	sol	vars	clauses	b-ratio	N3M2	Chancellor	Nüßlein	FullApprox	NAE3SAT	CaDiCaL
26	unsat	252	2049	98.98%	0.71	4.06	2.90	0.86	0.87	348.18
27	unsat	192	1252	98.72%	0.53	2.60	1.98	0.63	0.65	329.45
28	unsat	299	3182	99.28%	0.87	6.00	4.01	1.07	1.10	10000.00

Table 6. Processing time for COMP23 instances.

ID	instance name
1	03e9d1abe418a1727bbf2ead77d69d02-php15-mixed-15percent-blocked.cnf
2	0a4ed112f2cdc0a524976a15d1821097-cliquecoloring_n12_k9_c8.cnf
3	11db226d109e82f93aaa3b2604173ff9-posixpath__joinrealpath_13.cnf
4	246afd75cb97a21144f368c00252a656-BZ2File_write_11.cnf
5	27b4fe4cb0b4e2fd8327209ca5ff352c-grid_10_20.shuffled.cnf
6	328da7966b09b2f6e99c93c4e877fbff-sgen3-n260-s62321009-sat.cnf
7	37d40a1092b58ad28285b9453872d211-DecompressReader_read_12.cnf
8	41a8365f60db55b71d949df6954e0db7-FileObject_open_13.cnf
9	44092fcc83a5cba81419e82cfd18602c-php-010-009.shuffled-as.sat05-1185.cnf
10	571a2f223784fb92a53b4cc8cc8b569e-clqcolor-08-06-07.shuffled-as.sat05-1257.cnf
11	72b5ad031bf852634bc081f9da9a5a60-GzipFile_close_11.cnf
12	7aaf3275cbe217044ef305f0a1ca8eb5-CNFPlus_from_fp_12.cnf
13	824c21545e228872744675ae4ee32976-WCNFPlus_to_alien_14.cnf
14	964162c1faee2c1e3a4dfa4f9c75c34f-php18-mixed-15percent-blocked.cnf
15	965ca988015c9aee5a1a7b2136c1fe5d-os_fwalk_12.cnf
16	99d134de6323a845a2828596a48bbb1d-php17-mixed-15percent-blocked.cnf
17	a45b60e53917968f922b97c6f8aa8db3-unsat-set-b-fclqcolor-10-07-09.sat05-1282.reshuffled-07.cnf
18	a4b05fbc5be28207b704e1fae4b7c8a0-FileObject_open_12.cnf
19	a64f3c1afd7e0f6165efbe9fc2fc8003-pmg-12-UNSAT.sat05-3940.reshuffled-07.cnf
20	a6d7268b35eec18656a85ad91b0413e9-php17-mixed-35percent-blocked.cnf
21	ae9b7950ef1513068bb9339893ec8c50-WCNF_to_alien_14.cnf
22	af1e84bc2ab44d87d1c4c0cbf9e601c5-posixpath_expanduser_14.cnf
23	b09585f2346c207e9e14a3daf0de46cf-CNF_to_alien_11.cnf
24	b2145c28dbed385329ea73a06d9c519a-LZMAFile___init___14.cnf
25	b3840e295097a13e6697fff6be813eeb-php16-mixed-15percent-blocked.cnf

ID	instance name
26	c9af5b23c87350f5d817acc9ca7b69bb-CNF_to_alien_12.cnf
27	dd169198070f9aa35015de65e8209a05-LZMAFile_write_12.cnf
28	fd2af7622798171f23a4b8d2616df55e-StreamReader_readline_13.cnf

Table 7. Correspondence table between IDs and instance names of COMP23 instances.

Acknowledgments

We sincerely appreciate valuable comments and suggestions from the reviewers[24][25][26][27][28].

Footnotes

[1] available at https://researchmap.jp/t-sonobe/works/48813360

[2] https://github.com/dwavesystems/dwave-neal

[3] https://mat.tepper.cmu.edu/COLOR/instances.html

[4] https://satcompetition.github.io/2023/downloads.html

References

1. ^Isakov SV, Zintchenko IN, Rønnow TF, Troyer M. (2015). "Optimised simulated annealing for Ising spin glasses". Computer Physics Communications. 192: 265–271. doi:10.1016/j.cpc.2015.02.015.
2. ^Aramon M, Rosenberg G, Valiante E, Miyazawa T, Tamura H, et al. (2019). "Physics-inspired optimization for quadratic unconstrained problems using a digital annealer". Frontiers in Physics. 7. doi:10.3389/fphy.2019.00048.
3. ^Goto H, Endo K, Suzuki M, Sakai Y, Kanao T, et al. (2021). "High-performance combinatorial optimization based on classical mechanics". Science Advances. 7(6): eabe7953. doi:10.1126/sciadv.abe7953.
4. ^Li H, Wang J. (2025). "A collaborative neurodynamic algorithm for quadratic unconstrained binary optimization". IEEE Transactions on Emerging Topics in Computational Intelligence. 9(1): 228–239. doi:10.1109/TETCI.2024.3405370.

5. ^Lucas A. (2014). "Ising formulations of many NP problems". Frontiers in Physics. 2. doi:10.3389/fphy.2014.00005.

6. ^Chapuis G, Djidjev H, Hahn G, Rizk G. (2017). "Finding maximum cliques on a quantum annealer". In: Proceedings of the computing frontiers conference. pp. 63–70. doi:10.1145/3075564.3075575.

7. ^Cook SA. (1971). "The complexity of theorem-proving procedures". In: Proceedings of the 3rd annual ACM symposium on theory of computing. pp. 151–158. doi:10.1145/800157.805047.

8. ^Heule MJH, Kullmann O, Marek VW. (2016). "Solving and verifying the boolean pythagorean triples problem via cube-and-conquer". In: Theory and applications of satisfiability testing. pp. 228–245. doi:10.1007/978-3-319-40970-2_15.

9. ^Narodytska N, Kasiviswanathan SP, Ryzhyk L, Sagiv M, Walsh T. (2018). "Verifying properties of binarized deep neural networks". In: Proceedings of the thirty-second AAAI conference on artificial intelligence. pp. 6615–6624. doi:10.5555/3504035.3504845.

10. ^Stephan P, Brayton RK, Sangiovanni-Vincentelli AL. (2006). "Combinational test generation using satisfiability". Transactions on Computer-Aided Design of Integrated Circuits and Systems. 15(9): 1167–1176. doi:10.1109/43.536723.

11. ^Eén N, Sörensson N. (2003). "An extensible SAT-solver". In: Giunchiglia E, Tacchella A, editors. Theory and applications of satisfiability testing, 6th international conference. pp. 502–518. doi:10.1007/978-3-540-24605-3_37.

12. a, b, c, d, eChancellor N, Zohren S, Warburton PA, Benjamin SC, Roberts S. (2016). "A direct mapping of max k-SAT and high order parity checks to a chimera graph". Scientific Reports. 6(1): 37107. doi:10.1038/srep37107.

13. a, b, c, d, eNüßlein J, Zielinski S, Gabor T, Linnhoff-Popien C, Feld S. (2023). "Solving (max) 3-SAT via quadratic unconstrained binary optimization". In: International conference computational science. pp. 34–47. doi:10.1007/978-3-031-36030-5_3.

14. a, b, cZielinski S, Nüßlein J, Kölle M, Gabor T, Linnhoff-Popien C, et al. (2024). "Solving max-3SAT using QUBO approximation". arXiv preprint arXiv:240915891. doi:10.48550/arXiv.2409.15891.

15. a, bDouglass A, King AD, Raymond J. (2015). "Constructing SAT filters with a quantum annealer". In: Theory and applications of satisfiability testing — SAT 2015. pp. 104–120. doi:10.1007/978-3-319-24318-4_9.

16. ^Ibaraki T, Imamichi T, Koga Y, Nagamochi H, Nonobe K, et al. (2011). "Efficient branch-and-bound algorithms for weighted MAX-2-SAT". Math Program. 127(2): 297–343. doi:10.1007/s10107-009-0285-6.

17. a, bDe Kleer J. (1989). "A comparison of ATMS and CSP techniques". In: Proceedings of the 11th international joint conference on artificial intelligence - volume 1. pp. 290–296.

18. [a, b]*Zhang H, Stickely ME. (1996). "An efficient algorithm for unit propagation". Proc of AI-MATH. 96.*

19. ^*Glover F, Kochenberger G, Du Y. (2018). "A tutorial on formulating and using QUBO models". arXiv preprint arXiv:181111538. doi:10.48550/arXiv.1811.11538.*

20. ^*Zaman M, Tanahashi K, Tanaka S. (2021). "PyQUBO: Python library for QUBO creation". IEEE Transactions on Computers. doi:10.1109/TC.2021.3063618.*

21. ^*Tanahashi K, Takayanagi S, Motohashi T, Tanaka S. (2019). "Application of ising machines and a software development for ising machines". Journal of the Physical Society of Japan. 88(6): 061010. doi:10.7566/JPSJ.88.061010.*

22. ^*Lauria M, Elffers J, Nordström J, Vinyals M. (2017). "CNFgen: A generator of crafted benchmarks". In: Theory and applications of satisfiability testing. pp. 464–473. doi:10.1007/978-3-319-66263-3_30.*

23. [a, b]*Biere A, Faller T, Fazekas K, Fleury M, Froleyks N, et al. (2024). "CaDiCaL 2.0". In: Computer aided verification - 36th international conference, CAV. pp. 133–152. doi:10.1007/978-3-031-65627-9_7.*

24. ^*Hahn G. Review of: An approximated QUBO formulation for solving practical SAT problems. https://doi.org/10.32388/LE65KM. doi:10.32388/LE65KM.*

25. ^*Lavagna L. Review of: An approximated QUBO formulation for solving practical SAT problems. https://doi.org/10.32388/VODE1P. doi:10.32388/VODE1P.*

26. ^*Braida A. Review of: An approximated QUBO formulation for solving practical SAT problems. https://doi.org/10.32388/YFMPJ3. doi:10.32388/YFMPJ3.*

27. ^*Zhang Z. Review of: An approximated QUBO formulation for solving practical SAT problems. https://doi.org/10.32388/I86K3V. doi:10.32388/I86K3V.*

28. ^*Li H. Review of: An approximated QUBO formulation for solving practical SAT problems. https://doi.org/10.32388/NNN1AK. doi:10.32388/NNN1AK.*

Declarations

Funding: No specific funding was received for this work.

Potential competing interests: No potential competing interests to declare.