Binary Search-Based Methods for Solving Constraint Hierarchies over Finite Domains

Hiroshi Hosobe
Faculty of Computer and Information Sciences
Hosei University
Tokyo, Japan
hosobe@acm.org

Ken Satoh
Principles of Informatics Research Division
National Institute of Informatics
Tokyo, Japan
ksatoh@nii.ac.jp

***Abstract*—Constraint programming is a powerful tool for modeling and solving various problems. Especially, soft constraints are useful since they enable the treatment of over- and under-constrained real-world problems by relaxing conflicting constraints and introducing default constraints. Constraint hierarchies provide a soft constraint framework that introduces hierarchical preferences called strengths. In a constraint hierarchy, constraints are associated with strengths such as required, strong, medium, and weak, and a solution is obtained to maximally satisfy stronger constraints in the sense of a given solution criterion. In this paper, we propose three methods based on binary search for solving constraint hierarchies over finite domains by using a criterion called unsatisfied-count-better. Our methods solve constraint hierarchies by encoding them into ordinary constraint satisfaction problems and repeatedly solving the encoded problems with an external solver. We also present the implementations of our methods and the results of the experiment that we conducted to evaluate them.**

***Index Terms*—constraint programming, constraint solver, soft constraint, constraint hierarchy**

I. Introduction

Constraint programming (CP) is a paradigm of programming that allows the specification of a problem with *constraints* that express declarative relationships among variables. The technologies of CP derive from various fields including programming languages, artificial intelligence, databases, and operations research, and its applications range over various problems such as scheduling, planning, transportation, software design, network design, and bioinformatics [21].

A characteristic of CP is that it separates programming into modeling and solving. While the modeling of a problem is declaratively done with constraints, the constraints by themselves do not express how to solve the problem. Instead, the solving of the problem is automatically performed by software called a *constraint solver*. Therefore, programmers can devote themselves to modeling problems without the need to program how to solve the problems.

To realize this characteristic of CP, various frameworks of modeling and solving with constraints have been proposed. Constraint satisfaction problems (CSPs) have been widely used as the framework of modeling with constraints, and also there have been many studies on techniques for solving CSPs such as constraint propagation [8], [25]. However, such classical CSPs may easily become under- or over-constrained, causing many solutions or no solutions. To avoid such undesirable situations, programmers need to specify necessary and sufficient sets of constraints, which imposes extra burdens.

To solve this problem, *soft constraints* [18] have been proposed. While constraints in classical CSPs must always be satisfied, soft constraints are satisfied as much as possible. Therefore, even if there are conflicts among soft constraints, solutions can be obtained by relaxing them. Also, if constraints are insufficient, solutions can be reduced by introducing default constraints. Important instances of such soft CSPs are the maximum CSP (MaxCSP) and the weighted CSP (WCSP).

Constraint hierarchies [5] provide a soft constraint framework that introduces hierarchical preferences to further facilitate the modeling of problems with constraints. The preferences of constraints are called *strengths* and are often symbolically expressed as, e.g., required, strong, medium, and weak. In a constraint hierarchy, strong constraints are more respected than weaker ones in determining solutions. While weights that are sometimes used in other soft constraint frameworks are relative preferences, strengths in constraint hierarchies are absolute preferences in the sense that strong constraints are always more important than weaker ones. This property reduces the need to make detailed adjustments of strengths, making strengths easier-to-use than weights. In particular, the solution criterion called *unsatisfied-count-better* (UCB), which tries to satisfy larger numbers of stronger constraints, is useful because it can be regarded as generalizations of the MaxCSP and the WCSP.

In this paper, we propose three methods for solving constraint hierarchies over finite domains by using the UCB criterion. Our methods solve constraint hierarchies by encoding them into CSPs and repeatedly solving the encoded problems with an external solver. Although this encoding itself is basically similar to that of Hosobe and Satoh's HillClimbing method [14], our new methods are different in that all of them introduce binary search to improve the efficiency of optimization. Specifically, we propose the following three methods by introducing binary search in different ways:

- The Weighting method transforms a constraint hierarchy into a weighted CSP and optimizes its objective function;
- The Lexicographic method directly embeds the binary search-based optimization in the HillClimbing method;

This is the author's version. The final authenticated version is available online at https://doi.org/10.1109/ICTAI59109.2023.00035.

- The LevelWise method successively performs optimization from the strongest to the weakest level of a constraint hierarchy.

We present the implementations of our methods and the results of the experiment that we conducted to evaluate our methods. The results show that Lexicographic and LevelWise are more efficient than Weighting and HillClimbing for larger constraint hierarchies.

The rest of this paper is organized as follows. Section II presents previous research related to our work, and Section III briefly explains important preliminaries. Section IV provides the three methods that we propose in this paper. Section V presents the implementations of the proposed methods, and Section VI shows the experiment that we conducted to evaluate them. Section VII discusses our method, and finally Section VIII gives conclusions and future work.

II. Related Work

Modelling and solving soft constraints have been studied mainly in the context of soft CSPs [18]. Especially, specific instances of soft CSPs called the maximal CSP (MaxCSP) and the weighted CSP (WCSP) have been widely studied. In a MaxCSP, a solution is obtained by maximizing the number of satisfied constraints. In a WCSP, constraints are associated with weights, and a solution is obtained by maximizing the weighted count of satisfied constraints. Another framework closely related to soft CSPs is the constraint optimization problem (COP) [18]. A COP is represented as the pair of a classical CSP and an explicit objective function, unlike soft CSPs whose objective functions are implicitly constructed by the underlying frameworks. There has been much research on solving soft CSPs and COPs including consistency and search techniques.

Constraint hierarchies [5] have long been studied. Most of early research treated dataflow constraints by using a graph-based approach called local propagation [7]. However, algorithms based on local propagation were limited in their application areas because they were insufficient in processing simultaneous constraints and inequality constraints.

Algorithms for solving constraint hierarchies with linear constraints over real domains [2], [10], [16], [17] appropriately process simultaneous constraints and inequality constraints. Especially, Cassowary [2] is widely used as the internal solver of Apple Auto Layout [1]. Also, approximate algorithms [11], [15] have been proposed to solve constraint hierarchies with nonlinear constraints over real domains.

There has been research on the use of external solvers to solve constraint hierarchies. An example is the algorithm [12] that solves constraint hierarchies with nonlinear constraints over real domains by using the Z3 solver [4], [6]. This method is especially closely related to our work since both internally use binary search for optimization. However, it is focused on real domains and is not directly applicable to constraint hierarchies over finite domains that we tackle in this paper.

Although there have not been many studies on solving constraint hierarchies over finite domains, notable previous work was done by Bistarelli et al. [3]. They proposed consistency techniques for constraint hierarchies over finite domains. The techniques allowed the reduction of constraint hierarchies to make them more efficiently solvable. They also developed a branch-and-bound search algorithm for solving the reduced constraint hierarchies. Although their work and ours share the same goal of solving constraint hierarchies over finite domains, the approaches are completely different. Especially, it should be noted that our work is more convenient since our proposed methods can be easily implemented by using a state-of-the-art external CSP solver. Also, it should be noted that the experiments presented in their paper [3] were limited to small randomly generated problems with at most 10 variables.

III. Preliminaries

This section briefly describes important preliminaries to our work, namely, constraint hierarchies, how to encode constraint hierarchies into classical CSPs, the HillClimbing method for solving constraint hierarchies, and optimization based on binary search.

A. Constraint Hierarchies

In constraint hierarchies [5], constraints are associated with hierarchical preferences called strengths. A constraint hierarchy consists of a finite number of levels. The top level holds required (or hard) constraints that must always be satisfied, and lower levels contain preferential (or soft) constraints that can be relaxed if necessary. Strengths of constraints are often symbolically expressed as, e.g., required, strong, medium, and weak.

The solution set of a constraint hierarchy is determined by the optimization of potential solutions. For this purpose, a relation called a comparator is used to judge which of two potential solutions is better than the other. What solution set is obtained depends on the used comparator. Various concrete comparators have been proposed, and are roughly classified into global and local comparators. In our work, we use a global comparator called unsatisfied-count-better (UCB) that evaluates a potential solution in such a way that it should satisfy more constraints in upper levels.

In the following, we represent a strength as an integer between 0 and a certain positive integer l. More specifically, strength 0 corresponds to required constraints, and 1 and larger integers correspond to preferential, weaker constraints. We represent a constraint hierarchy as H, regard level k of H (which contains constraints with strength k) as consisting of m_k constraints, and let $c_{k,i}$ be the i-th constraint of level k of H. We use $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ to denote all the variables that appear in H.

B. Encoding Constraint Hierarchies

Hosobe and Satoh's HillClimbing method [14] for solving constraint hierarchies over finite domains first encodes a given

This is the author's version. The final authenticated version is available online at https://doi.org/10.1109/ICTAI59109.2023.00035.

constraint hierarchy H into the following classical CSP:

$$\mathsf{encode}(H) \equiv \left(\bigwedge_{i=1}^{m_0} c_{0,i}\right) \wedge \left(\bigwedge_{k=1}^{l}\bigwedge_{i=1}^{m_k}(s_{k,i} \in \{0,1\} \wedge (s_{k,i} = 1 \rightarrow c_{k,i}))\right),$$

where each $s_{k,i}$ is a variable called a *selector variable* that takes either 0 or 1. Intuitively, $\mathsf{encode}(H)$ indicates that $c_{k,i}$ with $k \geq 1$ must be satisfied when $s_{k,i}$ is 1. Solving $\mathsf{encode}(H)$ with various values of selector variables yields various potential solutions to H.

In the case of the UCB comparator, it is convenient to introduce the notion of the *degree of satisfaction* (DoS) of a potential solution to H. Given a potential solution to H characterized as the selector variables $\boldsymbol{s} = (s_{1,1}, \ldots, s_{1,m_1}, \ldots, s_{l,1}, \ldots, s_{l,m_l})$, the DoS $\delta(\boldsymbol{s})$ of this potential solution is defined as follows:

$$\delta(\boldsymbol{s}) = \left(\sum_{i=1}^{m_1} s_{1,i}, \sum_{i=1}^{m_2} s_{2,i}, \ldots, \sum_{i=1}^{m_l} s_{l,i}\right).$$

Intuitively, the DoS of a potential solution indicates how many constraints are satisfied in each preferential level. The UCB comparator better evaluates a DoS that is larger in the sense of the lexicographic order.

If a solver capable of treating constraints over finite domains and optimizing such a sequence of variable sums in the lexicographic order is available, solving a constraint hierarchy with the UCB comparator can be performed in a straightforward way. An example of such a solver is Z3 [4], [6] for satisfiability modulo theories (SMT). Especially, its ability to perform lexicographic multi-objective optimization enables the solving of H with the UCB comparator by optimizing $\delta(\boldsymbol{s})$ subject to $\mathsf{encode}(H)$. In Section VI, we evaluate our methods by using Z3 as a baseline.

C. The HillClimbing Method

The HillClimbing method [14] solves an encoded constraint hierarchy by performing optimization in a hill-climbing manner. The original method obtained all solutions to a constraint hierarchy for its use in constraint logic programming with constraint hierarchies (which is known as hierarchical constraint logic programming [29]). Also, because of its simplicity, it is easily modifiable to global comparators other than UCB and local comparators. However, in this paper, we restrict our attention to obtaining a single solution with UCB.

The HillClimbing method iteratively searches for a better potential solution according to the UCB comparator. During the search, it finds a single potential solution that is better than the best potential solution found thus far, and continues the search while there is such a potential solution. Since the UCB comparator is based on whether the constraints are satisfied, this iteration finally reaches an upper bound that is a solution to the constraint hierarchy.

Algorithm 1 shows the algorithm of the HillClimbing method. First, at line 2, it encodes a given constraint hierarchy

Algorithm 1: The HillClimbing method [14].

```
   Data: A constraint hierarchy H
   Result: A UCB solution v to H
1  begin
2      P ← encode(H);
3      (v, u) ← findSingleSolution(P);
4      if there exists no such (v, u) then
5          return None; // Unsatisfiable required
               constraints
6      end
7      while true do
8          (v', u') ← findSingleSolution(P ∧
                                (δ(s) >_lex δ(u)));
9          if there exists such (v', u') then
10             (v, u) ← (v', u');
11         else
12             return v; // Solution found
13         end
14     end
15 end
```

H into a classical CSP P. At line 3, it solves P by using an external CSP solver and finds a single solution $(\boldsymbol{v}, \boldsymbol{u})$, where $\boldsymbol{v}$ holds the values of the variables $\boldsymbol{x}$ appearing in H, and $\boldsymbol{u}$ holds the values of the selector variables $\boldsymbol{s}$ introduced in P. At lines 4 to 6, it checks whether the required constraints in H are satisfiable, and it terminates if they are unsatisfiable. At lines 7 to 14, it performs the optimization. At line 8, it tries to find a potential solution that is better than the current best one. For this purpose, it solves the conjunction of P and $\delta(\boldsymbol{s}) >_{\mathrm{lex}} \delta(\boldsymbol{u})$, where $>_{\mathrm{lex}}$ indicates the "greater than" relation for the lexicographic order. It should be noted that $\boldsymbol{s}$ is the vector of the selector variables (i.e., not yet uninstantiated) while $\boldsymbol{u}$ is the vector of the values of the selector variables for the current best (i.e., already instantiated). If there is such a better potential solution, it updates the current best at line 10. Otherwise, it returns the current best at line 12.

D. Binary Search-Based Optimization

Binary search is sometimes used to solve optimization problems. Especially, it is used for maximal satisfiability (MaxSAT) problems, where satisfiability (SAT) problems are relaxed in such a way that satisfied clauses should be maximized [9]. More specifically, such a method considers a SAT problem, in which the number of the satisfied clauses is encoded as a logical adder, and its range also is encoded in the way of combinatorial circuits. Then it maximizes the number of the satisfied clauses by successively updating the upper and lower bounds by binary search. The SAT-based constraint solver Scarab, which we use as an external solver for our proposed methods, is able to solve optimization problems by using binary search in this way [23].

This is the author's version. The final authenticated version is available online at https://doi.org/10.1109/ICTAI59109.2023.00035.

IV. Proposed Methods

This section proposes three methods called Weighting, Lexicographic, and LevelWise for solving constraint hierarchies over finite domains. These methods can be regarded as modifying the HillClimbing method described in Subsection III-C by introducing the binary search-based optimization described in Subsection III-D.

A. The Weighting Method

The Weighting method transforms a constraint hierarchy into a WCSP and optimizes its objective function by binary search. For this purpose, it first encodes a given constraint hierarchy H into $\mathsf{encode}(H)$, which is a classical CSP. It treats the UCB comparator by introducing the following objective function $\phi(\boldsymbol{s})$ of the selector variables:

$$\phi(\boldsymbol{s}) = \sum_{k=1}^{l} \left(w_k \sum_{i=1}^{m_k} s_{k,i} \right),$$

where each w_k is the weight for constraints with strength k defined as follows:

$$w_k = \prod_{k'=k+1}^{l} (m_{k'} + 1).$$

It is known that determining weights w_k as such sufficiently large values guarantees that strong constraints should always be more respected than weaker ones [19]. The lower bound of $\phi(\boldsymbol{s})$ is 0 (i.e., all the preferential constraints are unsatisfied), and the upper bound is $(\prod_{k=1}^{l}(m_k + 1)) - 1$ (i.e., all the preferential constraints are satisfied). Therefore, performing binary search with these lower and upper bounds as the starting point enables the maximization of $\phi(\boldsymbol{s})$ subject to $\mathsf{encode}(H)$, which obtains a solution to H.

Algorithm 2 shows the algorithm of the Weighting method. Lines 2 to 6 are the same as those for the HillClimbing method. At lines 7 and 8, it sets the current lower and upper bounds b_{low} and b_{up} to the initial values. At lines 9 to 18, it performs optimization based on binary search. At line 10, it computes the midpoint b of the lower and upper bounds. At line 11, it tries to find a potential solution that is better than or equal to the one corresponding to b. For this purpose, it solves the conjunction of P and $\phi(\boldsymbol{s}) \geq b$. If there is such a better potential solution, it updates the current best at line 13 and the lower bound at line 14. Otherwise, it updates the upper bound at line 16. Finally, after the binary search, it returns the solution at line 19.

Algorithm 2: The Weighting method.

```
1  begin
2      P ← encode(H);
3      (v, u) ← findSingleSolution(P);
4      if there exists no such (v, u) then
5          return None; // Unsatisfiable required
             constraints
6      end
7      b_low ← 0;
8      b_up ← (∏_{k=1}^{l}(m_k + 1)) − 1;
9      while b_low < b_up do // Perform binary search
10         b ← ⌈(b_low + b_up)/2⌉;
11         (v′, u′) ← findSingleSolution(P ∧ (φ(s) ≥ b));
12         if there exists such (v′, u′) then
13             (v, u) ← (v′, u′);
14             b_low ← φ(u);
15         else
16             b_up ← b − 1;
17         end
18     end
19     return v; // Solution found
20 end
```

B. The Lexicographic Method

The Lexicographic method directly embeds the binary search-based optimization in the HillClimbing method. While HillClimbing successively improved the DoS of the current best potential solution, the Lexicographic method performs binary search according to the satisfiability of the conjunction of $\mathsf{encode}(H)$ and $\delta(\boldsymbol{x}) \geq_{\mathrm{lex}} \boldsymbol{d}_{\mathrm{mid}}$ (where $\geq_{\mathrm{lex}}$ indicates the lexicographic "greater than or equal to" relation) by using the midpoint $\boldsymbol{d}_{\mathrm{mid}}$ of the current lower and upper bounds of DoS. The initial lower and upper bounds are $(0, 0, \ldots, 0)$ (i.e., all the constraints are unsatisfied) and $(m_1, m_2, \ldots, m_k)$ (i.e., all the constraints are satisfied) respectively.

Given the current lower and upper bounds $\boldsymbol{d}_{\mathrm{lb}} = (d_1^{\mathrm{lb}}, d_2^{\mathrm{lb}}, \ldots, d_l^{\mathrm{lb}})$ and $\boldsymbol{d}_{\mathrm{ub}} = (d_1^{\mathrm{ub}}, d_2^{\mathrm{ub}}, \ldots, d_l^{\mathrm{ub}})$, their midpoint $\boldsymbol{d}_{\mathrm{mid}} = (d_1^{\mathrm{mid}}, d_2^{\mathrm{mid}}, \ldots, d_l^{\mathrm{mid}})$ is defined as follows:

$$\boldsymbol{d}_{\mathrm{mid}} = \mathsf{uncomb}\left(\lceil (\mathsf{comb}(\boldsymbol{d}_{\mathrm{lb}}) + \mathsf{comb}(\boldsymbol{d}_{\mathrm{ub}}))/2 \rceil\right),$$

where comb is the function defined as

$$\mathsf{comb}(\boldsymbol{d}) = \sum_{k=1}^{l} \left(\left(\prod_{k'=k+1}^{l} (m_{k'} + 1) \right) d_k \right),$$

and uncomb is the inverse function of comb, where $\boldsymbol{d} = (d_1, d_2, \ldots, d_l)$.

Algorithm 3 shows the algorithm of the Lexicographic method. It is an intermediate between the HillClimbing and Weighting methods. As a binary search algorithm, it is similar to the Weighting method. However, at line 11, it tries to solve the conjunction of P and $\delta(\boldsymbol{s}) \geq_{\mathrm{lex}} \mathsf{uncomb}(b)$, which is more similar to the HillClimbing method because both use relations based on the lexicographic order.

C. The LevelWise Method

The LevelWise method successively performs optimization from the strongest to the weakest level. It can be regarded as repeatedly transforming a constraint hierarchy into MaxCSPs from the strongest to the weakest level. Such an approach to solving constraint hierarchies is known as the refining method [13]. More specifically, at the step of optimizing level

This is the author's version. The final authenticated version is available online at https://doi.org/10.1109/ICTAI59109.2023.00035.

Algorithm 3: The Lexicographic method.

Data: A constraint hierarchy H
Result: A UCB solution $\boldsymbol{v}$ to H

```
1  begin
2      P ← encode(H);
3      (v, u) ← findSingleSolution(P);
4      if there exists no such (v, u) then
5          return None; // Unsatisfiable required
             constraints
6      end
7      b_low ← 0;
8      b_up ← comb((m_1, m_2, ..., m_k));
9      while b_low < b_up do // Perform binary search
10         b ← ⌈(b_low + b_up)/2⌉;
11         (v', u') ← findSingleSolution(P ∧
                          (δ(s) ≥_lex uncomb(b)));
12         if there exists such (v', u') then
13             (v, u) ← (v', u');
14             b_low ← comb(δ(u));
15         else
16             b_up ← b − 1;
17         end
18     end
19     return v; // Solution found
20 end
```

Algorithm 4: The LevelWise method

Data: A constraint hierarchy H
Result: A UCB solution $\boldsymbol{v}$ to H

```
1  begin
2      P ← ⋀_{i=1}^{m_0} c_{0,i};
3      v ← findSingleSolution(P);
4      if there exists no such v then
5          return None; // Unsatisfiable required
             constraints
6      end
7      for k = 1 to l do // Process each level
8          P ← P ∧ (⋀_{i=1}^{m_k}(s_{k,i} ∈ {0,1} ∧
                          (s_{k,i} = 1 → c_{k,i})));
9          b_low ← 0;
10         b_up ← m_k;
11         while b_low < b_up do // Perform binary search
12             b ← ⌈(b_low + b_up)/2⌉;
13             (v', u') ← findSingleSolution(P ∧
                          (Σ_{i=1}^{m_k} s_{k,i} ≥ b));
14             if there exists such (v', u') then
15                 (v, u) ← (v', u');
16                 b_low ← Σ_{i=1}^{m_k} u_{k,i};
17             else
18                 b_up ← b − 1;
19             end
20         end
21         P ← P ∧ (Σ_{i=1}^{m_k} s_{k,i} = b_low);
22     end
23     return v; // Solution found
24 end
```

k, the LevelWise method treats the required constraints and the preferential constraints with strengths 1 to k, and maximizes the satisfied constraints with strength k while keeping the same numbers of the satisfied constraints with strengths 1 to $k-1$ as it was able to satisfy until the previous step.

Algorithm 4 shows the algorithm of the LevelWise method. At lines 2 to 6, it tries to solve only the required constraints, and terminates if there is no solution. At lines 7 to 22, it successively processes each level k from 1 to l. At line 8, it additionally encodes constraints with strength k. At lines 9 to 20, it maximizes the satisfied constraints by performing binary search. At line 21, it adds the constraint that keeps the number of the satisfied constraints with strength k. Finally, after processing all the levels, it returns the solution at line 23.

V. Implementation

Using the methods proposed in Section IV, we implemented a solver of constraint hierarchies over finite domains in the Scala language. In addition to the implementations of the three proposed methods, this solver includes an implementation of the HillClimbing method (described in Subsection III-C) and an implementation using the lexicographic multi-objective optimization of Z3 (described in Subsection III-B). We adopted a SAT-based CP system called Scarab [23] as the external solver of the three proposed methods and the HillClimbing method.[1] Scarab uses SAT4J as its standard external SAT solver. To encode constraints into SAT problems, Scarab provides three methods called order encoding [24], log encoding (see, e.g., [28]), and pseudo-Boolean encoding, and we used order and log encoding. Our solver supports linear constraints over finite integer domains, Boolean constraints with 0/1-valued variables, logical combinations of such linear and Boolean constraints, and all-different constraints. The implementation of our solver currently consists of approximately 2000 lines of code.

VI. Experiment

This section presents the experiment that we conducted to evaluate the proposed methods.

[1] Although we attempted to use the CP-SAT solver of OR-Tools [20] as an external solver, we found that it was not sufficient for our purpose. The main problem is that the CP-SAT solver is not general enough to encode a constraint hierarchy in the way described in Subsection III-B; it does not allow expressing selector variables as shown in encode(H). Alternatively, it can express selector variables by using a special method called `onlyEnforceIf`, but the method is limited to linear constraints. It should be noted that any of the problems used in our experiment in Section VI could not be treated with only such linear constraints.

This is the author's version. The final authenticated version is available online at https://doi.org/10.1109/ICTAI59109.2023.00035.

A. Procedure

In this experiment, we used two kinds of problems that we created by introducing constraint hierarchies into the following two kinds of problems:

- Pandiagonal Latin squares (PLSs);
- Existing benchmark problems from the MAX-CSP 2008 Competition, which was held as part of the Third International CSP Solver Competition [27] at the CP2008 conference.

A PLS is the problem of placing distinct numbers of 1 to n on each row, column, and generalized diagonal of $n \times n$ cells. It differs from an ordinary Latin square in that it generalizes diagonals of cells. An $n \times n$ PLS is modeled with $n \times n$ variables and $4n$ all-different constraints. If we consider $2 \leq n \leq 13$, $n \times n$ PLSs are satisfiable for $n = 5, 7, 11, 13$ and are unsatisfiable otherwise. In this experiment, we consider the problems of solving PLSs with constraint hierarchies, which we call hierarchical PLSs. More specifically, constraints for rows, columns, generalized diagonals falling to the right, and generalized diagonals rising to the right are assigned strengths of 1, 2, 3, and 4 respectively (there are no required constraints). With this modification, PLSs that are unsatisfiable in the classical cases get to have solutions. Fig. 1 shows solutions of hierarchical PLSs. In Fig. 1(a), distinct numbers are placed on each direction since the 5×5 classical PLS is satisfiable. In Fig. 1(b), distinct numbers are placed on each row and column, only 4 generalized diagonals falling to the right have distinct numbers, and no generalized diagonals rising to the right have distinct numbers since the optimal DoS of the 6×6 hierarchical PLS is $(6, 6, 4, 0)$.

(a)

3	4	5	1	2
5	1	2	3	4
2	3	4	5	1
4	5	1	2	3
1	2	3	4	5

(b)

3	1	4	6	5	2
6	5	2	4	3	1
4	2	6	3	1	5
2	6	5	1	4	3
1	4	3	5	2	6
5	3	1	2	6	4

Fig. 1. Solutions to (a) the 5×5 and (b) the 6×6 hierarchical PLS.

We selected 27 benchmark problems from the MAX-CSP 2008 Competition. The problems consist of n-ary intensional constraints, and fall in three categories, namely, chessboard coloration, Schur's Lemma, and radar surveillance. All the problems are unsatisfiable as classical CSPs, and were solved as MaxCSPs within the time limit of one hour by at least one solver in the competition. The problems consist of linear constraints over finite integer domains, Boolean constraints with 0/1-valued variables, and their logical combinations. In our experiment, we introduce constraint hierarchies into these problems, which we call the hierarchical benchmark problems. Specifically, for each problem, we associate the first, the second, the third, and the fourth quarter of the constraints with strengths 1, 2, 3, and 4 respectively.

We compared nine instances of our solver for the hierarchical PLSs and five instances for the hierarchical benchmark problems. We included Z3 (using lexicographic multi-objective optimization) and HillClimbing as baselines. For the hierarchical PLSs, we used both order and log encoding for HillClimbing, Weighting, Lexicographic, and LevelWise. For the hierarchical benchmark problems, we used only log encoding since our preliminary experiment had showed that order encoding was not efficient for these problems in terms of memory. We measured the execution times that these instances of our solver needed to solve the hierarchical PLSs with $2 \leq n \leq 13$ and the hierarchical benchmark problems. We assigned 4 GB of memory to the Java virtual machine, and limited the execution time to the maximum of 20 minutes.

We used a computer with an M1 Ultra processor running macOS 13.6. The external solvers and programming languages processors that we used were Z3 4.12.2, Scarab 1.9.6 (including SAT4J), Scala 2.12.18, and Eclipse Temurin JDK 17.0.8.1+1. It should be noted that Z3 was written in C++ whereas SAT4J and Scarab were written in Java and Scala respectively. In general, as the running time of Z3 becomes longer, the overhead of invoking Z3 from the Java virtual machine becomes more negligible. In these senses, Z3 as an external solver has advantages over Scarab.

B. Results

Tables I and II show the experimental results of the hierarchical PLSs and the hierarchical benchmark problems respectively. In these tables, "M.O." indicates that the execution was abnormally terminated due to `OutOfMemoryError`, and "T.O." indicates that the execution was not completed within the time limit. The consistent DoS was finally obtained in all the cases that the executions were normally completed. Especially, all the constraints were satisfied for the hierarchical PLSs with $n = 5, 7, 11, 13$, whose corresponding classical PLSs are satisfiable.

For the small hierarchical PLSs with $n = 2, 3, 5$ and the one with $n = 4$, the solver using Z3 and the HillClimbing method using order encoding respectively computed solutions for the shortest times. However, for the larger hierarchical PLSs with $n \geq 6$, either the Lexicographic or the LevelWise method using order encoding computed solutions for the shortest times. The Weighting method using order encoding not only needed long execution times but also caused out of memory for the hierarchical PLSs with $n \geq 6$. The Weighting, Lexicographic, and LevelWise methods using log encoding did not terminate within the time limit for the hierarchical PLSs with $n \geq 8$.

For most of the hierarchical benchmark problems, the solver using Z3 computed solutions for the shortest times. However, it should be emphasized again that Z3 was written in C++ whereas Scarab and SAT4J were written in Scala and Java respectively. Therefore, we consider that our methods were sufficiently faster than Z3 for the problems of chessboard coloration and Schur's Lemma, and also that our methods were comparable with Z3 for radar surveillance. Among our methods, LevelWise was always faster than Lexicographic for chessboard coloration and radar surveillance, but Lexicographic was faster than LevelWise for two problems of Schur's Lemma.

This is the author's version. The final authenticated version is available online at https://doi.org/10.1109/ICTAI59109.2023.00035.

TABLE I
EXECUTION TIMES IN MILLISECONDS NEEDED TO SOLVE THE $n \times n$ HIERARCHICAL PLSS.

			Order encoding				Log encoding			
n	Optimal DoS	Z3	Hill-Climbing	Weight-ing	Lexico-graphic	Level-Wise	Hill-Climbing	Weight-ing	Lexico-graphic	Level-Wise
2	(2,2,0,0)	**37**	104	140	125	108	127	139	152	121
3	(3,3,3,0)	**57**	130	223	146	138	181	176	187	166
4	(4,4,2,2)	177	**165**	2243	194	169	384	307	360	261
5	(5,5,5,5)	**153**	202	5562	181	177	426	245	317	256
6	(6,6,4,0)	T.O.	2386	M.O.	**2194**	4695	29833	46902	30568	44328
7	(7,7,7,7)	5857	319	M.O.	250	**242**	8316	3260	3835	2527
8	(8,8,6,6)	T.O.	2493	M.O.	1681	**1461**	68299	T.O.	T.O.	T.O.
9	-	T.O.	T.O.	M.O.	T.O.	T.O.	T.O.	T.O.	T.O.	T.O.
10	(10,10,8,8)	T.O.	828585	M.O.	106225	**92755**	T.O.	T.O.	T.O.	T.O.
11	(11,11,11,11)	T.O.	T.O.	M.O.	**3436**	49852	T.O.	T.O.	T.O.	T.O.
12	-	T.O.	T.O.	M.O.	T.O.	T.O.	T.O.	T.O.	T.O.	T.O.
13	(13,13,13,13)	T.O.	T.O.	M.O.	88893	**20733**	T.O.	T.O.	T.O.	T.O.

TABLE II
EXECUTION TIMES IN MILLISECONDS NEEDED TO SOLVE THE HIERARCHICAL BENCHMARK PROBLEMS. NOTE THAT OUR METHODS AND HillClimbing WERE EXECUTED ON THE JAVA VIRTUAL MACHINE WHILE Z3 WAS EXECUTED IN NATIVE CODE.

Problem	Number of variables	Number of constraints	Optimal DoS	Z3	Hill-Climbing	Weight-ing	Lexico-graphic	Level-Wise
cc-5-5-2	25	100	(25,25,25,23)	**555**	1040	637	876	**630**
cc-6-6-2	36	225	(56,56,56,45)	**2178**	6187	4103	4531	**2311**
lemma-12-9-mod	12	30	(7,8,7,7)	2598	**563**	**683**	874	939
lemma-15-9-mod	15	49	(12,12,12,10)	10340	**2650**	5776	**2609**	5772
lemma-24-3	24	132	(33,33,33,32)	**317**	732	711	600	**541**
radar-8-24-3-2-1	144	64	(16,15,16,16)	**157**	444	568	552	**397**
radar-8-24-3-2-2	144	64	(16,15,16,16)	**176**	499	585	577	**406**
radar-8-24-3-2-5	144	64	(16,15,16,16)	**164**	528	626	585	**417**
radar-8-24-3-2-18	144	64	(15,15,16,16)	**175**	464	741	669	**423**
radar-8-24-3-2-19	144	64	(15,16,16,16)	**174**	508	706	649	**413**
radar-8-24-3-2-32	144	64	(16,15,16,15)	**174**	550	515	629	**407**
radar-8-24-3-2-41	144	64	(16,14,16,16)	**173**	467	505	596	**399**
radar-8-24-3-2-44	144	64	(16,16,15,16)	**156**	532	490	473	**408**
radar-8-24-3-2-47	144	64	(16,14,16,16)	**163**	467	607	577	**402**
radar-9-28-4-2-2	168	81	(20,19,20,21)	**294**	881	1017	1099	**658**
radar-9-28-4-2-7	168	81	(20,20,20,20)	**394**	1050	767	769	**607**
radar-9-28-4-2-9	168	81	(19,20,20,21)	**343**	980	1312	1136	**637**
radar-9-28-4-2-12	168	81	(20,19,20,21)	**406**	953	1135	1104	**704**
radar-9-28-4-2-26	168	81	(19,20,20,21)	**300**	871	1273	1096	**599**
radar-9-28-4-2-29	168	81	(20,20,19,21)	**327**	1339	1001	1114	**809**
radar-9-28-4-2-30	168	81	(20,19,20,21)	**264**	971	847	1149	**660**
radar-9-28-4-2-31	168	81	(20,20,19,21)	**294**	1126	913	923	**690**
radar-9-28-4-2-34	168	81	(20,20,18,21)	**288**	1064	999	1037	**663**
radar-9-28-4-2-43	168	81	(19,20,20,21)	**300**	1129	1388	1188	**630**
radar-9-28-4-2-47	168	81	(19,19,20,21)	**304**	944	1225	1265	**673**
radar-9-28-4-2-48	168	81	(19,20,18,21)	**338**	1227	1447	1288	**795**
radar-9-28-4-2-49	168	81	(19,20,20,21)	**297**	969	1185	1114	**615**

VII. DISCUSSION

From the results of the experiment, we did not observe that either of the Lexicographic or the LevelWise method was more efficient than the other. However, we cannot immediately conclude that they have comparable efficiency since they internally perform largely different optimization. There might be situations that are appropriate or inappropriate for these methods although it was not revealed by our experiment. We need to make a detailed comparison of their internal processing as well as comparisons using other problems.

The Weighting method almost always did not exhibit better performance than the other methods whichever of order and log encoding was used. Although it is a simple and clear method based on the transformation of a constraint hierarchy into a WCSP, the method is not very appropriate for an external solver, which is because the resulting finite domains easily become large. However, there exist other SAT encoding methods [22], [26], [30]. For example, hybrid encoding [22] automatically switches between order and log encoding according to the sizes of finite domains. Therefore, it might be

This is the author's version. The final authenticated version is available online at https://doi.org/10.1109/ICTAI59109.2023.00035.

possible to improve the performance of the Weighting method by adopting a more appropriate SAT encoding method.

The efficiency of the three proposed methods as well as the HillClimbing method is affected by the quality of potential solutions found during the iterative search. However, unlike numerical problems that could use the derivatives of objective functions, combinatorial optimization problems generally have difficulty in searching for potential solutions in promising directions. Therefore, our methods currently do not adopt particular heuristics for this purpose.

The three proposed methods compute a single solution to a constraint hierarchy. However, in the same way as the HillClimbing method, it is easy to extend our methods to obtain all the solutions to a constraint hierarchy. This will enable our methods to be applied to hierarchical constraint logic programming [29].

It also is possible to modify our methods to use global comparators other than UCB. However, other global comparators such as least-squares-better often require larger domains than UCB, which might degrade the performance. By contrast, it is less clear whether our methods can be modified for local comparators since binary search is not easily applicable to local comparators.

VIII. Conclusions and Future Work

In this paper, we proposed three methods called Weighting, Lexicographic, and LevelWise for solving constraint hierarchies over finite domains. We constructed these methods by modifying the HillClimbing method by introducing optimization based on binary search. We also provided the implementations of these methods, and presented the results of the experiment that we conducted to evaluate these methods. The results showed that Lexicographic and LevelWise were more efficient than the others.

Our future work includes further evaluation of the proposed methods. Especially, we are considering the use of other standard benchmark problems. The MAX-CSP 2008 Competition of the Third International CSP Solver Competition had provided other benchmark problems that we did not use in our experiment. To perform further experiments using these problems, we need to enhance our solver by implementing missing facilities such as the treatment of extensional constraints. Other future work is to explore an external CSP solver that is suitable for our methods. Especially, we want to investigate whether CSP solvers other than SAT-based and SMT solvers will work well for our methods. Another direction is to explore the possibility of parallelizing our methods for the execution on a multi-core processor or a multi-processor computer.

Acknowledgment

This work was supported by JST AIP Trilateral AI Research Grant Number JPMJCR20G4.

References

[1] Apple, Inc., *Auto Layout Guide*, 2011.

[2] G. J. Badros, A. Borning, and P. J. Stuckey, "The Cassowary linear arithmetic constraint solving algorithm," *ACM Trans. Comput.-Human Interact.*, vol. 8, no. 4, pp. 267–306, 2001.

[3] S. Bistarelli, P. Codognet, K. C. Hui, and J. H.-M. Lee, "Solving finite domain constraint hierarchies by local consistency and tree search," *J. Exp. Theor. Artif. Intell.*, vol. 21, no. 4, pp. 233–257, 2009.

[4] N. Bjøner, L. de Moura, L. Nachmanson, and C. M. Wintersteiger, "Programming Z3," in *SETSS Tutorial Lectures*, ser. LNCS, vol. 11430, 2019, pp. 148–201.

[5] A. Borning, B. Freeman-Benson, and M. Wilson, "Constraint hierarchies," *Lisp Symbolic Comput.*, vol. 5, no. 3, pp. 223–270, 1992.

[6] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. TACAS*, ser. LNCS, vol. 4963, 2008, pp. 337–340.

[7] B. N. Freeman-Benson, J. Maloney, and A. Borning, "An incremental constraint solver," *Comm. ACM*, vol. 33, no. 1, pp. 54–63, 1990.

[8] E. C. Freuder and A. K. Mackworth, "Constraint satisfaction: An emerging paradigm," in *Handbook of Constraint Programming*. Elsevier, 2006, ch. 2, pp. 13–27.

[9] Z. Fu and S. Malik, "On solving the partial Max-SAT problem," in *Proc. SAT*, ser. LNCS, vol. 4121, 2006, pp. 252–265.

[10] H. Hosobe, "A scalable linear constraint solver for user interface construction," in *Proc. CP*, ser. LNCS, vol. 1894, 2000, pp. 218–232.

[11] ——, "A modular geometric constraint solver for user interface applications," in *Proc. ACM UIST*, 2001, pp. 91–100.

[12] ——, "Solving hierarchical soft constraints with an SMT solver," in *Proc. ICCAE*. ACM, 2020, pp. 42–46.

[13] H. Hosobe and S. Matsuoka, "A foundation of solution methods for constraint hierarchies," *Constraints*, vol. 8, no. 1, pp. 41–59, 2003.

[14] H. Hosobe and K. Satoh, "Solving constraint hierarchies for hierarchical constraint logic programming," in *Proc. JSAI Conf.*, no. 4F3-OS-8b-04, 2022, pp. 1–4, in Japanese.

[15] N. Hurst, K. Marriott, and P. Moulder, "Dynamic approximation of complex graphical constraints by linear constraints," in *Proc. ACM UIST*, 2002, pp. 191–200.

[16] N. Jamil, J. Müller, A. Naeem, C. Lutteroth, and G. Weber, "Extending linear relaxation for non-square matrices and soft constraints," *J. Comput. Appl. Math.*, vol. 308, pp. 346–360, 2016.

[17] K. Marriott and S. S. Chok, "QOCA: A constraint solving toolkit for interactive graphical applications," *Constraints*, vol. 7, no. 3–4, pp. 229–254, 2002.

[18] P. Meseguer, F. Rossi, and T. Schiex, "Soft constraints," in *Handbook of Constraint Programming*. Elsevier, 2006, ch. 9, pp. 281–328.

[19] H. Okamoto and K. Satoh, "An algorithm to compute minimal models in prioritized circumscription by 0-1 integer programming," *J. JSAI*, vol. 15, no. 3, pp. 511–517, 2000, in Japanese.

[20] L. Perron and V. Furnon, "OR-Tools, ver. 9.6," Google, 2023. [Online]. Available: https://developers.google.com/optimization/

[21] F. Rossi, P. van Beek, and T. Walsh, "Introduction," in *Handbook of Constraint Programming*. Elsevier, 2006, ch. 1, pp. 3–12.

[22] T. Soh, M. Banbara, and N. Tamura, "Proposal and evaluation of hybrid encoding of CSP to SAT integrating order and log encodings," *Intl. J. Artif. Intell. Tools*, vol. 26, no. 1, pp. 1–29, 2017.

[23] T. Soh, N. Tamura, and M. Banbara, "Scarab: A rapid prototyping tool for SAT-based constraint programming systems," in *Proc. SAT*, ser. LNCS, vol. 7962, 2013, pp. 429–436.

[24] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara, "Compiling finite linear CSP into SAT constraints," *Constraints*, vol. 14, no. 2, pp. 254–272, 2009.

[25] E. Tsang, *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[26] F. Ulrich-Oltean, P. Nightingale, and J. A. Walker, "Selecting SAT encodings for pseudo-boolean and linear integer constraints," in *Proc. CP*, ser. LIPIcs, vol. 235, no. 38, 2022, pp. 1–17.

[27] M. van Dongen, C. Lecoutre, and O. Roussel, "Third international CSP solver competition," 2008. [Online]. Available: https://www.cril.univ-artois.fr/CPAI08/

[28] T. Walsh, "SAT v CSP," in *Proc. CP*, ser. LNCS, vol. 1894, 2000, pp. 441–456.

[29] M. Wilson and A. Borning, "Hierarchical constraint logic programming," *J. Log. Program.*, vol. 16, no. 3–4, pp. 227–318, 1993.

[30] N.-F. Zhou and H. Kjellerstrand, "Optimizing SAT encodings for arithmetic constraints," in *Proc. CP*, ser. LNCS, vol. 10416, 2017, pp. 671–686.