外部ソルバを用いた有限領域上の制約階層の解消法

細部 博史　佐藤 健

制約プログラミングにおける問題の記述を容易にすることを目的として，制約の優先度を導入した制約階層が提案されている．本研究では，有限領域上の制約を扱い，大域的比較子と呼ばれる解の基準の一つを用いた制約階層を解消する手法を提案する．本手法は，対象とする制約階層を，優先度を含まない問題に変換し，外部ソルバを呼び出して問題を解く処理を繰り返すことで，制約階層を解消する．本研究ではさらに本手法の実装を示し，その動作に関する実験の結果を与える．

1　はじめに

制約プログラミング (constraint programming; CP) は，変数間の関係を宣言的に表現する**制約**を用いて問題を記述し解決するプログラミングの方法である．その技術はプログラミング言語，人工知能，データベース，オペレーションズリサーチ等の様々な分野に由来し，その応用はスケジューリング，プランニング，配送計画，ソフトウェア設計，ネットワーク設計，バイオインフォマティクス等の様々な問題に対して行われている [20].

CP の特徴は問題のモデル化と解決を明確に分離している点にある．問題のモデル化は制約によって宣言的に行われるが，制約自体は問題の解き方を表現しない．一方，問題の解決は，制約ソルバと呼ばれるソフトウェアによって自動的に制約を解消することで行われる．このため，プログラマは問題のモデル化に専念でき，その解き方をプログラムに記述する必要がない．

CP のこの特徴を実現するため，制約を用いた種々のモデル化と解決の枠組みが研究されている．初期の CP では，制約充足問題 (constraint satisfaction problem; CSP) と呼ばれるモデル化の枠組みが主に用いられ，それを対象とした制約伝播等の解決の技術が盛んに研究されていた [6] [24]．しかし，CSP によるモデル化では，制約を過不足なく指定していなければ，解が多数生成されたり，解が全くなくなったりといった状況が生じるため，プログラマの負担が大きかった．

この問題を解決するため，ソフト制約 [18] が提案された．CSP における制約が常に満たされる必要があったのに対して，ソフト制約は可能な限り満たされるものであり，必要に応じて緩和される．このため，ソフト制約同士に矛盾がある場合でも，それらが緩和されることで解が得られる．また，制約が不足する場合には，デフォルトで満たすべきソフト制約を利用することで，解の個数を削減できる †1.

制約階層 [3] は CP における問題のモデル化を容易にすることを目的として，制約の階層的な優先度を導入したソフト制約の枠組みである．制約の優先度は**強**

Solving Constraint Hierarchies over Finite Domains by Using an External Solver. (This is an unrefereed paper. The authors hold the copyright.)

Hiroshi Hosobe, 法政大学情報科学部, Faculty of Computer and Information Sciences, Hosei University.

Ken Satoh, 国立情報学研究所, National Institute of Informatics.

†1　多くのソフト制約の枠組みでは，必ず満たすべきハード制約をサポートしているため，重要な制約をハード制約として，デフォルトの制約をソフト制約として記述できる．

さと呼ばれ，強い制約は弱い制約よりも解を求める際に常に優先される．他のソフト制約の枠組みでしばしば用いられる重みが相対的な基準であるのに対して，制約階層における強さは絶対的な基準であるため，細かい調整が必要なく，重みよりも指定しやすい．

本研究では，有限領域上の制約を扱い，大域的比較子と呼ばれる解の基準の一つを用いた制約階層を解消する手法を三つ提案する．本手法は，対象とする制約階層を，優先度を含まない問題に変換し，外部ソルバを呼び出して問題を解く処理を繰り返すことで，制約階層を解消する．本手法は，著者らが過去に提案した制約階層解消法[15] を基礎として，二分探索による最適化を導入したものである．本研究ではさらに本手法の実装を示し，その動作に関する実験の結果を与える．

2 関連研究

制約階層の研究は多数行われている．初期の研究の多くはグラフ的な手法でデータフロー制約を扱っていた[5]．この種のアルゴリズムは，連立した制約や不等式制約の処理能力が不十分であったため，応用領域が限られていた．

線形制約を扱うアルゴリズム[2][8][17] は，制約階層における連立した制約や不等式制約を適切に処理できる．特に Cassowary [2] は，Apple Auto Layout [1] の内部のソルバとして広く利用されている．しかし，これらは幾何制約等の非線形制約を扱えない．

非線形制約を含む制約階層を解くために，近似的なアルゴリズム[10][16] が提案されている．また，階層的最小 2 乗法[11] や，階層的 Lagrange 乗数法[12] に基づく，非線形制約を含む制約階層のためのより高精度なアルゴリズムも提案されている．

制約階層を解くために外部ソルバを用いる研究も行われている．その一つとして，SMT ソルバ Z3 [4] を用いて，非線形制約からなる制約階層を解くアルゴリズム[13] が提案されている．

3 準備

本節では，制約階層と，本研究の基礎となる制約階層解消法，二分探索による最適化について概説する．

3.1 制約階層

制約階層[3] で，各制約には，強さと呼ばれる階層的な優先度 (選好) が付与される．制約階層は有限個のレベルからなり，最上位のレベルは，常に満たすべき必須制約 (ハード制約に相当) を保持し，下位のレベルは，必要ならば緩和されうる選好制約 (ソフト制約に相当) を保持する．制約の強さは記号的に required, strong, medium, weak と表されることが多い．

制約階層の解集合は，解候補に関する最適化によって求められる．このために，二つの解候補のいずれが良いかを判定する，比較子と呼ばれる関係が用いられる．得られる解集合は比較子によって異なり，具体的な比較子として様々なものが提案され，大きく大域的比較子と局所的比較子に分かれている．本研究では，満たされる制約の個数が上位のレベルほど多くなるように解候補の良さを判定する大域的比較子 unsatisfied-count-better (UCB) を用いる．

以下では，制約階層の強さを 0 から正の整数 l までの整数で表し，0 を必須制約，1 以上を選好制約の強さ (値が大きいほど弱い) とする．また，一つの制約階層を H で表し，強さ k に対応する H のレベルは m_k 個の制約からなるとし，その i 番目の制約を $c_{k,i}$ とする．簡単のため，必須制約同士には矛盾がないと仮定する．

3.2 Naive 法

著者らは過去に有限領域上の制約階層の解消法を提案した[15]．本研究で提案する手法はこの制約階層解消法を基礎としたものである．この基礎となる手法を以下では Naive 法と呼ぶ．なお，本来の Naive 法は，制約階層を導入した制約論理プログラミングで使用するために，与えられた制約階層の全ての解を求めていたが，以下では一つの解を求めることに限定して論じる．

Naive 法は基本的に，UCB 比較子に基づいてより良い解候補を探索する反復的なアルゴリズムである．探索時には，これまでに見つかった解候補よりも良い解候補を一つ求め，そのような解候補が存在する間は反復を続ける．UCB による比較は，制約が満たされているどうかの組合せに基づくため，この反復によっ

て必ず上限に達し，その上限が制約階層の解の一つとなる．

Naive 法は制約階層 H を次の問題 $P(H)$ に変換して扱う．

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

ただし，各 $s_{k,i}$ はセレクタ変数と呼ばれる，0 または 1 の値を取る変数である．$P(H)$ は，$s_{k,i}$ が 1 であるとき，$c_{k,i}$ が満たされなければならないことを示す．このため，セレクタ変数の値を変えて $P(H)$ を解くことで，制約階層 H の様々な解候補を得ることができる．

解候補の比較には，UCB 比較子に対応する次の関係 better を用いる †2．

$$\begin{aligned}&\mathrm{better}(S,T) \equiv \\ &\quad \exists k \in \{1,\ldots,l\}\left(\forall k' \in \{1,\ldots,k-1\}\right. \\ &\quad \left.\left(\sum_{i=1}^{m_{k'}} s_{k',i} = t_{k'}\right) \wedge \sum_{i=1}^{m_k} s_{k,i} > t_k\right)\end{aligned}$$

ただし，$S = (s_{1,1},\ldots,s_{1,m_1},\ldots,s_{l,1},\ldots,s_{l,m_l})$，$T = (t_1, t_2, \ldots, t_l)$ とする．S の各成分はセレクタ変数であり，T の各成分は現在の解候補に対応するセレクタ変数の値の和をレベルごとに求めたものである (S の値は未知，T の値は既知である)．T は現在の解候補の良さを示す指標と見なすことができ，以下ではこれを充足度と呼ぶ．Naive 法では，外部ソルバによって $P(H) \wedge \mathrm{better}(S,T)$ を解き，それが解を持つ場合は S の計算結果を用いて充足度 T の値を更新する処理を繰り返すことで，より良い解候補を求める．

3.3 二分探索による最適化

最適化問題を解くために二分探索を用いることがあり，特に充足可能性 (satisfiability; SAT) 問題を緩和して，充足される節を最大化する MaxSAT を実現するために用いられる [7]．具体的には，充足すべき節の個数を加算器によって符号化し，さらにその値の範囲を符号化した SAT 問題を考え，それが充足可能かどうか判定する処理を繰り返して，その上限と下限を二分探索で徐々に狭めることによって，充足される節を最大化する．本研究で外部ソルバとして使用する Scarab もこの方法を利用することで，最適化問題を解くことができる [21]．

†2 UCB 比較子は，各レベルで満たされている制約の個数を並べたベクトルに関する辞書式順序による比較と見なせる．

4 提案手法

本節では，3.2 節に述べた Naive 法を基礎として，3.3 節に述べた二分探索による最適化によって解を探索する，有限領域上の制約階層を解消するための以下の三つの手法を提案する．

Weighting 法： 制約階層を重み付き CSP (weighted CSP; WCSP) [18] に変換した上でその目的関数を最適化する

Lexicographic 法： Naive 法に二分探索を直接組み込む．

LevelWise 法： 強いレベルから順に最適化を行う．

4.1 Weighting 法

Weighting 法は，制約階層を WCSP に変換した上でその目的関数を二分探索によって最適化する．このためにまず，Naive 法と同様に制約階層 H を問題 $P(H)$ に変換する．一方，UCB 比較子は，セレクタ変数に関する次の目的関数 $f(S)$ によって扱う．

$$f(S) = \sum_{k=1}^{l}\left(w_k \sum_{i=1}^{m_k} s_{k,i}\right)$$

ただし，各 w_k は強さ k の制約の重みであり，次の通り定める．

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

重み w_k をこのように十分大きい値に定めることで，強い制約が弱い制約より優先されることを保証できる [19]．

$f(S)$ の下限は 0 (選好制約が全て満たされない場合) であり，上限は $(\prod_{k=1}^{l}(m_k + 1)) - 1$ (選好制約が全て満たされる場合) である．このため，これらの下限と上限を出発点とする二分探索によって，$P(H)$ に関して $f(S)$ を最大化すれば，制約階層 H の解が求められる．

4.2 Lexicographic 法

Lexicographic 法は，Naive 法に二分探索を直接組み込む．Naive 法では，現在の解候補を用いて充足度 T の値を徐々に更新していた．一方，Lexicographic 法では，二分探索中の現在の下限と上限の中間値を T に対して割り当てて $P(H) \wedge \mathrm{better}(S, T)$ を解き，その充足可能性に応じて二分探索の分岐を行う．二分探索開始時の T の下限は $(0, 0, \ldots, 0)$ (選好制約が全て満たされない) であり，上限は $(m_1, m_2, \ldots, m_k)$ (選好制約が全て満たされる) である．

T に割り当てるべき中間値 $T^{\mathrm{M}} = (t_1^{\mathrm{M}}, t_2^{\mathrm{M}}, \ldots, t_l^{\mathrm{M}})$ は，現在の下限と上限をそれぞれ $T^{\mathrm{L}} = (t_1^{\mathrm{L}}, t_2^{\mathrm{L}}, \ldots, t_l^{\mathrm{L}})$, $T^{\mathrm{U}} = (t_1^{\mathrm{U}}, t_2^{\mathrm{U}}, \ldots, t_l^{\mathrm{U}})$ としたとき，次の通り定める．

$$T^{\mathrm{M}} = \mathrm{uncomb}\left(\left\lceil (\mathrm{comb}(T^{\mathrm{L}}) + \mathrm{comb}(T^{\mathrm{U}}))/2 \right\rceil\right)$$

ただし，comb は

$$\mathrm{comb}(T) = \sum_{k=1}^{l} \left(\left(\prod_{k'=k+1}^{l} (m_{k'} + 1) \right) t_k \right)$$

として定められる関数であり，uncomb は comb の逆関数である．

4.3 LevelWise 法

LevelWise 法は，強いレベルから順に二分探索による最適化を行う．これは，制約階層を強いレベルから順に最大充足問題 (maximal CSP; MaxCSP) [18] に変換して解くことを繰り返すものとも見なせる．制約階層を強いレベルから順に解く方式は精製法と呼ばれ，制約階層の基本的な解消方式として知られている [9] [14]．

LevelWise 法のアルゴリズムを以下に示す．

1. $P \leftarrow \bigwedge_{i=1}^{m_0} c_{0,i}$ とする．
2. 1 から l までの k に対して以下を繰り返す．
   (a) $P \leftarrow P \wedge \left(\bigwedge_{i=1}^{m_k} (s_{k,i} = 1 \rightarrow c_{k,i})\right)$ とする．
   (b) 下限と上限の初期値をそれぞれ 0, m_k とする二分探索で，P に関して $\sum_{i=1}^{m_k} s_{k,i}$ を最大化し，その最適値を t_k とする．
   (c) $P \leftarrow P \wedge \left(\sum_{i=1}^{m_k} s_{k,i} = t_k\right)$ とする．

5 実装

前節の提案手法に基づき，制約階層のソルバを Scala 言語で実装した．本ソルバは，前節で提案した三つの手法の実装に加えて，Naive 法の再実装も含んでいる．制約階層の解消に必要となる外部ソルバとして，SAT 型 CP システム Scarab [21] [23] を使用した．Scarab で制約を SAT 問題に変換する際の符号化手法としては，Naive 法，Lexicographic 法，LevelWise 法では順序符号化のみを，Weighting 法では順序符号化または対数符号化のいずれかを使用できるようにした．本制約階層ソルバのプログラムの長さは合計で約 1000 行である．

6 実験

本研究の提案手法の性能評価のために，前節に述べた制約階層ソルバを用いて実験を行った．実験では，2.4 GHz Core i9-9980HK と 16 GB のメモリを搭載し，macOS 11.6.8 で動作するコンピュータ上でプログラムを実行した．使用した外部ソルバ，言語処理系は，Scarab 1.9.6，Scala 2.12.16，AdoptOpenJDK 11.0.16+8 である．

性能評価に用いた問題は，汎対角ラテン方陣の計算を制約階層で記述したものである．汎対角ラテン方陣は，$n \times n$ のマス目上で縦，横，斜め方向に 1 から n までの異なる数字が並ぶように配置したものである．斜め方向については，対角線上だけでなく，一般化された斜め方向に対しても異なる数字が並ぶようにされている点で，通常の対角ラテン方陣と異なる．$n \times n$ 汎対角ラテン方陣を求める問題は，$n = 5, 7, 11, 13$ に対して充足可能であり，$n = 3, 4, 6, 8, 9, 10, 12, 14, 15, 16$ に対して充足不能である [21]．

本実験では，制約階層を用いて汎対角ラテン方陣を求める問題を考え，以下ではこれをソフト汎対角ラテン方陣と呼ぶ．具体的には，必須制約を使わず，横，縦，右下がり斜め，右上がり斜め方向に異なる数字を並べる制約をそれぞれ強さ 1, 2, 3, 4 で記述する．これによって，通常の汎対角ラテン方陣では充足不能である場合でも，数字の配置が得られる．図 1(a), (b) にそれぞれ $n = 5, 6$ のソフト汎対角ラテン方陣の解

を示す．$n = 5$ に対しては通常の汎対角ラテン方陣でも充足可能であるため，図 1(a) では全ての方向で異なる数字が並んでいる．一方，$n = 6$ の解である図 1(b) では，解の充足度が $(6, 6, 4, 0)$ であるため，横，縦方向には異なる数字が並んでいるが，右下がり斜め方向では異なる数字が並んでいるのは 4 列だけであり，右上がり斜め方向では異なる数字が並ぶ列が存在しない．

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

(a)

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

(b)

図 1　(a) $n = 5$ と (b) $n = 6$ に対する $n \times n$ ソフト汎対角ラテン方陣の解

本実験では，$3 \leq n \leq 16$ に対する $n \times n$ ソフト汎対角ラテン方陣を解く際の実行時間を計測した．実行時に，Java 仮想マシンに 2 GB のメモリを割り当て，実行時間を最長で 20 分間に制限した．$n = 3, 4, 5, 6, 7, 8, 10, 11, 13$ に対する結果を表 1 に示す．Weighting/Order, Weighting/Log は，Weighting 法で Scarab による SAT 問題への変換の際にそれぞれ順序符号化，対数符号化を用いた場合を示す．表中で M.O. は，実行中に OutOfMemoryError で異常終了したことを，T.O. は，制限時間内に計算が終了しなかったことを示す．$n = 9, 12, 14, 15, 16$ に対しては制限時間内に正常終了した場合がなかったため，表に記載していない．計算が正常終了した場合の解の充足度は複数の手法で一致した．特に通常の汎対角ラテン方陣で充足可能である $n = 5, 7, 11, 13$ に対しては全ての制約が満たされた．

実験の結果，サイズの小さい $n = 3, 4$ の問題に対しては，Naive 法が最短時間で解を求めた．一方，より大きい $n \geq 5$ の問題に対しては，Lexicographic 法または LevelWise 法のいずれかが最短時間で解を求めた．Weighting 法に関しては，順序符号化を用いた場合には計算時間が長かっただけでなく，$n \geq 6$ でメモリ不足による異常終了を起こした．対数符号化を用いた場合にはメモリ不足にならなかったが，$n \geq 8$ では制限時間内に計算が終了しなかった．

7　議論

本研究の実験で，Lexicographic 法と LevelWise 法は，問題のサイズが大きくなっても，いずれかの手法がより高性能であるという傾向は見られなかった．しかし，内部で行う最適化は大きく異なっているため，単純に同程度の性能であると結論することはできない．本実験の範囲からは明らかにならなかったが，それぞれの手法に適した状況や適さない状況がある可能性もあるため，内部処理の詳細な比較や，他の問題を用いた比較を行う必要がある．

Weighting 法は，順序符号化，対数符号化のいずれを用いた場合にも，他の手法に比べて良い性能を示さなかった．Weighting 法は WCSP への変換に基づく簡明な手法であるが，目的関数の範囲が大きくなるため，有限領域を対象とした外部ソルバの利用にはあまり適していないと言える．しかし，変数の領域のサイズに応じて順序符号化と対数符号化を自動的に切り替えるハイブリッド符号化[22] のように，本研究の実験で用いなかった SAT 問題への符号化手法も存在する．このため，適切な符号化手法を採用することで，Weighting 法の性能を改善できる可能性もある．

本研究で提案した三つの手法は，制約階層の解を一つ求めるものであった．しかし，元の Naive 法と同様の方法で，制約階層の解を全て求めるように拡張することは容易である．これによって，本研究で提案した手法を，制約階層を導入した制約論理プログラミングのために使用することも可能である．

8　おわりに

本研究では，有限領域上の制約階層を解消するための手法として，Weighting 法，Lexicographic 法，LevelWise 法の三つを提案した．これらは，著者らが過去に提案した Naive 法を基礎として，二分探索による最適化を導入したものである．本研究ではさらに，これらの手法の実装を示し，性能を評価するために行った実験の結果を与えた．

表 1　$n \times n$ ソフト汎対角ラテン方陣の解消に要した時間 (ミリ秒)

n	3	4	5	6	7	8	10	11	13
解の充足度	(3,3,3,0)	(4,4,2,2)	(5,5,5,5)	(6,6,4,0)	(7,7,7,7)	(8,8,6,6)	(10,10,8,8)	(11,11,11,11)	(13,13,13,13)
Naive	**245**	**315**	383	3563	650	3912	T.O.	T.O.	T.O.
Weighting/Order	438	2805	8328	M.O.	M.O.	M.O.	M.O.	M.O.	M.O.
Weighting/Log	342	583	458	68633	5184	T.O.	T.O.	T.O.	T.O.
Lexicographic	274	387	352	**3519**	**488**	2950	179162	**5826**	139548
LevelWise	270	324	**343**	7173	**488**	**2571**	**150286**	75071	**31859**

謝辞　本研究は JST AIP ネットワークラボ「日独仏 AI 研究」「人工知能の実時間規範遵守機構の研究」(JPMJCR20G4) の支援を受けたものである.

参 考 文 献

[1] Apple, Inc.: *Auto Layout Guide*, 2011.
[2] Badros, G. J., Borning, A., and Stuckey, P. J.: The Cassowary Linear Arithmetic Constraint Solving Algorithm, *ACM Trans. Comput.-Human Interact.*, Vol. 8, No. 4 (2001), pp. 267–306.
[3] Borning, A., Freeman-Benson, B., and Wilson, M.: Constraint Hierarchies, *Lisp Symbolic Comput.*, Vol. 5, No. 3 (1992), pp. 223–270.
[4] de Moura, L. and Bjørner, N.: Z3: An Efficient SMT Solver, *Proc. TACAS*, LNCS, Vol. 4963, 2008, pp. 337–340.
[5] Freeman-Benson, B. N., Maloney, J., and Borning, A.: An Incremental Constraint Solver, *Comm. ACM*, Vol. 33, No. 1 (1990), pp. 54–63.
[6] Freuder, E. C. and Mackworth, A. K.: Constraint Satisfaction: An Emerging Paradigm, *Handbook of Constraint Programming*, Elsevier, 2006, chapter 2, pp. 13–27.
[7] Fu, Z. and Malik, S.: On Solving the Partial MAX-SAT Problem, *Proc. SAT*, LNCS, Vol. 4121, 2006, pp. 252–265.
[8] Hosobe, H.: A Scalable Linear Constraint Solver for User Interface Construction, *Proc. CP*, LNCS, Vol. 1894, 2000, pp. 218–232.
[9] 細部博史: ユーザーインターフェースにおける制約解消法の研究動向, コンピュータソフトウェア, Vol. 17, No. 6 (2000), pp. 73–85.
[10] Hosobe, H.: A Modular Geometric Constraint Solver for User Interface Applications, *Proc. ACM UIST*, 2001, pp. 91–100.
[11] Hosobe, H.: Hierarchical Nonlinear Constraint Satisfaction, *Proc. ACM SAC*, 2004, pp. 16–20.
[12] Hosobe, H.: A Hierarchical Method for Solving Soft Nonlinear Constraints, *Proc. SCSE*, Procedia CS, Vol. 62, 2015, pp. 378–384.
[13] Hosobe, H.: Solving Hierarchical Soft Constraints with an SMT Solver, *Proc. ICCAE*, ACM, 2020, pp. 42–46.
[14] Hosobe, H. and Matsuoka, S.: A Foundation of Solution Methods for Constraint Hierarchies, *Constraints*, Vol. 8, No. 1 (2003), pp. 41–59.
[15] 細部博史, 佐藤健: 階層的制約論理プログラミングのための制約階層解消法, 人工知能学会全国大会論文集, No. 4F3-OS-8b-04, 2022, pp. 1–4.
[16] Hurst, N., Marriott, K., and Moulder, P.: Dynamic Approximation of Complex Graphical Constraints by Linear Constraints, *Proc. ACM UIST*, 2002, pp. 191–200.
[17] Marriott, K. and Chok, S. S.: QOCA: A Constraint Solving Toolkit for Interactive Graphical Applications, *Constraints*, Vol. 7, No. 3–4 (2002), pp. 229–254.
[18] Meseguer, P., Rossi, F., and Schiex, T.: Soft Constraints, *Handbook of Constraint Programming*, Elsevier, 2006, chapter 9, pp. 281–328.
[19] 岡本英紀, 佐藤健: 0-1 整数計画法を用いた優先順位付き極小限定における極小モデルの計算, 人工知能学会誌, Vol. 15, No. 3 (2000), pp. 511–517.
[20] Rossi, F., van Beek, P., and Walsh, T.: Introduction, *Handbook of Constraint Programming*, Elsevier, 2006, chapter 1, pp. 3–12.
[21] 宋剛秀, 番原睦則, 田村直之: Scala 上で実現された SAT 型制約プログラミング システムのための開発ツール, 日本ソフトウエア科学会大会論文集, No. SOFT1-1, 2014, pp. 1–15.
[22] Soh, T., Banbara, M., and Tamura, N.: Proposal and Evaluation of Hybrid Encoding of CSP to SAT Integrating Order and Log Encodings, *Intl. J. Artif. Intell. Tools*, Vol. 26, No. 1 (2017), pp. 1–29.
[23] Soh, T., Tamura, N., and Banbara, M.: Scarab: A Rapid Prototyping Tool for SAT-Based Constraint Programming Systems, *Proc. SAT*, LNCS, Vol. 7962, 2013, pp. 429–436.
[24] Tsang, E.: *Foundations of Constraint Satisfaction*, Academic Press, 1993.