SAT 型制約ソルバーを用いたナンバーリンクの解法

田村直之 [1)]，宋剛秀 [1)]，番原睦則 [1)]，鍋島英知 [2)]
1) 神戸大学情報基盤センター
2) 山梨大学医学工学総合研究部

制約充足問題 (CSP) は与えられた制約を満たす解を探索する問題であり，多くの組合せ問題は CSP として定式化できる．SAT 型制約ソルバーは，CSP を命題論理の充足可能性判定問題 (SAT) に符号化し，SAT ソルバーを用いて探索することにより，CSP の解を求めるプログラムである．ここでは，国際的な競技会で優秀な成績を収めている制約ソルバー Sugar および SAT ソルバー GlueMiniSat を用い，SAT 型制約ソルバーでナンバーリンク問題の高速な求解が可能であることを示す．また，制約記述には Scala 上の制約プログラミングシステムである Copris を用いる．これにより，提案するシステムは高い拡張性も実現している．

Solving Numberlink by a SAT-based Constraint Solver

Naoyuki Tamura[1)], Takehide Soh[1)], Mutsunori Banbara[1)], Hidetomo Nabeshima[2)]
1) Information Science and Technology Center, Kobe University
2) Department of Research Interdisciplinary Graduate School of Medicine and Engineering, Yamanashi University

Constraint Satisfaction Prolbem (CSP) is a problem to find a solution satisfying all given constraints, and many combinatorial problems can be formalized as a CSP. SAT-based constraint solver is a program to find a solution of a given CSP by encoding it to a Boolean satisfiability testing program (SAT) and searching a solution with a SAT solver. In this paper, we show a SAT-based constraint solver can efficiently solve Numberlink puzzles with Sugar and GlueMiniSat solvers which performed well at the international solver competitions. A constraint programming system Copris on a Scala programming language is used for a constraint modeling. Therefore, the proposed Numberlink solver also realizes high extensibility.

1 はじめに

命題論理の充足可能性判定問題 (**SAT**) は，与えられた命題論理式を真にする値割り当てが存在するか否かを判定する問題である [2, 5, 16, 3]．近年になって，SAT を解くための非常に高速な **SAT ソルバー** が実現され [9]，ハードウェア検証，ソフトウェア検証，スケジューリング，制約充足等の問題について，それらを SAT に変換 (**SAT 符号化**; SAT encoding) した後，SAT ソルバーに解かせることにより，元の問題に対する求解を実現する **SAT 型システム** の研究が注目を集めている．

一方 **制約充足問題** (CSP; Constraint Satisfaction Problem) は与えられた制約を満たす解を探索する問題である [10, 3]．人工知能分野等で生じる多くの組合せ問題は，制約充足問題 (あるいは制約最適化問題) として定式化できる．**制約プログラミング** システムは CSP を取り扱うシステムであり，CSP を記述するための**制約モデリング言語** と，CSP の解を探索する **制約ソルバー** とからなる．ま

た **SAT 型制約ソルバー** は，何らかの制約モデリング言語で記述された CSP を，SAT に符号化し求解するシステムである [14].

本稿では，国際的な競技会で優秀な成績を収めている SAT 型制約ソルバー Sugar および SAT ソルバー GlueMiniSat を用いることで，ナンバーリンク問題[*1] の高速な求解が可能であることを示す．また，制約記述には Scala 上の制約プログラミングシステムである Copris を用いる [15]．これにより，提案するシステムは高い拡張性も実現している．

2 Sugar と Copris

SAT 型制約ソルバー Sugar [*2] は，**順序符号化** [11] と呼ばれる方法を用いて CSP を SAT に符号化し，外部の SAT ソルバーを用いて求解するシステムである [12, 13]．2008, 2009 年の国際制約ソルバー競技会のグローバル制約部門で優勝するなど，他の制約ソルバーに匹敵する性能を示している．

Copris [*3]は，Scala [*4] プログラミング言語上に実現された制約プログラミング用 DSL (Domain-Specific Language) である [15]．Scala 起動時に，Copris 用 jar ファイルをクラスパス中に指定するだけで，簡潔な制約記述が可能になる．また，Copris では複数のソルバーをバックエンドとして利用可能である．GlueMiniSat 等の SAT ソルバーはもちろん，Z3 などの SMT ソルバー，Choco などの制約ソルバーを利用できる．

図 1 に Copris のプログラム例を示す．最初の 2 行は，Copris パッケージ中のクラスや関数をインポートする命令である．3-5 行目で整数変数 x, y, z を宣言している．それぞれ 1 から 15 の値を取る．6-7 行目は制約 $x+y+z=5$ および $x+5y+10z=90$ を追加している．演算子定義により，自然な記法が利用できている．8 行目では，バックエンドのソルバーとして Sugar および GlueMiniSat の組合せを指定している．9 行目で `find` 関数により解を探索し，9 行目で解が出力される．`find` 関数の内部では，定義された CSP を Sugar を用いて SAT に符号化し，GlueMiniSat を用いて解を探索している．

```
 1: import jp.kobe_u.copris._
 2: import jp.kobe_u.copris.dsl._
 3: int('x, 1, 15)
 4: int('y, 1, 15)
 5: int('z, 1, 15)
 6: add('x + 'y + 'z === 15)
 7: add('x + 'y * 5 + 'z * 10 === 90)
 8: use(new sugar.Solver(csp, sugar.GlueMiniSat))
 9: if (find)
10:   println(solution)
```

図 1　Copris のプログラム例 (左端は行番号)

3 GlueMiniSat

GlueMiniSat は，与えられた SAT 問題の求解中に，その問題の簡単化を頻繁に行う軽量なアルゴリズム [8] を実装した SAT ソルバーである．代表的な SAT ソルバーである MiniSat をベースに，リテラルブロック距離 [1] と呼ばれる学習節の評価尺度に基づく節削減戦略を導入していることも特徴の 1 つである [7]．GlueMiniSat は，2011, 2013 年に開催された国際 SAT 競技会における産業応用部門の UNSAT トラックにおいて，それぞれ優勝，準優勝するなどの成果を収めている．

多くの SAT ソルバーと同様に GlueMiniSat は，連言標準形の命題論理式を入力として受け取り，その充足可能性を判定し，充足可能の場合は論理式を充足する値割り当てを返す．入力は DIMACS 形式[*5]で記述されたテキストファイルであり，出力は充足可能であれば各命題の値割り当てをテキスト形式でファイルに出力する．GlueMiniSat は C++で記述されており，テキストファイルを介さずに C++ の関数を通して直接入出力を行うことも可能であるが，現状では Copris との連携ではテキストファイルを経由して SAT 問題とその解のやりとりを行っている．

また，インクリメンタル SAT 解法 (incremental SAT solving)[4] に対応しており，例えば，充足可能な値割り当てを複数列挙するために発見した割り

[*1] ナンバーリンクはニコリ社によるパズルである (http://www.nikoli.co.jp).
[*2] http://bach.istc.kobe-u.ac.jp/sugar/
[*3] http://bach.istc.kobe-u.ac.jp/copris/
[*4] オブジェクト指向と関数型の両パラダイムを融合したプログラミング言語 (http://www.scala-lang.org)
[*5] http://www.satlib.org/Benchmarks/SAT/satformat.ps

	1	2	3	4	
	2	1	4	3	

図 2　ナンバーリンクの問題例

当てを否定する式を追加しながら繰り返し解くような場合に，学習節を再利用しながら求解することで探索の重複を抑制することが可能である．

4　ナンバーリンクの制約モデル

ニコリによるナンバーリンクのルールは以下の通りである．

1. 白マスに線を引いて，同じ数字どうしをつなげましょう．
2. 線は，マスの中央を通るようにタテヨコに引きます．線を交差させたり，枝分かれさせたりしてはいけません．
3. 数字の入っているマスを通過するように線を引いてはいけません．
4. 1 マスに 2 本以上の線を引いてはいけません．

図 2 にナンバーリンクの問題例を示す．

与えられたナンバーリンク問題の行数を m, 列数を n とする．行番号, 列番号は 0 から始まるとし，i 行目 j 列目のマスの位置を (i,j) で表す．位置の間には自然な順序を導入する．マス (i,j) の上下左右に隣接するマスの位置の集合を $adj((i,j))$ で表す．例えば $adj((2,4)) = \{(1,4),(2,3),(3,4),(2,5)\}$ である．

問題中に現れている数字の集合を N で表す．各数字は 2 ヶ所に現れる．それら 2 ヶ所を区別し一方を始点，他方を終点と考える．始点の位置の集合を S, 終点の位置の集合を T で表す．例えば図 2 の場合，$S = \{(1,1),(1,2),(1,3),(1,4)\}$, $T = \{(3,1),(3,2),(3,3),(3,4)\}$ となる．また，位置 (i,j) が数字マスの場合にその値を $num((i,j))$ で表す．最後に，白マスの位置の集合を B で表す．

4.1　基本モデル

ナンバーリンクの問題は，与えられたグラフに対して，ある条件を満たす部分グラフを求める問題と考えるとわかりやすい．つまり各マスをグラフの頂点とし，各マスと上下左右に隣接したマスの間に辺あるいは弧があると考える．無向グラフとしても有向グラフとしても定式化できるが，ここでは有向グラフとして定式化する．つまりマスを通る線に向きがあると考える．一見面倒になるようだが，無向グラフに比べ後述の次数の制約などが単純になり，より効率良く解ける SAT 問題に符号化できる．

有向グラフ $D(V,A)$ の頂点集合 V と弧集合 A は以下のように与えられる．

$$V = \{(i,j) \mid 0 \le i < m,\ 0 \le j < n\}$$
$$A = \{(u,v) \mid u \in V,\ v \in adj(u)\}$$

また，線で結ばれたマスには同一の数字が入るとし，各頂点に数字をラベル付けする関数 $f : V \to N$ を考える．

では，これらの元で制約モデルを考えよう．まず，各弧 $(u,v) \in A$ に対し 0-1 変数 $a_{u,v}$ を導入する．$a_{u,v} = 1$ は u から v の向きに線が引かれていることを表す．反対向きに同時に線が引かれることはないため $a_{u,v}$ と $a_{v,u}$ は同時に 1 にはならない．

$$a_{u,v} \in \{0,1\} \quad (\forall (u,v) \in A) \tag{1}$$
$$a_{u,v} + a_{v,u} \le 1 \quad (\forall (u,v) \in A,\ u < v) \tag{2}$$

次に，各頂点 $u \in V$ における入出次数を考える．頂点 $u \in S$ の出次数は 1，入次数は 0 になり，頂点 $u \in T$ の場合はその逆になる．白マスの出次数と入次数は等しく，0 か 1 である．

$$\textstyle\sum_{v \in adj(u)} a_{uv} = 1 \quad (\forall u \in S) \tag{3}$$
$$\textstyle\sum_{v \in adj(u)} a_{vu} = 0 \quad (\forall u \in S) \tag{4}$$
$$\textstyle\sum_{v \in adj(u)} a_{uv} = 0 \quad (\forall u \in T) \tag{5}$$
$$\textstyle\sum_{v \in adj(u)} a_{vu} = 1 \quad (\forall u \in T) \tag{6}$$
$$d_u \in \{0,1\} \quad (\forall u \in B) \tag{7}$$
$$\textstyle\sum_{v \in adj(u)} a_{uv} = d_u \quad (\forall u \in B) \tag{8}$$
$$\textstyle\sum_{v \in adj(u)} a_{vu} = d_u \quad (\forall u \in B) \tag{9}$$

最後に，各頂点 $u \in V$ に対しラベル付けされた数字を表す変数 f_u を導入する．数字の入っている

頂点 u について f_u の値はその数字に等しくなければならない．また線で結ばれている頂点間のラベルは同一でなければならない．

$$f_u \in N \quad (\forall u \in V) \tag{10}$$
$$f_u = num(u) \quad (\forall u \in S \cup T) \tag{11}$$
$$a_{uv} = 1 \Rightarrow f_u = f_v \quad (\forall (u,v) \in A) \tag{12}$$

これらにより，ナンバーリンクの制約モデルが得られた．しかし，この制約モデルの解がそのままナンバーリンクの解となるわけではない．たとえば 2 行，3 列で $(0,0)$ と $(1,0)$ に数字 1 がある問題を考える．この問題に対する制約モデルの解は，図 3 の (a)–(d) の 4 通りが得られる．(a)–(c) はナンバーリンクの解に対応しているが，(d) は余分なサイクルを含んでおりナンバーリンクの解とは見なせない．

余分なサイクルを除去するための制約を追加することは可能だが，かなり複雑なものとなり，求解速度も低下する恐れがある．代わりに，制約モデルの解からナンバーリンクの解を抽出する際に余分なサイクルを除去すれば良いので，サイクル除去の制約は追加しないことにした．

4.2 改良モデル

基本モデルでは，マスを結ぶ弧 (有向辺) を考えている．そこで，ある領域から出る (あるいは入る) 弧の個数に関する条件を加えることを考える．

ここでは $m' = \lfloor m/2 \rfloor, n' = \lfloor n/2 \rfloor$ として，$m'-1$ 行目と m 行目の間，$n'-1$ 列目と n 列目の間で盤面を分割し，R_1 (左上)，R_2 (右上)，R_3 (左下)，R_4 (右下) の 4 領域を考える．

$$R_1 = \{(i,j) \mid 0 \le i < m',\ 0 \le j < n'\}$$
$$R_2 = \{(i,j) \mid 0 \le i < m',\ n' \le j < n\}$$
$$R_3 = \{(i,j) \mid m' \le i < m,\ 0 \le j < n'\}$$
$$R_4 = \{(i,j) \mid m' \le i < m,\ n' \le j < n\}$$

領域から出て行く弧の個数から入ってくる弧の引いた値を，領域の出次数と定義する．

各領域の出次数 (出て行く弧の個数から入ってくる弧の個数を引いた値) を表すために，以下の整数変数 s_{12}, s_{34}, s_{13}, s_{24} を導入する．各 s_{kl} は 領域 R_k から領域 R_l への出次数を表している．

$$s_{12} \in \{-m', \ldots, m'\} \tag{13}$$
$$s_{12} = \sum_{i=0}^{m'-1} (a_{(i,n'-1)(i,n')} - a_{(i,n')(i,n'-1)}) \tag{14}$$
$$s_{34} \in \{m'-m, \ldots, m-m'\} \tag{15}$$
$$s_{34} = \sum_{i=m'}^{m-1} (a_{(i,n'-1)(i,n')} - a_{(i,n')(i,n'-1)}) \tag{16}$$
$$s_{13} \in \{-n', \ldots, n'\} \tag{17}$$
$$s_{13} = \sum_{j=0}^{n'-1} (a_{(m'-1,j)(m',j)} - a_{(m',j)(m'-1,j)}) \tag{18}$$
$$s_{24} \in \{n'-n, \ldots, n-n'\} \tag{19}$$
$$s_{24} = \sum_{j=n'}^{n-1} (a_{(m'-1,j)(m',j)} - a_{(m',j)(m'-1,j)}) \tag{20}$$

たとえば図 2 の問題例の場合，$s_{12} = (a_{(0,2)(0,3)} - a_{(0,3)(0,2)}) + (a_{(1,2)(1,3)} - a_{(1,3)(1,2)})$ である．

領域 R_1 から他の領域への出次数は $s_{12} + s_{13}$ であり，この値は領域 R_1 中にある始点の個数から終点の個数を引いたものに等しい．他の領域についても同様に考えると，以下の制約が得られる．

$$s_{12} + s_{13} = |R_1 \cap S| - |R_1 \cap T| \tag{21}$$
$$-s_{12} + s_{24} = |R_2 \cap S| - |R_2 \cap T| \tag{22}$$
$$-s_{13} + s_{34} = |R_3 \cap S| - |R_3 \cap T| \tag{23}$$
$$-s_{24} - s_{34} = |R_4 \cap S| - |R_4 \cap T| \tag{24}$$

たとえば図 2 の問題例で領域 R_1 について考えた場合，$(1,1)$ と $(1,2)$ の 2 ヶ所が始点であり，終点は存在しないので制約は $s_{12} + s_{13} = 2$ となる．同様に R_2 について $-s_{12} + s_{24} = 2$, R_3 について $-s_{13} + s_{34} = -2$, R_4 について $-s_{24} - s_{34} = -2$ が得られる．

5 探索方法の改善

ここまでで述べた制約モデルを Copris で記述し，Sugar で SAT 符号化，GlueMiniSat で求解した所，ほとんどの問題は数秒程度で解けるが，小さな問題でも数百秒を必要とする場合があることがわかった．

そこで，以下の 2 種類の追加制約を考えた．

- (追加制約 1) 回り道をしない．

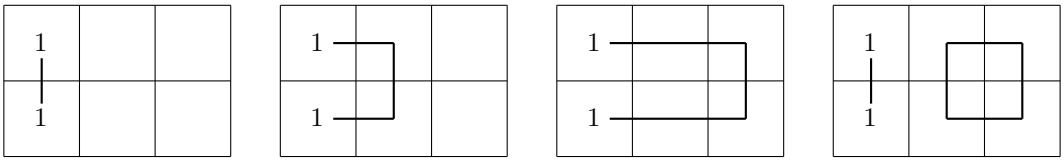
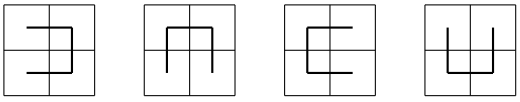


図 3　制約モデルから得られる 4 通りの解

- (追加制約 2) すべてのマスを通る.

追加制約 1 は，以下の 4 種類のコの字状の線が含まれないことを意味する.



マス u の右のマスを u', 下のマスを u'', u' の下のマス (u'' の右のマス) を u''' で表し，$a_{uv} + a_{vu}$ を e_{uv} で表すと，追加制約 1 は以下のように書ける.

$$e_{uu'} + e_{uu''} + e_{u'u'''} + e_{u''u'''} \leq 2 \quad (\forall u \in V,\ \{u', u'', u'''\} \subset V) \tag{25}$$

追加制約 2 は以下のように書ける[*6].

$$d_u = 1 \quad (\forall u \in B) \tag{26}$$

どちらの追加制約も，いわゆる関西解と呼ばれるものを排除する条件であり，ナンバーリンクのルールに添った解をもらしてしまう可能性がある．例えば，追加制約 1 を加えた場合，図 3 中では (a) のみが解として得られる．追加制約 2 だと (c) と (d) が得られ，両方を加えると解は見つからなくなる．特に，追加制約 2 はナンバーリンクのルールに沿った解が存在しても，全く解を発見できない可能性がある．一方，追加制約 1 はナンバーリンクのルールに沿った解が存在すれば，いくつかを漏らす可能性はあるが，必ず解を発見できる.

6　性能評価

まず http://www.janko.at/Raetsel/Arukone/ で公開されている 280 問について，追加制約を加えない基本モデルで最初の 5 個の解探索が 300 秒以内に終了するかどうかを調べた所，048 , 110, 127, 158, 160, 190 の 6 問が 300 秒以内に終了しなかった．なお計測は，Intel Xeon 3.46GHz の計算機上で Sugar version 2.2.0, GlueMiniSat version 2.2.8 を使用して行った.

表 1　Copris+GlueMiniSat による全解探索時間 (秒)

問題番号	048	110	127	158	160	190
列数	15	42	15	12	12	48
行数	15	25	15	12	12	35
基本	551	703	–	335	759	–
改良	35	49	–	187	283	–
基本 +1	36	47	–	26	178	–
改良 +1	49	23	–	30	140	–
基本 +2	37	175	–	36	167	–
改良 +2	70	25	–	24	60	–
基本 +1+2	6	16	6	4	4	34
改良 +1+2	6	16	7	4	4	30

これら 6 問について，制限時間を 1800 秒とし，基本モデルと改良モデル，2 種類の追加制約を加える場合と加えない場合で，最初の 5 個の解探索にかかる時間を調べた．結果を表 1 に示す．表中の “–” は制限時間内に探索が終了しなかったことを表す.

結果から，改良モデルにより性能がある程度向上すること，両方の追加制約を加えることで大幅に性能が向上することがわかった.

ただ，前述したように追加制約を加えると，すべてのマスを通らない解などを発見できない．したがって，最初は改良モデルに追加制約を加えた形で探索を行い，その後，追加制約なしの改良モデルで探索を行う方法などが考えられる.

7　おわりに

本稿では，SAT 型制約ソルバーである Sugar と SAT ソルバーである GlueMiniSat を用いて，ナン

[*6] 式 (7) を $d_u \in \{1\}$ に変更するのでも良い.

バーリンクを高速に解くための方法を示した．

今後の課題としては，incremental SAT 解法の導入，BDD や ZDD [6] を用いた方法 [17] との比較などが挙げられる．

参考文献

[1] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pp. 399–404, 2009.

[2] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, Vol. 185 of *Frontiers in Artificial Intelligence and Applications (FAIA)*. IOS Press, 2009.

[3] Lucas Bordeaux, Youssef Hamadi, and Lintao Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Surveys*, Vol. 38, No. 4, 2006.

[4] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, Vol. 89, No. 4, 2003.

[5] 井上克巳, 田村直之. SAT ソルバーの基礎. 人工知能学会誌, Vol. 25, No. 1, pp. 57–67, 2010.

[6] Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th International Design Automation Conference (DAC 1993)*, pp. 272–277, 1993.

[7] 鍋島英知, 岩沼宏治, 井上克巳. GlueMiniSat 2.2.5: 単位伝搬を促す学習節の積極的獲得戦略に基づく高速 SAT ソルバー. コンピュータ ソフトウェア, Vol. 29, No. 4, pp. 146–160, 2012.

[8] Hidetomo Nabeshima, Koji Iwanuma, and Katsumi Inoue. On-the-fly lazy clause simplification based on binary resolvents. In *ICTAI*, pp. 987–995. IEEE, 2013.

[9] 鍋島英知, 宋剛秀. 高速 SAT ソルバーの原理. 人工知能学会誌, Vol. 25, No. 1, pp. 68–76, 2010.

[10] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[11] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, Vol. 14, No. 2, pp. 254–272, 2009.

[12] 田村直之, 丹生智也, 番原睦則. SAT 変換に基づく制約ソルバーとその性能評価. コンピュータソフトウェア, Vol. 27, No. 4, pp. 183–196, 2010.

[13] Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara. Solving constraint satisfaction problems with SAT technology. In *Proceedings of the 10th International Symposium of Functional and Logic Programming (FLOPS 2010), LNCS 6009*, pp. 19–23, 2010.

[14] 田村直之, 丹生智也, 番原睦則. 制約最適化問題と SAT 符号化. 人工知能学会誌, Vol. 25, No. 1, pp. 77–85, 2010.

[15] 田村直之, 丹生智也, 番原睦則. Scala 上の制約プログラミング用ドメイン特化言語 Copris について. コンピュータソフトウェア, Vol. 29, No. 4, pp. 114–129, 2012.

[16] 梅村晃広. SAT ソルバ・SMT ソルバの技術と応用. コンピュータソフトウェア, Vol. 27, No. 3, pp. 24–35, 2010.

[17] Ryo Yoshinaka, Toshiki Saitoh, Jun Kawahara, Koji Tsuruma, Hiroaki Iwashita, and Shin-ichi Minato. Finding all solutions and instances of numberlink and slitherlink by ZDDs. *Algorithms*, Vol. 5, No. 2, pp. 176–213, 2012.