

SQL 型制約プログラミングシステム CombSQL+ の 複数制約ソルバー連携

Utilizing Multiple Constraint Solvers in an SQL-based Constraint Programming System CombSQL+

小菅 脩司^{1*} 酒井 正彦¹ 番原 睦則¹
Shuji Kosuge¹ Masahiko Sakai¹ Mutsunori Banbara¹

¹ 名古屋大学大学院情報学研究科

¹ Graduate School of Informatics, Nagoya University

Abstract: This paper presents an extension of an SQL-based constraint programming system CombSQL+, for utilizing multiple CSP solvers. The resulting system reads a problem instance expressed in database tables. In turn, an extended SQL query for problem-solving generates a constraint satisfaction problem (CSP), which can subsequently be solved by any off-the-shelf CSP solvers, in our case, a SAT-based CSP solver Sugar as well as an SMT solver Z3. Our declarative approach has obvious advantages. In particular, combinatorial (optimization) problems can be concisely modeled as extended SQL statements and then solved by general-purpose CSP solvers rather than dedicated implementations.

1 はじめに

制約プログラミング (Constraint Programming [3]) とは、解きたい問題を制約として宣言的にプログラム中に記述するプログラミングパラダイムである。制約プログラミングの言語は制約充足問題 (Constraint Satisfaction Problem; CSP [8]) とよばれる。近年、CSP を解くプログラムである CSP ソルバーの性能が大幅に向上し、AI 分野を中心に実用的な応用が拡大している。その一方で、CSP ソルバーを用いた問題解決は、問題の定式化およびプログラミングに関する高度な技能が求められるため、一般ユーザにとって敷居が高い。そのため、一般ユーザが解きたい問題を簡単に記述できる制約プログラミングシステムの研究開発は重要である。

データベース言語 SQL 上での制約プログラミングシステム CombSQL+ [4]¹ は、CSP を拡張 SQL 文で記述し、バックエンドの CSP ソルバーを用いて解を探索するシステムである。CombSQL+ の特長としては、CSP を SQL 文を使って簡潔に記述できる点、問題インスタンスと制約の記述を分離できる点、得られた解をデータベースファイルとして保持・提供できる点などが挙げられる。しかし、その一方で、CombSQL+ の実装はバックエンドの SMT ソルバー [2, 9] に強く依存しており、他の CSP ソルバーが利用できない点、CSP の

代表的なグローバル制約である alldifferent 制約 [1] に対応していない点など、改良の余地が残っている。

本稿では、CombSQL+ を複数異種の CSP ソルバーと連携可能にするための拡張について述べる。また、具体的な連携例として、CombSQL+ と SAT 型 CSP ソルバー Sugar [7, 6]² との連携方法を示す。ここで、Sugar は CSP を SAT に符号化し、高速 SAT ソルバーを用いて解くシステムであり、Sugar およびその継続ソルバーは、国際 CSP ソルバー競技会で優勝するなど優れた性能を示している。また、SAT 型 CSP ソルバーは、SMT ソルバーと比較して、CSP を解く性能が優れていることが実験的に示されており [5]、CombSQL+ の求解性能の向上が期待できる。

複数異種の CSP ソルバーと連携のため、バックエンドソルバーを管理する抽象クラス SolverWrapper を CombSQL+ に導入する。これにより、各 CSP ソルバーを SolverWrapper のサブクラスとして実装することができ、複数異種のソルバーとの連携が可能となる。次に、CombSQL+ と Sugar とのシームレスな連携を可能にするため、CSP ソルバー Sugar の Python インターフェース GlazeSugar の設計と実装を行う。なお、GlazeSugar は Sugar を CSP から SAT への符号化モジュールとして利用できるように設計されており、任意の高速 SAT ソルバーを利用できる。最後に、GlazeSugar の API を利用し Sugar を SolverWrapper のサブクラスとして簡単に実装できることを示す。

*連絡先：名古屋大学大学院情報学研究科
〒464-8601 名古屋市千種区不老町
E-mail: kosugesuji003@nagoya-u.jp

¹<https://git.trs.css.i.nagoya-u.ac.jp/combsql/combsqlplus/>

²<https://cspSAT.gitlab.io/sugar/>

(a) nodes テーブル	(b) edges テーブル	(c) colors テーブル	(d) solution テーブル																													
<table border="1"><thead><tr><th>node</th></tr></thead><tbody><tr><td>0</td></tr><tr><td>1</td></tr><tr><td>2</td></tr><tr><td>3</td></tr></tbody></table>	node	0	1	2	3	<table border="1"><thead><tr><th>source</th><th>target</th></tr></thead><tbody><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>2</td></tr><tr><td>2</td><td>3</td></tr><tr><td>0</td><td>2</td></tr></tbody></table>	source	target	0	1	1	2	2	3	0	2	<table border="1"><thead><tr><th>color</th></tr></thead><tbody><tr><td>1</td></tr><tr><td>2</td></tr><tr><td>3</td></tr></tbody></table>	color	1	2	3	<table border="1"><thead><tr><th>node</th><th>color</th></tr></thead><tbody><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>2</td></tr><tr><td>2</td><td>3</td></tr><tr><td>3</td><td>2</td></tr></tbody></table>	node	color	0	1	1	2	2	3	3	2
node																																
0																																
1																																
2																																
3																																
source	target																															
0	1																															
1	2																															
2	3																															
0	2																															
color																																
1																																
2																																
3																																
node	color																															
0	1																															
1	2																															
2	3																															
3	2																															

図 1: グラフ彩色判定問題を表すテーブル (図 1a, 1b, 1c; coloring.db) と実行後に得られる解のテーブル (図 1d)

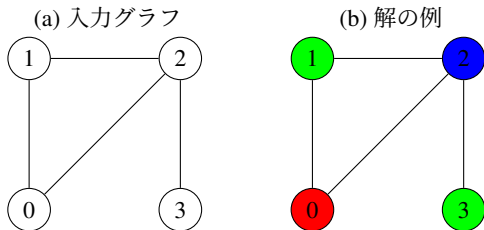


図 2: グラフ彩色判定問題

```

1 -- BEGIN COP
2 CREATE SET SearchSp AS SELECT node, CHOOSE(colors) AS color
  FROM nodes;
3
4 CREATE SET Solutions HAS t IN SearchSp
5 SUCH THAT NOT EXISTS(
6   SELECT target FROM edges
7   WHERE (SELECT color FROM t WHERE node = edges.source) =
8         (SELECT color FROM t WHERE node = edges.target)
9 )
10 CREATE TABLE solution AS t IN Solutions
11 -- END COP

```

コード 1: グラフ彩色判定問題を解くための SQL 文 (coloring.sql)

2 SQL 型制約プログラミングシステム CombSQL+

CombSQL+ のプログラム例として、グラフ彩色判定問題を考える。グラフ彩色判定問題とは、グラフ $G = (V, E)$ と自然数 k が与えられたとき、すべての辺 $\{i, j\} \in E$ に対して $c(i) \neq c(j)$ が成り立つような彩色関数 $c: V \rightarrow \{1, \dots, k\}$ が存在する否かを判定する問題である。制約を満たす彩色関数 c が存在するとき、すなわち G が k 色以下で彩色できるとき、 G は k -彩色可能であるという。図 2a のグラフ G と自然数 $k = 3$ が与えられたとする。グラフ G は図 2b のように彩色できるため 3-彩色可能である。

グラフ彩色判定問題の入力 (図 2) を表すデータベースのテーブルを図 1 に示す。左から、頂点を表すテーブル `nodes`、辺を表すテーブル `edges`、色数 k を表すテーブル `colors` である。

表 1: SolverWrapper クラスの主要な変数とメソッド

インスタンス変数	説明
<code>_solver</code>	ソルバーオブジェクトを格納
<code>_variables</code>	CSP の変数を格納
<code>_choose_variables</code>	CSP の選択変数を格納
<code>_constraints</code>	CSP の制約を格納

メソッド	説明
<code>create_variable</code>	CSP の変数を作成
<code>add_constraint</code>	CSP の制約を追加
<code>solve</code>	ソルバーを実行

グラフ彩色判定問題を解く拡張 SQL 文をコード 1³ に示す。2 行目の文は探索空間を表すテーブルの集合 `SearchSp` を生成する。`SearchSp` 中の各テーブルは、`nodes` テーブル (図 1a) の `node` カラムに対して、`colors` テーブル (図 1c) の値を `color` カラムの値として選ぶことにより、可能なすべての組合せを生成する (図 3)。4–8 行目の文は `SearchSp` からグラフ彩色判定問題の制約を満たすテーブルを要素としてもつ解の集合 `Solutions` を生成する。制約部分 (5–8 行目) は、`edges` テーブル (図 1b) の各辺に対して、両端点が同じ色で塗られるものが存在しないことを表している。10 行目は、集合 `Solutions` からテーブルを 1 つ取り出し、`solution` テーブルを生成する。`solution` テーブルの例を図 1d に示す。この例は、色 1, 2, 3 をそれぞれ赤, 緑, 青とすると、図 2b の彩色を表している。

3 複数異種の CSP ソルバーと連携するためのクラス改良

CombSQL+ は、データベース管理システム SQLite⁴ と Python を用いて実装されている。CombSQL+ は、データベースのテーブルとして表現された問題インスタンスに対して、拡張 SQL 文を実行することにより問題を解

³<https://git.trs.css.i.nagoya-u.ac.jp/combsql/combsqlplus-examples/-/blob/master/coloring/coloring.sql> から色数の最小化の文を除いたものである。

⁴<https://www.sqlite.org/>

$$\text{SearchSp} = \left\{ \begin{array}{cc|cc|cc|cc} \text{node} & \text{color} & \text{node} & \text{color} & \text{node} & \text{color} & \text{node} & \text{color} \\ \hline 0 & 1 & 0 & 1 & 0 & 3 & 0 & 3 \\ 1 & 1 & , \dots & 1 & 2 & , \dots & 1 & 3 \\ 2 & 1 & & 2 & 3 & & 2 & 3 \\ 3 & 1 & & 3 & 2 & & 3 & 3 \end{array} \right\}$$

図 3: 探索空間を表すテーブルの集合 SearchSp

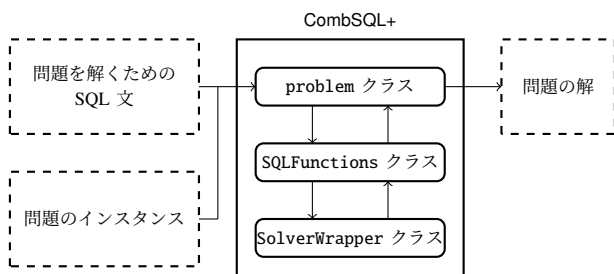


図 4: CombSQL+のクラス構成

くための CSP を生成し、バックエンドの CSP ソルバーを用いて解を求めるシステムである。CombSQL+のクラス構成を図 4 に示す。拡張 SQL 文は problem クラスによって構文解析される。SQLFunctions クラスは SQL 文を実行すると同時に、SQLite のユーザー定義関数の副作用を利用して CSP を生成する。生成された CSP は、SolverWrapper クラスで実行される。しかしながら、現行の SolverWrapper クラスは、SMT ソルバーの Python インターフェースである pySMT 専用に設計・実装されている。そのため、現行の CombSQL+は、SMT ソルバー以外の他の CSP ソルバーを利用することができない。

CombSQL+から複数異種の CSP ソルバーを利用可能にするため、バックエンドの CSP ソルバーを管理する SolverWrapper クラスを抽象クラスとして再設計する。新しい SolverWrapper クラスの主要なインスタンス変数とメソッドを表 1 に示す。この他にも、算術演算子や論理演算子を含む制約式を作成するメソッドがある。create_variable メソッドは、CSP の変数を新しく作成する。add_constraint メソッドは、CSP の制約を追加する。solve メソッドは、CSP ソルバーを実行し解を求める。

CombSQL+のバックエンドで使用する CSP ソルバーはすべて、この抽象クラス SolverWrapper のサブクラスとして実装される。第 5 節では、サブクラスの実装例として、SAT 型 CSP ソルバー Sugar を管理する CSPSolverWrapper クラスの実装を示す。

```

1 #!/usr/bin/env python3
2 from glazesugar.CSP import CSP, Var, Domain
3 from glazesugar.Sugar import Solver
4
5 def main():
6     color = [1, 2, 3]
7     nodes = [0, 1, 2, 3]
8     edges = [(0, 1), (1, 2), (2, 3), (0, 2)]
9     csp = CSP()
10    xs = []
11    for n in nodes:
12        x = csp.int(Var(f"INT_{n}"), Domain(color))
13        xs.append(x)
14    for e in edges:
15        csp.add(xs[e[0]] != xs[e[1]])
16    solver = Solver(csp)
17    result = solver.find()
18    if result:
19        print(solver.solution())
20
21 if __name__ == "__main__":
22    main()

```

コード 2: GlazeSugar のプログラム例 (coloring.py)

```

% python3 coloring.py
{'INT_0': 3, 'INT_1': 1, 'INT_2': 2, 'INT_3': 1}

```

コード 3: GlazeSugar の実行例

4 Sugar の Python インターフェース

SAT 型 CSP ソルバー Sugar の Python インターフェース GlazeSugar について述べる。GlazeSugar を用いることにより、Python プログラムから簡単に Sugar を利用できる。そのため、CombSQL+ と Sugar のシームレスな接続を実現する上でも有用である。

GlazeSugar のクラスは、CSP を表現するためのクラスと CSP ソルバーを管理するためのクラスの 2 つに大別される。表 2 に主要なクラスとメソッドを示す。

- **CSP 表現用クラス**には、ブール変数を表す Bool クラス、整数変数を表す Var クラス、ドメインを表す Domain クラス、制約を表す Constraint クラスなどがある。
- **CSP ソルバー用クラス**には、CSP ソルバーを管理する抽象クラス AbstractSolver、Sugar を管理する Solver クラス、Sugar で利用する SAT ソルバーを管理する SatSolver クラスなどがある。

表 2: GlazeSugar の主要なクラスとメソッド

クラス	説明	親クラス
Bool	ブール変数を表すクラス	Constraint
Var	整数変数を表すクラス	Term
Domain	ドメインを表すクラス	
Term	項を表すクラス	
Constraint	制約を表すクラス	
CSP	CSP を表すクラス	
AbstractSolver	CSP ソルバーを管理する抽象クラス	
Solver	Sugar ソルバーを管理するクラス	AbstractSolver
SatSolver	SAT ソルバーを管理するクラス	

メソッド	説明	クラス
bool	CSP のブール変数を生成し, CSP に追加する	CSP
int	CSP の整数変数を生成し, CSP に追加する	CSP
add	CSP に制約を追加する	CSP
find	CSP の解を探索し, 充足可能かを返す	Solver
solution	CSP の解を返す	Solver

GlazeSugar を用いてグラフ彩色判定問題 (図 2) を解くプログラムをコード 2 に示す。6-8 行目は図 2a のグラフと色をリストで表している。9-15 行目はグラフ彩色判定問題を表す CSP を生成している。10-13 行目は、CSP.int メソッドを使用し、各頂点を表す整数変数を定義する。各整数変数のドメインは色を表す整数値である。14-15 行目は、隣接する頂点と同じ色で塗られないことを表す等号否定制約を、CSP.add メソッドを用いて追加する。16 行目は生成した CSP を引数としてソルバーを生成している。17 行目は Solver.find メソッドを用いてソルバーを実行し、結果を result に格納する。18-19 行目は result が True、すなわち、充足可能な場合には、Solver.solution メソッドを用いて解を取得し、表示する。最後に、GlazeSugar の実行例をコード 3 に示す。

```

1 def create_variable(self, var_type, domain = None,
2   is_choose_variable = False):
3     i = len(self._variables)
4     if var_type == int:
5         var_name = f"INT{i}"
6         if domain is None:
7             整数変数はドメインが必要なためエラーを出力
8             v = self.csp.int(CSP.Var(var_name), CSP.Domain(
9               domain))
10        elif var_type == bool:
11            var_name = f"BOOL{i}"
12            v = self.csp.bool(CSP.Bool(var_name))
13        else:
14            想定されていない型の場合はエラーを出力
15            self._variables[var_name] = v
16            if is_choose_variable:
17                self._choose_variables[var_name] = v
18            return v

```

コード 4: create_variable メソッドの実装

```

1 def add_constraint(self, constraint):
2     < 中略 >
3     self._constraints.append(constraint)
4     self.csp.add(constraint)

```

コード 5: add_constraint メソッドの実装

5 CombSQL+ と Sugar の連携

CombSQL+のバックエンドで使用する CSP ソルバーはすべて、第 3 節で述べた抽象クラス SolverWrapper のサブクラスとして実装される。本節では、第 4 節で述べた GlazeSugar を利用し、SAT 型 CSP ソルバー Sugar を SolverWrapper のサブクラス CSPSolverWrapper として実装する。これにより、CombSQL+ から SMT ソルバーだけでなく、Sugar が利用可能となる。

Sugar を管理する CSPSolverWrapper クラスの主要なメソッド create_variable, add_constraint, solve について、擬似コードを示す。

create_variable メソッド (コード 4) は、変数の型 var_type, ドメイン domain, 選択変数かどうかを表すフラグ is_choose_variable を引数として受け取る。変数の型が int の場合は、CSP.int メソッドを使って整数変数を生成する (3-7 行目)。同様に、bool の場合は、CSP.bool メソッドを使ってブール変数を生成する (8-10 行目)。最後に 16 行目で生成した変数オブジェクトを返す。add_constraint メソッド (コード 5) は、制約 constraint を引数として受け取る。その制約を _constraint リストに格納し、CSP.add メソッド

```

1 def solve():
2     solver = ソルバーオブジェクト
3     timeout = 制限時間
4
5     def _find(lst):
6         is_sat = solver.find()
7         if is_sat:
8             lst[0] = (True, solver.solution())
9         else:
10            lst[0] = (False, None)
11
12     < 中略 >
13
14     manager = SyncManager()
15     manager.start()
16     lst = manager.list([(None, None)])
17     p = Process(target=_find, daemon=False, args=(lst,))
18     p.start()
19     p.join(timeout)
20     result, model = lst[0]
21     return result, model

```

コード 6: solve メソッドの実装

を使って CSP に追加する (3-4 行目). solve メソッド (コード 6) は Python の Process クラスを用いて, _find メソッドを実行する (17-19 行目). _find メソッドでは Solver.find メソッドを用いてソルバーを実行し, 結果を is_sat に格納する (6 行目). is_sat が True, すなわち, 充足可能な場合には, Solver.solution メソッドを用いて解を取得し, 共有リスト lst に格納する (7-8 行目). 最後に 21 行目で, 充足可能かどうかを表すフラグ (result) と解 (model) を返す.

Sugar 版 CombSQL+ の実行例として coloring.db (図 1) と coloring.sql (コード 1) を入力として, グラフ彩色判定問題を解いた結果をコード 7 に示す.

6 まとめ

本稿では, CombSQL+ を複数異種の CSP ソルバーと連携可能にするための拡張について述べた. また, 具体的な連携例として, 高速 CSP ソルバー Sugar との連携方法を示した. また実際に, CSP ソルバー Sugar の Python インターフェースの設計と実装を行い, これを CombSQL+ と連結した. 今後の課題としては, CSP の代表的なグローバル制約である alldifferent 制約への対応, CombSQL+ の記述例の蓄積が挙げられる.

参考文献

- [1] Ian P. Gent, Ian Miguel, and Peter Nightingale. Generalised arc consistency for the alldifferent constraint: An empirical survey. *Artificial Intelligence*, Vol. 172, No. 18, pp. 1973–2000, 2008.
- [2] 岩沼宏治, 鍋島英知. SMT: 個別理論を取り扱う SAT 技術. *人工知能学会誌*, Vol. 25, No. 1, pp. 86–95, 2010.

```

% combsql+ -f --solution --solver sugar coloring.sql coloring.
db
reading input query...
done.
executing pre-query...
done.
parsing COP query...
done.
analyzing query...
done.
translating query...
done.
generating constraints by sugar...
done.
solving constraints...
done.
constructing solution...

Table: solution
-----+-----+
| node | color |
-----+-----+
| 0 | 1 |
| 1 | 3 |
| 2 | 2 |
| 3 | 3 |
-----+-----+
done.
SAT
-----
Processing Times (sec.)
-----
query parsing:          0.0010688305
query analyzing:       0.0025942326
query translation:     0.0094368458
constraint generation: 0.0027430058
solving:               0.4189620018
solution construction: 0.0064232349
total:                 0.4486520290
-----
Solver Statistics
-----
number of variables:   10
number of constraints: 7

```

コード 7: CombSQL+ の実行例

- [3] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, Vol. 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.
- [4] Genki Sakanashi and Masahiko Sakai. Transformation of combinatorial optimization problems written in extended SQL into constraint problems. In David Sabel and Peter Thiemann, editors, *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming (PPDP 2018)*, pp. 19:1–19:13. ACM, 2018.
- [5] Takehide Soh, Mutsunori Banbara, and Naoyuki Tamura. Proposal and evaluation of hybrid encoding of CSP to SAT integrating order and log encodings. *Int. J. Artif. Intell. Tools*, Vol. 26, No. 1, pp. 1760005:1–1760005:29, 2017.
- [6] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint*

Programming (CP 2006), LNCS 4204, pp. 590–603, 2006.

- [7] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, Vol. 14, No. 2, pp. 254–272, 2009.
- [8] 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.
- [9] 梅村晃広. SMT 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.