

擬ブール制約の導入による組合せ最適化ソルバ CombSQL+ の高速化

岸 潤一郎[†] 酒井 正彦[†] 西田 直樹[†] 橋本 健二[†]

[†] 名古屋大学 大学院情報学研究科

あらまし 著者らはこれまでに、拡張した SQL で組合せ最適化問題を簡単に記述でき、かつ効率的に解くことのできる求解系 CombSQL+ を提案した。CombSQL+ はシステム内部で、入力 of SQL 式からプログラム変換によって得られる SQL 式を DB エンジンで実行することでいくつかの線形整数制約 (QFLIA) を生成し、SMT ソルバを利用してそれらの解を求めている。本稿では、変換後の SQL 式で用いられる制約生成関数を工夫することで、擬ブール制約として表現できる制約を直接、擬ブール制約として生成する方法を提案する。この改良により問題によっては 7 倍以上の高速化が達成できたことを報告する。

キーワード 組合せ最適化, SQL, SAT, SMT, 擬ブール制約

Speeding up combinatorial optimization solver CombSQL+ by introducing Pseudo-Boolean constraints

Junichiro KISHI[†], Masahiko SAKAI[†], Naoki NISHIDA[†], and Kenji HASHIMOTO[†]

[†] Graduate School of Informatics, Nagoya University

Abstract The authors have proposed a solver CombSQL+ for combinatorial optimization problems (COPs) described in extended SQL language. The system generates QFLIA constraints by executing an SQL query transformed from an input description, and then solves them by an SMT solver. This report proposes an improvement of the system to generate constraints in pseudo-Boolean forms if possible. We report that the enhancement accelerates more than seven times faster for a problem.

Key words Combinatorial optimization, SQL, SAT, SMT, Pseudo-Boolean constraint

1 はじめに

著者らはこれまでに、組合せ最適化問題を容易に記述可能で、かつ効率的に解くことができるシステム **CombSQL+** [5] を提案した。組合せ最適化問題を解くための制約プログラミングがこれまで提案され実用化されているが、問題記述のためにはプログラミングの知識が必要になるため一般ユーザにとっては敷居が高い。これに対して CombSQL+ ではデータベースの問合せに用いられる SQL 構文をほぼそのまま使用した問題記述を可能にすることで、その敷居を下げることに成功している。しかしながら、このシステムでは汎用の SMT 制約をソルバを用いて問題解決をしているものの、他の制約プログラミングシステムと比較して求解速度において不満が残る。

CombSQL+ では、TOTAL や COUNT など問題記述でも多用される SQL の集計関数を使用すると、ブール変数の値を If-Then-Else 制約により 0 と 1 に変換してから整数演算を施す形式の SMT 制約を生成する。SMT ソルバは背景理論の切り替えにより効率が悪化しやすいため、これが実行速度に大きな影響を与

える原因の一つと考えられる。

本稿では、CombSQL+ システムの SMT 制約の生成において可能な限り $2x + 3y - 4z < 5$ のような形式をした擬ブール制約 (**PB** 制約) を生成することにより、システムの実行速度の向上を目指す。

CombSQL+ では、入力の記述中の SQL 問合せから変換により得られる SQL 問合せを、DB エンジンで実行することにより、SMT ソルバに入力する線形整数制約 (QFLIA) を生成するように設計されている。この SQL から SQL への変換においては、TOTAL などの集計演算や等号 = などの比較演算の代わりに、CombSQL+ システムが用意する関数に置き換えており、変換結果の SQL 問合せの実行中にそれらの置き換えられた関数が制約を生成する。そのため、PB 制約を生成するための処理が複数の関数に分散してしまい、そのままでは PB 制約の生成は容易ではない。そこで、制約生成の関数間でやり取りするデータに工夫を加え、制約生成を遅延させることで PB 制約を生成する方法を提案する。実際にその手法を実装することで、PB 制約として表現できる問題が 7 倍以上高速に実行され

る例を報告する。

以降では、2節で準備、3節では CombSQL+ とその仕組みを簡潔に述べ、4節では本論文のアイデアを、5節ではその実装を、6節では実験の結果について述べる。

2 準備

制約変数 x_i ($i = 1, \dots, n$) への値 v_i の割当を $\alpha = \{x_1 \mapsto v_1, \dots, x_n \mapsto v_n\}$ と書く。制約変数上の論理式 ψ への割当 α の適用結果 $\alpha(\psi)$ が真となる時、 α は ψ を満たすという。

c_i を整数定数、 b_i をブール変数、 k を整数定数とする。このとき、擬ブール制約 (PB 制約) は、以下の形式からなる。

$$\sum_i c_i b_i \# k \text{ where } \# \in \{=, \neq, <, \leq, >, \geq\}$$

[例 1] PB 制約 $1x_1 + 2x_2 + 3x_3 = 3$ を満たす割当は、 $\{x_1 \mapsto \text{true}, x_2 \mapsto \text{true}, x_3 \mapsto \text{false}\}$ と $\{x_1 \mapsto \text{false}, x_2 \mapsto \text{false}, x_3 \mapsto \text{true}\}$ の二つである。

SQL 問合せは、表 1 のようなテーブルの集まりで構成される関係データベースから、データを取り出す式である。ここで、テーブルはレコード (行) の集まりからなり、また、レコードはカラム名がついた値の集まりである。例えば表 1

表 1 テーブル Obj

num
1
2
3

のテーブルに対して問合せ `SELECT TOTAL(num) FROM Obj` を実行することにより、そのレコードのカラム `num` の値の総和 6 が計算される。TOTAL(num) のようにすべてのレコードのカラム `num` の値から計算を行う関数は、集計関数と呼ばれる。本稿においては集計関数 $a(c)$ に対して、カラム c の値 d_1, \dots, d_n を集計する関数を $\bar{a}(d_1, \dots, d_n)$ と表す。

3 CombSQL+

本稿で提案する手法のベースとなる CombSQL+ 言語及びシステムの概略を述べる。

3.1 SQL に基づく組合せ最適化問題記述

文献 [3] で提案された SQL に基づく組合せ最適化問題記述言語を、次の簡単な例を用いて説明する。

[例 2] 組合せ最適化問題 Sel は、整数の集合が与えられて、制約「合計が 3」を満たす部分集合 (解) のうちで、目的関数「要素数」を最大にする最適解を求める問題である。集合 $I = \{1, 2, 3\}$ はこの問題のインスタンスの一つであり、その解の候補からなる集合 (解空間) は、 I のべき集合となる。また、このインスタンスにおける問題の解は $\{1, 2\}$ と $\{3\}$ であり、最適解は $\{1, 2\}$ である。

[例 3] 例 2 で示した問題のインスタンスは、表 1 に示すようにカラム名 `num` を持つテーブル `Obj` として表せる。CombSQL+ 言語では、この組合せ最適化問題は図 2 のように記述される。

図 2 の 1 行目では、テーブル `Obj` を集合とみなしたときに、すべての部分集合 (を表すテーブル) からなる集合を、新たに導入された `SUBTABLE OF` 構文を利用して作成し、新たに導入された `CREATE SET` 構文によりテーブルの集合 `SearchSp` を定義し

```

1 CREATE SET SearchSp AS SUBTABLE OF SELECT * FROM Obj;
2
3 CREATE SET Solutions HAS t IN SearchSp SUCH THAT
4   (SELECT TOTAL(num) FROM t) = 3
5
6 CREATE TABLE solution AS t IN Solutions
7   MAXIMIZING(SELECT COUNT(*) FROM t)

```

図 2 例 2 の問題の CombSQL+ による記述

ている。この記述は探索空間を生成することから **Generation** ステップと呼ばれる。3 行目と 4 行目では、`SearchSp` 中のテーブルのうちで、制約「合計が 3」を満たすテーブル (解) からなる集合 `Solutions` を定めている。この記述は探索空間を絞り込んで解空間を与えることから **Filtering** ステップと呼ばれる。6 行目と 7 行目では、テーブル `t` から整数を返す目的関数を与え、解の集合 `Solutions` の要素 `t` から目的関数を最大にするテーブルの一つを `solution` として定義している。この記述は最適解のうちの一つを求めることから **Selection** ステップと呼ばれる。このように、CombSQL+ 言語は、Generation, Filtering, Selection の 3 ステップから構成される記述により、組合せ最適化問題を記述する枠組みを提供している。

3.2 CombSQL+ システムの概要

図 3 に CombSQL+ [4], [5] のシステム構成を示す。

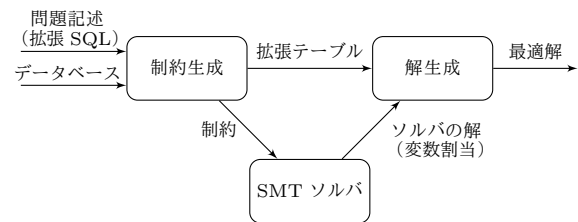


図 3 CombSQL+ システムの構成

システムの入力は、図 2 に示すような CombSQL+ 言語による問題記述と、表 1 に示すような問題のインスタンスが入ったデータベースファイル (テーブル) からなり、その最適解のうちの一つが格納されたテーブルをデータベースの中に生成する。

制約生成部は、問題記述中の SQL 式から 3.4 節で述べる変換で得られる SQL 問合せを、問題のインスタンスが格納されているデータベースのもとで実行することにより線形整数制約 (QFLIA) に変換し、SMT ソルバに渡す。ここで、変換後の SQL 問合せの実行では DB エンジンとして、副作用をもつユーザ関数の利用が可能な SQLite^(注1) を用いており、変換後の SQL 中の制約生成関数の副作用により制約を SMT ソルバに送っている。

目的関数に基づく解の最適化については以下のように処理されている。SMT ソルバには Filtering ステップの制約だけでなく、Selection ステップに記述されている目的関数値を格納する整数変数 x_g に関する制約も同時に送っておき、SMT ソルバの最初の実行により解のうちの一つが求められると、 x_g の値からその解の目的関数値 v が得られるようになっている。さら

(注1): <https://www.sqlite.org>

に、目的関数値を最大化する場合には阻止制約 $x_g > v$ (最小化
 する場合には $x_g < v$) を SMT ソルバに追加して実行すること
 を繰り返すことで、その最適解を求めている。

3.3 制約付きテーブル

CombSQL+ システムにおいては、拡張テーブル R^+ と制約
 ψ の組からなる制約付きテーブル [3] を使って、テーブルの集合
 を保持している。ここで拡張テーブル R^+ は、値として制約変
 数を持つことができ、かつ、ブール型のカラム **ext** を持つテ
 ーブルである。また、制約 ψ は拡張テーブルが持つ変数上の制約
 論理式である。

変数割当 α と拡張テーブル R^+ から定まるテーブル $\alpha(R^+)$
 は、 R^+ の変数に α の値を割り当てた上で、カラム **ext** の値が
 偽となる行をすべて取り除き、さらにカラム **ext** を削除して得
 られるテーブルである。制約付きテーブル $\langle R^+, \psi \rangle$ は、制約 ψ
 を満たすような割当 α で定まるテーブル $\alpha(R^+)$ すべてからなる
 集合を表す。このテーブルの集合を以下のように表記する。

$$\|R^+\|_{\psi} = \{\alpha(R^+) \mid \alpha \text{ は制約 } \psi \text{ を満たす割当}\}$$

[例 4] 例 3 における探索空間 **SearchSp** は、表 4 の制約付き
 テーブルと恒真な制約の組 $\langle R^+_{\text{SearchSp}}, \text{true} \rangle$ により表現される。
 例えば、割当 $\alpha = \{x_1 \mapsto \text{true}, x_2 \mapsto \text{true}, x_3 \mapsto \text{false}\}$ に対
 して、 $\alpha(R^+_{\text{SearchSp}})$ は表 5 に示されるテーブルである。したが
 って、 $\|R^+_{\text{SearchSp}}\|_{\text{true}}$ は探索空間を表していることが分かる。

表 4 制約付きテーブル R^+_{SearchSp}

num	ext
1	x_1
2	x_2
3	x_3

表 5 $\alpha(R^+_{\text{SearchSp}})$

num
1
2

3.4 SQL 変換

問題記述中の SQL 問合せから、評価によりソルバ制約を生
 成する SQL 問合せへの変換 tr [3] について説明する。拡張テ
 ーブルを含む問合せ $Q(R^+)$ から変換 tr によって得られる問合せ
 $tr(Q(R^+))$ は副作用を持つため、その実行により拡張テ
 ーブル S^+ が得られると共に副作用として制約 ψ が得られる。この
 とき、以下の性質を満たす拡張テーブル S^+ が生成されるように
 変換 tr が設計されている。

$$\|S^+\|_{\psi} = \{Q(R) \mid R \in \|R^+\|_{\text{true}}\}$$

このことは、生成された SQL 問合せ $tr(Q(R^+))$ が、 R^+ が表
 す集合の各要素に対する問合せ Q の実行結果からなる集合を
 実現していることを表している。

以下では、変換 tr について変換例を示す。

[例 5] 例 3 の問題記述 (図 2) の変換を考える。1 行目の
SearchSp では変換後の問合せの実行により表 4 に示される
 拡張テーブル R^+_{SearchSp} が得られているとする。4 行目の
 問合せ中の t を R^+_{SearchSp} に置き換えて得られる問合せ

$(\text{SELECT TOTAL}(\text{num}) \text{ FROM } R^+_{\text{SearchSp}}) = 3$

を tr により、次の問合せに変換する。

$\text{EQ}'((\text{SELECT TOTAL}'((\text{num}, \text{ext})) \text{ FROM } R^+_{\text{SearchSp}}), 3)$

ここで、**TOTAL'** と **EQ'** は図 6 の疑似コードで表される (Comb-
 SQL+ が用意した) SQL データ関数である (以下では制約生成
 関数と呼ぶ)。また、 $\text{ite}(x, v_1, v_2)$ は、 x が真のとき v_1 を、偽
 のとき v_2 を返す If-Then-Else 制約、**assert** は引数の制約を
 SMT ソルバに渡すための関数である。

```

1 def TOTAL'((m1,x1),..., (mn,xn))
2   foreach i in [1,...,n]
3     yi := fresh_variable()
4     assert( "yi <=> ite(xi, mi, 0)" )
5   y := fresh_variable()
6   assert( "y = y1 + ... + yn" )
7   return y
8 end
9 def EQ'(m1,m2)
10  z := fresh_variable()
11  assert( "z <=> m1 = m2" )
12  return z
13 end

```

図 6 制約生成関数 **TOTAL'** と **EQ'**

この変換で得られた問合せを評価することで、以下の制約 ψ
 を SMT ソルバに送ると同時に制約変数 z からなるテーブルを
 返す。

$\psi: y_1 \Leftrightarrow \text{ite}(x_1, 1, 0)$
 $\wedge y_2 \Leftrightarrow \text{ite}(x_2, 2, 0)$
 $\wedge y_3 \Leftrightarrow \text{ite}(x_3, 3, 0)$
 $\wedge y = y_1 + y_2 + y_3$
 $\wedge z \Leftrightarrow y = 3$

この制約生成については 4.1 節でもう一度議論する。

なお、図 2 の 3 行目では、4 行目で返された変数 z を真にす
 るように記述されているため、 z を真とする制約もソルバに送
 られるようになっている。

例 5 における **EQ'** や **TOTAL'** などの制約生成関数は、通常の
 SQL 問合せに用いられる関数 f のそれぞれに対して、それ
 に対応する関数 f' を CombSQL+ システムで準備している [4]。

以上のことから、制約生成関数は以下の定義にまとめられる。

[定義 6]

- (1) 非集計関数 $f(v_1, \dots, v_n)$ ($1 \leq n$) について、 $f'(v_1, \dots, v_n)$
 は新しい変数 y を返す関数である。ここで、副作用として
 $y = f(v_1, \dots, v_n)$ と等価な制約を生成する。
- (2) 集計関数 $a(c)$ について、それを計算する関数を \bar{a} とする。
 このとき、 $a'(c, b)$ を計算する関数 \bar{a}' は新しい変数 y を返
 す関数である。ここで $\bar{a}'(\{(v_1, b_1), \dots, (v_n, b_n)\})$ は、副
 作用として $y = \bar{a}(\{v_i \mid b_i = \text{true}\})$ と等価な制約を生成
 する。

ここで、制約付きテーブルにおいては、**ext** 列の値によつて
 はあるレコードがテーブルに存在しない場合が生じる。そのため、 a
 を計算する関数 \bar{a} は、それを考慮した定義になっている。

4 PB 制約の導入

本節ではまず、前節で述べた制約生成手法において、素直な
 拡張では PB 制約の生成が困難であることを示し、さらにこの

手法で生成する SMT 制約のうちで、PB 制約として表現可能な部分については PB 制約として生成するために遅延評価の手法を導入する。

4.1 SMT 制約の生成の問題点

例 5 における変換後の SQL 問合せ (以下) の実行をもう少し詳しく説明する。

```
EQ'((SELECT TOTAL'(num*ext) FROM RSearchSp+), 3)
```

まず内側の SELECT 文の実行によって、以下の部分

```
TOTAL'((1, x1), (2, x2), (3, x3))
```

が実行されることで、変数 y を返すと共に、例 4 の制約 ψ の最初の 4 行が生成される。次に $EQ'(y, 3)$ が実行されることで、変数 z が返されると共に残りの制約 $z \Leftrightarrow y = 3$ が生成される。

この制約 ψ は、PB 制約 $z \Leftrightarrow 3 = 1x_1 + 2x_2 + 3x_3$ と等価であるものの、現在の CombSQL+ の仕組みでは EQ' と $TOTAL'$ という 2 つの独立する関数の呼び出しは必須であるため、この PB 制約の生成は単純ではない。

4.2 PB 制約の生成のアイデア

PB 制約を導出するための方法として以下の 2 つの方法を検討する。

- 生成された制約を解析して、PB 制約としてまとめる。
- 制約生成関数に遅延評価を導入して、PB 制約が生成可能かが判断できるようになるまで制約生成を先延ばしする。

前者の方法は CombSQL+ 以外のシステムにも適用可能な汎用的な手法ではあるが、PB 制約に集約可能な制約が散らばっている可能性があることから、PB 制約に必要な制約の探索が必要になるため、その処理が重くなることが予想される。そこで本研究においては後者の手法を採用する。

4.1 節で述べた例における PB 制約の生成には、 $TOTAL'$ と EQ' の両方の制約生成関数に関わっている。一般に CombSQL+ システムにおいて制約生成関数は、定義 6 で定義されているように、引数としてデータ値かあるいは制約変数を受け取り、制約変数を返すように設計されている。実行を通じて PB 制約を生成するために、データ値および制約変数に加え、整数値とブール変数の組のリストも制約生成関数間でやり取りすることとし、必要に応じて制約の生成を遅延させることで PB 制約の生成を可能にする。

先の例においては、 $TOTAL'$ の実行では制約を生成する代わりに、制約を生成するために必要な情報、すなわち、リスト $[(1, x_1), (2, x_2), (3, x_3)]$ を返しておく。さらに、 EQ' では与えられた引数に応じて、可能な場合には PB 制約を生成し、そうでない場合には $TOTAL'$ において処理を遅延させていた制約生成を行った上で、さらに本来の EQ' の処理を行う。このような遅延処理を導入することにより PB 制約の生成が可能になる。

4.3 PB 制約生成のための制約生成関数

4.2 節のアイデアに従って PB 制約の生成を可能にする制約生成関数について述べる。

[定義 7] 整数値と制約変数の対のリスト $[(m_1, x_1), \dots, (m_n, x_n)]$ を整数リストという。

制約生成関数が受け渡すデータとして、これまで許されていたデータ値と制約変数に加えて、整数リストを許すように拡張

する。

この考え方に基づいて拡張された制約生成関数の概略は以下のようなになる。

[定義 8] 非集計関数 $f(v_1, \dots, v_n)$ ($1 \leq n$) について $f(v_1, \dots, v_n)$ を、集計関数 $a(c)$ についてそれを計算する関数 $\bar{a}(\{(v_1, b_1), \dots, (v_n, b_n)\})$ を以下のように設計する。

- 可能な場合には、戻り値を整数リストに変換して返す。
- 可能な場合には PB 制約を生成し、新たな制約変数を返すと共にさらに必要な制約を生成する。
- そうでない場合には、新たな制約変数を返すと共に必要な制約を生成する。

[例 9] 拡張された制約生成関数 $TOTAL'$ および EQ' は以下の疑似コードで具体的に表すことができる。ここで $type$ は引数だが、具体的な値のとき $const$ を、制約変数のとき var を、整数リストのとき $intList$ を返す関数である。また、33 行目の整数リストの減算 $m_1 - m_2$ は、それぞれの整数リストが表す値の減算となるような整数リストを表す。

```

1 def list2var( m )
2   case type(m) of
3     intList =>
4       foreach (mi,xi) in m
5         yi := fresh_variable()
6         assert( "yi <=> ite(xi, mi, 0)" )
7         y := fresh_variable()
8         assert( "y <=> y1 + ... + yn" )
9         return y
10      otherwise => return m
11
12 def TOTAL'((m1,x1),..., (mn,xn))
13   if type(m1) = intList then
14     foreach i in [1,...,n]
15       mi := list2var( mi )
16     case (type(m1), type(x1)) of
17       (const, const)
18         => return ite(m1,x1,0)+ ... +ite(mn,xn,0)
19       (const, var )
20         => return [(m1,x1),..., (mn,xn)]
21     otherwise
22       => return list2var( [(m1,x1),..., (mn,xn)] )
23
24 def list2PB_EQ( [(m1,x1),..., (mn,xn)], m )
25   y := fresh_variable()
26   assert( "y <=> (m1*x1 + ... + mn*xn = m)" )
27   return y
28
29 def EQ'(m1,m2)
30   case (type(m1), type(m2)) of
31     (intList, const) => return list2PB_EQ( m1, m2 )
32     (const, intList) => return list2PB_EQ( m2, m1 )
33     (intList, intList) => return list2PB_EQ( m1-m2, 0 )
34     otherwise =>
35       m1 := list2var( m1 )
36       m2 := list2var( m2 )
37       y := fresh_variable()
38       assert( "y <=> (m1 = m2)" )
39       return y

```

図 7 拡張された制約生成関数 $TOTAL'$ および EQ'

変換 tr により得られる以下の問合せを考える。

```
EQ'((SELECT TOTAL'(num*ext) FROM RSearchSp+), 3)
```

この問合せは、図 7 の拡張された制約生成関数と表 4 の制約付

きテーブルのもとで、次のように実行される。まず、

```
TOTAL'((1,x1),(2,x2),(3,x3))
```

が実行され 20 行目の処理がなされることにより、以下の整数リストが生成される。

```
[(1,x1),(2,x2),(3,x3)]
```

その後、

```
EQ'([(1,x1),(2,x2),(3,x3)], 3)
```

の実行により 31 行目から list2PB.EQ が呼び出されることで、PB 制約 $y \Leftrightarrow (1x_1 + 2x_2 + 3x_3 = 3)$ を生成した上で、制約変数 y が返される。なお、PB 制約が生成不可能な場合には、最終的には拡張前と同じ制約が生成されるように工夫されている。

4.4 目的関数値の最適化における PB 制約の生成

4.3 節までに述べた方法のみでは、Selection ステップで繰り返し実行される目的関数値の最適化のために追加される制約が、そのままでは PB 制約として追加できない。例 3 の例題の Selection ステップ (図 2 の 7 行目) を用いて説明する。目的関数を与える MAXIMIZING 中の SELECT 文は以下に示す問合せに変換される。

```
SELECT COUNT'((num,ext)) FROM RSearchSp+
```

これを拡張前の制約生成関数と表 4 の制約付きテーブルのもとで実行されることにより得られる整数変数 z を保持しておき、ソルバの実行結果の割当から x の値である目的関数値を利用して 3.2 節で説明したような阻止制約を生成していた。制約生成関数を拡張することにより、制約変数の代わりに整数リスト $[(1, x_1), (1, x_2), (1, x_3)]$ が得られるため、MAXIMIZING における阻止制約の処理を変更する必要が生ずる。すなわち、制約変数に加えてその整数リストも保持しておくことで、制約変数とソルバの実行結果から目的関数値が得られ、保持していた整数リストと合わせて PB 制約の形式で阻止制約が生成できる。

5 実装

CombSQL+ バージョン 0.9.5 を、以下で述べるように PB 制約を生成できるように拡張してバージョン 0.9.6 を作成した。さらに、本稿では触れていないが、これまで未実装だった SUBTABLE OF 構文も実装しバージョン 0.9.7 とした。これらのソースコードは、公開されている^(注2)。

5.1 pySMT の拡張

CombSQL+ は、図 3 の構成において、ライブラリ pySMT [2] を介して SMT ソルバを利用している。しかしながら pySMT には PB 制約を取り扱うためのインタフェースが存在しないため、そのアプリケーションインタフェースを実装した。さらに、SMT ソルバ Z3 [1] では PB 制約が取り扱えるため、PB 制約の受け渡しができるように pySMT から Z3 を利用するインタフェースを拡張した。これにより、3 種類の PB 制約 (PBLE, PBGE, PBEQ) が pySMT で使用可能になった。これらの拡張済の pySMT は optimization+pb ブランチにて現在公開されて

いるが^(注3)、準備が整い次第 pySMT の開発元^(注4)にプルリクエストを出す予定である。

5.2 CombSQL+ の拡張

SQLite で使用可能な集計関数のうち、COUNT, TOTAL, 等式・不等式 $\{=, \neq, \leq, <, \geq, >\}$ の制約生成関数に前節で述べた変更を行った。その他、整数を引数に取る制約生成関数についても、定義 8 に従って適切に遅延されていた制約を生成できるように変更した。

6 実験

いくつかの組合せ最適化問題に対して、PB 制約を生成する場合としない場合の差を、生成される制約の数、PB 制約の数、実行速度の比較を行った。実験環境は、OS として macOS Catalina, CPU は Core i5 2.6GHz, メモリ 8GB である。

表 8 商品のテーブル PL

product	size
1	3
2	4
⋮	⋮
49	5
50	1

まず、以下に示すナップサック問題に対する結果を示す。

- 探索空間：50 個の商品とそのサイズ情報 (表 8) が与えられて、いくつかの商品を選ぶ。
- 制約：選んだ商品の合計サイズが 50 以下である。
- 目的関数最適化：選択した商品数を最大にする。

```
1 CREATE SET SearchSp AS
2   SELECT product, size, CHOOSE(BOOLS) AS selected FROM PL;
3
4 CREATE SET Solutions HAS t IN SearchSp SUCH THAT
5   (SELECT TOTAL(size) FROM t WHERE selected=TRUE) <= 50
6
7 CREATE TABLE solution AS t IN Solutions
8   MAXIMIZING(SELECT COUNT(*) FROM t WHERE selected=TRUE)
```

図 9 CombSQL+ によるナップサック問題の記述

この問題の CombSQL+ 上での記述を図 9 に示す。ここで、2 行目のテーブル BOOLS は、ブール値 true と false の二つのレコードからなる表である。ここで、CHOOSE(BOOLS) AS selected の記述においてカラム selected を追加しており、各行に対して product が選ばれているとき、かつそのときに限り、カラム selected に true が入るように探索空間を設計した。これは、当初は未実装だった SUBTABLE OF 構文を用いない設計である。

表 10 ナップサック問題の実行結果の比較

	PB 制約なし	PB 制約あり
目的関数最適値	17	
実行時間 [秒]	3057	425
(内制約生成)	(0.07)	(0.12)
変数生成数	201	153
制約生成数	252	204
(内 PB 制約)	-	(10)

(注3) : <https://git.trs.css.i.nagoya-u.ac.jp/combsql/pysmt>

(注4) : <https://github.com/pysmt/pysmt>

(注2) : <https://git.trs.css.i.nagoya-u.ac.jp/combsql/combsqlplus>

この問題に対する結果を表 10 に示す。ここで、図 9 の仕様の 5 行目の (SELECT TOTAL(size) FROM t) <= 50 に関して一つの PB 制約が生成される。さらに、8 行目 MAXIMIZING(SELECT COUNT(*) FROM t) の目的関数値の最適化においては、現在の最適値より優れた目的関数値が得られるたびに一つずつ PB 制約が生成され、合計 9 個の PB 制約が生成された。

実行時間はそのほとんどが SMT ソルバの求解に費やされている。また、PB 制約の利用効果は大きく、7 倍以上の高速化が確認された。

表 11 SUBTABLE OF 利用した場合の比較

	PB 制約なし	PB 制約あり
目的関数最適値	17	
実行時間 [秒]	1708	443
(内制約生成)	(0.04)	(0.09)
変数生成数	151	103
制約生成数	102	54
(内 PB 制約)	-	(13)

次に、図 9 の 2 行目、5 行目、8 行目をそれぞれ、

```
SUBTABLE OF (SELECT * FROM PL)
(SELECT TOTAL(size) FROM t) <= 50
MAXIMIZING(SELECT COUNT(*) FROM t)
```

に書き換えて実行した場合について表 11 で示す。この結果から、PB 制約を使わない場合においても、SUBTABLE OF の利用は高速化に貢献していることがわかる。PB 制約を生成する場合においては、SUBTABLE OF の利用による速度差は殆どなく、PB 制約は速度的に安定した効果があったと言える。

最後にグルーピング問題について PB 制約の有無による差を示す。この問題は 60 名程度を 4 名のグループに分割することを 3 回繰り返すもので、どの 2 名も高々一度しか同一グループにならないという制約を持つ。その結果、PB 制約の利用の有無によらず、制約を満たす解が得られた。表 12 に示すように、PB 導入後の方が、制約生成に 5 秒ほど多くかかるものの、SMT ソルバにかかる時間が 25 秒ほど短縮され、全体として高速になっている。

表 12 グルーピング問題の実行結果の比較

	PB 制約なし	PB 制約あり
実行時間 [秒]	69.4	49.6
(内制約生成)	(11.6)	(16.6)
変数生成数	39651	34027
制約生成数	40017	34393
(内 PB 制約)	-	(144)

7 まとめ

CombSQL+ の制約生成の際、PB 制約も扱えるようにするための制約生成手法を提案した。また、その実装により、PB 制約が生成可能な問題の実行の高速化が可能であることを示した。

SQL の集約関数には TOTAL と同様に数値を加算する関数 SUM がある。それらの違いは、計算対象の行が存在しないときに

TOTAL が 0 を返すのに対して、SUM は NULL を返す点にある。未定義を意味する NULL を適切に処理することにより、SUM の制約生成関数においても PB 制約の生成が可能と考えられ、その実装は今後の課題である。その他の課題としては、SQL 上の文字列処理への対応や、全ての最適解を求めるモードの実装、SUM の生成が挙げられる。また、整数を複数のブール値で表現するにより、線形整数制約を生成する代わりに PB 制約とブール論理式のみを生成し、SAT ソルバを利用することで、さらなる高速化が可能と考えている。これについても今後の課題である。

文 献

- [1] Nikolaj Björner, Anh-Dung Phan, and Lars Fleckenstein. *vZ: An optimizing SMT solver*. In Christel Baier and Cesare Tinelli, editors, *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Vol. 9035 of *Lecture Notes in Computer Science*, pp. 194–199. Springer, 2015.
- [2] Marco Gario and Andrea Micheli. PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In *Proceedings of the 13th International Workshop on Satisfiability Modulo Theories*, 2015. <http://smt2015.csl.sri.com/accepted-papers/>.
- [3] 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*, pp. 19:1–19:13. ACM, 2018.
- [4] Genki Sakanashi and Masahiko Sakai. Transformation of SQL-based combinatorial optimization problems into constraint problems. Proceedings of the 112th meeting of Special Interest Group on Fundamental Problems in Artificial Intelligence SIG-FPAI-B903-03, The Japanese Society for Artificial Intelligence, 2020.
- [5] 坂梨元軌, 酒井正彦, 西田直樹, 橋本健二. 組合せ最適化問題の記述から SMT ソルバの入力式を生成する SQL 問合せ. 信学技法 SS2018-66, 電子情報通信学会, 2019. Vol. 118, No. 471, pp. 85–90.