

Answer Set Programming を用いた圧縮指標の計算

クップル ドミニク *

番原 睦則 †

概要

この論文では、最小の bidirectional macro scheme と最小の straight line program との圧縮指標を計算するため、answer set programming (ASP) 言語で作成した符号化を提案する。既存研究では、両方の圧縮指標は NP 困難だと知られ、その上で satisfiability problem (SAT) の符号化が提案された。本論では、符号化を簡略することで、行数の少ない ASP 言語のコードを表現する。暫定の実験によって、簡略した符号化と ASP 言語の実装は優れていると言えよう。

1 序論

可逆圧縮は、圧縮のうち、復元前の文字列を完全な形で取り出すことが可能な圧縮方法である。しかし、文字列の可逆圧縮における多様な圧縮表現の計算は難しいとみなされる。その中でも、最小の straight line program (SLP) と最小の bidirectional macro scheme (BMS) の計算は NP 困難である [11, 12]。Bannai ら [13, 3] は両方の圧縮指標を MAX-SAT で符号化し、pySAT [9] との検証ソフトウェア (ソルバー) で計算を行った。それらを踏まえ、より速い計算速度を目指し、提案された符号化を簡略することで、最小の BMS・SLP を ASP で計算する方法を提案する。

2 予備知識

Σ をアルファベットとし、 Σ^* の要素を文字列と呼ぶ。文字列 T の長さを $|T|$ と表記する。文字列 $T = XYZ$ について X, Y, Z をそれぞれ文字列 T の接頭辞、部分文字列、接尾辞と呼ぶ。 $T[i]$ は T 中の i 文字目の文字、 $T[i..j]$ は T の i 文字目から j 文字目までの部分文字列を表す。 $j = |T|$ のとき、 $T[i..j]$ は開始位置 i を持つ接尾辞であり、 $T[i..]$ とも書く。たまた、最後の位置を省きたいとき、 $T[i..j+1) = T[i..j]$ を書く。 ϵ は空文字列を示す。

以降、文字列 T を入力として固定する。 $\sigma := |\Sigma|$ と $n := |T|$ はアルファベットサイズと $T[1..n] = T$ の長さをそれぞれ示す。この論文で文字列 $T = \text{abaababaabaab}$ を使用例として実行する。

文字列分解とは T を複数の部分文字列 F_1, \dots, F_z に分解する。ただし、 $T = F_1 \dots F_z$ である。各部分文字列 F_x を項と呼ぶ。項 F_x の開始位置と F_x の長さは、 dst_x と λ_x がそれぞれ示している。言い換えると、 $F_x = T[\text{dst}_x..\text{dst}_x + \lambda_x)$ 。 $T[1..n] = F_1 \dots F_z$ を F_1, \dots, F_z の項に分解する BMS は、以下の状況を満たす。

1. 各 $x \in [1..z]$ に対して $|F_x| \geq 2$ の場合、 F_x は参照先 $\text{src}_x \in [1..n]$ を持つ。ただし、 $T[\text{src}_x..\text{src}_x + \lambda_x) = F_x$ である。
2. 説明のため、関数 $R(\text{dst}_x) := \text{src}_x \forall x$ を定義する。 $(R$ は参照元から参照先に写像する。) R の定義域を広げた場合、

$$\forall x \in [1..z], |F_x| \geq 2 : \quad (1)$$

$$R(\text{dst}_x + k) = \text{src}_x + k \quad \forall k \in [0..|F_x|)$$

*山梨大学 大学院 総合研究部 工学域 電気電子情報工学系

†名古屋大学 大学院情報学研究所 情報システム学専攻 計算論

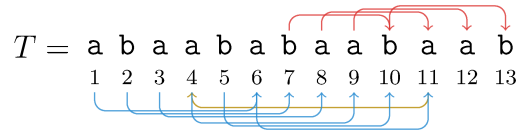
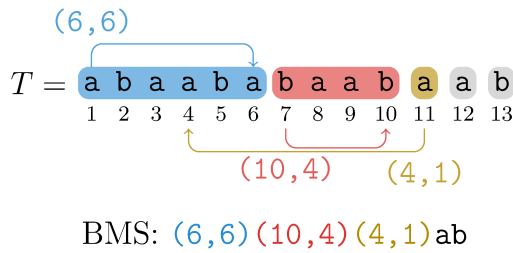


図 1: 左から文字を読む貪欲法で作成した BMS . 左 : 各項は丸い四角で表示されている . 項の長さは 2 以上である場合 , (参照先, 長さ) の二組で表現されている . 右 : 各位置の R 値を矢印で表示している . 図 2 と比較すると , 最小ではない . グラフ $([1..n], \{(i, R(i)) \mid \text{定義された } R(i)\})$ に対して , 最長の経路は $3 \rightarrow 8 \rightarrow 11 \rightarrow 4 \rightarrow 9 \rightarrow 12$ である .

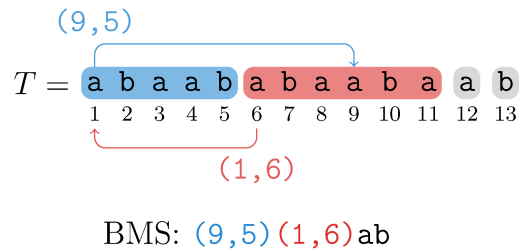
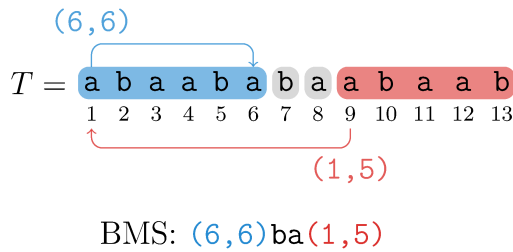


図 2: 最小の BMS の例 . 両方の側は最小のもので , 複数の答えがありうる .

となり , グラフ (V, E) は閉路を持ち得ない . ただし , V はテキスト位置の集合 $[1..n]$ と $E := \{(i, R(i)) \mid i \in V, \text{定義された } R(i)\}$ である .

図 1 は例を示すが , 最小の BMS にならない (図 2 と比較) .

テキスト位置 j は R の像ではない ($j \neq R(i) \forall i$) 場合 , j を森構造の根と見なす . その j は参照先なしの項で表現する . 特別に , その項は長さ 1 を持つ . 図 1,2 で灰色に彩られている .

SLP はたいてい特徴的な文法で定義されているが , この論文で特殊な BMS として扱う . なぜなら , 下記の定義と定理を利用するためである .

定義 1 (文法分解, [3, 節 3.2]). **文法分解**とは , 下記の条件を満たす BMS $F_1 \cdots F_z = T$ である .

- 各項 $F_x = [\text{dst}_x \dots \text{src}_x + \lambda_x]$ に対して , $\lambda_x \geq 2$ の場合 , 空間 $Q_x = [\text{src}_x \dots \text{src}_x + \lambda_x]$ の存在がある . ただし ,

- $\text{src}_x + \lambda_x \leq \text{dst}_x$,
- $F_x = T[\text{src}_x \dots \text{src}_x + \lambda_x]$,
- $\exists y \in [1..x-1] : \text{src}_x = \text{dst}_y$, かつ ,
- $\exists y' \in [y+2..z] : \text{src}_x + \lambda_x = \text{dst}_{y'}$.

- 任意の 1 より長い 2 つの項 F_x と F_y ($|F_x| \geq 2, |F_y| \geq 2$) に対して , 空間 Q_x と Q_y は交わりを持たない , または , いずれかのひとつは他の空間に含まれる . 言い換えると , $(Q_x \cap Q_y = \emptyset \vee Q_x \subseteq Q_y \vee Q_y \subseteq Q_x)$.

定理 1 ([3, Lemma 2]). 任意のテキスト T に対して , T の SLP を文法分解で帰着できる . 文法分解のサイズは z とすると , 帰着した SLP のサイズは $z + \sigma - 1$ になる .

定理 1 に従って , 最小の文法分解で最小の SLP に帰着できる . 図 3 は文法分解の例を示す .

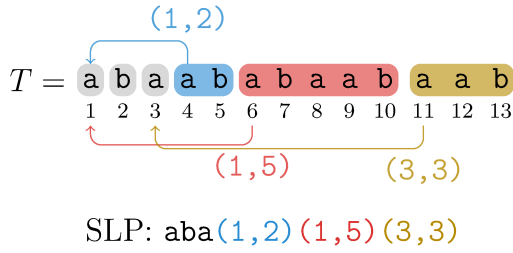


図 3: 文法分解の例．定義 1 によって，参照先は項の開始位置より前になる．さらに，参照先は別の項の開始位置となり，参照先の範囲は複数の項を完全に参照する．

3 ASP 表現

ASP の符号化は MAX-SAT の符号化と整数の符号化，両方を利用できる．下記に説明する圧縮指標の計算で， p_1, \dots, p_n は項の開始位置を表現する．つまり， $p_i \implies \exists x \in [1..z] : \text{dst}_x = i$ である．目的は，できるだけすべての p_i の真理値を偽にセットする．計算を可能にするため，真と偽をそれぞれ 1 と 0 で整数的に表現する．以下の p_i に関する節をできるだけ充足すれば良いが，その以外の節は必ず充足しなければならない．

$$\sum_{i=1}^n p_i \text{ を最小化} \quad (\text{MINP})$$

実装で様々な入力を扱うため，入力をバイトアルファベットで表現する．そのため，各文字を整数に変更する．例えば，ASCII を利用すれば， $a \mapsto 97$ ， $b \mapsto 98$ になる．

例 1. 例文字列として，入力は以下ようになる．
`#const textlength=13. t(1,97). t(2,98).`
`t(3,97). t(4,97). t(5,98). t(6,97).`
`t(7,98). t(8,97). t(9,97). t(10,98).`
`t(11,97). t(12,97). t(13,98).`

次項では，最小の BMS と最小の文法分解の ASP 符号化を提案する．文法分解は BMS と特徴ので，一般的な BMS で始める．

3.1 最小の BMS

Bannai ら [13] の説明に従って， R との関数を選択した後，BMS の項を計算する． R の定義によって，各テキスト位置は参照先を持ちうる． R から BMS を帰着できるため，定義したグラフ (V, E) は森構造になると確認すべき．森構造から，以下のことを定める．参照先を持たない位置 v は長さ 1 の項になる．その一方で， v は参照先 w があるとすれば， w は v の親である．2 つの隣接なテキスト位置 $T[i]$ と $T[i+1]$ は下記の 2 つの条件を満たせば，同じ項に含める可能性がある．

1. i と $i+1$ は親を持つ，つまり $R(i), R(i+1) \in [1..n]$ は定義されている．
2. $R(i+1) = R(i) + 1$ である（図 1 の右側のようにならずしている参照先）．

つまり，項の開始位置 p_i と R の選択で BMS を帰着できる．

選択変数.

以上の議論に従って，符号化のため，選択変数 p_1, \dots, p_n で項の開始位置と $r_{i,j}$ で写像 $R(i) = j$ を表現する．そのとき，任意 $i, j \in [1..n]$ に対して， $r_{i,j}$ は頂点の親子関係を示す． $r_{i,j}$ は真の場合，頂点 v_i の親頂点は v_j となる．

節.

ここでは， p_i と $r_{i,j}$ を充足すべき節を説明する．まずは，2 つの同じ文字を持つテキスト位置は互いに参照する関係性を持ちうる．

$$\forall i, j \in [1..n], i < j \wedge T[i] = T[j] : r_{i,j} + r_{j,i} \leq 1 \quad (\text{BMS1})$$

[3, 方程式 (9)] $\{O(n^2), O(1)\}$

各節を表示する方程式に対して，節の個数と節に含まれている変数は右側の波型の括弧で表示する．この場合，節の個数と節に含まれている変数はそれぞれ $O(n^2)$ と $O(1)$ となる．次は，各テキスト位置は参照先を最大ひとつ持つことができると表現する．

$$\forall i \in [1..n] : \sum_{j=1}^n r_{i,j} \leq 1 \quad (\text{BMS2})$$

[3, 方程式 (11)] $\{O(n), O(1)\}$

テキスト位置は参照先を持たない場合，項の開始位置になる．

$$\sum_{j=1}^n r_{i,j} = 0 \implies p_i \quad (\text{BMS3})$$

[3, 方程式 (13)] $\{O(n), O(1)\}$

方程式 1 を満たすため，テキスト位置 i の直前の位置は異なる参照先を持つ場合， i は項の開始位置になる．

$$\forall i, j \in [1..n] : r_{i,j} \wedge \neg r_{i-1,j-1} \implies p_i \quad (\text{BMS4})$$

[3, 方程式 (15)] $\{O(n^2), O(1)\}$

最後に， $r_{i,j}$ は森構造を満たすための制限を追加する．その目的は， $r_{i,j}$ で閉路を作らせないためである．

$$\forall k \geq 2 : \#(i_1, \dots, i_k) : r_{i_1, i_2} \wedge r_{i_2, i_3} \wedge \dots \wedge r_{i_{k-1}, i_k} \wedge r_{i_k, i_1} \quad (\text{BMS5})$$

方程式 BMS5 は既存研究でこのように提案された．[13] は各頂点に深さの情報を追加する． $r_{i,j}$ はさらに浅い頂点 v_j を指すことしかできないため，閉路は発生しない．しかし，その場合，深さの次元が追加されることで，3次元の選択変数になり，計算時間がかかってしまう．その一方で，[3] の符号化方法では，推移閉包 $t_{i,j}$ を $(r_{i,j} \implies t_{i,j}, t_{i,j} \wedge r_{j,k} \implies t_{i,k})$ で計算し， $t_{i,j} \wedge t_{j,i}$ が真にならないように制限が加わる．そこで，ASP の力をコマンド #edge [6] でブラックボックスとして利用する．

コード 1: 最小の BMS の ASP 符号化．最後の行 (コマンド show) は出力を作るためのものである．

```
{ r(I,J) ; r(J,I) } 1 :- t(I,C), t(J,C),
    <math>\implies I < J.</math> %(BMS1)
:- not { r(I,_) } 1, I=1..textlength. %(
    <math>\implies</math> BMS2)
p(I) :- 0 { r(I,_) } 0, I=1..textlength.
    <math>\implies</math> %(BMS3)
p(I) :- r(I,J), not r(I-1,J-1). %(BMS4)
```

```
#edge (I,J): r(I,J). %(BMS5)
#minimize { 1,I : p(I) }. %(MINP)
#show r/2.
#show p/1.
```

コード 1 は提案した BMS の符号化を ASP 言語で表示する．符号化を例 1 に応用すると，以下の出力を求める．

図 2 の左: p(1) p(7) p(8) p(9) r(1,6) r(2,7)
r(3,8) r(4,9) r(5,10) r(6,11) r(9,1)
r(10,2) r(11,3) r(12,4) r(13,5)

図 2 の右: p(1) p(12) p(13) p(6) r(1,9)
r(2,10) r(3,11) r(4,12) r(5,13) r(6,1)
r(7,2) r(8,3) r(9,4) r(10,5) r(11,6)

3.2 最小の文法分解

SLP を計算するため，定義 1 を利用し，最小の文法分解を計算する．そこで，以下の 2 つの性質に着目する．

1. $R(i)$ は定義された場合， $R(i) < i$ である．
2. 項 F_x は参照先 src_x を持つ場合，3 つのテキスト位置 $\text{src}_x / \text{src}_x + j / \text{src}_x + |F_x|$ は，すべて異なる項の開始位置になる．ただし， $j \in [\text{src}_x + 1.. \text{src}_x + |F_x|]$ である．

前半の性質によって，閉路を作ることができないため，閉路の確認を省くことができる．それによって，符号化は簡易になる．他方で，後半の性質を確認するため，項の長さを符号化しない場合は，困難になる．従って， l_i は位置 i から始まる項を符号とする場合， $(\text{dst}_x = i$ とすると， $l_i = \lambda_x)$ ，単純に l_i を p_i の直後の真になる p_j の距離 $j - i$ で計算できる．

$$\forall i \in [1..n], j \in [i+1..n] :$$

$$\sum_{k=i+1}^{j-1} p_k = 0 \wedge p_i \wedge p_j \implies l_i = j - i \quad (2)$$

[3, 方程式 (1, 2)] $\{O(n^2), O(n)\}$

その結果，大規模な重い節が発生し，計算が難しくなる．原因は j の広い定義域である．定義域を狭めるため，以下の2つの関数を定義する．関数の値は l_i の上界になる．

- 写像 $\text{lce} : [1..n] \times [1..n] \rightarrow [1..n]$ は $T[i..]$ と $T[j..]$ の最長共通接頭辞の長さを出力する．公式的に言えば， $\text{lce}(j, i) = d \Leftrightarrow T[i..i+d-1] = T[j..j+d-1] \wedge (\max(i, j) = n - d + 1 \vee T[i..i+d] \neq T[j..j+d])$ である．
- 写像 $\text{lpnf} : [1..n] \rightarrow [1..n]$ は位置 i に対して i の前から始まる接尾辞の最大 lce 値を示す [5]．ただし， $T[i..]$ と重複しないように任意の接尾辞を i の直前まで比較する． lce を用い， lpnf を $\text{lpnf}(i) := \max_{j \in [1..i-1]} \min(\text{lce}(j, i), i - j)$ で計算できる．図4は例を表示する．

方針は， i から始まる項と見なせば， $l_i \leq \text{lpnf}(i)$ である． $R(i) = j$ なら， $l_i \leq \text{lce}(j, i)$ である．

ソルバーが2つの関数を利用できるため，前処理ですべての $\text{lce} \cdot \text{lpnf}$ 値を ASP で列挙し，以下のように入力ファイルとして保存する． $\text{lce}(i, j, 1)$ ． $\text{lpnf}(i, 1)$ ．そうすると，ASP の入力は整数になる．整数の個数は $|\text{lpnf}| = n$ ， $|\text{lce}| = n(n-1)/2$ である．ただし，各整数の定義域は $[1..n]$ である．

例 2. #const n=13.

$\text{lce}(2, 7, 5)$. $\text{lce}(2, 8, 0)$. $\text{lce}(2, 9, 0)$.
 $\text{lce}(2, 10, 4)$ $\text{lpnf}(2, 0)$. $\text{lpnf}(3, 1)$.
 $\text{lpnf}(4, 3)$. $\text{lpnf}(5, 2)$

l_i を正しく計算するため，最初の項の開始位置と最後の項の終了位置の直後を定義する．

$$p_1 = p_{n+1} = 1 \quad (\text{SLP1})$$

$$\{\mathcal{O}(1), \mathcal{O}(1)\}$$

以下では， $T[i..]$ から項 F_x が始まると考える ($\text{dst}_x = 1$ とのこと)． $\text{lpnf}(i) \leq 1$ の場合， F_x の長さは1だと定められた．

$$\forall i \in [1..n] : p_i \wedge \text{lpnf}(i) \leq 1 \implies p_{i+1} \quad (\text{SLP2})$$

$$\{\mathcal{O}(n), \mathcal{O}(1)\}$$

$\text{lpnf}(i) \geq 2$ の場合， F_{x+1} の開始位置は $[i+1..i+\text{lpnf}(i)]$ の範囲であるはず．

$$\forall i \in [1..n] : p_i \wedge \text{lpnf}(i) \geq 2 \implies \sum_{j=i+1}^{i+\text{lpnf}(i)} p_j \geq 1 \quad (\text{SLP3})$$

$$\{\mathcal{O}(n), \mathcal{O}(m)\}$$

ただし， $m := \mathcal{O}(\sum_{i=1}^n \text{lpnf}(i))$ は lpnf の値の総和を示す．項の長さ l_i を下記のように計算する．

$$\forall i \in [1..n], j \in [i+1..i+\text{lpnf}(i)+1] :$$

$$\left(\sum_{k=i+1}^{j-1} p_k = 0 \right) \wedge p_i \wedge p_j \implies l_i = j - i \quad (\text{SLP4})$$

$$\{\mathcal{O}(nm), \mathcal{O}(m)\}$$

方程式2と比較すれば， m は小さければ小さいほど効果がある．次は， $l_i \geq 2$ の場合， i から始まる項は参照先を持つべき．参照先 j を r_{i,j,l_i} 表現する．ただし， $T[j..j+l_i] = T[i..i+l_i]$ である．すなわち， $\text{lce}(i, j) \geq l_i$ ．

$$\forall i \in [1..n] :$$

$$l_i \geq 2 \implies \sum_{j=1}^{i-l_i} (\text{lce}(i, j) \geq l_i \wedge r_{i,j,l_i}) = 1 \quad (\text{SLP5})$$

$$[3, \text{方程式}(3+4)] \{\mathcal{O}(n), \mathcal{O}(n)\}$$

定義1に定義された空間 Q_x を $q_{\text{dst}_x, \lambda_x}$ で象る． $j = \text{dst}_x$ と $l_i = \lambda_x$ とすると，以下のすべての条件が満たされる．

- j から項が始まる，
- $j + l_i$ から項が始まる，かつ，
- $T[j..j+l_i]$ は項にならない(つまり， $T[j+1..j+l_i]$ の範囲で別な項が始まる)．

i	1	2	3	4	5	6	7	8	9	10	11	12	13
$T[i]$	a	b	a	a	b	a	b	a	a	b	a	a	b
$\text{lpnf}(i)$	0	0	1	3	2	5	5	4	5	4	3	2	1

図 4: lpnf の値についての例 . $\text{lce}(6,1) = 6$ であるが , $T[1..6]$ は $T[6..]$ と重複してるので , 6 は $\text{lpnf}(6)$ の値にならない .

$$\forall i \in [1..n], j \in [1..n], \ell_i \in [1..\text{lpnf}(i)] : \quad (\text{SLP6})$$

$$r_{i,j,\ell_i} \implies p_j \wedge p_{j+\ell_i} \wedge \ell_j \neq \ell_i \wedge q_{j,\ell_i}$$

[3, 方程式 (6+7)] $\{\mathcal{O}(n^2m), \mathcal{O}(1)\}$

最後に , 任意の 2 つ参照された空間は重複しないように確認する .

$$\forall i, j, d_i, d_j \in [1..n] :$$

$$q_{i,d_i} \wedge q_{j,d_j} \implies \neg(i < j < i + d_i < j + d_j) \quad (\text{SLP7})$$

[3, 方程式 (8)] $\{\mathcal{O}(n^2m^2), \mathcal{O}(1)\}$

方程式 SLP7 のすべての d_j の定義域の大きさは $\mathcal{O}(m)$ になる理由は , 参照元は i とすると , $d_j \leq \text{lpnf}(i)$ である . 節の説明は以上ある . 利用された補助変数は ℓ_i と $q_{i,d}$ である . ℓ_i と $q_{i,d}$ は長さ λ_x と空間 Q_x を象る .

コード 2: 最小の文法分解の ASP 符号化.

```
p(1). p(textlength+1). %(SLP1)
p(I+1) :- p(I), lpnf(I,E), E <= 1, I <=
    ↪ textlength. %(SLP2)
1 { p(J) : J = I+1..I+E } :- p(I), lpnf(I
    ↪ ,E), E > 1. %(SLP3)
1(I,J-I) :- p(I), p(J), lpnf(I, E); J = I
    ↪ +1..I+E+1; not p(Z) : Z=I+1..J-1.
    ↪ %(SLP4)
1 { r(I,J,L) : J <= I-L, lce(J,I,E), E >=
    ↪ L } 1 :- l(I, L), L > 1. %(SLP5)
4 { not l(J,L); p(J); p(J+L); q(J,L) } 4
    ↪ :- r(_,J,L). %(SLP6)
```

```
:- q(I,L), q(J,M), I < J, J < I+L, I+L <
    ↪ J+M. %(SLP7)
#minimize { 1,I : p(I), I <= textlength
    ↪ }. %(MINP)
#show r/3.
#show p/1.
```

コード 2 は提案した文法分解の符号化を ASP 言語で表示する . 符号化を例 2 に応用すると , 以下の出力を求める . $p(1) p(14) p(2) p(3) p(4) p(6) p(11) r(4,1,2) r(6,1,5) r(11,3,3)$. 出力で図 3 の例に帰着できる .

3.3 余分の節

既存研究より , 提案した節が少ない . その一方で , 既存研究で正しさが証明された . 従って , 正しさを保つため , 省いた節は余分であると証明できれば , 十分である .

BMS の場合 , 以下の方程式 [3, (14)] を省いた . なぜなら , 以下の方程式で既に扱われている .

- 任意の j に対して , $r_{0,j}$ は偽である . もし , $r_{1,j}$ は真になる位置 j が存在すると , 方程式 BMS4 の条件を満たす . そうでなければ , 方程式 BMS3 の条件が満たされる . 両方で , p_1 は真になる .
- $r_{i-1,0}$ は真にならないので , $r_{i,1} \implies p_i$ は方程式 BMS4 で含んでいる .
- $T[i-1] \neq T[j-1]$ の場合 , $r_{i-1,j-1}$ を選択する方程式 BMS1 を満たさない . 従って , 方程式 BMS4 の条件に当たる .

$i = 1 \vee j = 1 \vee T[i - 1] \neq T[j - 1]$ との条件を満たす任意 $i \in [1..n], j \in \{j : T[i] = T[j]\}$:

$$r_{i,j} \implies p_i \quad ([3, (14)])$$

文法分解の場合、以下の方程式 [3, (5)] を省いた。 $r_{i,j,d}$ は真になる理由は、方程式 SLP5 を満たすことである。しかし、方程式 SLP5 によって、 $\ell_i = d$ であるはずなので、方程式 [3, (5)] は余計である。

$$\begin{aligned} i \in \{k \mid T[k..k + \ell] = T[j..j + \ell], k \in [1..j - \ell]\} \\ \text{との条件を満たす任意} \\ j \in [1..n], \ell \in [2..n + 1 - j] : \\ r_{i,j,d} \implies \ell_j = d \quad ([3, (5)]) \end{aligned}$$

それに、以下の方程式 [3, (6)] の \implies 方向を省いた。 $q_{i,d}$ を真になるため、方程式 SLP6 の条件を満たすべき。しかし、方程式 SLP6 の条件によって、下記の包容は既に満たされた。

$$\begin{aligned} T[i'..i' + \ell] \text{ は } T[i' + \ell..n] \text{ の部分文字列になる} \\ \text{条件を満たす } \forall i' \in [1..n - 1], \ell \in [2..n + 1 - i'] : \\ q_{i',\ell} \iff \bigvee_{1 \leq i' + \ell \leq i \leq n} r_{i' \leftarrow i, \ell} \quad ([3, (6)]) \end{aligned}$$

4 実験

データセットとして、Calgary Corpus [4] を利用した。各ファイルの長さ 128 または 256 がある接頭辞を入力として利用する。ただし、OBJ1 と PIC は計算しにくいので、実験で割愛した。実験で、[13] が利用した pySAT [9]¹ の実装² と比較する。複数の設定の中に、SAT ソルバー Glucose 4 [2, 1] と Max-SAT


ソルバー RC2 [10] の組み合わせを選択した。ASP 符号化を clingo [8, 7] に読ませ、問題の答えを計算させる。ASP 符号化の実装を https://github.com/koepl/asp_compression で提供する。

計算環境として、i5-1135G7 の CPU を持つノートパソコンを利用した。CPU は 8 のスレッドで並列的に計算できることを clingo で利用した。その一方で、Glucose 4 と RC2 はシングルスレッドしか動かすことができない。各実験が 1 分を超えた場合、止めて、N/A でマークした。

表 1 で全体の時間計算量が現れている。ASP の符号化はすべての答えを 1 分で見つけながらも、BMS の SAT 符号化は何回か制限時間を超えた。念の為、[13] が定義した節をそのままに ASP で表現し、₀ の添字でマークした。本論で簡略された節の ASP 符号化と比較した場合、SLP の ASP₀ はすべての例の方が遅い。その一方で、BMS の ASP₀ はたまに一番速い方法になる。あいにく、変数の個数 (表 2) と節の個数 (表 3) を見ると、ASP はすべての例で一番速いはずなので、原因は不明である。

SAT の符号化と ASP の符号化と比較すれば、SLP の場合、SAT は ASP₀ より優れているが、提案した ASP 符号化のように速い。変数を見れば、ASP の個数は最良の場合、二倍ほど縮んでるが、ほぼ同じ個数である。他方で、BMS の符号化を見ると、提案した ASP の符号化は優れている。閉路の確認を #edge のコマンドで行うことは重要な点である。

謝辞

本研究は JSPS 科研費 JP23H04378 と  AFSA JP21K17701 の助成を受けたものです。

参考文献

- [1] Gilles Audemard and Laurent Simon. “Glucose and Syrup: Nine years in the SAT competitions”. In: *Proc. SAT Competition*.

¹<https://pysathq.github.io/>

²<https://github.com/koepl/satcomp>

- Vol. B-2018-1. Department of Computer Science Series of Publications B. 2018, pp. 24–25.
- [2] Gilles Audemard and Laurent Simon. “Predicting Learnt Clauses Quality in Modern SAT Solvers”. In: *Proc. IJCAI*. 2009, pp. 399–404.
- [3] Hideo Bannai, Keisuke Goto, Masakazu Ishihata, Shunsuke Kanda, Dominik Köppl, and Takaaki Nishimoto. “Computing NP-hard Repetitiveness Measures via MAX-SAT”. In: *Proc. ESA*. Vol. 244. LIPIcs. 2022, 12:1–12:16. DOI: 10.4230/LIPIcs.ESA.2022.12.
- [4] Timothy C. Bell, Ian H. Witten, and John G. Cleary. “Modeling for Text Compression”. In: *ACM Comput. Surv.* 21.4 (1989), pp. 557–591. DOI: 10.1145/76894.76896.
- [5] Supaporn Chairungsee, Tida Butrak, Surangkanang Chareonrak, and Thana Charuphanthuset. “Longest Previous Non-overlapping Factors Computation”. In: *Proc. DEXA*. 2015, pp. 5–8. DOI: 10.1109/DEXA.2015.21.
- [6] Martin Gebser, Tomi Janhunen, and Jussi Rintanen. “Answer Set Programming as SAT modulo Acyclicity”. In: *Proc. ECAI*. Vol. 263. Frontiers in Artificial Intelligence and Applications. 2014, pp. 351–356. DOI: 10.3233/978-1-61499-419-0-351.
- [7] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. “Theory Solving Made Easy with Clingo 5”. In: *Proc. ICLP*. Vol. 52. OASICS. 2016, 2:1–2:15. DOI: 10.4230/OASICS.ICLP.2016.2.
- [8] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. “Multi-shot ASP solving with clingo”. In: *Theory Pract. Log. Program.* 19.1 (2019), pp. 27–82. DOI: 10.1017/S1471068418000054.
- [9] Alexey Ignatiev, António Morgado, and João Marques-Silva. “PySAT: A Python Toolkit for Prototyping with SAT Oracles”. In: *Proc. SAT*. Vol. 10929. LNCS. 2018, pp. 428–437. DOI: 10.1007/978-3-319-94144-8_26.
- [10] Alexey Ignatiev, António Morgado, and João Marques-Silva. “RC2: an Efficient MaxSAT Solver”. In: *J. Satisf. Boolean Model. Comput.* 11.1 (2019), pp. 53–64. DOI: 10.3233/SAT190116.
- [11] Hiroshi Sakamoto, Shinichi Shimosono, Ayumi Shinohara, and Masayuki Takeda. *On the Minimization Problem of Text Compression Scheme by a Reduced Grammar Transform*. Technical Report 195. Kyushu University, 2001.
- [12] James A. Storer and Thomas G. Szymanski. “Data compression via textural substitution”. In: *J. ACM* 29.4 (1982), pp. 928–951. DOI: 10.1145/322344.322346.
- [13] 坂内 英夫, 後藤 啓介, 石畠 正和, 神田 峻介, クップル ドミニク, and 西本 崇晃. *SAT ソルバを用いた NP 困難な圧縮指標の高速計算*. Tech. rep. 11. 人工知能学会研究会資料 人工知能基本問題研究会, 2022, pp. 61–66. DOI: 10.11517/jsaifpai.120.0_61.

表 1: 時間 (単位 : 秒)

ファイル		BMS			SLP		
名前	n	ASP	ASP ₀	SAT	ASP	ASP ₀	SAT
BIB	128	0.02	0.05	0.28	0.03	1.03	0.01
BOOK1	128	0.02	0.04	0.22	0.03	1.06	0.01
BOOK2	128	0.02	0.09	0.55	0.03	1.03	0.01
GEO	128	0.07	4.49	50.77	0.15	N/A	0.10
NEWS	128	0.02	0.05	0.15	0.03	1.13	0.01
OBJ2	128	0.04	1.49	3.64	0.07	2.64	0.07
PAPER1	128	0.09	0.32	0.56	0.04	1.15	0.02
PAPER2	128	0.27	0.08	0.26	0.04	1.21	0.01
PAPER3	128	2.53	0.13	0.61	0.04	1.21	0.02
PAPER4	128	0.04	0.04	0.19	0.04	1.09	0.01
PAPER5	128	8.02	0.11	1.12	0.05	1.26	0.03
PAPER6	128	0.04	0.13	0.33	0.04	1.12	0.02
PROGC	128	0.02	0.11	0.44	0.04	1.07	0.01
PROGL	128	0.15	6.85	18.23	6.49	N/A	1.70
PROGP	128	0.02	0.06	0.24	0.03	1.08	0.01
TRANS	128	0.03	0.09	0.45	0.05	1.15	0.05
BIB	256	0.49	3.99	44.13	0.13	N/A	0.04
BOOK1	256	0.13	10.99	N/A	0.21	26.90	0.11
BOOK2	256	0.14	10.79	N/A	0.16	45.01	0.10
NEWS	256	0.27	5.57	13.61	0.19	17.83	0.12
OBJ2	256	19.89	N/A	N/A	1.23	N/A	1.32
PAPER1	256	0.12	3.69	25.18	0.16	20.97	0.12
PAPER2	256	27.29	14.31	29.78	0.22	N/A	0.06
PAPER4	256	0.46	1.42	31.75	0.16	18.00	0.04
PROGC	256	0.21	4.86	N/A	0.12	21.31	0.11
PROGL	256	39.84	48.78	N/A	56.64	N/A	26.12
PROGP	256	1.10	N/A	N/A	0.14	43.43	0.21
TRANS	256	2.65	11.43	N/A	0.18	23.90	0.22

表 2: 変数の個数

ファイル		BMS			SLP		
名前	n	ASP	ASP ₀	SAT	ASP	ASP ₀	SAT
BIB	128	926	9413	27 001	308	12 864	617
BOOK1	128	790	7621	21 743	259	11 765	556
BOOK2	128	1004	10 150	28 822	616	13 524	892
GEO	128	9650	451 829	919 665	6265	N/A	9513
NEWS	128	842	6295	13 923	370	9944	680
OBJ2	128	3808	110 310	235 960	3264	18 263	4593
PAPER1	128	1168	11 709	29 137	695	14 301	1034
PAPER2	128	1040	9524	25 046	542	14 187	883
PAPER3	128	1256	13 586	33 103	781	14 377	1126
PAPER4	128	836	6357	15 696	498	13 524	804
PAPER5	128	1266	10 771	27 351	2029	13 622	2236
PAPER6	128	1064	10 261	25 188	1004	14 058	1255
PROGC	128	1110	13 013	37 116	496	15 331	791
PROGL	128	17 596	875 807	1 730 977	53 953	N/A	92 115
PROGP	128	1000	8482	19 834	509	12 547	840
TRANS	128	918	9305	26 657	1659	12 373	2339
BIB	256	3546	65 581	204 806	1461	N/A	1950
BOOK1	256	3500	75 844	N/A	1156	56 220	1633
BOOK2	256	3608	70 976	N/A	1441	59 715	1965
NEWS	256	3114	43 784	101 441	1458	52 314	2124
OBJ2	256	18 982	N/A	N/A	23 108	N/A	36 335
PAPER1	256	3220	46 336	124 272	1660	61 414	2319
PAPER2	256	3290	46 643	134 968	1895	N/A	2573
PAPER4	256	3216	44 979	131 283	1206	59 633	1704
PROGC	256	4014	102 305	N/A	1548	63 748	2127
PROGL	256	42 112	3 098 763	N/A	203 680	N/A	366 277
PROGP	256	4154	N/A	N/A	2874	58 931	3366
TRANS	256	3652	82 674	N/A	3140	57 158	3830

表 3: 節の個数

ファイル		BMS			SLP		
名前	n	ASP	ASP ₀	SAT	ASP	ASP ₀	SAT
BIB	128	267	1269	83 132	81	12 247	1431
BOOK1	128	237	1029	65 853	59	11 173	1250
BOOK2	128	270	1272	89 344	236	12 763	2386
GEO	128	303	6619	3 160 737	1602	N/A	31 354
NEWS	128	279	1201	41 689	121	9362	1641
OBJ2	128	240	2912	794 509	1327	15 931	23 781
PAPER1	128	237	1243	91 183	282	13 486	2731
PAPER2	128	234	1162	77 254	211	13 449	2233
PAPER3	128	216	1252	104 618	312	13 508	3049
PAPER4	128	237	1043	47 242	199	12 793	2075
PAPER5	128	303	1471	85 299	879	12 233	7226
PAPER6	128	267	1311	78 085	392	13 085	3755
PROGC	128	291	1459	116 631	167	14 592	2018
PROGL	128	315	9699	5 982 279	8389	N/A	518 134
PROGP	128	294	1366	60 868	165	11 852	2325
TRANS	128	255	1179	81 589	867	11 248	14 195
BIB	256	633	4195	670 873	512	N/A	5139
BOOK1	256	636	4210	N/A	389	54 739	4031
BOOK2	256	654	4376	N/A	566	58 092	5307
NEWS	256	618	3822	327 701	599	50 714	7585
OBJ2	256	639	N/A	N/A	5563	N/A	147 666
PAPER1	256	642	3796	401 521	650	59 610	6957
PAPER2	256	633	3783	436 216	779	N/A	7751
PAPER4	256	639	3939	425 296	409	58 086	4367
PROGC	256	633	4707	N/A	612	62 061	6021
PROGL	256	717	23 175	N/A	22 172	N/A	2 432 092
PROGP	256	714	N/A	N/A	1125	56 560	11 849
TRANS	256	636	4292	N/A	1320	54 688	19 454