1B5-5

文生成を題材とした両方向からのモンテカルロ木探索

A Bidirectional Monte Carlo Tree Search Motivated by Sentence Generation

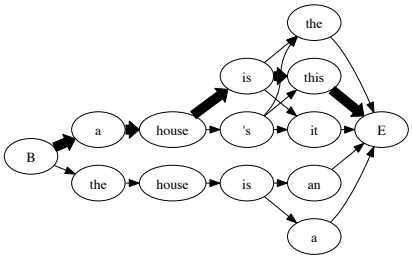
安田宜仁 平尾努 永田昌明
Norihito Yasuda Tsutomu Hirao Masaaki Nagata

日本電信電話株式会社 NTT コミュニケーション科学基礎研究所
NTT Communication Science Laboratories, NTT Corporation

Sentence generation can be defined as a problem of selecting a proper path from a given directed acyclic graph (DAG). To generate high-quality sentences, it is crucial to consider sentence-level features such as word redundancy in the sentence. Unfortunately, such non-linear features prevents us to use efficient search algorithms. This paper describes the prospect of using Monte Carlo tree search (MCTS) algorithm for sentence generation by assuming the DAG as a search tree for MCTS. Moreover, a notable characteristic of sentence generation is that each sentence must has start and end of the sentence. To exploit this characteristic, we propose a bidirectional MCTS that searches best path from both end of the DAG. We conducted two types of experiments; 1) generating machine-translated sentences and 2) artificial problem that finds a path requiring both high divergence and short path length. Both experimental results show that the proposed method outperforms uni-directional MCTS except the case of DAGs whose degrees are extremely unbalanced.

1. はじめに

自然言語処理における文生成は与えられた有向非循環グラフ (DAG) からのパスの選択と捉えることができる問題が存在する. このような種類の文生成では, 語やフレーズをノードとし, 言語モデル等によって得られたスコアをエッジの重みなっているような DAG から, 文頭に相当するルートノードから文末に相当する終端ノードまでを結ぶパスを選択することで文を生成することができる. たとえば, 以下ような DAG では, 「B」「E」のノードが文頭, 文末を示し, 太い矢印で結ばれたパスは「a house is this」という文に相当する.



品質の良い文を生成するためには, 単にエッジの重みの和が高い文を選択するだけではなく, 文全体にかかる性質を考慮することが重要である. たとえば, 要約文生成においては生成された文の中に冗長な表現が含まれていないことが重要であるし, 後述する機械翻訳のオラクル文生成であれば冗長性に加えて語順の考慮も重要である.

生成された文の善し悪しを示す評価値が局所的なスコアの線形和であればダイクストラ法やベルマンフォード法といった効率的なアルゴリズムを適用することで最適なパスを効率的に得ることができるであろう. しかし, 文全体に関連する性質を含めて決定される評価値の場合, 局所的なスコアの和で表現できないため, 直接効率的なパスを求めることはできない.

仮に, グラフが小さければ全解探索も可能かもしれないが, 数十語程度を対象とする文生成においては全てのパスをすべて探索するのは現実的ではない.

連絡先: yasuda.n@lab.ntt.co.jp

そこで, 本稿ではモンテカルロ的試行を繰り返すことにより高い評価を得ることができそうなノードを先頭から順次選択しているような方法を検討する. なかでも, モンテカルロ木探索 (Monte Carlo tree search, MCTS) はモンテカルロシミュレーションと探索を組合せた方法の総称であり, 投じた計算量に応じて高い精度が得られることや, ドメイン知識がなくとも高い性能が得られるといった特徴からコンピュータ碁を中心にさまざまな用途に使われている [2].

本稿では, まず, モンテカルロ木探索における探索木に代えて, 与えられた DAG を探索対象として用いることで, より質の高い文を生成することをまず検討する. さらに, 文生成の場合の特徴として以下の 2 点を挙げることができる.

1. 対戦相手がいるわけではなく, いわば一人ゲームであること. したがって, 相手がどういう手を打ってくるかは分からないという意味での不確実性はなく, パスに対する評価値そのものが最終的に高めたい値であるということ.
2. 文は確実に文頭と文末を持つという強力な情報があること. したがって, 文頭から単語を順に選択することと同様に文末から単語を順に選択することも可能であるということ.

本稿ではこれらの特徴に着目し, 一方向からだけではなく文末側からも順次ノード選択を進めていく, 「両方向からのモンテカルロ木探索」を提案する. これにより, より確度の高い選択に基づいた次ノード選択を行えるようになり, 結果的に高い評価値のパスの選択に繋がることが期待される.

ところで, 本稿で取り扱うような両端が閉じた DAG は, なにも自然言語処理における文生成だけに出現するものではない. そこで, 一般的な人工問題として, パス長とパス中のノード種の多様性を考慮するような問題を考え, 文生成と異なる状況下でも提案法の有効性を検証する.

2. DAG に対する両方向からの MCTS

2.1 DAG に対する片方向からの MCTS

本稿で提案する両方向からの MCTS について述べる前に, まず DAG を対象とした片方向からの MCTS について述べる.

Algorithm 1 文生成のための DAG に対する MCTS

```
function FindpathUnidirect(v_s,v_t,n)
    s_b ← 0, v ← v_s, P ← {v_s}
    repeat
        s_bb ← s_b
        for i = 0 to n do
            s ← Playout(v, v_t, P, {}, FORWARD)
            if s > s_b then
                r ← 1, s_bb ← s
            else
                r ← 0
            end if
            for all v_h in P do
                N(v_h) ← N(v_h) + 1
                W(v_h) ← W(v_h) + r
                S(v_h) ← S(v_h) + s
            end for
        end for
        v ← arg max_{v' ∈ children of v} S(v')/N(v')
        append(P, v)
        s_b ← s_bb
    until v ≠ v_t
    return P
end function
```

Algorithm 2 試行関数と次ノード決定関数

```
function Playout(v_s, v_t, P_f, P_b, direction)
    if direction = FORWARD then
        v_s0 ← v_s, v_f0 ← v_t,
    else
        v_s0 ← v_t, v_f0 ← v_s,
    end if
    v ← v_s0, P ← P_f
    repeat
        if direction = FORWARD then
            C ← children of v
        else
            C ← parents of v
        end if
        if zero played candidate exists in C then
            c ← first zero played in C
        else
            c ← arg max_{v' ∈ C} W(v')/N(v') + α sqrt(2 ln N(v) / N(v'))
        end if
        append(P, c)
    until v ≠ v_t0
    concat(P, P_b)
    return GetEvaluationScore(P)
end function
```

本稿では与えられた DAG を探索木とみなし, ルートノード (文頭に相当) から終端ノード (文末) までのパスの選択に用いることでパスの決定を行う. 通常の MCTS における探索木のノードの展開に対応する操作として, パス中のノードの選択を行う. これは文生成での文中の単語の確定に相当する.

便宜上, DAG はただひとつのルートノードを持つとする. なお, 親を持たないノードが複数ある DAG の場合ルートに相当するノードを追加し, そこから親を持たない各ノードへのエッジを張ることで上記を満たすよう変換することができる.

DAG に対する MCTS の手順を Algorithm 1 に示す. v_s としてルートノードを, v_t として終端ノードを指定する. n はパス中のひとつのノードを確定するための試行回数である. ここではパスの未決定部分, すなわち最後に決定したノードから終端ノードまでについて試行を繰り返し, 有望な子ノードを順次決定していく.

関数 Playout は試行を行うための関数である. 試行中, 子のうちどのノードを次ノードとして選択するかには, UCB1 値を参考にした以下の式で求められる値を用い, この値がもっとも高いノードを選択する.

$$\frac{W(v)}{N(v)} + \alpha\sqrt{\frac{2\ln(\text{子全体の試行回数})}{N(v)}} \tag{1}$$

ここで, $\frac{W(v)}{N(v)}$ はノード v からたどったパスがひとつ前のノードを決定するまでに得られた最大のスコアよりも高いスコアを得ることができた割合を示す. 関数 GetEvaluationScore ではパスに対する評価値を算出する. たとえば後述の評価実験 1 では機械翻訳の評価尺度である RIBES[1] を用いた.

与えられた回数の試行が済めば, 平均スコアのもっとも高い子を最終的に出力するパスの次ノードとして選択する. 以上を DAG 終端ノードまで繰り返す.

2.2 両方向からの MCTS

順にノードを確定していく際, そのノードが必ずしも最適スコアを構成する保証はないので, ノードの確定を進めるにつれ探索漏れがあったノードの上での危うい選択を重ねていると見ることもできる. そういった危うい選択を重ねることを避けることができれば, より精度を高めることが期待される. ここで, 文生成においては文頭同様に, 確率 1 で文末も存在するという強力な情報に着目し, 文頭側, 文末側それぞれから順にノードを選択していくことで上述のような危うい選択を減らすことを提案する.

便宜上, DAG はただひとつのルートノードと, ただひとつの終端ノードを持つとする. 片方向からの議論同様, 任意の DAG はこれを満たすように変換できる.

両方向からの MCTS の手順を Algorithm 3 に示す. 基本的には片方向の場合と同様であるが, ルートノードからだけでなく, 終端ノードからのも交互に選択を行っていくことが特徴である.

ただし, ここでひとつ問題がある. 片方向の場合であればルートから子ノードを順に選択していっても必ず終端に辿りつくことができたのに対し, 両方向からの場合には選択したノードによっては, 前方 (ルート側からの選択) の最も子のノードと後方 (端側からの選択) の最も親のノードを結ぶようなパスが存在しないことが起こり得る. そこで, 後方の最も親のノードへたどりつけないようなノードを取り除く処理を行う. 関数 TrimUnreachables では深さ優先探索を 2 度行うことによりこれを実現する. 1 度目では深さ優先でのたどり順でのノードの親子関係を記録し, 2 度目では深さ優先でのたどり順での子が到達可能であれば親も到達可能であることを再帰的に記録する. この処理は片方向の場合には不要であり, 両方向にすることが速度低下の要因となりうる. 実行時間については 4 節で検証する.

3. 翻訳候補近似オラクル文生成

本節では文生成の例として, 機械翻訳におけるオラクル文生成について述べる. 何らかの前処理の結果得られた機械翻訳システム内部の DAG(ラティス) から潜在的に得られるすべての文のうち, 機械翻訳の評価値がもっとも高くなるような文を, ここではオラクル文と呼ぶ. オラクル文は, 「うまくやればここまで計算機が出力することができる」という文なので, その

Algorithm 3 両方向からの MCTS

```
function FindpathBidirect(v_s, v_t, n)
    s_b ← 0, v_f ← v_s, v_b ← v_t
    direction ← FORWARD, P_f ← {}, P_b ← {}
    repeat
        s_bb ← s_b
        for i = 0 to n do
            s ← Playout(v_f, v_t, P_f, P_b, direction)
            if s > s_b then
                r ← 1, s_bb ← s
            else
                r ← 0
            end if
            for all v_h in P do
                N(v_h) ← N(v_h) + 1
                W(v_h) ← W(v_h) + r
                S(v_h) ← S(v_h) + s
            end for
        end for
        if direction = FORWARD then
            v_f ← argmax_{v' ∈ children of v} S(v')/N(v')
            append(P_f, v_f)
            direction = BACKWARD
        else
            v_b ← argmax_{v' ∈ parents of v} S(v')/N(v')
            append(P_b, v_b)
            direction = FORWARD
        end if
        s_b ← s_bb
        TrimUnreachables(v_f, v_b)
    until v_f ≠ v_b
    return concat(P_f, P_b)
end function
```

ような文が出力されるように機械翻訳システムのパラメータをチューニングすることで翻訳システムの改善に利用できる可能性があるため, なるべく高い評価値の近似オラクル文を見つけることは重要である.

次に, 生成した文に対する評価値について述べる. RIBES[1] はシステム翻訳と人手で作成した参照翻訳との間で共通して出現するような単語の出現順序に着目した自動評価法である. 翻訳結果の文が単に単語集合として参照翻訳と類似しているかだけではなく, 双方の単語列の並びを考慮することで, 人間による評価結果との高い相関があることが報告されている. RIBES スコアは以下の式で与えられる.

$$\mathrm{NKT} \times P^{\alpha} \tag{2}$$

ここで, P は単語正解精度であり, システム翻訳と参照翻訳で共通な単語数を, システム翻訳の単語数で除した値である. NKT は, システム翻訳と参照翻訳双方に出現する単語集合間のケンドール順位相関係数を $[0, 1]$ 区間に正規化した値である. また, α は重みパラメータである (本実験では配布パッケージ*1のデフォルト値に従い $\alpha = 0.25$ とした).

たびたびの確認となるが, このような結果の単語列に対する順位相関は, 結果得られたパスに対する値であり, 探索時の局所的なスコアを使うことができないため, 効率的に探索することができない.

4. 評価

提案手法の有効性を評価するため, 2 種類の実験を行った. ひとつは機械翻訳の近似オラクル文生成であり, もうひとつは文生成以外での一般的な課題として, 選択したパスの長さとパス中のノード種の多様性を考慮するような人工問題である.

4.1 近似オラクル文の実験条件

オラクル文生成の探索対象として, 英日翻訳用にセットアップされた機械翻訳システム Moses[3] のサーチグラフ*2を用いた. 生成対象データとして英語特許文 2000 文を用いた. 各文について日本語の参照翻訳がある. 各英文について Moses のサーチグラフを出力し, 冗長なノードを排除するとで各文の翻訳候補を示すフレーズをノードとする DAG を作成した. 選択したパスの評価には RIBES を用いた.

片方向, 両方向の MCTS に加え, 貪欲法によるベースライン手法と比較した. 貪欲法による方法は, 先頭から順にノードを選択す方法で, 子ノードの選択において当該子ノードまでに対応する文を翻訳結果だとみなした場合に RIBES スコアがもっとも高くなるような子ノードを順次選択していく.

MCTS での各ノードの選択時の試行数は, 1 回から 10 倍刻みで 100 万回までの 7 通りとした.

実行時間はプロセス開始から終了までの実時間で計測した. 実験には多数の計算機を使用したため, 各事例についての計測条件はまちまちであるが, これらを平均することにより手法間・試行回数間の比較はできると考える.

4.2 人工問題の実験条件

文生成以外の, パスの評価値が各ノードのスコアの線形和で表現にならないような例題として, パスの長さとパス中のノードの多様性の双方を考慮しなければならないような人工問題を設定した. DAG 中の各ノードには種別を示すための「色」がついているとする. 探索の目的は DAG と始点と終点が与えられた場合に, 始点と終点を結ぶパスのうち以下に定めるパスのスコアがもっとも高くなるようなパスを見つけることである.

$$\frac{C - l}{C} \times \frac{d}{C} \tag{3}$$

ここで, C は始点と終点を結ぶパスのうち最も長いパス (クリティカルパス) のノード数, l はパス中のノード数, d は選択したパス中での色の種類を示す. 探索の目的は DAG 中の始端と終端を結ぶパスのうち, 上記スコアができるだけ高いものを選ぶことである.

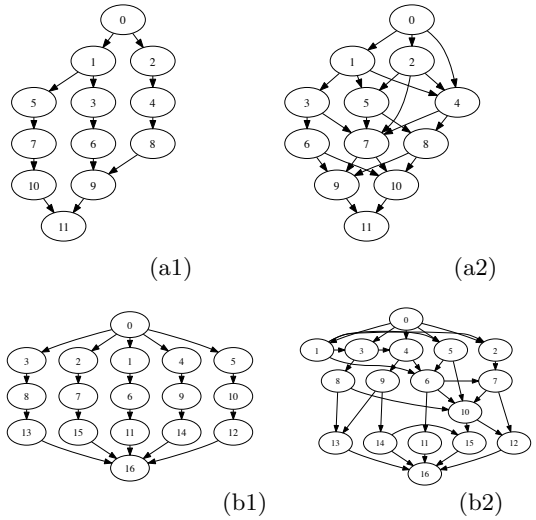
検証に用いる DAG は大きく分けて 2 種類を作成した. 一方はどのノードも平均して同じ出次数と入次数となるように, ルートノードから次第に「幅」を広げていき, 終端ノードに向けては徐々に狭めていく, ひし形のような DAG を基本として, 幅の範囲内で各ノードからランダムにエッジを張ったものである. もう一方は, 中間ノードはどれも次数 1 とし, ルートノードから指定された幅数まで分岐し, 終端ノードで幅数分だけ集約するような, 長方形のような DAG を基本として, 幅の範囲内で各ノードからランダムにエッジを張ったものである. DAG は最大幅数 w, 段数 s, 平均出次数 d を指定し作成した. 図 1 において, 上段は $w = 3$, $s = 5$ の場合の「ひし形」に相当し, (a1) は $d = 1$ の基本形であり, (a2) は $d = 2$ の場合の一例である. 同じく下段は $w = 5$, $s = 5$ の場合の「長方形」に相当し (a1) が $d = 1$ の基本形, (a2) が $d = 2$ の場合の一例である.

*1 http://www.kecl.ntt.co.jp/icl/lirg/ribes

*2 `-output-search-graph-extended` オプション

表 1: 実験結果

1 ノード決定の試行回数	n=1	n=10	n=100	n=1,000	n=10,000	n=100,000	n=1,000,000
近似オラクル文の生成							
両方向平均スコア	**0.6075**	**0.7335**	**0.7923**	**0.8211**	**0.8276**	**0.8307**	**0.8345**
片方向平均スコア	0.5879	0.7239	0.7892	0.8189	0.8265	0.8301	0.8317
貪欲法によるスコア	0.6875						
両方向平均時間 (秒)	1.37	1.73	1.62	2.01	6.52	54.25	564.34
片方向平時間 (秒)	0.72	1.10	0.97	1.39	5.73	52.58	545.13
人工問題 (ひし形 DAG)							
両方向平均スコア	**0.1506**	**0.1797**	**0.1886**	**0.1891**	**0.1893**	**0.1891**	**0.1881**
片方向平均スコア	0.1426	0.1773	0.1845	0.1845	0.1842	0.1839	0.1841
人工問題 (長方形 DAG)							
両方向平均スコア	**0.1020**	**0.1172**	0.1243	0.1259	0.1261	0.1260	0.1258
片方向平均スコア	0.0988	0.1171	**0.1253**	**0.1272**	**0.1274**	**0.1272**	**0.1272**



(a1) (a2)

(b1) (b2)

図 1: 人工問題のための DAG

色のつけ方は DAG をトポロジカルソートした上で, 指定された色数に分けることで行った.

「ひし形」「長方形」のいずれについても以下の 21 種類の条件を変えた DAG を作成した. 段数 s と最大幅 w は常に同じ値で 20,30,50,100 の 4 通り, 平均出次数 d は 2,3,4 の 3 通り, 色数は 20 色と段数 s と同じ値の 2 通り (ただし段数 20 の DAG については 1 通り). これら 21 通りの組合せについてそれぞれについて 20 ずつ計 420 の DAG を作成した. なお, ひし形の場合, $w = s$ であっても, 実効的な幅は $\min(w, \lceil s/2 \rceil)$ となる.

試行数はオラクル文同様の 7 通りとした.

4.3 評価結果と考察

実験の結果を表 1 に示す. スコアはいずれも平均スコアで, オラクル文は 2,000 文の平均, 人工問題は 420 個の DAG の平均である. 近似オラクル文生成と, 人工問題においては試行回数によらず提案した両方向からの MCTS が最も高いスコアを得ていることが分かる. また, 近似オラクル文生成において, 片方向, 両方向問わずに $n \geq 10$ であれば貪欲法によるベースラインよりも高いスコアでった. これから, オラクル文生成において MCTS が有効であると言える.

一方, 長方形 DAG を取り扱った人工問題においては, 必ずしも両方向 MCTS が良い結果ではなく, $n \geq 100$ ではすべて片方向が良い結果になっている. この原因として, 長方形 DAG はルートの出次数と終端の入次数が極端に多いため, これらの選択が難しい問題となっていることが挙げられる. たとえば, $w = s = 100$, $d = 2$ の長方形 DAG では, ルートの出次数と終端の入次数は 100 であり, その他のノードは平均次数 2 となる (長方形ではそれぞれ 2, 2). このため, 片方向の時であれば難しい選択が最初の 1 度だけであるのに対し, 両方向の場合には最初の 2 回が難しい選択となるためスコアが下がっていると考えられる. この問題への対策として, 提案した両方向 MCTS では, 毎回交互に選択する方向を変更しているが, 次の候補の次数に応じて選択する方向を変更するような方法が考えられる.

実行時間について, 両方向 MCTS ではノードの確定の度に 2 度の深さ優先探索を行うため試行回数によらず片方向の場合よりも長い時間がかかっている. しかし, 試行回数が増えるにつれて計算時間の支配的要因は各試行となる. 実際, $n = 100$ 万の場合での実行時間の差は 4%未満であり十分無視できると考える.

5. まとめ

文生成を題材として, DAG に対する MCTS の適用について述べた. さらに, 文は必ず文頭と文末を持ち, 生成候補集合はルートと終端がそれぞれただひとつの DAG として表現できるという特色を活かし, 両方向からの MCTS を提案した. RIBES スコアを最大化するような近似オラクル文生成, 色分けされた DAG からの多様性を考慮したパス選択という 2 種類の実験を行った. 文生成においては, MCTS を用いた方法が貪欲法によるベースライン手法の結果を大きく上回り, また, 両方向からの MCTS が試行回数によらず片方向からの MCTS の結果を上回ることを確認した. 一方, 人工問題においては, DAG の形状によっては必ずしも両方向 MCTS が有向に機能しないことが分かった. 今後は DAG の形状に応じたノードの選択戦略を検討したい.

参考文献

[1] 平尾 努, 磯崎 秀樹, kevin duh, 須藤 克仁, 塚田 元, 永田 昌明: Ribes: 順位相関に基く翻訳の自動評価法. 言語処理学会年次大会, pp. 1115–1118, 3 月 2011.

[2] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[3] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. Moses: open source toolkit for statistical machine translation. ACL '07, 2007.