自然言語処理 74－10
（1989. 9. 21）

発見的グラフ探索法を用いた日本語形態素解析

長尾健司　　菅野祐司
松下電器産業（株）　情報通信東京研究所

グラフ探索のアルゴリズムA＊を、探索コストが、最小コスト＋α（αは任意の正定数）以下の範囲で、m番目（mは任意整数）までのパスを取り出すことができるように拡張し、これをベースに、高速な形態素解析を実現した。この形態素解析アルゴリズムの時間計算量は、入力文字列長をLとすると、O（L^2）となり（αやmに無関係ゆえ）求める解の範囲を大きくしても、処理の負荷は大きくならないという特徴がある。本稿では、合わせて、本アルゴリズムをもとに、文節数最小法を実現し、最小文節数の解析をただ一つ取り出す場合と、最小文節数＋1以内の解析を全て取り出す場合の実験を行い、処理時間がさほど変わらないことを確認し、アルゴリズムの有効性を立証する。

A JAPANESE MORPHOLOGICAL ANALYSIS USING HEURISTIC SEARCH ALGORITHM

Kenji NAGAO　　Yuji KANNO

Tokyo Information and Communications
Research Laboratory,
Matsushita Electric Industrial Co., Ltd.
3－10－1 Higashimita, Tama－Ku,
Kawasaki, Kanagawa 214, Japan

An efficient algorithm for Japanese morphological analysis has been developed based on a heuristic graph search procedure which is a new extension of the A*. This extension enables to find n paths of which costs are the least and less than or equal to the minimal cost + α, where n and α are a given integer and an arbitrary positive value, respectively. This algorithm runs in $O(L^2)$ steps for an input Japanese sentence of which length is L. The advantage of this algorithm is its speed since the running time to find n paths is comparable to the time for finding the minimal one. This is expected from the fact that the function, $O(L^2)$, is independent of n or α. The experiments have been carried out on a system in which the costs of graphs are determined by Minimum Pause Group Method.

1．はじめに

筆者らは、種々の日本語処理応用システムの中枢部に組み込めるような汎用の日本語処理基本システムの開発を進めている[1]。システムの基本的構成（解析システムのみ）を図1に示す。

モジュール構成は、一文単位に、形態素解析が終了した後で結果を受け取って構文解析を行うというもっともよく見られる処理手順に基づいたものとなっている。特徴は、この間に受け渡される結果のデータ構造と、これを前提とした処理のアルゴリズムにある。

即ち、日本語の様々なレベルでの曖昧性を強く意識して、受渡しのデータである形態素解析結果をグラフにより表現している。さらにこのグラフを出力する形態素解析は発見的なグラフ探索のアルゴリズムを用い、グラフを入力として受け取る構文解析では、構文解析固有の曖昧性と形態素解析結果の曖昧性を統一的に扱える独自のアルゴリズムを開発し[3]、それぞれ処理の効率化をはかっている。

本稿では、このうちの日本語形態素解析システムの探索アルゴリズムとその処理効率について報告する。

一般に、漢字かな混じりの入力文字列に対する辞書検索の結果は複数あり、通常の形態素解析で行なうような接続ルールを用いた絞り込みでは、これを一通りに決めることはできない。これが形態素解析レベルでの曖昧性である。

ここで、接続ルールを絶対的なものとして受け入れてしまうと、形態素解析は、入力文字列と日本語辞書及び接続ルールによって決まる探索グラフの中で、経験的な知識等に照らしてもっともらしい幾つかの解析に対応するサブグラフを探索するという問題として捉えることができる。

この捉え方に素直に基づくならば、この探索にはグラフ探索のアルゴリズムを用いることができる。さらに、ここで形態素解析という問題領域を考えると、探索においては発見的な知識を利用することが可能であり、より効率化をはかることができることに気付く。

そこで、本形態素解析システムでは、発見的なグラフ探索の手法Ａ＊をもっともらしい解パスよりなるサブグラフを取り出せるように拡張し、これをベースにしたアルゴリズムを用いる。

2．Ａ＊アルゴリズムの拡張

Ａ＊は、与えられた探索グラフからコストが最小である任意のパスをただ一つ探索して停止するものであり、現在でも頻繁に利用さ

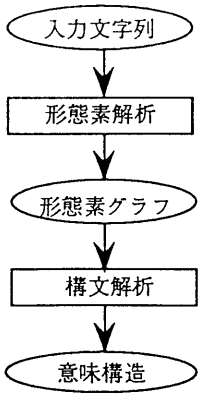

図1　日本語処理基本システム構成

れている。Ａ＊アルゴリズムを表1に示すが、これはオリジナルのもの[4]に対して、Albert Martelliが改良を施し[5]、効率アップをはかったものである。

アルゴリズムの時間計算量として、ノードの選択－展開の回数をとるならば、次のような集合Ｖの要素数をＮ、即ち、Ｎ＝ｄｉｍＶとすると、オリジナルのＡ＊が、最悪の場合ＥＸＰ（Ｎ）のオーダーになるのに対して、このアルゴリズムは、N^2のオーダーに抑えることができる。

（Ⅰ）Ｓ∈Ｖ

（Ⅱ）ｆ（ｎ）＜＝ｆ＊（Ｓ）
かつ　$^{\exists}n_0 \in V$、$n_0 \in P(n)$
ならば　ｎ∈Ｖ■

以下では、この改良されたアルゴリズムをＡ＊と呼ぶ。

さて、拡張の主目的は、

（１）もっともらしいパスよりなるサブグラフを取り出せるようにする。

であるが、サブグラフを取り出すとなると、当然処理の負荷の増大が予想されるので、

（２）（１）に伴う処理の負荷の増大を緩和する。

も要求される。以下では、上の（１）（２）を実現するため、２段階に分けてＡ＊の変更を試みる。第一の変更は、主として、（１）を確実に実現しようとするものである。このアルゴリズムをA^*_{+1}と呼ぶ。第２の変更は、A^*_{+1}を処理効率改善のために、A^*_{+1}をさらに改良するものである。このアルゴリズムをA^*_{+2}と呼ぶ。

2－1．第一の変更：A^*_{+1}の設計

上述のように、ここでは（１）に主眼を置くが、（２）についても考慮する。改良をそれぞれ次のようにリファインする。

表1　A＊アルゴリズム

記号の定義	アルゴリズム
S=ｽﾀｰﾄﾉｰﾄﾞ G=ｺﾞｰﾙﾉｰﾄﾞ（解ﾉｰﾄﾞ） g(n)=ｽﾀｰﾄﾉｰﾄﾞSからﾉｰﾄﾞnまでのコスト h(n)=ﾉｰﾄﾞnからGまでのコストの予測値 h*(n)=ﾉｰﾄﾞnからGまでのコストの正確な最小値 f(n)=g(n)+h(n) f*(n)=g(n)+h*(n) c(■,n)=ﾉｰﾄﾞm,n間のパスのコスト CLOSE=すでに探索されたﾉｰﾄﾞの集合 OPEN=CLOSEの中のﾉｰﾄﾞの継続ﾉｰﾄﾞで、CLOSEに属していないﾉｰﾄﾞの集合（次の段階で探索される候補となっているﾉｰﾄﾞの集合） Γ(n)=ﾉｰﾄﾞnの継続のﾉｰﾄﾞの集合 P(n)=ﾉｰﾄﾞnの親のﾉｰﾄﾞの集合	step 0: Fを十分小さい値にする。 step 1: スタートノードSを集合CLOSEに入れ、すべてのΓ(S)をつくり、これらをOPENにいれる。 step 2: もし、OPENが空ならば、失敗として外に出る。 step 3: 集合OPENの中で最もf(x)の小さいノード$n_{f_{min}}$を見つける。もし、$f(n_{f_{min}})<F$ であれば、$f(n)<F$ であるようなnの中で、g(n)が最小であるようなノードnを候補として選択する。そうでなければ、$n_{f_{min}}$を候補とし、$F=f(n_{f_{min}})$とする。 step 4: 候補ノードnがゴールノードであれば、親ノードとの関係を示すポインタによって、スタートノードからゴールノードまでのパスを見出し、探索を終了する。さもなければ、次にいく。 step 5: ノードnが展開可能であれば、すべての継続ノード$x\in\Gamma(n)$を生成し、$f(x)_{new}$を計算する。ノードnを集合CLOSEに移す。 step 6: CLOSEにもOPENにも入っていない継続ノード、及びCLOSEあるいはOPENに入っているが以前の評価関数の値より新しく計算した$f(x)_{new}$の方が小さいような継続ノードについては、親ノードをnとするポインタをつけ、評価関数の値を$f(x)=f(x)_{new}$としてOPENに移す。それ以外の継続ノードについては棄却する。 step 7: setp 2に行く。

① 最小コスト＋α以下の範囲内で、最小のパスからm番目までのパスを取り出せるようにする。

② 探索の過程に於て、発見的関数が徐々に高精度になっていく場合でもサブグラフを完全に取り出せるようにする。

形態素解析の場合は、文節数最小法等の非常に有効なヒューリスティックがあるため、αは最小コストに比べて小さくてよいが、形態素解析応用システムのことを考慮すると、m番目までの解という指定の仕方も可能である方がよいと考える。

以下にそれぞれの実現について説明する。

【①の実現】

既に述べたように、A＊はコストが最小であるパスをひとつだけ探索するアルゴリズムである。この点は、ｓｔｅｐ４とｓｔｅｐ６で実現されている。ｓｔｅｐ４では、選択された候補ノードがゴールノードであれば、探索を終了する。また、ｓｔｅｐ６では、親ノードの展開によって生成されたノードが既に他の親ノードから生成されているノードである場合は、常にＳからのコストが最小であるパスのみを残すようにしている。即ちＳからのコストが最小でないパス（パス上のノード）は全て棄却するようにしている。

すると、①を実現するためには、以下の２点の変更を行えばよい。。

（ｉ） ｓｔｅｐ４で、選択されたノードがゴールノードであっても停止せず、探索を続ける。

（ii） ｓｔｅｐ６で、新たに生成されたノードが既に生成されＯＰＥＮ或はＣＬＯＳＥ内に存在しても、それが最終的に求めるパス上のノードである可能性があるならば繰り返して有効とする（ＯＰＥＮに入れる）。

（ii）のためには、同一のノードであっても、それを生成した親ノードや、Ｓからのコスト（ｇに相当する）が異なればあたかも別々のノードのように扱われなければならない。この結果、実質的に、探索はノードをｆやｇの値で評価して、選択（探索）するのではなく、ノードと親ノードとｇの値の３つ組を選択することを繰り返すという形を取ることになる。

【②の実現】

①の実現のために（ｉ）を実現すると、ゴールノードが選択される度に、親ノードを逆に

表2　Ａ＊-1

記号の定義	拡張したアルゴリズム
n_p=<node、ancestor、cost> =拡張ﾉｰﾄﾞ=A*のﾉｰﾄﾞに変わって 探索の単位となるもの n_p.node=ﾉｰﾄﾞ n_p.ancestor=np.nodeの 祖先ﾉｰﾄﾞ=ﾉｰﾄﾞ n_p.cost=Sからn_p.nodeまでのｺｽﾄ S_p=<S, NIL, 0>=拡張ｽﾀｰﾄﾉｰﾄﾞ Gp=拡張ｺﾞｰﾙﾉｰﾄﾞ $g_p(n_p)$=n_p.cost $h_p(n_p)$=h(n_p.node)　hsup, Cmin h_p*(n_p)=h(n_p.node) $f_p(n_p)$=$g_p(n_p)$+$h_p(n_p)$ f_p*(n_p)=$g_p(n_p)$+h_p*(n_p) $c_p(\blacksquare_p, n_p)$=c(m_p.node, n_p.node) CLOSEP=すでに探索された 拡張ﾉｰﾄﾞの集合 OPENP=CLOSEP中の拡張ﾉｰﾄﾞの 継続拡張ﾉｰﾄﾞで、 CLOSEに属していない拡張ﾉｰﾄﾞの 集合(次の段階で探索される候補と なっている拡張ﾉｰﾄﾞの集合) $\Gamma_p(n_p)$=拡張ﾉｰﾄﾞn_pの 継続拡張ﾉｰﾄﾞの集合 $P_p(n_p)$=拡張ﾉｰﾄﾞn_pの 親拡張ﾉｰﾄﾞの集合 α=ﾕｰｻﾞの指定するﾊﾟﾗﾒﾀで、求めるﾊﾟｽのｺｽﾄの範囲を定める ■=ﾕｰｻﾞが指定するﾊﾟﾗﾒﾀで、最小ｺｽﾄからm番目までの解ﾊﾟｽを全て求める	step 0:F_pを十分小さい値にkを0に Cminを未設定とする. step 1:S_pを集合CLOSEPに入れ、すべて の$\Gamma_p(S_p)$をつくり、これらを集合OPENPに入れる. step 2:もしOPENPが空ならば、kが0のままなら 失敗として外に出る、そうでない場合は探索を終了する.OPENPが空でないならstep3に行く. step 3:集合OPENPの中で最も$f_p(x)$の小さい拡張ﾉｰﾄﾞn_{pfmin}を見つける.もし、$f(n_{pfmin})<F_p$であれば、$f_p(n_p)<F_p$であるようなnの中で、$g_p(n_p)$が最小であるような拡張ﾉｰﾄﾞn_pを候補として選択する.そうでなければn_{pfmin}を候補とし、$F_p=f_p(n_{pfmin})$とする. step 4:候補拡張ﾉｰﾄﾞn_pがG_pであれば、 n_{p1}.ancestor=n_{p2}.node n_{p1}.cost =n_{p2}. cost+$c_p(n_{p1}.node, n_{p2}.node)$の関係によって、$S_p$から$G_p$までのﾊﾟｽを見出し、ﾊﾟｽ上のﾉｰﾄﾞにﾏｰｸする。k=k+1とする。 k>=mの場合は探索を終了する. Cminが未設定のままなら$C_{min}=f_p(n_{pfmin})$ とする。 k<mの場合及びn_pがG_pでない場合はstep5に行く。 step 5:拡張ﾉｰﾄﾞn_pが展開可能ならば、すべての継続拡張ﾉｰﾄﾞ$n_{pneu}\in\Gamma_p(n_p)$を生成し、これらについて、$g_p(n_{pneu})$を計算する.C_{min}設定済みの場合は、$f_p(n_{pneu})\leq C_{min}+\alpha$をみたす$n_{pneu}$に対してのみstep6を行なう。 step 6:OPENP或はCLOSEPの拡張ﾉｰﾄﾞの中に、node =n_{pneu}.nodeであるようなものが既に存在する場合、それらのg_pの中で($g_p(n_{pneu})$を含めて)、最小の値をg_{pmin}とすると、$g_p(n_{pneu})\leq g_{pmin}+\alpha$となる場合に限り、同一の値も数えて、m番目までに$g_p(n_{pneu})$が入っている場合は、$n_{pneu}$をopenpに入れ、この結果m番目からあふれた拡張ﾉｰﾄﾞは、棄却する。そうでない場合は、n_{pneu}を棄却する。 これ以外の場合は、n_{pneu}をOPENPに入れる。 n_pをCLOSEPに移す. step 7:step 2に行く.

辿りパスを見いだすことになるが、この際、次の計算によって解のパス上の各ノードｎからゴールまでの正確な最小コストｈ＊がわかる。ｈ＊（ｎ）＝ｇ（Ｇ）－ｇ（ｎ）
但しこれは、同一のノードについては、最初に計算された値のみについて言えることである。従って、探索が進むに連れノードからゴールまでの正確な最小コストがわかっていくわけで、以降の探索にはこの正確な値を利用するのがよいのではないかと考えられる。

これは、発見的関数が真の値に近付いていくことが、無駄な探索の減少につながることを期待するものであり、発見的関数が（スタティックに）高精度であるほど、無駄な探索が少ないというＡ＊の性質からの類推である。

このような探索の途中段階で得られたゴールまでの正確なコストを利用することも含めて、次の条件を定める。

《条件Ｔ》

時刻ｔに於けるノードｎの発見的関数値をｈ（ｎ，ｔ）とすると、ｈは次の条件を充さなければならない。

$h(n, t_1) \leq h(n, t_2) \leq h^*(n)$
$(t_1 \leq t_2)$ ■

条件Tの右辺の不等式により、アルゴリズムは、求めるサブグラフを完全に取り出すことができる。このことは、A*に於ける、実行可能性と同様にして証明することができる。

以上のような変更により得られるA*+1をその記述の中で用いる記号の定義と供に表2に示す。A*の記述で用いた記号は、その定義に従う。

【A*+1の時間計算量】

A*+1の時間計算量について考察する。

《定義》

OPENPからCLOSEPに移された拡張ノードのf_p値の列を順に、

$\$_0 : f_1, f_2, \cdots f_r$

$\$_0$の部分列$\$_1$を

$\$_1 : fi_1, fi_2, \cdots fi_p$

を$fi_1 = f_1$とし、fi_{k+1} $(1 \leq k \leq i_{p-1})$をfi_kのノードがCLOSEPに移されて後、最初に、$fi_{k+1} \geq fi_k$を満足するものとする。■

《系1》

発見的関数を任意の拡張ノードn_pについて、時間的変化無しとした場合、即ち、$h_p(n_p, t) = h_p'(n_p)$とした場合、A*+1の時間計算量は、$m^2 \cdot N^2$以下である。

《lemma1》

$dim\$_1 \leq m \cdot N$

ここで、NはA*の説明のところで定義済み、mはユーザの指定した、求める解の数である。

（証明） fi_kの列は非減少列ゆえ、$f_p(n_p) \in \$_1$であるような$n_p$で、

$n_{p1}.node = n_{p2}.node$

$n_{p1}.cost > n_{p2}.cost$

を充す、n_{p1}, n_{p2}がこの順に繰り返して、OPENPからCLOSEPに移されることはない。

従って$f_p(n_p) \in \$_1$である拡張ノード$n_p$で$n_p.node$が同一のものについては、$n_p.cost$の小さい順に、高々m個選択される。 よって、$dim\$_1 \leq m \cdot N$■

《lemma2》

$\$_1$に$fi_k$が加えられてから$fi_{k+1}$が加えられるまでの間に、CLOSEPに移される拡張ノードのf_pの値を順に

$fi_k = fj, fj_{+1}, fj_{+2}, \cdots, fj_{+1} = fi_{k+1}$ とすると、

$l \leq m \cdot N$

である。

（証明） fj_{+r} $(0 \leq r < l)$ に対応する拡張ノードn_{p1}がCLOSEPに移されたとき、$f_p < fi_k$ を満足するf_p値を持つOPENP中の拡張ノードn_pのg_pの中で$g_p(n_{p1})$が最も小さかったわけで、以後、j+l回のステップに至るまでに、

$n_{p2}.node = n_{p1}.node$

なる拡張ノードn_{p2}がCLOSEPに移されても必ず、$g_p(n_{p1}) \leq g_p(n_{p2})$

が成り立つ。

従って、$\$_1$に$fi_k$が加えられてから$fi_{k+1}$が加えられるまでの間に、$n_p.node$が同一のものについては、$n_p.cost$の小さい順に、高々m個選択される。

よって、$l \leq m \cdot N$■

《系1の証明》

lemma1及びlemma2より、

時間計算量$= dim\$_1 \cdot l \leq m^2 \cdot N^2$■

2－2．第2の変更：A*+2の設計

A*+1の設計で行われた2つの変更には、それぞれ次のような問題点がある。

《問題点：Q_1》

A*+1の様に3つ組を探索の単位としてしまうと、同一のノードなのに、親ノードやgの値が異なるがために、別々の探索ユニットとして扱われ、以降生成される子孫ノードの系列は、全く同じであるにもかかわらず、（探索ユニットが異なるため）繰り返して生成されてしまうということが起こりうる。

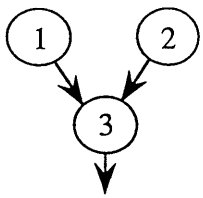

図2 ノードの選択

《問題点：Q_2》

hが条件Tを充しながら正確な値に近づいたとしても必ずしも、選択（探索）回数が減少するとは限らない（実際このことは後の実験により確認される）。むしろ、逆の場合も起こり得る。これは、同じノードを選択する場合でも、選択する順序が異なれば、全体の選択回数が異なってくることがあるからである。例えば、図2に於て、ノード①②③がこの順に大きいf値を持っていて、①→②→③の順に選択されるとすれば、それぞれ一回の選択、合計3回で済むが、①が選択された後で、②のhが条件Tを充す範囲で変化してf値が③のf値より大きくなったとすると、①→③→②→③と合計4回になってしまう。

表3　A*+2（A*+1の変更点）

step4:拡張ﾉｰﾄﾞ n_pがG_pであれば、CLOSEPの中をS_pの方に遡り、n_{p1}. ancestor=n_{p2}. nodeなる親拡張ﾉｰﾄﾞとの関係を基に、$g_p(n_{p2})+c_p(n_{p1},n_{p2})+h_p*(n_{p1})+\leqq C_{min}+\alpha$を充す拡張ﾉｰﾄﾞ全て辿ってﾏｰｸする、この際、h*が、n_{p1}について設定済みで、かつn_{p2}について未設定の場合に、$g_p(n_{p2})+c_p(n_{p1},np2)+h_p*(n_{p1})=f_p*(n_{p1})$の関係がある場合のみ、$h_p*(n_{p2})=f_p*(n_{p1})-g_p(n_{p2})$を計算して保持しておく。そうでない場合は、$h_{psup}(n_{p2})=f_p*(n_{p1})-g_p(n_{p2})$を計算以前の値より小さい場合はこの値を保持しておく。

step6:OPENP或はCLOSEPの中に、n_{pnew}. node=n_{pold}. nodeなる拡張ﾉｰﾄﾞn_{pold}が既に存在する場合、これらのg_pの中で最小の値をg_{pmin}、その拡張ﾉｰﾄﾞをn_{pmin}とする。ここで、以降の記述の簡単のために、つぎの3つの命題をさだめる。

(a)$g_p(n_{pnew})\leqq(g_{pmin}+\alpha)$

(b)C_{min}設定済みの場合は、$f_p(n_{pnew})\leqq C_{min}+\alpha$

(c)同一ﾉｰﾄﾞを持つものの中で同一の値も数えて、$g_p(n_{pnew})$が最小のものからm番目までに入っている。

n_{pmin}がCLOSEPの中に在って、既にﾏｰｸされている場合はstep(6-1)を実行する。ﾏｰｸされてない場合はstep(6-2)を実行する。そのようなn_{pold}が存在しない場合は、step(6-3)を実行する。step(6-1):$h_p*(n_{pold})$が既にわかっている場合step(6-1-1)を実行する。わかっていない場合はstep(6-1-2)を実行する。

step(6-1-1):$g_p(n_{pnew})+h_p*(n_{pold})\leqq m_0+\alpha$かつ、(c)を充す場合は、$n_{pnew}$を起点として、Spに至るまで、親拡張ﾉｰﾄﾞを遡り、解ﾊﾟｽを見いだす、これはstep4でG_pを起点として遡る場合と全く同じである。上式を充さない場合は、n_{pnew}を棄却する。

step(6-1-2):$g_p(n_{pnew})<g_{pmin}$の場合、step(6-1-2-1)を実行する。そうでない場合はstep(6-1-2-2)を実行する。

step(6-1-2-1):解ﾊﾟｽを見いだすと同時にﾊﾟｽ上の各拡張ﾉｰﾄﾞのh_{psup}を計算、保持し、n_{pnew}をOPENPに入れる。n_{pmin}がOPENP中に存在していた場合は、CLOSEPに移す。この結果、(a)かつ(b)か(c)から漏れた拡張ﾉｰﾄﾞは棄却する。

step(6-1-2-2):$g_p(n_{pnew})+h_{sup}(n_{pold})\leqq m_0+\alpha$かつ(c)の場合、step(6-1-2-2-1)を実行する。そうでない場合は、step(6-1-2-2-2)を実行する。

step(6-1-2-2-1):解ﾊﾟｽを見いだすと同時にﾊﾟｽ上の各拡張ﾉｰﾄﾞのh_{psup}を計算、保持し、n_{pnew}のCLOSEPに入れる。

step(6-1-2-2-2):n_{pnew}をCLOSEPに入れる。

step(6-2):a)かつb)かつc)の場合、特に、$g_p(n_{pnew})<g_{pmin}$の場合は、n_{pnew}をOPENPに入れ、n_{pmin}がOPENPに存在していた場合はn_{pmin}をCLOSEPに移す。$g_p(n_{pnew})\geqq g_{pmin}$の場合は、$n_{pnew}$を即座にCLOSEPに移す。この結果、a)かつb)かつc)の条件から漏れた拡張ﾉｰﾄﾞは棄却する。n_{pnew}がa)かつb)かつc)を充さない場合は、棄却する。

step(6-3):n_{pnew}をOPENPに入れる。

《解決策》

Q_1,Q_2のようなことがおこるのは、探索の単位を<ノード、親ノード、g>として、ノード、親ノード、gのどのひとつでも、異なれば全く違うものとしていることが原因であり、結局はQ_1とQ_2の本質は同じである。

そこで、A*+1に対して、次のような改良を加える。

（1）同一のノードを有する拡張ノードが既に生成されている場合は、たとえこれが最終的に解パスに含まれる可能性のある場合でも、必ずしも、新たに探索候補とはしない（拡張ノードをOPENPに入れない）。最もgの小さい拡張ノードのみを成長させるようにする。そして、解パスを取り出すときに、はじめてgや親ノードの違いを考慮して漏れなく解パス上のノードを拾い出すようにする。

（2）既に解パス上のノードだとわかっている拡張ノードは、生成された時点で解パスに含まれることがわかる場合があるので、その場合その子孫の成長を止めて、解パスの取り出しにかかる。

変更は、A*+1のstep4とstep6に対してだけである。こうして得られるA*+2を表3に示す。

【A*+2の時間計算量】

A＊+2の時間計算量について考察する。

【系２】

発見的関数を任意の拡張ノードnpについて、時間的変化無しとした場合、即ち、hp（np, t）＝hp′（np）とした場合、A＊+2の時間計算量は、N^2以下である。■

証明は省略するが、系１と類似の展開によって示される。

系２によれば、A＊+2では求める解の範囲を大きくしても、処理の負荷はさほど大きくならないことが期待できる。

３．形態素解析アルゴリズム

以下に、A＊+1やA＊+2をベースにした形態素解析アルゴリズムを示すが、データ構造の対応と、発見的関数として如何なる情報を用いるかということ、それから時間計算量について示せば十分だろう。

【データ構造の対応】

表４に示す。

表４　データ構造の対応

ﾉｰﾄﾞ=>(文中左位置、文中右位置、品詞ﾎﾟｲﾝﾀ)
文中左位置=入力文字列中の位置
文中右位置=入力文字列中の位置
品詞ﾎﾟｲﾝﾀ=文中左位置と文中右位置の間に挟まれた文字列に対応する単語で、品詞を同じくするものの情報を格納するﾃﾞｰﾀ構造へのﾎﾟｲﾝﾀ
ｽﾀｰﾄﾉｰﾄﾞ=>(0, 0, NIL)
ｺﾞｰﾙﾉｰﾄﾞ=>ﾉｰﾄﾞの文中右位置が入力文字列長と一致するもの.
ﾊﾟｽのｺｽﾄ=>ﾊﾟｽ上の単語列の中の文節の数や、単語の頻度情報等をもとにする。
継続ﾉｰﾄﾞ=>ﾉｰﾄﾞの単語に接続しうる単語よりなる.

【発見的関数】

既に述べた条件Ｔを充すようにすることを念頭において一例をあげる。

ヒューリスティックは文節数最小法を用いる。コストはパス上の文節の数とする。初期値の設定には、入力文字列を末尾から先頭に向かってスキャンし、各文字位置で、その文字からはじまる自立語や付属語列の最大長についての予め用意した情報を基に、末尾までの文節数の予測値を計算すればよい。解パスが見いだされ、パス上のノードからゴールまでの正しい最小コストがわかった後は、その値を用いる。

【形態素解析アルゴリズムの時間計算量】

入力文字列の各位置から始まる単語の品詞の数は有限個、Ｋ以下とすると、入力文字列長をＬとすれば、Ｎ≦Ｋ・Ｌであるから、発見的関数を一定とすると、系１よりA＊+1に基づく場合は、$K^2 \cdot m^2 \cdot L^2$以下、系２よりA＊+2に基づく場合は、$K^2 \cdot L^2$以下となる。

４．評価実験

A＊+1、A＊+2をベースにした形態素解析アルゴリズムの性能評価を行うため、比較実験を行う。文節数が最小の解析、最小＋１以内の解析をすべて取り出すという実験で、実際に解析に要した処理時間、ノードの選択回数を測定する。

《解析対象文》

新聞記事（漢字仮名混じり）５０文

《実験機種・言語》

ＳＵＮ３－６０上のＣ言語

《比較アルゴリズム》

表５に示す。

表５　比較アルゴリズム

X	A*+1で全てのﾉｰﾄﾞについて h=0 とする
Y	A*+1でhを初期値のまま一定とする
Z	A*+1でh*が求まった後はh=h*とする
V	A*+2をﾍﾞｰｽにする
W	唯一つだけ解パスを見い出して停止するように m=1 とする。

Ｘから、発見的探索を用いないで通常のダイナミック・プログラムにより実現した場合を類推することができる。また、Ｗは、Ａ＊アルゴリズムをもとに最小コストのパスをただ一つ見いだして終了する場合に等しい。

《発見的関数の初期値》

４．で例として示したものとする。

【実験結果・考察】

実験結果を図３、図４にグラフで示す。

図３、図４は、それぞれ、入力文字列長に対する処理時間ｔ（１/６０SEC）、ｃｌｏｓｅ（ＯＰＥＮＰからＣＬＯＳＥＰに移すこと）の回数ｒ、の常用対数値を示したものである。但し図中の数字はそのままの値を示している。

グラフよりＶ（Ａ＊+2に基づく）が圧倒的に優位であることは明白である。文長Ｌが大きくなるほど、この傾向は大きくなり、最小文節数＋１以下の解析を全て求める場合、Ｌ＝８０（解パスの数＝３１５）では、処理

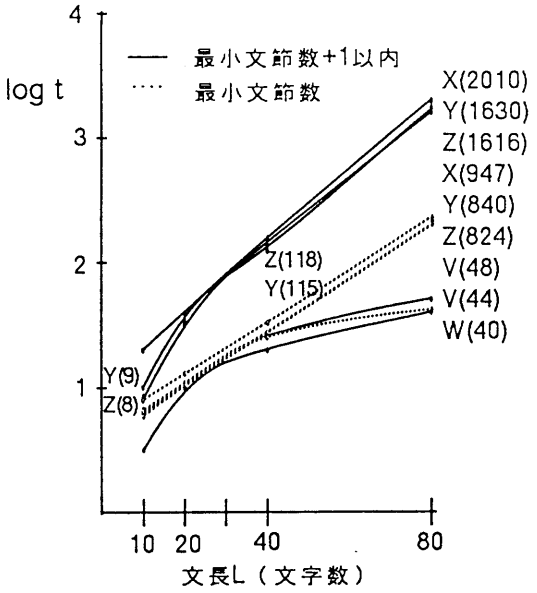

図3 処理時間(t)-文長(L)

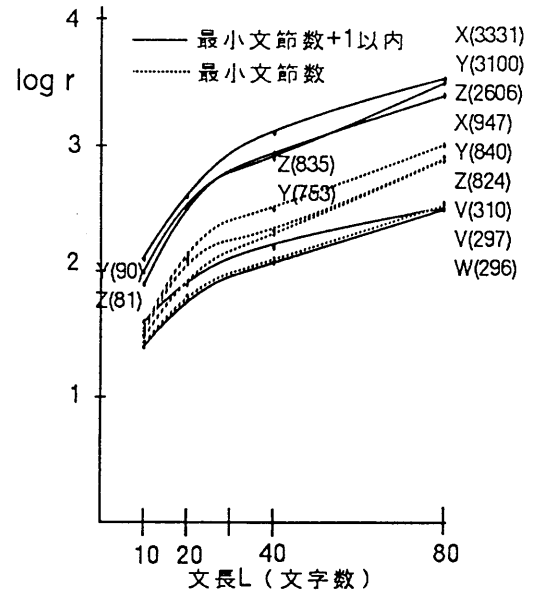

図4 クローズ回数(r)-文長(L)

時間においてVはXの約５０倍、Zの４０倍、ｃｌｏｓｅの回数でもXの１０倍、Zの８倍の効率となっている。又、Vは最小文節数の解析、最小文節数＋１以内の解析を取り出す場合で、処理時間、及びｃｌｏｓｅの回数において差異がない。しかも、VはWに比べても、処理時間、クローズの回数ともさほど劣ることはない。即ちVは、最小コストの解パスをただ一つ求めて停止する場合とほとんど同じ処理時間で終了する。

これは、かなり驚くべき結果であるが、アルゴリズムを比較すると、納得できる部分も多い。即ち、X、Y、Zでは、親ノードから子ノードが生成されるとき、子ノードが既に他の親或いは、異なったｇ値をもって既に生成されている場合でも、繰り返して子ノードをＯＰＥＮＰに入れることがあるが、Vでは、新しい子ノードのｇ値が古い子ノードの最小ｇ値よりも小さくないかぎり、子ノードを再びＯＰＥＮＰに入れることがない。従って、求めるコストの範囲（αやm）が大きければ大きいほど、この効果は大きくなる。

また、Ｌ＝４０のときにYがZを上回る効率を示したのは、既に述べたように、発見的関数が条件Ｔを充しながら変化するときに、ノードの選択回数が増えてしまうことも有り得るからである。

５．おわりに

１．グラフ探索のアルゴリズムＡ＊を探索コストが、（最小コスト＋α）以下の範囲内で、m番目までの解パスを出力できるように拡張し、これをベースに高速な形態素解析アルゴリズムを設計した。

２．このアルゴリズムの時間計算量は、入力文長をＬとするとＯ（L^2）となり、αやmに無関係であることを示した。

３．さらに、本アルゴリズムによれば求める解の範囲を大きくしても、処理時間がほとんど増加しないことを実験で確認し、２．を立証すると共に、アルゴリズムの有効性を示した。

参考文献

１）長尾：日本語処理基本システム（１）情報処理学会第３７回全国大会

２）長尾：文節数最小法・形態素解析の実現について　情報処理学会第３５回全国大会

３）菅野：曖昧さの効率的処理のための構文解析手法について　本研究会

４）PETER E. HART, et al. : A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE TRANSACTIONS ON SYSTEMS SCIENCE AND CYBERNETICS., 7, 1968, pp100-107

５）Alberto Martelli : On the Complexity of Addmissible Search Algorithms, Artif. Intell., 8, 1977, pp1-13