プライオリティ制御機構を有するOR並列Prologの分子系統樹作成への応用

松田秀雄† 金田悠紀夫†

OR並列で実行されるゴールに優先順位としてプライオリティを付加し，このプライオリティによりゴールをスケジュールするプライオリティ制御機構を持つ並列Prologの分子系統樹作成への応用について述べている．プライオリティ制御機構により，代表的な探索アルゴリズムの一つであるA*アルゴリズムをOR並列実行環境で容易に記述することができる．共有メモリ型並列計算機上で並列実行を行なったところ，問題によっては26プロセッサで40倍とスーパーリニアの台数効果が得られた．以上により，本方式の有効性が示された．

An Application of an OR-Parallel Prolog System with a Priority Control Mechanism to Molecular Phylogenetic Tree Construction

HIDEO MATSUDA† and YUKIO KANEDA†

We applied our OR-parallel Prolog system to a practical search problem, construction of molecular phylogenetic trees. The system provides a priority-control mechanism which makes it possible to implement efficient heuristic search algorithms, such as A* algorithm, in OR-parallel execution environment. This mechanism enables us to develop an algorithm to construct phylogenetic trees based on A* algorithm. The effectiveness of the method is demonstrated by measuring execution times on a tight-coupled multiprocessor machine.We attained a maximum of 40-fold (super-linear) speedup on 26 processors.

1. はじめに

探索は人工知能を代表とする多くの分野で問題解決の普遍的な手法となっており，これらの分野では探索効率の向上は重要な意味を持っている[1]．

OR並列[2]は，実行可能な複数の候補節を並列に呼び出して実行していくもので，探索問題で特に効果を発揮する．しかし，単純に全ての呼出しを並列に行なうと並列度が爆発的に増え，プロセス切替えなどのオーバヘッドの増大により台数効果が抑えられてしまう．これを解決するためには，並列処理の効果が高い部分のみを並列に実行し残りは逐次に実行するなど，プログラムの特性に応じたスケジューリングを行なう必要がある．このようなスケジューリングを行なっている処理系としてはAurora[3]，Muse[4]などがある．

これに対して，著者等は，OR並列で実行されるゴールに優先順位としてプライオリティを付加し，このプライオリティによりゴールをスケジュールする処理系を開発した[5]．ある評価関数を最大に（または最小に）するような最適解を求める問題では，この評価関数に基づいてゴールにプライオリティを設定すれば，処理系のプライオリティ制御機構により最適解に近いゴールのみが選択されて実行される．これにより探索効率を向上させることができる．

プライオリティ制御機構を持つOR並列Prologの応用として，実用的な探索問題の一つである，分子系統樹（molecular phylogenetic tree）の作成を行なった[6),7]．分子系統樹とはDNAやRNAの塩基配列など分子レベルのデータを使用してつくられた生物または遺伝子の系統樹である．系統樹作成法としては最尤法[8]を選んだ．最尤法は，可能な候補系統樹を生成し，そのそれぞれについて分子レベルの進化をモデル化した確率モデルを基に尤度を計算し，その値が最大のものを解とする方法である．

最尤法は確率モデルにより尤度最大の系統樹を作成

† 神戸大学工学部情報知能工学科
Department of Computer and Systems Engineering, Faculty of Engineering, Kobe University

するので，生物または遺伝子ごとの配列データを単純に比較する方法に比べて，より正確な系統樹の作成が行なえるとされているが，比較すべき配列データの数が増えると可能な候補系統樹の数が急激に増え，膨大な処理時間を必要とするのが欠点であった．そこで，著者等は，候補系統樹の探索をOR並列で行ない，これをプライオリティで制御するアルゴリズムを開発した．このアルゴリズムは，後述するように，代表的な探索アルゴリズムの一つであるA*アルゴリズム[9]を並列化したものと見なすことができる．

以下では本処理系による探索アルゴリズムの記述と，本処理系を分子系統樹の作成に応用した結果について述べる．

2. OR並列におけるプライオリティの設定

Prologの実行過程は，ユーザが入力した最初のゴールを根として，実行の各時点でのゴールを節点とする探索木の展開ととらえることができる．OR並列は，この探索木を根から葉に向かって幅優先で並列に展開する方式である．OR並列は，与えられた条件を満たす解を全て求めるような問題で特に有効であるが，ある評価関数を最大に（または最小に）する最適解を求めるときには，通常の逐次Prologと同様setof，bagofといった組込み述語により解の集合を求めた後，それらの中から最適解を選択することになる．しかし，探索木が非常に大きく広がるような問題では結果的に無駄な処理を数多く行なう可能性が高く，実行時間，メモリ消費ともに膨大なものとなってしまう恐れがある．

そこで著者等はOR並列での効率の良い最適解探索のため，プライオリティによる実行制御を提案した[5]．プライオリティは，概念的には探索木の各枝についたラベルであり，その枝の下のゴールの実行の優先度を表す．全体の実行はスケジューラにより管理され，プライオリティの高い枝から順に選択されて，その下のゴールが実行される．これにより探索木の展開は理想的には常に最適解の方向に向かうことになり，無駄な展開を抑えることができる．

プライオリティは次の3つの組込み述語により設定される．

setPriority（プライオリティ値）
upPriority（プライオリティ増加値）
downPriority（プライオリティ減少値）

setPriorityは引数の値（整数）を新しいプライオリティとして設定する．upPriority，downPriorityはそれぞれ現在のプライオリティに引数の値を加えるかまたは引いたものを新しいプライオリティとして設定する．

3. 探索アルゴリズムの記述

3.1 A*アルゴリズム

代表的な探索アルゴリズムの一つにA*アルゴリズムがある[9]．A*アルゴリズムは最良優先探索アルゴリズムの一種であり，問題固有の知識を使って探索効率を向上させるのが特徴である．

例えばグラフの最短経路を求める問題の場合，A*では，任意の点 p について開始点から点 p に至るコストを表す関数 $g(p)$ に加えて，p からゴール点に至るためのコストの推測値 $\hat{h}(p)$ を用いる．そして，点 p における評価関数 $f(p)$ を，$f(p)=g(p)+\hat{h}(p)$ とし，$f(p)$ の小さいものから優先的に探索していく．

具体的には，A*アルゴリズムでは，探索途中の点の情報をOPEN，CLOSEという2つのリストに保持しながら，以下のように探索を進める．

(1) OPENに開始点を入れる．
(2) OPENが空なら失敗終了．
(3) OPENから $f(p)$ を最小にする点 p を取り出し，CLOSEに入れる．
(4) p がゴール点なら成功終了．
(5) p のすべての継続点を生成し，それぞれについて f を計算する．
(6) 継続点のうち，OPEN，CLOSEのいずれにも含まれないものをOPENに入れる．
(7) 継続点のうち，OPEN，CLOSEのいずれかにすでに含まれるものについて，f の旧値と(5)で求めた新しい f の値とを比較し，小さい方と組み合わせる．このときCLOSEに含まれていたものがより小さい f の値を持った場合にはOPENに入れ直す．
(8) (2)に戻る．

A*の性質として，$\hat{h}(p)$ をゴール点に至るコストの下限値に選べば，解がある限り常に収束し，しかもコスト最小解が得られることが知られている．

3.2 プライオリティ制御機構による記述

A*でのOPENリストはコストが小さい順に点の情報を並べた待ち行列で表現できる．前述のプライオリティ制御機構を使えば，プログラム中で明示的にOPENリストを表現しなくても，評価関数に基づいてプライオリティを設定するだけでA*と同等の探索

アルゴリズムを記述できる．

後述するように本処理系は，共有メモリ上に実行可能キューを置き，実行可能なゴールをプライオリティ順に並べて保持する．実行可能キューは処理系が管理するので，ユーザはプライオリティ順の並べ替えなどのキュー操作を記述する必要がない．

したがって，プライオリティ制御機構があれば，グラフを探索しながら，現在の点までのコストとその点からゴール点までのコストの推測値の和を求め，それに負号をつけたものをプライオリティとして設定するだけで，コストの小さいものから優先的に探索するよう記述することができる[10]．

4. 分子系統樹の作成

実用規模の探索問題として分子系統樹（molecular phylogenetic tree）作成問題をとり上げた．分子系統樹とはDNAやRNAの塩基配列など分子レベルでのデータを使用してつくられた生物または遺伝子の系統樹である．分子系統樹の例を**図1**にあげる．図1は，ラン藻の一種（*Synechococcus* sp.），乳酸菌の一種（*Lactobacillus casei*），枯草菌（*Bacillus subtilis*），リケッチアの一種（*Rickettsia prowazekii*），大腸菌（*Escherichia coli*），メタン細菌の一種（*Methanobacterium formicicum*）の6種のバクテリアの系統樹を示している．図1の枝の長さは，進化の過程で起きる塩基置換の度合を表す．図の下の.10の線が平均0.1塩基の置換に対応する長さを示している．

系統樹の作成には最尤法を使用した．最尤法では，以下のような方法で系統樹を求める[5]．

(1) 3種のDNA配列から系統樹を作成し，種の数を表す変数kに4を代入する．

(2) $k-1$種の系統樹にk番目の種を加えてk種の系統樹の樹形（topology，系統樹の形）を$2k-5$個生成する．それぞれについて進化の確率モデルから尤度が最大になるように系統樹の枝長を計算し，k種の系統樹を構成する．

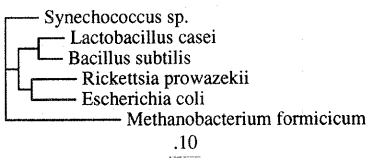


図1 分子系統樹の例
Fig. 1 An example of a molecular phylogenetic tree.

(3) $k<n$の間，kを1ずつ増やしながら(2)を繰り返す．$k=n$になったとき，尤度が最大の系統樹が最終的な系統樹となる．

樹形の数は，単純に生成すると，種の数nのとき，

$$\prod_{k=3}^{n}(2k-5)=\frac{(2n-5)!}{2^{n-3}(n-3)!} \qquad (1)$$

という膨大な数になる．ここで，求めたいのは尤度最大の系統樹であるので，A*アルゴリズムにより尤度を評価関数としてこの値の大きいものを優先的に探索するようにすれば，尤度最大の系統樹を効率良く求めることができると考えられる．

しかし，実際には，単純に尤度そのものを評価関数として設定しても，それほど探索効率を上げることができない．これは，ほとんどの場合，新たに種を加えてできる系統樹は，種を加える前のものと比べて（たとえそれが最適な樹形を持つ樹であっても）尤度が小さくなるからである．

そこで，A*アルゴリズムにおける，解への近さを表す推測値を尤度に足したものを評価関数として設定すれば，さらに探索効率を上げることができる．この推測値はkの関数になっており，kが増えるにつれて（nに近くなるにつれて）単調に増加する．その値は，種を加えて系統樹を拡張していく過程で，最適な樹形を選んでいく限り評価関数の値ががだいたい一定になるように，種ごとの塩基配列データの差分の値から経験的に導き出したものを使用する（差分が大きければ尤度低下が大きくなる）．

5. 並列実行方式

5.1 並列計算機の構成

本処理系は，Sequent社製の並列計算機Symmetry S81上で動作する．Symmetryは，要素プロセッサとしてIntel80386（クロック16MHz）を使用した共有メモリ型の並列計算機である．また，各プロセッサには，浮動小数点演算アクセラレータ（WeitekWTL1167）が付加されている．これにより，浮動小数点演算が80387と比べ約2倍の速度で実行される．なお，使用した並列計算機では28台のプロセッサが実装されている．

Symmetryでは，全ての要素プロセッサが対等であり，一本のバスにより結ばれている．プロセッサ間の

通信はバスに接続された共有メモリを介して行う．バスと共有メモリでの競合を抑えるため各プロセッサには64KBのコピーバック方式のキャッシュがつけられている．OSはDYNIXと呼ばれるUNIXをベースにしたもので，プロセスを単位としてプロセッサへの割当てなどの並列実行管理を行ない，相互排除のためのロック（専用ハードウェアを備えている），プロセス間の共有メモリ割当てなどの機能を提供する．

5.2 実行モデル

本処理系では，PE（Prolog Engine）とタスクという2つの概念でPrologプログラムを並列に実行する[5]．PEはプロセッサを抽象化したものであり，逐次Prolog処理系と同様，プログラムを深さ優先で実行するソフトウェア仮想マシンである．PEはDYNIXから見ればユーザプロセスの1つであり，並列実行に先だってプロセッサ台数以下の個数だけ生成され，次に述べるタスクを処理していく．DYNIXでは実行可能なユーザプロセスの数がプロセッサ台数以下の時には，プロセスとプロセッサとは1対1に対応する．

タスクはゴールの逐次的な実行過程である．本処理系では，OR並列で実行すべき節をプログラム中で並列宣言により宣言することになっている．この宣言がされていない節は全て1つのタスクにより実行される．並列宣言されている節の実行では節の数に応じて複数のタスクが生成され，プライオリティでソートされた実行可能キュー（Ready Queue）につながれる．

実行可能キューは共有メモリ上におかれ，処理系全体で一つだけである．タスクのPEへの割当ては，グローバルなスケジューラではなく個々のPEが実行可能キューを見て，そこから先頭のタスクを取り出すことにより行なわれる．PEは(1)タスクが割り当てられていない，(2)割り当てられたタスクを完了した，(3)割り当てられたタスクより高いプライオリティを持つタスクが新たに実行可能キューにつながれた，のいずれかで，実行可能キューから新たにタスクを獲得してそれを実行する．(3)の条件は，各PEにより2節で述べた組込み述語によるプライオリティ設定時および候補節の呼出しごとにチェックされる．

6. 性能評価

6.1 分子系統樹作成の入力データとプログラム

種々の細菌の塩基配列をもとにそれらの系統樹の作成を行なった．使用した配列は，リボソームの構成要素の一つである16Sリボソーム RNAをアラインメントにより整列させたものであり，長さは1892塩基である．この整列データはリボソーム・データベース・プロジェクト[11]により作成されたもので，イリノイ大学のFTPサイト（rdp.life.uiuc.edu）からanonymous ftpにより入手可能である．

種の数が6から30までの系統樹を，SICStus Prolog (Ver.2.1)[12]による逐次実行と，本処理系によるOR並列実行とで，それぞれ作成した．各々のプログラムを，付録(a)（SICStus Prolog用）と付録(b)（本処理系用）に示す．いずれも，A*に基づいたアルゴリズムを使っているが，(a)では3.1節で述べたOPENリストの操作をPrologのリスト処理機能により実現し，(b)ではプライオリティ制御により実現している．基本的なアルゴリズムが同じため，両者のプログラムの実行結果として得られる系統樹はかならず一致する．なお，文献5)とは，逐次，並列実行ともに全解探索ではなく，A*アルゴリズムによる発見的探索を行なっている点が異なっている．

系統樹の枝長および尤度の計算は大量の数値計算を必要とするため，付録のプログラムでは，文献13)のプログラム（C言語で記述されている）の一部を，本処理系では組込み述語として，SICStus Prologでは他言語インターフェースによりリンクして使用している．なお，付録(b)のプログラムの一番先頭の行は並列宣言であり述語treeEvalをヘッドに持つ節だけがOR並列で実行され，残りの節は逐次に実行されることを示している．また，A*における解への近さを表す推測値は，どちらのプログラムでも述語heuristicにより，あらかじめ与えられているものとしている．

6.2 実行結果

PE1台の実行時間を**表1**に示す．表1では本処理系での実行時間をプライオリティ，SICStus Prologの時間をA*と表している．本処理系の方がやや短い時間で解が求められている．これは，SICStus PrologではOPENリストをPrologのリスト処理機能により操作しているのに対して，本処理系では処理系内部の実行可能キューに置き換えて操作しているためと考え

表1 PE 1台による分子系統樹作成の実行時間（秒）
Table 1 Execution times of molecular phylogenetic tree construction on 1 PE.

種の数	6	10	20	30
プライオリティ	75	490	43729	92665
A*	123	692	45030	94061

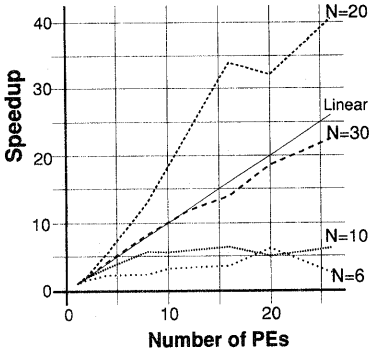


図2 分子系統樹作成での台数効果
Fig. 2 Speedup in molecular phylogenetic tree construction.

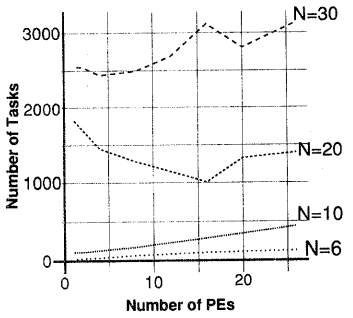


図3 分子系統樹作成での総タスク数
Fig. 3 Total numbers of tasks in molecular phylogenetic tree construction.

られる．

OR 並列での台数効果を図2に示す．台数効果は種の数が少いうちは PE 数台で頭打ちになっているが，20 種と 30 種の一部（PE 台数 4 台から 8 台）で線形以上（スーパーリニア）の台数効果が出ている．これにより，例えば 20 種の PE26 台の実行時間は 1078 秒にまで短縮され，PE1 台の時に比べて 40 倍も実行速度が向上している．

このように大きな台数効果が出た原因を調べるため，生成されたタスクの総数を測定した（図3）．図2と図3から，台数効果と生成タスク数の間に明確な相関があることがわかる．つまり，PE2 台以上でのタスク数が PE1 台と比べて大きいところ（$N=6$，10 および $N=30$ の PE10 台以上）では台数効果が落ちているが，タスク数が小さいところ（$N=20$ および $N=30$ の PE が 10 台より少ないところ）でスーパーリニアの台数効果が得られている．

タスク数についてより詳しく調べるため，20 種の

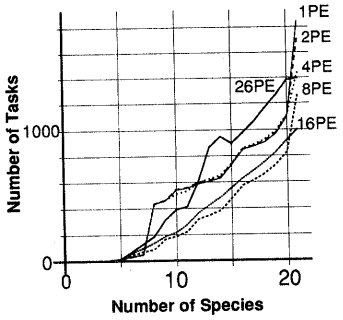


図4 分子系統樹作成でのタスク数の増加
Fig. 4 Increase of the number of tasks in molecular phylogenetic tree construction.

系統樹を作成するとき，最尤法により種の数 k の値を増やしながら中間的な系統樹を構成していく過程でのタスク数の変化を計測した．図4は，個々の k の値での中間的な系統樹のうち，最も尤度の高いものが見つかった時点でのタスク数を表している．なお，図4で，$k=21$ の値は，最終的に解が求められたときの総タスク数を表し，これと $k=20$ の値とが異なるのは，$k=20$ の系統樹が見つかってからも，他の中間的な系統樹からそれ以上尤度の大きい系統樹が生成されないかどうか調べないと，最終的に解と判定できないためである．

PE が 4 台以下のところでは，$k=7$ から $k=8$ に変わるときに大きくタスク数が増加している．実行のより詳細な解析から，これは，$k=7$ で尤度最大だった系統樹から，$k=8$ で尤度最大の系統樹を見つけるのに，いったん $k=15$ まで系統樹の生成が進んでからその系統樹の系列から尤度最大のものが得られないことがわかり，$k=8$ まで後戻りして $k=7$ で尤度が次善の値を持つ系統樹からの生成をやり直しているためであることがわかった．

しかし，PE が 8 台以上の時には，ある PE が $k=7$ で尤度最大だった系統樹をもとに $k=8$ の系統樹（$2k-5=11$ 個）を生成するのと並行して，別の PE が $k=7$ の尤度が次善の系統樹からの生成を行なっており，こちらの方から $k=8$ で尤度最大となる系統樹が生成され，4 台以下のときのようなタスク数の急激な増大が抑えられている．

PE1 台のときでも，尤度最大の系統樹以外に，次善の系統樹からの生成をも並行して行なうようスケジューリングを変更することは可能であり，このときには上述のタスク数の急激な増大を抑えることができ，実行時間を短縮できると考えられる．しかし，常に尤

度が次善の系統樹をも調べるようにすると，余計なものまで探索している可能性もあり，必ずしもよいスケジューリングとは限らない．より探索効率の優れたスケジューリング手法の開発が今後の課題である．

7. おわりに

本稿ではPrologのOR並列実行でプライオリティ制御が行なえるよう拡張した処理系を実用的な探索問題の一つである分子系統樹作成に応用した結果について述べた．プライオリティ制御により，A*アルゴリズムをOR並列実行環境で容易に記述することができ，実行速度もリスト処理で記述したときと比べて向上している．

また，問題によってはPE台数を増やすことにより探索効率が向上し線形以上（スーパーリニア）の台数効果が得られた（PE26台の時，PE1台と比べて40倍の速度向上）が，これはPE台数を増やした時のタスク数の減少によるものであることを示した．この結果をもとに，タスク数をさらに減らすことのできる，より探索効率の優れたスケジューリング手法の開発が今後の課題である．

謝辞 本研究は一部，文部省科学研究費補助金重点領域研究（課題番号04235103および05254209）によっている．

参考文献

1) 諏訪 基，実近憲昭：探索，人工知能学会編，人工知能ハンドブック，pp.24–32，オーム社（1990）．
2) Conery, J.S.: Parallel Execution of Logic Programs, Chap.3, pp.35–61, Kluwer Academic Publishers, Norwell, MA (1987).
3) Lusk, E.et al.: The Aurora OR-Parallel Prolog System, *New Generation Computing*, Vol.8, No.7, pp.243–271 (1990).
4) Ali, K. and Karlsson, R.: Full Prolog and Scheduling Or-Parallelism in Muse, *Int'l J. Parallel Programming*, Vol.19, No.6, pp.445–475 (1990).
5) 松田秀雄，鈴鹿重雄，金田悠紀夫：OR並列Prologにおけるプライオリティ制御機構とその応用，情報処理学会論文誌，Vol.34, No.4, pp.773–781 (1993).
6) Nei, M.: Molecular Evolutionary Genetics Chap. 11, Columbia University Press (1987).（五條堀孝，斎藤成也訳：分子進化遺伝学，培風館（1990））
7) Swofford, D.L. and Olsen, G.J.: Phylogeny Reconstruction, *Molecular Systematics*, ed. Hillis, D. and Moritz, C., pp.411–501,Sinauer Associates,Sunderland,MA (1990).
8) Felsenstein, J.: Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach, *J. Molecular Evolution*, Vol.17, pp.368–376 (1981).
9) Hart, P.E., Nilsson, N.J., and Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Trans. Syst. Sci. Cybern.*, Vol.SSC–4, No.2, pp.100–107 (1968).
10) 松田秀雄，鈴鹿重雄，金田悠紀夫：プライオリティ制御機構を有するOR並列Prologの探索問題への応用，並列処理シンポジウムJSPP'93論文集，pp.55–62 (1993).
11) Olsen, G.J., et al.: The Ribosomal Database Project, *Nucleic Acids Research*, Vol.20, Supplement, pp.2199–2200 (1992).
12) Carlsson, M. et al.: SICStus Prolog User's Manual, Swedish Institute of Computer Science (1993).
13) Olsen, G.J., Matsuda, H., Hagstrom, R., and Overbeek, R.: fastDNAml: A Tool for Construction of Phylogenetic Trees of DNA Sequences Using Maximum Likelihood, *Computer Applications in Biosciences*, Vol.10, No.1, pp.41–48 (1994).

(平成5年 9 月 16 日受付)
(平成5年 11 月 11 日採録)

付録　分子系統樹作成プログラム

(a) A* アルゴリズム

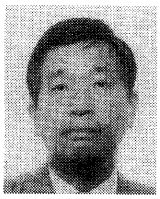
```
go(OptLike) :-
  c_init,
  c_allocTreeArea(NumSpec), c_buildFirstTree(NumTips,Tree),
  astar([Tree,NumTips],[],[[Tree,NumTips]],NumSpec,
        [OptTree,OptNumTips]),
  c_getLikelihood(OptTree,OptLike), c_showBestTree(OptTree),
  c_freeTreeArea.

astar([Tree,NumTips],OPEN,CLOSE,NumSpec,OptTreeRec) :-
  NumTips < NumSpec, !,
  c_buildNewTip(Tree,TreeBuf,NumTrees), NumTips1 is NumTips+1,
  collectTrees(0,NumTrees,TreeBuf,NumTips1,OPEN,OPEN1),
  checkEmpty(OPEN1), maximum(OPEN1,[MaxTree,MaxNumTips]),
  changeClose([MaxTree,MaxNumTips],OPEN1,CLOSE,OPEN2,CLOSE2),
  c_getLikelihood(MaxTree,MaxLike), heuristic(MaxNumTips,Heu),
  MaxScore is MaxLike+Heu,
  astar([MaxTree,MaxNumTips],OPEN2,CLOSE2,NumSpec,OptTreeRec).
astar(OptTreeRec,_,_,_,OptTreeRec).

collectTrees(TreeId,NumTrees,TreeBuf,NumTips,OPEN,OPEN1) :-
  TreeId < NumTrees, !,
  c_allocTreeArea(_), c_treeEvaluate(TreeBuf,TreeId,Tree,_),
  updateOPEN(Tree,NumTips,OPEN,OPEN2), TreeId1 is TreeId+1,
  collectTrees(TreeId1,NumTrees,TreeBuf,NumTips,OPEN2,OPEN1).
collectTrees(_,_,_,_,OPEN,OPEN).

updateOPEN(Tree,NumTips,OPEN,[[Tree,NumTips]|OPEN]).

checkEmpty([]):- !, write('FAILURE TO SEARCH'), nl, fail.
checkEmpty([_|_]).

maximum([Tree],Tree).
maximum([Tree1,Tree2|Set],MaxTree) :-
  maximum([Tree2|Set],MaxTreeSoFar),
  max2(Tree1,MaxTreeSoFar,MaxTree).

max2([Tree1,NumTips1],[Tree2,NumTips2],[Tree1,NumTips1]) :-
  c_getLikelihood(Tree1,Like1), c_getLikelihood(Tree2,Like2),
  heuristic(NumTips1,Heu1),     heuristic(NumTips2,Heu2),
  New1 is Like1+Heu1,           New2 is Like2+Heu2,
  New1 >= New2, !.
max2(_, Tree2, Tree2).

changeClose(_,[],CLOSE,[],CLOSE).
changeClose([MaxTree,NumTips],[[MaxTree,NumTips]|OPEN],CLOSE,
            OPEN1,[[MaxTree,NumTips]|CLOSE1]) :- !,
  changeClose([MaxTree,NumTips],OPEN,CLOSE,OPEN1,CLOSE1).
changeClose(MaxTreeRec,[Tree|OPEN],CLOSE,[Tree|OPEN1],CLOSE1) :-
  changeClose(MaxTreeRec,OPEN,CLOSE,OPEN1,CLOSE1).
```

(b) プライオリティ制御付き OR 並列

```
:- para treeEval/5.

go(OptLike) :-
  c_init,
  c_allocTreeArea(NumSpec),
  c_buildFirstTree(NumTips,Tree),
  makeDenovoTree(NumTips,NumSpec,Tree,OptTree),
  c_getLikelihood(OptTree,OptLike),
  c_showBestTree(OptTree),
  c_freeTreeArea.

makeDenovoTree(NumTips,NumSpec,Tree,OptTree) :-
  NumTips<NumSpec,
  c_buildNewTip(Tree,TreeBuffer,NumTrees),
  NumTips1 is NumTips+1,
  treeEval(NumTips1,TreeBuffer,0,NumTrees,NewTree),
  makeDenovoTree(NumTips1,NumSpec,NewTree,OptTree).
makeDenovoTree(NumTips,NumSpec,OptTree,OptTree) :-
  NumTips>=NumSpec.

treeEval(NumTips,TreeBuffer,TreeId,NumTrees,Tree) :-
  TreeId<NumTrees,
  c_allocTreeArea(_),
  c_treeEvaluate(TreeBuffer,TreeId,Tree,Likelihood),
  heuristic(NumTips,Heu),
  Priority is Likelihood+Heu,
  setPriority(Priority).
treeEval(NumTips,TreeBuffer,TreeId,NumTrees,Tree) :-
  TreeId<NumTrees,
  TreeId1 is TreeId+1,
  treeEval(NumTips,TreeBuffer,TreeId1,NumTrees,Tree).
```

松田 秀雄 (正会員)

昭和34年生．昭和57年神戸大学理学部物理学科卒業．昭和59年同大学院工学研究科システム工学専攻（修士課程）修了．昭和62年同大学院自然科学研究科（博士課程）修了．同年同大学工学部助手となり，現在同大学講師．この間，平成3年4月より10か月間米国アルゴンヌ国立研究所客員研究員．学術博士．論理型言語による並列処理，遺伝子情報処理の研究に従事．IEEE CS，ACM各会員．



金田悠紀夫 (正会員)

昭和15年生．昭和39年神戸大学工学部電気工学科卒業．昭和41年神戸大学大学院電気工学専攻修士課程修了．昭和41年電気試験所（現電総研）入所．電子計算機研究に従事．昭和51年神戸大学工学部システム工学科，現教授．工学博士．コンピュータシステムのハードウェア，ソフトウェアの研究に従事．高級言語マシン，並列マシン，AIに興味を持っている．