実用的な経路計画生成のための時間制約付きヒューリスティック探索

平石広典† 大和田勇人† 溝口文雄†

指定された時間内に準最適な経路を求めるヒューリスティック探索アルゴリズムを提案する．このアルゴリズムは従来の ϵ-近似による A^* を基本にするが，探索中に ϵ の値を動的に変化させることで制限時間に間に合うように探索を終了させることができる．出発点に近いノードは厳密に探索し，目的地点に近いノードは ϵ の値を増加させて粗く探索する．これは移動中の走行車におけるリアルタイムな経路生成に適したものである．カーナビなどで使用される実際のデジタルマップを用いて本アルゴリズムを適用し，アルゴリズムの特徴を実験的に明らかにした．その結果，探索の終了時刻を正確に推定することができ，制限時間を十分に利用しながら最適な解が得られることが分かった．さらにこの特徴はマシンの性能に関係なく成り立つもので，本方法は汎用なリアルタイム経路探索になっている．

Time-constrained Heuristic Search for Practical Route Finding

Hironori Hiraishi,† Hayato Ohwada† and Fumio Mizoguchi†

In this paper, we describe a heuristic search algorithm that finds a provably optimal solution subject to a specified time interval. While this algorithm is amenable to previous works on an approximate A^* search with a bounded error (ϵ), it allows us to terminate the search to retain the specified time interval by changing the value of ϵ during the search. Our basic search strategy is that node expansion around the starting node is pessimistic (exact search), and we accomplish the approximate exploration of nodes around the goal by increasing ϵ. This strategy is suitable for real-time route finding in automobile navigation systems. We conducted our experiments to clarify the practical features of the algorithm, using a digital map of a commercial automobile navigation system. The resulting advantage is that the estimation of the finishing time of the search is quite accurate, and optimal solutions are produced by making full use of the permissible search time, regardless of different machine performances.

1. はじめに

ヒューリスティック探索は人工知能における問題解決の基本的な方法として古くから研究されてきた．それは，あるノードからゴールに到達するのに要するコストを見積もり，その見積りに従って探索経路を決定する方法である．A^* アルゴリズムはそうした方法の代表的なもので，あるノード n の評価値を，出発点から n までの実コストと n からゴールまでの許容見積りコストの和とし，評価値が最小のノードを選択して，最終的にコスト最小の経路を求めるものである．これは最短経路問題をはじめ，様々な問題に適用可能である[6]．

† 東京理科大学理工学部
Faculty of Science and Technology, Science University of Tokyo

しかしながら，A^* アルゴリズムはコスト最小を保証するために，探索すべきノードの数が指数オーダーになることが知られている．そのため，A^* アルゴリズムの様々な変形が提案されてきた．たとえば，探索を効率化するためのヒューリスティック関数の設定[9]，メモリ消費量の少ない探索法[10]，双方向探索[5]など，A^* アルゴリズムに対して様々な工夫がなされてきた．

このような A^* に基づくヒューリスティック探索はオフライン型であり，出発点からゴールまでの経路計画を移動の前に完全に終えておくものといえる．これに対して，Korf らの研究[3),4)]に見られる Real-Time A^* は与えられた定数時間の範囲内での探索結果をもとに判断を下し，実際に移動しながら探索を続けるアルゴリズムになっている．これは解の最適性が保証されないこともあるが，計画と実行を交互に行うという点に特徴があり，移動ロボットやカーナビにおける経

路計画のような現実の問題に対してはオフラインよりも有効である．

従来の A^* アルゴリズムと違って，Real-Time A^* は一度通ったノードへ後戻りするコストも考慮に入れた評価関数を採用しており，後戻りを許すことで探索における完全性を保証している．しかしながら，移動体の最短経路問題を考えると，探索コストよりも実行コストの方がはるかに大きい．たとえば，自動車が一度通った道を後戻りするコストは探索コストとは比較にならないほど大きい．したがって，このような最短経路問題では後戻りは許されず，実行を開始する前に確実に探索を終えておく必要がある．

本論文ではリアルタイムな経路計画を時間的な制約のもとでの最適化問題としてとらえ，指定された制限時間内において，準最適な経路を生成するための探索アルゴリズムを提案する．ここで準最適という意味は ϵ-近似[5),7),13)]を意味しており，評価関数以外は通常の A^* アルゴリズムに基づく．従来の ϵ-近似との違いは，ϵ を固定するのではなく，探索中に ϵ の値を動的に変化させ，制限時間に間に合うように探索を終了させることである．一般に ϵ を大きくすれば探索すべきノードが減るので，探索の制限時間が少なくなれば ϵ を大きくすることでゴールまでの探索をすばやく終了させることができる．

我々の探索戦略は，探索開始直後は ϵ の値を小さくして出発点に近いノードは厳密に探索し，ゴール付近では ϵ の値を増加させで粗く探索するというものである．これは移動体におけるリアルタイムな経路探索に適している．なぜなら，移動体がゴール付近に移動するまでにはかなりの時間がかかり，その間に再度探索して，より最短な経路を生成すればよいからである．ϵ-近似において ϵ を動的に変化させることはすでにPohl[8)]が提案しているが，このような戦略は導入されていない．

本方法の有効性を示す例として，ここではカーナビにおけるリアルタイムな経路探索問題を取り上げる．カーナビでは走行前に決めた経路に完全に従うことは実際には少なく，誤って経路をはずれたり，交通渋滞のために別の道を選択することの方が多い．その場合，走行中において経路を動的に変更する探索方法が必要になる．本論文では，本方法の実用性を明らかにするためにカーナビで実際に使用されているデジタルマップを使って探索を実行した．これに基づいて，探索時間，展開されるノード数，解の質，ϵ の値といったパラメータ間の関係を実験的に明らかにする．また，Korf の Real-Time A^* との比較も行う．

本論文の構成は次のようである．まず，本方法の基礎となっている ϵ-近似探索を 2 章で述べる．3 章に時間制限付きの探索方法，4 章にそのアルゴリズムを示す．5 章は実験結果を分析する．6 章は関連研究を述べ，7 章は結論である．

2. ϵ-近似探索

ϵ-近似探索における探索アルゴリズムは A^* と同様であるが，評価関数 $f(n)$ は次式で与えられるものとする．

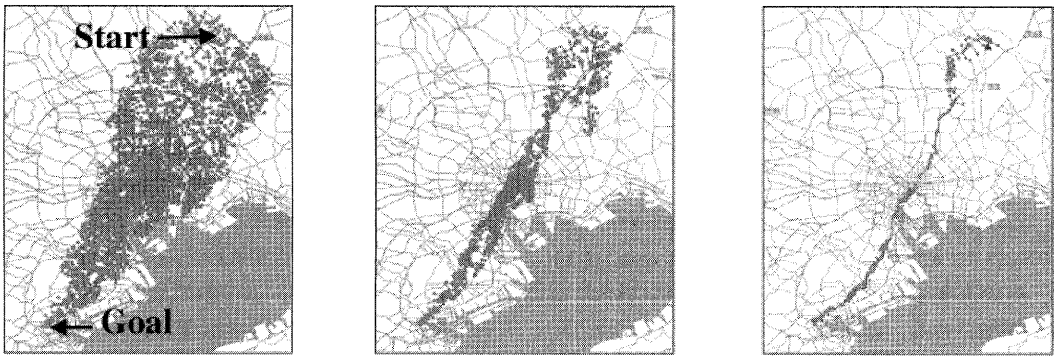
$$f(n) = g(n) + \epsilon h(n) \qquad h(n) \leq h^*(n)$$

ここで，$g(n)$ はスタートノードからノード n までの実コストであり，$h(n)$ はノード n からゴールノードまでに必要とされる予測コスト，$h^*(n)$ はノード n からゴールノードまでの実際のコストである．$h(n) \leq h^*(n)$ なので，$h(n)$ は許容的ヒューリスティックであり，$\epsilon = 1.0$ では $f(n)$ の最小なノードを探索することで最適な経路を求めることができる．

ϵ-近似探索では ϵ の値によって解の質と探索空間が変化する．$\epsilon h(n) > h^*$ のとき，評価関数 f は非単調になり最適性が保証されなくなる．このとき ϵ-近似探索で得られる解の質（経路長）q は，最適解を q^* とした場合，$q^* \leq q \leq \epsilon q^*$ (ただし，$\epsilon \geq 1$) となるが，探索空間は ϵ を大きくすることで大幅に短縮されることが知られている[8)]．$g(n)$ が無視できるほど ϵ が十分に大きくなると $f(n) = h(n)$ となり，これは greedy 探索☆に等しくなる．この場合ゴールノードに近いノードを展開するように探索が進む．一般的に greedy 探索の計算量は最悪の場合，$O(b^m)$（分岐度 b，m は探索空間の最大の深さ）である．しかしながら，道路地図のようにノードとアークが十分に密なグラフでは，探索が進むにつれて $h(n)$ が単調に減少し，着実にゴールへ近づくことになる．したがって，探索のバックトラックはほとんど起らず，解の深さ d としたときに，計算量は $O(b*d)$ に近い値になり，高速な探索が可能となる．

図 1 は，関東圏における経路探索において，異なる ϵ に対する探索空間を示しており，ドットが探索されたノードを表している．図中にある解の質は最適解（67.809 km）を基準として求めたものである．ϵ の値が大きくなるにつれて探索空間は減少するため，それにともない探索時間も減少している．具体的には ϵ の値を 1.0 から 1.1 に増加させただけで，探索時間が

☆ ここでの greedy 探索は繰り返し状態をチェックし，無限ループを回避できるものである[12)]．



$\epsilon = 1.0$
$\#nodes = 9655$
$time = 139.1\,\text{sec.}$
$quality = 1.00$

$\epsilon = 1.1$
$\#nodes = 2874$
$time = 43.4\,\text{sec.}$
$quality = 0.99$

$\epsilon = 2.0$
$\#nodes = 424$
$time = 3.7\,\text{sec.}$
$quality = 0.95$

図 1 ϵ-近似探索の探索空間（最適経路長 67.809 km）
Fig. 1 Search space of ϵ-approximate search (The length of the optimal solution is 67.809 km).

69%もカットされており，2.0の場合は97%もカットされていることが分かる．

次章で述べる時間制限付き経路探索は，このようなϵ-近似探索の特徴を利用したもので，ϵの値を変化させることで，解の質と探索時間のトレードオフをうまく調節する．

3. 時間制限付き経路探索

制限時間内で探索を終了させるための単純な方法は，最も単純な解を導き出し，制限時間の間に少しずつ条件を厳しくして解の質を向上させる方法である．すなわち，ϵの初期値を高く設定して解を求め，徐々にϵの値を小さくしていき，制限時間になったならば，それまでに得られた中で最適な解を出力するというものである．しかしこの方法は時間に余裕がある場合に質の悪い解を何度も生成してしまうという欠点がある．さらに，解生成の際に一度通ったノードを再び通ることになり，反復深化と似た現象を引き起こしてしまう．

これに対して，我々はϵの値を動的に変化させて制限時間を十分に活用した探索方法を提案する．この方法では，残された時間で探索を終了するために，どの程度の近似（ϵの値）で探索を実行すればよいかの見積りを行う必要がある．

我々はϵの見積りを実現するために，以下に示すパラメータを定義する．

- **距離** $D = h(start)$：
  スタートノードからゴールまでの予測コスト．
- **制限時間** T：
  探索を終了するまでに与えられた時間．
- **基準速度** $V = D/T$：
  制限時間で探索を終了するための探索速度．
- **有効距離** $d = D - \min h$：
  探索が進んだ距離（$\min h$は展開されたノードの中で最小のh）．
- **探索時間** t：
  探索を開始してからの経過時間．
- **有効速度** $v = d/t$：
  探索開始から現時点までの探索の平均速度．

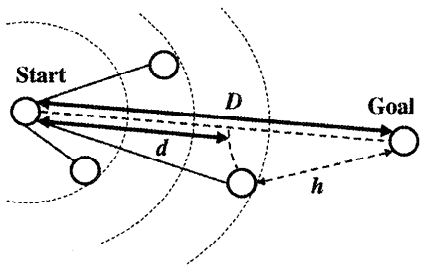


図 2 時間制限付き経路探索のパラメータ
Fig. 2 Parameter for Time-constrained Heuristic Search.

距離と**制限時間**は経路探索問題の難しさを表すパラメータである．道路地図における経路探索の場合，距離は**図 2** のようにスタートノードとゴールノードとの間の直線距離で表される．**距離**が長く，**制限時間**が短い問題は難しい問題としてとらえることができる．また，**有効距離**と**探索時間**は探索の現在の状況を表すパラメータである．ここで，探索の**有効速度**が**基準速度**に一致するならば，**制限時間**ぎりぎりで探索が終了することになる．しかしながら，**基準速度**で探索を実行するようにϵの値を決めることは不可能である．なぜなら，同じϵの値であっても**有効速度**が一定である

とは限らないからである．したがって，**基準速度**を保つためには，状況に応じて ϵ の値を変化させなければならない．我々は**基準速度**と**有効速度**を比較することで，ϵ の値を以下のように制御する．

(1) $v - V < Lower$ ならば $\epsilon = \epsilon + \delta$

(2) $v - V > Upper$ ならば $\epsilon = \epsilon - \delta$

(3) それ以外ならば，ϵ の値を変化させない．

ここで $c \geq 1$，$\delta > 0$ で，ϵ の初期値は1である．(1)では，**有効速度** v が**基準速度** V より小さい場合に ϵ の値を増加させる．これは，ϵ を大きくすることで，図1のように，展開されるノードの数を減少させるためである．結果的に，ノードはゴール方向に向かって直線的に展開されるようになり，**有効速度**を上昇させることができる．一方，(2)では v が V より大きい場合に ϵ の値を下げて**有効速度**を減少させる．このように ϵ の値を変化させて**有効速度**を**基準速度**に保つように制御する．

実際，**有効速度**は探索が進むにつれて減少する傾向にある．これは，図2に示したように探索空間が楕円を描くように広がるためである．始めの段階では展開されるノードは少なく，ゴール方向へノードが展開される割合は高い．しかしながら，探索が進むにつれて展開されるノードは広範囲になる．そのために，ゴール方向へノードが展開される割合は低くなり，**有効速度**は低下する．

このような**有効速度**の特性と上述の ϵ の値の制御によって，実際の ϵ の値は探索が進むにつれて大きくなる．この結果，探索空間の広がりと ϵ の増加によって展開されるノード数はほぼ一定となり，**有効速度**も一定となる．このように ϵ の値を変化させることは，スタートノードに近い部分を厳密に探索し，ゴールノードに近い部分を粗く探索することを意味している．これは，リアルタイムな経路生成に適した探索になる．なぜなら，リアルタイムな経路生成では，移動中に探索を繰り返し実行するが，実際に移動するのは生成した経路の前半部分であり，その部分さえ厳密に探索すればよいからである．

本経路探索において，与えられた**制限時間**が十分長ければ，**基準速度**は小さくなり，**有効速度**は**基準速度**を下回ることはない．したがって，ϵ の値は変化せず初期値1のままになり，探索は A^* 同様に最適経路探索となる．一方，制限時間があまりにも短い場合には，つねに**有効速度**を下回るため，ノードが展開されるたびに ϵ の値は増加し，結果的にgreedy探索に近い探索時間で，高速に探索を終了することになる．

4. アルゴリズム

時間制限付き経路探索アルゴリズムは，スタートノード *Start*，ゴールノード *Goal*，制限時間 T を入力とし，経路が見つかればその経路を，そうでなければ *failure* を出力する．アルゴリズム内で使用される変数は次のとおりである．

- *OPEN*：探索中の経路の先端のノードを集合として格納したもの．この中で最小の評価値 f を持つノードが選択され，展開される．
- *CLOSE*：すでに展開されたノードを集合として格納したもの．同じノードが繰り返し展開されるのをチェックするために利用される．
- min_h：展開されたノードの h の中で最小の値である．これは，有効距離の計算に利用される．
- $time_{start}$：探索開始時刻である．

これ以外に前章で述べたパラメータが変数として使用される．

以下に本アルゴリズムを示す．

Algorithm Time_Constrained_Search(*Start*, *Goal*, T)

```
1  OPEN ← {Start}, CLOSE ← φ
2  D = h(Start)
3  V = D/T
4  t = 0, d = 0, v = 0
5  ε = 1
6  min_h ← h(start)
7  time_start ← get_time()
8  while OPEN ≠ φ
9      min_node ← get_min_node(OPEN, ε)
10     CLOSE ← CLOSE ∪ {min_node}
11     OPEN ← OPEN ∪ expand_node(min_node,
       CLOSE, GOAL, ε)
12     min_h ← get_min_h(OPEN, min_h)
13     if min_h = 0 then
14         return route_to(Start, min_node)
15     d = D − min_h
16     t = get_time() −time_start
17     v = d/t
18     if v − V > Upper then
19         ε ← (ε − δ)
20     else if v − V < Lower then
21         ε ← (ε + δ)
22     if ε < 1.0 then
23         ε ← 1.0
24 return failure
```

1–7 行目において各々の変数が初期化される．get_time は現在の時刻を返す関数であり，探索開始時刻が $time_{start}$ にセットされる．

while 文は *OPEN* が空のときに終了し，そのとき

はゴールまでの経路が存在しないため，*failure* を返しアルゴリズムを終了する（24行）．そうでなければ次の手続きを繰り返し実行する．

まず，get_min_node で *OPEN* の中で最小の評価値 f を持つノード *min_node* を取り出し，それを展開されたノードとして *CLOSE* に入れる（9–10行）．次に *min_node* を展開する（11行）．expand_node は *min_node* に接続するノードの集合を返す関数であり，*CLOSE* の中にないものが返される．expand_node の中では g，h，f を計算し，get_min_node によって展開されたノードの中で最小の h の値を求める（12行）．13–14行はゴールかどうかの判定で，*min_h* の値が0ならば得られた経路を出力してアルゴリズムを終了する．ここで，route_to は *Start* から *min_node* につながる経路を生成する関数である．15–17行では，有効距離 d，探索時間 t，有効速度 v を計算し，18–23行で有効速度 v と基準速度 V を比較することで ϵ の値を変化させる．

通常，A^* では *OPEN* に格納されているノードを f の大小によって管理する際にヒープ木を使うことができる．これにより，$O(\log|OPEN|)$ の手間でノードを管理することができる，しかしながら，本アルゴリズムではヒープ木を使えない．なぜなら，ϵ の値が変化したときは，*OPEN* にあるすべてのノードに対して f の再計算が必要であり，その結果，ヒープ木を再構成しなければならないからである．

本アルゴリズムは評価関数が異なる点と ϵ の操作を除けば，A^* と同じアルゴリズムである．A^* では，評価値 f は単調であり，最初に得られた解が最適であることが保証されているため，そこでアルゴリズムを終了する．しかしながら，f が非単調の場合には，最初に発見された経路よりも短い経路が存在する可能性がある．ここで，ϵ の値を固定して考える．上記のアルゴリズムで得られる経路を $\{start, n_1, n_2, \cdots, goal\}$ とする．最適経路を $\{start, m_1, m_2, \cdots, m_k, goal\}$ とすると，

$$f(goal) \leq g(m_i) + \epsilon h(m_i)$$

となる m_i $(1 \leq i \leq k)$ が存在する．最短経路長 q^* において，

$$\begin{aligned} q^* \leq f(goal) &\leq g(m_i) + \epsilon h(m_i) \\ &\leq g(m_i) + \epsilon h^*(m_i) \\ &= g(m_i) + h^*(m_i) + (\epsilon - 1)h^*(m_i) \\ &= q^* + (\epsilon - 1)h^*(m_i) \\ &\leq q^* + (\epsilon - 1)q^* \\ &= \epsilon q^* \end{aligned}$$

となり，本アルゴリズムで得られる経路長は ϵq^* 以下になることが保証される☆．ここで，ϵ の上限値 ϵ_{max} を定めれば，ϵ をどう変化させても，得られる経路長は $\epsilon_{max} q^*$ を超えることはない．すなわち，ϵ_{max} はアルゴリズムが出力する解の質を制約することになる．

次に，探索の残り時間がほとんど0に等しい場合を考える．ϵ はパラメータ δ を使ってステップ関数的に変化するが，δ を十分大きな値にするとスタートの次のノードから greedy 探索になる．したがって，探索の終了時間は制限時間と greedy 探索に要する時間の和になる．これは最悪の場合であり，実際は探索時間と制限時間はほぼ等しくなることが次章の実験結果から分かる．

以上の議論から，本アルゴリズムは与えられた制限時間と解の質に関する制約を充足する解が存在すれば，それを出力する．なお，アルゴリズムが有限時間内で停止することは展開されるノードが有限であることから明らかである．

5. 実験結果

本アルゴリズムを Java 言語で実装し，様々なマシン環境上で実験を行った．道路地図は実際のカーナビゲーションシステムで使用されているものを用いた．

探索経路はノードが密になっている首都圏を通り，高速道路が最短とならないようにして，探索時間が最もかかるものを選択した（図1のスタートとゴールを結ぶ経路）．CPU が Sun UltraSPARC 168 Mhz，メモリ 128 Mbyte のマシンでハードディスクに納められたデジタルマップを利用した場合，この問題を A^* で解くと 139.1 秒を要した．δ は実験から最適なものを選択して行い，0.05 に設定した．

以下では，与えた制限時間と解の質の関係および制限時間と探索時間の関係を調べる．次に，ϵ の動的な変化の効果について述べる．最後に，スタートからゴールまでの間に探索と移動を繰り返し実行するといった実時間反復探索に適用し，Real-Time A^* と比較することで，本アルゴリズムの有効性を明らかにする．

5.1 制限時間と解のトレードオフ

図3 は制限時間と得られる解の質との関係を示している．ここで，解の質は最短経路長に対する得られた経路長の割合で表している．制限時間 70 秒までは解の質は上昇し，それ以降からは最適解が得られ，解の質は 1.0 になっている．これより，最適解が得られるまでは，与えられる制限時間の増加にともない，単

☆ この証明は Pohl[8] によって与えられており，さらに，石田ら[3] でも再度証明されている．

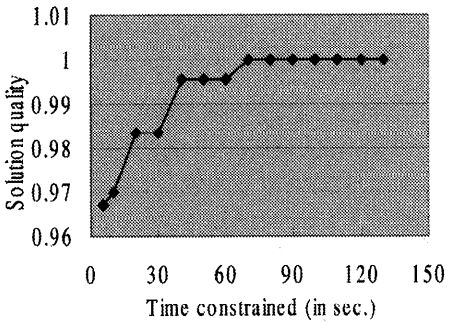
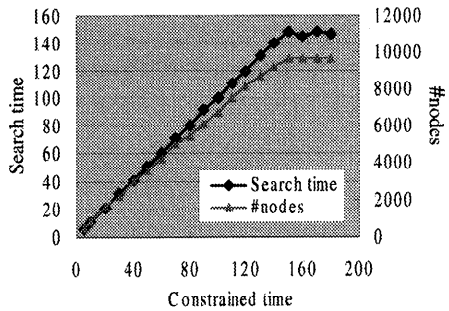
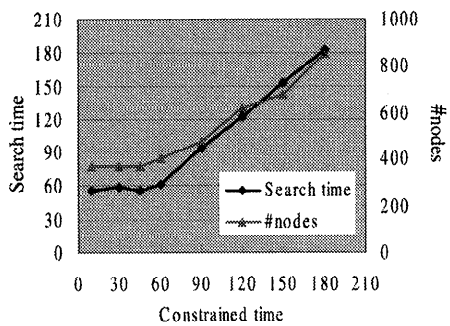


図3 制限時間と解の質の関係
Fig. 3 Solution quality with respect to time constrained.



(A) CPU: Sun UltraSPARC 168 Mhz, Memory: 128 Mbyte, Map: Hard disk



(B) CPU: Intel 486DX4 75 Mhz, Memory: 16 Mbyte, Map: CD-ROM

図4 制限時間とノード数およびサーチ時間の関係
Fig. 4 The relation between Constrained time, #nodes and Search time.

調に解の質は向上することが分かる．

5.2 制限時間と探索時間の関係

図4には，Sun UltraSPARCでハードディスクに納められたデジタルマップを利用した場合と，Intel 486DX4でCD-ROMのマップを利用した場合☆の，制限時間と実際に費やした探索時間の関係，および，そのときに展開されたノード数の関係を示した．

図4(A)において，制限時間140秒以降から探索時間が頭打ちになっている．これは140秒はA^*の場合の探索時間であるため，すでに最適経路を求めており，それ以上のノードは展開されず，それ以上の時間を必要としないからである．図4(B)では制限時間が約60秒までは，与えられた制約時間にかかわらず探索時間が約60秒かかっている．これはマシン性能の限界と見なすことができる．つまり，このマシン環境において解を求めるためには最低60秒☆☆を必要とすることを意味している．

しかしながら，本アルゴリズムは制限時間とほぼ同じ時間で経路探索を終了しており，ノードに関しては制限時間に対応して展開されるノード数が上昇することが分かる．すなわち，本アルゴリズムは制限時間内というよりも，制限時間を十分に活用して探索を実行している．これは図3の結果において，制限時間の増加に従って解の質が向上することを裏付けるものといえる．また，図4の結果から，このような特徴はマシンの性能にかかわらず成り立つことが分かる．

5.3 ϵの変化による効果

図5には本アルゴリズムによって展開されるノード，**図6**には探索中のϵの実際の変化を示した．

図5より，制限時間が減少するにつれて展開されるノード数も減少していることが分かる．これは，図1のϵの値に対するノード数の変化に似ている．しかしながら，明らかに異なるのはスタート地点に近い部分は展開されるノード数は多く，ゴール地点に近い部分はノード数は少ないということである．これは，図6のϵの変化を具体的に示す結果となっている．図6より，ϵの変化パターンは，どの制限時間においても探索の前半部分におけるϵの値はほとんど1.0であり，後半になって高い値をとっていることが分かる．つまり，探索の前半は厳密な探索を実行し，後半部分は粗い探索となることを示している．

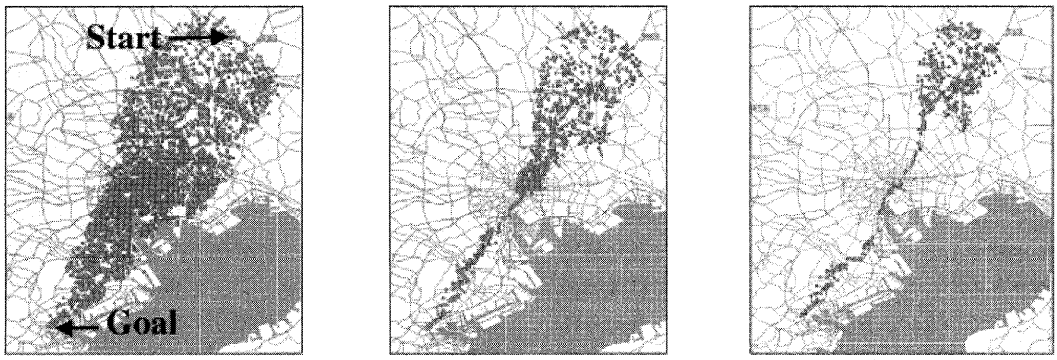
5.4 実時間反復探索と解の質の関係

ここでは，実時間反復探索に本アルゴリズムを適用した結果について述べる．

走行車は一定ノード数nを移動する間に探索を実行し，得られた経路に沿って移動するものとする．このとき，現在のノードを$node(i)$とすると，次のn番目のノード$node(i+n)$に到着する前に，$node(i+n)$からゴールまでの探索を終了しておく必要がある．ここで，$node(i)$から$node(i+n)$の移動時間は，走行

☆ A^*で最適解を求めるのに約1時間かかる．

☆☆ これはgreedy探索の探索時間である．



120 sec.
$\#nodes = 8139$
$time = 119.9$ sec.
$quality = 1.0$

30 sec.
$\#nodes = 2189$
$time = 30.6$ sec.
$quality = 0.98$

10 sec.
$\#nodes = 862$
$time = 10.0$ sec.
$quality = 0.97$

図5 時間制限付き経路探索の探索空間
Fig. 5 Search Space of Time-constrained Heuristic Search.

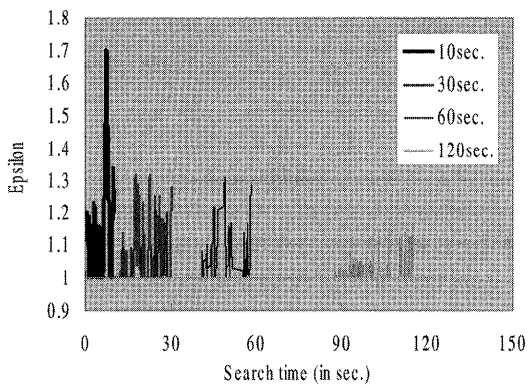


図6 ϵ の変化
Fig. 6 Value change for ϵ.

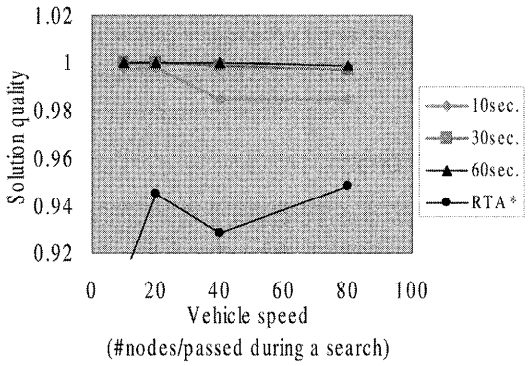


図7 実時間反復探索の解の質
Fig. 7 Solution quality of Real-time search.

車の移動速度から予測することができる．つまり，本アルゴリズムを利用すると，この移動時間を制限時間として設定することで，$node(i+n)$ に到着するまでに探索を終了し，$node(i+n)$ からは新たに得られた経路を移動することができる．

図7には，n 個のノードの移動時間（制限時間）を10秒，30秒，60秒に固定した状況下での，ノード数 n に対する本アルゴリズムの解の質と，Real-Time A^* を用いて60秒ごとに探索と移動を行った場合の結果を示した．ここでの解とは走行車がスタートからゴールまでに実際に移動した経路であり，縦軸にその解の質を示した．また，横軸の Vehicle speed は移動時間ごとに走行車が通過したノード数を示した．これは走行車の移動速度と見なすことができる．1つのノード間の距離は約0.2 km であり，たとえば移動時間60秒で移動するノード数が1の場合は，走行車の速度は時速12 km となる．

図7より本アルゴリズムでは，どの移動時間においても Vehicle speed が小さい方が解の質は高く，大きくなるにつれて質は低下している．図6で示したように，本アルゴリズムの特徴として ϵ の値は後半で大きな値をとる．これは，生成される経路の前半部分は最短経路であるが，後半部分は最短経路とはならないことを意味する．したがって，Vehicle speed が大きい場合には得られた経路が最適である可能性は低くなり，解の質は低下する．また，同じ Vehicle speed では移動時間が大きい方が解の質は高くなっている．これは，図3の結果から明らかなように，制限時間が長ければ解の質は高いからである．

さらに，図7に示した解の質と各移動時間に対応する図3の制限時間の解の質を比較すると，繰り返し探索を実行することで結果的に解の質は向上していることが分かる（たとえば，図3の制限時間30秒の解の

質は約0.985であり，図7の移動時間30秒では0.997以上である)．これは，得られた経路の前半部分が準最適な経路になっており，走行車はその経路のところまでを移動するからである．したがって，本アルゴリズムは移動中の走行車に適したリアルタイムな探索を実現しているものといえる．

ここで，本アルゴリズムの結果とReal-Time A^*の結果を比較すると，本アルゴリズムではほぼ最短経路を移動できるのに対して，Real-Time A^*では最短経路は得られていない．Real-Time A^*では，与えられた制限時間内にゴールまでの経路を発見できないときは，再度探索すると，後戻りするような経路が得られる可能性がある．特に，袋小路に陥った際は，この現象が顕著に現れ，実際に移動する経路は最短経路とはならない．しかし，本アルゴリズムでは時間内にゴールまでの経路を発見するため，袋小路に陥ることはなく，後戻りするような経路は得られない．すなわち，本アルゴリズムを利用して実時間反復探索を実行することで，結果的に走行車は最短経路を移動することになる．

6. 関連研究

制限時間内に準最適な経路を生成する問題は，ある制約のもとでの最適化問題としてとらえることができ，走行車をエージェントと見なせばこれはRussellの提案したBounded-optimalなエージェントの一例となる[11]．Russellはこのようなエージェントを設計・実現する一般的な方法を提案している．それに対して本研究は最短経路問題に限定しているが，その分，制限時間内での最適経路発見の方法やアルゴリズムを具体的に示すことに成功した．Russellではエージェントが探索問題において，どのような方法で近似解を生成すればよいかは述べていない．一方，我々は残された探索時間からϵの値をどのように変化させればよいのかを含めた探索アルゴリズムを示した．

制限時間が長ければ，良い解が得られるという特徴はAnytimeアルゴリズム[1]と似ている．Anytimeアルゴリズムはいつでも割り込みでき，その時点での最良結果を出力する．アクションの効用関数は時間に対して単調であるという性質をベースとしており，さらに計算を止めてよいかを決定するメタレベルの手続きがある．しかしながら，具体的にどのようなクラスの問題に対して適用可能かは明らかにされていない．しかも，アルゴリズムの評価はなされていない．この種の問題ではiterative deepening探索が使われるが，我々の方法では近似の度合(ϵ)を変化させる探索になっている点が異なる．

ϵを動的に変化させることは本研究が初めてではなく，古くはPohlの研究がある[8]．Pohlは巡回セールスマン問題に対してϵの値を探索の深さに反比例させる方法を提案した．これは，解に到達するまでのステップ数を見積もり，そのステップ数が多いときは粗に探索し，解に近付いてくると厳密に探索するという戦略になっている．これは我々の方法と逆の戦略である．また，制限時間とリンクさせることは我々の独自の方法であり，結果としてリアルタイムアルゴリズムを構成している点も異なる．

Korfの Real-Time A^*では，計画と実行を交互に繰り返すことで，局所最適な解を生成する[4]．これは一度通ったノードに後戻りできるためである．しかし，本論文のような走行車の経路生成では後戻りは非常に高いコストがつく．そのため実行の前に近似的な解を生成する必要がある．本方法はスタートノードに近いところは厳密に探索するため，移動中に再探索させることで，図7に示したような，ほぼ最適な経路が得られる．本手法はReal-Time A^*と違って，近似の度合いをコントロールすることで，時間内にゴールまでの探索を終了できる点に特徴がある．

ϵを増加させることは，ノードの分岐数を徐々に減らすことに相当する．これはGinsbergらが提案したIterative Broadening[2]とは正反対の特徴を持つ．Iterative Broadeningはノードの分岐数を増やして探索空間を徐々に広げていくものである．我々の方法は分岐数が残りの時間とともに動的に設定される．

本研究のもともとの動機は商用のカーナビシステムにおける経路探索アルゴリズムに端を発している．現在のカーナビに搭載されているものはアルゴリズムが不明であり，実際に最適解は生成されない．また，可能解が複数出力されたり，動的に経路を変更することは非常に困難である．本方法はこうした問題を解決している．

7. おわりに

本論文では，リアルタイムな経路計画を時間制約付きの最適化問題として定式化し，従来の近似A^*を拡張した動的な探索アルゴリズムを示した．本アルゴリズムの最大の特徴は従来のリアルタイムな経路計画とは異なり，制限時間内にゴールまでの探索を終了する点である．カーナビで使用される実際のデジタルマップを用いて本アルゴリズムを適用した結果から，探索の終了時刻を正確に推定することができ，制限時間を十分に利用しながら解を求めることが分かった．さら

に，この特徴はマシンの性能に関係なく成り立つもので，本方法は汎用なリアルタイム経路探索になっていることを明らかにした．

時間制限付き経路探索を実用化するためには，制限時間をどのように設定・変更するかといった課題を解決する必要がある．この答えの1つに，走行車の移動速度から制限時間を決定する方法が考えられる．移動速度から次に探索を実行するノードへの移動時間が予測できる．この移動時間を制限時間に設定することでリアルタイムな経路生成が実現される．また，移動中に生成した経路をはずれたときや，渋滞が発生したときなど，新たな経路を急に必要とする場合には，次のノードに到着するまでの時間を制限時間に設定するといったように，それぞれの状況に応じて制限時間を設定することで実用的な経路生成が可能となる．これは今後の研究課題である．

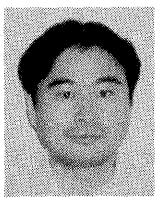
参考文献

1) Dean, T. and Boddy, M.: An Analysis of Time-Dependent Planning, *Proc. 7th National Conference on Artificial Intelligence (AAAI-88)*, pp.49–54 (1988).
2) Ginsberg, M.L. and Harvey, W.D.: Iterative Broadening, *Proc. 8th National Conference on Artificial Intelligence (AAAI-90)*, pp.216–220 (1990).
3) 石田 亨，新保 仁：実時間探索による経路学習，人工知能学会誌，Vol.11, No.3 (1996).
4) Korf, R.E.: Real-Time Heuristic Search, *Artificial Intelligence*, Vol.42, pp.189–211 (1990).
5) Köll, A.L. and Kaindl, H.: Bidirectional Best-First Search with Bounded Error: Summary of Results, *13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp.217–223 (1993).
6) Nilsson, N.J.: *Principles of Artificial Intelligence*, Morgan Kaufmann, San Mateo, California (1980).
7) Pearl, J. and Kim, J.H.: Studies in Semi-Admissible Heuristics, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.PAMI-4, No.4, pp.392–399 (1982).
8) Pohl, I.: The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving, *3rd International Joint Conference on Artificial Intelligence (IJCAI-73)*, pp.12–17 (1973).
9) Prieditis, A.E.: Machine discovery of effective admissible heuristics, *Machine Learning*, Vol.12, No.1-3, pp.117–141 (1993).
10) Russell, S.J.: Efficient memory-bounded search methods, *10th European Conference on Artificial Intelligence Proceedings ECAI92*, pp.1–5 (1992).
11) Russell, S.J. and Subramanian, D.: Provably Bounded-Optimal Agents, *Journal of Artificial Intelligence Research*, Vol.2, pp.575–609 (1995).
12) Russell, S.J. and Norvig, P.: *Artificial Intelligence: A Modern Approach*, Prntice Hall (1995).
13) 志村正道：機械知能論，昭晃堂 (1983).

(平成10年9月9日受付)
(平成11年9月2日採録)

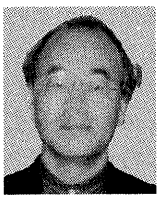
平石 広典（学生会員）

1971年生．1997年東京理科大学大学院理工学研究科経営工学専攻修士課程修了．同年同大学大学院博士後期課程入学，現在に至る．人工知能学会，ソフトウェア科学会，日本ロボット学会各会員．



大和田勇人（正会員）

1960年生．1988年東京理科大学大学院理工学研究科経営工学専攻博士後期課程修了．工学博士．同年同大学理工学部経営工学科助手．現在は講師．データマイニング，帰納論理プログラミングに興味を持つ．人工知能学会，ソフトウェア科学会各会員．



溝口 文雄（正会員）

1941年生．1968年東京理科大学大学院修士課程修了，同年同大学理工学部経営工学科助手．1987年教授となり，現在大学院研究科委員および情報メディアセンター長．工学博士．1994年Stanford大学のCenter for the Study of Languagc and Information（CSLI）の上級研究員となる．人工知能，認知科学およびソフトウェア工学に興味を持ち，Javaを用いた情報家電や自律型ロボットの研究を中心に進めている．現在Artificial Intelligence Journalの論文編集委員．日本ソフトウェア科学会，日本認知科学会，米国人工知能学会各会員．