D* Lite

スヴェン・ケーニッヒ
ジョージア工科大学コンピューティング学部
Atlanta, GA 30312-0280
skoenig@cc.gatech.edu

Maxim Likhachev
School of Computer Science
カーネギーメロン大学
ピッツバーグ、ペンシルバニア州 15213
maxim+@cs.cmu.edu

概要

インクリメンタルヒューリスティック探索は，ヒューリスティックな手法により探索の焦点を絞り，過去の探索から得た情報を再利用することで，一連の類似した探索課題に対して，それぞれの探索課題をゼロから解くよりもはるかに高速に解を求めることができる手法である．本論文では、生涯計画A*を未知地形におけるロボットナビゲーションに適用し、未知地形における目標指向型ナビゲーションや未知地形の地図作成などを実現した。その結果得られたD* Liteアルゴリズムは、理解・解析が容易である。これはStentzのFocussed Dynamic A*と同じ動作を模倣しているが、アルゴリズム的には異なるものである。D* Liteの特性を証明し、研究対象のアプリケーションにおいて、漸進的探索とヒューリスティック探索を組み合わせることの利点を実験的に実証した。これらの結果は、人工知能やロボット工学における高速再走査法のさらなる研究のための強力な基盤を提供するものであると考える。

はじめに

DynamicSWSF-FP (Ramalingam & Reps 1996)のようなインクリメンタルサーチは、現在人工知能の分野ではあまり使われていない。この手法は、過去の検索から得た情報を再利用し、一連の類似した検索課題に対して、それぞれの検索課題を一から解決するよりもはるかに速く解を見つけることができる。その概要は(Frigioni, Marchetti-Spaccamela, & Nanni 2000)で述べられている。一方、A* (Nilsson 1971)などのヒューリスティック探索法は、ゴール距離の近似値という形でヒューリスティック知識を用いて探索を集中させ、情報を持たない探索法よりもはるかに高速に探索問題を解決する。その概要は(Pearl 1985)に示されている。我々は最近、DynamicSWSF-FPとA*を一般化したLPA* (Lifelong Plan- ning A*)を導入し、二つの異なる手法により計画時間の短縮を図った (Koenig & Likhachev 2001)。本論文では、LPA*を未知地形におけるロボットナビゲーションに適用する。ロボットが未知の障害物を発見した後、経路を再計画する際には、従来のグラフ探索手法を用いることができる。しかし，大規模な地形では計画時間が数分となり，実質的なアイドルタイムとなる（Stentz 1994）．ダイナミックA* (D*) (Stentz 1995)は巧妙なヒューリスティック探索法であり、A*探索の繰り返しに対して、D*を修正することにより1〜2桁(!)の速度向上を達成する。

は、過去の検索結果を局所的に検索することができる。D*は、屋外用HMMWV（Stentz & Hebert 1995）を含む実際のロボットで広く使用されている。また、現在、火星ローバーのプロトタイプや都市偵察のための戦術的な移動ロボットのプロトタイプに統合されている（Matthies *et al.*）しかし、他の研究者によって拡張されたことはない。そこで、我々はLPA*をベースに、D*と同じナビゲーション戦略を実装しながらも、アルゴリズムが異なる新しいリプランニング手法であるD* Liteを提案する。D* LiteはD*より大幅に短く、優先度比較の際に同点判定基準を1つだけ用いるため優先度の管理が容易であり、3行にも及ぶ複雑な条件のif文が不要なためプログラムの流れの解析が容易であることが特徴です。また、これらの特性により、例えば、許されないヒューリスティックや異なるタイブレーク基準を用いて効率化を図るなど、容易に拡張することができる。LPA*の挙動を理解するために、D* Liteにも適用可能なLPA*の様々な理論的特性を示す。理論的な特性から、LPA*は効率的であり、よく知られ理解されている探索アルゴリズムであるA*と類似していることが示された。また、D* LiteはD*と同等以上の性能を持つことが実験的に示された。また、目標指向型ナビゲーションやマッピングなど、未知の地形における様々なナビゲーションタスクにおいて、インクリメンタルサーチとヒューリスティックサーチを組み合わせることの利点について実験的に評価した結果を示している。我々は、D* Liteの理論的・実証的な分析が、人工知能やロボット工学における高速リプランニング手法のさらなる研究のための強力な基盤となることを確信している。

モチベーション

ロボットは常に隣接する8つのセルのうちどれが通過可能かを観察し、そのうちの1つまでコスト1で移動する、未知の地形における目標指向型ロボットナビゲーションタスクを考える。ロボットは開始セルからスタートし、ゴールセルまで移動しなければならない。このとき、閉塞状態が不明なセルは通過可能であると仮定し、現在のセルからゴールセルまでの最短経路を常に計算する。その結果、ゴールセルに到達するまでこの経路をたどり、その時点で正常に停止するか、あるいは、通過不可能なセルに遭遇した場合、現在のセルからゴールセルまでの最短経路を再計算する。図1は、ロボットが移動する前と移動した後の、すべての走行可能なセルのゴール距離と、現在のセルからゴールセルまでの最短経路を示したものである。

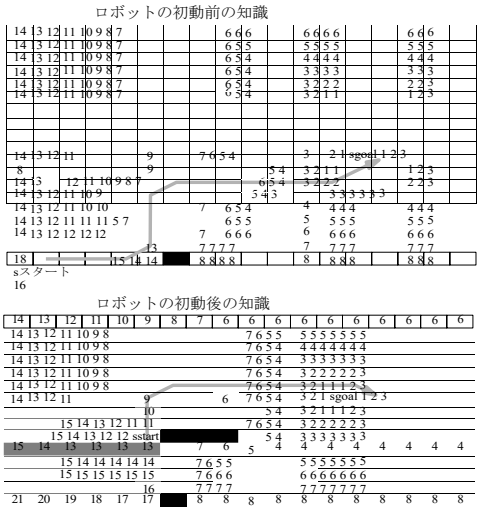


図1：簡単な例

を通過し、最初にブロックされたセルを発見した。ゴールまでの距離が変化したセルは灰色の影で表示されている。ゴール距離が重要なのは、一度ゴール距離が計算されると、ゴール距離を貪欲に減少させることによって、ロボットの現在のセルからゴールセルまでの最短経路を容易に決定することができるからである。ゴール距離が変化したセルの数は少なく、変化したゴール距離のほとんどは、現在のセルからゴールセルまでの最短経路を再計算するのに関係ないことに注意してください。したがって、現在のセルからゴールセルまでの最短経路を効率的に再計算するには、ゴール距離が変更された（または以前に計算されていない）、最短経路の再計算に関連するゴール距離のみを再計算すればよい。これがD*
Liteの機能である。課題は、このようなセルを効率的に特定することである。

生涯設計 A*

生涯計画A*（LPA*）を図2に示す。LPA*はA*のインクリメンタルバージョンである。LPA*は、時間経過とともに辺のコストが増減する既知のグラフの有限グラフ探索問題に適用される（辺や頂点の追加・削除のモデル化にも使用できる）。

頂点

の後継者
の集合同様に、頂点の前任者の集合を表す。
は頂点から頂点

移動するコストを表す．LPA*は
常に、与えられた開始頂点

から与えられたゴール頂点への最短経路グラフのトポロジーと現在のエッジコストの

この疑似コードでは、以下の関数を使って優先キューを管理しています。U.Top は
優先度待ち行列に含まれるすべての頂点のうち，最も小さい優先度を持つ頂点を返します．U.TopKeyは、優先度キューに含まれるすべての頂点のうち、最も小さい優先度を返す。(空の場合、U.TopKey は戻る 。)
U .Pop
は、優先度キューで最も小さい優先度を持つ頂点を削除し、その頂点を返す。U.Insert優先度付きキューに優先度.の頂点を挿入する．U.Update優先度キューを変更する。
の優先順位を.に変更します（現在の頂点の優先順位が.の場合は何も行いません）。
はすでに等しい．）最後に，U.Remove は，優先度キューから頂点を削除します．

```
procedure CalculateKey
 01 リターン                                                    ;
手続き 初期化
 02          ;
 03 for all                                  ;
 04                        ;
 05  U.インサート     CalculateKey          ;
procedure UpdateVertex
 06  もし                                                      ;
 07  もし        U.リムーブ   ;
 08  もし                 U.Insert     CalculateKey     ;
procedure ComputeShortestPath
 09  while U.TopKey      キーを計算する       または
 10          U.P. ;
 11    もし
 12                        ;
 13      for allUpdateVertex ;
 14    さもなくば
 15                  ;
 16      for allUpdateVertex ;
手順 メイン
 17  初期化;
 18  末永く
 19    ComputeShortestPath ;
 20    エッジコストの変化を待つ。
 21    を持つすべての有向辺について，その辺のコストを変更する
 22      エッジコストを更新する   ;
 23      UpdateVertex ;
```

図2：生涯設計A*。

生涯設計A*。変数

LPA*はスタート距離の推定値を保持している
の各頂点の値である。これらの値は、A*探索のg値に直接対応する。LPA*はこれらの値を探索から探索へと引き継ぎます。また、LPA*は2種類の開始距離の推定値も保持しています。rhs値はg値に基づく1ステップ先読み値であり、g値よりも情報が豊富である可能性がある。この値は常に次の関係 *（不変量1）を満たす。*

そうでな
い場合は (1)

頂点は、そのg値がそのrhs値と等しい場合に局所整合と呼ばれ、そうでない場合は局所不整合と呼ばれる。すべての頂点が局所的に矛盾しない場合、すべての頂点のg値がそれぞれの開始距離と等しい。このとき
両方を知った決定する　から使用頂点
の開始距離、つまり
からの最短パスの長さに注目A*と同様に、LPA*は頂点のゴール距離を近似するヒューリスティックを用いる。このヒューリスティックは非負で矛盾がないこと(Pearl
1985)，すなわち，三角形の不等式に従うことが必要である．
と
すべての頂点に対して　と　で
.

起点として、常に現在の頂点から遷移することで
、
から任意の頂点までの最短経路。
を最小化する任意の前任者に、 である。
(同点は任意に解消できる)
に到達するまで図1とは異なり
、最短経路の決定には開始距離の代わりに目標距離が用いられ、最短経路をたどることができるfro
mto
を出発点として 、

、最小化任意の 後継者
へ移動する方法によって最短経路をたどることができます)。ただし、LPA*はすべての頂点が最短
経路になるわけではありません。
は、いくつかのエッジコストが変更された後、局所的に一貫性を保つことができる。その代わりに
、ヒューリスティックを用いて探索を集中させ、最短経路の算出に関連するg値のみを更新する。
このため、LPA*は優先キューを保持する。その

優先度待ち行列には、常に局所的に無矛盾な頂点が含まれています（*不変量2*）。これは、LPA*が潜在的にg値を更新してローカルに整合させる必要がある頂点である。優先度キューにおける頂点の優先度は、常にそのキーと同じであり（*不変量3*）、これは次の2つの成分を持つベクトルである。

と

01（括弧内の数字は図2 の 行番号を示す）。 キーの最初の構成要素

は、そのままf値に対応する

LPA*のg値およびrhs値はA*のg値に対応し、LPA*のh値はA*のh値に対応するため、A*が使用する。鍵の第2成分はA*のg値に対応する。鍵の比較は、辞書的な順序で行われる。たとえば、ある鍵が(

で示されると同等以下であるのは、
または（ and
）です。LPA*は、常に優先度待ち行列の中で最も小さいキーを持つ頂点を拡張します（頂点の拡張とは、10-16を実行することを意味します）。これは、A*がg値の小さい方への紐付けを解除すると、必ずf値の小さい優先キュー内の頂点を展開するのと同様である。また、LPA*とA*の結果の動作も類似している。LPA*によって展開される頂点のキーは、A*によって展開される頂点のf値と同様に、時間の経過とともに減少しない（ヒューリスティックスが一貫しているため）。

生涯設計A*。アルゴリズム

LPA*のメイン関数Main()は，まずInitialize()を呼び出して探索問題を初期化する17．Initialize()は、全ての頂点のg値をinfinityに設定し、そのrhs値を式1 03-04に従って設定する。、初期状態では、
が唯一の局所的に無矛盾な頂点であり、他は空の優先順位キュー05に挿入。この初期化により、ComputeShortestPath()の最初の呼び出しは、正確にA*探索を行うこと、つまり、A*と全く同じ頂点を全く同じ順序で展開することが保証される。実際の実装では、Initialize()は探索中に頂点に出会ったときだけ初期化すればよいので、
すべての頂点を前もって初期化する必要はないことに注意されたい。これは、頂点の数が膨大になり、探索中に到達する頂点が少なくなる可能性があるため、重要なポイントになります。LPA*はその後、エッジコスト20の変化を待ちます。また，ローカルに矛盾する頂点を優先順に展開する最短経路 19 を ComputeShortestPath() で再計算する．

ローカルに矛盾する頂点は

場合、ローカルにオーバーコンシステントと呼ばれるComputeShortestPath()がローカルにオーバーコンシステントな頂点12-13を指定すると、
その頂点のg値をそのrhs値12に設定し、その頂点をローカルにコンシステントな状態にする。局所的に無矛盾な頂点は、以下の場合、局所的に無矛盾であるという

。ComputeShortestPath()が局所的に無矛盾な頂点15-16を展開する場合、その頂点のg値をinfinity
15に設定するだけである。これにより、その頂点はローカルに

一貫性があるか、過剰一貫性があるか。拡張された頂点が緩やかに過剰無矛盾であった場合、そのg値の変化は、その
後継者の局所無矛盾性に影響を与えることができる13
。同様に、拡張された頂点が局所的に非整合であった場合、その頂点とその
後継者が影響を受ける可能性がある16
。したがって、不変量1～3を維持するために、ComputeShortestPath()
は、これらの頂点のrhs値を更新し、それらのローカルな一貫性をチェックし、それに応じて優先順位キューに追加または削除する　　06～08
。ComputeShortestPath()は、
　　.探索の結果,　　で
から.NETへの最短経路は現在の頂点 , から
　　最小化する任意の先行頂点
(tiesは任意に切断できる)
　　に達するまで常に遷移することにより、tob
からの最短経路をたどることができる。
　　これは、A*がバックポインタを使用しない場合にできることと同様である。

分析結果

ここで、LPA*のいくつかの特性を示し、それが終了し、正しく、A*と同様であり、効率的であることを示す。すべての証明は(Likhachev & Koenig 2001)に記載されている。

終端と正しさ

第一定理は、LPA*が終了し、正しいことを示す。

定理1
ComputeShortestPath()は各頂点を最大2回、すなわち局所的に無矛盾である場合は最大1回、局所的に無矛盾である場合は最大1回展開し、終了する。ComputeShortestPath()が終了した後、以下のような最短経路をたどることができる。
で始まる現在の頂点から常に遷移することにより
*　　を最小化する任意の前任者に*
に達するまで（同点は任意に解消することができる）
。

A*との類似性

LPA*の説明の際に、LPA*とA*のアルゴリズム的な強い類似性を既に指摘しました。ここでは、LPA*とA*のさらなる類似性を示す。定理1では、ComputeShortestPath()が各頂点を最大2回展開することを既に示しましたが、これは、A*が各頂点を最大2回展開することと類似しています。これは、各頂点を最大1回ずつ展開するA*と同様である。さらに、次の定理は、ComputeShortest-Path()によって展開される頂点のキーは、時間に対して単調に非減少であることを述べている。これは、A*によって展開される頂点のf値の順序が非減少であることと同様である。

定理2　　*10行目でComputeShortest-Path()が展開のために選択する頂点のキーは、ComputeShortestPath()が終了するまで、時間と共に単調に非減少となる。*

次の3つの定理は、ComputeShortest-Path()が、A*が頂点を拡張する方法と非常に類似した方法で、局所的に過保証の頂点を拡張することを示す。次の定理は、例えば、A*のキーの最初の成分が、局所的にオーバーコンシステントな頂点を
展開することを示す。

局所的に矛盾がなく、次に展開する頂点のキーが .NETのキーより小さ

ComputeShortest- Path() がそれを展開したときのローカルにオーバーコンシステントな頂点は、その頂点の f 値と同じである。そのキーの2番目のコンポーネントは、その開始距離です。

定理3

ComputeShortestPathが10行目で緩やかな過保証頂点を選択して展開する場合、そのキーは

定理2、3により、ComputeShortestPath()は、局所的に無矛盾な頂点をf値が単調に減少しない順に、同じf値を持つ頂点を開始距離が単調に減少しない順に配置することを意味している。A* は、開始距離の小さい頂点を優先して同値を解消することを条件として、同じ性質を持つ。

定理4

*ComputeShortestPathは、A*が常に同じf値を持つ頂点間の同値を最小の開始距離を持つ頂点に優先して分割し、残りの同値を適切に分割する場合に、有限のf値を持つ局所的に過剰整合な頂点をA*と同じ順序で拡張する。*

次の定理は、ComputeShortestPath()は、f値がゴール頂点のf値より小さい局所的に過保存な 頂点と、f値がゴール頂点のf値と等しく、開始距離がゴール頂点の開始距離以下 の頂点を最大でも除外することを示している。A*は、開始距離がより小さい頂点を優先して同値を解消することを条件として、同じ性質を持つ。

定理 5*ComputeShortestPath* *最大で、局所的に過剰無矛盾な頂点を*拡張 *である。*

.

効率性

ここで、LPA*は定理1で提案されたよりもはるかに少ない頂点を展開することを示す。次の定理は、LPA*がインクリメンタルサーチを行うことにより、コスト変更の影響を受けたり、以前のサーチでまだ計算されていないg値のみを計算し、効率的であることを示しています。

定理6

ComputeShortestPath()は、ComputeShortestPath()が呼ばれる前にg値がそれぞれの開始距離と等しかった頂点を展開しない。

この定理は、LPA*がヒューリスティック探索を行うことにより、最短経路を決定するために重要な頂点のg値のみを計算し、効率的であることを示している。定理5では、ComputeShortestPath()で展開される局所的な過保存頂点の数がヒューリスティック探索により制限されることを既に示した。次の定理は、この結果をComputeShortestPath()によって展開されるすべての局所的に無矛盾な頂点に一般化するものである。

定理7 *10行目でComputeShort- estPath()が展開のために選択する頂点のキーは、以下の値を超えることはない。*

.

この定理がLPA*の効率に与える影響を理解するために、頂点のキーが次のとおりであることを思い出してください。

.

したがって、ヒューリスティックの情報量が多いほど、また、ヒューリスティックの規模が大きいほど、以下の条件を満たす頂点は少なくなります。

というように、拡張されています。

```
procedure CalculateKey
 01'リターン                                                                    ;
手続き 初期化
 02'            ;
 03'
 04' for all                                    ;
 05'                    ;
 06' U.インサート      計算キー                ;
procedure UpdateVertex
 07' if                                                                        ;
 08' if                U.Remove     ;
 09' の場合                    U.Insert      計算キー          ;
procedure ComputeShortestPath
 10' while U.TopKeyCalculateKey                        OR 11'          U.TopKeyの
 12'             U.Pop ;
 13'      もし         キーを計算する
 14'        U.Insert      計算キー
 15'      else if
 16'                          ;
 17'         forall
          UpdateVertex; 18'          さもなくば
 19'                          ;
 20'         for allUpdateVertex ;
手順 メイン
 21'
 22' 初期化 ;
 23' ComputeShortestPath ;
 24'while
 25'      /* ifthen 既知のパスが存在しない場合 */。
 26'                                                                            ;
 27'      に移動する。   ;
 28'変更されたエッジコストについ
 てグラフを
          スキャンする; 29'
          いずれかのエッジコストが
 変更された場合
 30'
 31'
 エッジコストが変更されたすべての有向エッジに対して32'。
 33'           エッジコストの更新          ;
 34'     アップデートバーテックス
                          ;
 35'          ComputeShortestPath ;
```

図3：D* Lite。

D* Lite

これまで，グラフの始点と終点の間の最短経路をグラフの辺費用として繰り返し求めるLPA*について述べてきた．次に、LPA*を利用して、ロボットがゴール頂点に向かって移動しながらグラフの辺コストが変化するのに応じて、ロボットの現在頂点とゴール頂点間の最短経路を繰り返し決定するD* Liteを開発した。D* Liteを図3に示す。D* Liteは、辺コストがどのように変化するか、上がるか下がるか、ロボットの現在の頂点に近いところで変化するか遠いところで変化するか、世界で変化するかロボットが初期推定値を修正しただけで変化するかについて、何の仮定も置いていない。D* Liteは、未知の地形における目標指向のナビゲーション問題を解くことができる（「動機」のセクションで説明したとおり）。地形は8連結のグラフとしてモデル化される。その辺のコストは最初1である。そして、ロボットがその辺を通れないと判断すると、その辺のコストは無限大になる。このグラフにD* Lite を適用し、ロボットの 現在の頂点をはゴール頂点である。

検索方向

まず、LPA*の探索方向を変更する必要があります。図2のLPA*は開始頂点からゴール頂点を探索するため、そのg値は開始距離の推定値となる。D* Liteはゴール頂点からスタート頂点を探索するため、そのg値はスタート距離の推定値である。

ゴール距離のLPA*から派生したもので、擬似コード中のスタートとゴールの頂点を交換し、すべてのエッジを反転させたものです。したがって、D* Liteは元のグラフ上で動作し、LPA*と同様に頂点の後継*者*と先行者を決定できる必要がある以外は、グラフに対する制限はありません。ComputeShortestPath ()が返った後、

現在の頂点から、常に
を 最小化する任意の後継者に
に達するまで（同点は任意に解消可能）。

ヒープリオーダリング

未知の地形におけるロボットナビゲーションの問題を解決するために、Main() は CalculatePath() によって決定された経路に沿ってロボットを移動させる必要があります。Main()は、ロボットが移動した後、辺のコストに変化があるたびに、優先順位キュー内の頂点の優先順位を再計算することができる。しかし、優先順位の再計算が行われない限り、優先順位はロボットの旧頂点を基準として計算されたヒューリスティックに基づいているため、In-variant 3を満たさない。しかし、優先度待ち行列には多くの頂点が含まれるため、優先度待ち行列の並べ替えを繰り返すのはコストがかかる。そこで、D* Liteは、D*（Stentz 1995）から派生した方法、すなわち、LPA*が対応する頂点に対して使用する優先度の下限となる優先度を使用して、優先度キューの並べ替えを回避しているのである。ヒューリスティックは、現在、非負で以下を満たす必要がある。

と に対して
ここで は頂点
から頂点

最短パスのこの要件は制限的なものではなく、ヒューリスティックが探索問題を緩和することによって導出される場合、両方の性質が成立することが保証されているからであり、これはほとんどの場合、本論文で使用するヒューリスティックで成立する。ロボットが頂点
からエッジコストの変化を検出
した頂点
移動優先順位の最初の要素は最大で
.減少することができる（2番目の要素はヒューリスティックに依存しないので変化しないままである）。したがって、下界を維持するために、 D* Liteは優先度キュー内のすべての頂点の優先度の最初の要素から
減算する必要がしかし、優先度待ち行列のすべての頂点で同じであるため、D* Liteは、優先度待ち行列のすべての頂点の優先度の最初の要素から。
の順序は、減算が実行されない場合、優先度キュー内の頂点の順序は変化しない。そして、新しい優先順位が計算されるとき、その最初の成分は優先順位待ち行列の
優先順位に対して小さすぎるのです。従って、その都度、その第1成分を追加しなければならない。エッジのコストが変化する。もしロボットが再び移動して、再びコストの変化を検出したら、定数を足し合わせる必要がある。これは変数 30' で行う。したがって，新しい優先順位が計算されるたびに，01'で行ったように，その最初の構成要素にこの変数を追加しなければならない．このように、ロボットが移動しても、優先順位列の頂点の順番は変わらないので、優先順位列の並び替えは必要ない。一方、優先順位は、LPA*の優先順位の第1成分を現在の値である.Nで増加させた後、LPA*の対応する優先順位の下限となるように工夫されている。この性質を利用し、ComputeShortestPathを以下のように変更する。を変更する。

```
procedure CalculateKey
 01 "リターン                                                                    ;
手続き 初期化
 02"          ;
 03"
 04 "をすべて                              ;
 05"                    ;
 06インチU.S.A.                              ;
procedure UpdateVertex
 07" if (                    AND          U.アップデート                    ;
 08" else if                 AND          U.インサート                      ;
 09" else if                 AND          U.Remove    ;
procedure ComputeShortestPath
 10" while U.TopKeyCalculateKey                  OR 11 "       U.Top ;
 12 "              U.TopKey ;
 13"                CalculateKey     ;
 14"     もし
 15"       U.アップデート
 16 "    else if
 17"                    ;
 18"       U.リムーブ   ;
 19「すべての            人に
 20"         if                                                          ;
 21 "        UpdateVertex ;
 22"     さもなくば
 23"                    ;
 24"                  ;
 25        インチ
 26"         もし
 27"           if                                                        ;
 28 "        UpdateVertex（アップデートバーテックス）;
手順 メイン
 29"
 30" 初期化 ;
 31" ComputeShortestPath ;
 32インチながら
 33"     /* ifthen 既知のパスが存在しない */
 34"                                                                     ;
 35"     に移動する。   ;
 36
"変更されたエッジコストについ
てグラフを              スキャン;
 37 "
        エッジコストが変更された
場合
 38"
 39"
 40"       すべての有向のエッ      エッジコストの変更に
           ジに対して              伴い
 41"                  ;
 42"        エッジコストの更新       ;
 43"         もし              )
 44"           もし                                                    ;
 45 "        else if
 46"           if                                                      ;
 47"         アップデートバーテックス          ;
 48 "       ComputeShortestPath ;
```

図4：D* Lite（最適化版）。

ter ComputeShortestPathは、優先度U
.TopKeyが最も小さい　　頂点を
優先度キュー12'から削除した場合、Calcu
lateKey
使用して、その頂点が持つべき優先度を計算する。もし
CalculateKeyなら、CalculateKeyによって

計算された優先度を持つ削除された頂点を優先度キュー13'-
14'に再挿入する。このように、LPA*の優先度の最初の成分（）を現在の値
）で増加さ後も、優先度待ち行列のすべての頂点の優先度はLPA*の相関する優先度の下限であることが真であることに変わりはない。IfCalculateKey

ならば、以下が成立する。　を計算する。
は、
CalculateKey()が返す値の下限であったので、これが成立する。　　場合ComputeShortestPath
はLPA*と同様に頂点
展開する（頂点を展開するという15'-
20'を実行）。

最適化

図　4　は、いくつかの最適化を施した　D*　Lite を示している。一例として、ComputeShortestPath()の終了条件を変更することで、ComputeShortestPath()をより効率的に使用することができる。前述の通り、ComputeShortestPath()は、開始頂点がローカルに整合し、　かつそのキーがU.TopKey()　10'以下である場合に終了する。しかし、Com puteShortestPath()は、開始頂点がローカルに無矛盾でなく、そのキーが　U.TopKey()以下であれば、既に終了することが可能である。この理由を理解するために、開始頂点が局所的に過保存で、かつ、そのキーがU.TopKey()以下であると仮定する。U.TopKey()は局所的に無矛盾な頂点の最小のキーであるため、そのキーはU.TopKey()と等しいはずである。したがって、C omputeShort-estPath()は、次に開始頂点を展開することができ、この場合、そのg値をそのrhs値に設定する。このとき、開始頂点はローカルに整合し、そのkeyはU.TopKey()以下となり、ComputeShortestPath()は終了する。この時点で、開始頂点のg値は、U.TopKey()と等しい。は、その目標距離となる。したがって、ComputeShort estPath()は、開始頂点がローカルに無矛盾でなく、かつ、そのkeyがU.TopKey()10以下であれば、既に終了することができる。この場合、ComputeShortestPath()が終了しても、開始頂点はローカルに無矛盾であり続け、そのg値はゴール距離と等しくないかもしれない（しかし、そのrhs値は等しい）。これは、ロボットがどのように動くべきかを決定するのにg値を使用しないため、問題にはならない。

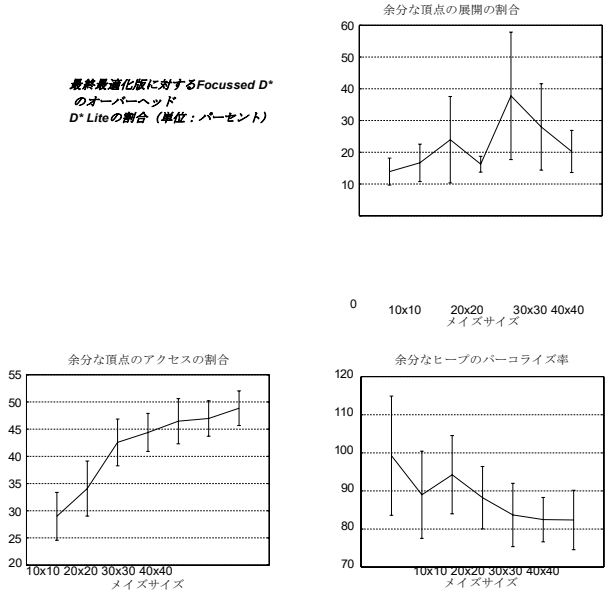
解析結果 D*

LiteのComputeShortestPathは、LPA*のCom-puteShortestPath()と似ており、多くの特性を共有しています。を持つ。例えば、D* LiteのComputeShortestPath()は、各頂点を最大2回、戻るまでパンドする。以下の定理は、D* LiteのComputeShortestPath()が終了し、正しいことを示す。

定理8　*D*　*LiteのComputeShortestPathは常にter-minateであり、D* Liteから最短経路をたどることができる。を最小化する任意の後継に、常に現在の頂点，から，へと移動することによって行われます。に達するまで（同点は任意に解消可能）。*

実験結果

次に、D*と最適化版D* Liteの各バージョンを比較した。すべての手法は標準的なバイナリヒープを優先度キューとして実装した（ただし、フィボナッチヒープのようなより複雑なデータ構造を優先度キューとして使用すれば、U.Update()



がより効率的になる可能性がある）。ロボットは常に隣接する8つのセルのうちどれが通過可能かを観察し、そのうちの1つに移動する。このとき、任意の2つのセルのX座標とY座標の差の絶対値の最大値を、その距離の近似値として使用した。どの方法も同じようにロボットを移動させ、D*はすでに実際のロボットで大きな成功を収めているため、あとはシミュレーションを行うだけである。我々は、各方式の計画時間の合計を比較する必要がある。実際の計画時間は実装や機種に依存するため，我々の性能比較の結果を他者が再現することは困難である．そこで

図5：D* LiteとD*の比較。

すなわち、頂点拡張の総数、ヒープパーコレーション（ヒープ内の親と子の交換）の総数、頂点のアクセス（例えば、値の読み取りや変更）の総数である。図5は、7種類の大きさの未知の地形（「動機」のセクションで説明）において、障害物密度が10〜40%に変化する各サイズの50のランダムに生成された地形を平均したゴール指向ナビゲーションのD* LiteとD*を比較したものである。地形は一様な解像度を持つセルに離散化されている。図では，D*の3つの性能指標をD* Liteとの相対的な差分としてグラフ化している．このように，D* Lite は常に 0 点であり，0 点以上の手法は D* Lite よりも性能が低い．D* Liteは3つの指標すべてにおいてD*より性能が良く、D*と少なくとも同程度の効率であるという我々の主張を正当化している。また、図には95%信頼区間も示しており、我々の結論が統計的に有意であることを証明している。以下では、D* Liteが実装するインクリメンタルサーチとヒューリスティックサーチの組み合わせが、どの程度、インクリメンタルサーチやヒューリスティックサーチを上回るかを検討する。これは、未知の地形における目標指向ナビゲーションと未知の地形のマッピングという、異なるが関連する2つのタスクについて、前回の実験と同様のセットアップを用いて行うものである。

未知の地形における目標指向型ナビゲーション

図 6 は、前実験と同じ設定で、未知地形での目標指向ナビゲーションについて、D* Lite、ヒューリスティック探索なしの D* Lite、インク リメンタル探索（つまり A*）なしの D* Lite の比較を行ったものである。ヒューリスティック探索とインクリメンタル探索の両方を行わないD* Liteは、性能が低すぎてグラフ化が問題になるため、比較に含めないことにした。D* Lite は

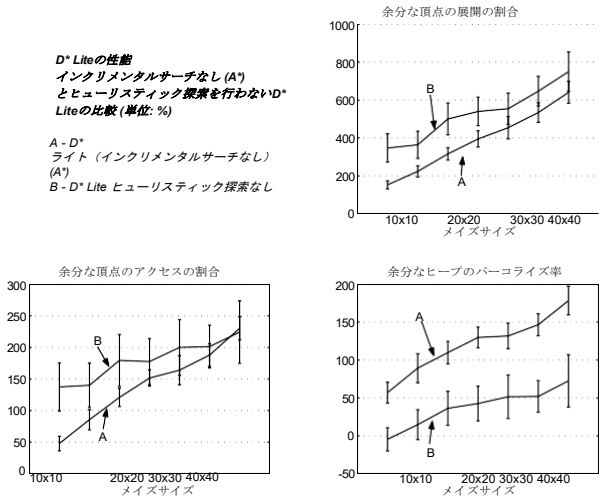


図6：目標指向型ナビゲーション（ユニフォーム）。

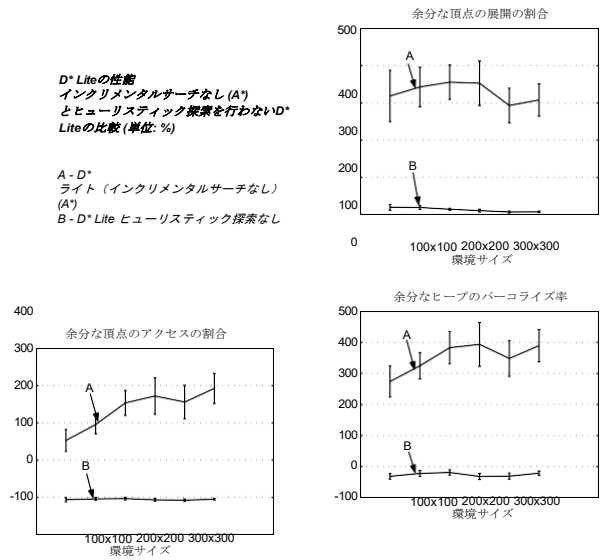


図7：目標指向型ナビゲーション（Adaptive）。

は、3つの性能指標すべてにおいて、他の2つの探索手法を上回り、頂点拡張では7倍以上の差をつけている。さらに、その優位性は地形が大きくなるにつれて増していくようである。地形サイズが10×10と15×15の場合、ヒープパーコレート数においてのみ、D* Liteとヒューリスティック探索なしのD* Liteの差は統計的に有意でなかった。これらの結果は、未知の地形における目標指向ナビゲーションにおいて、D*がA*を1桁以上上回るという以前の実験結果とも一致する(Stentz 1995)。

また、地形は不均一な解像度で離散化することができる。一様な離散化では、粒度が粗すぎると経路探索ができなくなり（例えば、障害物間の小さな隙間に気づかない）、粒度が大きすぎると効率的に探索できな

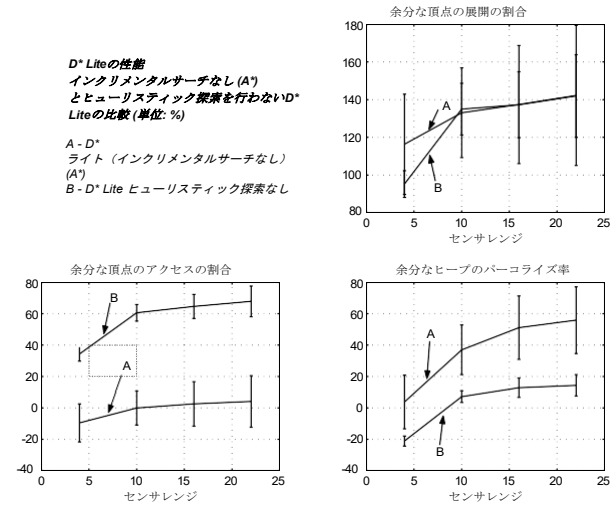


図8：マッピング（ユニフォーム）。

スキーム(Moore & Atkeson 1995; Yahja *et al.* 1998)がある。そこで我々は、D* Liteを用いて決定論的手法のParti-Game アルゴリズム (Moore & Atkeson 1995)に、地形を非一様な解像度のセルに離散化する適応離散化を加えたものである。図 7 は、障害物密度30%でランダムに生成された各サイズ 25個の地形を平均化し、6種類のサイズの未知地形における目標指向型ナビゲーションの D* Lite、ヒューリスティック探索なしのD* Lite、増分探索（つまり A*）なしの D* Liteを比較した結果である。D* Liteは、3つの性能指標すべてにおいて、増分探索を行わないD* Lite（つまりA*）を上回り、頂点拡張ではさらに4倍以上の性能向上を達成した。一方、一様離散化された未知の地形における目標指向ナビゲーションとは異なり、D* Liteとヒューリスティック探索を行わないD* Liteはほぼ同等の性能を発揮することがわかった。

未知なる地形のマッピング

いグラフになってしまう。そこで研究者たちは、適応的解像度

D* Lite はまた、貪欲マッピング (Koenig, Tovey, & Halliburton 2001) を実装するために使用することができる。これは単純だが強力なマッピング戦略で、異なる研究グループによってモバイルロボットに繰り返し使用されてきた (Thrun *et al.*1998; Koenig, Tovey, & Halliburton 2001; Romero, Morales, & Sucar 2001).貪欲マッピングは地形を一様な解像度を持つセルに離散化し、地形がマッピングされるまで、常に現在のセルから走行可能性が未知の最も近いセルまでロボットを移動させる。この場合、グラフは8連結の格子である。グラフの辺のコストは最初1である。このとき，グラフの辺のコストは1であり，ロボットがその辺を通れないと判断すると，コストが0になる．すべてのグリッドの頂点に接続された頂点が1つ追加される。これらの辺のコストは最初1である。対応するグリッドの頂点に到達すると、コストが無限大になる。このグラフに D* Lite を適用することで、

ロボットの現在の頂点をbeingとして、greedy mapping を実装することができる。

が追加された頂点である。

図8は、D* Lite、ヒューリスティック探索なしのD* Lite、インクリメンタル探索（つまり、A*）なしのD* Liteの比較である。

を、異なるセンサ範囲を持つ貪欲なマッピングのために、サイズ64×25のランダムに生成された50個のグリッドを平均化した。テールレインは一様な解像度を持つセルに離散化されている。ロボットのセンサーレンジを変化させ、短距離センサーと長距離センサーの両方をシミュレートした。例えば、センサーの範囲を 4
とすると、ロボットはロボットから任意の方向に 4
セルまで離れたすべての非可逆的セルを、他の非可逆的セルによって視界から遮られない限り感知することができる。D*
Liteの頂点展開の数は、他の2つの方法よりも常にはるかに少ない。これは、ヒープパーコレーションと頂点アクセスの回数についても同様である。ただし、ヒープパーコレーションのセンサーレンジ4と、インクリメンタルサーチなしのD*
Liteの頂点アクセス回数は例外である。

結論

本論文では、Focussed Dy-namic A*（D*）と同じナビゲーション戦略を実装した、未知の地形におけるロボットナビゲーションのための新しい高速リプランニング法であるD*
Liteを発表した。両アルゴリズムとも、ゴール頂点からロボットの現在の頂点に向かって探索し、ヒューリスティックを用いて探索を促進し、優先度待ち行列の並び替えを最小にするために同様の方法を用いています。D*
Liteは、確かな理論的基盤を持ち、A*と強い類似性を持ち、効率的で（g-
値がそれぞれのゴール距離と既に等しい頂点を展開しないため）、多くの方法で拡張されてきた我々のLPA*をベースにしています。このように、D*
LiteはD*とはアルゴリズム的に異なる。D*
LiteはD*とはアルゴリズムが異なるが、理解しやすく、拡張しやすく、かつD*と同程度の効率を持つ。我々は、D*
Liteに関する実験と解析の結果は、人工知能やロボット工学における高速リプランニング法のさらなる研究のための強力なアルゴリズム的基礎を提供し、人工知能におけるシンボリックリプランニング法の研究（Hanks & Weld 1995）や、アルゴリズム理論（Frigioni, Marchetti-Spaccamela, & Nanni 2000）や人工知能における漸進探索法の研究（Edelkamp 1998）を補足すると確信している。

謝辞

Anthony Stentz のこの研究に対する支援に感謝する。Intelligent Decision-Making Group は NSF の IIS-9984827, IIS-0098807, ITR/AP-0113881, IBM faculty fellowship award によって一部支援されている。この文書に含まれる見解と結論は著者のものであり、スポンサーである組織や機関、またはIBMの公式な方針（明示または黙示）を代表するものと解釈されるべきではない。
米国政府

参考文献

Edelkamp, S. 1998.最短経路の更新.*欧州人工知能会議論文集*, 655-659.

Frigioni, D.; Marchetti-Spaccamela, A.; and Nanni, U. 2000.最短経路樹を維持するための完全に動的なアルゴリズム。*アルゴリズムジャーナル*34(2):251-281.

ハンクス、S.、ウェルド、D.1995年。計画適応のための領域非依存型アルゴリズム。*人工知能研究* 2:319-360.

Koenig, S., and Likhachev, M. 2001.インクリメンタルA*.*神経情報処理システム論文集.*

ケーニッヒ，S.，トビー，C.，ハリバートン，W.，2001．地形の貪欲な地図作成．このような場合、「貪欲な地形マッピング」が有効である。

Likhachev, M., and Koenig, S. 2001.生涯計画A* と動的A* Lite:その証明。技術報告、College of Com-puting, Georgia Institute of Technology, Atlanta (Georgia).

このような状況下において、「偵察ロボット」を開発するためには、「偵察ロボット」としての機能を十分に発揮させる必要がある。このような場合，"偵察ロボット"と呼ばれる．このような場合、「偵察ロボット」と呼ばれる。

Moore, A., and Atkeson, C. 1995.多次元状態空間における可変解像度強化学習のためのparti-gameアルゴリズム。*機械学習*21(3):199-233.

Nilsson, N. 1971.*人工知能における問題解決の方法.* McGraw-Hill.

パール、J.
1985年*ヒューリスティック。コンピュータの問題解決のための知的な探索戦略。*Addison-Wesley.

Ramalingam, G., and Reps, T. 1996.最短経路問題の一般化に対するインクリメンタルアルゴリズム。このような場合、「*漸進的なアルゴリズム」が有効である。*

ロメロ、L.、モラレス、E.、スカー、E.2001.このような場合，"Scar "は "Scar "であることを意味し，"Scar "は "Scar
"であることを意味する．このような場合、「曖昧さ」を解消するために、「曖昧さ」を解消する方法として、*「曖昧さ」を解消する方法を提案する。*

このような場合、「未知環境における完全なナビゲーションシステム」が必要となる。未知環境における目標達成のための完全なナビゲーションシステム。このような場合、*「自律型ロボット」を選択することになる。*

Stentz, A. 1994.部分既知環境に対する最適かつ効率的な経路計画.このような環境下での最適・効率的な経路探索を実現するために必要な技術を紹介したのが、*「ロボティクスとオートメーションに関する国際会議」である。*

Stentz, A. 1995.実時間再計画のための焦点化 D* アルゴリズム.*人工知能に関する国際合同会議論文*集, 1652-1659.

セイヤー、S.、ディグニー、B.、ディアス、M.、ステンツ、A.、ナッベ、B.、ヘバート、M.2000.このような場合、「ディアス」「ステンツ」「ナッベ」「ヘベール」の3つのカテゴリーに分類される。*SPIE：Mobile Robots XV and Telemanipulator and Telepresence Technologies VII*, volume 4195.にて。

Thrun, S.; Bu¨cken, A.; Burgard, W.; Fox, D.; Fro¨hlinghaus, T.; Hennig, D.; Hofmann, T.; Krell, M.; and Schmidt, T. 1998.地図学習と高速ナビ。RHINOにおける地図学習と高速ナビゲーション。RHINOにおける地図学習と高速ナビゲーション。*人工知能を用いた移動型ロボット：成功するロボットシステムのケーススタディ*.MIT Press.21-52.

このような状況下で、移動ロボットのためのフレームドクワッドツリー経路計画。このような状況において、*ロボティクスとオートメーションに関する国際会議が開催される。*