Vol.2016-SLDM-176 No.3
2016/5/11

モンテカルロ木探索のTSP問題への適用

下村 聖人[1]　髙島康裕[1,a)]

概要：本稿では，最適化手法として注目されはじめているモンテカルロ木探索を，トラベリングセールスマン (TSP) 問題に適用した事例を報告する．評価として，一般的な確率的手法である Simulated Annealing と比較し，単純な実装では，十分な性能を発揮できないが，TSP 問題の特性を反映した実装を行なうと，非常に高い最適化能力を示し，モンテカルロ木探索の有効性を確認できた．

1. はじめに

近年，計算機の性能向上は目覚しいものがあり，それを利用した最適化手法も日々進歩している．しかし，問題規模も増大しており，高性能な最適化手法は求められている．特に，NP 困難 [1] に属している問題に対しての解法は，問題規模に対し，最適化時間が強く影響を受けるため，最適化手法は非常に重要である．

この最適化手法は，大きく分けて，1) 厳密解法，2) 近似解法，3) 発見的解法の 3 種類に分類できる．まず，1) の厳密解法であるが，最適解を失なうことなく解く手法であり，充足可能性判定手法 (SAT) [2] や整数線形計画法 (ILP) [3,4] 等に定式化して解く手法がその典型である．これらの手法は，この 10 年間に速度が飛躍的に向上し，かなりの問題規模まで高速に解くことができるようになっている．しかし，その一方で，指数的に計算時間が増大することは手法の特性上，避けえない．次に，2) の近似手法であるが，これはアルゴリズムから出力される解の性能が担保されているものである．[1] ではいくつかの近似手法が紹介されている．これは，厳密解法程遅くなく，解質もある程度保証されているという利点がある一方で，近似解法が「ほぼ」存在しないことが証明されている問題もあり，常に利用可能でないという問題点が存在する．最後に，3) の発見的手法であるが，これは，解質には保証が無い一方で，現実的な時間で解を得る手法である．このクラスに属する手法としては，Simulated Annealing(SA) 法や，Genetic Algorithm(GA) 法等が良く知られている．これらの手法はその枠組みとして，general な手法であり，どのような問題に対しても適用可能である．しかし，十分な時間をかけて最適化しないと良い質の解を得にくく，また，解に制約が厳しい場合，探索効率が悪化するということが知られている．

近年，モンテカルロ木探索という手法が，特にゲーム木探索と言った人工知能分野で注目を浴びている．また，この手法を LSI の EDA 等の最適化手法への応用も検討されている [5]．本稿では，この [5] の提案に沿って，モンテカルロ木探索を組み合わせ最適化問題の一つであるトラベリングセールスマン (TSP) 問題に適用した事例を報告する．その結果，SA 法と比較し，同程度の最適化時間であれば，40%程の解質の改良が認めらた．また，同程度以上の最適化性能を得るために，685 倍程度の高速化が確認できた．以上より，モンテカルロ木探索の有効性が確認した．

以降，2 節ではモンテカルロ木探索の概要を紹介し，7 節で本稿で考えるモンテカルロ木探索の TSP 問題への適用について説明する．そして，4 節で，モンテカルロ木探索の有効性を確認する実験について報告し，最後に 5 節でまとめる．

2. モンテカルロ木探索

モンテカルロ法とは，ランダムサンプリングを用いて，期待値等の統計値を求める手法である．近年，このモンテカルロ法を，木の探索に用いるモンテカルロ木探索が注目を浴びている．囲碁等のゲーム木探索に用いられたのが端緒である．しかし，その能力を最適化問題に適用するという事例も少しずつではあるが，提案されている．また，最近では，CAD 問題への適用も検討されている [5]．

図 1 にモンテカルロ木探索の概念図を示す．モンテカルロ木探索では，各ノードは部分解に対応し，終端ノードが解に相当する．図 1 においては，矩形のノードが終端ノード，すなわち解に対応し，それ以外のノードが部分解となる．各ノード間の枝が，ある部分解を元により大きな部分

[1] 北九州市立大学
〒 808–0135 福岡県北九州市若松区ひびきの 1–1
[a)] takasima@kitakyu-u.ac.jp

解を構成することに対応する．このとき，元となる部分解を親，新たに構成されたより大きな部分解が子と呼ばれる．図 1 では，ノード p に対し，ノード c1，c2，c3 が子となる．すなわち，ノード p は，ノード c1，c2，c3 の親である．子ノードには，展開済みと未展開ノードの 2 種類が存在する．図 1 では，ノード c1，c2 が展開済み，ノード c3 が未展開となっている．また，各終端ノードにおいて，評価が一意に計算できるとする．図 2 にモンテカルロ木探索

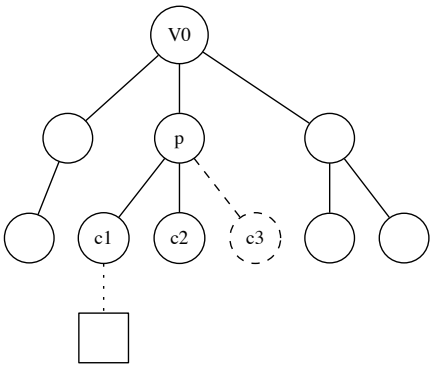

図 1 モンテカルロ木探索の概念図

の疑似コードを示す [5]．まず，V_0 は，探索の開始点である．繰り返し条件としては，時間や回数を予め定めておいて，それに達するまで繰り返すことを考える．先程の 1 において，図 6 の適用を考える．まず，TREE_POLICY において，探索がノード V_0 から開始する．ノード V_0 の子ノードは全て展開済みであるので，一番良い評価を持つ子ノードである p が選択される．p は未展開ノードが存在するので，その中から乱数で選ばれ，ノード c3 が V_l として選択される (図 3)．その後，DEFAULT_POLICY により，終端ノード Ve が選択され，その評価値を計算する (図 4)．そして，BACKUP により，情報が親ノードに伝搬され，情報を更新する (図 5)． 以降，各ステップ毎に説明する．なお，以下の説明では，最適化は最大化問題として説明する．

```
function MCT_SEARCH(V_0) {
    while (繰り返し条件を満たしている) {
        V_l ← TREE_POLICY (V_0)
        Δ ← DEFAULT_POLICY (V_l)
        BACKUP (V_l, Δ)
    }
    return BEST_CHILD (V_0)
}
```

図 2 モンテカルロ木探索の疑似コード

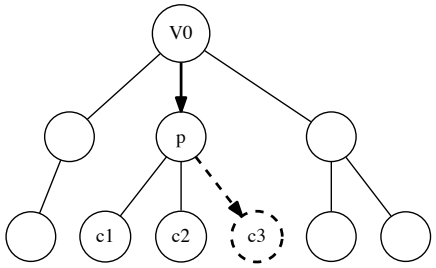

図 3 TREE_POLICY の例

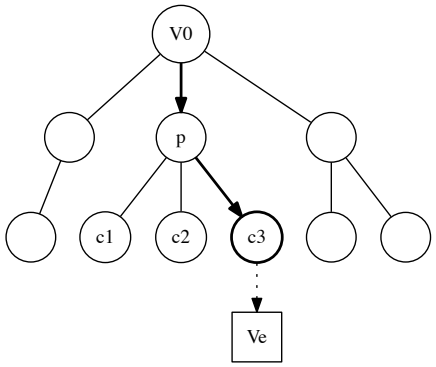

図 4 DEFAULT_POLICY の例

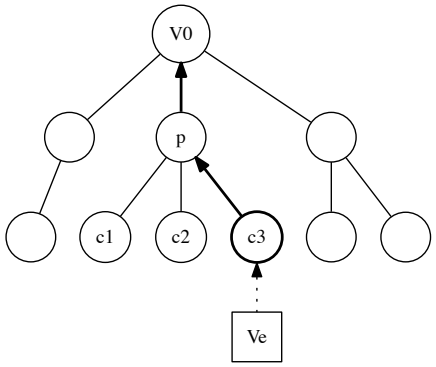

図 5 BACKUP の例

2.1 TREE_POLICY

このステップでは，K 個ある選択肢から統計的に最良のものを選択する Multi-Armed Bandit 問題と呼ばれる問題として定式化する．図 6 に疑似コードを示す．この問題は，十分な数の試行があれば，確度の高い選択が可能となる．しかし，実際には，試行回数には制限あるため，その上での選択する必要がある．それに対しては，Upper Confidencial Bound for Trees (UCT) と呼ばれるアルゴリ

```
function TREE_POLICY (V) {
  while (V が終端ノードでない) {
    if (V の子ノードが全て展開済み) {
      V の子供で指標 UCT(式 (1)) が最も良い子を
      V とする.
    } else {
      V の子供のうち，展開されていない子供 V′ を選択.
      return V′
    }
  }
  return V.
}
```

図 6 TREE_POLITY の疑似コード

ズムが提案されている．これは，ノード j に対し，式 1 で示される式で評価し，その評価が最も高いものを選択する手法である．

$$\mathrm{UCT} = \overline{x_j} + 2C_p\sqrt{\frac{2\ln n}{n_j}} \qquad (1)$$

ここで，$\overline{x_j}$ は以前の探索でのノード j における解の評価の平均値，n は総探索数，n_j はその中でノード j を選択した回数である．また，C_p は重みである．つまり，基本は，以前の探索でのノード j における解の評価の平均値の高いノードが評価が高くなる．しかし，過去の探索回数が少ない場合は，良い解が存在する可能性が高いため，第 2 項により，ばらつきを考慮している．

2.2 DEFAULT_POLICY

このステップでは，TREE_POLICY で選択されたノードに対応した部分解から乱数を用いて解を構成し，評価する．図 7 に疑似コードを示す．

```
function DEFAULT_POLICY (V) {
  while (V が終端ノードでない) {
    V の子 V′ を乱数で選択.
    V に V′ を代入.
  }
  return (V の評価値)
}
```

図 7 DEFAULT_POLICY の疑似コード

2.3 BACKUP

このステップでは，DEFAULT_POLICY で得られた解と評価値を用いて，開始点に再帰的に戻りながら，TREE_POLICY で用いる値を更新する．具体的には，DEFAULT_POLICY で得られた解と評価値に対し，探索木を開始点に戻りながら，各ノードの平均 $\overline{x_j}$ と，探索回数 n_j を更新する．

3. モンテカルロ木探索の TSP 問題への適用

本稿では，モンテカルロ木探索をトラベリングセールスマン (TSP) 問題に適用した事例を報告する．ここで，本稿で考える TSP 問題の定義を定義 1 に示す．

定義 1 (トラベリングセールスマン (TSP) 問題)
入力: グラフ $G = (V, E)$，枝重み $w(e|e \in E)$
出力: 最小パス長
制約: パスは各点を丁度一度通過する

今回の TSP ではパスを探索する問題を考慮する．この問題は，NP 困難な問題であることが知られており [1]，多項式時間で最適解を出力することはほぼ無理であると考えられている．この問題に対し，様々な解法が提案されており，特に [1] では，最小全域木 (MST) を利用した近似解法が紹介されている．

本稿では，この TSP に対しモンテカルロ探索木の適用を検討した．以下にその概要を述べる．

3.1 解表現

まず，TSP において，解は，頂点の順列である．そのため，部分解は部分順列として表現する．

3.2 TREE_POLICY

次に，TREE_POLICY においては，指標 MCT が必要である．TSP 問題は最小化問題であるため，式 (1) を式 (2) のように変更する．この変更は，第 2 項の符号が − になっていることである．

$$\mathrm{UCT} = \overline{x_j} - 2C_p\sqrt{\frac{2\ln n}{n_j}} \qquad (2)$$

また，TREE_POLICY で最良の指標 MCT とは，この式 (2) が最小となるノードを選択する．なお，C_p の決定については，特に定まった手法が無い．そこで，(1) MST の値を利用する方法，と，(2) サンプリング値を利用する方法，を検討した．まず (1) については，グラフに対し，MST を求め，その大きさを利用して，C_p を決定する手法である．また，(2) は，TREE_POLICY のアルゴリズムでは，開始ノードの直下の子ノードを展開するときは，UCT を利用していない．そこで，開始ノードの直下の展開が終了した時点での各ノードの経路長の標準偏差を計算し，その値を利用して，C_p を決定する．これらの結果は，4 節で示す．

3.3 DEFAULT_POLICY

DEFAULT_POLICY に関しては，ランダムに解を生成する部分に多様性が存在する．ここでは，(1) 一様乱数を用いた手法，と，(2) ルーレット選択を利用した手法，の 2 通りを検討した．まず，TREE_POLICY で選択されたノードに対応する部分経路に対し，未訪問の頂点から次に探索する頂点を決定する．ここで，(1) の一様乱数を用いた手法では，未訪問の全頂点が等確率で選択される手法である．(2) のルーレット選択を利用した手法では，部分経路の最後の頂点から未訪問の各頂点への枝重みの逆数を計算し，その総計に占める割合を確率として選択する手法である．これらに関しては，(1) では (2) と比較して簡単に計算できる為，処理が高速になることが期待される．しかし，最小パス長を考えると連続する 2 点間は小さいことが望ましいとできるため，より良い解を生成しやすいと考えられる．これらの結果も，4 節で比較を報告する．

3.4 BACKUP

このステップに関しては，特段の工夫は無い．ただ，各ノードに対応する部分経路を含む中で，これまでの探索で最良の解を記憶している．

4. 実験

提案手法を評価するため，計算機に実装し，実験を行なった．使用した計算機環境は，プロセッサ Intel Core i5 3.2GHz，メモリ 32GB 1600 MHz DDR3, OSX 10.11.4 である．また，実験は，Simulated Annealing (SA)，モンテカルロ木探索 [一様乱数+MST の係数倍] (MCT_UM)，モンテカルロ木探索 [ルーレット選択+MST の係数倍] (MCT_RM)，モンテカルロ木探索 [ルーレット選択+標準偏差の係数倍] (MCT_RS) で行なった．SA は，初期温度 10000 度，終了温度 10 度，各温度での繰り返し回数 1000 回，冷却係数 0.99 で行なった．また，摂動は，任意の 2 点を選び，順番を入れかえることで実現している．そして，モンテカルロ木探索では，終了条件として，SA でかかった時間までとしている．

ベンチマークは [6] で提供されているベンチマークデータから，77 個のデータを抜粋して利用した．頂点数は，48 点から 15112 点までの問題となっている．

まず，DEFAULT_POLICY での乱数の違いによる性能を確認する実験を行なった．ここでは，$C_p = 2 * (\text{最小全域木の長さ})$ とした結果を実験結果を図 8 に示す．ここで，横軸はベンチマークデータの頂点数を，縦軸は最小全域木の大きさで正規化した経路長をそれぞれ対数目盛で示している．この結果によると，一様分布を用いた場合，SA の結果よりも若干悪くなっている傾向がある．一方，ルーレット選択の場合，かなりの改善が見られる．SA の結果に対し，77 個の平均では，一様分布では 1.08 と 8%程長くなっているのに対し，ルーレット選択は，0.632 と 37%程の改善となった．この結果により，DEFAULT_POLICY の乱数での選択は，最適化性能に大きな影響を及ぼすことがわかる．

更に今回のルーレット選択の有効性を確認するため，SA の結果を上回るのに必要な時間を計測した．ここで，モンテカルロ木探索としては，ルーレット選択，かつ，$C_p = 2 * (\text{最小全域木の長さ})$ を用いて，探索を行なった．その結果，SA の実行時間に対し，平均 0.15%程度の時間で，結果を上回ることが可能であった．すなわち，同程度以上の結果であれば，685 倍程度の高速化が実現できている．

次に C_p の値による影響を確かめた．図 9 に結果を示す．ここでも先程と同様，横軸はベンチマークデータの頂点数を，縦軸は最小全域木の大きさで正規化した経路長を，それぞれ対数目盛で示している．DEFAULT_POLICY としてははルーレット選択を，また，k*MST は C_p として最小全域木の k 倍を，k*SD はサンプリングにおける標準偏差の k 倍を用いた結果である．これによると，ほとんど差が生じていないことがわかる．実際，77 個の SA からの改善量の平均では，0*MST が 0.628，2*MST が 0.632，4*MST が 0.633，0*SD が 0.629，2*SD が 0.633，4*SD が 0.634 と 1%程度の違いである．特に，0*MST や 0*SD は，式 (2) の第 2 項の効果を無くすことに対応しており，この項があっても無くても同程度の結果であることを示している．これは，TSP において，今回用いたルーレット選択が十分効果的であり，その結果，C_p を利用したばらつきの考慮が効果を発揮しないことにあると考えられる．しかし，サンプリング量を利用した C_p 決定法は，解の範囲を想定することなく利用でき，別の問題に適用するときには有効であることが期待できる．

5. まとめ

本稿では，最適化手法として注目を受けているモンテカルロ木探索を TSP 問題に適用した．その結果，SA と比較し，1) 同程度の時間であれば，40%程の経路長が削減できること，2) 同程度以下の経路長を得るためには，685 倍程度の高速化が実現できること，を確認した．また，この結果を得るには，DEFAULT_POLICY で用いる乱数による選択が大きな影響を及ぼすことも確認した．一方で，式 (2) での第 2 項の影響は，今回の実装ではあまり見ることができなかった．以上より，モンテカルロ木探索の最適化問題への有効性が確認できた．

今後の課題は，1) EDA 問題への適用，2) 適切な MCT の決定法，が上げられる．1) に関しては，DEFAULT_POLICY の選択が結果に大きな影響を及ぼすことがわかったので，問題に応じて適切なものを考える必要がある．2) に関して

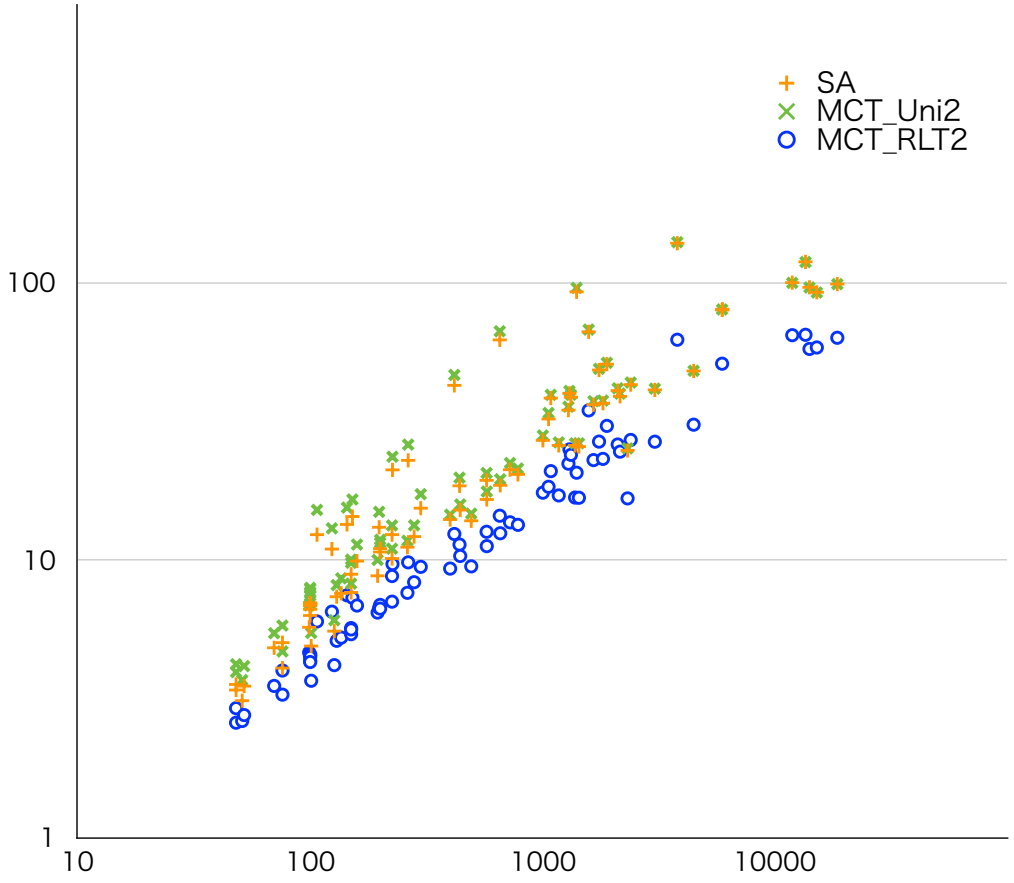

図 8 DEFAULT_POLICY の違いによる経路長の比較

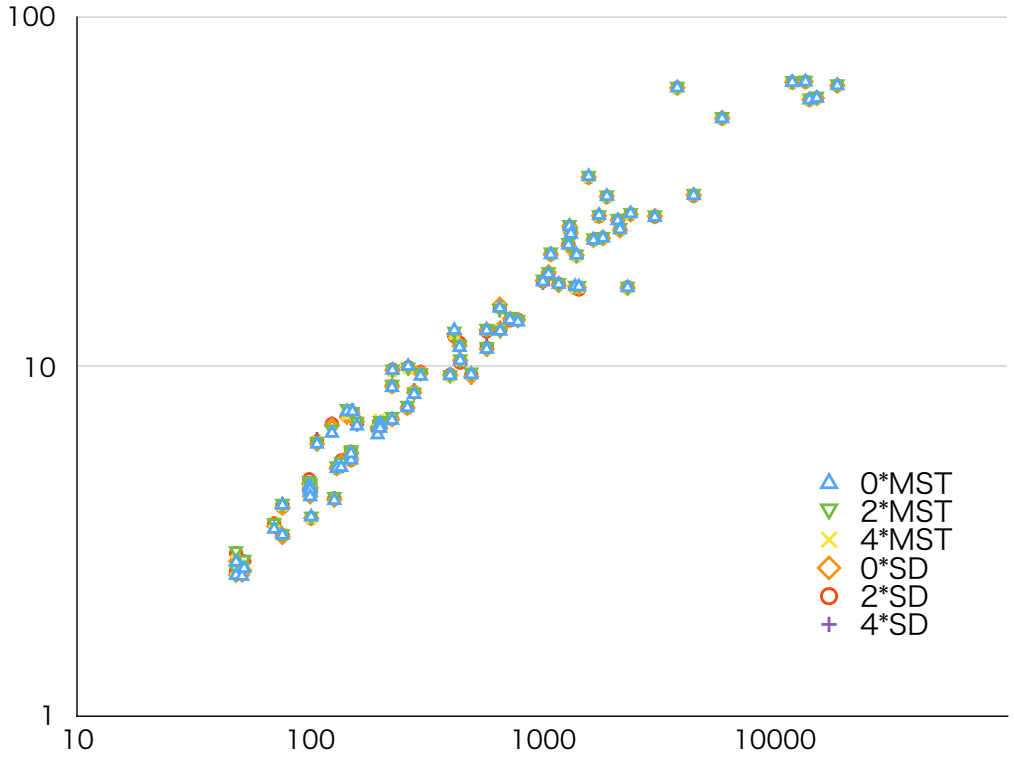

図 9 C_p の違いによる経路長の比較

は，今回の TSP 問題では，あまり大きな差が発生していない．しかし，問題によっては，結果に大きな影響を及ぼすことが考えられるので，考慮が必要であると考えている．

参考文献

[1] M. R. Garey and D. S. Johnson, “COMPUTERS AND INTRACTABILITY”, 1979.
[2] MiniSat, http://minisat.se.
[3] IBM CPLEX Optimizer, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.
[4] Gurobi Optimizer, http://www.gurobi.com.
[5] 松永 裕介, “モンテカルロ木探索の CAD 問題への応用について”, 信学技報, VLD2015-46, pp. 51–55, 2015.
[6] Symmetric traveling salesman problem, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/.