モンテカルロ木探索ガイスターにおける遺伝的プログラミングを用いたプレイアウト方策の作成

栃川純平[1]　竹内聖悟[2]

概要：ガイスターにおけるモンテカルロベースのプレイヤーは，プレイアウトではランダムプレイアウトを行っている．ランダムプレイアウトでは，指手をランダムに選択しているため実際には指さないような手を指すことが多くなる．これを少なくするために囲碁や General Game Playing，カードゲームではプレイアウトにある程度人間の知識を使用したルールベースの方策を持たせているものがある．Ms.Pac-Man において遺伝的プログラミング (GP) を使ってプレイアウトポリシーを作成する研究があった．本研究では，GP を用いてプレイアウト方策を作成し，モンテカルロベースプレイヤーの性能向上を目指す．GP で方策の作成を行い，得られた方策を使用したモンテカルロ木探索プレイヤーとそうでないプレイヤーで対戦実験を行った結果から提案手法の有効性を確認した．

Using Genetic Programming to Create Playout Policy fo Monte Carlo Tree Search in Geister

JUNPEI TOCHIKAWA[1]　SHOGO TAKEUCHI[2]

Abstract: In Geister, Monte Carlo-based players use random playouts. Since the move selection is random, unrealistic moves are often chosen. To reduce this problem, rule-based policies using human knowledge are used in Go, General Game Playing, and card games. There was a research on creating a playout policy using genetic programming (GP) in Ms.Pac-Man. In this study, we aim to improve the performance of player by creating playout policy using GP. We confirmed the effectiveness of the proposed method from the results of the experiments using the created policies.

1. はじめに

ガイスターは，相手の駒の色がわからない二人不完全情報ゲームである．ガイスターでは，モンテカルロベースのプレイヤー [1][2] やミニマックス法を用いたプレイヤー [3] などがある．ガイスターにおいて，モンテカルロベースプレイヤーではランダムプレイアウトを行っている．ランダムプレイアウトでは，指手をランダムで選択しているためを指すため実際には指さないような手を指すことが多くなる．これを少なくするために，囲碁や General Game Playing，カードゲームでは，プレイアウトにある程度人間の知識を使用したルールベースの方策を持たせているものがある．そこで，ガイスターにおいてもプレイアウト時に人間の知識を用いたルールベースの方策を持たせることで性能向上を試みる．

方策の作成では，ニューラルネットワークを学習させることが多くなってきているが，今回はプレイアウト中で使用することもあり速度が求められるためニューラルネットワークは不適切であると考えた．Ms.Pac-Man では遺伝的プログラミング (Genetic programming 以下 GP) でルールベースの方策を作成し，プレイアウトで使用することで性能が向上していた [4]．そのため，本研究では方策の作成に GP を使用することにした．しかし，ガイスターは Ms.pac-man と違い不完全情報ゲームである．不完全情報ゲームでは，毎回ルート局面で不完全情報の決定化をし，完全情報化を行っているものがある [5]．そのため，ガイス

[1] 高知工科大学大学院 工学研究
[2] 高知工科大学 情報学群

ターのプレイアウトでは，自分の駒色の配置は正確であるが相手の配置は正しいとは限らないため，各プレイヤーで駒色情報を使用する指手の信頼性に違いが出てくるため，色情報を使用したルールベースの方策の効果があるか不明である．

本研究では，ガイスターにおいて GP を用いてプレイアウト方策を作成することでモンテカルロベースプレイヤーの性能向上を目指す．

2. ガイスター

ガイスターは，互いのプレイヤーが青駒と赤駒を各 4 個ずつ計 8 個の駒を用いて行う二人不完全情報ゲームである．このゲームは，各プレイヤーは自身の駒の色は確認することができるが相手の駒の色は確認することができないという特性を持っている．ゲームは，6×6 の盤面で行い，以下の勝利条件を目指す．

- 自分の青駒を脱出させる
- 相手の青駒を全て取る
- 自分の赤駒を全て取らせる

脱出は自陣の反対側にある脱出口 (図 1 の黒いマス) である．ゲーム開始時に各プレイヤーは決められた場所に 8 個の駒を自由に配置し，交互に駒を動かしていく．自身のターンでは，自身の駒 1 つを上下左右に 1 マス動かすことができる．移動先に相手の駒がある場合はその駒を取ることができ，取った駒の色を確認することができる．ゲームを進めていきどちらかのプレイヤーが勝利条件のいずれかを満たしたらゲーム終了である．

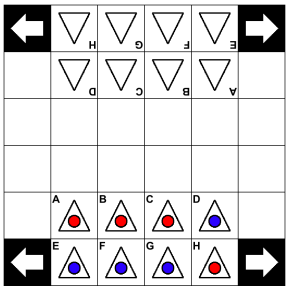

図 1 ガイスターの盤面

3. 関連研究

3.1 モンテカルロベースのガイスタープレイヤー

モンテカルロベースのガイスタープレイヤーとして三塩らのプレイヤー [1] や鴛淵らのプレイヤー [2] がある．これらのプレイヤーは，どちらも推定を行っており，推定した内容は相手駒の配置にのみ使用されている．プレイアウトはランダムプレイアウトを行っている．ランダムプレイアウトでは，指手がランダムに選択されるため実際には指さないような手を指すことがが多くなってしまう．

3.2 遺伝的プログラミング (GP)

遺伝的プログラミングとは，関数の集合を用意しておき，その関数の組み合わせにより与えられた現象を説明できるプログラムを求めることである [6]．GP では遺伝子を木構造で表現しており，ランダムに個体の集合を生成し評価を行い，その中で最もよかった個体を親として次の世代を作成していく．本研究では，プレイアウトの方策を個体としている．次の世代を生成するときに，突然変異や交叉など遺伝的操作を適応することでプログラムを進化させていく．

これを過程として表すと以下の通りになる

(1) 初期個体の集合を生成
(2) 各個体の適応度を評価
(3) 最もよかった個体を親として選択
(4) 突然変異や交叉などを交えながら，次の世代を生成
(5) (1)〜(4) を世代数繰り返す

3.3 GP を用いたルールベース方策の作成の研究

GP を用いプレイアウトで使用するルールベースの方策を作成する研究として，Ms.Pac-Man を題材にしたもの [4] がある．この研究では，GP を用いてルールベースの方策を作成，モンテカルロ木探索のプレイアウトで使用した結果，平均スコアがランダムプレイアウトの時よりも 18%向上したという結果が得られている．

4. 提案する内容

本研究では，ガイスターにおいて GP を用いてプレイアウト中の方策を作成することでモンテカルロベースプレイヤーの性能向上を目指す．プレイアウト方策は，事前にいくつかの関数を定義し，それを用いて GP によって作成する．定義する関数は，ガイスターのルールを基に必要と思われるものを実装した．関数の内容に関しては第 4.1 節で説明する．また，GP 内で使用する適応度を定める必要がある．適応度については第 4.2 節で述べる．今回用いるプレイヤーは，作成したプレイアウト方策を使用したモンテカルロ木探索プレイヤーである．対戦実験で性能評価を行う．

4.1 GP で用いた関数セット

本研究では以下に述べる関数を実装し，GP で使用した．下記に示したもの以外にも，if_then_else，比較演算子，論理演算子，0 以上 6 未満のランダムな整数値を使用している．

4.1.1 条件として使用する関数

- **自身の青駒の残数を取得 (getNumOfMyBlue)**
  整数値を返す，if 文の条件式に用いる関数である．ガイスターの敗北条件に「自身の青駒を全て取られる」があるため，自身が負けそうかの指標にできると考えた．自身が負けそうかということは行動選択のために必要な要素だと考えたため実装した．

- **自身の赤駒の残数を取得 (getNumOfMyRed)**
  整数値を返す，if 文の条件式に用いる関数である．ガイスターの勝利条件に「自身の赤駒を全て取らせる」があるため，自身が勝てそうかの指標にできると考えた．自身が勝てそうかということも行動選択のために必要な要素だと考えたため実装した．
- **相手の青駒の残数を取得 (getNumOfOppBlue)**
  整数値を返す，if 文の条件式に用いる関数である．ガイスターの勝利条件に「相手の青駒を全て取る」があるため，自身が勝てそうかの指標にできると考えた．「自身の赤駒の残数を取得」と同様の理由より実装した．
- **相手の赤駒の残数を取得 (getNumOfOppRed)**
  整数値を返す，if 文の条件式に用いる関数である．ガイスターの敗北条件に「相手の赤駒を全て取る」があるため，自身が負けそうかの指標にできると考えた．「自身の青駒の残数を取得」と同様の理由より実装した．
- **脱出口までの最短の距離を取得 (getMinimumDistanceToGoal)**
  整数値を返す，if 文の条件式に用いる関数である．ガイスターの勝利条件に「自身の青駒を脱出させる」があり，脱出口に駒を進めることが重要であると考えられた．脱出口までの最短の距離が，脱出口に駒を進めるなどの行動選択の閾値になると考えたため実装した．
- **自陣脱出口までの最短の距離を取得 (getMinimumDistanceToMyGoal)**
  整数値を返す，if 文の条件式に用いる関数である．ガイスターの敗北条件に「相手に脱出される」があり，伊藤らの研究 [7] において，キーパー戦略が一定の有効性があることが示されている．そのため，自陣に戻りやすいように，自陣脱出口までの距離が必要だと考えた．
- **相手駒と隣接しているか (isNextToOpponents)**
  真理値を返す，if 文の条件式に用いる関数である．ガイスターのルール上駒取りがあり，勝利条件にも関わってくるため必要な要素と考え，実装した．
- **自陣脱出口付近に相手駒がある (isOppNearMyGoal)**
  真理値を返す，if 文の条件式に用いる関数である．ガイスターの敗北条件に「相手の青駒に脱出される」があり，伊藤らの研究 [7] において，キーパー戦略が一定の有効性があることが示されている．そのため，脱出口周辺の状況が重要であると考えた．今回は，脱出口と脱出口に隣接しているマスと脱出口から斜め上のマスの 4 マスについて調べている．
- **自陣脱出口付近に自駒がある (isHavingKeeper)**
  真理値を返す，if 文の条件式に用いる関数である．「自陣脱出口付近に相手駒がある」と同じ理由から実装した．調べている範囲も同じである．

4.1.2 指手選択として使用する関数

- **ランダムに手を指す (doRandomMove)**
  指すことが出来る手からランダムに選択して指す関数である．指して選択の基本的なものとして用意した．
- **隣接している相手駒を取る (doCaptureMove)**
  隣接している相手駒があれば，その駒を取る手を選択し指す関数である．指手が複数存在している場合，その中からランダムに選ばれる．勝利条件に「相手の青駒を全て取る」があり，優先的に相手駒を取るものが必要だと考えた．
- **最も脱出口に近い自駒を脱出口に近づける (doMoveCloserToGoal)**
  脱出口に最も近い自駒を脱出口に近づくように指手を選択し指す関数である．盤面を左右の半分に分け，脱出口方向と左側のとき左，右側のとき右を選択するようにしている．勝利条件に「自分の青駒を脱出させる」があるため，脱出口に向かわせる行動は重要だと考えた．
- **隣接している相手駒から逃げる (doEscapeMoveFromOpponents)**
  隣接している相手駒から離れるような指手を選択し指す関数である．隣接している相手駒がある場合，その駒から離れるような指手のうち，移動先も相手駒と隣接していない指手を選択する．逃げる指手であるにも関わらず，指した後すぐに取られてしまってはいけないと考え，このような選択にした．敗北条件に「自身の青駒を全て取られる」があるため，相手から逃げることも必要だと考えた．
- **脱出口から遠ざける (doMoveLeavingFromGoal)**
  脱出口から遠ざけるような指手を選択し指す関数である．盤面を左右の半分に分け，脱出口逆方向と中央方向を選択するようにしている．勝利条件に「自分の青駒を脱出させる」があるため，脱出口に向かわせることは重要であると考えられる．しかし，脱出口に向かわせると駒を取られるリスクが大きくなるため，脱出口から遠ざける指手も必要になると考え，実装した．
- **最も自陣脱出口に近い自駒を自陣脱出口近づける (doMoveCloserToMyGoal)**
  最も自陣脱出口に近い自駒を自陣脱出口に近づける指手を選択し指す関数である．盤面を左右の半分に分け，自陣脱出口方向と左側のとき左，右側のとき右を選択するようにしている．伊藤らの研究 [7] において，キーパー戦略が一定の有効性があることが示されている．そのため，自陣脱出口付近に自駒があることが重要になってくると考え実装した．

指手を指す関数では，該当する手が存在しない場合，ランダムな手を指すようにしている．

4.1.3 色情報を使用する関数

- **隣接している駒が青駒なら取る (doCaptureBlueOpponentMove)**
  色情報を使用し隣接している青駒があるなら，青駒を取る手を選択し指す関数である．勝利条件に「相手の青駒を全て取る」があり，青駒と隣接しているなら優先的に取るほうが良いと考えた．
- **隣接している駒が赤駒なら逃げる (doEscapeMoveFromRedOpponent)**
  色情報を使用し隣接している赤駒があるなら，赤駒から離れるような指手を選択し指す関数である．隣接している駒が赤駒の時に限定しており，動きとしては「隣接している相手駒から逃げる」と同じである．敗北条件に「相手の赤駒を全て取る」があり，赤駒と隣接していたとき，それをとってしまい敗北に一歩近づく恐れがある．そのため，それを避けるようにするために実装した．

4.2 GP の適応度

本研究では，ある局面に対して Naotti-2020[*1] の指手との一致率を適応度として用いた．一致率を適応度として用いた例として，遺伝的アルゴリズムをチェスで行ったもの [8] がある．この研究では，強い人間の指手との一致率を適応度として採用していた．ガイスターでは，プロの人間がいないため強い AI の指手との一致率を適応度として採用することが妥当だと考えた．そこで，GAT2020 のガイスター AI 大会で優勝した Naotti-2020 は，ガイスタープレイヤーの中でも強いプレイヤーであるため，Naotti-2020 の指手を教師データとすることとした．

文献 [4] では実際にシミュレーションを行い，スコアを適応度として用いた．ガイスターにおいて対戦の勝率を適応度とすることもできるが，時間が掛かることから現実的ではない．

5. 実験

本実験では，GP によりプレイアウト方策を作成し，作成した方策を使用したモンテカルロ木探索プレイヤーで対戦実験を行う．

GP は，個体数 100 と 300，世代数 30 と 50 で行った．関数セットは第 4.1 節の物のうち，色情報を使用しない物のみの場合と使用するものも含めた場合で行った．実験では，過去のログから 1,000 局面用意し，その局面において Naotti-2020 が指す手との一致数を適応度とした．Naotti-2020 の探索の深さは 7 である．指手の決定はプレイアウト方策を用いたモンテカルロ木探索を使用している．コンパイルが失敗した場合は，大きな負の値とした．

[*1] https://github.com/kawakami-naoto/Naotti-2020

対戦実験は，1000 試合行い，勝利数と引き分け数，勝率を調べた．勝率は，勝利を 1 勝，引き分けを 0.5 勝として計算した．対戦相手は，モンテカルロ木探索プレイヤー MCTS とした．

実験で使用するプレイヤーのプレイアウト回数は 10,000 回とする．また，ルート局面で実現可能な相手駒色の配置で決定化を行い，完全情報化を行っている．これは，1 プレイアウトごとに行っている．

6. 結果

GP によって作成された方策はアルゴリズム 1〜4 である．掲載されているものは，出力されたものそのものでなく，重複や絶対に起こり得ないところを省いたものとなっている．

適応度の推移は図 2 と図 3 の通りである．世代数 30 が図 2，世代数 50 が図 3 となっている．赤いグラフが個体数 100，青いグラフが個体数 300 を表しており，実線が適応度の最大値，破線が適応度の平均値を表している．

適応度は世代が進むにつれて上がっていった．図 2 と図 3 から，同じ世代数でも個体数が 100 よりも 300 の方が適応度が高い傾向があった．

対戦結果は，表 1 の通りである．全ての場合において，勝率は 50%を越え，個体数が 300 の時は勝率 60%を越えた．世代数の変化による勝率の変化は見られなかったが，個体数が 100 と 300 とでは個体数が 300 の時の方が勝率が良くなった．

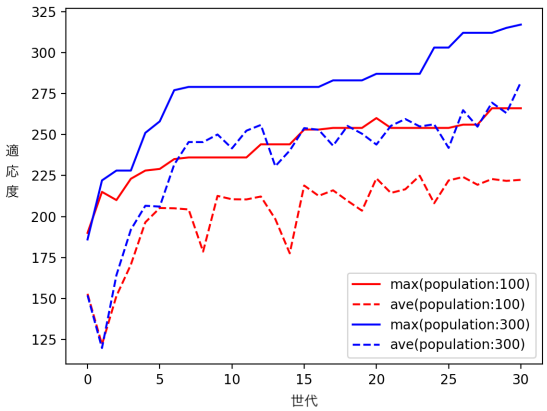

図 2 世代数 30 の適応度の推移 (色情報なしの場合)

表 1 MCTS との対戦結果 (色情報なしの場合)

個体数	世代	勝利	引き分け	勝率
100	30	511	59	54.05
100	50	486	111	54.15
300	30	613	40	63.3
300	50	621	30	63.6

色情報ありの場合の適応度の推移は，図 4 の通りである．

Algorithm 1 個体数:100，世代:30 でできた方策

```
if getMinimumDistanceToGoal=0 または isOppNearMyGoal
かつ
(!(isNextToOpponents) または isHavingKeeper) then
    doRandomMove
else
    if isNextToOpponents then
        doCaptureMove
    else
        doMoveCloserToGoal
```

Algorithm 2 個体数:100，世代:50 でできた方策

```
if isNextToOpponents かつ isHavingKeeper then
    doCaptureMove
else
    doMoveCloserToGoal
```

Algorithm 3 個体数:300，世代:30 でできた方策

```
if isOppNearMyGoal == getMinimumDistanceToGoal then
    if isOppNearMyGoal then
        doMoveCloserToMyGoal
    else
        doEscapeMoveFromOpponents
else
    if getMinimumDistanceToGoal=0 かつ
(0 != isNextToOpponents != getNumOfOppRed) then
        if isOppNearMyGoal then
            if isNextToOpponents then
                doRandomMove
            else
                doMoveCloserToGoal
        else
            doEscapeMoveFromOpponents
    else
        doMoveCloserToMyGoal
```

Algorithm 4 個体数:300，世代:50 でできた方策

```
if isOppNearMyGoal ≤ getMinimumDistanceToGoal then
    if isOppNearMyGoal > getMinimumDistanceToMyGoal
then
        doCapturMove
    else
        doMoveCloserToMyGoal
else
    if (getNumOfOppRed が 1 かつ !(isHavingKeeper)
 または (isNextToOpponents かつ getNumOfOppRed ≤ 2))
 または (getNumOfMyBlue ≤ getMinimumDistanceToMy-
Goal == getNumOfOppRed == isOppNearMyGoal)
 または (isOppNearMyGoal かつ getMinimumDistanceTo-
Goal=0) then
        if ( thenisOppNearMyGoal)
            doRandomMove
        else
            doMoveCloserToMyGoal
    else
        doMoveCloserToGoal
```

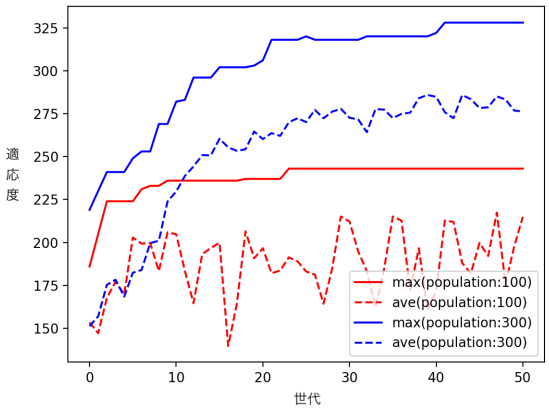

図 3 世代数 50 の適応度の推移 (色情報なしの場合)

赤いグラフが個体数 100，青いグラフが個体数 300 を表しており，実線が適応度の最大値，破線が適応度の平均値を表している．

個体数の違いよる最終的な適応度に差はなかった．個体数 100 の場合は，10 世代を越えたあたりから適応度の上昇が止まった．

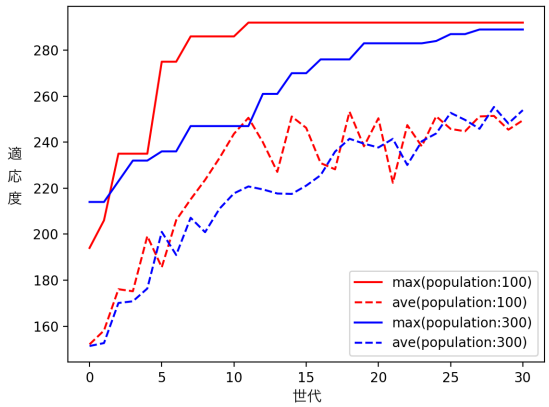

図 4 世代数 30 の適応度の推移 (色情報ありの場合)

色情報ありの場合の対戦結果は，図 2 の通りである．勝率は 50%を越えているが，個体数による勝率の差はなかった．

表 2 MCTS との対戦結果 (色情報ありの場合)

個体数	世代	勝利	引き分け	勝率
100	30	513	51	53.85
300	30	521	28	53.5

7. 考察

作成された方策では，敵駒と隣接していたら「隣接している相手駒を取る」，そうでないなら「最も脱出口に近い駒を脱出口に向かわせる」など妥当な方策を含むもの得られた．しかし，得られた方策は，最適化されておらず解釈す

ることが困難であり，人の手で最適化を行わなければいけない状態であった．

図 2 と図 3 から世代が進むにつれて適応度が上昇しているため，学習はうまく動作していると考えられる．図 3 を見てみると，30 世代くらいには適応度が最終的な最大値に近い値になってしまっている．対戦実験では，個体数が 300 の方が勝率が高くなったが，世代数の変化では勝率に差は生じなかった．個体数が多い方が，最初の世代でより良い個体が出現しやすくなり，選択されやすくなっていたと考えられる．しかし，図 3 から 30 世代ほど学習が止まり始めているため，30 世代以降の効果は薄いと考えられる．

色情報を使用する実験で得られた方策は，色情報を使用した関数は含まれていたが，色情報を使わない実験と比べて勝率は高くない．ここまでの実験結果からは色情報を使用することが有効であるかは確認できない．

8. まとめ

本研究ではガイスターにおいて GP を用いてプレイアウト方策を作成し，モンテカルロベースプレイヤーの性能向上を目指した．実験の結果，敵駒と隣接していたら「隣接している相手駒を取る」，そうでないなら「最も脱出口に近い駒を脱出口に向かわせる」など妥当な方策を含むもの得られた．また，通常のモンテカルロ木探索プレイヤーに対して個体数 100 のときは勝率 54%，個体数 300 のときは勝率 63%と勝ち越すことができた．よって，GP により有効なプレイアウト方策を作成できた．しかし，色情報を使用することの有効性は現在の段階では結論を出すことはできない．今後も引き続き実験していく．

今後は，完全情報状態の局面に対する Naotti-2020 の指手との一致率を適応度した場合に得られる方策との比較を行いたい．

参考文献

[1] 三塩武徳, 小谷善行. ゲームの不完全情報推定アルゴリズム upp とそのガイスターへの応用. 情報処理学会研究会報告, 第 2014-GI-31 巻, pp. 1–6, 2014.

[2] 鴛淵隆斗, 佐藤直之. ガイスターゲームにおけるモンテカルロ法を利用した駒推定およびブラフ手の生成可能性の検証. ゲーム情報学研究報告, No. 20, pp. 1–7, 2020.

[3] 川上直人, 橋本剛. ガイスター ai の研究. Technical Report 6, 情報処理学会, 情報処理学会, feb 2018.

[4] Atif M Alhejali and Simon M Lucas. Using genetic programming to evolve heuristics for a monte carlo tree search ms pac-man agent. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8. IEEE, 2013.

[5] Peter I Cowling, Edward J Powley, and Daniel Whitehouse. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 4, No. 2, pp. 120–143, 2012.

[6] 平野廣美. 遺伝的アルゴリズムと遺伝的プログラミング-オブジェクト指向フレームワークによる構成と応用. パーソナルメディア株式会社, 2000.

[7] 伊藤雅士, 大久保壮浩, 木谷裕紀, 小野廣隆. ガイスター ai のキーパー戦略の有効性. ゲーム情報学研究報告, No. 3, pp. 1–7, 2019.

[8] Omid E David, H Jaap van den Herik, Moshe Koppel, and Nathan S Netanyahu. Genetic algorithms for evolving computer chess programs. *IEEE transactions on evolutionary computation*, Vol. 18, No. 5, pp. 779–789, 2013.