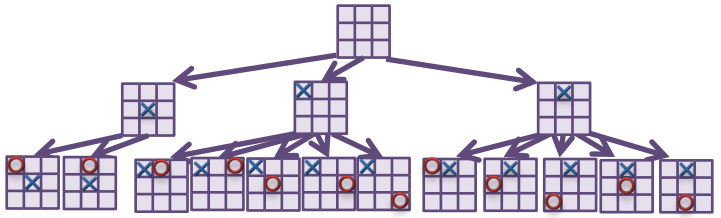
探索と推定：
並列モンテカルロ木探索

美添 一樹 Kazuki Yoshizoe
探索と並列計算ユニット、ユニットリーダー
理化学研究所 革新知能統合研究センター （理研AIP）

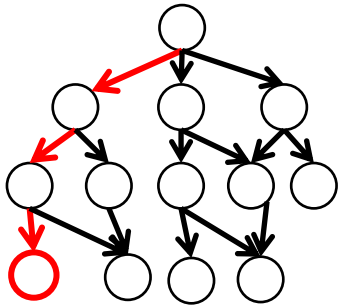
2019年9月22日 行動モデル夏の学校2019

グラフ探索



今日のテーマは
グラフ探索

グラフから
特定の節点又は経路
（の集合）
を発見すること

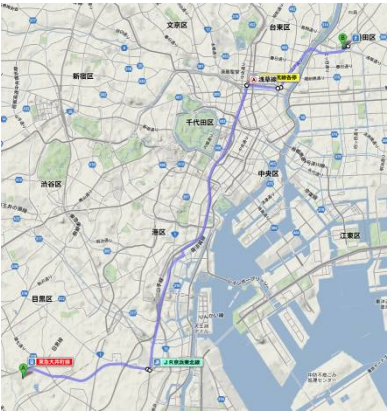


グラフが実際に与えられる例

路線図
道路地図
social network

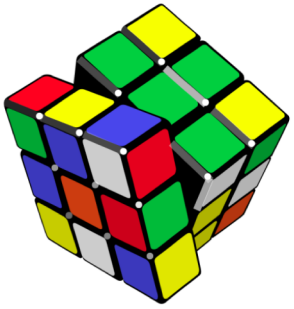
ルールで生成される例

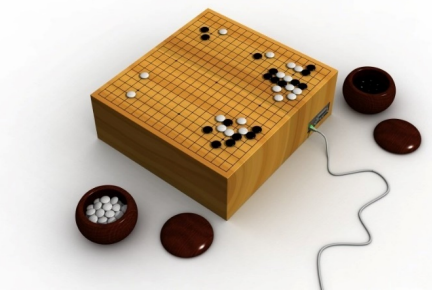
組合せ最適化
SAT
ゲーム パズル
制約充足問題



これが
「最短経路」
「最善の組合せ」
「最善手」
などに対応

AlphaGo = DeepLearning + 探索

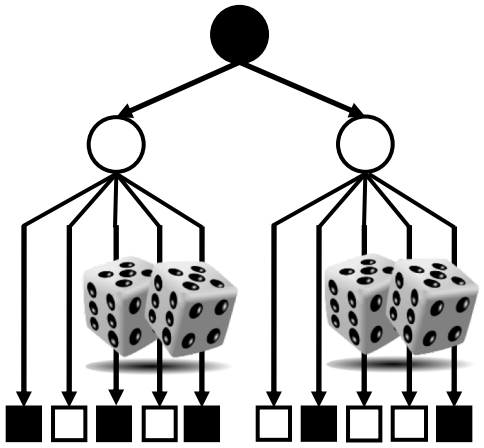




人間のチャンピオンを倒した
Google DeepMind 製の囲碁プログラム

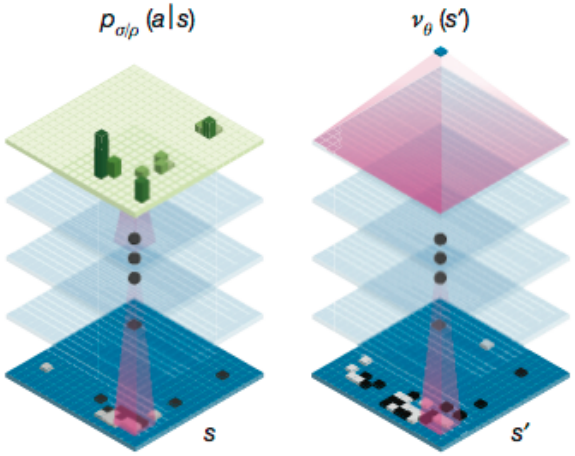
AlphaGo を支えるの3つの技術

モンテカルロ木探索
評価関数なしで探索
(2006年～)



[Coulom 2006]

Deep Learning (深層学習)
評価関数を作る
(2006年～)



[Silver, Huang et al. 2016] Fig. 1b

強化学習
「報酬」を元に学習
(試みは古くからあり)

DQN

https://deepmind.com/research/dqn/

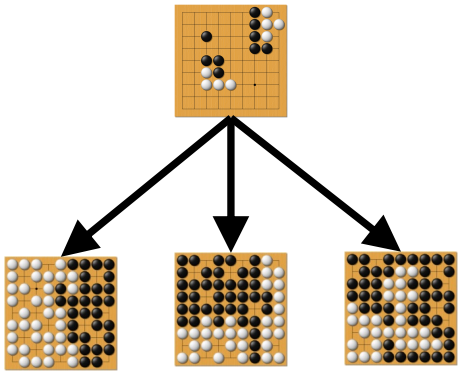
Arcade Learning Environment

https://github.com/mgbellemare/Arcade-Learning-Environment
https://www.youtube.com/watch?v=nzUiEkasXZI

[Coulom 2006] CrazyStone

Monte Carlo Tree Search
モンテカルロ木探索

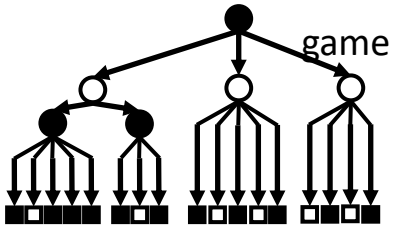
節点をサンプリングで評価
(決定的な評価関数は使わない)



UCB1 algorithm

$$\frac{w}{s} + \sqrt{\frac{2\ln t}{s}}$$

確率的に「有望」な枝を選択



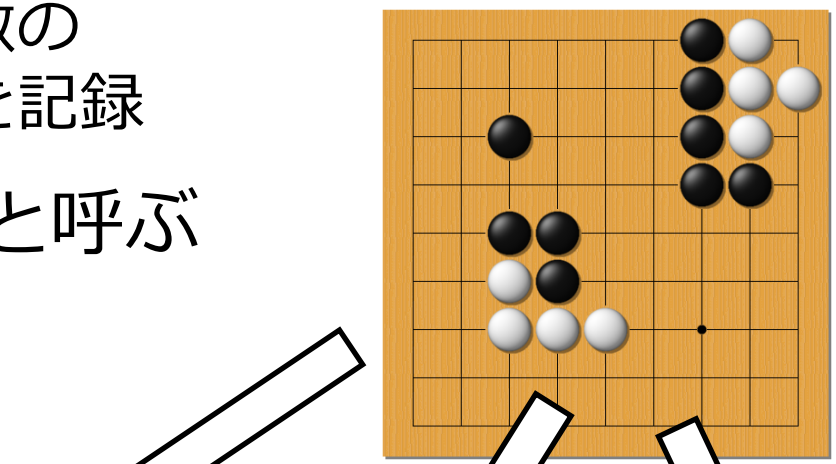
これを繰り返して
グラフ探索を行う

乱数を使って囲碁を打つ

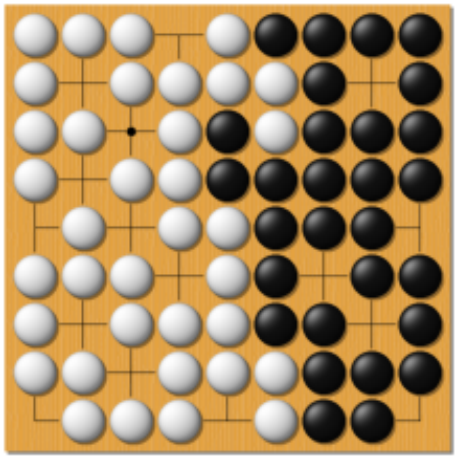
乱数を使って、多数の
終局図を作り、勝敗を記録

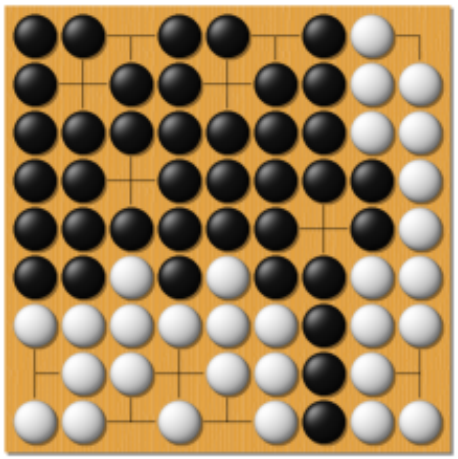
これを**ロールアウト**と呼ぶ

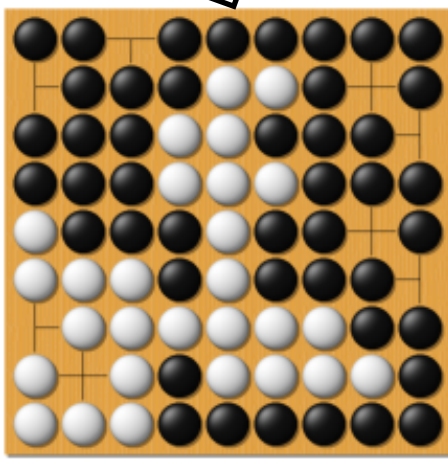
便宜上この方法を
原始モンテカルロ囲碁
と呼ぶことにする



黒
2勝1敗



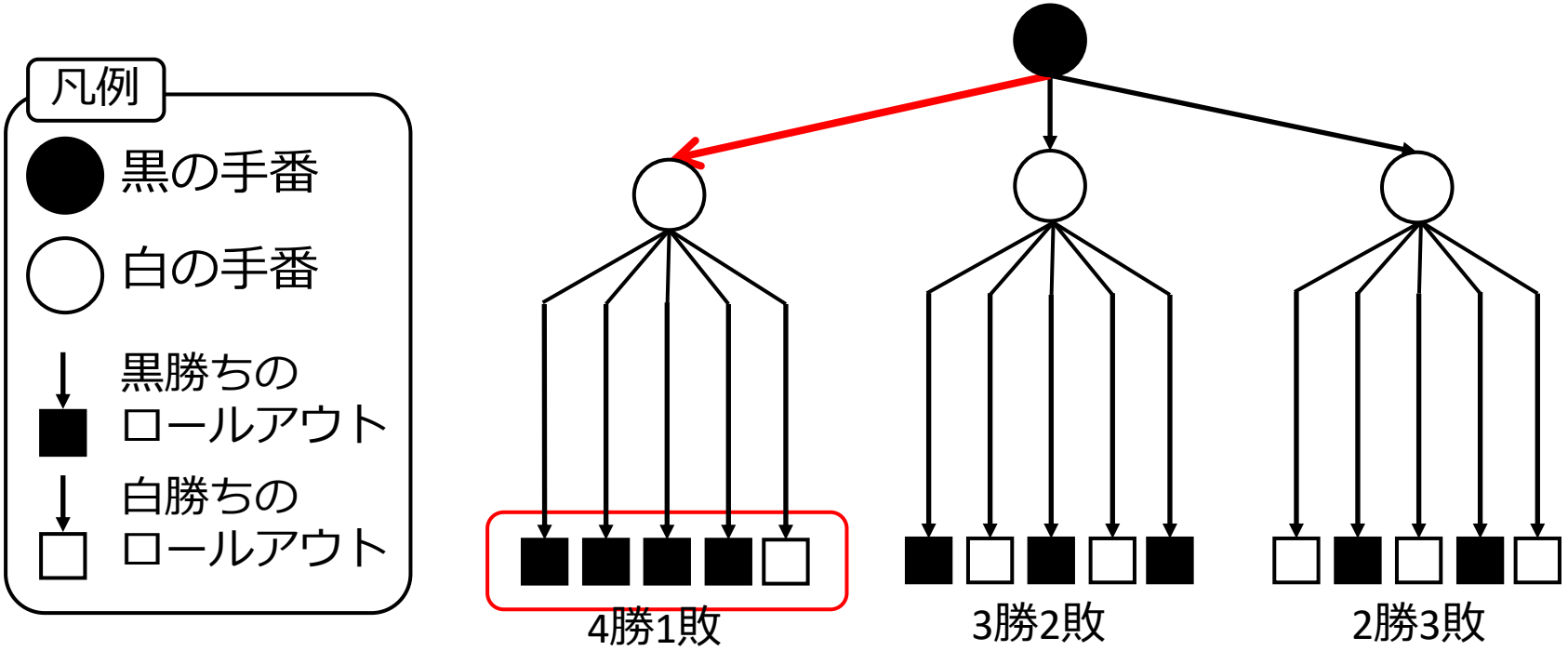




注：**眼**を残す

[Brügmann 1993]

原始モンテカルロ囲碁



一番勝てそうな手を選ぶ

注：この方法は弱い
　　無限の時間を掛けても最善手は分からない

注：こんな方法が研究された理由は囲碁の
　　評価関数が作れなかったから
　　（今は深層学習で作れる。今日は省略）

原始モンテカルロ囲碁を改良

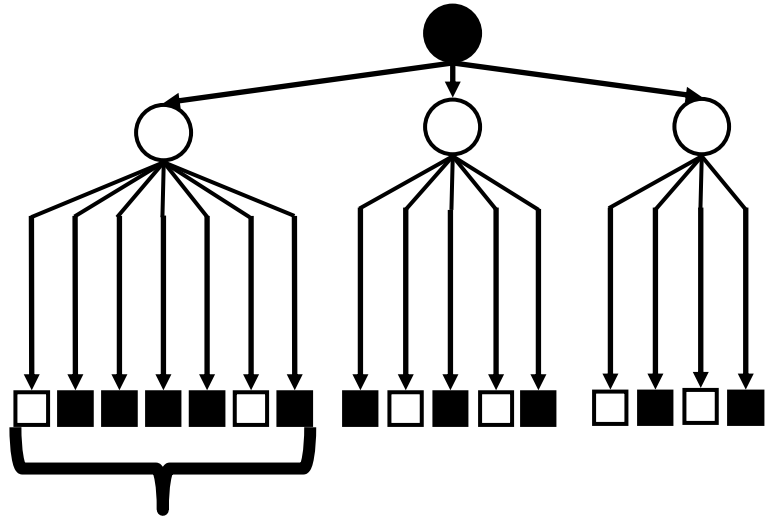
原始モンテカルロ囲碁は
そのままだと**弱い**

相手のミスに期待した手を打つ
飛車の頭に歩を打つような手

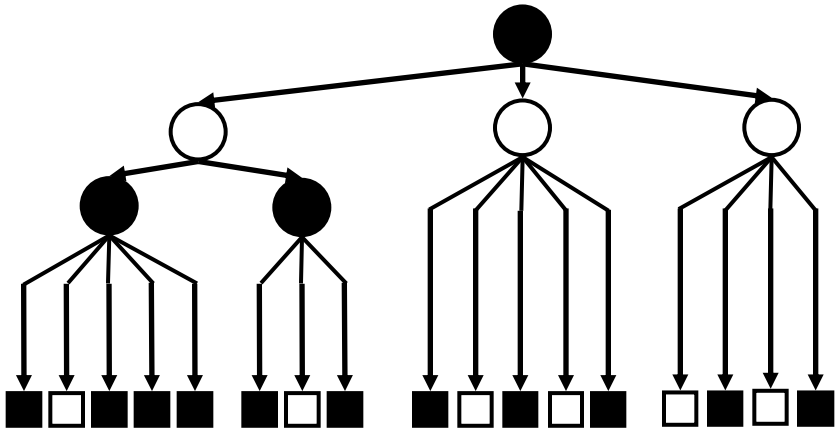
そこで、
良さそうな所を
深く読むように改良

これが
モンテカルロ木探索

プログラム CrazyStone [Coulom 2006]

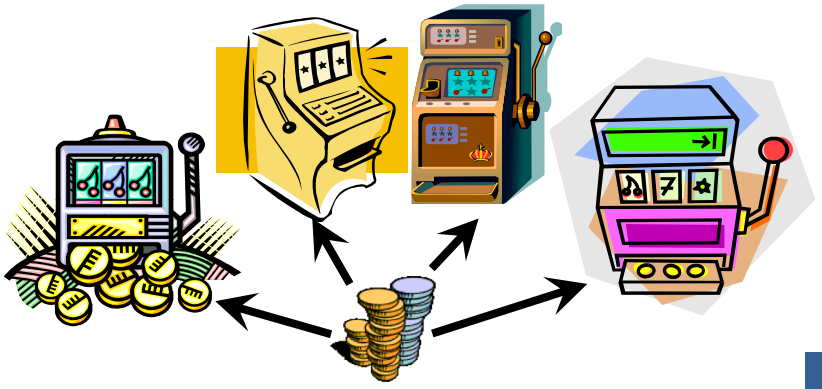


ロールアウト数が閾値以上なら展開



[Robbins 1952][Lai and Robbins 1984]

Multi-Armed Bandit Problem



[Auer, Cesa-Bianchi, Fischer 2002]

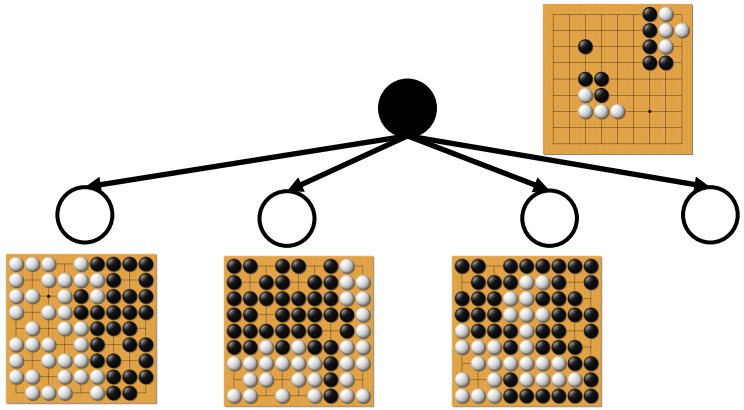
UCB1 アルゴリズム

cumulative regret を最小化する

$$\frac{w}{s} + \sqrt{\frac{2\ln t}{s}}$$

mean項 + bias項

w : その腕の報酬
s : 投入したコイン
t : s の合計



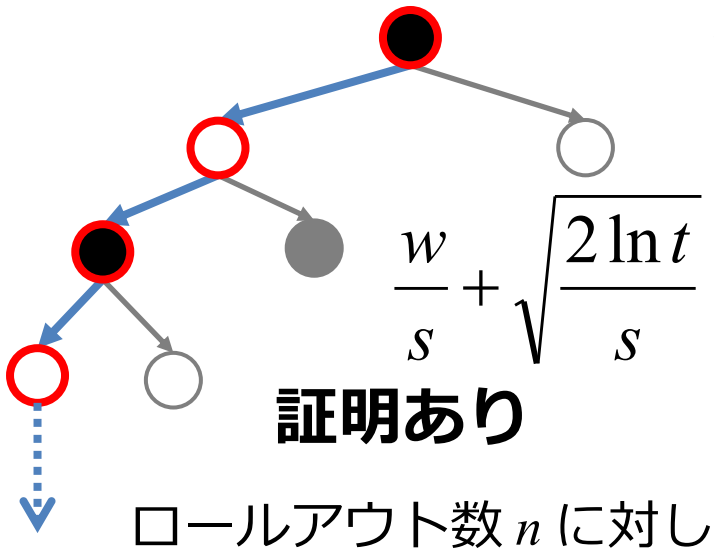
MAB と 原始モンテカルロ囲碁

腕	⇔	合法手
コイン	⇔	ロールアウト
コインの枚数	⇔	考慮時間
regret 最小化が目的	?	最善手が目的

[Kocsis, Szepesvári 2006]

UCT algorithm

Upper Confidence bound applied to Trees



$$\frac{w}{s}+\sqrt{\frac{2\ln t}{s}}$$

証明あり

ロールアウト数 n に対し、

$$\frac{w}{s}+O\left(\frac{\log n}{n}\right)$$

また、**1手目**が間違える確率は 0 に収束.

つまり(いずれは)最善手が分かる

節点の比較に**UCB1**を用いる

木探索アルゴリズム

値が**最大の枝**をたどる

（末端の節点を展開）

末端で**ロールアウト**

経路上の節点を更新

繰り返し

UCT 擬似コード

```
UCT() {
  while(time_remaining) {
    iter()
  }
}
iter() {
  node = root
  while (true) {
    node = SelectBestChild(node)
    if(node is leaf) break
  }
  if(visit_count(node) > expand_threshold) {
    expand(node)
    node =SelectBestChild(node)
  }
  reward = rollout(node)
  backup(reward)
}
```

実際は以下も必要
- 経路の記録
- サイクルの回避
- 値のbackup

UCB1, UCT: 証明関連の補足

報酬が $[0, 1]$ なら $c=1$ で証明成立

[Auer, Cesa-Bianchi and fischer 2002]

UCB1の証明

Chernoff bound を用いる

[Kocsis, Szepesvári 2006]

UCTの証明

各部分木を確率過程とみなす
Hoeffding-Azuma 不等式を用いる

値域が $[0, 1]$ より広がると
(c=1のままでは) 証明は不成立

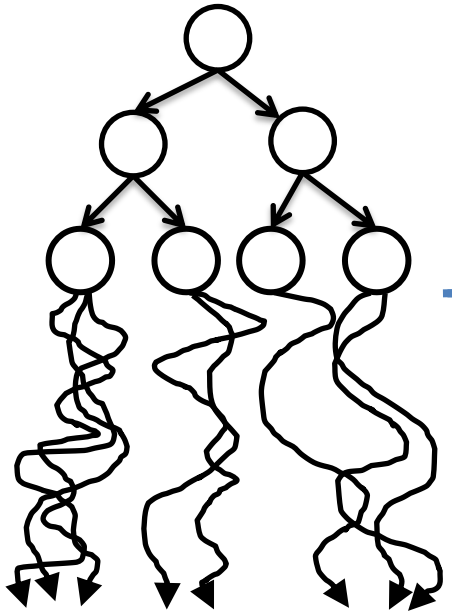
1回プレイアウトするごとに
更新しないと証明は不成立

exploration constant

$$\frac{w}{s} + c\sqrt{\frac{2\ln t}{s}}$$

実践的には問題に合わせて
c を調整する

ほとんどの場合、
最善の c はかなり小さい



木探索
ほぼ共通

ロールアウト
問題ごとに
取り替える

MCTS の汎用性

評価関数を
・作れない
・**作るのがめんどくさい**
等の時にも効率よく探索

もちろん、何でもできる
万能のツール**ではない**

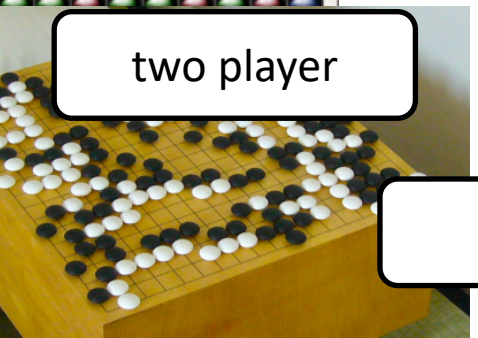
MCTS の
得意、不得意は？

枠組みは
汎用性が高い

single player

multiplayer

two player



scheduling

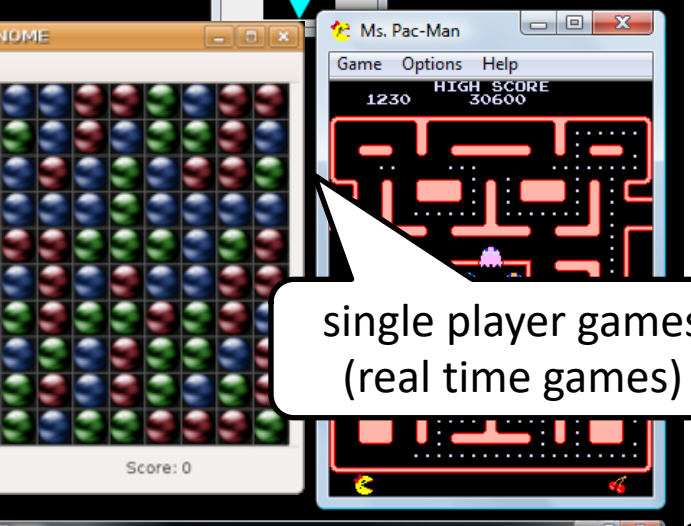
planning

biometric security

例えば、
囲碁は得意
将棋は苦手

MCTS の応用

囲碁の研究が様々な応用に貢献



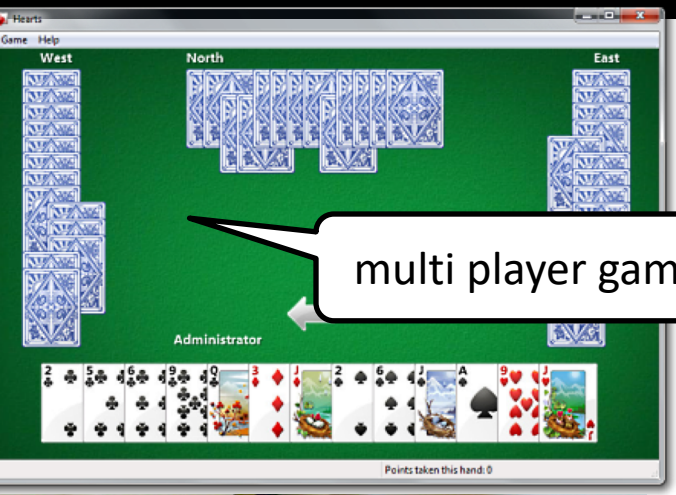
single player games
(real time games)

[Yang, Zhang, Yoshizoe, Terayama, Tsuda 2017]
“ChemTS: an efficient python library for de novo molecular generation”.

新規化合物発見

Canadian Traveler Problem

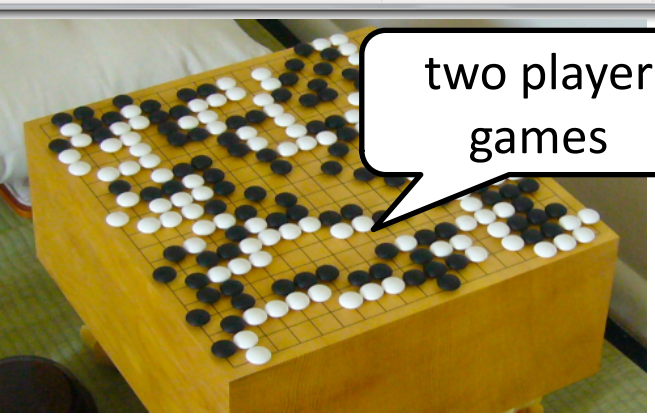
[Z. Bnaya, A. Felner, D. Fried, O. Maksin and S. E. Shimony]
“Repeated-Task Canadian Traveler Problem”, SoCS 2011



multi player games

biometric security

[Tanabe, Yoshizoe and Imai 2009]
“A study on security evaluation methodology for image-based biometrics authentication systems”



two player games

[Cazenave, Balbo, Pinson 2009]
“Monte-Carlo bus regulation”

自然言語処理

scheduling

[Chevelu, Putois, Lepage 2010]
“The true score of statistical paraphrase generation”

ロールアウトが妥当か

囲碁

候補手が減っていくので
自然と終局する
（簡単な制約は必要）

将棋

ランダムなプレイで
自然な終局図を
迎えることは困難

ロールアウトで「**自然な**」報酬が
得られるかどうかが問題

何が「自然」かつきつめると難しいが…

注意：MCTS 将棋は
初段よりは強い

初段あるのに苦手とは生意気な…

MCTSの弱点

確率的な探索なので

細い一本道は苦手

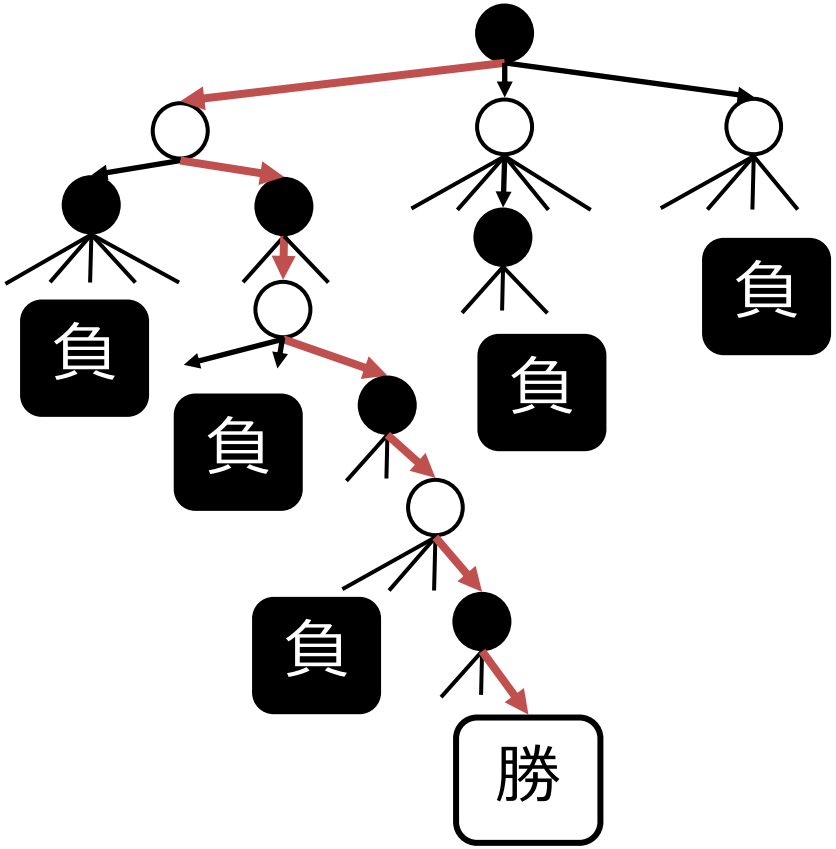
理論的解析の例 [Coquelin, Munos 2007]
人工的に苦手な木を作れば
ほとんどいくらでも収束を遅くできる

実践的解析の例 [Ramanujan, Selman 2011]
同じゲームでも序盤は得意
中盤は苦手という例

囲碁は得意

将棋は苦手

苦手な形の木
(trap, だまし構造)



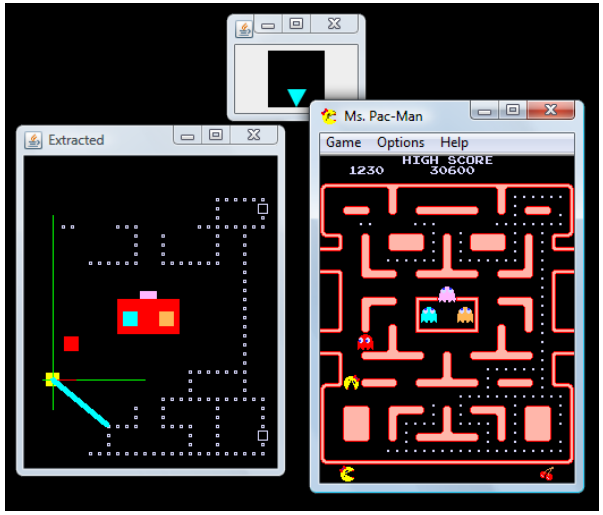
モンテカルロ木探索の Anytime 性
リアルタイムゲームへの適用

Ms. Pac-Man AI Competition

MCTS を用いたプログラムが
世界記録を更新 [Ikehata, Itoh 2011]

挟み撃ちをさけるように作られた
ロールアウトを使って MCTS

Ms. Pac-Man　1/15秒ごとに行動



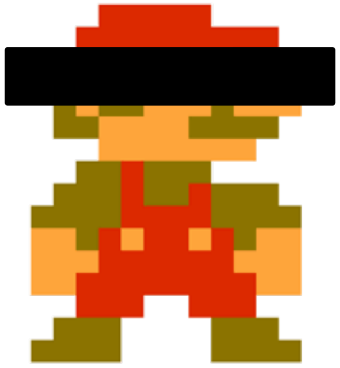
http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html

マ○○AI Competition

某有名アクションゲーム
のクローンを使ったAI大会

「人間のプレイに似せる部門」
(Turing Track)
A*探索と MCTS を比較 （動画）

マ○○

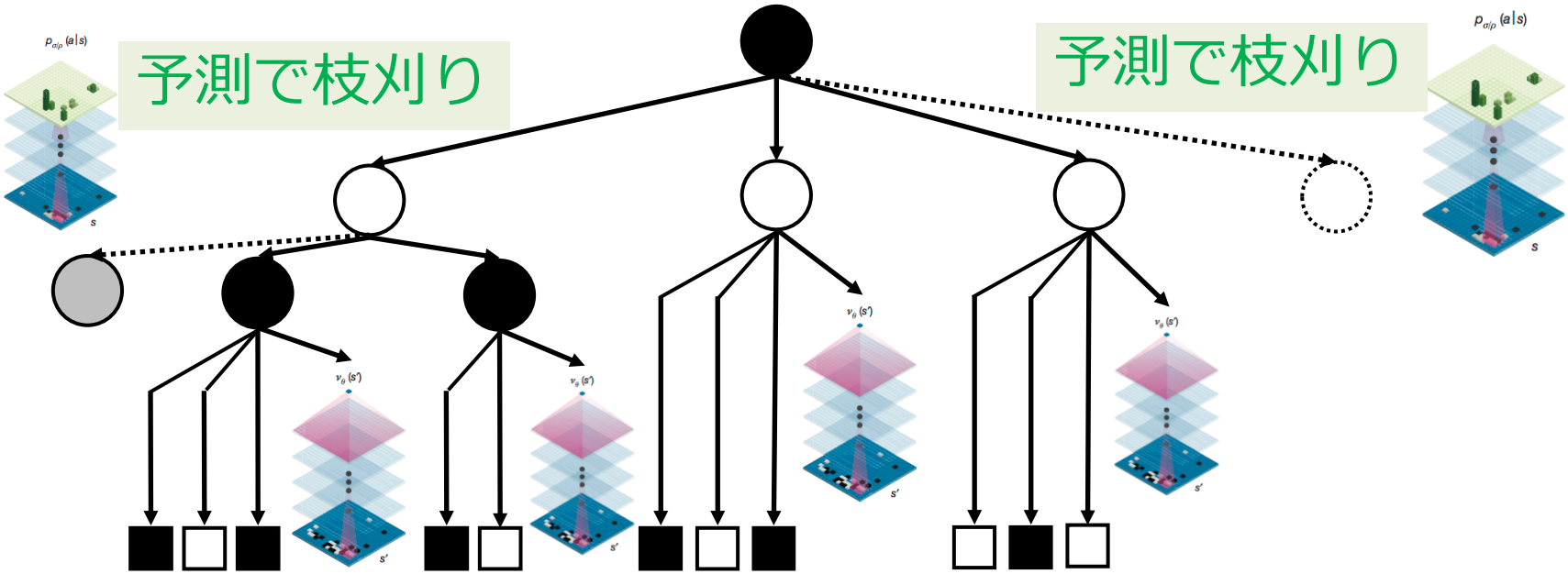


プライバシに
配慮しています

1/24秒ごとに行動

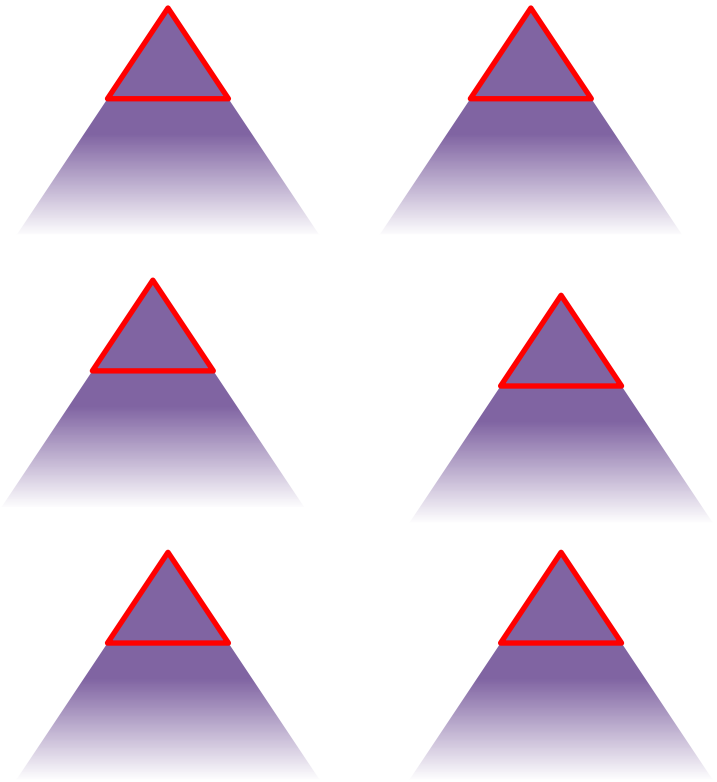
http://www.platformersai.com/

Deep Neural Network との組合せ



ロールアウトの結果を、ニューラルネットが出力する値（≒勝率）に置き換え

MCTS の単純な並列化: Root 並列化



異なる乱数シードで
複数の探索を行う

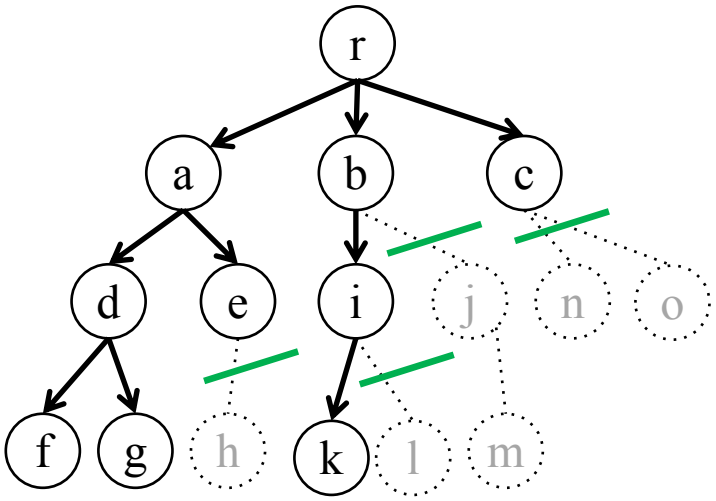
深さ 1〜3 程度の
部分を共有する

少し工夫すると20並列
くらいまでは有効

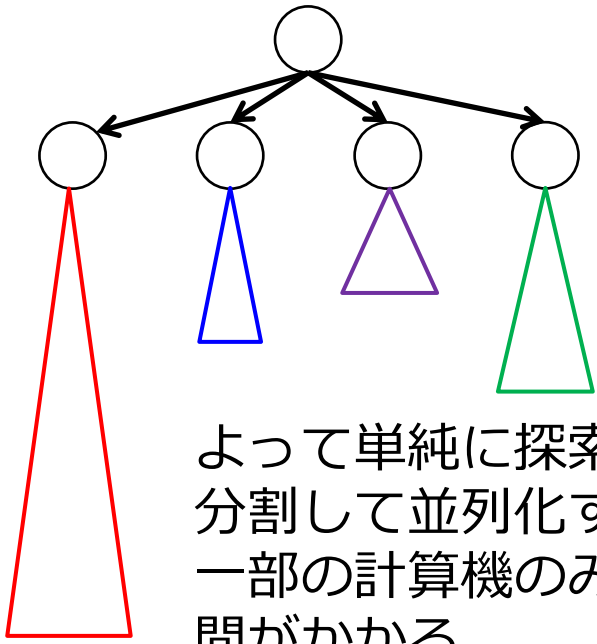
探索の並列化は難しいので
これでもそれなりに良い

参考: SAT solver の portfolio 並列化

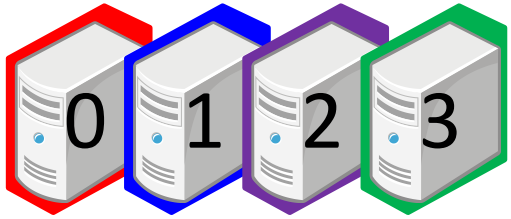
（真面目な）並列探索の難しさ



実用的な探索アルゴリズムは探索空間を枝刈りして重要な部分に集中する

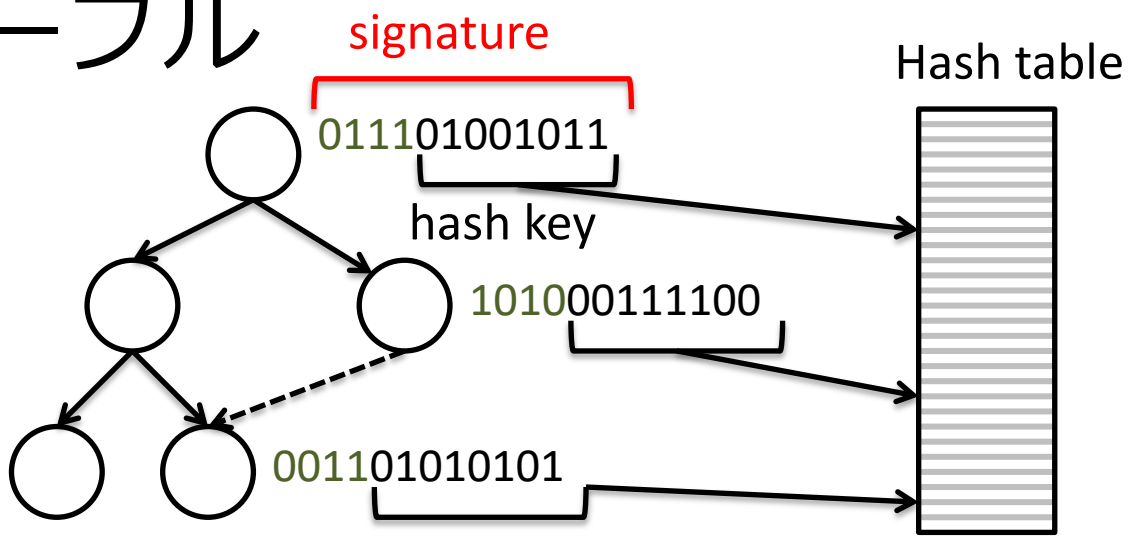


よって単純に探索空間を分割して並列化すると一部の計算機のみ計算時間がかかる



均等な負荷分散の実現がまず難しい

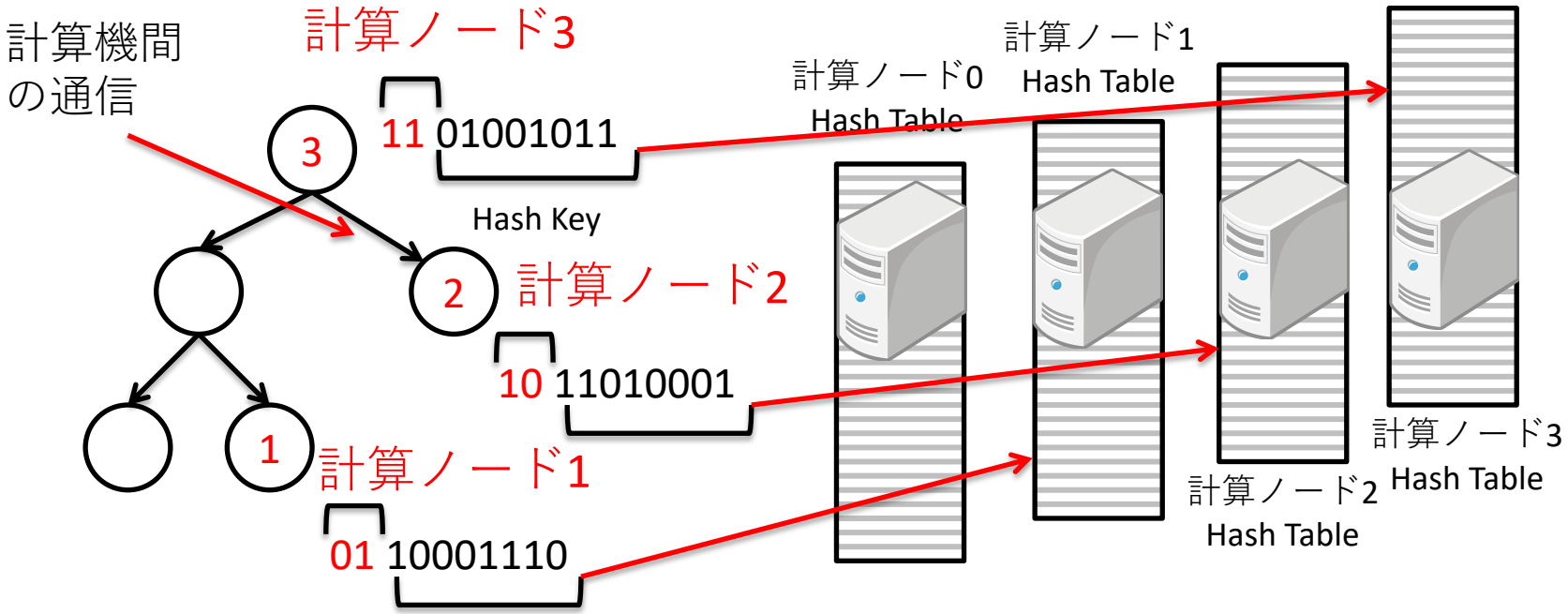
ハッシュテーブルと探索



- 探索アルゴリズムはしばしばハッシュ表を使う
  - Transposition Table : TT 遷移表 と呼ぶことも
  - 当然、節点の signature を計算するハッシュ関数が必要
- 主に二つの目的
  - 探索の履歴を用いた高速化手法
    - 反復深化, History Heuristics, など
  - 合流を検知し、冗長な計算を省略

[Evett et al. 1995] PRA*　[Romein et al. 1999] TDS

Hash driven parallel search



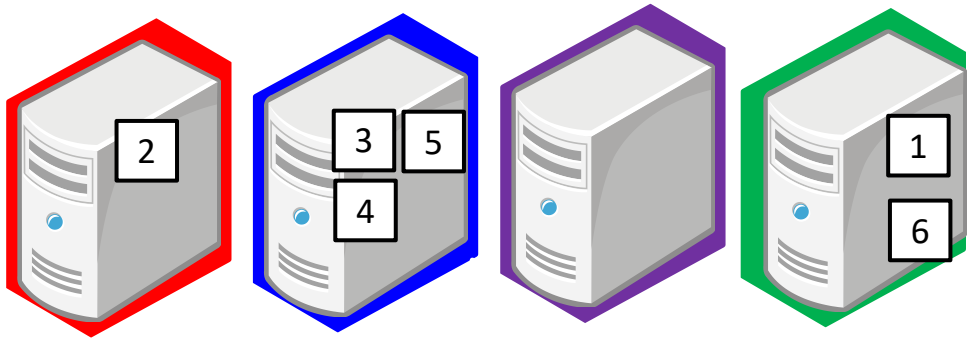
この手法 (Hash table driven parallel search) の特徴

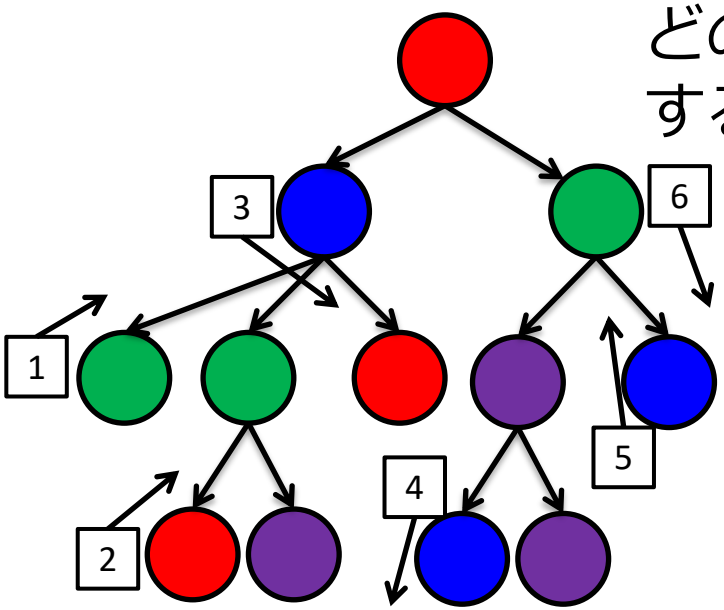
- **均等な負荷分散が自然と実現**される
- しかし1対1の通信が多発する

} トレードオフ

- 通信オーバーヘッドを軽減できるかどうかが重要
  うまく通信と計算をオーバーラップできると良い

ハッシュ駆動 並列探索の動作





1 ジョブおよびジョブ番号

どの節点をどの計算ノードが担当するかはハッシュ関数に依存する

補足：根節点を担当する計算ノードもさらに深い節点を担当している

データは計算ノードに固定されている。「ジョブ」が移動する。

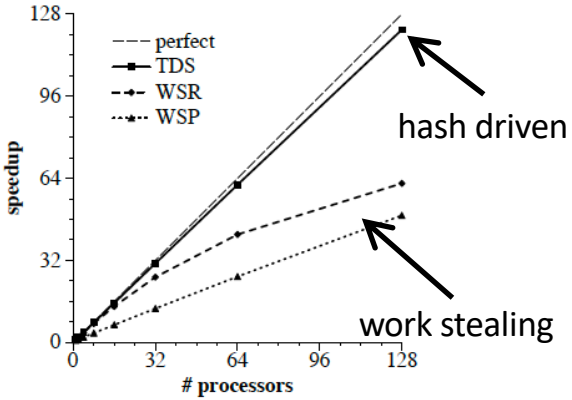
当然、ジョブの総数は計算ノードより多くないといけない

実験では計算ノード数 x 3～20 個のジョブを生成している

注意：節点も計算機もどちらも「ノード」なので非常にややこしい

Hash driven Parallel Search の並列化による速度向上

TDS algorithm
(Parallel IDA*)



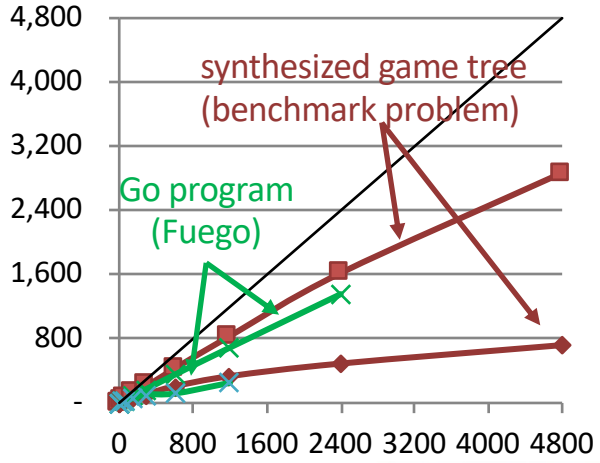
[Romein+ 1999] Fig. 4 (c)

HDA*
(Parallel A*)

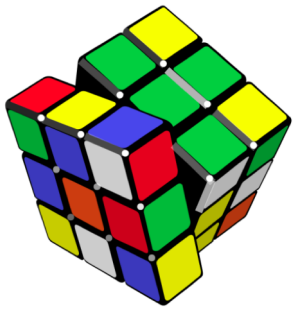
パズル、
プランニング問題、
sequence alignment
などに応用
最高で 2400コア使用
1,000倍以上高速化

[Kishimoto+ 2012]

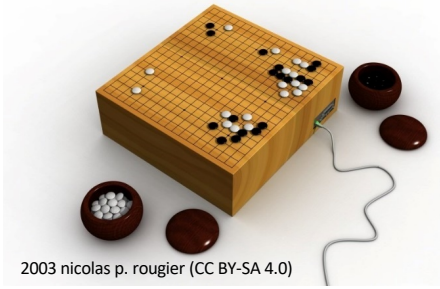
TDS-df-UCT algorithm
(Parallel MCTS)



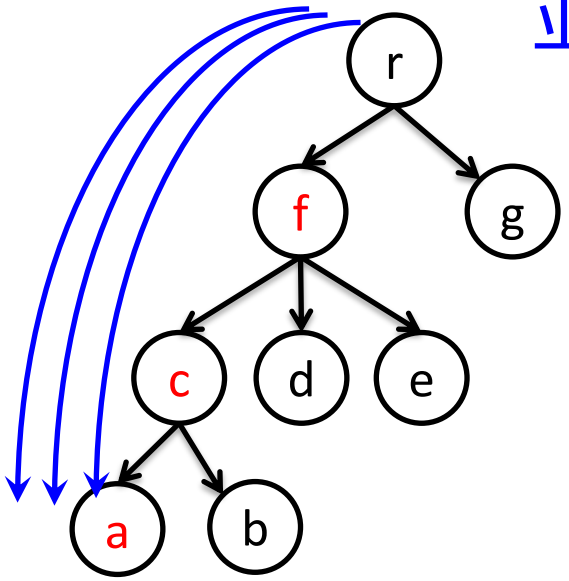
[Yoshizoe+ 2011]



注意：この性能は、アルゴリズムを変形してオーバーヘッドを取り除くことによって達成されている
（これから説明する）



virtual loss によるジョブの分散



並列化 と Virtual Loss

UCTの場合、何も工夫しないと全ての計算機が同じ所を探索する

そこで、探索中の節点には Virtual Loss というペナルティーを加える
（今探索中の節点の報酬は最低だと仮定するので virtual loss という）

d は節点を探索中のプロセス数
D は兄弟節点の d の合計

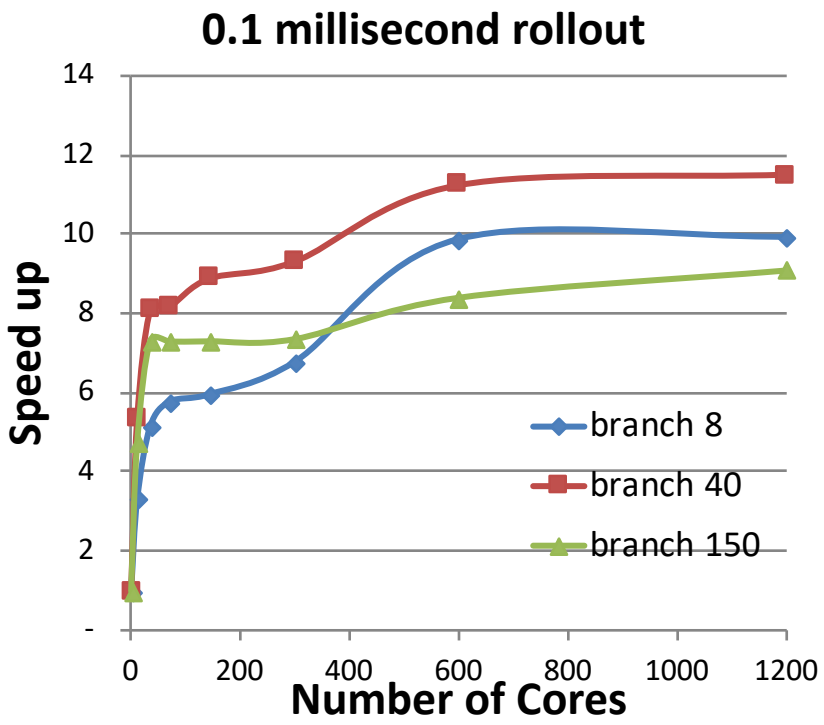
$$\frac{w}{s}+\sqrt{\frac{2\ln t}{s}} \quad \Rightarrow \quad \frac{w}{s+d}+\sqrt{\frac{2\ln(t+D)}{s+d}}$$

[発明者不明 2007ごろ?]

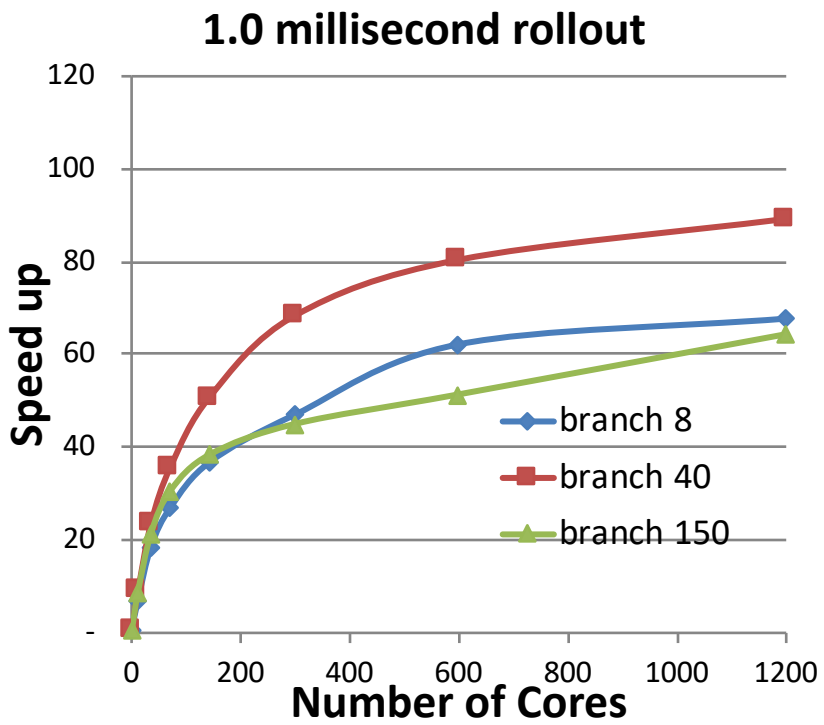
virtual loss + 並列UCT の性能

まず人工的な探索木 (P-game) で3通りの分岐数で実験

ロールアウトが高速で
性能が出にくい場合



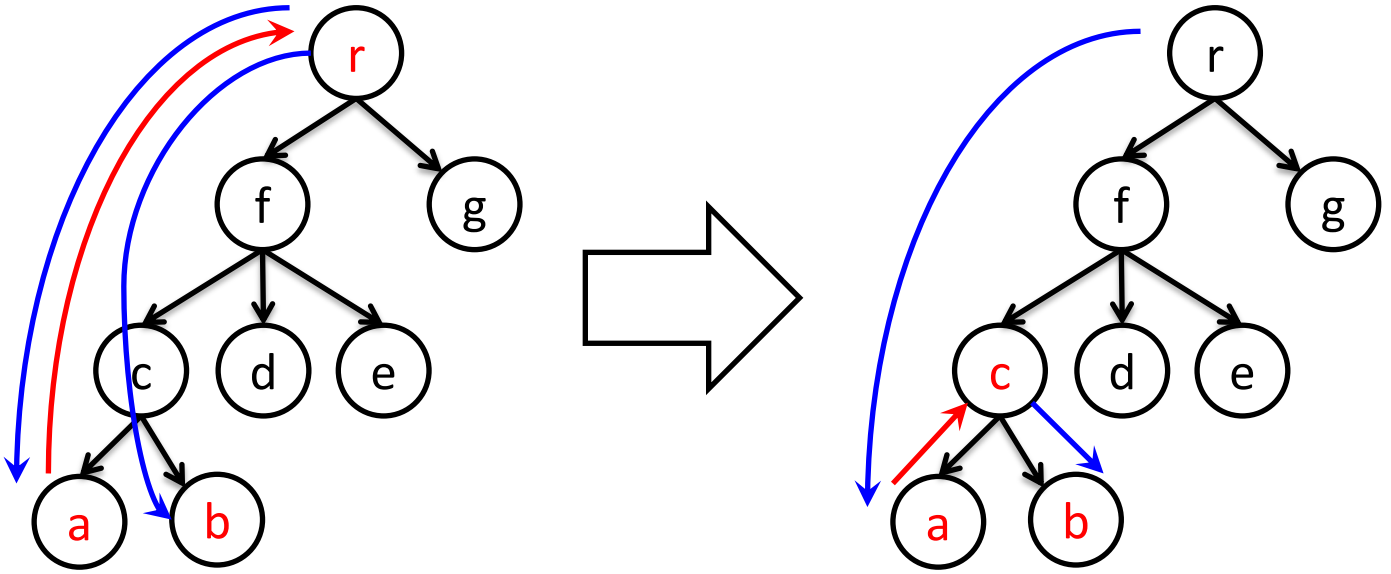
ロールアウトが遅く
性能が出やすい場合



Virtual Loss による並列UCTは性能が悪い

ボトルネックは通信の集中

深さ優先変形による通信の集中の回避



(vanilla) UCT

不必要に毎回 root 節
点まで戻っている

Depth First UCT

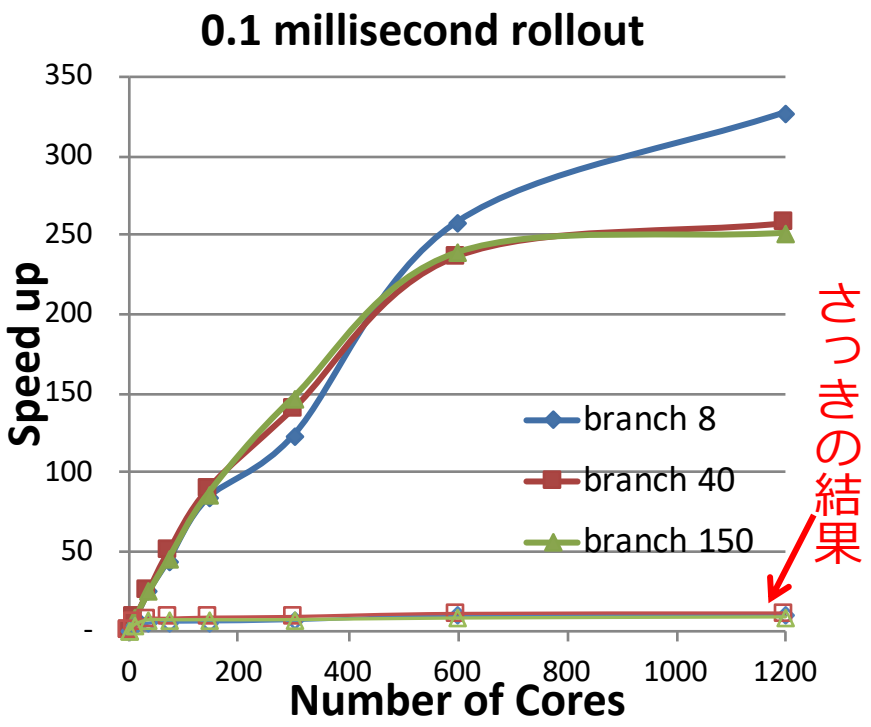
不必要なリターンを削除

[吉本, 岸本, 金子, 美添 2007]

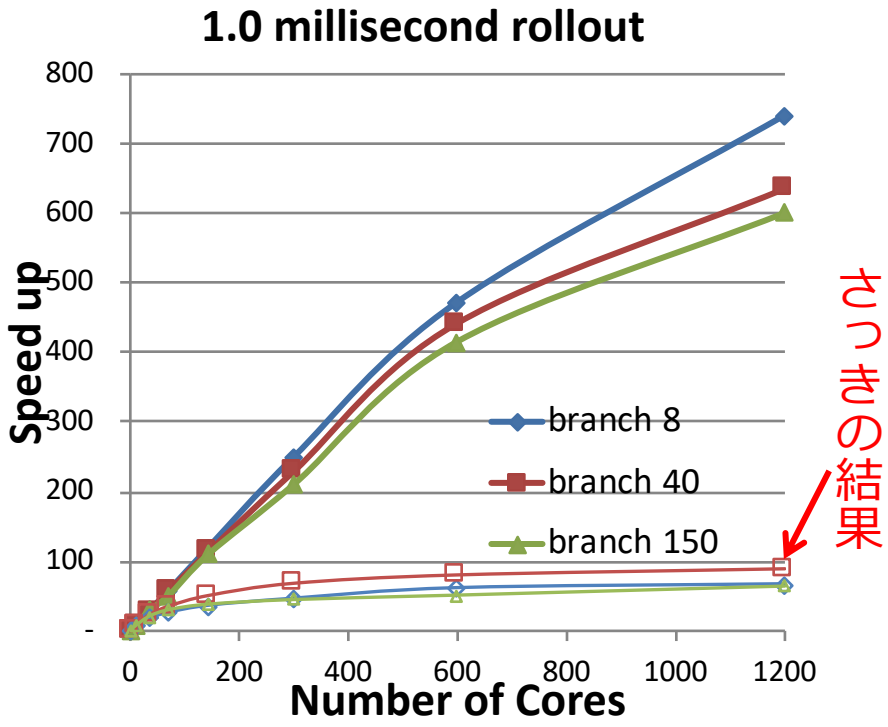
局所性を高めるのは並列化の基本

通信集中を回避した場合

ロールアウトが高速で
性能が出にくい場合



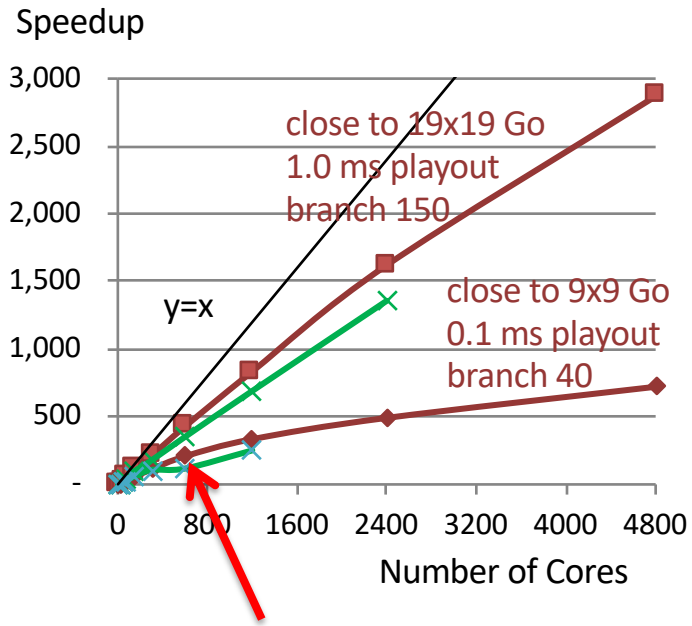
ロールアウトが遅く
性能が出やすい場合

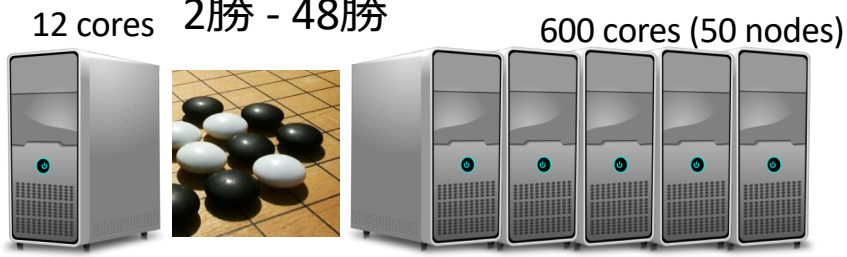


深さ優先変形によって
見た目の性能は向上している

TDS + depth first UCT = TDS-df-UCT [Yoshizoe, Kishimoto, Kaneko, Yoshimoto, Ishikawa 2011]

並列MCTS: 囲碁での性能





Hardware: TSUBAME2
CPU: Xeon 2.93GHz x 2 (12 cores)
Network: Infiniband QDR x 2
Library: MVAPICH2 library

- P-Game (人工的なゲーム木)
  - 二人ゲームの性質を模倣する木
- 並列囲碁プログラム
  - MP-Fuego (Fuegoの並列版)
    序盤戦と終盤戦
    （終盤の方が rollout が早く終わる）
- 速度向上に影響する要素
  - rollout にかかる時間
  - 通信速度
  - その他パラメータ

参考：
AlphaGo は単純な並列化を採用
詳細は不明な点が多い

40CPU+8GPU 対 1202CPU+176GPU

大規模版が勝率75%（効率悪い）

MCTS まとめ

- MCTS はサンプリング (rollout, playout) によって節点を評価する
  - 決定的な評価関数を用いない
- 特に重要な UCT アルゴリズムは Multi-Armed Bandit を理論的背景とする
  - 最善解への収束が保証されている
- 得意不得意がある
  - 細く長い正解手順を見逃しやすい (例, AlphaGoの敗局)
- 機械学習手法と相性が良い
  - 節点の評価が確率で与えられた場合に自然に対応
- 並列化
  - Hash driven parallel search などで並列化可能
  - ただし実装は難しい
  - 容易に利用可能な並列MCTSライブラリの実現が待たれる