Monte Carlo Search

Tristan Cazenave

LAMSADE CNRS

Université Paris-Dauphine

PSL

Tristan.Cazenave@dauphine.fr

Outline

- Monte Carlo Tree Search

- Nested Monte Carlo Search

- Nested Rollout Policy Adaptation

- Playout Policy Adaptation

- Zero Learning (Deep RL)

- Imperfect Information Games

Intelligence artificielle

Une approche ludique

Tristan Cazenave

ellipses

Monte Carlo Tree Search

Monte Carlo Tree Search

Monte Carlo Tree Search, a method so bold,
Exploring the tree, with its stories untold,
Simulating the future, with random rolls,
To find the path with the greatest goals.

It starts with a root, and expands the tree,
Each node a choice, a potential key,
To unlock the door, to the desired goal,
Monte Carlo guides, in finding the whole.

It's not always perfect, with a fixed plan,
But it adapts, to the situation at hand,
With each simulation, it learns and it grows,
To find the best move, with more and more those.

Monte Carlo Tree Search, a powerful tool,
To navigate the game, and be the cool,
So let's explore the tree, and see what we find,
And use Monte Carlo, to be one of a kind.

Monte Carlo Tree Search

Monte Carlo Tree Search, a method so grand,
A path to victory, with a strategic hand.

It starts with a tree, and a game to play,
Simulations to run, to find the best way.

Each node a move, and each branch a choice,
We explore them all, with a clear and steady voice.

With randomness guiding us, and statistics to aid,
We search for the path, that will surely lead the way.

Monte Carlo Tree Search, a brilliant mind,
With its power and wisdom, it will never be confined.

It's a strategy that's proven, and a method that's sound,
Monte Carlo Tree Search, will always be found.

In the game of life, and the game of chance,
Monte Carlo Tree Search, will always enhance,
Our ability to win, to be victorious,
It's a path to success, so mysterious.

Monte Carlo Go

- 1993 : first Monte Carlo Go program
  - Gobble, Bernd Bruegmann.
  - How nature would play Go ?
  - Simulated annealing on two lists of moves.
  - Statistics on moves.
  - Only one rule : do not fill eyes.
  - Result = average program for 9x9 Go.
  - Advantage : much more simple than alternative approaches.

Monte Carlo Go

- 1998 : first master course on Monte Carlo Go.

- 2000 : sampling based algorithm instead of simulated annealing.

- 2001 : Computer Go an AI Oriented Survey.

- 2002 : Bernard Helmstetter.

- 2003 : Bernard Helmstetter, Bruno Bouzy, Developments on Monte Carlo Go.

Monte Carlo Phantom Go

- Phantom Go is Go when you cannot see the opponent's moves.
- A referee tells you illegal moves.
- 2005 : Monte Carlo Phantom Go program.
- Many Gold medals at computer Olympiad since then using flat Monte Carlo.
- 2011 : Exhibition against human players at European Go Congress.

UCT

- UCT : Exploration/Exploitation dilemma for trees [Kocsis and Szepesvari 2006].
- Play random random games (playouts).
- Exploitation : choose the move that maximizes the mean of the playouts starting with the move.
- Exploration : add a regret term (UCB).

UCT

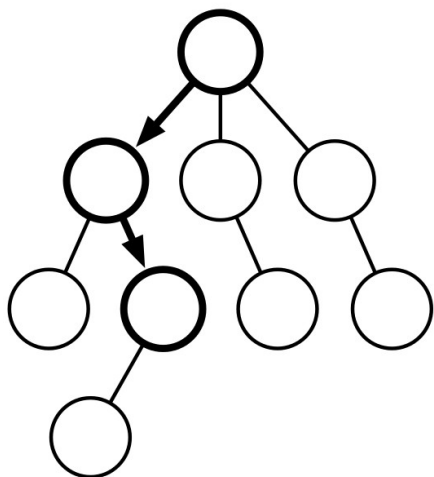- UCT : exploration/exploitation dilemma.
- Play the move that maximizes

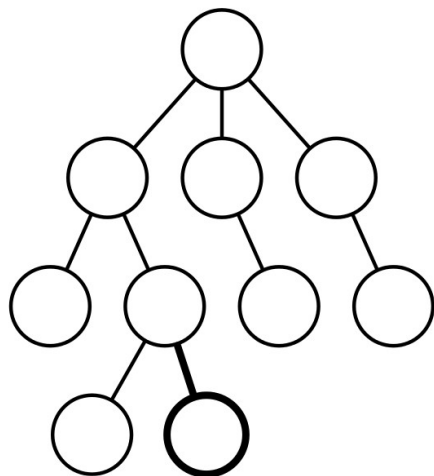$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

In which

- w_i = number of wins after the i-th move
- n_i = number of simulations after the i-th move
- c = exploration parameter (theoretically equal to $\sqrt{2}$)
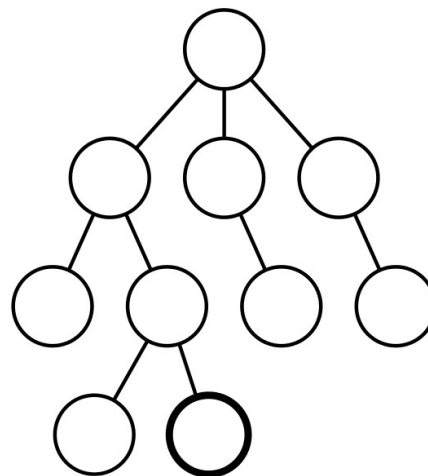- t = total number of simulations for the parent node
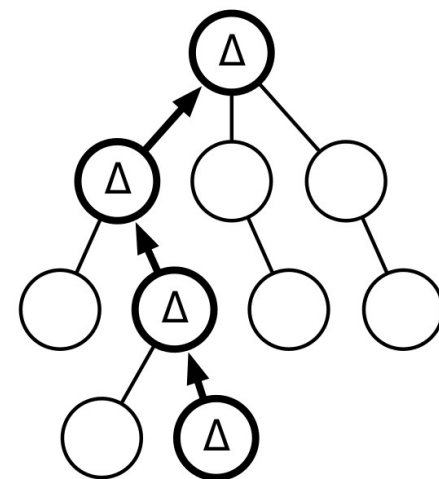
UCT

Selection

Expansion

Sampling

Backpropagation

Tree Policy

Default Policy

RAVE

- A big improvement for Go, Hex and other games is Rapid Action Value Estimation (RAVE) [Gelly and Silver 2007].

- RAVE combines the mean of the playouts that start with the move and the mean of the playouts that contain the move (AMAF).

RAVE

- Parameter β_m for move m is :

  $\beta_m \leftarrow pAMAF_m / (pAMAF_m + p_m + bias \times pAMAF_m \times p_m)$

- β_m starts at 1 when no playouts and decreases as more playouts are played.

- Selection of moves in the tree :

  $argmax_m((1.0 - \beta_m) \times mean_m + \beta_m \times AMAF_m)$

GRAVE

- Generalized Rapid Action Value Estimation (GRAVE) is a simple modification of RAVE.

- It consists in using the first ancestor node with more than n playouts to compute the RAVE values.

- It is a big improvement over RAVE for Go, Atarigo, Knightthrough and Domineering [Cazenave 2015].

Atarigo
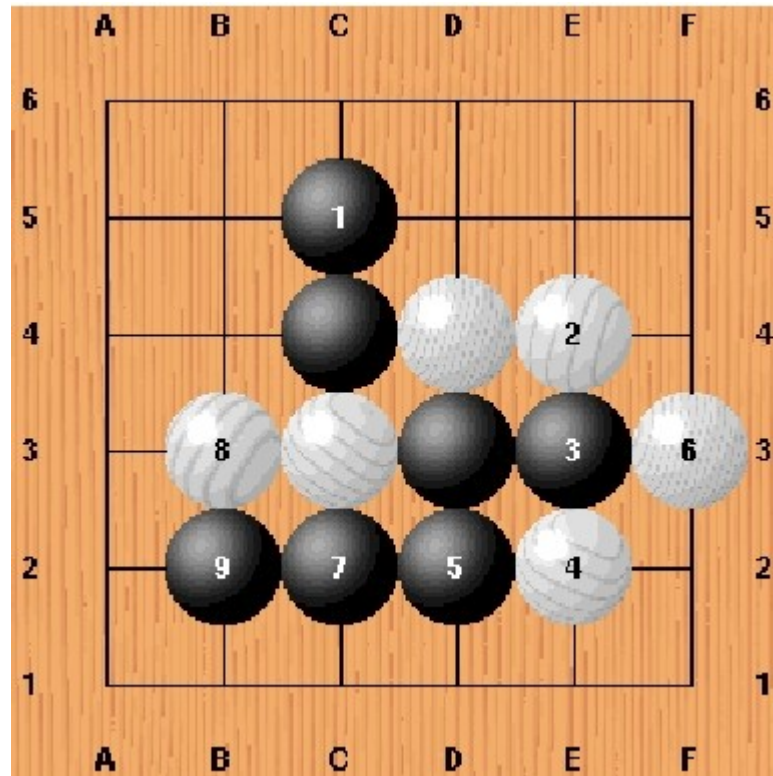
Knightthrough
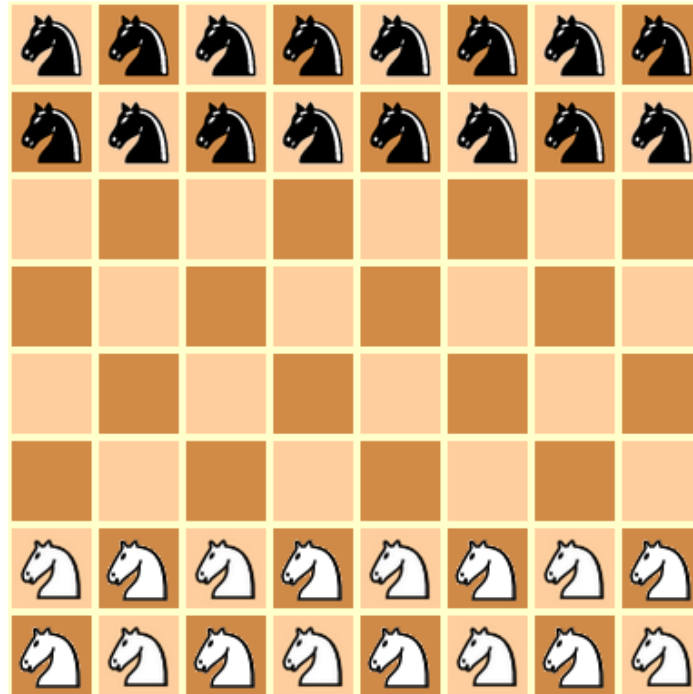
Domineering
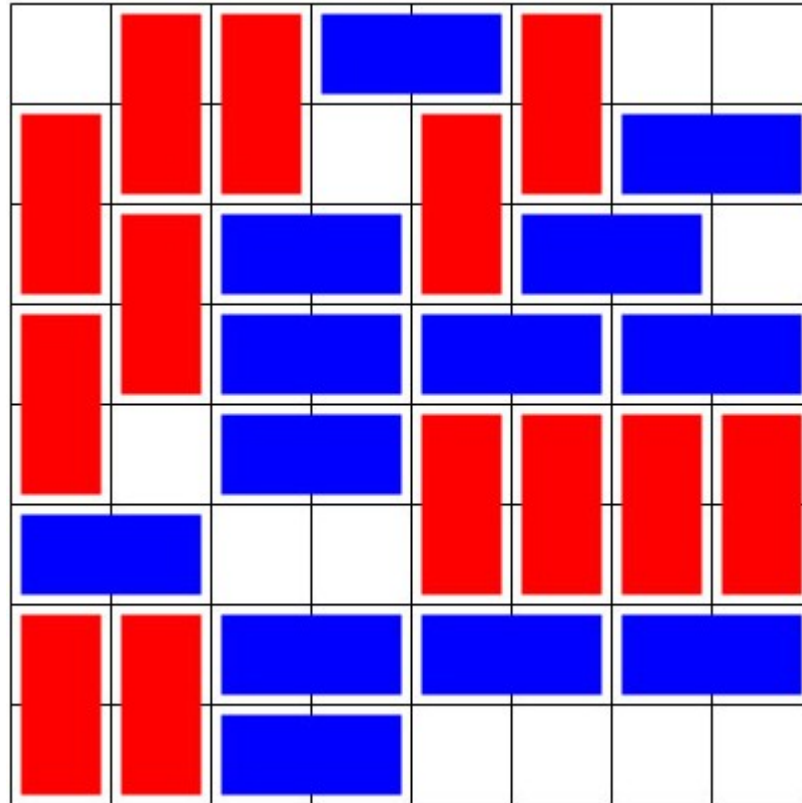
Go

RAVE vs UCT

Game	Score
Atarigo 8x8	94.2 %
Domineering	72.6 %
Go 9x9	73.2 %
Knightthrough	56.2 %
Three Color Go 9x9	70.8 %

GRAVE vs RAVE

Game	Score
Atarigo 8x8	88.4 %
Domineering	62.4 %
Go 9x9	54.4 %
Knightthrough	67.2 %
Three Color Go 9x9	57.2 %

Parallelization of MCTS

- Root Parallelization.

- Tree Parallelization (virtual loss).

- Leaf Parallelization.

MCTS

- Great success for the game of Go since 2007.
- Much better than all previous approaches to computer Go.

AlphaGo

Lee Sedol is among the strongest and most famous 9p Go player :

AlphaGo has won 4-1 against Lee Sedol in March 2016

AlphaGo Master wins 3-0 against Ke Jie, 60-0 against pros.

AlphaGo Zero wins 89-11 against AlphaGo Master in 2017.

General Game Playing

- General Game Playing = play a new game just given the rules.
- Competition organized every year by Stanford.
- Ary world champion in 2009 and 2010.
- All world champions since 2007 use MCTS.

Other two-player games

- Hex : 2009

- Amazons : 2009

- Lines of Action : 2009
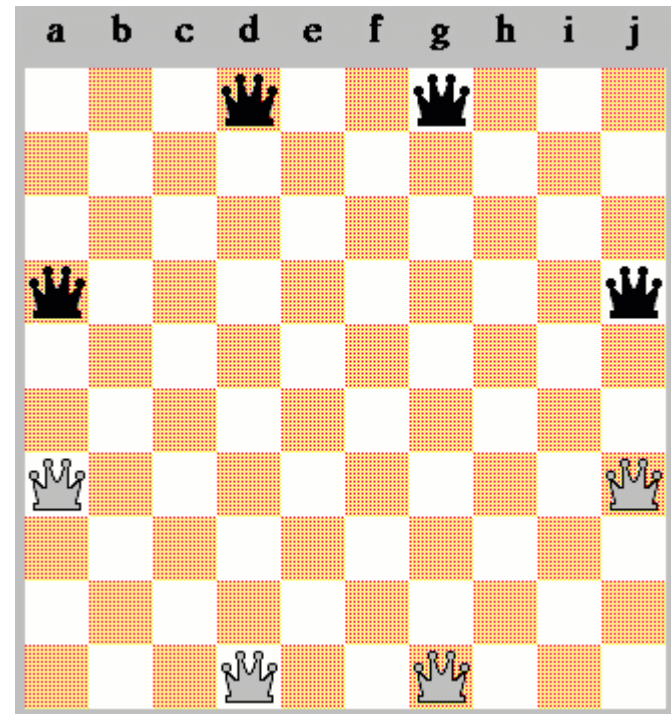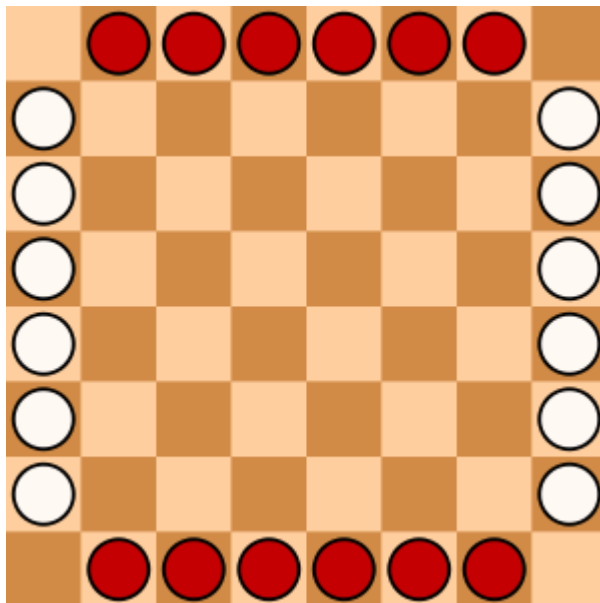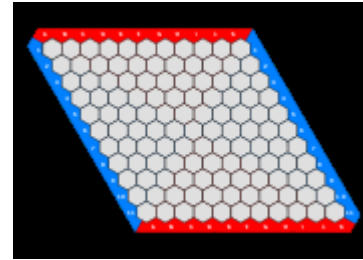
MCTS Solver

- When a subtree has been completely explored the exact result is known.

- MCTS can solve games.

- Score Bounded MCTS is the extension of pruning to solving games with multiple outcomes.

Article

PaccMann^RL: De novo generation of hit-like anticancer molecules from transcriptomic data via reinforcement learning
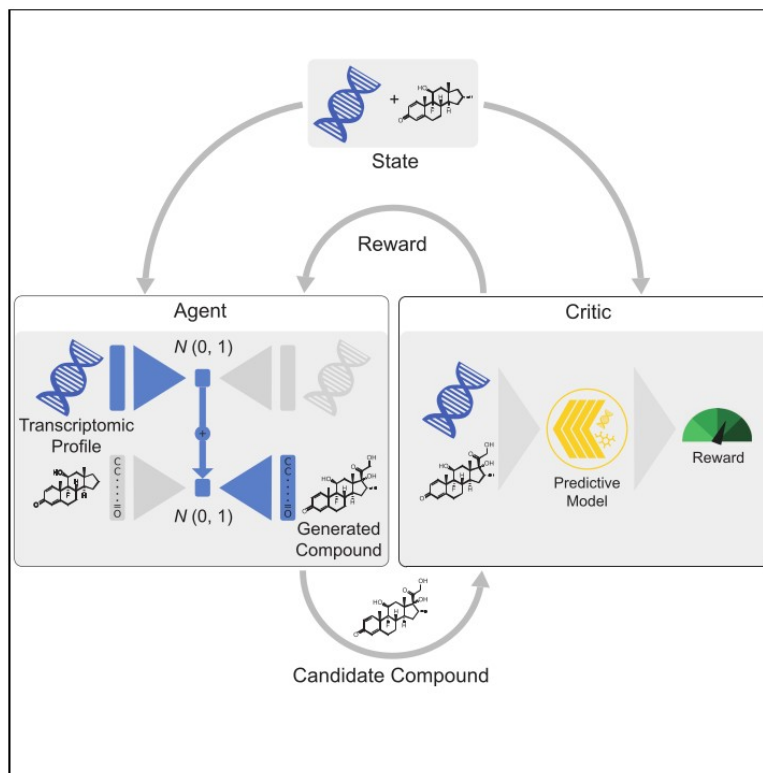
Jannis Born,
Matteo Manica, Ali
Oskooei, Joris
Cadow, Greta
Markert, María
Rodríguez
Martínez

jab@zurich.ibm.com (J.B.)
tte@zurich.ibm.com (M.M.)
mrm@zurich.ibm.com (M.R.M.)

Highlights

A conditional generative
model for de novo design
of anticancer hit
molecules is devised

Drug sensitivity and
toxicity models steer the
molecule design via
reinforcement learning

Molecules are designed to
target individual
transcriptomic profiles of
cell lines

Targeted, hit-like
molecules are generated
more frequently, even for
unseen cell lines

In silico, the molecules
exhibit similar
physicochemical
properties to real cancer
drugs

Predicting the structure of large protein complexes using AlphaFold and Monte Carlo tree search

Patrick Bryant[1,2]*, Gabriele Pozzati[1,2], Wensi Zhu[1,2], Aditi Shenoy[1,2], Petras Kundrotas[1,3] and Arne Elofsson[1,2]

[1]Science for Life Laboratory, 172 21 Solna, Sweden
[2]Department of Biochemistry and Biophysics, Stockholm University, 106 91 Stockholm, Sweden
[3]Center for Computational Biology, The University of Kansas, Lawrence, KS 66047, USA

*Corresponding author, email: patrick.bryant@scilifelab.se

Abstract

AlphaFold can predict the structure of single- and multiple-chain proteins with very high accuracy. However, the accuracy decreases with the number of chains, and the available GPU memory limits the size of protein complexes which can be predicted. Here we show that one can predict the structure of large complexes starting from predictions of subcomponents. We assemble 91 out of 175 complexes with 10-30 chains from predicted subcomponents using Monte Carlo tree search, with a median TM-score of 0.51. There are 30 highly accurate complexes (TM-score ≥0.8, 33% of complete assemblies). We create a scoring function, mpDockQ, that can distinguish if assemblies are complete and predict their accuracy. We find that complexes containing symmetry are accurately assembled, while asymmetrical complexes remain challenging. The method is freely available and accesible as a Colab notebook
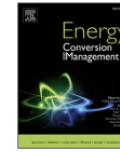https://colab.research.google.com/github/patrickbryant1/MoLPC/blob/master/MoLPC.ipynb.

Keywords
Protein structure prediction
AlphaFold
Complex assembly
Monte Carlo tree search

ELSEVIER

Wind farm layout optimization using adaptive evolutionary algorithm with Monte Carlo Tree Search reinforcement learning

Fangyun Bai [a], Xinglong Ju [b], Shouyi Wang [c], Wenyong Zhou [a], Feng Liu [d,*]

[a] Department of Management Science and Engineering, Tongji University, Shanghai, China
[b] Price College of Business, University of Oklahoma, Norman, OK, 73019, USA
[c] Department of Industrial, Manufacturing, & Systems Engineering, The University of Texas at Arlington, Arlington, TX 76019, USA
[d] School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ 07030, USA

ARTICLE INFO

ABSTRACT

Recent years have witnessed an enormous growth of wind farm capacity worldwide. Due to the wake effect, the velocity of incoming wind is reduced for the wind turbines in the downwind directions, thus causing discounted power generation in a wind farm. Previously, a self-informed adaptivity mechanism in evolutionary algorithms was introduced by the authors, which is inspired by the individuals' self-adaptive capability to fit the environment in the natural world, where relocating the worst wind turbine with a surrogate model informed mechanism was found to be effective in improving the power conversion efficiency. In this paper, the exploitation capability in the adaptive genetic algorithm is further improved by casting the relocation of multiple wind turbines into a single-player reinforcement learning problem, which is further addressed by Monte-Carlo Tree Search embedded within the evolutionary algorithm. In contrast to the moderate improvements of the authors' previous algorithms, significant improvement is achieved due to the enhanced algorithmic exploitation. The new algorithm is also applied to solve the optimal layout problem for a recently approved wind farm in New Jersey, and showed better performance against the benchmark algorithms.

1. Introduction

Climate change and global warming have been a major concern for sustainable social and economic development around the world. It is estimated that the portion of renewable energy should be at least 67% among all resources of energies in 2050 compared to 20% in 2018 [1], in order to meet the target of limiting the global temperature within 1.5 °C above the preindustrial level according to Intergovernmental Panel on Climate Change (IPCC) [2] on Climate Change [2]. Wind energy has become an indispensable alternative to fossil fuels given its advantage of being sustainable, economically competitive, and abundant [3], which has shown steady growth of capacity and power generation over the past decades. In 2020 alone, the US has grown the capacity of wind energy by 23 Gigawatts (GW), the largest in history. Optimal design of wind farms has been thoroughly investigated from different perspectives, such as cite selection [4], wind turbine design [5], electrical cable placement [6], wake effect modeling [7], wind speed forecasting [8], and wind power prediction [9].

One challenge for maximizing the power output is to find an optimal layout of the wind turbines to reduce the wake effect [10]. Wake effect refers to the situation when the input wind speed for the wind turbines in the downwind directions are discounted after the wind turbines in the upwind directions absorb the kinetic energy from the wind [11]. In addition to the energy output decrease caused by the wake effect, the wake effect can also cause fatigue loads due to the increased turbulence of wind flow, which can cause mechanical failure and shorten the life expectancy of wind turbines [12]. Every percentage of improvement in efficiency can mean significant profit income, thus requires a meticulous effort of investigation. The wind farm layout optimization problem (WFLOP) is a highly complicated problem as even 30 wind turbines could lead to a high 10^{44} potential solutions given discrete and uniform turbine types [13] and suffer from "curse of dimensionality" for increase numbers of wind turbines [14]. With the recent trend of constructing wind farms with larger capacities, the WFLOP is even more challenging to solve. The nonconvex and NP-hard nature in WFLOP poses challenges for exact solution methods such as linear programming, mixed integer programming. However, there are some attempts using mixed integer programming [15,16]. Many nature-inspired, population-based meta-heuristic algorithms have been proposed to solve the WFLOP, such as

* Corresponding author.
E-mail addresses: 1510353@tongji.edu.cn (F. Bai), xinglong.ju@ou.edu (X. Ju), shouyiw@uta.edu (S. Wang), liu22@stevens.edu (F. Liu).

Décodage guidé par un discriminateur avec le Monte Carlo Tree Search pour la génération de texte contrainte

Antoine Chaffin[1, 2]   Vincent Claveau[1]   Ewa Kijak[1]
(1) Univ. Rennes – CNRS – IRISA, Rennes, France
(2) IMATAG, Rennes, France
`prenom.nom@irisa.fr`

RÉSUMÉ

Dans cet article, nous explorons comment contrôler la génération de texte au moment du décodage pour satisfaire certaines contraintes (e.g. être non toxique, transmettre certaines émotions...), sans nécessiter de ré-entrainer le modèle de langue. Pour cela, nous formalisons la génération sous contrainte comme un processus d'exploration d'arbre guidé par un discriminateur qui indique dans quelle mesure la séquence associée respecte la contrainte. Nous proposons plusieurs méthodes originales pour explorer cet arbre de génération, notamment le Monte Carlo Tree Search (MCTS) qui fournit des garanties théoriques sur l'efficacité de la recherche. Au travers d'expériences sur 3 jeux de données et 2 langues, nous montrons que le décodage par MCTS guidé par les discriminateurs permet d'obtenir des résultats à l'état-de-l'art. Nous démontrons également que d'autres méthodes de décodage que nous proposons, basées sur le re-ordonnancement, peuvent être réellement efficaces lorsque la diversité parmi les propositions générées est encouragée.

ABSTRACT

Discriminator-guided decoding with Monte Carlo Tree Search for constrained text generation

In this paper, we explore how to control text generation at decoding time to satisfy certain constraints (eg. being non-toxic, conveying certain emotions...) without fine-tuning the language model. Precisely, we formalize constrained generation as a tree exploration process guided by a discriminator that indicates how well the associated sequence respects the constraint. We propose several original methods to search this generation tree, notably the Monte Carlo Tree Search (MCTS) which provides theoretical guarantees on the search efficiency.Through 3 tasks and 2 languages, we show that discriminator-guided MCTS decoding achieves state-of-the-art results without having to tune the language model. We also demonstrate that other proposed decoding methods based on re-ranking can be really effective when diversity among the generated propositions is encouraged.

MOTS-CLÉS : Génération de texte, génération collaborative, décodage, Monte Carlo Tree Search.

KEYWORDS: Text generation, collaborative generation, decoding, Monte Carlo Tree Search.

1   Introduction et état de l'art

Les modèles de langue génératifs (LM pour *Language Models*) existent depuis longtemps, mais avec l'avènement de l'architecture des transformers (Vaswani *et al.*, 2017) et l'augmentation des capacités de calcul, ils sont maintenant capables de générer des textes longs et bien écrits dans beaucoup de situations. Ces LM, tels GPT-2 et 3 (Radford *et al.*, 2019; Brown *et al.*, 2020), ont été

Sensor tasking in the cislunar regime using Monte Carlo Tree Search

Samuel Fedeler [a],[*], Marcus Holzinger [a], William Whitacre [b]

[a] *University of Colorado at Boulder, Boulder, CO, USA*
[b] *The Charles Stark Draper Laboratory, Inc., Cambridge, MA, USA*

Abstract

Maintaining tracks on space objects with limited sets of observers is a critical problem, made more urgent with exponential growth in the population of near-Earth satellites. An optimally convergent decision making methodology is proposed for sensor tasking, using the Monte Carlo Tree Search methodology. This methodology is underpinned by the partially observable Markov decision process framework; it utilizes polynomial exploration of the action space, and double progressive widening to avoid curses of history. The developed tasking techniques are applied to a large-scale application considering the tracking problem in the emerging cislunar regime. Uncertainty studies are performed for a set of 500 objects in a variety of candidate periodic and highly elliptical orbits, with realistic sensor models incorporating physical parameters and explicit probability of detection. These simulations are utilized as a means to evaluate observer quality, considering candidate space-based sensors following L1 Lyapunov and L2 Northern Halo orbits. Results demonstrate the importance of space-based observers for maintaining estimates on objects in cislunar space and give insight into the criticality of relative motion between observers and targets when optical measurements are utilized.
© 2022 COSPAR. Published by Elsevier B.V. All rights reserved.

Keywords: Sensor Tasking; Monte Carlo Tree Search; Cislunar SSA; Optical Sensor Systems; Orbit Determination

1. Introduction

Choosing tasking policies for a set of sensors maintaining custody of space objects in various orbit regimes has long been a relevant problem in Space Domain Awareness (SDA). As a result of accelerating growth in space object (SO) populations, it is imperative that limited observational assets are utilized efficiently. Collision concerns have increased in recent years, especially in well-populated environments such as low-Earth orbit; as such, ensuring collision avoidance requires careful tracking of in-orbit satellites and debris. The problem at hand quickly becomes combinatoric as the object catalog considered expands, and

multiple competing objectives are often desired to leverage uncued detection of objects in addition to catalog maintenance. As such, the sensor tasking problem is largely broken into tractable subproblems, in which the objective is to capture a single aspect of the overarching goal.

Also of interest when considering the sensor tasking problem is application to the cislunar regime of space. Relatively little literature has been produced on the subject, and the region is expected to be a growing frontier for space exploration in coming years (Holzinger et al., 2021; Bobskill, 2012). As volumes of space further from Earth are considered, dynamic complexities are introduced, and it is no longer sufficient to neglect perturbations from the Moon and the Sun. Trajectories in the cislunar regime are not necessarily stable, and many initial conditions are chaotic even when the circular restricted three-body simplification is applied for analysis. Periodic orbits exist in the circular and elliptic-restricted three-body problems (Folta

---

Check for
updates

Artificial intelligence-based inventory management: a Monte Carlo tree search approach

Deniz Preil[1] · Michael Krapp[1]

Abstract

The coordination of order policies constitutes a great challenge in supply chain inventory management as various stochastic factors increase its complexity. Therefore, analytical approaches to determine a policy that minimises overall inventory costs are only suitable to a limited extent. In contrast, we adopt a heuristic approach, from the domain of artificial intelligence (AI), namely, Monte Carlo tree search (MCTS). To the best of our knowledge, MCTS has neither been applied to supply chain inventory management before nor is it yet widely disseminated in other branches of operations research. We develop an offline model as well as an online model which bases decisions on real-time data. For demonstration purposes, we consider a supply chain structure similar to the classical beer game with four actors and both stochastic demand and lead times. We demonstrate that both the offline and the online MCTS models perform better than other previously adopted AI-based approaches. Furthermore, we provide evidence that a dynamic order policy determined by MCTS eliminates the bullwhip effect.

Keywords Monte Carlo tree search · Supply chain inventory management · Artificial intelligence · Bullwhip effect

1 Introduction

The supply chain management literature spans a wide range of topics, such as facility location, production, scheduling, transportation, return of goods, forecasting, and inventory management, this last being the main subject of this paper. One key task of supply chain management is the integrated planning and control of the inventory of all actors in the supply chain, from the source of supply to the end user, to reduce the overall inventory costs while improving customer service (Ellram 1991). Reducing inventory costs is of major importance, as these costs account for a considerable proportion of the companies' total logistics costs (Rood-

---

✉ Deniz Preil
deniz.preil@wiwi.uni-augsburg.de

[1] Department of Quantitative Methods in Economics, University of Augsburg, Universitaetsstr. 16, 86159 Augsburg, Germany

A novel real-time energy management strategy based on Monte Carlo Tree Search for coupled powertrain platform via vehicle-to-cloud connectivity

Xiao Yu [a, b], Cheng Lin [a, b, *], Peng Xie [a, b], Sheng Liang [a, b]

[a] National Engineering Research Center of Electric Vehicles, Beijing Institute of Technology, Beijing, 100081, China
[b] Collaborative Innovation Center of Electric Vehicles in Beijing, Beijing Institute of Technology, Beijing, 100081, China

ARTICLE INFO

ABSTRACT

To improve the performance and efficiency of the energy management strategy used in electric vehicles equipped with a dual-motor coupled powertrain platform, this study proposes a systematic real-time search approach via vehicle-to-cloud (V2C) connectivity to reduce the battery degradation and electrical consumption by control working mode and split torque. To be specific, the Monte Carlo Tree Search (MCTS) is employed to search for optimal control sequence in the velocity feasible range in the cloud platform, considering battery loss and electric cost. The logic of time and velocity range updating is proposed as the solution for abrupt traffic changes. To evaluate the effectiveness of the proposed method, a rule-based and an online DP (Dynamic Programming) -based strategy is developed as the baseline approach. Meanwhile, the assessment conditions include standard cycles following power noise and real-world driving cycles. Finally, actual vehicle and hardware-in-the-loop (HIL) experimental results demonstrate that the proposed method significantly outperforms other strategies, the average total cost is 0.36 USD/km, and the improvements are **12.9**% and **11.4**% compared to the rule-based and online DP-based approaches, respectively.

© 2022 Elsevier Ltd. All rights reserved.

1. Introduction

1.1. Background

Electrification and increased energy efficiency of vehicles, particularly commercial vehicles, are considered critical factors for carbon emission mitigation [1,2]. Consequently, the demand for applications in complex traffic scenarios over the past few years has led to significant growth in research of powertrain configuration and energy management strategy (EMS) design while enhancing the collaboration and efficiency of these technologies [3,4]. For instance, in recent years, multi-motor with multi-gear coupled powertrains have been widely used in electric vehicles (EVs) [5,6]. However, the cost and performance of the powertrain-battery system are heavily dependent on its EMS [7]. Therefore, it is essential to equip the complex system with a collaborative EMS to minimize EV power consumption and battery degradation costs. Meanwhile, along with the rapid development of vehicle-to-cloud (V2C) connectivity, cloud computing, and intelligent transportation system, efficient machine learning arithmetic and accurate real-time approaches are becoming feasible for EVs [8,9].

1.2. Method review

A number of EMS have been developed, which can be classified into rule-based, optimization-based, and optimal rule extraction methods to achieve better performance in the operation of EVs [10]. The former is most prevalent in practical engineering, which is characterized by clear logical architecture, straightforward operation, and rapid verification [11,12]. However, this strategy constantly demands plenty of calibration for diverse operation conditions to validate the core control thresholds, making it resistant to optimizing operation [13].

On the contrary, the optimization-based approaches are typically used in real-time EMS, such as dynamic programming (DP) [14,15], model predictive control (MPC) [16], and pontryagin's minimum principle (PMP) [17]. DP provides a global optimization

---

Monte Carlo Tree Search based Hybrid Optimization of Variational Quantum Circuits

Jiahao Yao[*]   JIAHAOYAO@BERKELEY.EDU
Department of Mathematics, University of California, Berkeley
Berkeley, CA 94720, USA

Haoya Li[*]   LIHAOYA@STANFORD.EDU
Department of Mathematics, Stanford University
Stanford, CA, 94305, USA

Marin Bukov
Department of Physics, St. Kliment Ohridski University of Sofia
5 James Bourchier Blvd, 1164 Sofia, Bulgaria
Max Planck Institute for the Physics of Complex Systems, Nöthnitzer Str. 38, 01187 Dresden, Germany

Lin Lin
Department of Mathematics, University of California, Berkeley
Computational Research Division, Lawrence Berkeley National Laboratory
Challenge Institute for Quantum Computation, University of California, Berkeley
Berkeley, CA 94720, USA

Lexing Ying
Department of Mathematics, Stanford University
Stanford, CA 94305, USA

Abstract

Variational quantum algorithms stand at the forefront of simulations on near-term and future fault-tolerant quantum devices. While most variational quantum algorithms involve only continuous optimization variables, the representational power of the variational ansatz can sometimes be significantly enhanced by adding certain discrete optimization variables, as is exemplified by the generalized quantum approximate optimization algorithm (QAOA). However, the hybrid discrete-continuous optimization problem in the generalized QAOA poses a challenge to the optimization. We propose a new algorithm called MCTS-QAOA, which combines a Monte Carlo tree search method with an improved natural policy gradient solver to optimize the discrete and continuous variables in the quantum circuit, respectively. We find that MCTS-QAOA has excellent noise-resilience properties and outperforms prior algorithms in challenging instances of the generalized QAOA.

---

[*] J.Y. & H.L. contributed equally

Symbolic Physics Learner: Discovering governing equations via Monte Carlo tree search

Fangzheng Sun[1], **Yang Liu**[2], **Jian-Xun Wang**[3], and **Hao Sun**[4,5,*]

[1]Department of Civil and Environmental Engineering, Northeastern University, Boston, MA 02115, USA
[2]School of Engineering Sciences, University of the Chinese Academy of Sciences, Beijing, 101408, China
[3]Department of Aerospace and Mechanical Engineering, University of Notre Dame, Notre Dame, IN, USA
[4]Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, 100872, China
[5]Beijing Key Laboratory of Big Data Management and Analysis Methods, Beijing, 100872, China

Abstract

Nonlinear dynamics is ubiquitous in nature and commonly seen in various science and engineering disciplines. Distilling analytical expressions that govern nonlinear dynamics from limited data remains vital but challenging. To tackle this fundamental issue, we propose a novel Symbolic Physics Learner (SPL) machine to discover the mathematical structure of nonlinear dynamics. The key concept is to interpret mathematical operations and system state variables by computational rules and symbols, establish symbolic reasoning of mathematical formulas via expression trees, and employ a Monte Carlo tree search (MCTS) agent to explore optimal expression trees based on measurement data. The MCTS agent obtains an optimistic selection policy through the traversal of expression trees, featuring the one that maps to the arithmetic expression of underlying physics. Salient features of the proposed framework include search flexibility and enforcement of parsimony for discovered equations. The efficacy and superiority of the PSL machine are demonstrated by numerical examples, compared with state-of-the-art baselines.

1 Introduction

We usually learn the behavior of a nonlinear dynamical system through its nonlinear governing differential equations. These equations can be formulated as

$$\dot{\mathbf{y}}(t) = d\mathbf{y}/dt = \mathcal{F}(\mathbf{y}(t)) \tag{1}$$

where $\mathbf{y}(t) = \{y_1(t), y_2(t), ..., y_n(t)\} \in \mathbb{R}^{1 \times n}$ denotes the system state at time t, $\mathcal{F}(\cdot)$ a nonlinear function set defining the state motions and n the system dimension. The explicit form of $\mathcal{F}(\cdot)$ for some nonlinear dynamics remains underexplored. For example, in a mounted double pendulum system, the mathematical description of the underlying physics might be unclear due to unknown viscous and frictional damping forms. These uncertainties yield critical demands for the discovery of nonlinear dynamics given observational data. Nevertheless, distilling the analytical form of the governing equations from limited and noisy measurement data, commonly seen in practice, is an intractable challenge.

Ever since the early work on the data-driven discovery of nonlinear dynamics [1, 2], many scientists have stepped into this field of study. In the recent decade, the escalating advances in machine learning, data science, and computing power enabled several milestone efforts of unearthing the governing equations for nonlinear dynamical systems. Notably, a breakthrough model named SINDy based on

---

*Corresponding author

Controlling Perceived Emotion in Symbolic Music Generation
with Monte Carlo Tree Search

Lucas N. Ferreira[1], **Lili Mou**[1], **Jim Whitehead**[2], **Levi H. S. Lelis**[1]

[1]Alberta Machine Intelligence Institute (Amii), University of Alberta
[2]Computational Media Department, University of California, Santa Cruz
lnferrei@ualberta.ca, lmou@ualberta.ca, ejw@soe.ucsc.edu, levi.lelis@ualberta.ca

Abstract

This paper presents a new approach for controlling emotion in symbolic music generation with Monte Carlo Tree Search. We use Monte Carlo Tree Search as a decoding mechanism to steer the probability distribution learned by a language model towards a given emotion. At every step of the decoding process, we use Predictor Upper Confidence for Trees (PUCT) to search for sequences that maximize the average values of emotion and quality as given by an emotion classifier and a discriminator, respectively. We use a language model as PUCT's policy and a combination of the emotion classifier and the discriminator as its value function. To decode the next token in a piece of music, we sample from the distribution of node visits created during the search. We evaluate the quality of the generated samples with respect to human-composed pieces using a set of objective metrics computed directly from the generated samples. We also perform a user study to evaluate how human subjects perceive the generated samples' quality and emotion. We compare PUCT against Stochastic Bi-Objective Beam Search (SBBS) and Conditional Sampling (CS). Results suggest that PUCT outperforms SBBS and CS in almost all metrics of music quality and emotion.

Introduction

Neural language models (LMs) are currently one of the leading generative models for algorithmic music composition (Yang et al. 2019). Neural LMs are trained to predict the next musical token with a data set of symbolic music pieces (Huang et al. 2018). A major problem with neural LMs is the lack of control for specific musical features on the decoded pieces. For example, one cannot control an LM trained on classical piano pieces to compose a tense piece for a scene of a thriller movie. It is hard to control the generative process of these models because they typically have a large number of parameters, and it is not clear what parameters affect what musical features. Controlling the perceived emotion of generated music is a central problem in Affective Music Composition (Williams et al. 2015b), with applications in games (Williams et al. 2015a), stories (Davis and Mohammad 2014), and sonification (Chen, Bowers, and Durrant 2015).

Controlling neural LMs to generate music with a target emotion started to be explored only recently. Two

prominent decoding methods are Conditional Sampling (CS, Hung et al. 2021) and Stochastic Bi-Objective Beam Search (SBBS, Ferreira, Lelis, and Whitehead 2020). The former consists of sampling from the LM with conditional signals that represent a target emotion. The latter is a variant of beam search that steers the probability distribution of LMs toward a target emotion by using a music emotion classifier at decoding time. In this paper, we propose a new decoding algorithm inspired by AlphaZero (Silver et al. 2017). AlphaZero uses Predictor Upper Confidence for Trees (PUCT, Rosin 2011) as a policy improvement operator. We use PUCT to change an initial distribution given by an LM to a distribution that has higher musical quality and conveys a target emotion in the decoded pieces.

We start with a neural LM that represents a prior probability distribution over tokens. At decoding time, we run multiple PUCT iterations to build a distribution of node visits that improves the LM distribution and steers it towards a target emotion. This is achieved by using both a music emotion classifier and a music discriminator as the value function of PUCT, while the LM is used as its policy. The policy and value function jointly define the set of nodes evaluated in the search. The frequency of node visits offers a distribution that produces music with higher quality than the LM initial distribution, shifted to convey the emotion given by the emotion classifier. We sample the next token in the sequence from this new distribution.

We train a neural LM with a Linear Transformer (Katharopoulos et al. 2020) on the unlabelled pieces of the VGMIDI data set (Ferreira and Whitehead 2019). The VGMIDI data set is a collection of 928 piano pieces from video game soundtracks, where 200 pieces are labelled according to the circumplex model of emotion (Russell 1980). We extend the number of unlabelled pieces of the VGMIDI data set from 728 to 3,850. Following the approach of Ferreira, Lelis, and Whitehead (2020), we train the music emotion classifier with the 200 labelled pieces of the VGMIDI data set by fine-tuning the Linear Transformer LM with a classification head. We train the music discriminator with the same fine-tuning approach. This discriminator distinguishes real data from the data created by sampling from the LM.

We conducted two experiments to evaluate our PUCT decoding method. The first one evaluates the quality and emotion of the generated samples with respect to human com-

Breakthrough
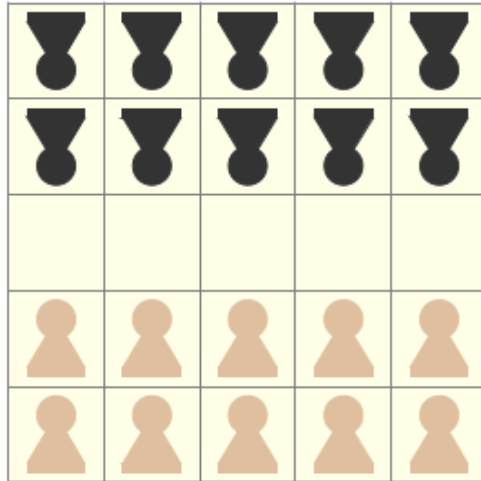
(a) Starting position on size 5 × 5.
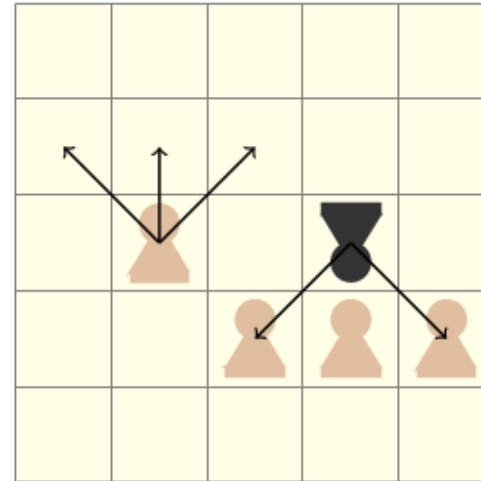
(b) Possible movements.

- Write the Board and Move classes for Breakthrough 5x5.
- Write the function for the possible moves.
- Write a program to play random games at Breakthrough 5x5.

Breakthrough

- The Move class contains the color, the starting and arriving locations of a pawn.

```
class Move(object):

    def __init__(self, color, x1, y1, x2, y2):

        self.color = color

        self.x1 = x1

        self.y1 = y1

        self.x2 = x2

        self.y2 = y2
```

Breakthrough

- The Board class initializes the board with two rows of Black and two rows of White pawns:

```
Dx = 5

Dy = 5

Empty = 0

White = 1

Black = 2

class Board(object):

    def __init__(self):

        self.h = 0

        self.turn = White

        self.board = np.zeros ((Dx, Dy))

        for i in range (0, 2):

            for j in range (0, Dy):

                self.board [i] [j] = White

        for i in range (Dx - 2, Dx):

            for j in range (0, Dy):

                self.board [i] [j] = Black
```

Breakthrough

- Test, in the Move class, if a move is valid for a given board:

```
def valid (self, board):
    if self.x2 >= Dx or self.y2 >= Dy or self.x2 < 0 or self.y2 < 0:
        return False
    if self.color == White:
        if self.x2 != self.x1 + 1:
            return False
        if board.board [self.x2] [self.y2] == Black:
            if self.y2 == self.y1 + 1 or self.y2 == self.y1 - 1:
                return True
            return False
        elif board.board [self.x2] [self.y2] == Empty:
            if self.y2 == self.y1 + 1 or self.y2 == self.y1 - 1 or self.y2 == self.y1:
                return True
            return False
        ...
```

Breakthrough

```python
elif self.color == Black:
    if self.x2 != self.x1 - 1:
        return False
    if board.board [self.x2] [self.y2] == White:
        if self.y2 == self.y1 + 1 or self.y2 == self.y1 - 1:
            return True
        return False
    elif board.board [self.x2] [self.y2] == Empty:
        if self.y2 == self.y1 + 1 or self.y2 == self.y1 - 1 or self.y2 == self.y1:
            return True
        return False
return False
```

Breakthrough

- Generate the legal moves in the Board class:

```python
def legalMoves(self):
    moves = []
    for i in range (0, Dx):
        for j in range (0, Dy):
            if self.board [i] [j] == self.turn:
                for k in [-1, 0, 1]:
                    for l in [-1, 0, 1]:
                        m = Move (self.turn, i, j, i + k, j + l)
                        if m.valid (self):
                            moves.append (m)
    return moves
```

Playouts

- Write, in the Board class, a score function to score a game (1.0 if White wins, 0.0 else) and a terminal function to detect the end of the game.

- Write, in the Board class, a playout function that plays a random game from the current state and returns the result of the random game.

Flat Monte Carlo

- For each move of the current state, do a fixed number of playouts starting with the move.

- Keep the move with the greatest number of won playouts.

- Play the move with the greatest mean when all the playouts are finished.

UCB

- Keep statistics for all the moves of the current state.

- For each move of the current state, keep the number of playouts starting with the move and the number of playouts starting with the move that have been won.

- Play the most simulated move when all the playouts are finished.

UCB

Choose the first move at the root according to UCB before each playout:

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

In which

- w_i = number of wins after the i-th move
- n_i = number of simulations after the i-th move
- c = exploration parameter (theoretically equal to $\sqrt{2}$)
- t = total number of simulations for the parent node

UCB vs Flat

- Make UCB with 200 playouts play 100 games against Flat with 200 playouts.

- Winrate ?

- Tune the UCB constant (hint 0.4).

Table de transposition

- Chaque etat est associé à un hash code.

- On utilise le hachage de Zobrist.

- Chaque case de chaque pièce est associée à un nombre aléatoire différent.

- Le code d'un état est le XOR des nombres aléatoires des pièces présentes sur le plateau de jeu.

- Pourquoi utilise-t-on un XOR ?

- Combien utilise-t-on de nombres aléatoires différents pour coder un échiquier ?

Table de transposition

- On utilise un XOR parce que :

- Le XOR de valeurs bien réparties est bien réparti.

- Le XOR est une opération très rapide sur les bits.

- (b XOR a) XOR a = b

- Pour ajouter une pièce ou pour la retirer il suffit de faire le XOR avec le nombre aléatoire de la pièce => on peut calculer le code d'une position efficacement à partir du code du parent.

Table de transposition
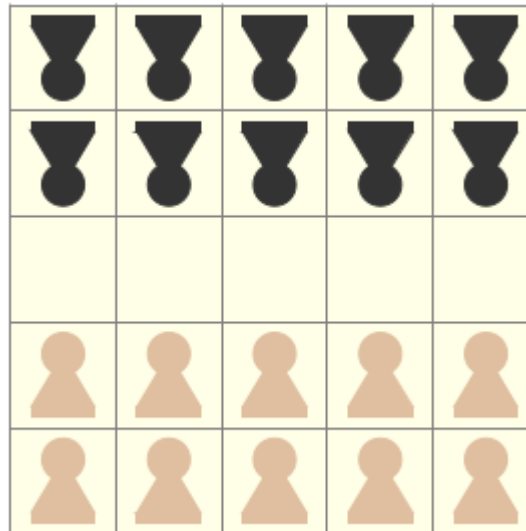
- Aux échecs :
- pièces * cases = 12 * 64 = 768
- droits de roque 4
- prise en passant 16
- trait 1

- total 789

- A Breakthrough 5x5 : 50 + 1 for the turn

Table de transposition

- En supposant que les nombres aléatoires pour Breakthrough 5x5 vont de 1 à 25 pour noir et de 26 à 50 pour blanc (la première ligne vaut 1, 2, 3, 4, 5 pour noir et 26, 27, 28, 29, 30 pour blanc) et que le nombre aléatoire du trait vaut 51 :

- Quel est le hashcode h1 de la position de départ ?

- Comment calculer le hash h2 de la position où on a avancé le pion blanc le plus à gauche, à partir du hash h1 de la position de départ ?

Table de transposition

h1 = 0

h2 = h1 ^ 41 ^ 36 ^ 51 = 62

Table de transposition

- Ecrire le code pour le tirage des nombres aléatoires associés aux cases et aux pions.

- Modifier le programme pour qu'un damier soit toujours associé à un hashcode de Zobrist calculé incrémentalement lorsqu'on joue un coup.

Table de transposition

```python
hashTable = []
for k in range (3):
    l = []
    for i in range (Dx):
        l1 = []
        for j in range (Dy):
            l1.append (random.randint (0, 2 ** 64))
        l.append (l1)
    hashTable.append (l)
hashTurn = random.randint (0, 2 ** 64)
```

Transposition Table

- An entry of a state in the transposition table contains :

- The hashcode of the stored state.

- The total number of playouts of the state.

- The number of playouts for each possible move.

- The number of wins for each possible move.

Transposition Table

- First Option (C++ like) :
  - Write a class TranspoMonteCarlo containing the data associated to a state.
  - Write a class TableMonteCarlo that contains a table of list of entries.
  - Each entry is an instance of TranspoMonteCarlo. The size of the table is 65535. The index in the table of a hashcode h is h & 65535.
  - The TableMonteCarlo class also contains the functions :
    - look (self, board) which returns the entry of board.
    - add (self, t) which adds en entry in the table.

Transposition Table

- Alternative : use a Python dictionary with the hash as a key and lists as elements.

- Each list contains 3 elements :
  - the total numbers of playouts,
  - the list of the number of playouts for each move,
  - the list of the number of wins for each move.

- Write a function that returns the entry of the transposition table if it exists or else None.

- Write a function that adds an entry in the transposition table.

UCT

procedure UCTSEARCH(s_0)
    while time available **do**
        SIMULATE($board, s_0$)
    end while
    $board.SetPosition(s_0)$
    return SELECTMOVE($board, s_0, 0$)
end procedure

procedure SIMULATE($board, s_0$)
    $board.SetPosition(s_0)$
    $[s_0, \ldots, s_T] =$ SIMTREE($board$)
    $z =$ SIMDEFAULT($board$)
    BACKUP($[s_0, \ldots, s_T], z$)
end procedure

UCT

procedure SIMTREE(*board*)
    $c = exploration\ constant$
    $t = 0$
    while not *board.GameOver*() **do**
        $s_t = board.GetPosition()$
        if $s_t \notin tree$ **then**
            NEWNODE(s_t)
            return $[s_0, \ldots, s_t]$
        end if
        $a = \text{SELECTMOVE}(board, s_t, c)$
        $board.Play(a)$
        $t = t + 1$
    end while
    return $[s_0, \ldots, s_{t-1}]$
end procedure

UCT

procedure SIMDEFAULT(*board*)
    while not *board.GameOver*() **do**
        $a =$ DEFAULTPOLICY(*board*)
        board.Play(a)
    end while
    return *board.BlackWins*()
end procedure

procedure BACKUP($[s_0, \ldots, s_T], z$)
    for $t = 0$ **to** T **do**
        $N(s_t) = N(s_t) + 1$
        $N(s_t, a_t) += 1$
        $Q(s_t, a_t) += \frac{z - Q(s_t, a_t)}{N(s_t, a_t)}$
    end for
end procedure

UCT

procedure SELECTMOVE(*board, s, c*)
    legal = *board.Legal*()
    if *board.BlackToPlay*() **then**
$$a^* = \text{argmax}_{a \in legal}\left(Q\left(s, a\right) + c\sqrt{\frac{\log N\left(s\right)}{N\left(s,a\right)}}\right)$$
    else
$$a^* = \text{argmin}_{a \in legal}\left(Q\left(s, a\right) - c\sqrt{\frac{\log N\left(s\right)}{N\left(s,a\right)}}\right)$$
    end if
    return a^*
end procedure

procedure NEWNODE(*s*)
    tree.Insert(*s*)
    $N(s) = 0$
    for all $a \in \mathcal{A}$ **do**
        $N(s, a) = 0$
        $Q(s, a) = 0$
    end for
end procedure

Algorithm 4 The UCT algorithm

UCT $(board, player, policy)$
$moves \leftarrow$ possible moves on $board$
if $board$ is terminal **then**
   return winner $(board)$
end if
$t \leftarrow$ entry of $board$ in the transposition table
if t exists **then**
   $bestValue \leftarrow -\infty$
   for m in $moves$ **do**
      $t \leftarrow t.totalPlayouts$
      $w \leftarrow t.wins[m]$
      $p \leftarrow t.playouts[m]$
      $value \leftarrow \frac{w}{p} + c \times \sqrt{\frac{log(t)}{p}}$
      if $value > bestValue$ **then**
         $bestValue \leftarrow value$
         $bestMove \leftarrow m$
      end if
   end for
   play $(board, bestMove)$
   $player \leftarrow$ opponent $(player)$
   $res \leftarrow$ UCT $(board, player, policy)$
   update t with res
else
   $t \leftarrow$ new entry of $board$ in the transposition table
   $res \leftarrow$ playout $(board, player, policy)$
   update t with res
end if
return res

UCT

- Exercise : write the Python code for UCT.
- The available functions that are problem specific are :
- board.playout () that returns the result of a playout.
- board.legalMoves () that returns the list of legal moves for the board.
- board.play (move) that plays the move on board.
- look (board) that returns the entry of the board in the transposition table.
- add (board) that adds an empty entry for the board in the transposition table.

AMAF

- All Moves As First (AMAF).

- AMAF calculates for each possible move of a state the average of the playouts that contain this move.

$$Pr(z = 1 \mid s_t = s, \ a_t = a) = Q^{\pi}(s, a),$$

$$Pr(z = 0 \mid s_t = s, \ a_t = a) = 1 - Q^{\pi}(s, a),$$

$$Pr(z = 1 \mid s_t = s, \ \exists u \geqslant t \ \text{s.t.} \ a_u = a) = \tilde{Q}^{\pi}(s, a),$$

$$Pr(z = 0 \mid s_t = s, \ \exists u \geqslant t \ \text{s.t.} \ a_u = a) = 1 - \tilde{Q}^{\pi}(s, a).$$

AMAF

- Exercise :

- Write a playout function memorizing the played moves.

- Add an integer code for moves in the Move class.

- Add AMAF statistics to the Transposition Table entries.

- Update the AMAF statistics of the Transposition Table.

RAVE

$$Q_\star(s,a) = \big(1 - \beta(s,a)\big) Q(s,a) + \beta(s,a) \tilde{Q}(s,a)$$

$$\mu = Q(s,a),$$

$$\tilde{\mu} = \tilde{Q}(s,a),$$

$$\mu_\star = Q_\star(s,a),$$

$$b = Q^\pi(s,a) - Q^\pi(s,a) = 0,$$

$$\tilde{b} = \tilde{Q}^\pi(s,a) - Q^\pi(s,a) = \tilde{B}(s,a),$$

$$b_\star = Q_\star^\pi(s,a) - Q^\pi(s,a),$$

$$\sigma^2 = \mathbb{E}\big[\big(Q(s,a) - Q^\pi(s,a)\big)^2 \,\big|\, N(s,a) = n\big],$$

$$\tilde{\sigma}^2 = \mathbb{E}\big[\big(\tilde{Q}(s,a) - \tilde{Q}^\pi(s,a)\big)^2 \,\big|\, \tilde{N}(s,a) = \tilde{n}\big],$$

$$\sigma_\star^2 = \mathbb{E}\big[\big(Q_\star(s,a) - Q_\star^\pi(s,a)\big)^2 \,\big|\, N(s,a) = n,\, \tilde{N}(s,a) = \tilde{n}\big],$$

$$e_\star^2 = \mathbb{E}\big[\big(Q_\star(s,a) - Q^\pi(s,a)\big)^2 \,\big|\, N(s,a) = n,\, \tilde{N}(s,a) = \tilde{n}\big].$$

RAVE

$$e_\star^2 = \sigma_\star^2 + b_\star^2$$
$$= (1-\beta)^2\sigma^2 + \beta^2\tilde{\sigma}^2 + \left(\beta\tilde{b} + (1-\beta)b\right)^2$$
$$= (1-\beta)^2\sigma^2 + \beta^2\tilde{\sigma}^2 + \beta^2\tilde{b}^2.$$

Differentiating with respect to β and setting to zero,

$$0 = 2\beta\tilde{\sigma}^2 - 2(1-\beta)\sigma^2 + 2\beta\tilde{b}^2,$$
$$\beta = \frac{\sigma^2}{\sigma^2 + \tilde{\sigma}^2 + \tilde{b}^2}.$$

RAVE

$$\sigma^2 = \frac{Q^\pi(s,a)(1 - Q^\pi(s,a))}{N(s,a)} \approx \frac{\mu_\star(1 - \mu_\star)}{n},$$

$$\tilde{\sigma}^2 = \frac{\tilde{Q}^\pi(s,a)(1 - \tilde{Q}^\pi(s,a))}{\tilde{N}(s,a)} \approx \frac{\mu_\star(1 - \mu_\star)}{\tilde{n}},$$

$$\beta = \frac{\tilde{n}}{n + \tilde{n} + n\tilde{n}\tilde{b}^2/\mu_\star(1 - \mu_\star)}.$$

In roughly even positions, $\mu_\star \approx \frac{1}{2}$, we can further simplify the schedule,

$$\beta = \frac{\tilde{n}}{n + \tilde{n} + 4n\tilde{n}\tilde{b}^2}.$$

RAVE

procedure Mc–Rave(s_0)
   while time available **do**
      Simulate($board, s_0$)
   end while
   $board.SetPosition(s_0)$
   return SelectMove($board, s_0, 0$)
end procedure

procedure Simulate($board, s_0$)
   $board.SetPosition(s_0)$
   $[s_0, a_0, \ldots, s_T, a_T] = \text{SimTree}(board)$
   $[a_{T+1}, \ldots, a_D], z = \text{SimDefault}(board, T)$
   Backup($[s_0, \ldots, s_T], [a_0, \ldots, a_D], z$)
end procedure

RAVE

procedure SIMDEFAULT($board, T$)
    $t = T + 1$
    while not $board.GameOver()$ **do**
        $a_t = $ DEFAULTPOLICY($board$)
        $board.Play(a_t)$
        $t = t + 1$
    end while
    $z = board.BlackWins()$
    return $[a_{T+1}, \ldots, a_{t-1}], z$
end procedure

procedure SIMTREE($board$)
    $t = 0$
    while not $board.GameOver()$ **do**
        $s_t = board.GetPosition()$
        if $s_t \notin tree$ **then**
            NEWNODE(s_t)
            $a_t = $ DEFAULTPOLICY($board$)
            return $[s_0, a_0, \ldots, s_t, a_t]$
        end if
        $a_t = $ SELECTMOVE($board, s_t$)
        $board.Play(a_t)$
        $t = t + 1$
    end while
    return $[s_0, a_0, \ldots, s_{t-1}, a_{t-1}]$
end procedure

RAVE

procedure SELECTMOVE(*board*, *s*)
    legal = *board*.Legal()
    if *board*.BlackToPlay() **then**
        return $\text{argmax}_{a \in legal}$ EVAL(*s*, *a*)
    else
        return $\text{argmin}_{a \in legal}$ EVAL(*s*, *a*)
    end if
end procedure

procedure EVAL(*s*, *a*)
    b = *pretuned constant bias value*
    $\beta = \frac{\tilde{N}(s,a)}{N(s,a) + \tilde{N}(s,a) + 4N(s,a)\tilde{N}(s,a)b^2}$
    return $(1 - \beta) Q(s, a) + \beta \tilde{Q}(s, a)$
end procedure

RAVE

procedure BACKUP($[s_0, \ldots, s_T], [a_0, \ldots, a_D], z$)
    for $t = 0$ **to** T **do**
        $N(s_t, a_t) += 1$
        $Q(s_t, a_t) += \frac{z - Q(s_t, a_t)}{N(s_t, a_t)}$
        for $u = t$ **to** D **step** 2 **do**
            if $a_u \notin [a_t, a_{t+2}, \ldots, a_{u-2}]$ **then**
                $\tilde{N}(s_t, a_u) += 1$
                $\tilde{Q}(s_t, a_u) += \frac{z - \tilde{Q}(s_t, a_t)}{\tilde{N}(s_t, a_t)}$
            end if
        end for
    end for
end procedure

procedure NEWNODE($board, s$)
    $tree.Insert(s)$
    for all $a \in board.Legal()$ **do**
        $N(s, a), Q(s, a), \tilde{N}(s, a), \tilde{Q}(s, a) =$ HEURISTIC($board, a$)
    end for
end procedure

RAVE

- Exercise :

- Compute the AMAF statistics for each node.

- Modify the UCT code to implement RAVE.

GRAVE

- State of the art in General Game Playing (GGP)
- Best AI of the Ludii system (https://ludii.games/)
- Simple modification of RAVE
- Uses statistics both for Black and White at all nodes
- "In principle it is also possible to incorporate the AMAF values, from ancestor subtrees. However, in our experiments, combining ancestor AMAF values did not appear to confer any advantage."

GRAVE

- Use the AMAF statistics of the last ancestor with more than n playouts instead of the AMAF statistics of the current node.

- More accurate when few playouts.

- Published at IJCAI 2015.

- GRAVE is a generalization of RAVE since GRAVE with n=0 is RAVE.

Algorithm 1 The GRAVE algorithm

GRAVE $(board, tref)$
$moves \leftarrow$ possible moves
if $board$ is terminal **then**
    return $score(board)$
end if
$t \leftarrow$ entry of $board$ in the transposition table
if t exists **then**
    if $t.playouts > ref$ **then**
        $tref \leftarrow t$
    end if
    $bestValue \leftarrow -\infty$
    for m in $moves$ **do**
        $w \leftarrow t.wins[m]$
        $p \leftarrow t.playouts[m]$
        $wa \leftarrow tref.winsAMAF[m]$
        $pa \leftarrow tref.playoutsAMAF[m]$
        $\beta_m \leftarrow \frac{pa}{pa+p+bias \times pa \times p}$
        $AMAF \leftarrow \frac{wa}{pa}$
        $mean \leftarrow \frac{w}{p}$
        $value \leftarrow (1.0 - \beta_m) \times mean + \beta_m \times AMAF$
        if $value > bestValue$ **then**
            $bestValue \leftarrow value$
            $bestMove \leftarrow m$
        end if
    end for
    play$(board, bestMove)$
    $res \leftarrow GRAVE(board, tref)$
    update t with res
else
    $t \leftarrow$ new entry of $board$ in the transposition table
    $res \leftarrow playout(player, board)$
    update t with res
end if
return res

GRAVE

- Exercise :

- Modify the RAVE code to implement GRAVE.

Sequential Halving

Sequential Halving, a method so wise

Dividing tasks with great precision and size

Starting from many, it reduces the few

Towards a solution that's both true and true

With each iteration, the choices do narrow

Till the answer shines bright like a beacon so sparrow

No guesses, no chances, no luck needed here

Just a systematic approach, crystal clear

From the simplest problems to the hardest of quest

Sequential Halving never fails to impress

A friend to all seekers, a guide in the night

Bringing order to chaos, and making things right

So let us embrace it, in all we embark

With Sequential Halving, success is just a mark.

Sequential Halving

- Sequential Halving [Karnin & al. 2013] is a bandit algorithm that minimizes the simple regret.

- It has a fixed budget of arm pulls.

- It gives the same number of playouts to all the arms.

- It selects the best half.

- Repeat until only one move is left

Sequential Halving

$S \leftarrow [possibleMoves]$

while $|S| > 1$ **do**

  for each move m in S **do**

    play (m)

    perform $\left\lfloor \dfrac{budget}{|S| \times \lceil log_2(|possibleMoves|) \rceil} \right\rfloor$ playouts

    undo (m)

  end for

  $S \leftarrow$ set of $\left\lceil \dfrac{|S|}{2} \right\rceil$ moves in S with the largest empirical average

end while

SHOT

- SHOT is the acronym for Sequential Halving Applied to Trees [Cazenave 2015].

- When the search comes back to a node it considers the spent budget and the new budget as a whole.

- It distributes the overall budget with Sequential Halving.

SHOT
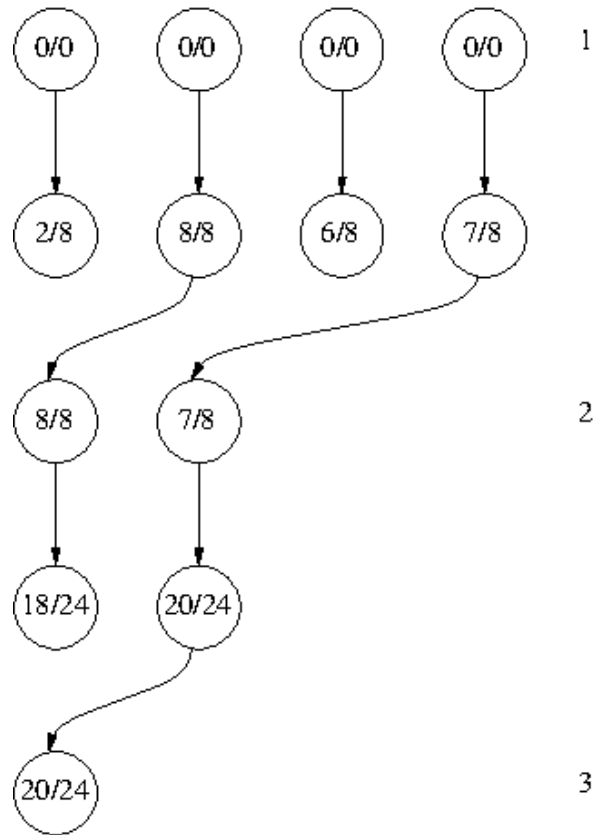
SHOT
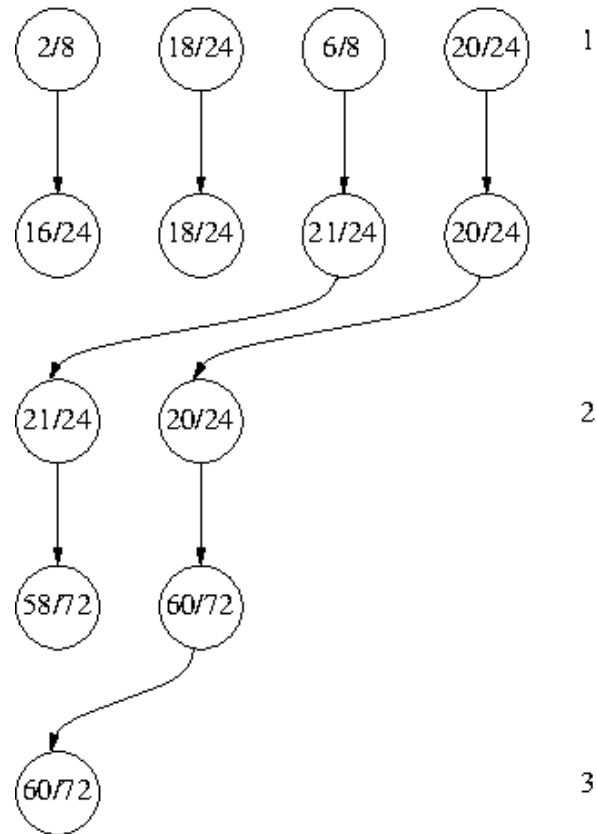
SHOT

- SHOT gives good results for Nogo.

- Combining SHOT and UCT :

  SHOT near the root
  UCT deeper in the tree

- The combination gives good results for Atarigo, Breakthrough, Amazons and partially observable games.

Sequential Halving

- Exercise:

- Write the code to perform Sequential Halving at the root on top of UCT.

SHUSS

Sequential Halving Using Scores,
A method to find the best of many,
It starts with many choices,
And narrows them down, through many voices.

It divides the options in groups,
And test them with different scores,
Eliminating the ones that lag,
Until the best one, it ensures.

This method is efficient and fast,
It saves time and resources,
And finds the best solution, at last,
Among many possible courses.

Sequential Halving Using Scores,
A powerful tool for decision,
It helps us to find the right doors,
And make the best decision.

SHUSS

- Sequential Halving combined with other statistics such as AMAF statistics.

- Instead of selecting the best half with the mean (mu_i), use:

$$mu_i + c * AMAF_i / p_i$$

with p_i the number of playouts of move i and $c \geq 128$.

- Combining SH with AMAF = SHUSS (Sequential Halving Using Scores) [Fabiano et al. 2021]

SHUSS

Table 1: Comparison of Hybrid-SHUSS with AMAF score against RAVE.

Game	Playouts	0	128	256	512	1024	2048	4096	8192	16384	∞
Atarigo 7x7	10 000	44.2	47.2	49.6	**50.2**	50.0	49.6	45.2	47.8	46.4	45.2
Atarigo 9x9	10 000	35.6	41.4	40.0	38.2	41.0	41.2	**43.4**	41.4	36.4	40.2
Ataxx 8x8	10 000	30.2	33.6	35.2	34.2	42.0	46.2	55.0	62.4	62.0	**71.8**
Breakthrough 8x8	10 000	54.0	**57.8**	56.8	56.0	56.6	55.2	53.8	51.0	55.0	52.4
Domineering 8x8	10 000	41.4	47.8	44.8	**49.0**	46.2	47.2	46.2	45.6	43.0	42.4
Go 7x7	10 000	45.2	49.2	46.2	53.8	**58.6**	50.2	42.6	33.2	31.0	15.8
Go 9x9	10 000	43.4	53.2	**58.2**	52.2	50.8	43.8	35.6	26.4	19.0	12.2
Hex 11x11	10 000	15.8	43.0	43.4	**51.4**	48.4	50.2	46.4	46.6	43.4	42.6
Knightthrough 8x8	10 000	61.0	61.6	**65.0**	63.8	62.2	60.2	54.2	54.4	56.2	52.8
NoAtaxx 8x8	10 000	**91.0**	87.4	76.8	72.0	62.8	55.2	53.8	44.6	45.8	43.2
NoBreakthrough 8x8	10 000	37.8	40.8	44.0	46.2	**51.4**	44.2	46.4	44.0	50.0	46.6
NoDomineering 8x8	10 000	40.4	45.6	49.4	46.0	48.4	**50.0**	47.6	47.4	45.0	47.6
NoGo 7x7	10 000	38.8	40.8	45.6	44.0	50.8	47.6	50.8	49.4	47.6	**51.8**
NoGo 9x9	10 000	30.0	37.8	38.8	40.0	41.0	42.0	42.8	45.0	**45.8**	37.4
NoHex 11x11	10 000	46.4	48.0	48.6	49.0	**49.2**	48.6	48.6	49.2	48.8	49.2
NoKnightthrough 8x8	10 000	29.0	36.8	38.8	39.6	47.8	46.2	46.0	45.2	**48.2**	47.6

SHUSS

Algorithm 2 Sequential Halving USing Scores (SHUSS)

Parameter: cutting ratio λ, $\tilde{t}'$

Input: total budget T, set of arms S, online scores $\tilde{X}_r^{(i)}$

$S_0 \leftarrow S, T_0 \leftarrow T$

$R \leftarrow$ number of rounds before $|S_R| = 1$

for $r = 0$ **to** $R - 1$ **do**

$\quad t_r \leftarrow \lfloor \frac{T_r}{|S_r| \cdot (R-r)} \rfloor$

$\quad T_{r+1} \leftarrow T_r - t_r |S_r|$

$\quad$ sample each arm in S_r t_r times, giving an empirical mean $p_r^{(i)}$ to arm i out of t_r^+ trials

$\quad q_r^{(i)} = p_r^{(i)} + \frac{\tilde{t}'}{t_r^+} \tilde{X}_r^{(i)}$

$\quad S_{r+1} \leftarrow S_r$ without the fraction $1 - \lambda$ of the worst arms in terms of $q_r^{(i)}$

end for

Output: arm in S_R

SHUSS

- Exercise:

  Write the code to perform SHUSS at the root on top of GRAVE.

PUCT

- MCTS used in AlphaGo and AlphaZero.
- A neural network gives a policy and a value.
- No playouts, evaluation with the value at the leaves.
- P(s,a) = probability for move a of being the best.
- Bandit for the tree descent:

$$U(s, a) = c_{\text{puct}} P(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)}$$

Algorithm 1 The PUCT algorithm.

1: PUCT (*board*, *player*)
2:     *moves* ← possible moves on *board*
3:     **if** *board* is terminal **then**
4:         **return** evaluation (*board*)
5:     **end if**
6:     *t* ← entry of *board* in the transposition table
7:     **if** *t* exists **then**
8:         *bestValue* ← −∞
9:         **for** *m* in *moves* **do**
10:           *t* ← *t.totalPlayouts*
11:           *mean* ← *t.mean*[*m*]
12:           *p* ← *t.playouts*[*m*]
13:           *prior* ← *t.prior*[*m*]
14:           *value* ← *mean* + *c* × *prior* × $\frac{\sqrt{t}}{p}$
15:           **if** *value* > *bestValue* **then**
16:             *bestValue* ← *value*
17:             *bestMove* ← *m*
18:           **end if**
19:         **end for**
20:         play (*board*, *bestMove*)
21:         *player* ← opponent (*player*)
22:         *res* ← PUCT (*board*, *player*)
23:         update *t* with *res*
24:     **else**
25:         *t* ← new entry of *board* in the transposition table
26:         *res* ← evaluation (*board*, *player*)
27:         update *t*
28:     **end if**
29:     **return** *res*

PUCT

- Exercise :

  Modify the UCT code into PUCT.

  Suppose a random policy and a random value.

Nested Monte Carlo Search

Nested Monte Carlo Search

Nested Monte Carlo Search, a complex game,
A method to find the best move, it can claim,
It looks deeper, it goes beyond,
To find the winning move, it has fond.

It takes the Monte Carlo Tree Search,
And adds another layer, to research,
It explores the branches, with great care,
To find the best outcome, with much flair.

It simulates the game, again and again,
And analyzes the data, to win.
It's like a Russian doll, inside and out,
Nested Monte Carlo Search, without a doubt.

It's a powerful tool, for AI,
To make machines better, that's its aim high,
It's a step towards true intelligence,
Nested Monte Carlo Search, a true excellence.

Single Agent Monte Carlo

- UCT can be used for single-agent problems.
- Nested Monte Carlo Search often gives better results.
- Nested Rollout Policy Adaptation is an online learning variation that has beaten world records.

Nested Monte-Carlo Search
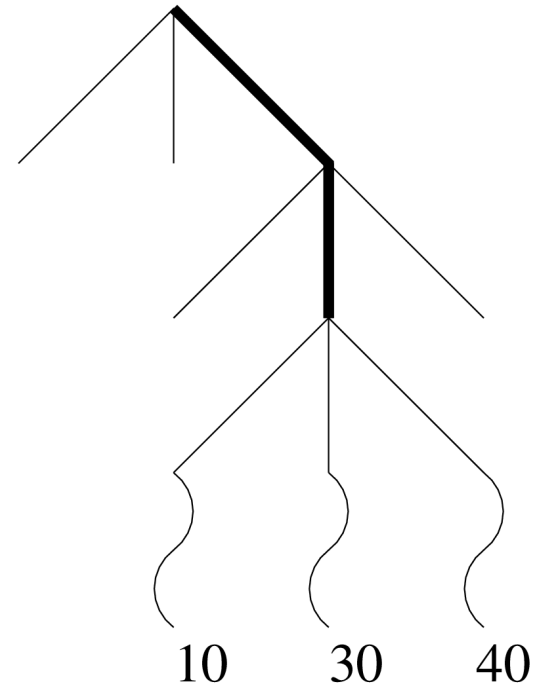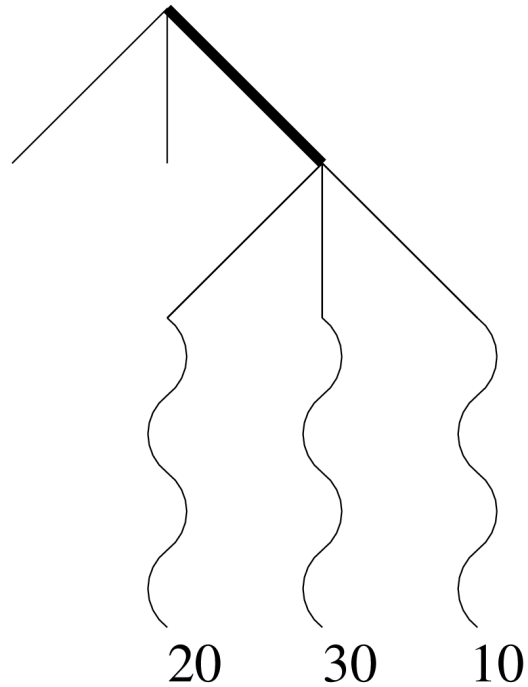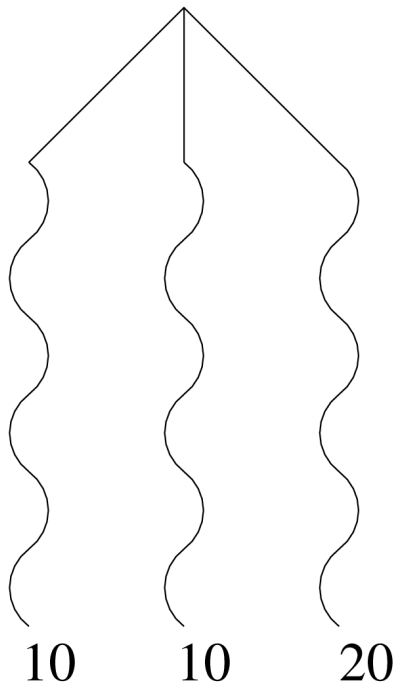
Nested Monte-Carlo Search

- Play random games at level 0
- For each move at level n+1, play the move then play a game at level n
- Choose to play the move with the greatest associated score
- Important : memorize and follow the best sequence found at each level

Algorithm 4 The NMCS algorithm.

NMCS ($state$, $level$)

if level == 0 **then**

    return playout ($state$, $uniform$)

end if

$BestSequenceOfLevel \leftarrow \emptyset$

while $state$ is not terminal **do**

    for m in possible moves for $state$ **do**

        $s \leftarrow$ play ($state$, m)

        NMCS (s, $level - 1$)

        update $BestSequenceOfLevel$

    end for

    $bestMove \leftarrow$ move of the $BestSequenceOfLevel$

    $state \leftarrow$ play ($state$, $bestMove$)

end while

Analysis

- Analysis on two very simple abstract problems.

- Search tree = binary tree.

- In each state there are only two possible moves: going to the left or going to the right.

Analysis

- The scoring function of the leftmost path problem consists in counting the number of moves on the leftmost path of the tree.

3   2 1   1 0   0 0   0
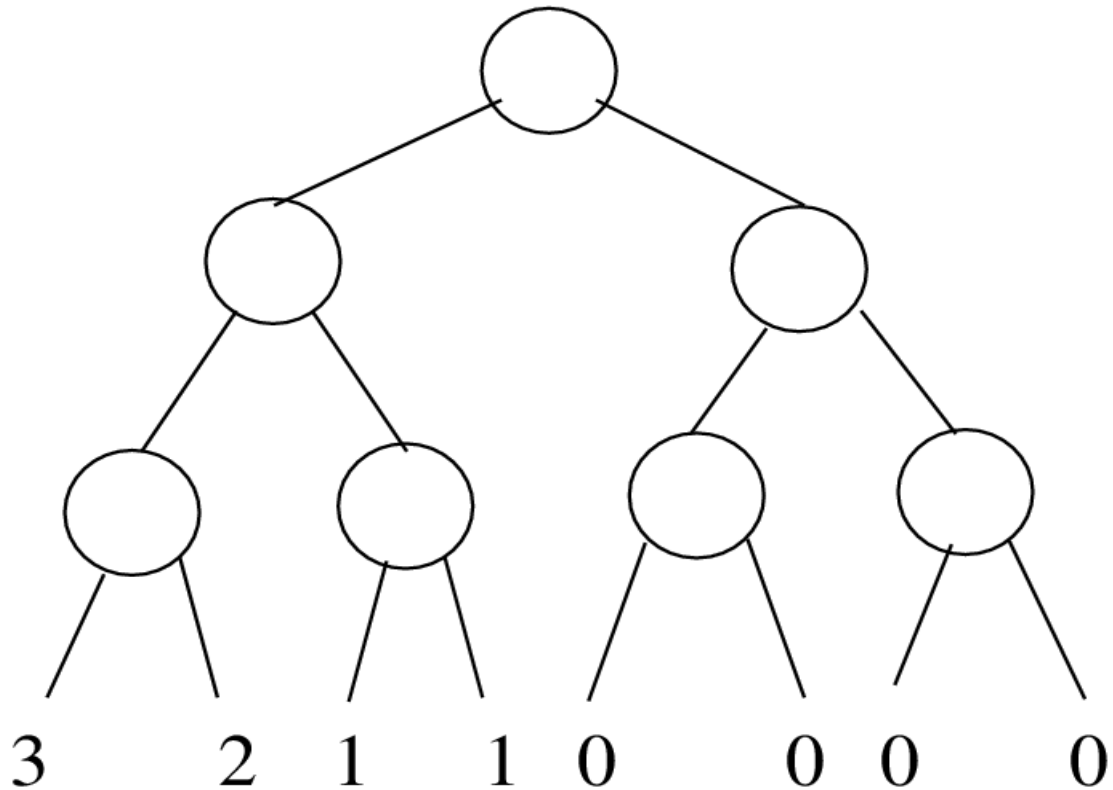
Analysis

- Sample search : probability 2^{-n} of finding the best score of a depth n problem.

- Depth-first search : one chance out of two of choosing the wrong move at the root, so the mean complexity $> 2^{n-2}$.

- A level 1 Nested Monte-Carlo Search will always find the best score, complexity is n(n-1).

- Nested Monte-Carlo Search is appropriate for the leftmost path problem because the scores at the leaves are extremely correlated with the structure of the search tree.

Analysis

- The scoring function of the left move problem consists in counting the number of moves on the left.

3   2 2 1 2   1 1   0
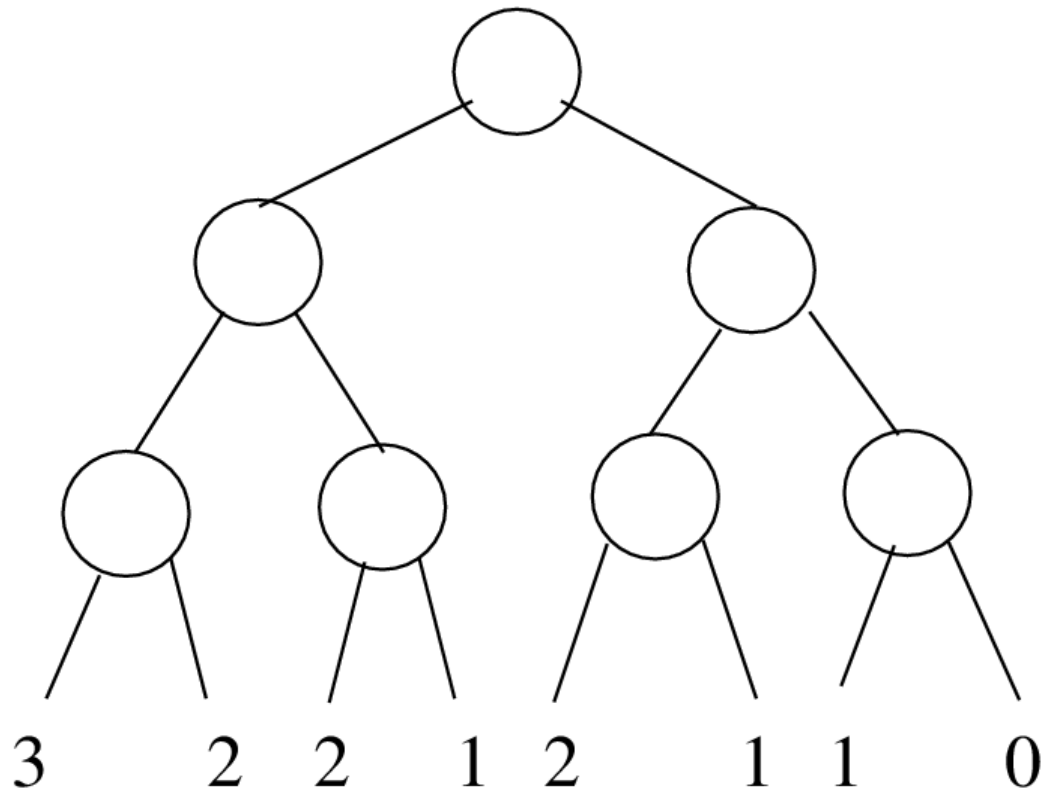
Analysis

- The probability distribution can be computed exactly with a recursive formula and dynamic programming.

- A program that plays the left move problems has also been written and results with 100,000 runs are within 1% of the exact probability distribution.
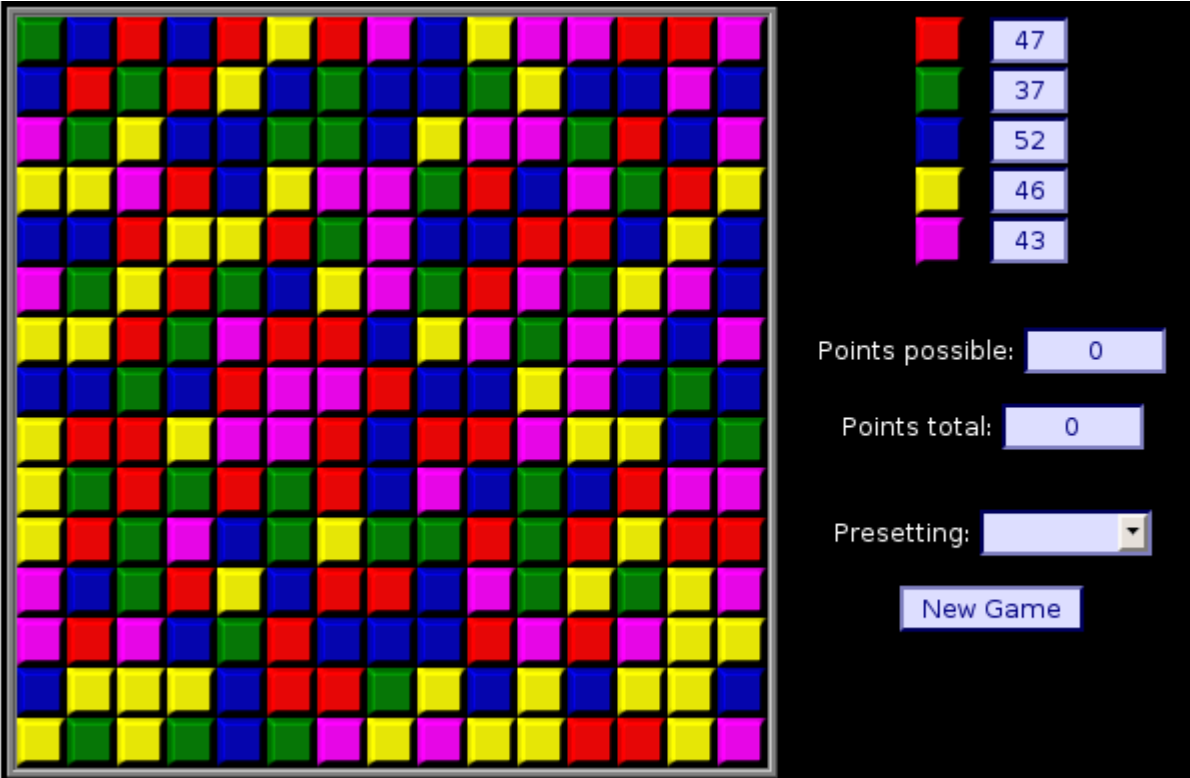
Analysis

- Distributions of the scores for a depth 60 problem.
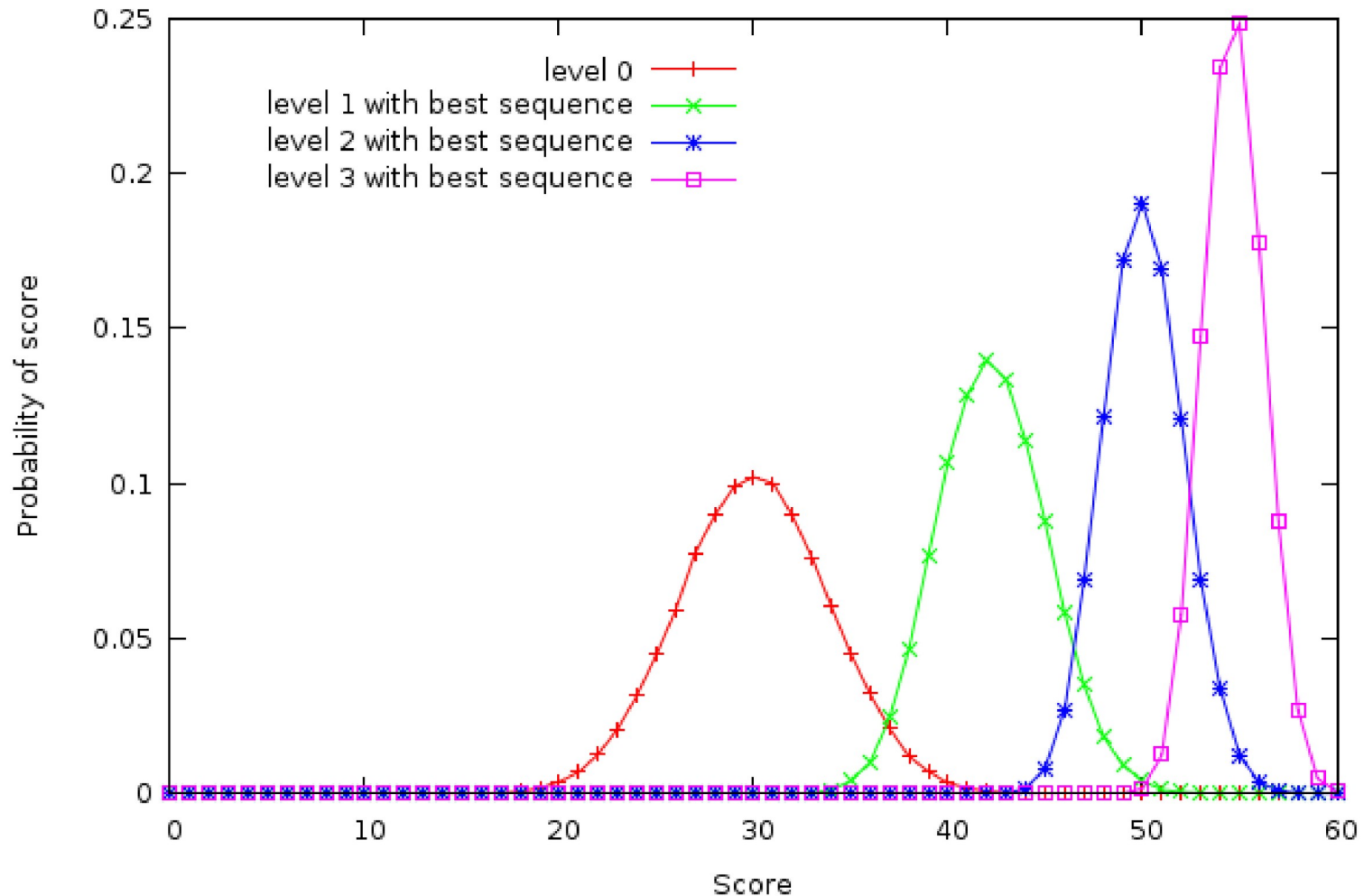
Analysis

- Mean score in real time
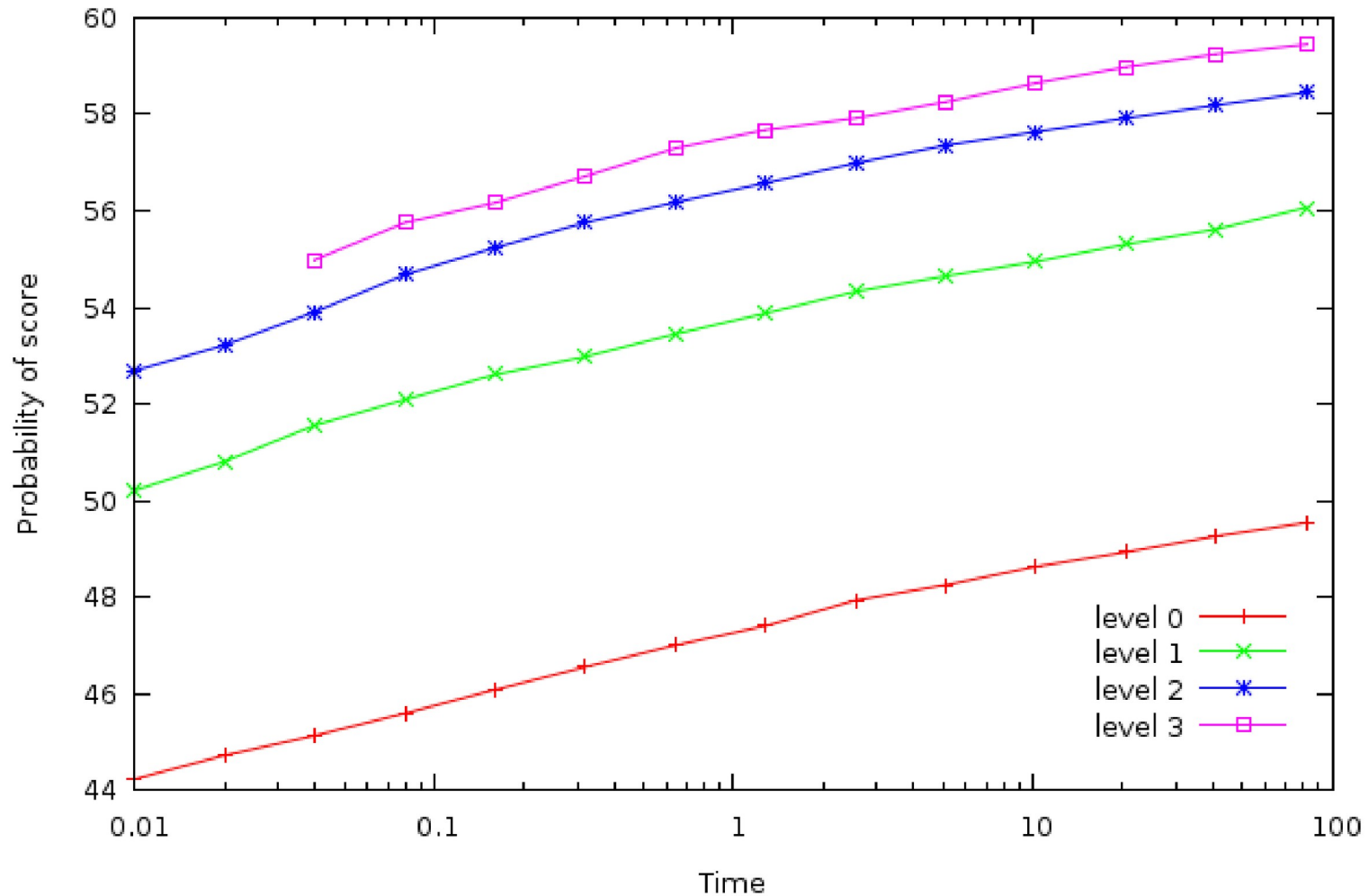
Morpion Solitaire

- Morpion Solitaire is an NP-hard puzzle and the high score is inapproximable within $n^{1-epsilon}$

- A move consists in adding a circle such that a line containing five circles can be drawn.

- In the disjoint version a circle cannot be a part of two lines that have the same direction.

- Best human score is 68 moves.

- Level 4 Search => 80 moves, after 5 hours of computation on a 64 cores cluster.

Morpion Solitaire

- 80 moves :
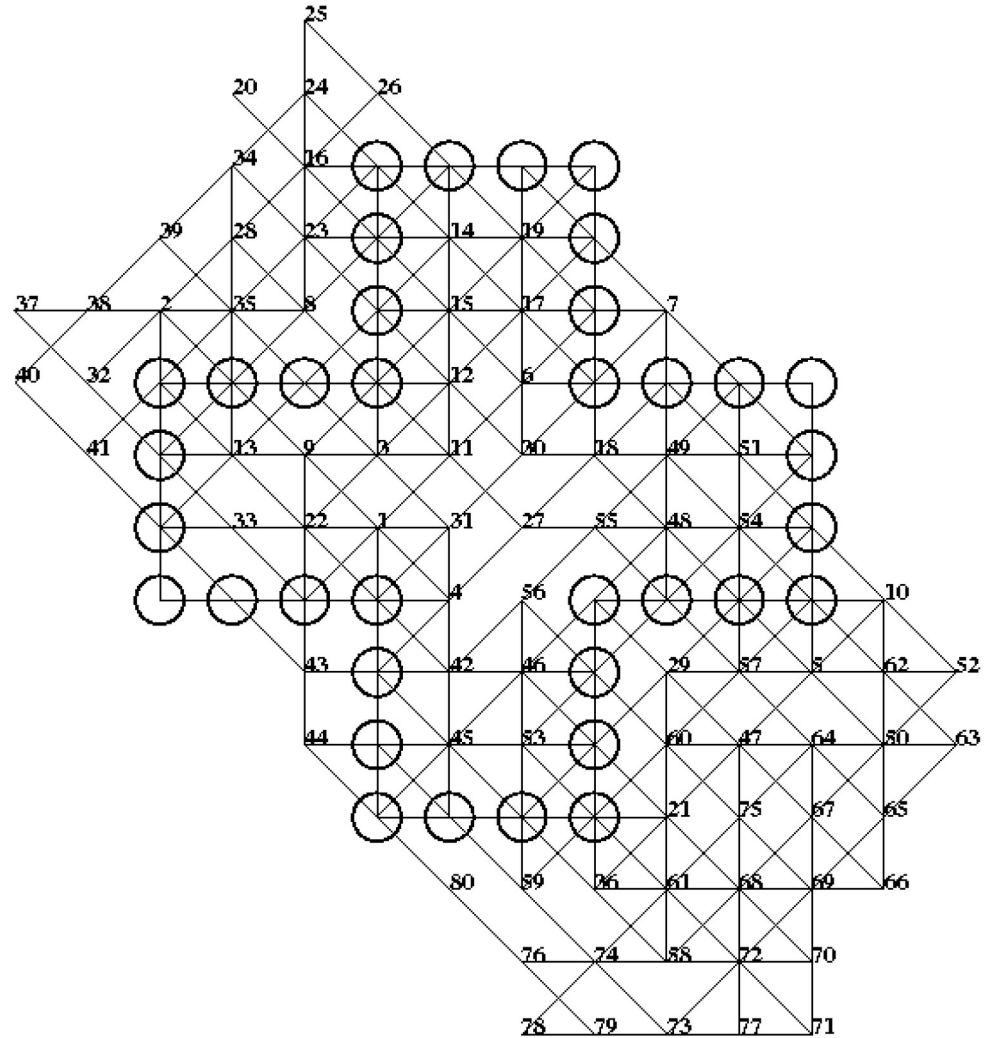
Morpion Solitaire

- Distribution of the scores
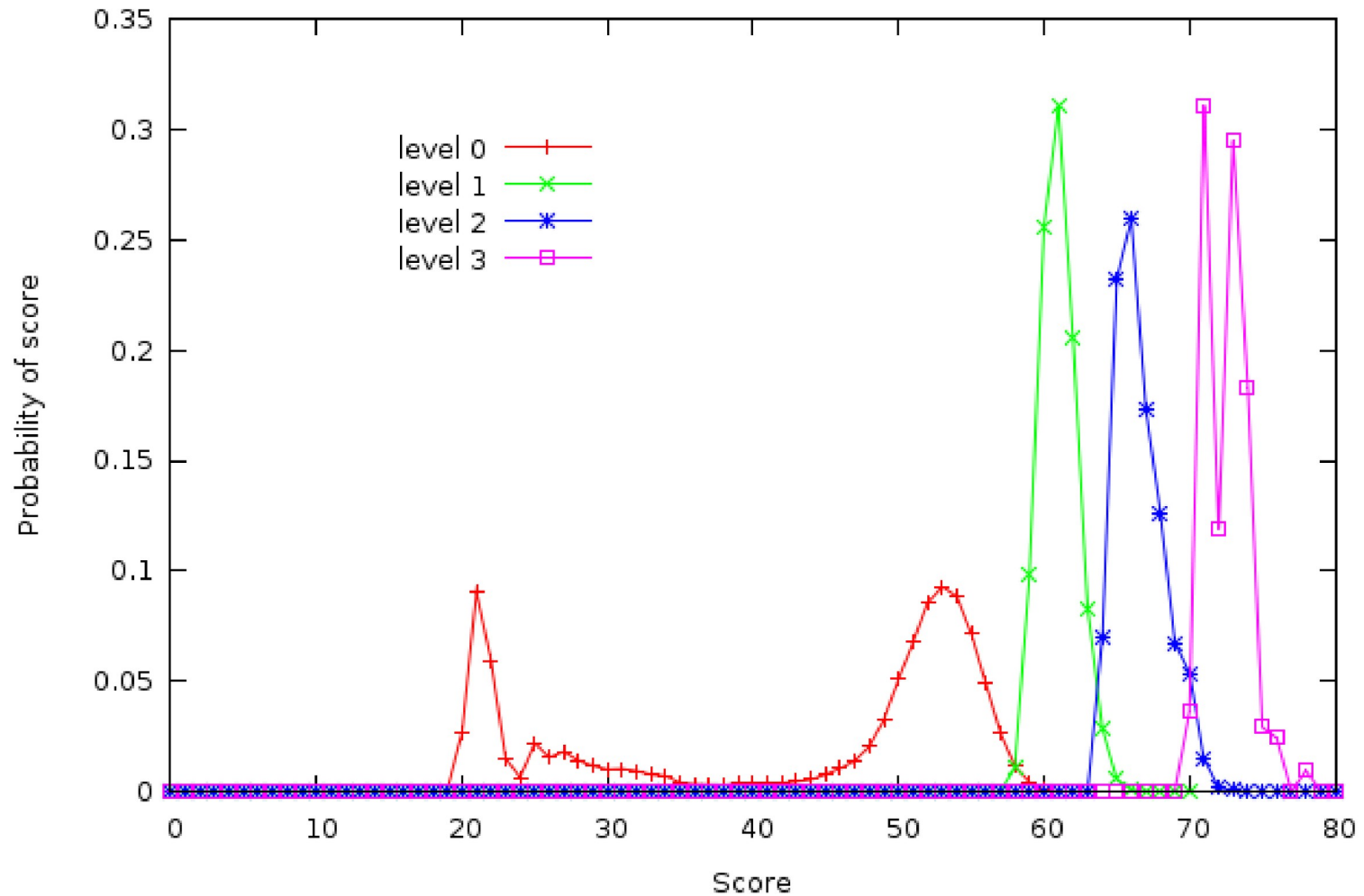
Morpion Solitaire

- Mean scores in real-time
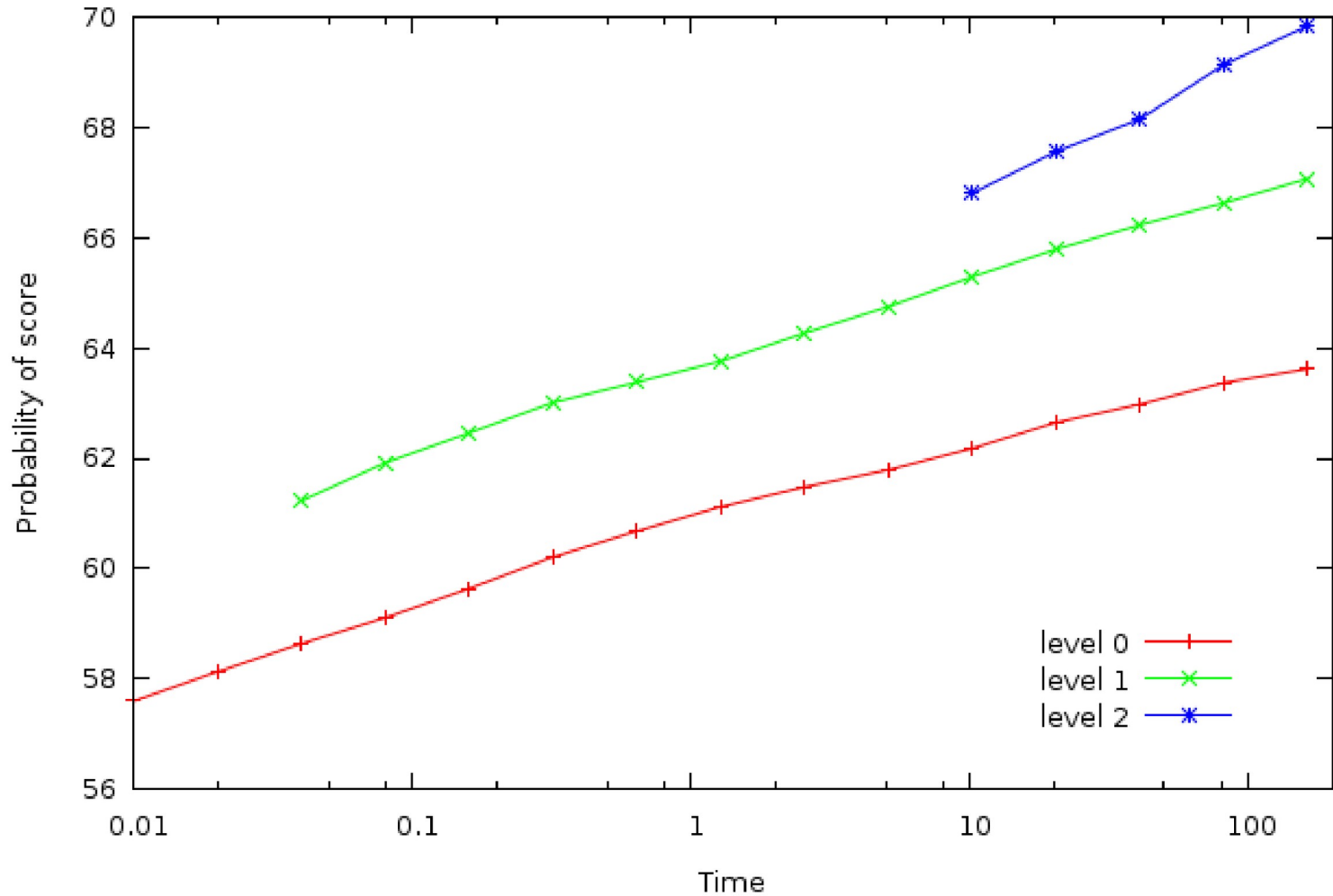
SameGame

- NP-complete puzzle.

- It consists in a grid composed of cells of different colors. Adjacent cells of the same color can be removed together, there is a bonus of 1,000 points for removing all the cells.

- TabuColorRandom strategy: the color that has the most cells is set as the tabu color.

- During the playouts, moves of the tabu color are played only if there are no moves of the others colors or it removes all the cells of the tabu color.

Same Game

Same Game

- SP-MCTS = restarts of the UCT algorithm
- SP-MCTS scored 73,998 on a standard test set.
- IDA* : 22,354
- Darse Billings program : 72,816.
- Level 2 without memorization : 44,731
- Nested level 2 with memorization : 65,937
- Nested level 3 : 77,934

Application to Constraint Satisfaction

- A nested search of level 0 is a playout.
- A nested search of level 1 uses a playout to choose a value.
- A nested search of level 2 uses nested search of level 1 to choose a value.
- etc.
- The score is always the number of free variables.

Sudoku

- Sudoku is a popular NP-complete puzzle.
- 16x16 grids with 66% of empty cells.
- Easy-Hard-Easy distribution of problems.
- Forward Checking (FC) is stopped when the search time for a problem exceeds 20,000 s.

Sudoku

- FC :   > 446,771.09 s.
- Iterative Sampling :   61.83 s.
- Nested level 1 :   1.34 s.
- Nested level 2 :   1.64 s.

Kakuro

	24	25	20	26	24
18	.	.	.	.	.
26	.	.	.	.	.
28	.	.	.	.	.
26	.	.	.	.	.
21	.	.	.	.	.

A 5x5 grid

Kakuro

	24	25	20	26	24
18	1	7	5	3	2
26	4	5	3	8	6
28	5	6	7	2	8
26	8	4	1	6	7
21	6	3	4	7	1

Solution

Kakuro

Algorithme	Solved problems	Time
Forward Checking	8/100	92,131.18 s.
Iterative Sampling	10/100	94,605.16 s.
Monte-Carlo level 1	100/100	78.30 s.
Monte-Carlo level 2	100/100	17.85 s.

8x8 Grids, 9 values, stop at 1,000 s.

Bus Regulation

- Goal : minimize passengers waiting times by making buses wait at a stop.

- Evaluation of an algorithm : sum of the waiting times for all passengers.

Regulation Algorithms

- Rule-based regulation: The waiting time depends on the number of stop with the next bus

- Monte-Carlo regulation : Choose the waiting time that has the best mean of random playouts

- Nested Monte-Carlo regulation : Use multiple levels of playouts
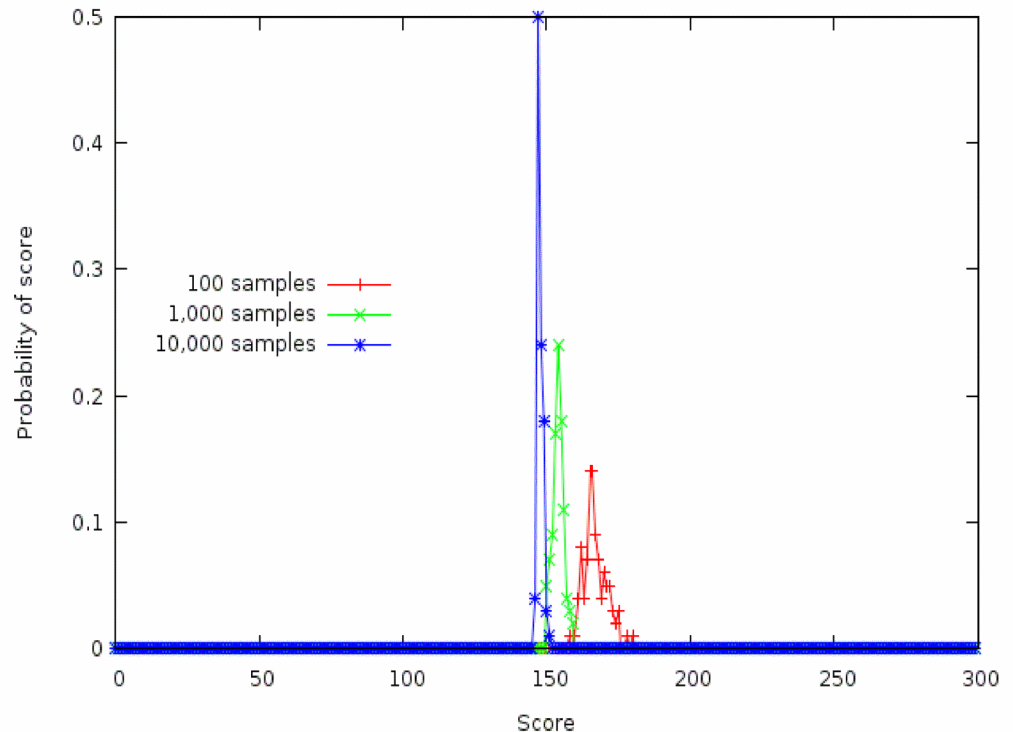
Rule-based regulation

- δ : number of stop before the next bus.
- w : waiting time if the next bus is at more than δ.
- No regulation : 171
- Wait during 4 if more than 7 stops : 164

TABLE I

SCORES FOR DIFFERENT RULES

	w=1	w=2	w=3	w=4
$\delta = 4$	171	192	193	199
$\delta = 5$	171	191	192	195
$\delta = 6$	171	175	176	198
$\delta = 7$	171	169	166	**164**
$\delta = 8$	171	169	167	165
$\delta = 9$	171	169	167	166
$\delta = 10$	171	170	170	170

Monte-Carlo Regulation

- 165 for N = 100
- 154 for N = 1000
- 147 for N = 10000

better than rule-based regulation (164).

Parallel Nested Monte-Carlo Search

- Play the highest level sequentially
- Play the lowest levels in parallel
- Speedup = 56 for 64 cores at Morpion Solitaire
- A more simple parallelization : play completely different searches in parallel (i.e. use a different seed for each search).
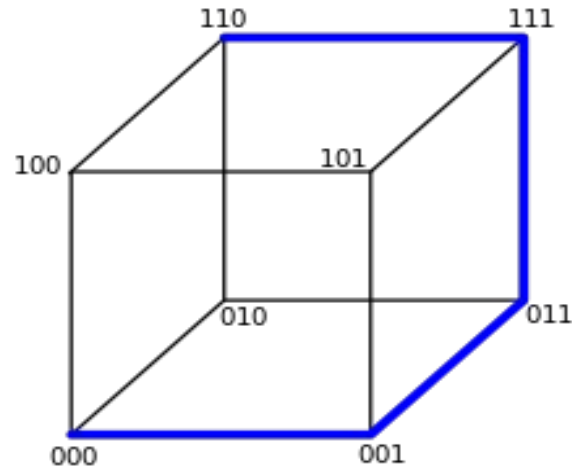
Monte Carlo Beam Search
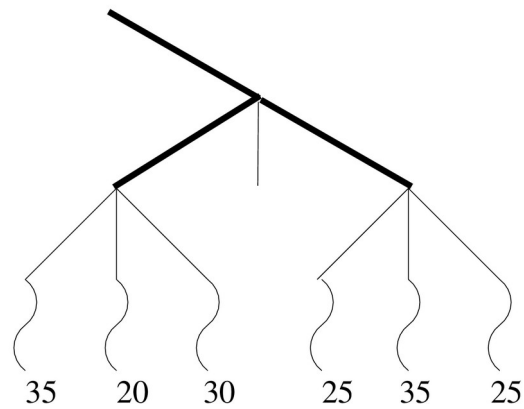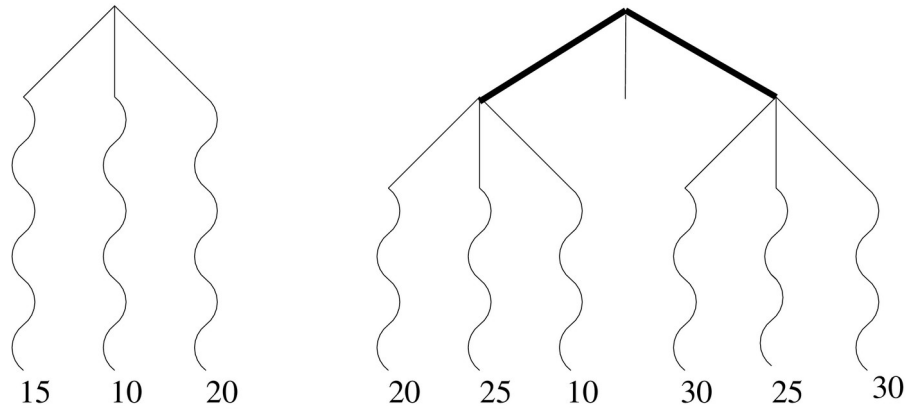
Single-Agent General Game Playing

- Nested Monte-Carlo search gives better results than UCT on average.

- For some problems UCT is better.

- Ary searches with both UCT and Nested Monte-Carlo search and plays the move that has the best score.

Snake in the box

- A path such that for every node only two neighbors are in the path.

- Applications: Electrical engineering, coding theory, computer network topologies.

- World records with NMCS [Kinny 2012].

Multi-agent pathfinding

- Find routes for the agents avoiding collisions.

- Monte Carlo Fork Search enables to branch in the playouts.

- It solves difficult problems faster than other algorithms [Bouzy 2013].

The Pancake Problem
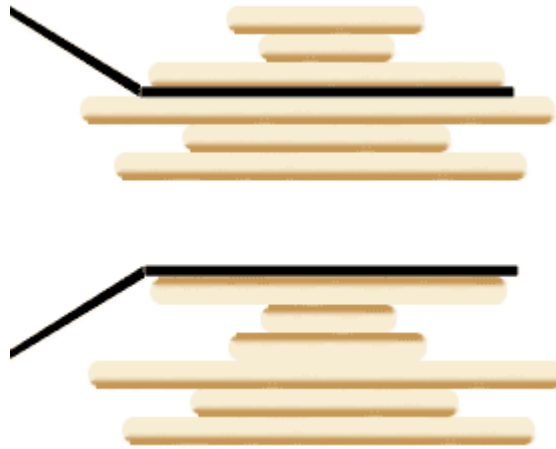
- Nested Monte Carlo Search has beaten world records using specialized playout policies [Bouzy 2015].

Software Engineering

- Search based software testing [Feldt and Poulding 2015].

- Heuristic Model Checking [Poulding and Feldt 2015].

- Generating structured test data with specific properties [Poulding and Feldt 2014].

RNA Inverse Folding

- Molecule Design as a Search Problem
- Find the sequence of nucleotides that gives a predefined structure.
- A biochimist applied Nested Monte Carlo Search to this problem [Portela 2018].
- Better than the state of the art.

Eterna100 Benchmark

- Molecule Design as Puzzles:
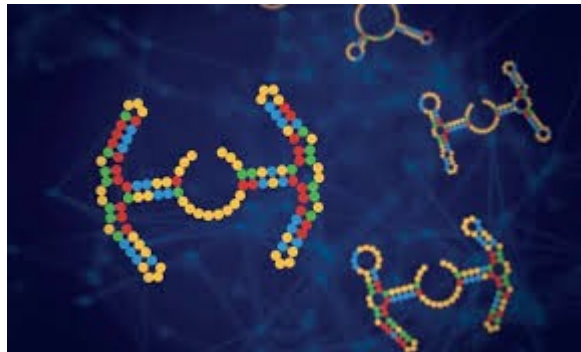
Applications

Nested Monte Carlo Search :

- Morpion Solitaire [Cazenave 2009]
- SameGame [Cazenave 2009]
- Sudoku [Cazenave 2009]
- Expression Discovery [Cazenave 2010]
- The Snake in the Box [Kinny 2012]
- Cooperative Pathfinding [Bouzy 2013]
- Software Testing [Poulding et al. 2014]
- Heuristic Model-Checking [Poulding et al. 2015]
- Pancake problem [Bouzy 2015]
- Games [Cazenave et al. 2016]
- Cryptography [Dwivedi et al. 2018]
- RNA inverse folding problem [Portela 2019]
- …

Exercise

- Write a Nested Monte Carlo Search for the left move problem.
- Functions to write :

  legalMoves (state)

  play (state, move)

  terminal (state)

  score (state)

  playout (state)

- Then write a Nested Monte Carlo Search using these functions.

Monte-Carlo Discovery of Expressions
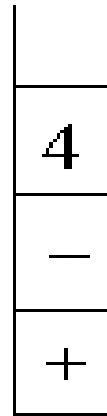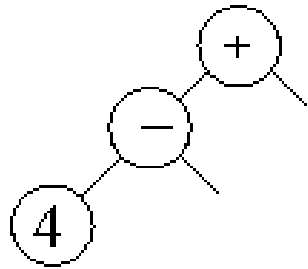
- Possible moves are pushing atoms.

- Evaluation of a complete expression.

- Better than Genetic Programming for some problems [Cazenave 2010, 2013].

Monte-Carlo Discovery of Expressions
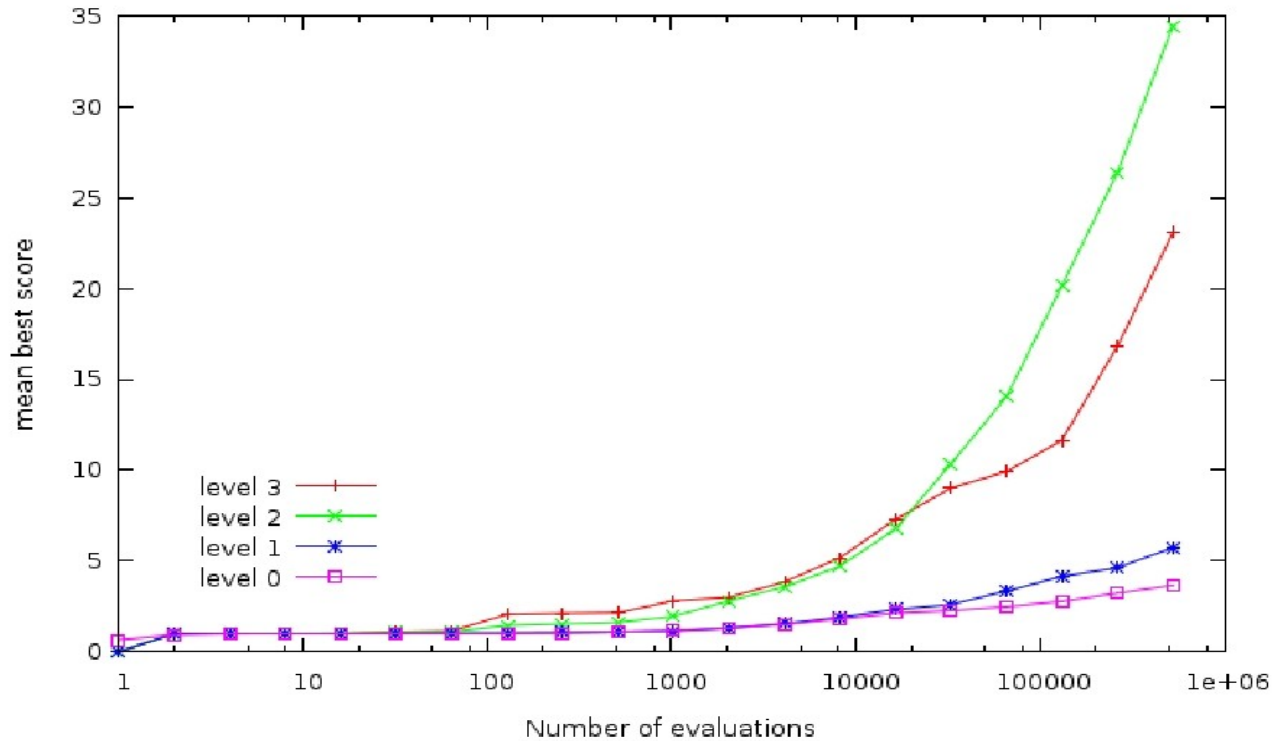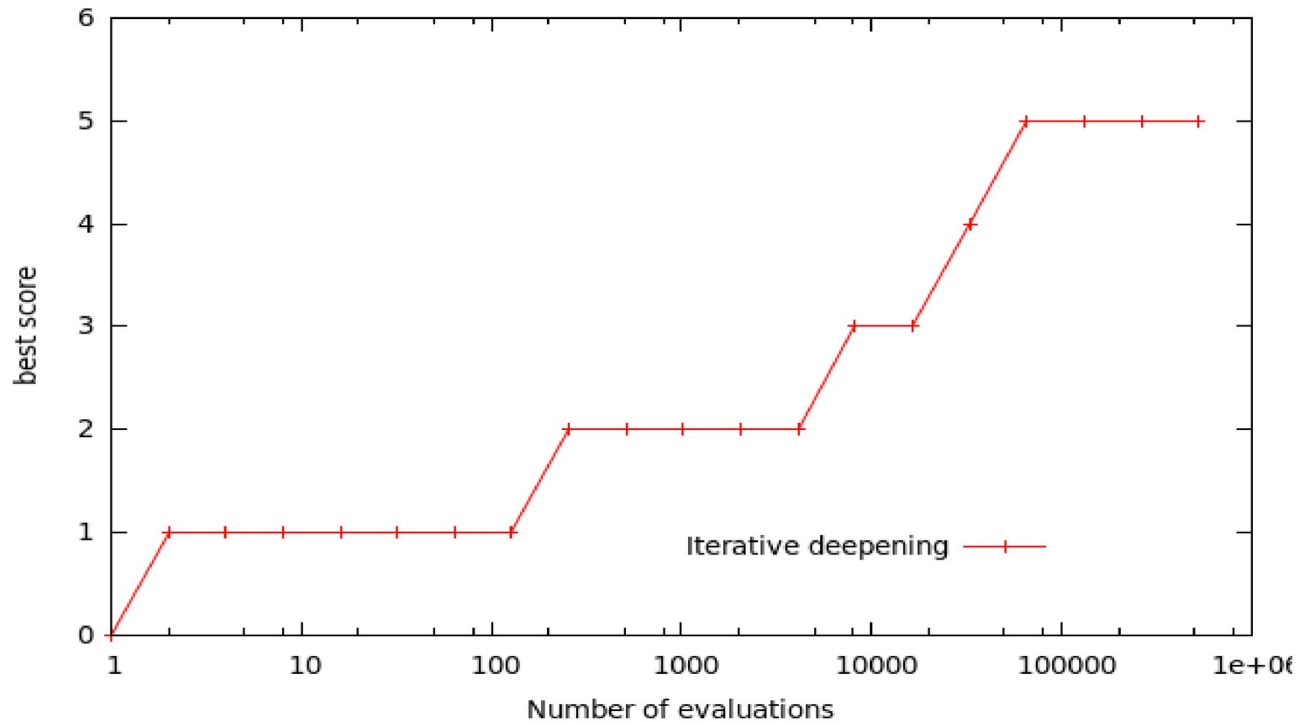
Prime Generating Polynomials:

The score of an expression is the number of different primes it generates in a row for integer values of x starting at zero and increasing by one at each step.

Nested Monte-Carlo search is better than UCT and Iterative Deepening search.

Monte-Carlo Discovery of Expressions

Monte-Carlo Discovery of Expressions

Monte-Carlo Discovery of Expressions

Monte-Carlo Discovery of Expressions

- N prisoners are assigned with either a 0 or a 1.
- A prisoner can see the number assigned to the other prisoners but cannot see his own number.
- Each prisoner is asked independently to guess if he is 0 or 1 or to pass.
- The prisoners can formulate a strategy before beginning the game.
- All the prisoners are free if at least one guesses correctly and none guess incorrectly.
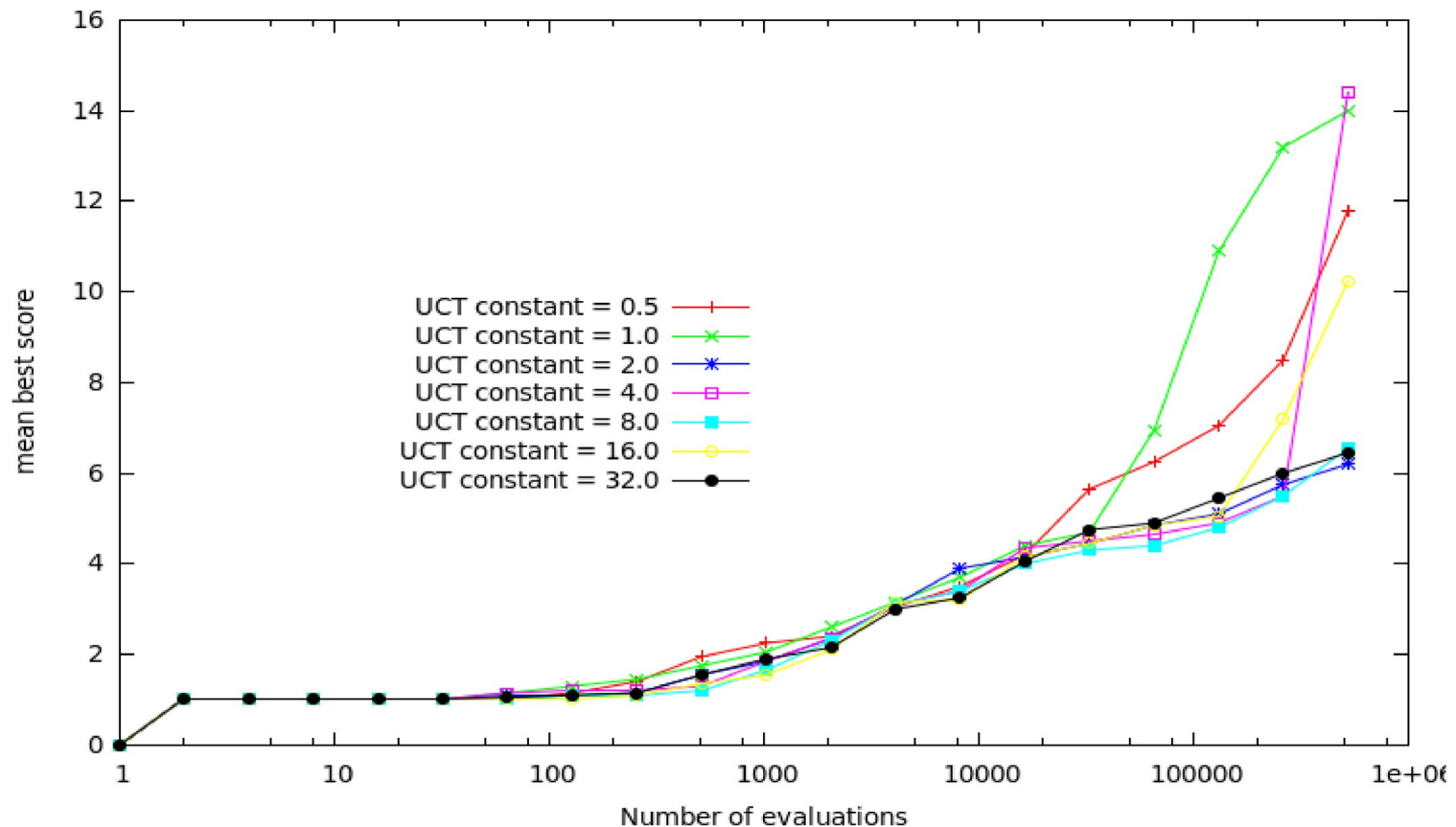- A possible strategy is for example that one of the prisoners says 1 and the others pass, this strategy has fifty percent chances of winning.

Monte-Carlo Discovery of Expressions
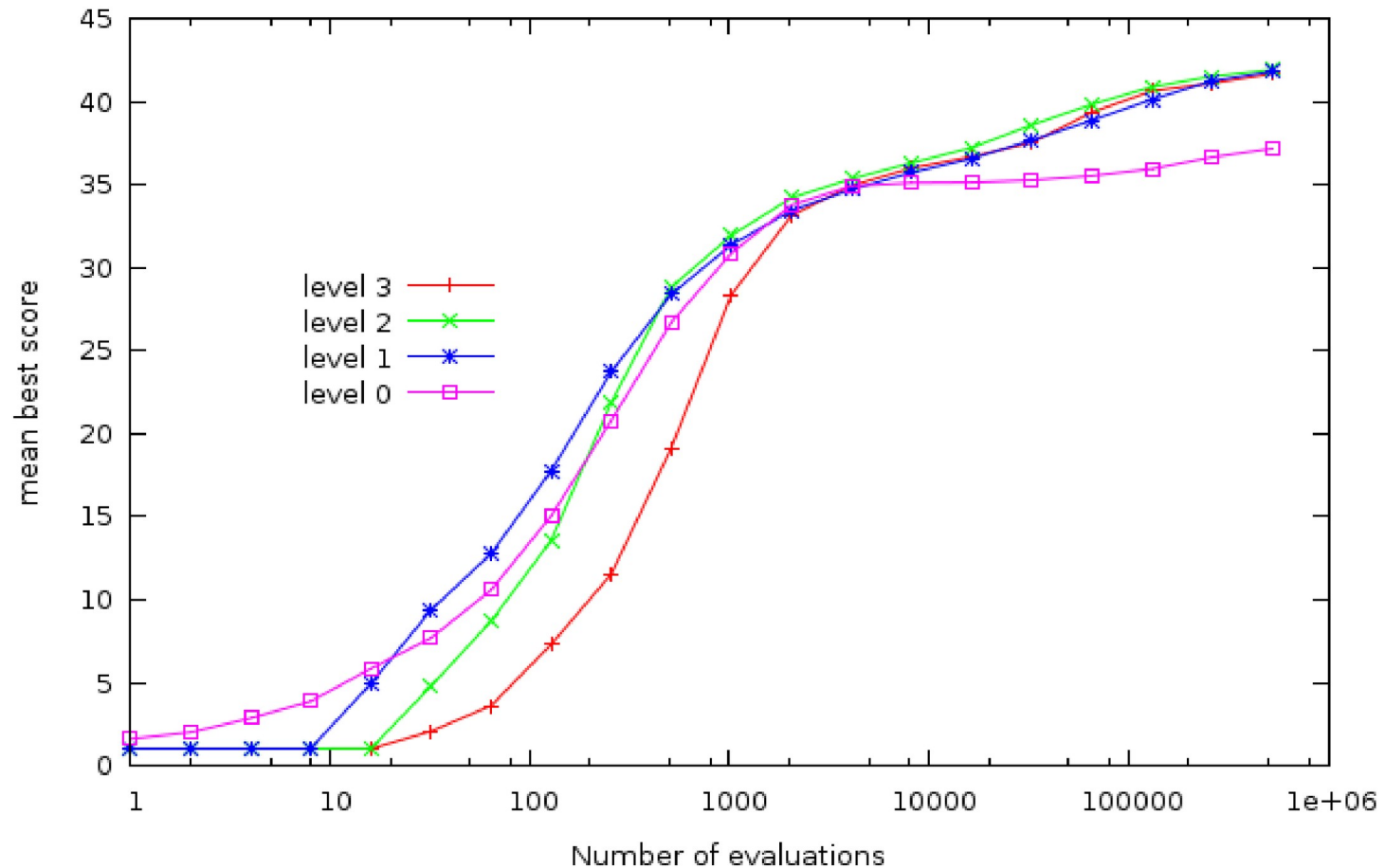
Monte-Carlo Discovery of Expressions
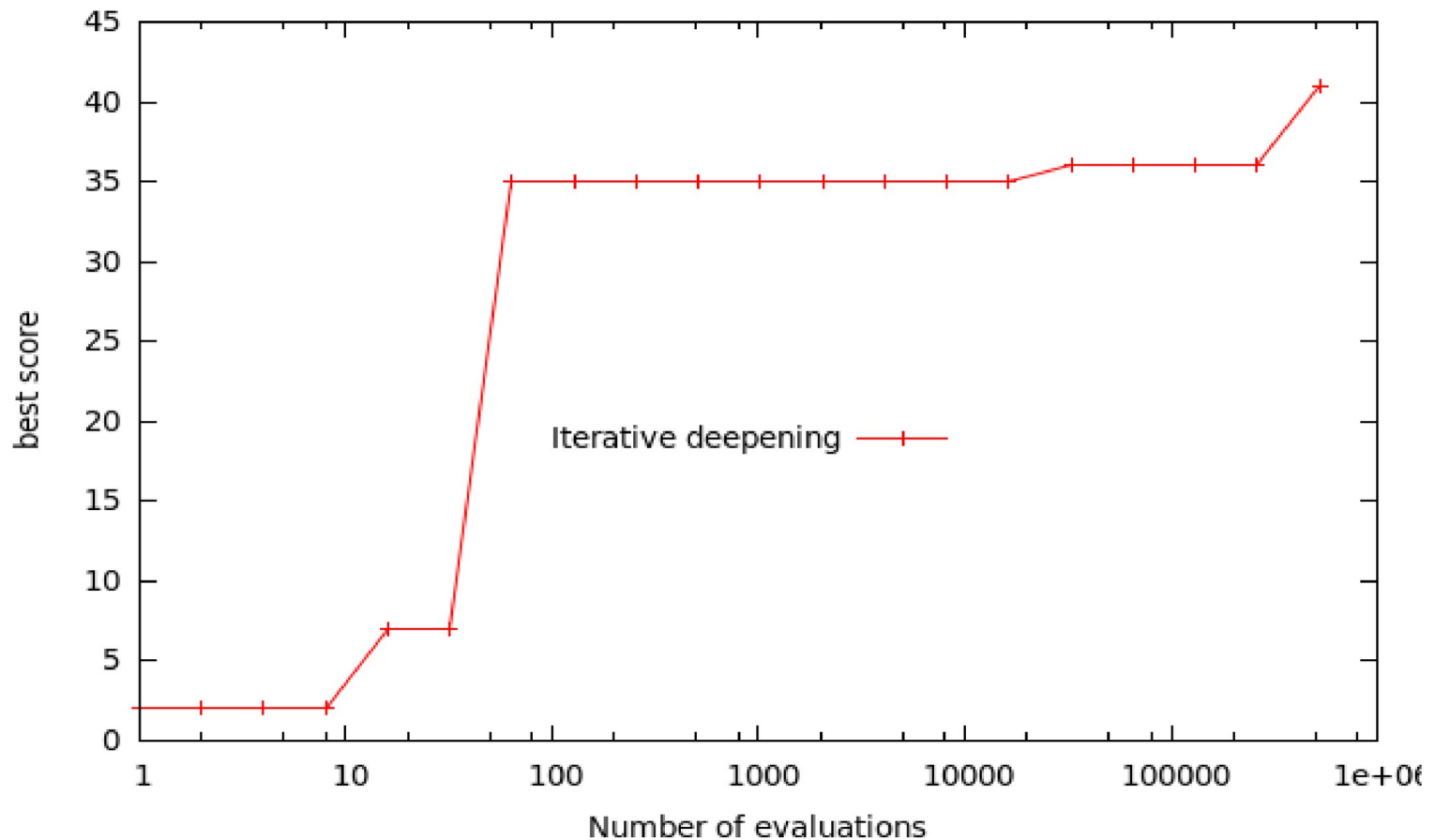
Monte-Carlo Discovery of Expressions
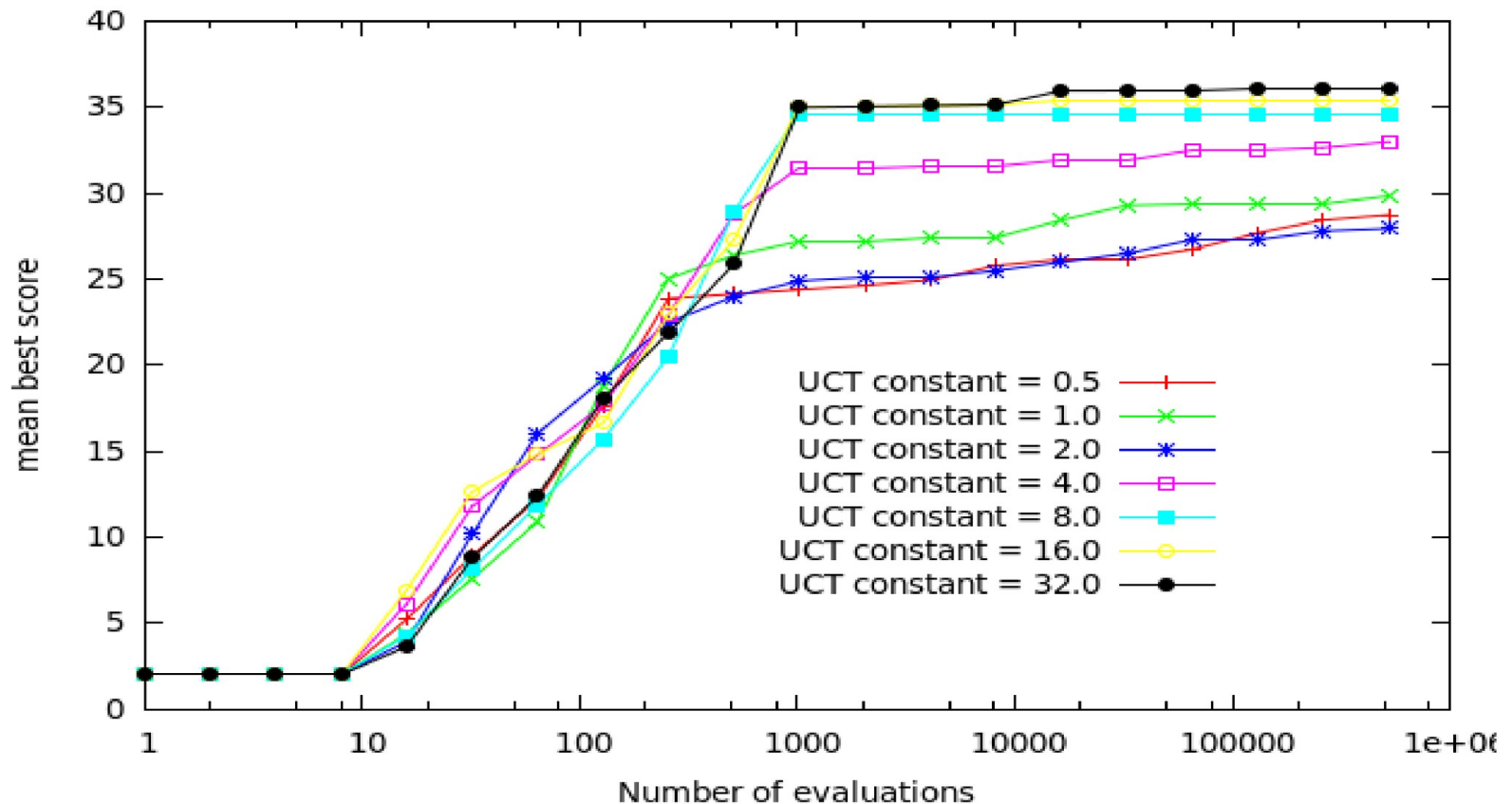
Application to financial data

- Data used to perform our empirical analysis are daily prices of European S&P500 index call options.

- The sample period is from January 02, 2003 to August 29, 2003.

- S&P500 index options are among the most actively traded financial derivatives in the world.

Atom Set

+   Addition

-       Subtraction

*   Multiplication

%   Protected Division

ln   Protected Natural Log

Exp   Exponential function

Sqrt   Protected Square Root

cos   Cosinus

sin   Sinus

Ncfd   Normal cumulative distribution

C/K Call Price/Strike Price

S/K Index Price/Strike Price

tau Time to Maturity

Fitness function

- Each formula found by NMCS or GP is evaluated to test whether it can accurately forecast the implied volatility for all entries in the training set.

- Fitness = Mean Squared Error (MSE) between the estimated volatility and the target volatility.

Mean Square Error
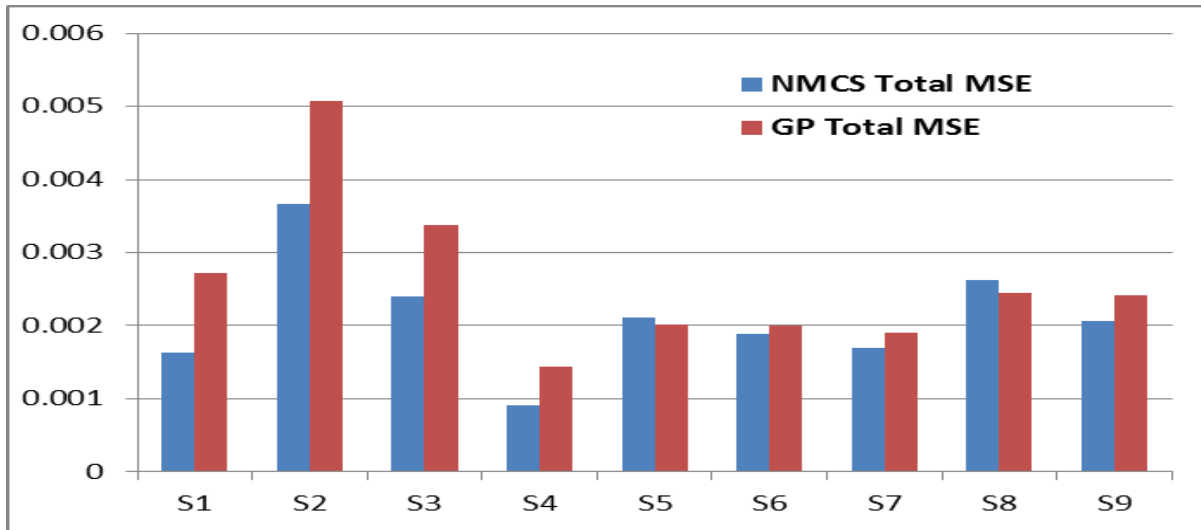
Poor Fitted Observations
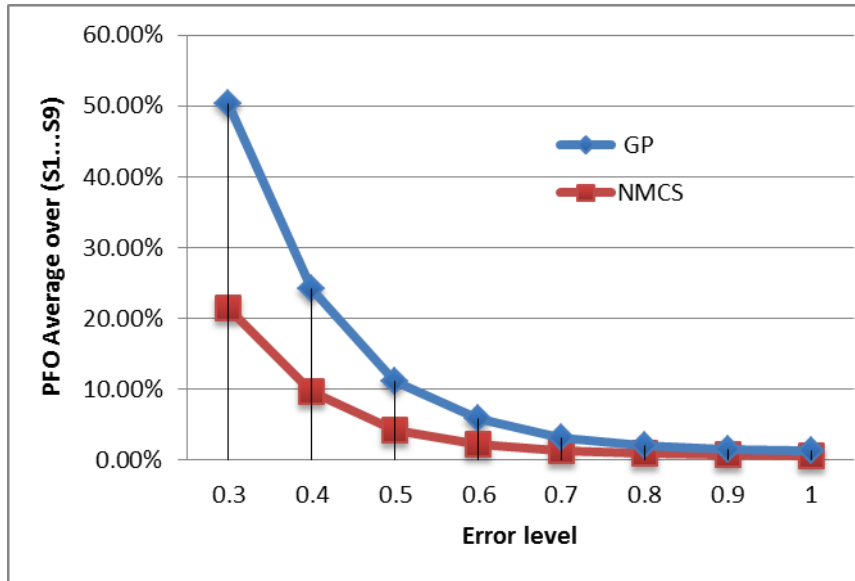
Expression Discovery

Exercise :

- Possible atoms : 1, 2, 3, +, -

- Goal : find expressions containing less than 11 atoms that have great evaluations.

- Generate random expressions (i.e. list of atoms).

- Evaluate an expression given as a list of atoms.

- Use NMCS to generate expressions

Expression Discovery

```
            +
           / \
          +   1
         / \
        2   +
           / \
          1   3
```

+ + 2 + 1 3 1
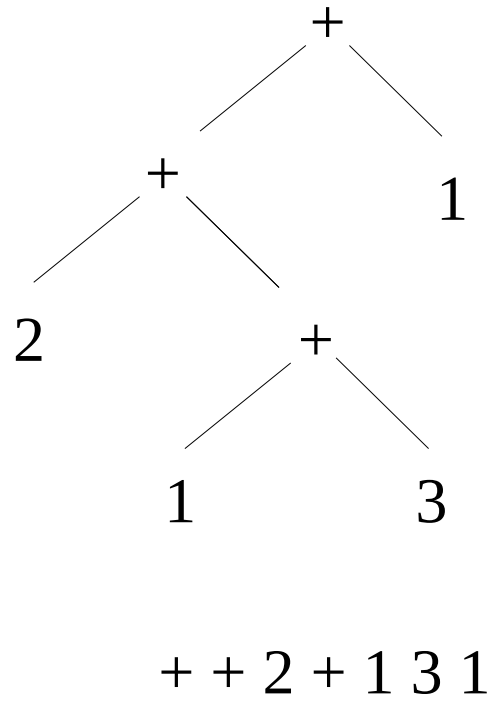
Nested Monte-Carlo Search for Two-player Games

- The quality of information propagated during the search can be increased via a discounting heuristic, leading to a better move selection for the overall algorithm.

- Improving the cost-effectiveness of the algorithm without changing the resulting policy by using safe pruning criteria.

- Long-term convergence to an optimal strategy can be guaranteed by wrapping NMCS inside a UCT-like algorithm.

Nested Monte-Carlo Search for Two-player Games

- The discounting heuristic turns a win/loss game into a game with a wide range of outcomes by having the max player preferring short wins to long wins, and long losses to short losses.

- A playout returns $v(s_t) / (t + 1)$ with $v(s_t)$ in $\{-1,1\}$
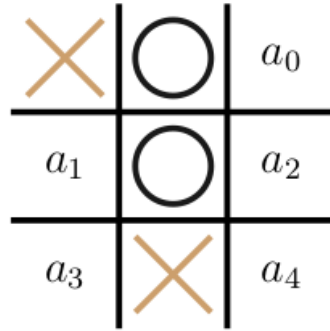
(a) The X player is to play. Any move except a_3 leads to a draw with perfect play.
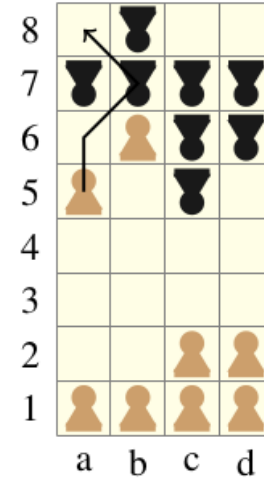
(b) White's winning move is `a5-a6`. `b6-a7` and `b6-c7` initially seem good but are blunders.

Figure 1: Partially played games of TicTacToe and Breakthrough with a single winning move for the turn player.

Table 1: Effect of discounting on the distribution of nested level 2 policies applied to Figure 1a, across 1000 games.

Move	$\Pi(V(\textsc{nmc}(2)))$		$\Pi(V_D(\textsc{nmc}_D(2)))$	
	Value	Frequency	Value	Frequency
a_0	$\{0\}$	0	$\{0\}$	0
a_1	$\{0, 1\}$	176	$\{0\}$	0
a_2	$\{0, 1\}$	123	$\{0, \frac{1}{4}\}$	0
a_3	$\{1\}$	575	$\{\frac{1}{2}\}$	1000
a_4	$\{0, 1\}$	126	$\{0, \frac{1}{4}\}$	0
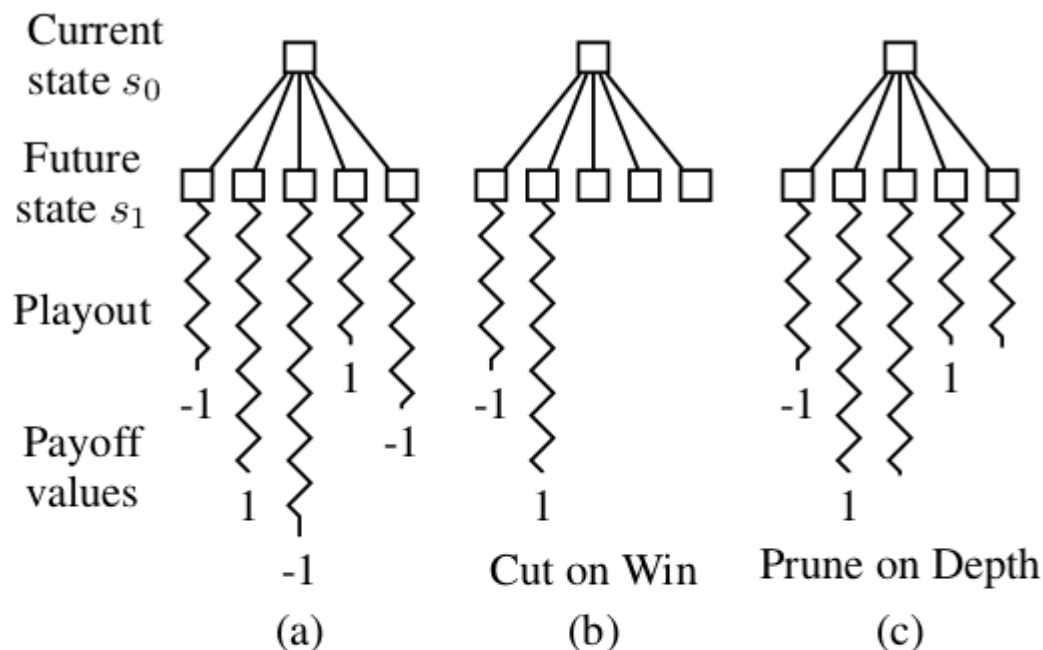
Figure 2: Effect of the pruning strategies on an NMCS run. We assume a max root state and a left-to-right evaluation order. (a) In the standard case, the second and the fourth successor states are equally preferred. With discounting, the fourth successor state is the most-preferred one. (b) This fourth state may fail to be selected when Cut on Win is enabled. (c) With Pruning on Depth and discounting, however, this fourth state would be found and preferred too.

Nested Monte-Carlo Search for Two-player Games

Table 2: Performance of NMC(3) and NMC$_D$(3) starting from Figure 1b, averaged over 900 runs; showing how discounting and pruning affect the number of states visited and the correct move frequency.

Discounting	Pruning	States Visited(k)	Freq(%)
No	None	$4,459 \pm 27$	11.9 ± 2.2
No	COW(≤ 1)	$1,084 \pm 8$	12.3 ± 2.6
No	COW(≤ 2)	214 ± 2	10.9 ± 2.0
No	COW(≤ 3)	25 ± 1	9.8 ± 2.0
Yes	None	$2,775 \pm 26$	64.1 ± 3.4
Yes	POD(≤ 1)	$1,924 \pm 20$	64.7 ± 3.5
Yes	POD(≤ 2)	$1,463 \pm 16$	58.6 ± 3.5
Yes	POD(≤ 3)	627 ± 19	62.4 ± 3.3

Nested Monte-Carlo Search for Two-player Games

Table 3: Winrates (%) of NMCS with discounting vs. NMCS without it for nesting levels 0 to 2 and game engine speed.

Game	Nesting Level			States visited
	0	1	2	per second (k)
Breakthrough	79.6	99.6	99.4	411
misère	42.4	80.8	90.0	409
Knightthrough	78.6	100.0	100.0	264
misère	46.0	83.2	85.8	328
Domineering	71.2	77.0	83.8	550
misère	43.4	63.2	68.4	592
NoGo	62.8	76.4	83.4	357
misère	53.2	65.6	67.2	648
AtariGo	69.6	97.2	100.0	280

Table 4: Win percentages of NMCS against a standard MCTS player for various settings and thinking times.

Game	n	COW	POD	10ms	20ms	40ms	80ms	160ms	320ms
Breakthrough	1			3.2	6.0	12.0	11.6	7.8	6.4
	1		✔	27.6	22.6	16.8	21.6	15.4	20.4
	1	✔		22.6	25.2	30.4	34.6	35.2	39.6
	2	✔		4.6	2.0	2.4	1.4	2.4	3.8
Breakthrough misère	1			85.4	83.4	70.2	60.8	57.0	56.4
	1		✔	91.4	95.6	97.0	97.8	98.8	98.8
	1	✔		95.2	95.2	98.0	99.0	99.8	**99.8**
	2	✔		1.0	27.6	43.6	87.0	93.2	95.6
Knightthrough	1			42.2	57.2	9.8	49.4	50.2	50.0
	1		✔	68.6	50.2	42.4	42.4	46.4	44.6
	1	✔		27.2	25.4	28.0	43.4	49.2	49.6
	2	✔		20.0	16.4	5.8	1.8	29.2	38.2
Knightthrough misère	1			43.0	31.6	20.0	15.4	11.2	12.6
	1		✔	54.6	72.2	80.6	88.4	94.2	98.4
	1	✔		77.8	82.2	88.8	94.4	98.2	**98.6**
	2	✔		20.8	18.6	32.2	42.2	54.0	67.0
Domineering	1			13.4	8.6	8.6	6.0	14.2	28.0
	1		✔	40.8	34.4	37.4	48.4	50.0	50.0
	1	✔		44.4	38.6	40.6	49.4	50.0	50.0
	2	✔		11.2	14.4	20.2	25.2	32.2	45.4
Domineering misère	1			33.4	25.2	20.0	18.8	13.2	12.2
	1		✔	45.4	47.2	56.8	60.2	62.8	54.2
	1	✔		69.4	66.6	71.6	70.4	68.4	**58.6**
	2	✔		37.0	45.2	45.6	51.0	57.8	53.6
NoGo	1			5.8	3.0	2.6	3.0	0.6	0.8
	1		✔	7.2	16.0	31.8	35.2	35.4	40.6
	1	✔		37.6	39.2	38.4	40.8	47.8	48.0
	2	✔		0.4	2.8	5.4	15.0	20.6	17.0
NoGo misère	1			14.6	6.6	5.2	3.0	2.4	1.8
	1		✔	17.2	25.0	38.8	51.2	48.2	48.8
	1	✔		55.4	56.6	57.0	57.6	54.6	**60.8**
	2	✔		5.2	10.6	19.4	35.6	37.2	47.8
Atari-Go	1			0.6	2.2	4.6	5.4	6.8	7.6
	1		✔	0.2	19.2	42.0	42.0	55.4	67.2
	1	✔		42.0	59.0	60.2	71.0	71.2	**77.2**
	2	✔		0.2	0.0	0.6	7.4	8.6	4.8

Algorithm 1: Two-player two-outcome NMCS.

1   nested(*nesting n, state s, depth d, bound λ*)
2   **while** $s \notin T$ **do**
3       $s^* \leftarrow \text{rand}(\delta(s))$
4   **if** $\tau(s) = \max$ **then** $l^* \leftarrow \frac{-1}{d}$ **else** $l^* \leftarrow \frac{1}{d}$
5   **if** *d-pruning* **and** $\tau(s)\{-l^*, \lambda\} = \lambda$ **then return** λ
6   **if** $n > 0$ **then**
7   **foreach** s' **in** $\delta(s)$ **do**
8           $l \leftarrow \text{nested}(n - 1, s', d + 1, l^*)$
9   **if** $\tau(s)\{l, l^*\} \neq l^*$ **then** $s^* \leftarrow s'; l^* \leftarrow l$
10   **if** *cut on win* **and** $\tau(s)\{l, 0\} \neq 0$ **then break**
11       $s \leftarrow s^*$
12       $d \leftarrow d + 1$
13   **if** *discounting* **then return** $\frac{v(s)}{d}$
14   **else return** $v(s)$

Exercise

- Modify Breakthrough to play Misere Breakthrough.
- Modify playouts for discounted rewards.
- Nested playouts.
- UCT with nested discounted playouts.
- Compare to standard UCT.

Nested Rollout Policy Adaptation

Nested Rollout Policy Adaptation

- NRPA [Rosin 2011] is NMCS with policy learning.

- It uses Gibbs Sampling as a playout policy.

- It adapts the weights of the moves according to the best sequence of moves found so far.

- During adaptation each weight of a move of the best sequence is incremented and the other moves in the same state are decreased proportionally to the exponential of their weights.

Nested Rollout Policy Adaptation

- Each move is associated to a weight w_i

- During a playout each move is played with a probability:

$$\exp(w_i) / \Sigma_k \exp(w_k)$$

Nested Rollout Policy Adaptation
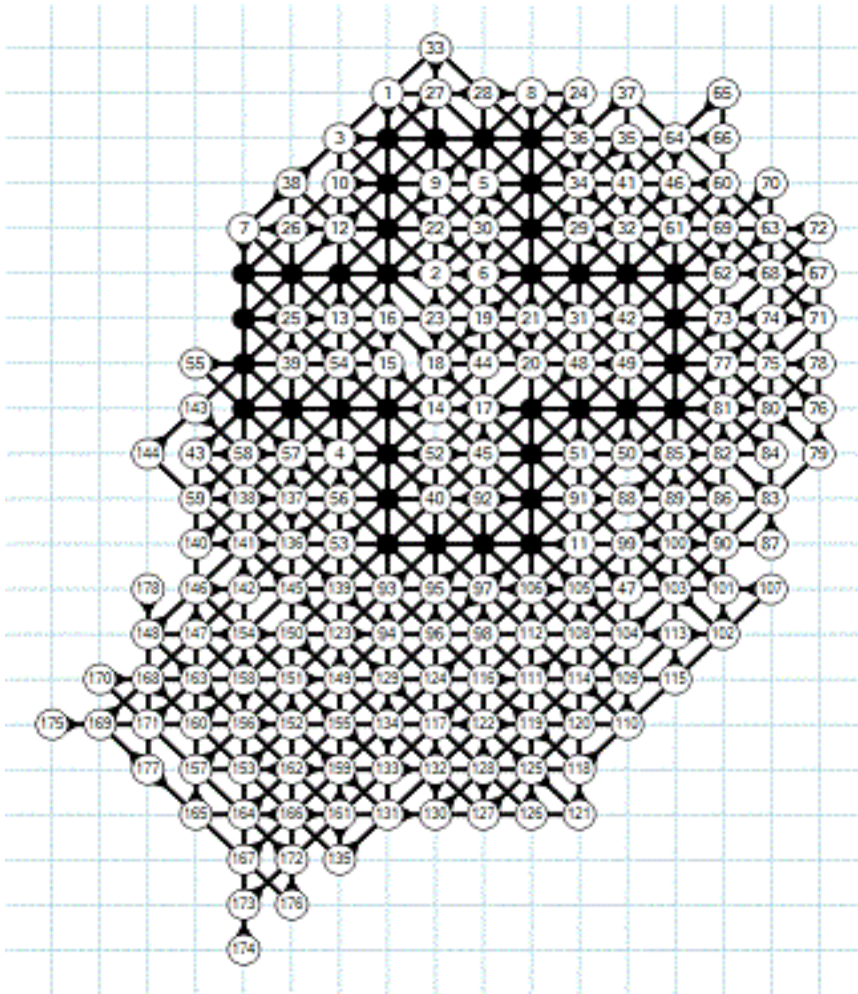
- For each move of the best sequence:

$w_i = w_i + 1$

- For each possible move of each state of the best sequence:

$w_j = w_j - \exp(w_j) / \Sigma_k \exp(w_k)$

Morpion Solitaire

World record [Rosin 2011]

Applications of NRPA

- 3D packing with object orientation.

Applications of NRPA

- Improvement of some alignments for Multiple Sequence Alignment [Edelkamp & al 2015].
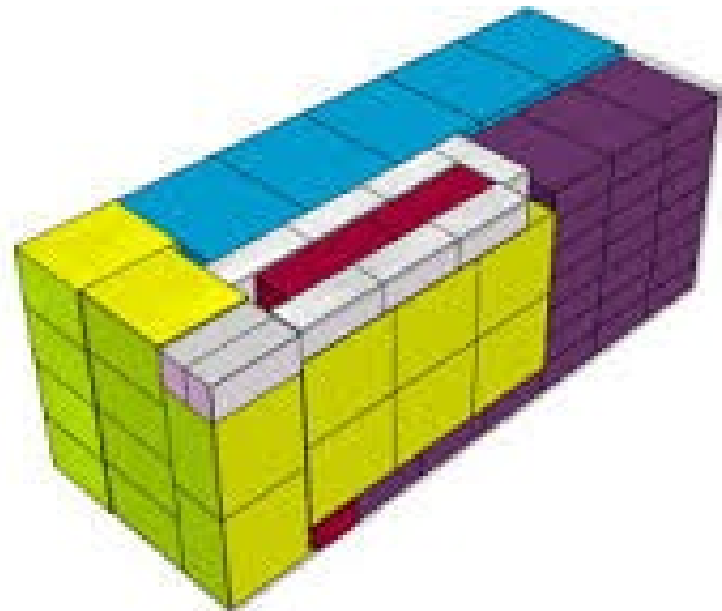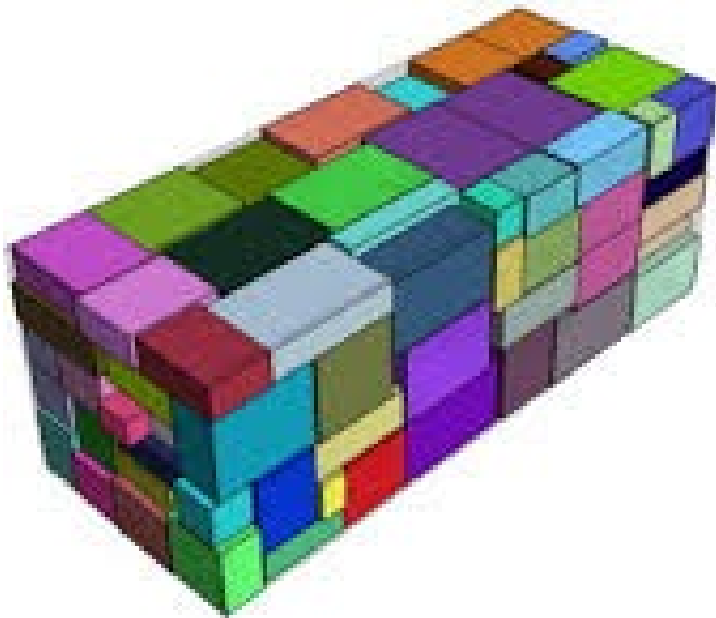
Applications of NRPA

- Traveling Salesman Problem with Time Windows [Cazenave 2012].
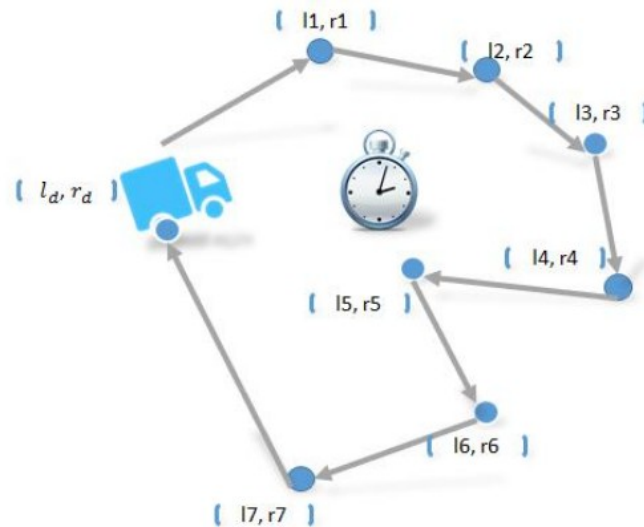
- Physical traveling salesman problem.

Applications of NRPA

- State of the art results for Logistics [Edelkamp & al. 2016].
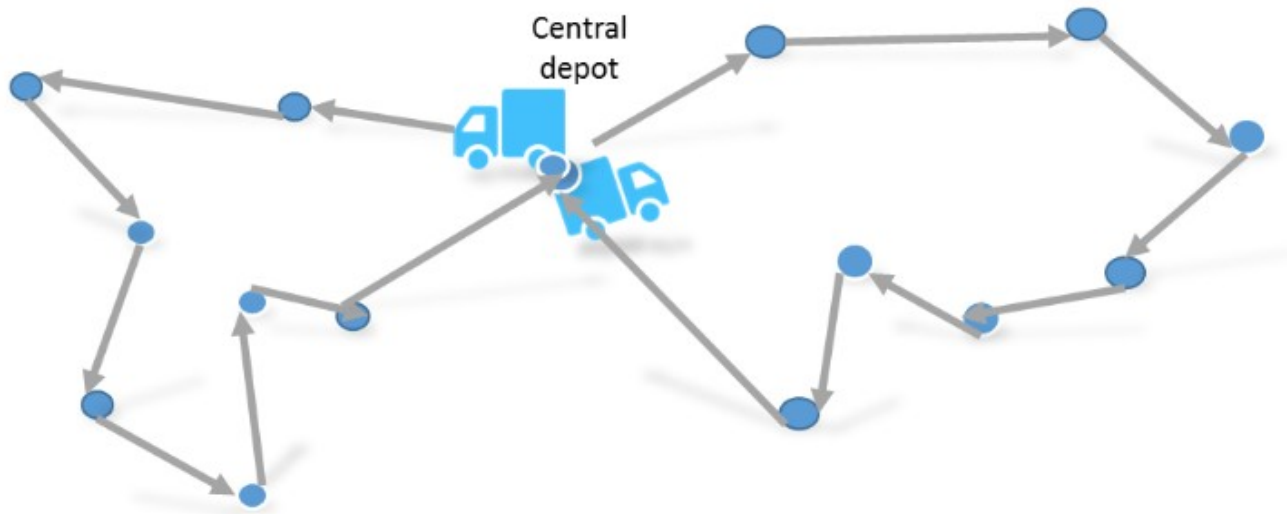
ENEDIS Agents

- ENEDIS fleet of vehicles is one of the largest.

- They plan interventions every day.

- Monte Carlo Search is 5% better than the specialized algorithms they use.

- Millions of kilometers saved each year [Cazenave et al. 2021].

RNA Molecule Design

• Find a sequence that has a given folding [Cazenave et al. 2020].
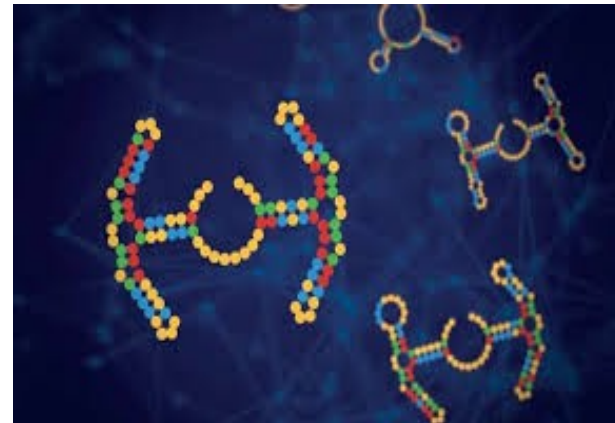
Network Traffic Engineering

- Provide routing configurations in networks that:

  - Miminize ressources

  - Preserve QoS.
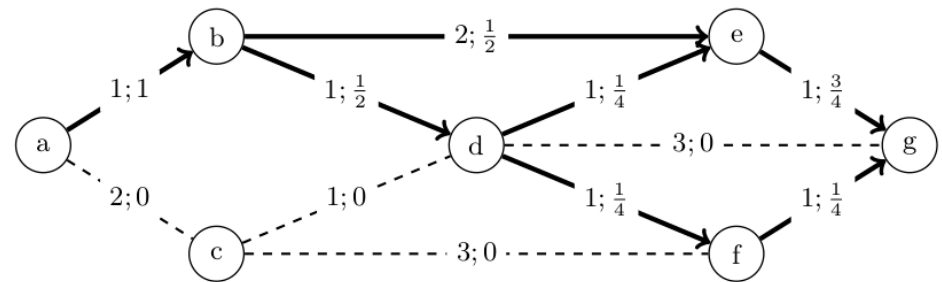
- Better than local search [Dang et al. 2021]:
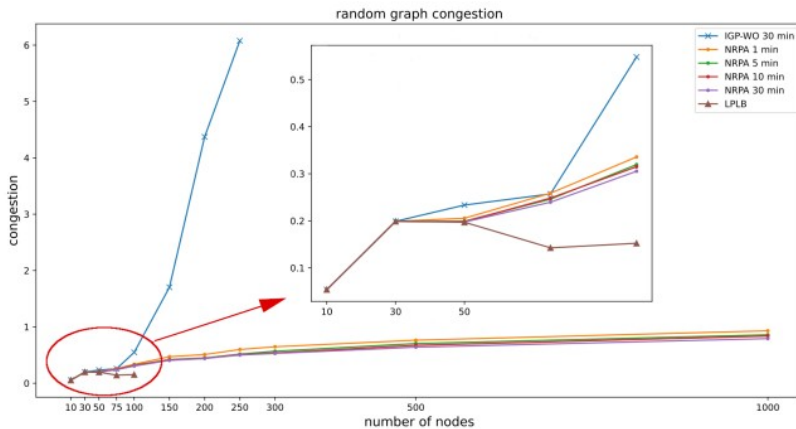
Snake in the box
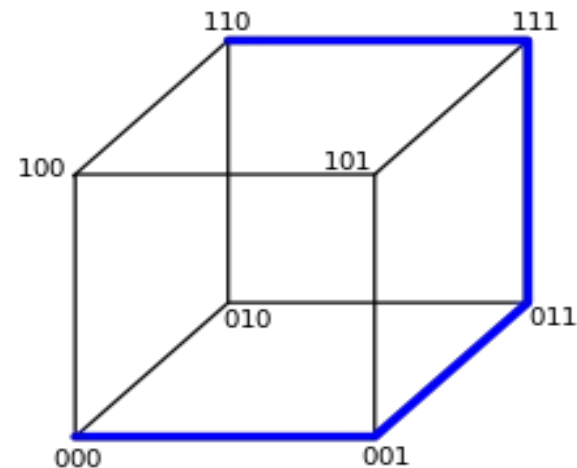
- A path such that for every node only two neighbors are in the path.

- Applications: Electrical engineering, coding theory, computer network topologies.

- World records with NRPA [Dang et al. 2023].

Nested Rollout Policy Adaptation

- Morpion Solitaire [Rosin 2011]

- CrossWords [Rosin 2011]

- Traveling Salesman Problem with Time Windows [Cazenave et al. 2012]

- 3D Packing with Object Orientation [Edelkamp et al. 2014]

- Multiple Sequence Alignment [Edelkamp et al. 2015]

- SameGame [Cazenave et al. 2016]

- Vehicle Routing Problems [Edelkamp et al. 2016, Cazenave et al. 2020]

- Graph Coloring [Cazenave et al. 2020]

- RNA Inverse Folding [Cazenave & Fournier 2020]

- Network Traffic Engineering [Dang et al. 2021]

- Slicing 5G [Elkael et al. 2021]

- Snake in the Box [Dang et al. 2023]

- …

Algorithm 1 The playout algorithm

playout ($state$, $policy$)
$sequence \leftarrow$ []
while true **do**
   if $state$ is terminal **then**
      return (score ($state$), $sequence$)
   end if
   $z \leftarrow 0.0$
   for m in possible moves for $state$ **do**
      $z \leftarrow z + \exp(policy\,[\text{code}(m)])$
   end for
   choose a $move$ with probability $\frac{exp(policy[code(move)])}{z}$
   $state \leftarrow$ play ($state$, $move$)
   $sequence \leftarrow sequence + move$
end while

Algorithm 2 The Adapt algorithm

Adapt ($policy$, $sequence$)
$polp \leftarrow policy$
$state \leftarrow root$
for $move$ in $sequence$ **do**
    $polp$ [code($move$)] $\leftarrow polp$ [code($move$)] + α
    $z \leftarrow 0.0$
    for m in possible moves for $state$ **do**
        $z \leftarrow z$ + exp ($policy$ [code(m)])
    end for
    for m in possible moves for $state$ **do**
        $polp$ [code(m)] $\leftarrow polp$ [code(m)] - $\alpha * \frac{exp(policy[code(m)])}{z}$
    end for
    $state \leftarrow$ play ($state$, $move$)
end for
$policy \leftarrow polp$

Algorithm 3 The NRPA algorithm.

NRPA ($level$, $policy$)
if level == 0 **then**
    return playout (root, $policy$)
end if
$bestScore \leftarrow -\infty$
for N iterations **do**
    (result,new) $\leftarrow$ NRPA($level - 1, policy$)
    if result $\geq$ bestScore **then**
        bestScore $\leftarrow$ result
        seq $\leftarrow$ new
    end if
    policy $\leftarrow$ Adapt (policy, seq)
end for
return (bestScore, seq)

Exercise

- Apply NRPA to the Left Move problem.

- Write a function playout (state) that plays a playout using Gibbs sampling.

- The probability of playing a move is proportional to the exponential of the weight of the move.

- weight is a dictionary that contains the weights of the moves.

- Write the Adapt function

- Write the NRPA function

Selective Policies

- Prune bad moves during playouts.
- Modify the legal moves function.
- Use rules to find bad moves.
- Different domain specific rules for :
  - Bus regulation,
  - SameGame,
  - Weak Schur numbers.

Bus Regulation

- At each stop a regulator can decide to make a bus wait before continuing his route.

- Waiting at a stop can reduce the overall passengers waiting time.

- The score of a simulation is the sum of all the passengers waiting time.

- Optimizing a problem is finding a set of bus stopping times that minimizes the score of the simulation.
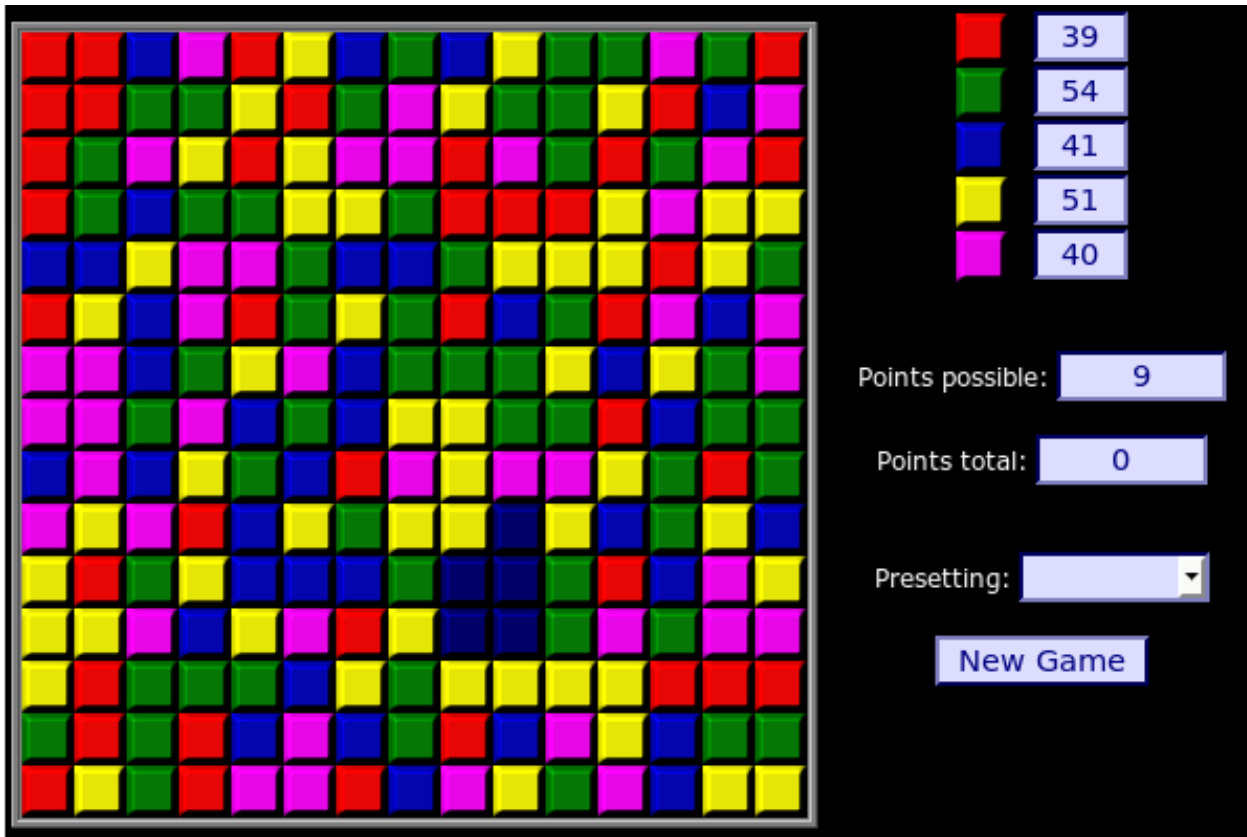
Bus Regulation

- Standard policy: between 1 and 5 minutes

- Selective policy : waiting time of 1 if there are fewer than δ stops before the next bus.

- Code for a move:
  - the bus stop,
  - the time of arrival to the bus stop,
  - the number of minutes to wait before leaving the stop.

Bus Regulation

Time	No δ	δ = 3
0.01	2,620	2,147
0.02	2,441	2,049
0.04	2,329	2,000
0.08	2,242	1,959
0.16	2,157	1,925
0.32	2,107	1,903
0.64	2,046	1,868
1.28	1,974	1,811
2.56	1,892	1,754
5.12	1,802	1,703
10.24	1,737	1,660
20.48	1,698	1,640
40.96	1,682	1,629
81.92	1,660	1,617
163.84	1,632	**1,610**

SameGame

SameGame

- Code of a move = Zobrist hashing.

- Tabu color strategy = avoid moves of the dominant color until there is only one block of the dominant color.

- Selective policy = allow moves of size two of the tabu color when the number of moves already played is greater than t.

SameGame

Time	No tabu	tabu	t > 10
0.01	155.83	**352.19**	257.59
0.02	251.28	**707.56**	505.05
0.04	340.18	**927.63**	677.57
0.08	404.27	**1,080.64**	822.44
0.16	466.15	**1,252.14**	939.30
0.32	545.78	**1,375.78**	1,058.54
0.64	647.63	**1,524.37**	1,203.91
1.28	807.20	**1,648.16**	1,356.81
2.56	1,012.42	**1,746.74**	1,497.90
5.12	1,184.77	**1,819.43**	1,605.86
10.24	1,286.25	**1,886.48**	1,712.17
20.48	1,425.55	**1,983.42**	1,879.10
40.96	1,579.67	**2,115.80**	2,100.47
81.92	1,781.40	2,319.44	**2,384.24**
163.84	2,011.25	2,484.18	**2,636.22**

SameGame

Standard test set of 20 boards:

NMCS	SP-MCTS	NRPA	web
77,934	78,012	80,030	87,858

Same Game

- Hybrid Parallelization [Negrevergne 2017].

- Root Parallelization for each computer. Leaf Parallelization of the playouts using threads.

- New record of 83 050.

- Parallelization for Morpion Solitaire [Nagorko 2019].

Weak Schur Numbers

- Find a partition of consecutive numbers that contains as many consecutive numbers as possible

- A partition must not contain a number that is the sum of two previous numbers in the same partition.

- Partition of size 3 :

  1 2 4 8 11 22

  3 5 6 7 19 21 23

  9 10 12 13 14 15 16 17 18 20

Weak Schur Numbers

- Often a good move to put the next number in the same partition as the previous number.

- If it is legal to put the next number in the same partition as the previous number then it is the only legal move considered.

- Otherwise all legal moves are considered.

- The code of a move for the Weak Schur problem takes as input the partition of the move, the integer to assign and the previous number in the partition.

Weak Schur Numbers

Time	ws(9)	ws-rule(9)
0.01	199	2,847
0.02	246	3,342
0.04	263	3,717
0.08	273	4,125
0.16	286	4,465
0.32	293	4,757
0.64	303	5,044
1.28	314	5,357
2.56	331	5,679
5.12	362	6,065
10.24	384	6,458
20.48	403	6,805
40.96	422	7,117
81.92	444	7,311
163.84	473	**7,538**

Selective Policies

- We have applied selective policies to three quite different problems.

- For each problem selective policies improve NRPA.

- We used only simple policy improvements.

- Better performance could be obtained refining the proposed policies.

Exercise

- Apply NRPA to the Weak Schur problem.

- Write a function that plays a playout using Gibbs sampling.

- The probability of playing a move is proportional to the exponential of the weight of the move.

- weight is a dictionary that contains the weights associated to the moves.

- code (move) returns the integer associated to the move in the weight dictionary.

Exercise

- Write the adapt function that modifies the weights of the moves according to the best sequence of moves.

- Weights of the moves of the best sequence are incremented.

- For each state of the best sequence, weights of all the moves are reduced proportional to their probabilities.

Exercise

- Write the multi level NRPA code that retains a best sequence per level and recursively calls lower levels.

- Level zero is a playout with Gibbs sampling.

Analysis of Nested Rollout Policy Adaptation

The playouts use Gibbs sampling. Each move m_{ik} is associated to a weight w_{ik}. The probability p_{ik} of choosing the move m_{ik} in a playout is the softmax function:

$$p_{ik} = \frac{e^{w_{ik}}}{\Sigma_j e^{w_{ij}}}$$

The cross-entropy loss for learning to play move m_{ib} is $C_i = -log(p_{ib})$. In order to apply the gradient we calculate the partial derivative of the loss: $\frac{\delta C_i}{\delta p_{ib}} = -\frac{1}{p_{ib}}$. We then calculate the partial derivative of the softmax with respect to the weights:

$$\frac{\delta p_{ib}}{\delta w_{ij}} = p_{ib}(\delta_{bj} - p_{ij})$$

Where $\delta_{bj} = 1$ if $b = j$ and 0 otherwise. Thus the gradient is:

$$\nabla w_{ij} = \frac{\delta C_i}{\delta p_{ib}} \frac{\delta p_{ib}}{\delta w_{ij}} = -\frac{1}{p_{ib}} p_{ib}(\delta_{bj} - p_{ij}) = p_{ij} - \delta_{bj}$$

If we use α as a learning rate we update the weights with:

$$w_{ij} = w_{ij} - \alpha(p_{ij} - \delta_{bj})$$

This is the formula used in the NRPA algorithm to adapt weights.

Generalized Nested Rollout Policy Adaptation

We propose to generalize the NRPA algorithm by generalizing the way the probability is calculated using a temperature τ and a bias β_{ij}:

$$p_{ik} = \frac{e^{\frac{w_{ik}}{\tau} + \beta_{ik}}}{\Sigma_j e^{\frac{w_{ij}}{\tau} + \beta_{ij}}}$$

Generalized Nested Rollout Policy Adaptation

The formula for the derivative of $f(x) = \frac{g(x)}{h(x)}$ is:

$$f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{h^2(x)}$$

So the derivative of p_{ib} relative to w_{ib} is:

$$\frac{\delta p_{ib}}{\delta w_{ib}} = \frac{\frac{1}{\tau}e^{\frac{w_{ib}}{\tau}+\beta_{ib}}\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}} - \frac{1}{\tau}e^{\frac{w_{ib}}{\tau}+\beta_{ib}}e^{\frac{w_{ib}}{\tau}+\beta_{ib}}}{(\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}})^2}$$

$$\frac{\delta p_{ib}}{\delta w_{ib}} = \frac{1}{\tau}\frac{e^{\frac{w_{ib}}{\tau}+\beta_{ib}}}{\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}}}\frac{\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}} - e^{\frac{w_{ib}}{\tau}+\beta_{ib}}}{\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}}}$$

$$\frac{\delta p_{ib}}{\delta w_{ib}} = \frac{1}{\tau}p_{ib}(1 - p_{ib})$$

Generalized Nested Rollout Policy Adaptation

The derivative of p_{ib} relative to w_{ij} with $j \neq b$ is:

$$\frac{\delta p_{ib}}{\delta w_{ij}} = -\frac{1}{\tau} \frac{e^{\frac{w_{ij}}{\tau}+\beta_{ij}} e^{\frac{w_{ib}}{\tau}+\beta_{ib}}}{(\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}})^2}$$

$$\frac{\delta p_{ib}}{\delta w_{ij}} = -\frac{1}{\tau} p_{ij} p_{ib}$$

We then derive the cross-entropy loss and the softmax to calculate the gradient:

$$\nabla w_{ij} = \frac{\delta C_i}{\delta p_{ib}} \frac{\delta p_{ib}}{\delta w_{ij}} = -\frac{1}{\tau} \frac{1}{p_{ib}} p_{ib}(\delta_{bj} - p_{ij}) = \frac{p_{ij} - \delta_{bj}}{\tau}$$

If we use α as a learning rate we update the weights with:

$$w_{ij} = w_{ij} - \alpha \frac{p_{ij} - \delta_{bj}}{\tau}$$

Generalized Nested Rollout Policy Adaptation

Let the weights and probabilities of playing moves be indexed by the iteration of the GNRPA level. Let w_{nij} be the weight w_{ij} at iteration n, p_{nij} be the probability of playing move j at step i at iteration n, δ_{nbj} the δ_{bj} at iteration n.

$$p_{0ij} = \frac{e^{\frac{1}{\tau} w_{0ij} + \beta_{ij}}}{\Sigma_k e^{\frac{1}{\tau} w_{0ik} + \beta_{ik}}}$$

$$w_{1ij} = w_{0ij} - \frac{\alpha}{\tau}(p_{0ij} - \delta_{0bj})$$

$$p_{1ij} = \frac{e^{\frac{1}{\tau} w_{1ij} + \beta_{ij}}}{\Sigma_k e^{\frac{1}{\tau} w_{1ik} + \beta_{ik}}} = \frac{e^{\frac{1}{\tau} w_{0ij} - \frac{\alpha}{\tau^2}(p_{0ij} - \delta_{0bj}) + \beta_{ij}}}{\Sigma_k e^{\frac{1}{\tau} w_{1ik} + \beta_{ik}}}$$

$$w_{2ij} = w_{1ij} - \frac{\alpha}{\tau}(p_{1ij} - \delta_{1bj}) = w_{0ij} - \frac{\alpha}{\tau}(p_{0ij} - \delta_{0bj} + p_{1ij} - \delta_{1bj})$$

By recurrence we get:

$$p_{nij} = \frac{e^{\frac{1}{\tau} w_{nij} + \beta_{ij}}}{\Sigma_k e^{\frac{1}{\tau} w_{nik} + \beta_{ik}}} = \frac{e^{\frac{w_{0ij}}{\tau} - \frac{\alpha}{\tau^2}(\Sigma_k p_{kij} - \delta_{kbj}) + \beta_{ij}}}{\Sigma_k e^{\frac{1}{\tau} w_{nik} + \beta_{ik}}}$$

Generalized Nested Rollout Policy Adaptation

From this equation we can deduce the equivalence between different algorithms. For example $GNRPA_1$ with $\alpha_1 = (\frac{\tau_1}{\tau_2})^2 \alpha_2$ and τ_1 is equivalent to $GNRPA_2$ with α_2 and τ_2 provided we set w_{0ij} in $GNRPA_1$ to $\frac{\tau_1}{\tau_2} w_{0ij}$. It means we can always use $\tau = 1$ provided we correspondingly set α and w_{0ij}.

Another deduction we can make is we can set $\beta_{ij} = 0$ provided we set $w_{0ij} = w_{0ij} + \tau \times \beta_{ij}$. We can also set $w_{0ij} = 0$ and use only β_{ij} which is easier.

The equivalences mean that GNRPA is equivalent to NRPA with the appropriate α and w_{0ij}. However it can be more convenient to use β_{ij} than to initialize the weights w_{0ij} as we will see for SameGame.
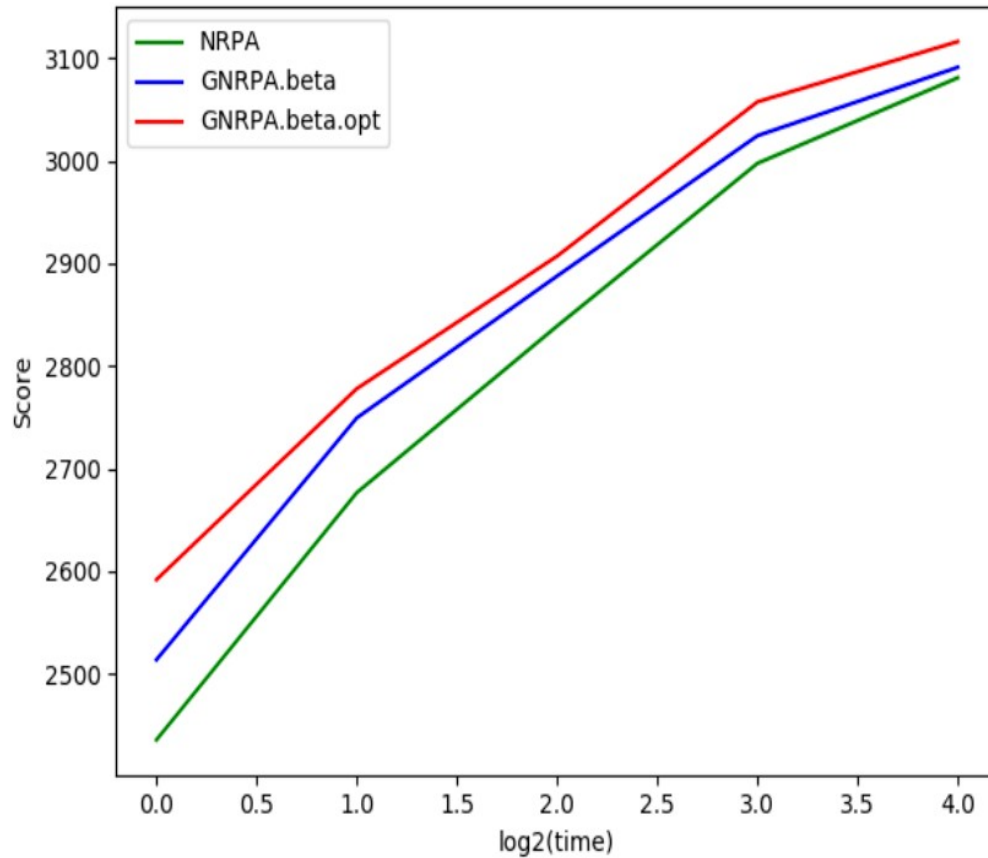
SameGame

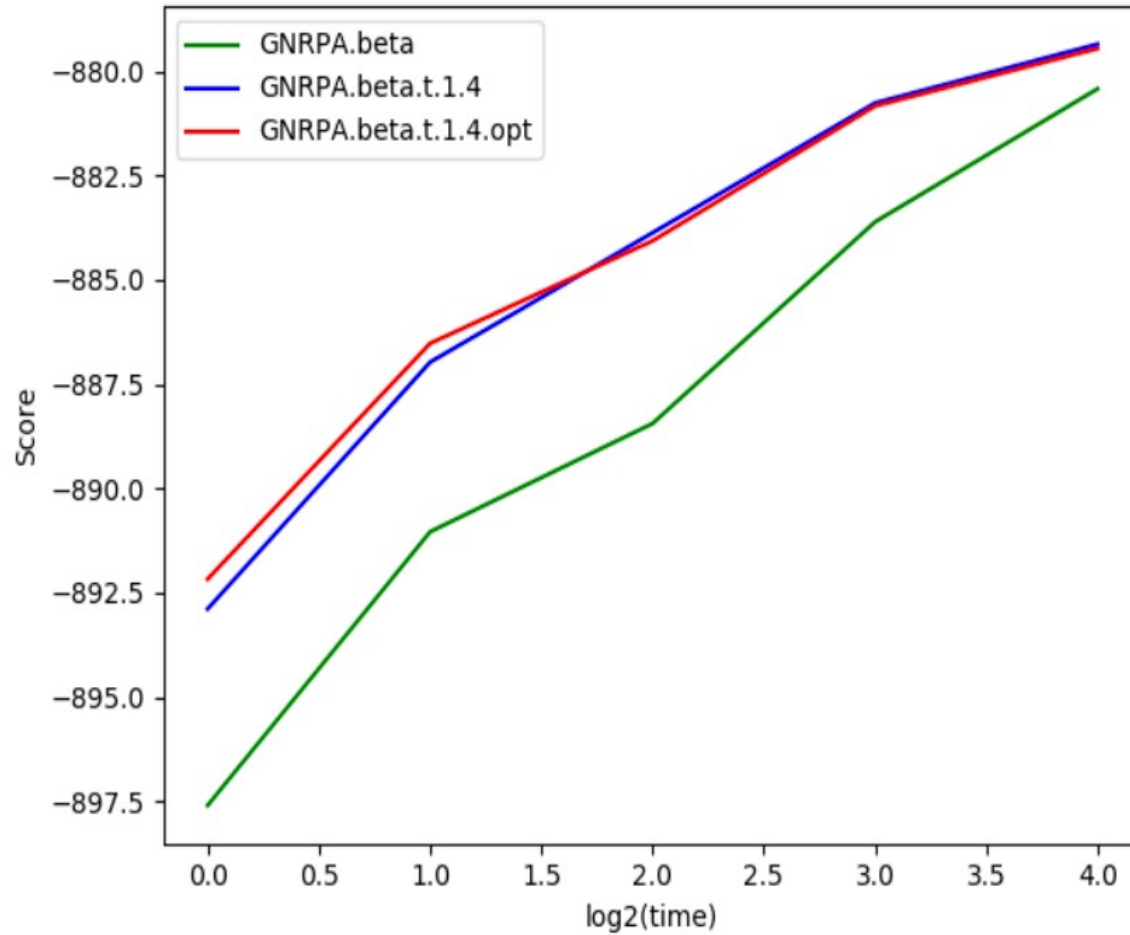Fig. 1: Evolution of the average scores of the three algorithms at SameGame.

TSPTW

Fig. 2: Evolution of the average scores of the three algorithms for TSPTW.

TSPTW

Table 2: Results for the TSPTW rc204.1 problem

Time	NRPA	GNRPA.beta	GNRPA.beta.t.1.4	GNRPA.beta.t.1.4.opt
40.96	-3745986.46 (245766.53)	-897.60 (1.32)	-892.89 (0.96)	-892.17 (1.04)
81.92	-1750959.11 (243210.68)	-891.04 (1.05)	-886.97 (0.87)	-886.52 (0.83)
163.84	-1030946.86 (212092.35)	-888.44 (0.98)	-883.87 (0.71)	-884.07 (0.70)
327.68	-285933.63 (108975.99)	-883.61 (0.63)	-880.76 (0.40)	-880.83 (0.32)
655.36	-45918.97 (38203.97)	-880.42 (0.30)	-879.35 (0.16)	-879.45 (0.17)

GNRPA

- NRPA with a bias.
- Equivalent to the initialization of the weights.
- More convenient to use a bias.
- We can always set the temperature to 1 without a loss of generality.
- Good results for SameGame and TSPTW.

GNRPA

- Exercise:
- Apply GNRPA to the Weak Schur problem.

Warm-Starting Nested Rollout Policy Adaptation with Optimal Stopping

Chen Dang,[1,2] **Cristina Bazgan,**[2] **Tristan Cazenave,**[2] **Morgan Chopin,**[1] **Pierre-Henri Wuillemin** [3]

[1] Orange Labs, Châtillon, France
[2] Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, F-75016 Paris, France
[3] Sorbonne Université, CNRS, UMR 7606, LIP6, F-75005 Paris, France
chen.dang@orange.com, cristina.bazgan@dauphine.psl.eu, tristan.cazenave@dauphine.psl.eu, morgan.chopin@orange.com,
pierre-henri.wuillemin@lip6.fr

Abstract

Nested Rollout Policy Adaptation (NRPA) is an approach using online learning policies in a nested structure. It has achieved a great result in a variety of difficult combinatorial optimization problems. In this paper, we propose Meta-NRPA, which combines optimal stopping theory with NRPA for warm-starting and significantly improves the performance of NRPA. We also present several exploratory techniques for NRPA which enable it to perform better exploration. We establish this for three notoriously difficult problems ranging from telecommunication, transportation and coding theory, namely Minimum Congestion Shortest Path Routing, Traveling Salesman Problem with Time Windows and Snake-in-the-Box. We also improve the lower bounds of the Snake-in-the-Box problem for multiple dimensions.

Introduction

Search algorithms can be used to solve many difficult combinatorial optimization problems. Monte Carlo Search algorithms rely on randomness to discover good sequences of decisions for difficult problems. Following their success in games (Coulom 2007; Kocsis and Szepesvári 2006; Browne et al. 2012; Silver et al. 2016, 2017, 2018), they were applied with success to multiple combinatorial optimization problems (Cazenave 2009; Rosin 2011). They work particularly well when combined with machine learning.

We propose new general techniques to Monte Carlo Search algorithms that improve the algorithms for multiple applications. Using these techniques we get better results than the previous ones for three difficult combinatorial optimization problems from varied fields, namely telecommunication, transportation and coding theory.

The problems we address were already successfully addressed using Nested Rollout Policy Adaptation (NRPA). They are the Minimum Congestion Shortest Path Routing problem (Dang et al. 2021), the Traveling Salesman Problem with Time Windows (Edelkamp et al. 2013) and the Snake-in-the-Box problem (Edelkamp and Cazenave 2016). For these three problems we improve the results compared to standard NRPA. In particular for the Snake-in-the-Box problem, we provide new lower bounds for several dimensions.

Our contributions deal with the initialization of NRPA using the optimal stopping theory and with better exploration avoiding already scored sequences of decisions. More precisely, we observed that whenever NRPA gets unsatisfactory solutions from the beginning it is highly unlikely that it will find significantly better solutions in subsequent iterations. To remedy this problem, we cast the initialization step of NRPA as an instance of the secretary problem, a well-known optimal stopping problem: a decision maker (DM) wants to recruit a secretary for a job position, n candidate secretaries are thus interviewed one after the other in a random order, which can be ranked among the examined candidates. The DM can decide whether to terminate the recruitment process by accepting the last interviewed candidate. The decision of DM about recruiting a candidate needs to be just after the interview of the candidate and it is irrevocable. In addition, DM has no knowledge of the quality of the upcoming candidates. The goal is to maximize the probability of selecting the best candidate. In our case, we are interested in a variant of this problem, where candidates are NRPA runs and we aim at minimizing the expected rank of the chosen candidate. To our knowledge, this is the first time that the optimal stopping theory is used in the context of Monte Carlo search.

The paper is organized in five sections. The second section deals with related works. The third section details our contributions. The fourth section gives experimental results for the Minimum Congestion Shortest Path Routing problem, the Traveling Salesman Problem with Time Windows and the Snake-in-the-Box problem. Conclusions and future works are given in section five.

Related Works

Monte Carlo Search

Monte Carlo Search has many applications in games and difficult combinatorial optimization problems. When combined with deep learning it surpasses the level of the best human players in games such as Go, Chess and Shogi (Silver et al. 2018). The combination has been applied to many other games with success (Cazenave et al. 2020). It is also the best general algorithm to solve a problem when given only the raw description of the problem as is the case in General Game Playing. Since 2007, all the world champions of the General Game Playing competition have used Monte Carlo

Force Explore

- When a policy has been reinforced a lot, for example in the end of the iterations loop, the playouts are almost deterministic.

- NRPA very often replays the same playout.

- Force Explore detects when a terminal state has already been evaluated before.

- In this case it randomly chooses a move in the playout, modifies it and performs another playout.

- "Warm-Starting Nested Rollout Policy Adaptation with Optimal Stopping", Dang et al. AAAI 2023.

Force Explore

- Exercise:
- Apply Force Explore to the Weak Schur problem.

Warm Starting

- Warm starting performs multiple recursive calls before starting to adapt.

- The optimal stopping criterion is the one of the secretary problem :

  $R_i \leq ic \, / \, (n + 1 - i)$

  with R_i relative rank of the ith item,

  n the total number of items,

  c a constant.

- "Warm-Starting Nested Rollout Policy Adaptation with Optimal Stopping", Dang et al. AAAI 2023.

Warm Starting

- Exercise:
- Apply Warm Starting to the Weak Schur problem.

Warm Starting

- World records for the Snake-in-the-Box.

Bias Weights Learning

- Bias Learning dynamically learns the weight to associate to a bias in GNRPA.

- "Learning the Bias Weights for Generalized Nested Rollout Policy Adaptation", Sentuc et al. LION 2023.

Learning the Bias Weights for Generalized Nested Rollout Policy Adaptation

Julien Sentuc[1], Farah Ellouze[1], Jean-Yves Lucas[2], and Tristan Cazenave[1]

[1] LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France
[2] OSIRIS department, EDF Lab Paris-Saclay, Electricité de France, France

Abstract. Generalized Nested Rollout Policy Adaptation (GNRPA) is a Monte Carlo search algorithm for single player games and optimization problems. In this paper we propose to modify GNRPA in order to automatically learn the bias weights. The goal is both to obtain better results on sets of dissimilar instances, and also to avoid some hyperparameters settings. Experiments show that it improves the algorithm for two different optimization problems: the Vehicle Routing Problem and 3D Bin Packing.

1 Introduction

Monte Carlo Tree Search (MCTS) [20, 12] has been successfully applied to many games and problems [3]. It originates from the computer game of Go [2] with a method based on simulated annealing [4]. The principle underlying MCTS is learning the best move using statistics on random games.

Nested Monte Carlo Search (NMCS) [5] is a recursive algorithm which uses lower level playouts to bias its playouts, memorizing the best sequence at each level. At each stage of the search, the move with the highest score at the lower level is played by the current level. At each step, a lower-level search is launched for all possible moves and the move with the best score is memorized. At level 0, a Monte Carlo simulation is performed, random decisions are made until a terminal state is reached. At the end, the score for the position is returned. NMCS has given good results on many problems like puzzle solving, single player games [22], cooperative path finding or the inverse folding problem [23].

Based on the latter, the Nested Rollout Policy Adaptation (NRPA) algorithm was introduced [26]. NRPA combines nested search, memorizing the best sequence of moves found, and the online learning of a playout policy using this sequence. NRPA achieved world records in Morpion Solitaire and crossword puzzles and has been applied to many problems such as object wrapping [17], traveling salesman with time window [10, 15], vehicle routing problems [16, 8] or network traffic engineering [13].

GNRPA (Generalized Nested Rollout Policy Adaptation) [6] generalizes the way the probability is calculated using a temperature and a bias. It has been applied to some problems like Inverse Folding [7] and Vehicle Routing Problem (VRP) [27].

This work presents an extension of GNRPA using bias learning. The idea is to learn the parameters of the bias along with the policy. We demonstrate that learning the bias parameters improves the results of GNRPA for Solomon instances of the VRP and for 3D Bin Packing.

The probability of choosing the move c at the index i with this bias is:

$$p_{ic} = \frac{e^{\frac{w_{ic}}{\tau}+(w_1 \times \beta_{1ic}+w_2 \times \beta_{2ic})}}{\sum_k e^{\frac{w_{ik}}{\tau}+(w_1 \times \beta_{1ik}+w_2 \times \beta_{2ik})}}$$

Let $A_{ik} = e^{\frac{w_{ik}}{\tau}+(w_1 \times \beta_{1ik}+w_2 \times \beta_{2ik})}$.
The formula for the derivative of $f(x) = \frac{g(x)}{h(x)}$ is :

$$f'(x) = \frac{g'(x) \times h(x) - g(x) \times h'(x)}{h(x)^2}$$

So the derivative of p_{ic} relative to w_1 is:

$$\frac{\delta p_{ic}}{\delta w_1} = \frac{\beta_{1ic} A_{ic} \times \sum_k A_{ik} - A_{ic} \times \sum_k \beta_{1ik} A_{ik}}{(\sum_k A_{ik})^2}$$

$$\frac{\delta p_{ic}}{\delta w_1} = \frac{A_{ic}}{\sum_k A_{ik}} \times (\beta_{1ic} - \frac{\sum_k \beta_{1ik} A_{ik}}{\sum_k A_{ik}})$$

$$\frac{\delta p_{ic}}{\delta w_1} = p_{ic} \times (\beta_{1ic} - \frac{\sum_k \beta_{1ik} A_{ik}}{\sum_k A_{ik}})$$

The cross-entropy loss for learning to play a move is $C_i = -log(p_{ic})$. In order to apply the gradient, we calculate the partial derivative of the loss: $\frac{\delta C_i}{\delta p_{ic}} = -\frac{1}{p_{ic}}$. We then calculate the partial derivative of the softmax with respect to the weight:

$$\nabla w_1 = \frac{\delta C_i}{\delta p_{ic}} \frac{\delta p_{ic}}{\delta w_1} = -\frac{1}{p_{ic}} \times p_{ic}(\beta_{1ic} - \frac{\sum_k \beta_{1ik} A_{ik}}{\sum_k A_{ik}}) =$$

$$\frac{\sum_k \beta_{1ik} A_{ik}}{\sum_k A_{ik}} - \beta_{1ic}$$

Bias Weights Learning

If we use α_1 and α_2 as learning rates, we update the weight with (line 16 of algorithm 2):

$$w_1 \leftarrow w_1 + \alpha_1 (\beta_{1ic} - \frac{\sum_k \beta_{1ik} A_{ik}}{\sum_k A_{ik}})$$
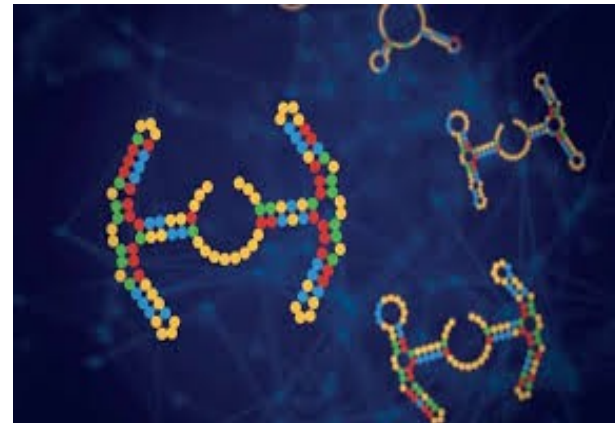
Similarly, the formula for w_2 is (line 17 of algorithm 2):

$$w_2 \leftarrow w_2 + \alpha_2 (\beta_{2ic} - \frac{\sum_k \beta_{2ik} A_{ik}}{\sum_k A_{ik}})$$

Bias Weights Learning

- Exercise:

- Apply Bias Weights Learning to the Weak Schur problem.

Eterna 100

- Find a sequence that has a given folding

Eterna 100

- Human experts have managed to solve the 100 problems of the benchmark

- No program has so far achieved such a score.

- The best score so far is 95/100 by NEMO:

  NEsted MOnte Carlo RNA Puzzle Solver

NEMO

- NEMO uses two sets of heuristics

- General ones that give probabilities to pairs of bases.

- More specific ones that are tailored to puzzle solving.

GNRPA

Let w_{ib} be the weight associated to move b at index i in the sequence. In NRPA the probability of choosing move b at index i is:

$$p_{ib} = \frac{e^{w_{ib}}}{\Sigma_k e^{w_{ik}}}$$

We propose to try Generalized NRPA (GNRPA) [9] for Inverse Folding and to replace it with:

$$p_{ib} = \frac{e^{w_{ib}+\beta_{ib}}}{\Sigma_k e^{w_{ik}+\beta_{ik}}}$$

where we use for β_{ib} the logarithm of the probabilities used in NEMO.

Other Improvements

- Stabilized GNRPA

- Beam GNRPA

- Zobrist Hashing

- Restarts

- Parallelization

Experimental Results

Level	α	N	β_{ib}	P	Beam	H	Solved
1	1.0	100	no	1	1.1.1	0	3
1	1.0	100	yes	1	1.1.1	0	30
1	1.0	100	yes	1	1.1.1	1	32
1	1.0	100	yes	1	4.1.1	1	42
1	1.0	100	yes	1	8.1.1	1	53
1	1.0	100	yes	1	16.1.1	1	54
1	1.0	100	yes	4	8.1.1	1	69
1	1.0	100	yes	4	8.1.1	2	69
2	1.0	100	no	1	1.1.1	0	49
2	1.0	100	yes	1	1.1.1	0	73
2	1.0	100	yes	1	1.1.1	1	75
2	1.0	100	yes	2	1.1.1	0	73
2	1.0	100	yes	3	1.1.1	0	74
2	1.0	100	yes	4	1.1.1	0	80
2	1.0	100	yes	5	1.1.1	0	77
2	1.0	100	yes	6	1.1.1	0	75
2	1.0	100	yes	7	1.1.1	0	80
2	1.0	100	yes	8	1.1.1	0	79
2	1.0	100	yes	9	1.1.1	0	81
2	1.0	100	yes	10	1.1.1	0	80
2	1.0	100	yes	4	8.1.1	1	85
3	1.0	100	yes	1	1.1.1	0	85

Experimental Results

- Leaf Parallelization

Table 2: Parallelization efficiency.

Algorithm	1	2	4	6	8	12
GNRPA(level=1,N=100,P=4,Beam=8)	11.916	6.889	4.526	3.657	3.169	3.359

Experimental Results

Table 3: Number of problems solved by GNRPA using different parameters and a fixed time limit.

β_{ib}	P	Beam	//	R	N	Start	H	1m	2m	4m	8m	16m	32m	64m
no	1.1	1.1	n	n	100.100	0.0	0	30	33	41	50	61	67	69
yes	1.1	1.1	n	n	100.100	0.0	0	58	64	68	72	74	75	79
yes	4.1	4.1	n	n	100.100	0.0	0	71	75	78	79	81	83	84
yes	4.1	8.1	n	n	100.100	0.0	0	75	75	79	80	81	83	84
yes	4.1	8.1	n	n	100.100	0.0	1	75	78	80	82	83	85	87
yes	4.1	8.1	n	n	100.100	4.4	1	76	80	81	82	84	85	87
yes	4.1	8.1	y	n	100.100	4.4	1	78	84	83	86	87	87	88
yes	4.1	8.1	y	3	$\infty.\infty$	4.4	1	80	84	85	85	88	89	92

Experimental Results

- Root Parallelization

Table 6: Number of problems solved with root parallel GNRPA.

Process	1m	2m	4m	8m	16m	32m	64m
20	82	84	85	86	87	88	89

Conclusion

- 95/100 problems solved, same as NEMO.
- Less domain knowledge.
- Various improvements of NRPA.
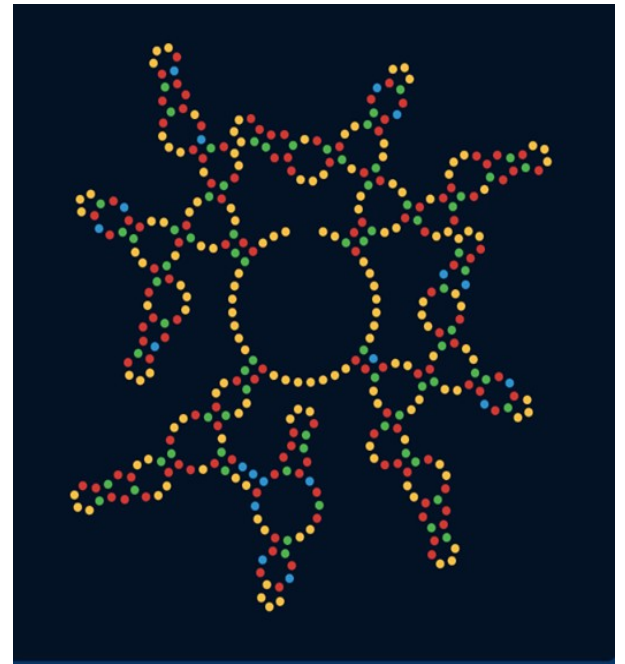
Playout Policy Adaptation

Offline learning of a playout policy

- Offline learning of playout policies has given good results in Go [Coulom 2007, Huang 2010] and Hex [Huang 2013], learning fixed pattern weights so as to bias the playouts.

- Patterns are also used to do progressive widening in the UCT tree.

Online learning of a playout policy

- The RAVE algorithm [Gelly 2011] performs online learning of moves values in order to bias the choice of moves in the UCT tree.

- RAVE has been very successful in Go and Hex.

- A development of RAVE is to use the RAVE values to choose moves in the playouts using Pool RAVE [Rimmel 2010].

- Pool RAVE improves slightly on random playouts in Havannah and reaches 62.7% against random playouts in Go.

Online learning of a playout policy

- Move-Average Sampling Technique (MAST) is a technique used in the GGP program Cadia Player so as to bias the playouts with statistics on moves [Finnsson 2010].

- It consists of choosing a move in the playout proportionally to the exponential of its mean.

- MAST keeps the average result of each action over all simulations.

Online learning of a playout policy

- Later improvements of Cadia Player are N-Grams and the last good reply policy [Tak 2012].

- They have been applied to GGP so as to improve playouts by learning move sequences.

- A recent development in GGP is to have multiple playout strategies and to choose the one which is the most adapted to the problem at hand [Swiechowski 2014].

Online learning of a playout policy

- Playout Policy Adaptation (PPA) also uses Gibbs sampling.

- The evaluation of an action for PPA is not its mean over all simulations such as in MAST.

- Instead the value of an action is learned comparing it to the other available actions for the state where it has been played.

Playout Policy learning

- Start with a uniform policy.

- Use the policy for the playouts.

- Adapt the policy for the winner of each playout.

Playout Policy learning

- Each move is associated to a weight w_i.

- During a playout each move is played with a probability :

$$\exp(w_i) / \Sigma_k \exp(w_k)$$

Playout Policy learning

- Online learning :

- For each move of the winner :

$w_i = w_i + 1$

- For each possible move of each state of the winner :

$w_i = w_i - \exp(w_i) / \Sigma_k \exp(w_k)$

Breakthrough
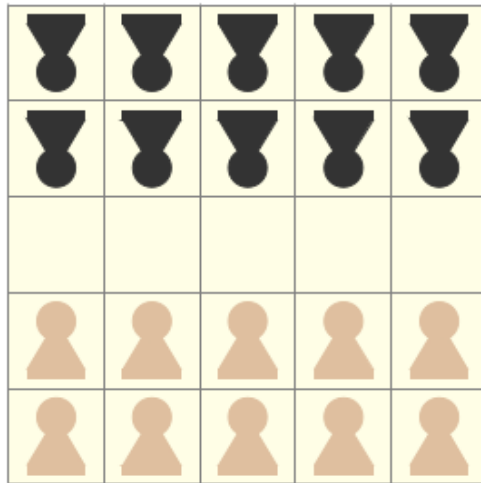
(a) Starting position on size 5 × 5.
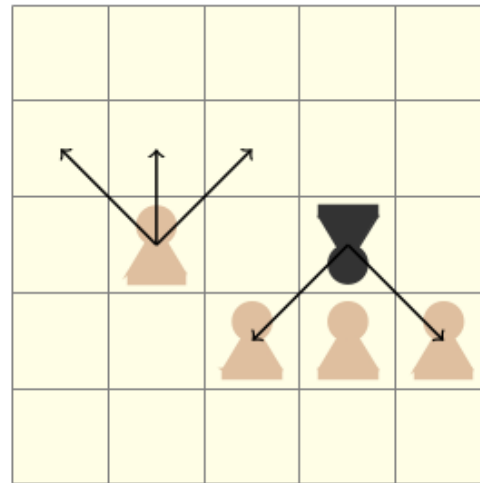
(b) Possible movements.

- The first player to reach the opposite line has won

Misère Breakthrough
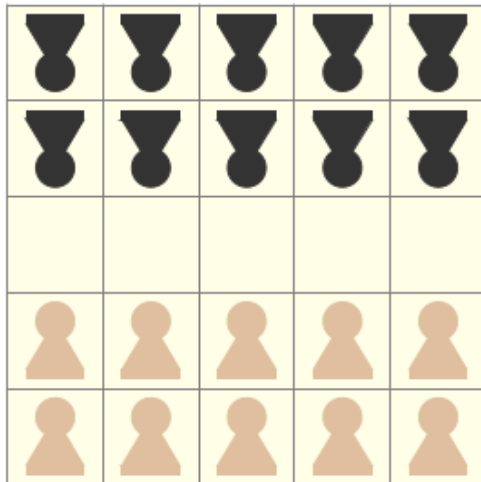
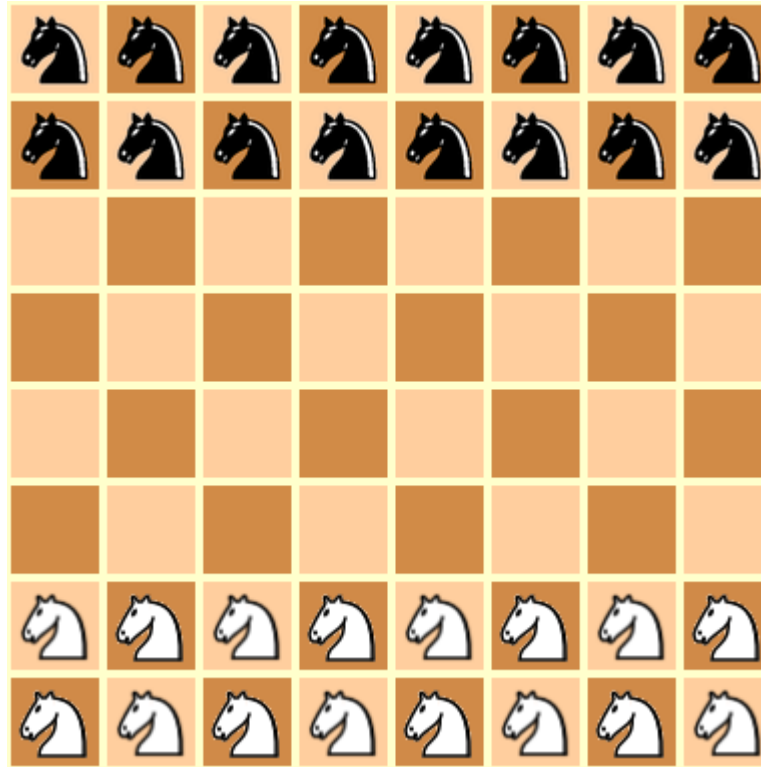(a) Starting position on size 5 × 5.
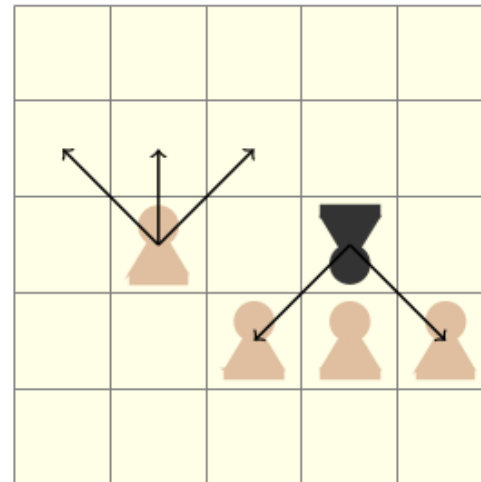
(b) Possible movements.

- The first player to reach the opposite line has lost
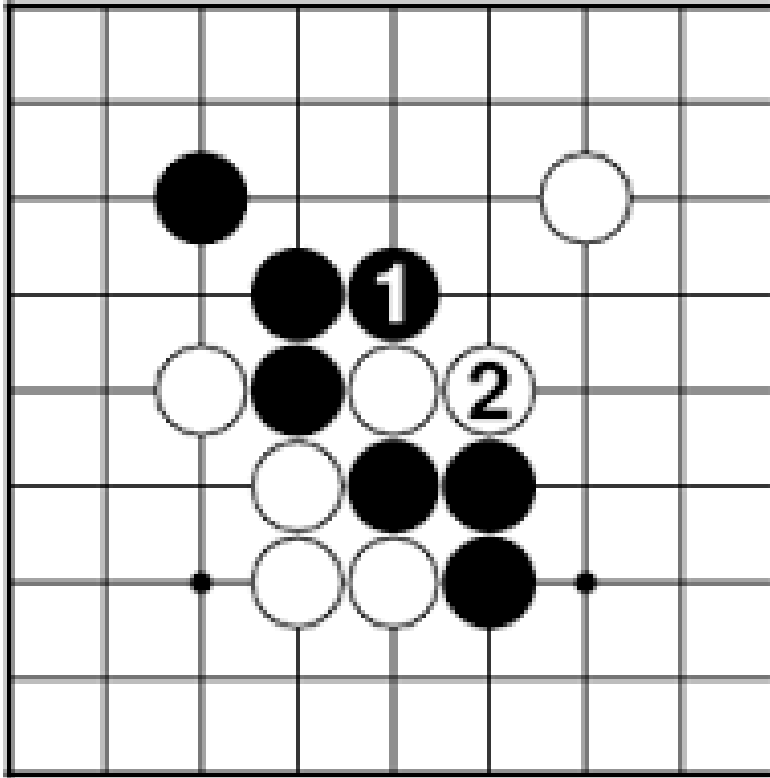
Knightthrough
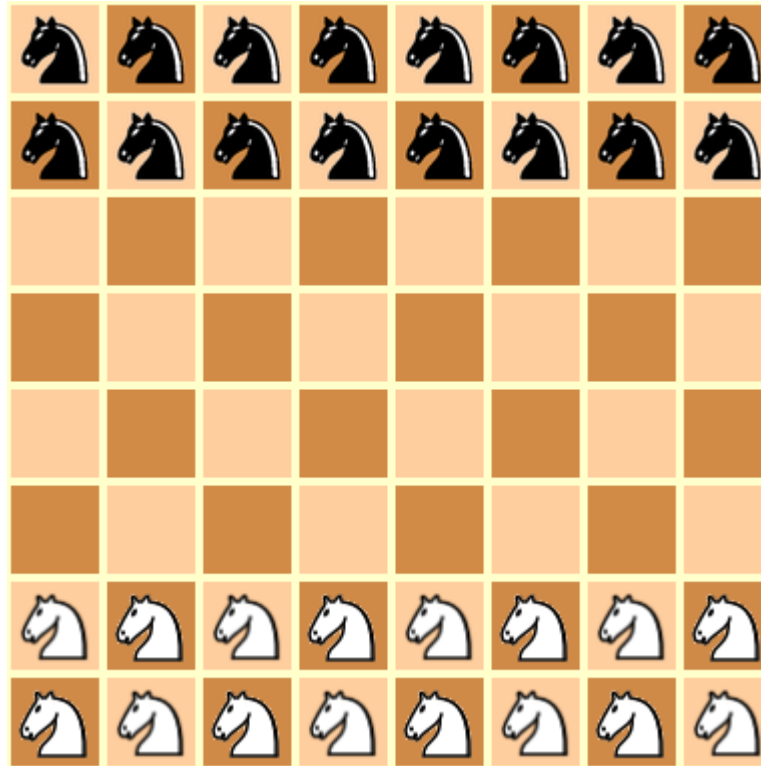
- The first to put a knight on the opposite side has won.
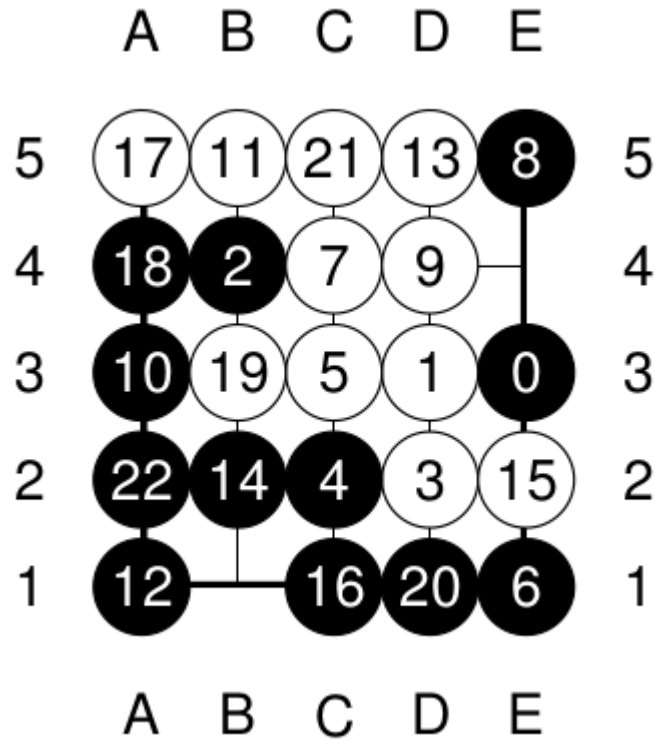
Misère Knightthrough

- The first to put a knight on the opposite side has lost.
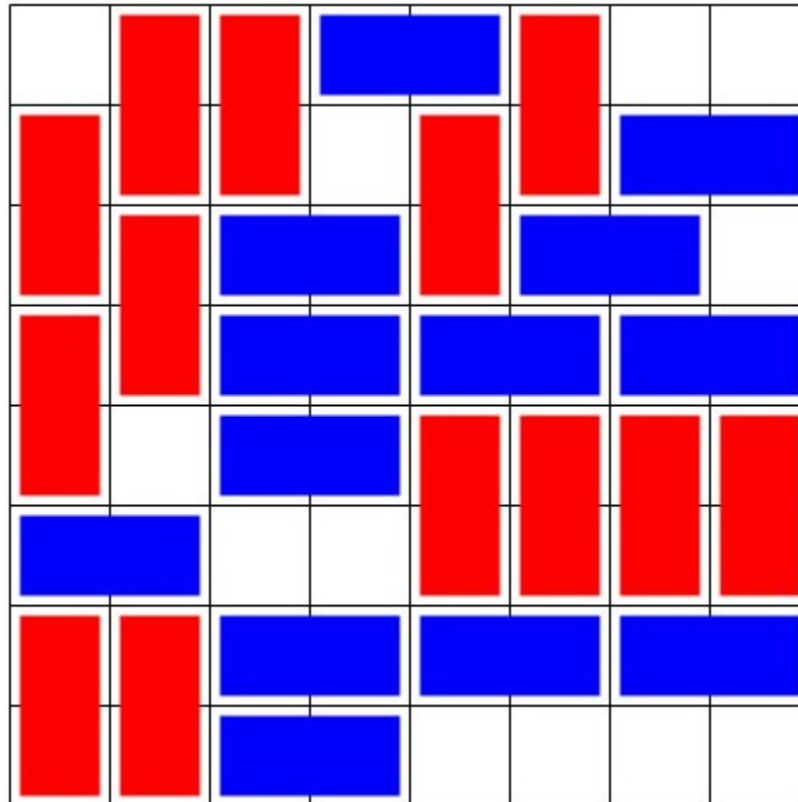
Atarigo

- The first to capture has won

Nogo

- The first to capture has lost

Domineering
Misère Domineering

- The last to play has won / lost.

Experimental results

	Size	Playouts	
		1,000	10,000
Atarigo	8 x 8	72.2	94.4
Breakthrough	8 x 8	55.2	54.4
Misere Breakthrough	8 x 8	99.2	97.8
Domineering	8 x 8	48.4	58.0
Misere Domineering	8 x 8	76.4	83.4
Go	8 x 8	23.0	1.2
Knightthrough	8 x 8	64.2	64.6
Misere Knightthrough	8 x 8	99.8	100.0
Nogo	8 x 8	64.8	46.4
Misere Nogo	8 x 8	80.6	89.4

Playout Policy learning with Move Features

- Associate features to the move.

- A move and its features are associated to a code.

- The algorithm learns the weights of codes instead of simply the weights of moves.

Playout Policy learning with Move Features

- Atarigo : four adjacent intersections
- Breakthrough : capture in the move code
- Misère Breakthrough : same as Breakthrough
- Domineering : cells next to the domino played
- Misère Domineering : same as Domineering
- Knightthrough : capture in the move code
- Misère Knighthrough : same as Knighthrough
- Nogo : same as Atarigo

Experimental results

- Each result is the outcome of a 500 games match, 250 with White and 250 with Black.

- UCT with an adaptive policy (PPAF) is played against UCT with a random policy.

- Tests are done for 10,000 playouts.

- For each game we test size 8x8.

- We tested 8 different games.

Experimental results

	Size	Winning %
Atarigo	8 x 8	94.4 %
Breakthrough	8 x 8	81.4 %
Misere Breakthrough	8 x 8	100.0 %
Domineering	8 x 8	80.4 %
Misere Domineering	8 x 8	93.0 %
Knightthrough	8 x 8	84.0 %
Misere Knightthrough	8 x 8	100.0 %
Nogo	8 x 8	95.4 %

PPAF and Memorization

- Start a game with an uniform policy.

- Adapt at each move of the game.

- Start at each move with the policy of the previous move.

PPAF and Memorization

- A nice property of PPAF is that the move played after the algorithm has been run is the most simulated move.

- The memorized policy is related to the state after the move played by the algorithm since it is the most simulated move.

- When starting with the memorized policy for the next state, this state has already been partially learned

PPAFM versus PPAF uniform

Game	Score
Atarigo	66.0%
Breakthrough	87.4%
Domineering	58.0%
Knightthrough	84.6%
Misere Breakthrough	97.2%
Misere Domineering	56.8%
Misere Knightthrough	99.2%
Nogo	49.4%

PPAFM versus UCT

Game	Score
Atarigo	95.4%
Breakthrough	94.2%
Domineering	81 .8%
Knightthrough	96.6%
Misere Breakthrough	100.0%
Misere Domineering	95.8%
Misere Knightthrough	100.0%
Nogo	91.6%

PPA Adapt Algorithm

Algorithm 4 The PPA adapt algorithm

---

adapt ($winner$, $board$, $player$, $playout$, $policy$)
$polp \leftarrow policy$
for $move$ in $playout$ **do**
  if $winner = player$ **then**
    $polp\ [\text{code}(move)] \leftarrow polp\ [\text{code}(move)] + \alpha$
    $z \leftarrow 0.0$
    for m in possible moves on $board$ **do**
      $z \leftarrow z + \exp\ (policy\ [\text{code}(m)])$
    end for
    for m in possible moves on $board$ **do**
      $polp\ [\text{code}(m)] \leftarrow polp\ [\text{code}(m)] - \alpha * \frac{exp(policy[code(m)])}{z}$
    end for
  end if
  play ($board$, $move$)
  $player \leftarrow$ opponent ($player$)
end for
$policy \leftarrow polp$

Exercise

- Try PPA for Misere Breakthrough.
    - The playout function
    - The adapt function
    - Combination with UCT
- Take capture into account (PPAF).
- Memorize the policy (PPAFM).
- Compare to UCT.

Exercise

- Modify GRAVE to incorporate a policy and a bias.

- Use the AMAF statistics of the root node of GRAVE to bias the playouts as in GNRPA.

- Update the Adapt to take the bias into account.
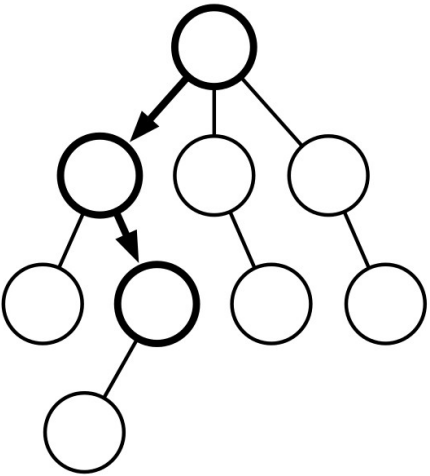
- Write the main function that calls GRAVEPolicyBias and updates the policy.

Outline

- Algorithm for solving games

- GRAVE and PPAF

- Monte Carlo move ordering

- Experiments
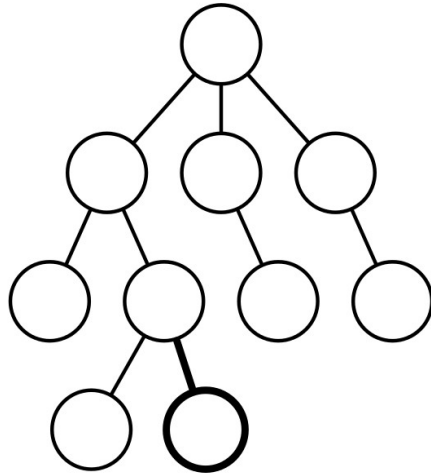
- Conclusion

Solving Games

- Proof-Number Search (PN)

- PN^2

- Alpha-Beta

- Iterative Deepening Alpha-Beta

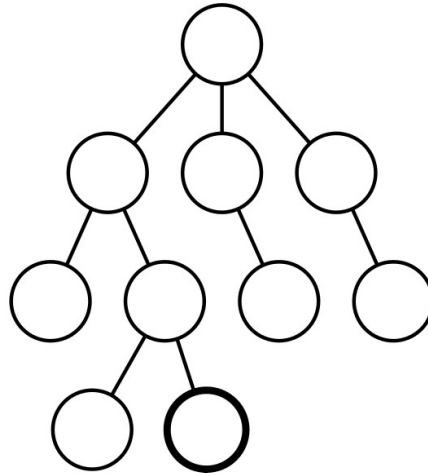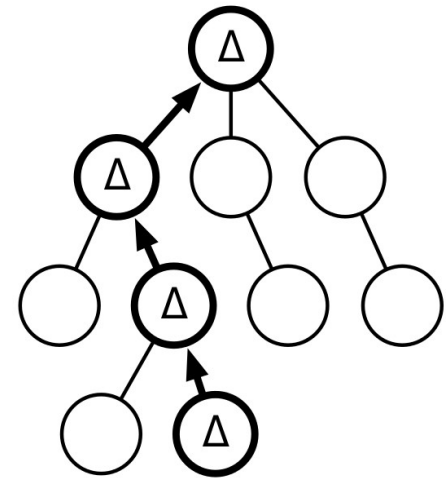- Retrograde Analysis

UCT

Selection

Expansion

Sampling

Backpropagation

Tree Policy

Default Policy

RAVE

- A big improvement for Go, Hex and other games is Rapid Action Value Estimation (RAVE) [Gelly and Silver 2007].

- RAVE combines the mean of the playouts that start with the move and the mean of the playouts that contain the move (AMAF).

RAVE

- Parameter β_m for move m is :

  $\beta_m \leftarrow pAMAF_m / (pAMAF_m + p_m + bias \times pAMAF_m \times p_m)$

- β_m starts at 1 when no playouts and decreases as more playouts are played.

- Selection of moves in the tree :

  $argmax_m((1.0 - \beta_m) \times mean_m + \beta_m \times AMAF_m)$

GRAVE

- Generalized Rapid Action Value Estimation (GRAVE) is a simple modification of RAVE.

- It consists in using the first ancestor node with more than n playouts to compute the RAVE values.

- It is a big improvement over RAVE for Go, Atarigo, Knightthrough and Domineering [Cazenave 2015].

Playout Policy learning

- Start with a uniform policy.

- Use the policy for the playouts.

- Adapt the policy for the winner of each playout.

Playout Policy learning

- Each move is associated to a weight w_i.

- During a playout each move is played with a probability :

$$\exp(w_i) / \Sigma_k \exp(w_k)$$

Playout Policy learning

- Online learning :

- For each move of the winner :

  $w_i = w_i + 1$

- For each possible move of each state of the winner :

  $w_i = w_i - \exp(w_i) / \Sigma_k \exp(w_k)$

Monte Carlo Game Solver

- Use the order of moves of GRAVE when the state is in the GRAVE tree.

- Use the order of moves of Playout Policy Adaptation when the state is outside the GRAVE tree.

Table 1: Different algorithms for solving Atarigo.

| Size	5×5	
Result	Won	
	Move count	Time
PN^2	14 784 088 742	37 901.56 s.
ID $\alpha\beta$ TT	> 35 540 000 000	> 86 400.00 s.
$\alpha\beta$ TT	> 37 660 000 000	> 86 400.00 s.
ID $\alpha\beta$ TT MC	62 800 334	126.84 s.
$\alpha\beta$ TT MC	**3 956 049**	**12.79 s.**

Size	6×5	
Result	Won	
	Move count	Time
PN^2	> 33 150 000 000	> 86 400.00 s.
ID $\alpha\beta$ TT	> 37 190 000 000	> 86 400.00 s.
$\alpha\beta$ TT	> 7 090 000 000	> 44 505.91 s.
ID $\alpha\beta$ TT MC	12 713 931 627	27 298.35 s.
$\alpha\beta$ TT MC	**329 780 434**	**787.17 s.**

Table 2: Different algorithms for solving Nogo.

Size	7×3	
Result	Won	
	Move count	Time
PN^2	> 80 390 000 000	> 86 400.00 s.
ID $\alpha\beta$ TT	10 921 978 839	12 261.64 s.
$\alpha\beta$ TT	3 742 927 598	4 412.21 s.
ID $\alpha\beta$ TT MC	1 927 635 856	2 648.91 s.
$\alpha\beta$ TT MC	**35 178 886**	**49.72 s.**

Size	5×4	
Result	Won	
	Move count	Time
PN^2	> 101 140 000 000	> 86 400.00 s.
ID $\alpha\beta$ TT	1 394 182 870	1 573.72 s.
$\alpha\beta$ TT	1 446 922 704	1 675.64 s.
ID $\alpha\beta$ TT MC	73 387 083	134.26 s.
$\alpha\beta$ TT MC	**33 850 535**	**74.77 s.**

Fig. 1: Solution of Nogo 5 × 5.
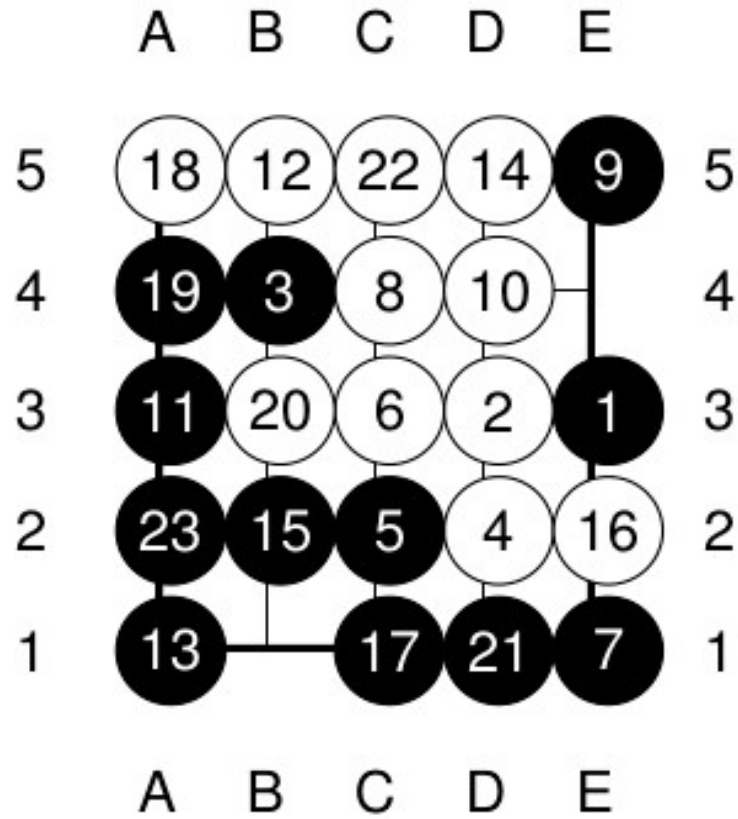
	1	2	3	4	5	6	7	8	9	10
1	2	1	1	2	1	1	1	1	1	1
2	1	1	2	2	1	1	1	1	2	2
3	1	2	1	2	1	1	1	1		
4	2	2	2	2	1	1				
5	1	1	1	1	1					
6	1	1	1	1						
7	1	1	1							
8	1	1	1							
9	1	2								
10	1	2								

Table 3: Winner for Nogo boards of various sizes

Table 4: Different algorithms for solving Go.

Size	3×3	
Result	Won	
	Move count	Time
PN^2	246 394	3.72 s.
ID $\alpha\beta$ TT	840 707	11.34 s.
$\alpha\beta$ TT	420 265	11.50 s.
ID $\alpha\beta$ TT MC	375 414	5.62 s.
$\alpha\beta$ TT MC	**6 104**	**0.16 s.**

Size	4×3	
Result	Won	
	Move count	Time
PN^2	43 202 038	619.98 s.
ID $\alpha\beta$ TT	39 590 950	515.71 s.
$\alpha\beta$ TT	107 815 563	1 977.86 s.
ID $\alpha\beta$ TT MC	22 382 730	348.08 s.
$\alpha\beta$ TT MC	**4 296 893**	**96.63 s.**

Table 5: Different algorithms for solving Breakthrough.

Size	5×5	
Result	Lost	
	Move count	Time
PN^2	$> 38\,780\,000\,000$	$> 86\,400.00$ s.
ID $\alpha\beta$ TT	$13\,083\,392\,799$	$33\,590.59$ s.
$\alpha\beta$ TT	$19\,163\,127\,770$	$43\,406.79$ s.
ID $\alpha\beta$ TT MC	$3\,866\,853\,361$	$11\,319.39$ s.
$\alpha\beta$ TT MC	**$3\,499\,173\,137$**	**$9\,243.66$ s.**

Table 6: Different algorithms for solving Misere Breakthrough.

Size	4×5	
Result	Lost	
	Move count	Time
PN^2	> 42 630 000 000	> 86 400 s.
ID $\alpha\beta$ TT	> 43 350 000 000	> 86 400 s.
$\alpha\beta$ TT	> 42 910 000 000	> 86 400 s.
ID $\alpha\beta$ TT MC	1 540 153 635	3 661.50 s.
$\alpha\beta$ TT MC	**447 879 697**	**1 055.32 s.**

Table 7: Different algorithms for solving Knightthrough.

Size	6×6	
Result	Won	
	Move count	Time
PN^2	$> 33\,110\,000\,000$	$> 86\,400$ s.
ID $\alpha\beta$ TT	$1\,153\,730\,169$	$4\,894.69$ s.
$\alpha\beta$ TT	$2\,284\,038\,427$	$6\,541.08$ s.
ID $\alpha\beta$ TT MC	**17 747 503**	**102.60 s.**
$\alpha\beta$ TT MC	$528\,783\,129$	$1\,699.01$ s.

Size	7×6	
Result	Won	
	Move count	Time
PN^2	$> 30\,090\,000\,000$	$> 86\,400$ s.
ID $\alpha\beta$ TT	$> 17\,500\,000\,000$	$> 86\,400$ s.
$\alpha\beta$ TT	$> 29\,980\,000\,000$	$> 86\,400$ s.
ID $\alpha\beta$ TT MC	**2 540 383 012**	**13 716.36 s.**
$\alpha\beta$ TT MC	$6\,650\,804\,159$	$23\,958.04$ s.

Table 8: Different algorithms for solving Misere Knightthrough.

Size	5×5	
Result	Lost	
	Move count	Time
PN^2	> 45 290 000 000	> 86 400 s.
ID $\alpha\beta$ TT	> 52 640 000 000	> 86 400 s.
$\alpha\beta$ TT	> 56 230 000 000	> 86 400 s.
ID $\alpha\beta$ TT MC	> 41 840 000 000	> 86 400 s.
$\alpha\beta$ TT MC	**20 375 687 163**	**42 425.41 s.**

Table 9: Different algorithms for solving Domineering.

Size	7×7	
Result	Won	
	Move count	Time
PN^2	$> 41\,270\,000\,000$	$> 86\,400$ s.
ID $\alpha\beta$ TT	$18\,958\,604\,687$	$35\,196.62$ s.
$\alpha\beta$ TT	$197\,471\,137$	376.23 s.
ID $\alpha\beta$ TT MC	$2\,342\,641\,133$	$5\,282.06$ s.
$\alpha\beta$ TT MC	**$29\,803\,373$**	**123.76 s.**

Table 10: Different algorithms for solving Misere Domineering.

Size	7×7	
Result	Won	
	Move count	Time
PN^2	$> 44\,560\,000\,000$	$> 86\,400$ s.
ID $\alpha\beta$ TT	$> 49\,290\,000\,000$	$> 86\,400$ s.
$\alpha\beta$ TT	$> 49\,580\,000\,000$	$> 86\,400$ s.
ID $\alpha\beta$ TT MC	$7\,013\,298\,932$	$14\,936.03$ s.
$\alpha\beta$ TT MC	**$72\,728\,678$**	**212.25 s.**

Conclusion

- For the games we solved, Misere Games are more difficult to solve than normal games.

- In Misere Games the player waits and tries to force the opponent to play a losing move.

- This makes the game longer and reduces the number of winning sequences and winning moves.

- Monte Carlo Move Ordering improves much the speed of αβ with transposition table compare to depth first αβ and Iterative Deepening αβ with transposition table but without Monte Carlo Move Ordering.

- The experimental results show significant improvements for nine different games.

Zero Learning

Deep Reinforcement Learning

- AlphaGo
- Golois
- AlphaGo Zero
- Alpha Zero
- Mu Zero
- Polygames

David Silver   Aja Huang

AlphaGo

Fan Hui is the european Go champion and a 2p
 professional Go player :

AlphaGo Fan won 5-0
against Fan Hui in
November 2015.

Nature, January 2016.

AlphaGo

Lee Sedol is among the strongest and most famous 9p Go player :

AlphaGo Lee won 4-1 against Lee Sedol in march 2016.

AlphaGo

Ke Jie is the world champion of Go according to Elo ratings :

AlphaGo Master won 3-0 against Ke Jie in may 2017.

AlphaGo

- AlphaGo combines MCTS and Deep Learning.

- There are four phases to the development of AlphaGo :

- Learn strong players moves => policy network.

- Play against itself and improve the policy network => reinforcement learning.

- Learn a value network to evaluate states from millions of games played against itself.

- Combine MCTS, policy and value network.

AlphaGo
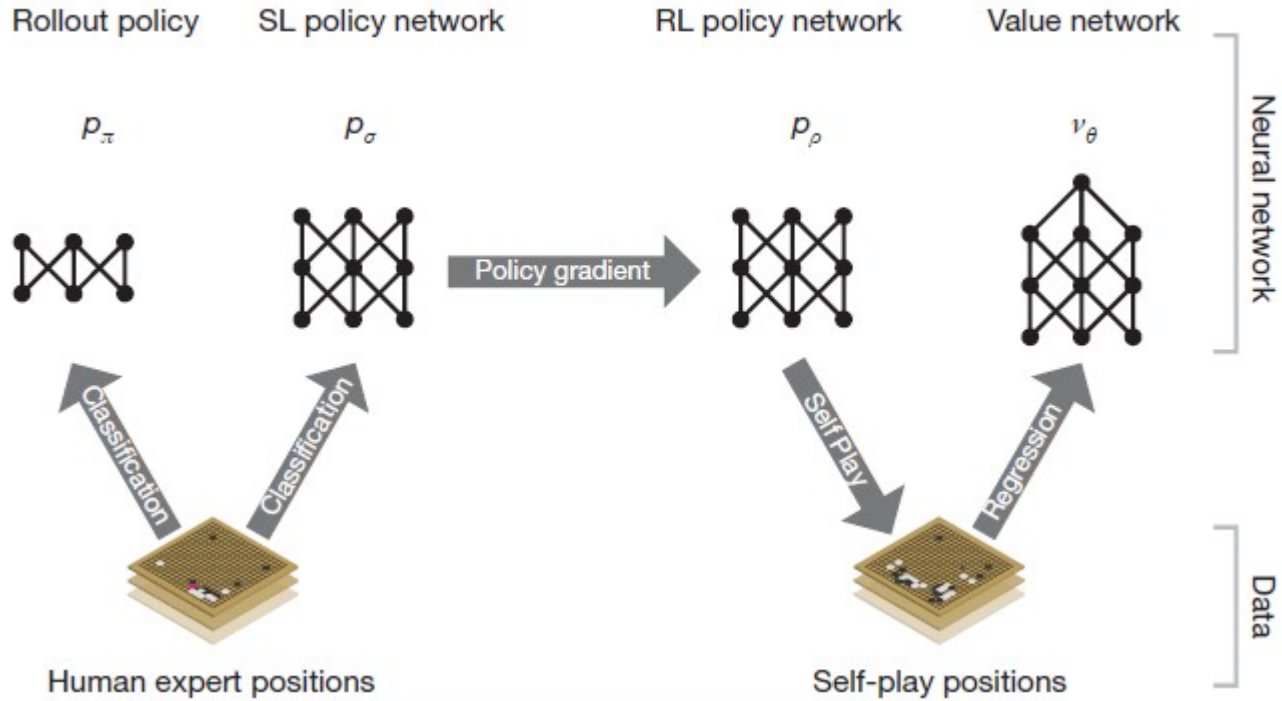
AlphaGo

b

Policy network   Value network

$p_{\sigma|\rho}\,(a|s)$ $\nu_\theta\,(s')$

s s'
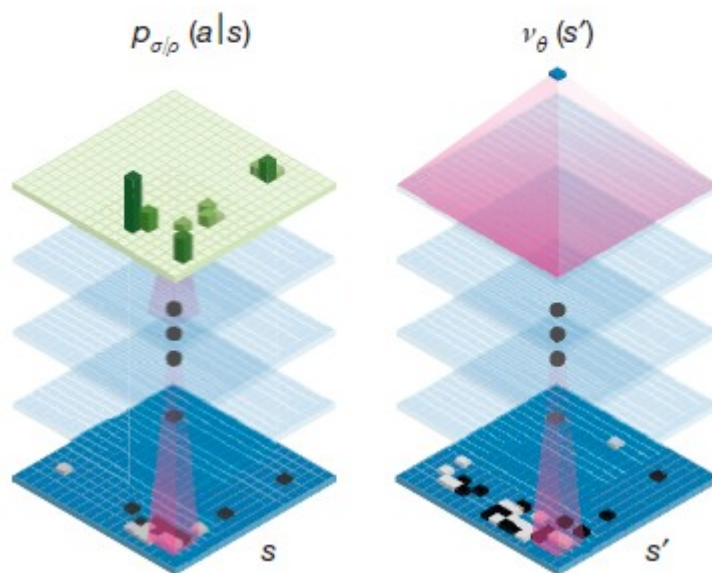
AlphaGo

- The policy network is a 13 layers network.

- It uses either 128 or 256 feature planes.

- It is fully convolutional.

- It learns to predict moves from hundreds of thousands of strong players games.

- Once it has learned, it finds the strong player move 57.0 % of the time.

- It takes 3 ms to run.

AlphaGo

- The value network is also a deep convolutional neural network.

- AlphaGo played a lot of games and kept for each game a state and the corresponding terminal state.

- It learns to evaluate states with the result of the terminal state.

- The value network has learned an evaluation function that gives the probability of winning.

AlphaGo

Figure legend:
- Uniform random rollout policy
- Fast rollout policy
- Value network
- SL policy network
- RL policy network

Y-axis: Mean squared error on expert games

X-axis: Move number

AlphaGo

a Selection   **b** Expansion   **c** Evaluation   **d** Backup
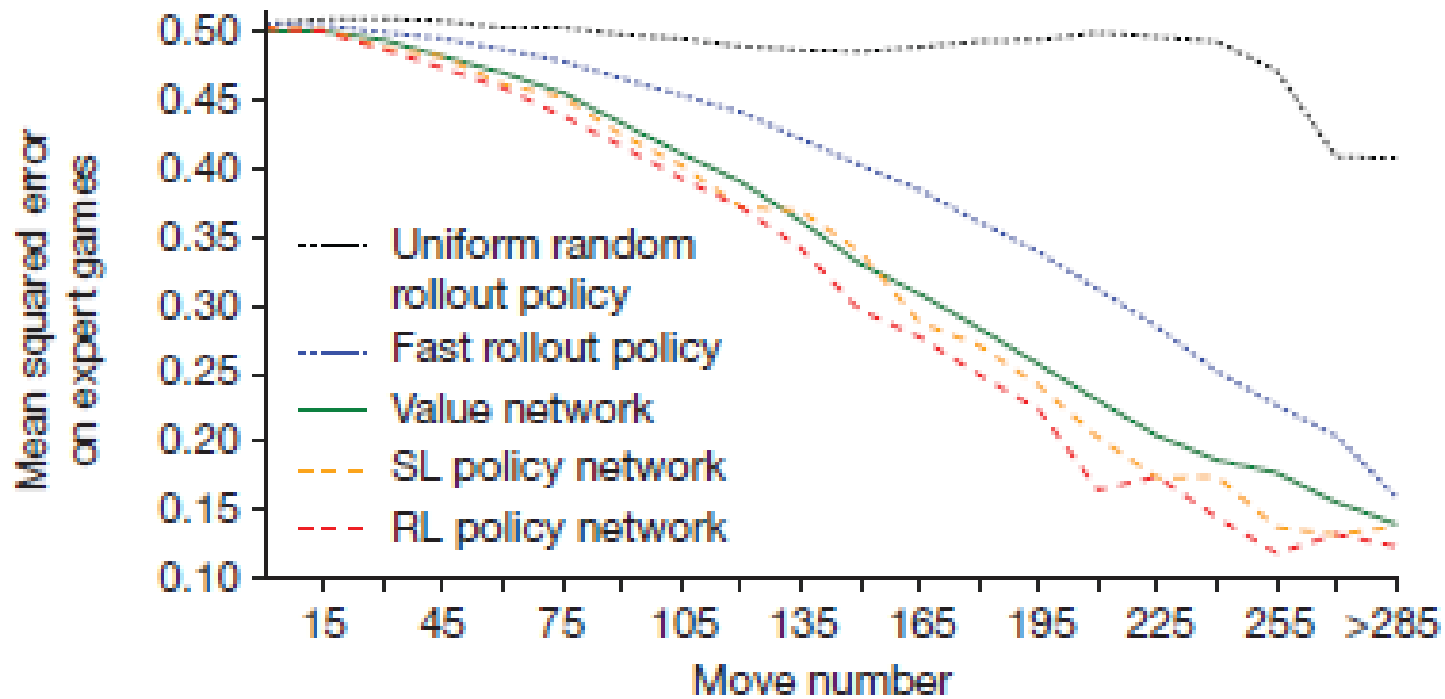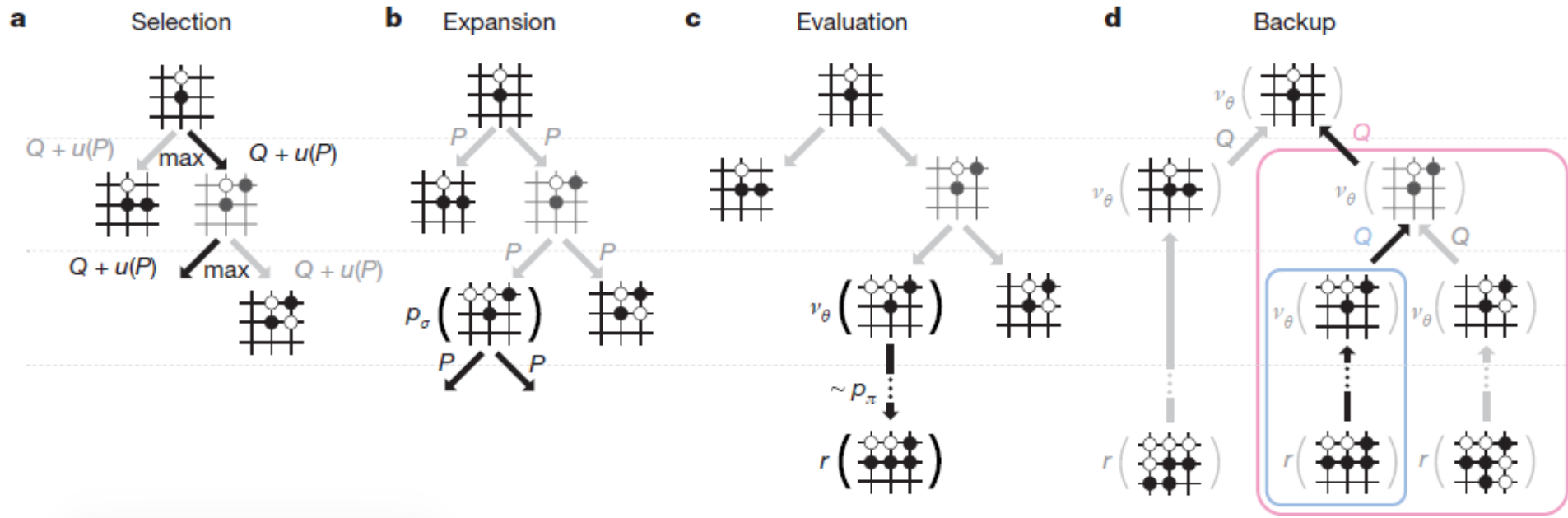
AlphaGo

- The policy network is used as a prior to consider good moves at first.

- Playouts are used to evaluate moves

- The value network is combined with the statistics of the moves coming from the playouts.

- PUCT :

$$U(s,a) = c_{\text{puct}} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$

AlphaGo

a

Figure showing AlphaGo win rate (%) versus Training accuracy on KGS dataset (%) for 128 filters, 192 filters, 256 filters, and 384 filters.
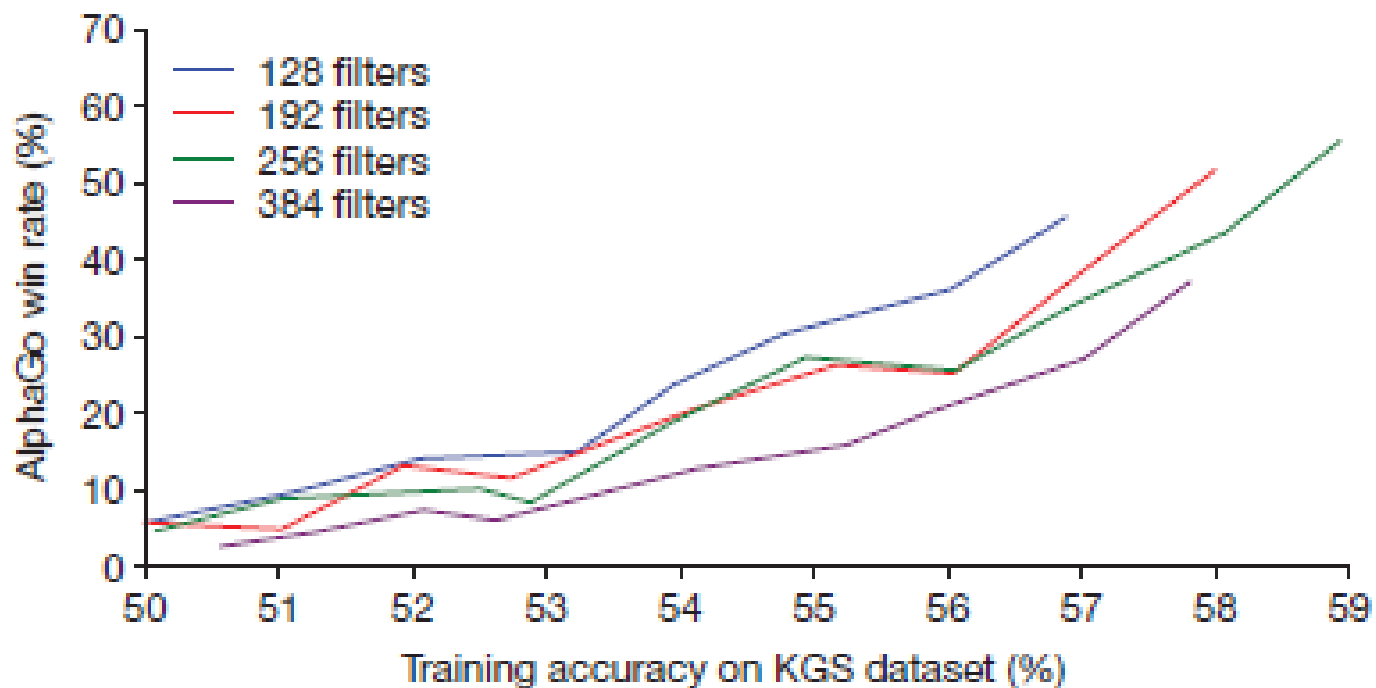
AlphaGo

- AlphaGo has been parallelized using a distributed version.

- 40 search threads, 1,202 CPUs and 176 GPU.

AlphaGo
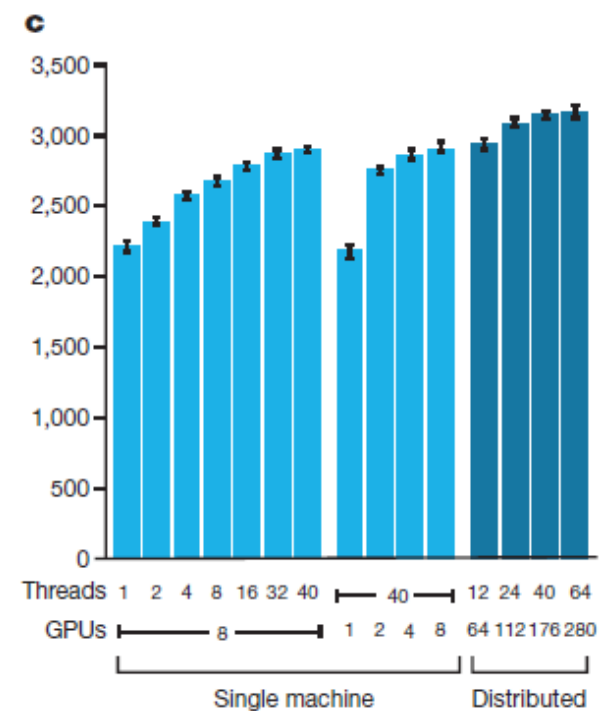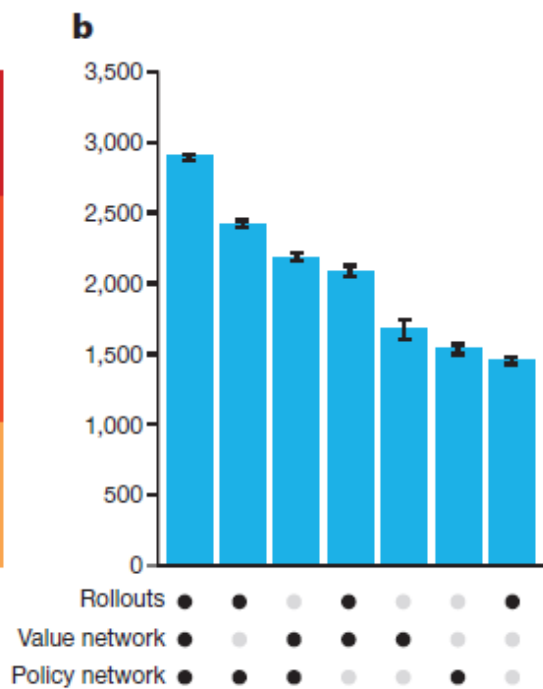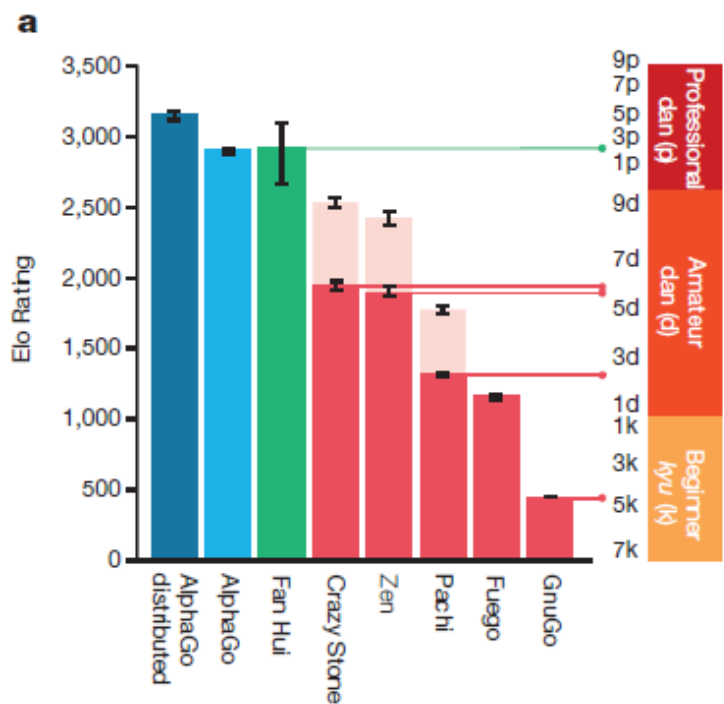
AlphaGo

Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Feature planes used by the policy network (all but last feature) and value network (all features).

AlphaGo

Extended Data Table 3 | Supervised learning results for the policy network

Architecture			Evaluation				
Filters	Symmetries	Features	Test accuracy %	Train accuracy %	Raw net wins %	*AlphaGo* wins %	Forward time (ms)
128	1	48	54.6	57.0	36	53	2.8
192	1	48	55.4	58.0	50	50	4.8
256	1	48	55.9	59.1	67	55	7.1
256	2	48	56.5	59.8	67	38	13.9
256	4	48	56.9	60.2	69	14	27.6
256	8	48	57.0	60.4	69	5	55.3
192	1	4	47.6	51.4	25	15	4.8
192	1	12	54.7	57.1	30	34	4.8
192	1	20	54.7	57.2	38	40	4.8
192	8	4	49.2	53.2	24	2	36.8
192	8	12	55.7	58.3	32	3	36.8
192	8	20	55.8	58.4	42	3	36.8

The policy network architecture consists of 128, 192 or 256 filters in convolutional layers; an explicit symmetry ensemble over 2, 4 or 8 symmetries; using only the first 4, 12 or 20 input feature planes listed in Extended Data Table 1. The results consist of the test and train accuracy on the KGS data set; and the percentage of games won by given policy network against AlphaGo's policy network (highlighted row 2): using the policy networks to select moves directly (raw wins); or using AlphaGo's search to select moves (AlphaGo wins); and finally the computation time for a single evaluation of the policy network.

AlphaGo

Extended Data Table 7 | Results of a tournament between different variants of AlphaGo

Short name	Policy network	Value network	Rollouts	Mixing constant	Policy GPUs	Value GPUs	Elo rating
α_{rvp}	p_σ	v_θ	p_π	$\lambda = 0.5$	2	6	2890
α_{vp}	p_σ	v_θ	—	$\lambda = 0$	2	6	2177
α_{rp}	p_σ	—	p_π	$\lambda = 1$	8	0	2416
α_{rv}	$[p_\tau]$	v_θ	p_π	$\lambda = 0.5$	0	8	2077
α_{v}	$[p_\tau]$	v_θ	—	$\lambda = 0$	0	8	1655
α_{r}	$[p_\tau]$	—	p_π	$\lambda = 1$	0	0	1457
α_{p}	p_σ	—	—	—	0	0	1517

Evaluating positions using rollouts only (α_{rp}, α_r), value nets only (α_{vp}, α_v), or mixing both (α_{rvp}, α_{rv}); either using the policy network p_σ (α_{rvp}, α_{vp}, α_{rp}), or no policy network (α_{rvp}, α_{vp}, α_{rp}), that is, instead using the placeholder probabilities from the tree policy p_τ throughout. Each program used 5 s per move on a single machine with 48 CPUs and 8 GPUs. Elo ratings were computed by BayesElo.

AlphaGo

Extended Data Table 8 | Results of a tournament between AlphaGo and distributed AlphaGo, testing scalability with hardware

AlphaGo	Search threads	CPUs	GPUs	Elo
Asynchronous	1	48	8	2203
Asynchronous	2	48	8	2393
Asynchronous	4	48	8	2564
Asynchronous	8	48	8	2665
Asynchronous	16	48	8	2778
Asynchronous	32	48	8	2867
Asynchronous	40	48	8	2890
Asynchronous	40	48	1	2181
Asynchronous	40	48	2	2738
Asynchronous	40	48	4	2850
Distributed	12	428	64	2937
Distributed	24	764	112	3079
Distributed	40	1202	176	3140
Distributed	64	1920	280	3168

Each program played with a maximum of 2 s thinking time per move. Elo ratings were computed by BayesElo.

Golois

Golois

- I replicated the AlphaGo experiments with the policy and value networks.

- Golois policy network scores 58.54% on the test set (57.0% for AlphaGo).

- Golois plays on the kgs internet Go server.

- It has a strong 4d ranking just with the learned policy network (AlphaGo policy network is 3d).

Data

- Learning set = games played on the KGS Go server by players being 6d or more between 2000 and 2014.

- No handicap games.

- Each position is rotated to eight possible symmetric positions.

- 160 000 000 positions in the learning set.

- Test set = games played in 2015.

- 100 000 different positions not mirrored.

Training

- In order to train the network we build minibatches of size 50 composed of 50 states chosen randomly in the training set.

- Each state is randomly mirrored to one of its eight symmetric states.

- We define an epoch as 5 000 000 training examples.

- The accuracy and the error on the test set are computed every epoch.

- The learning rate is divided by two each time the training error stalls, i.e. is greater than the previous average training error over the last epoch.

Residual Nets

- Residual Nets :
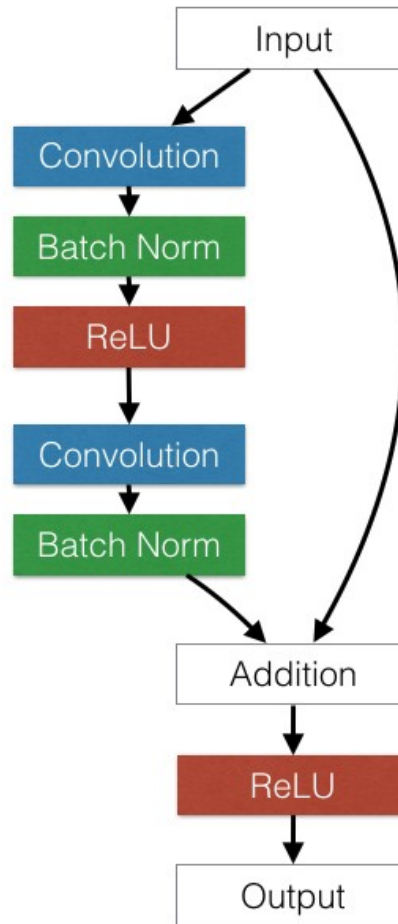
Evolution of the error
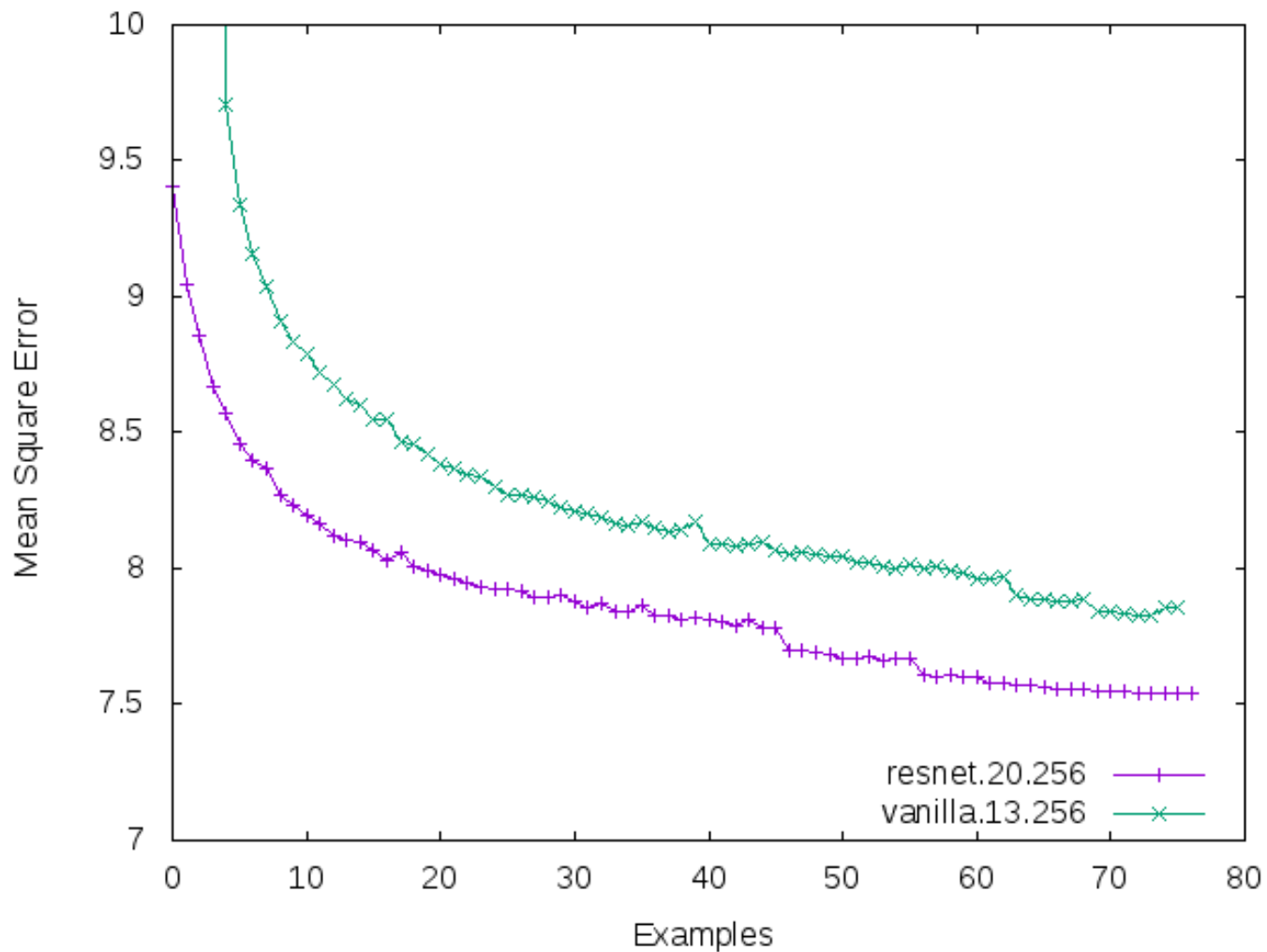
Evolution of the accuracy
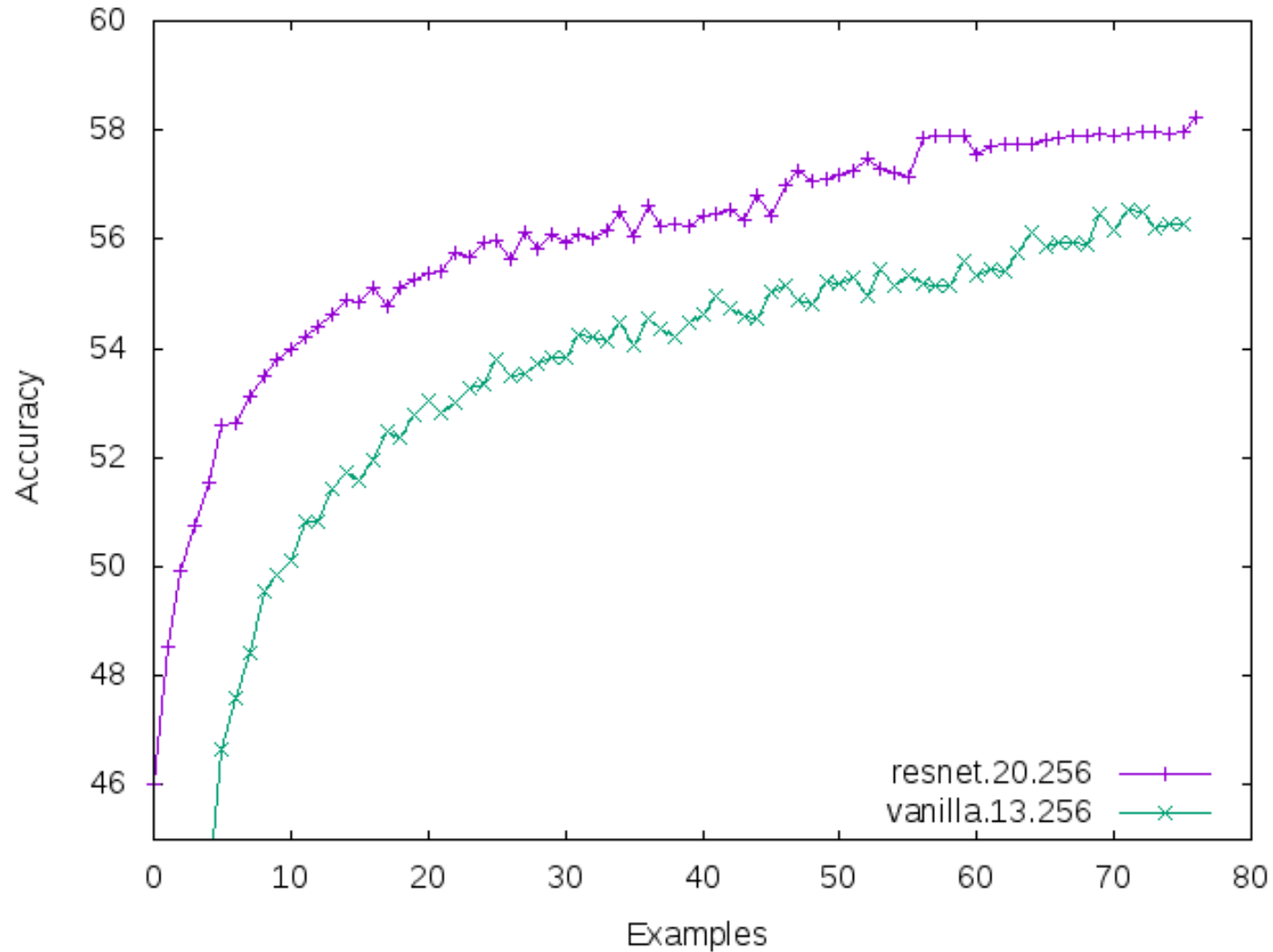
Golois Policy Network

- Using residual network enables to train deeper network.

- It enables better accuracy than AlphaGo policy network.

- It has a 4 dan level on kgs, playing moves instantly.

- Next: value network and parallel MCTS.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
         Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Usual Layer
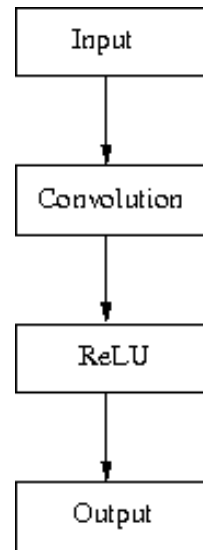
DarkForest Layer
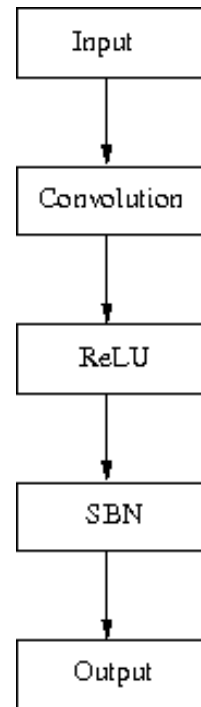
Residual Layer
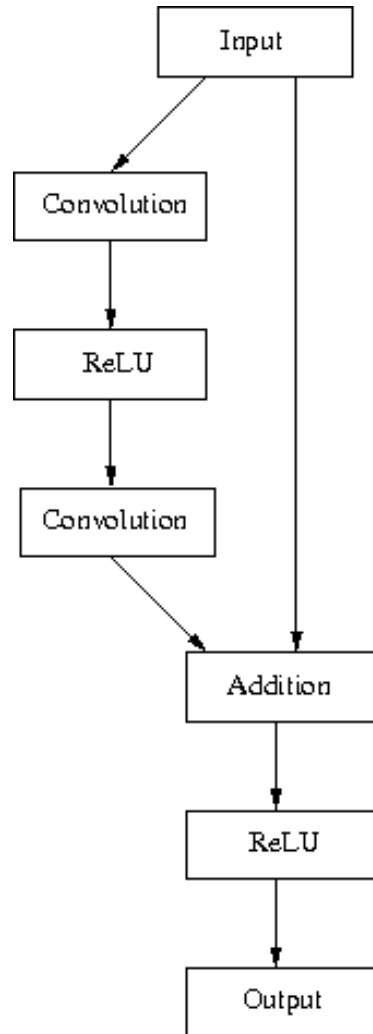
Residual Layers for Images
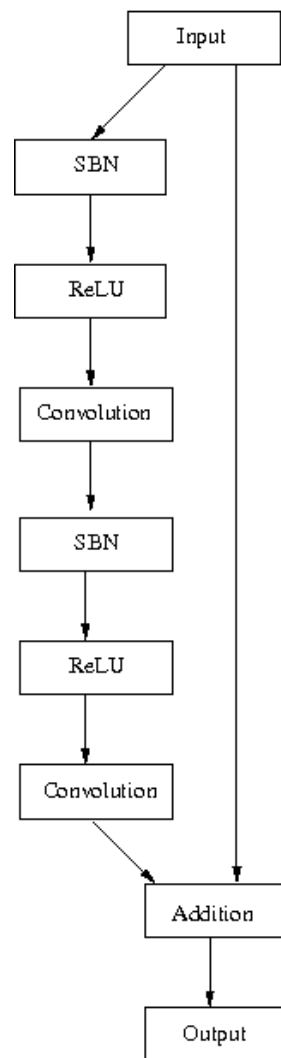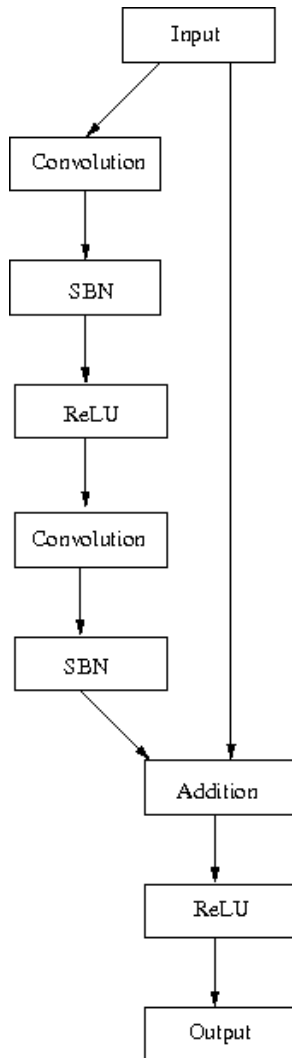
Golois Layer
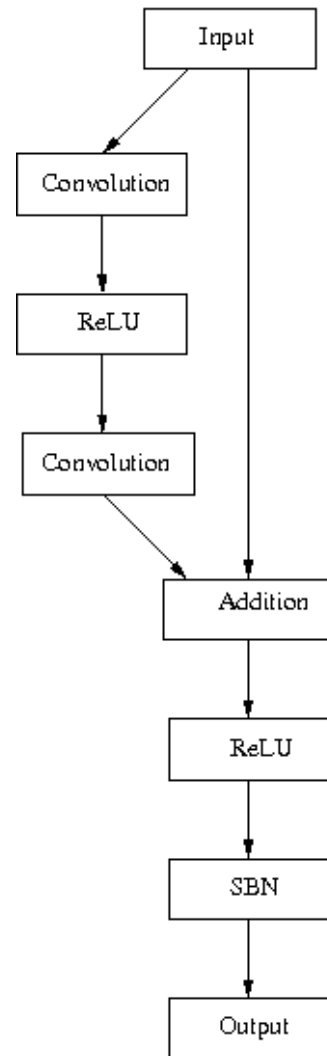
Multiple Output Planes

- In DarkForest it improves the level of play to predict the next three moves instead of only the next move.

- We compared the next move prediction and the next three moves prediction with the Golois layer.

Data

- We use the GoGoD dataset.

- It is composed of many professional games played until today.

- We used the games from 1900 to 2014 for the training set and the games from 2015 and 2016 as the test set.

- The first 500 000 positions of the test set to evaluate the error and the accuracy of the networks.

Multiple Output Planes
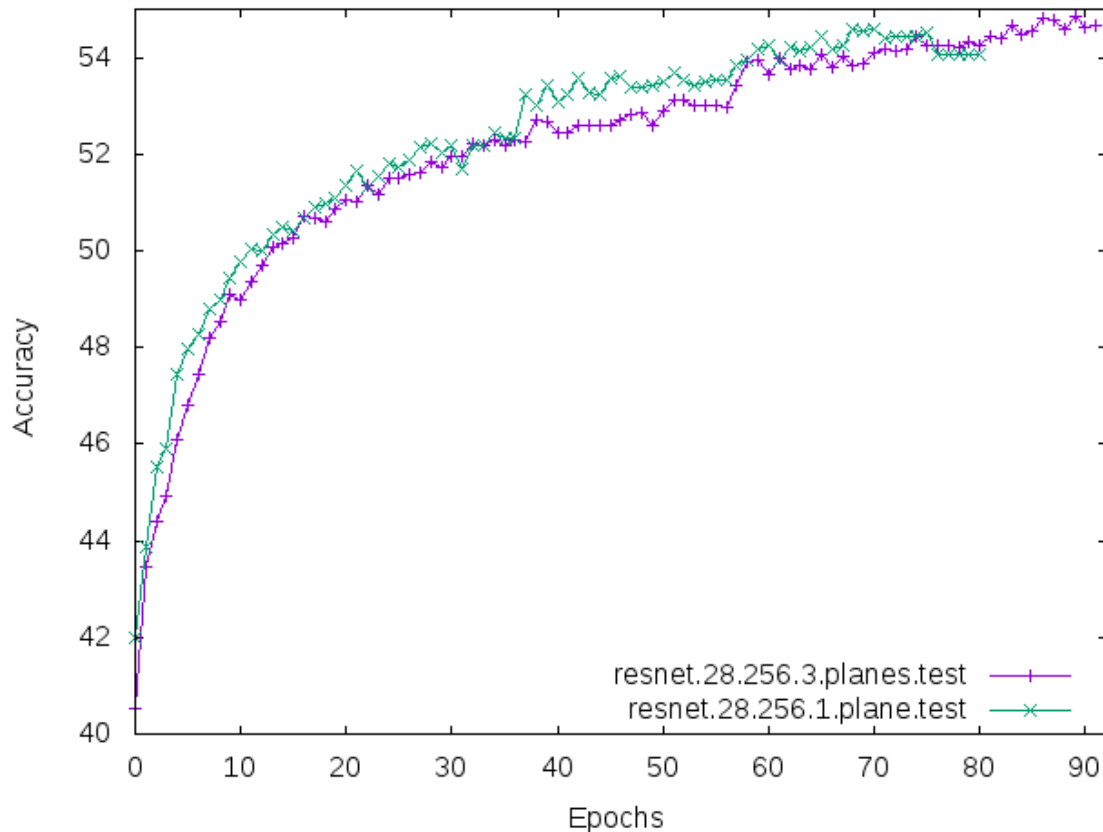
- Evolution of the accuracy for 1 and 3 output planes

Multiple Output Planes
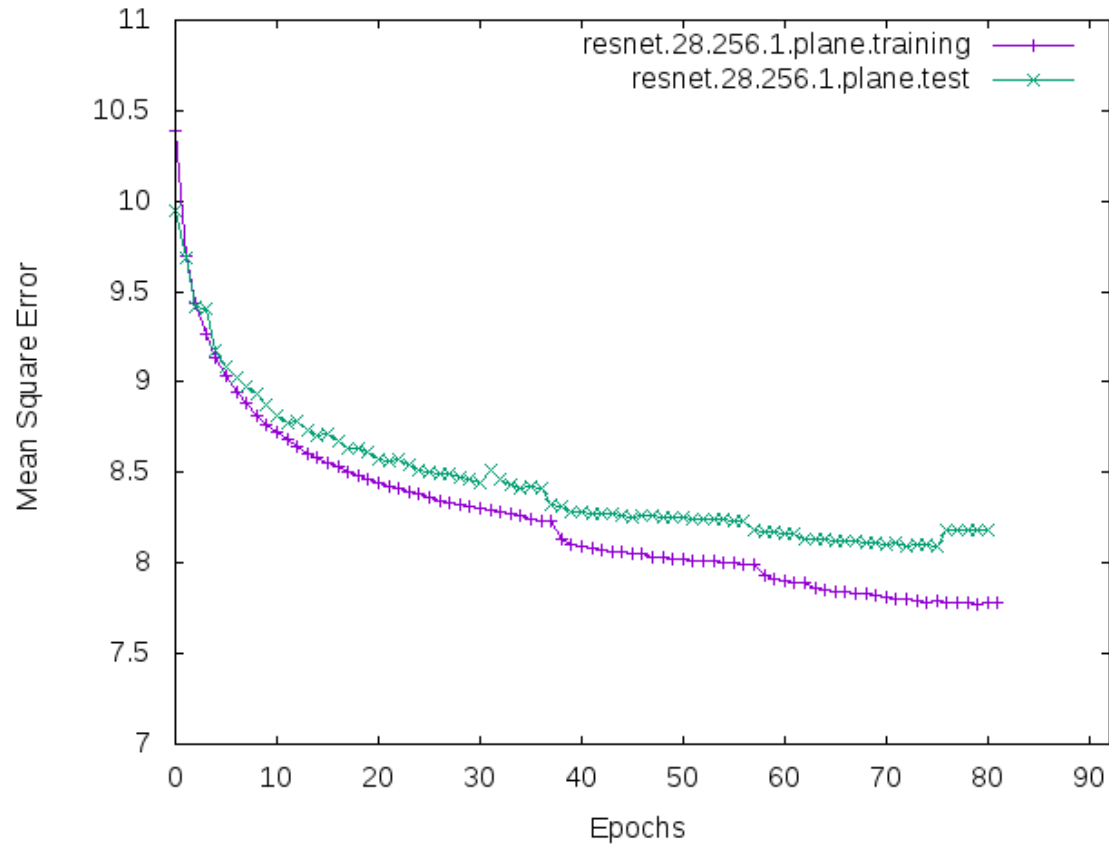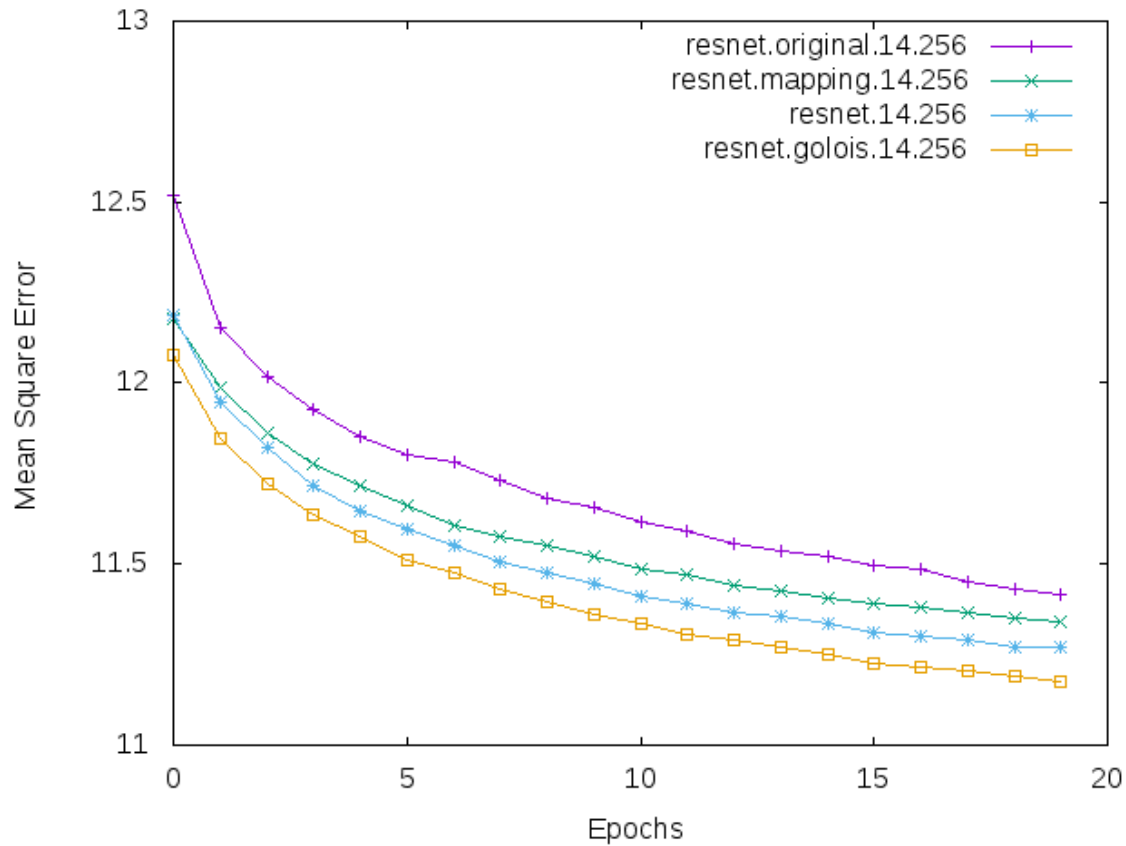

- Evolution of the errors for one output plane

Multiple Output Planes
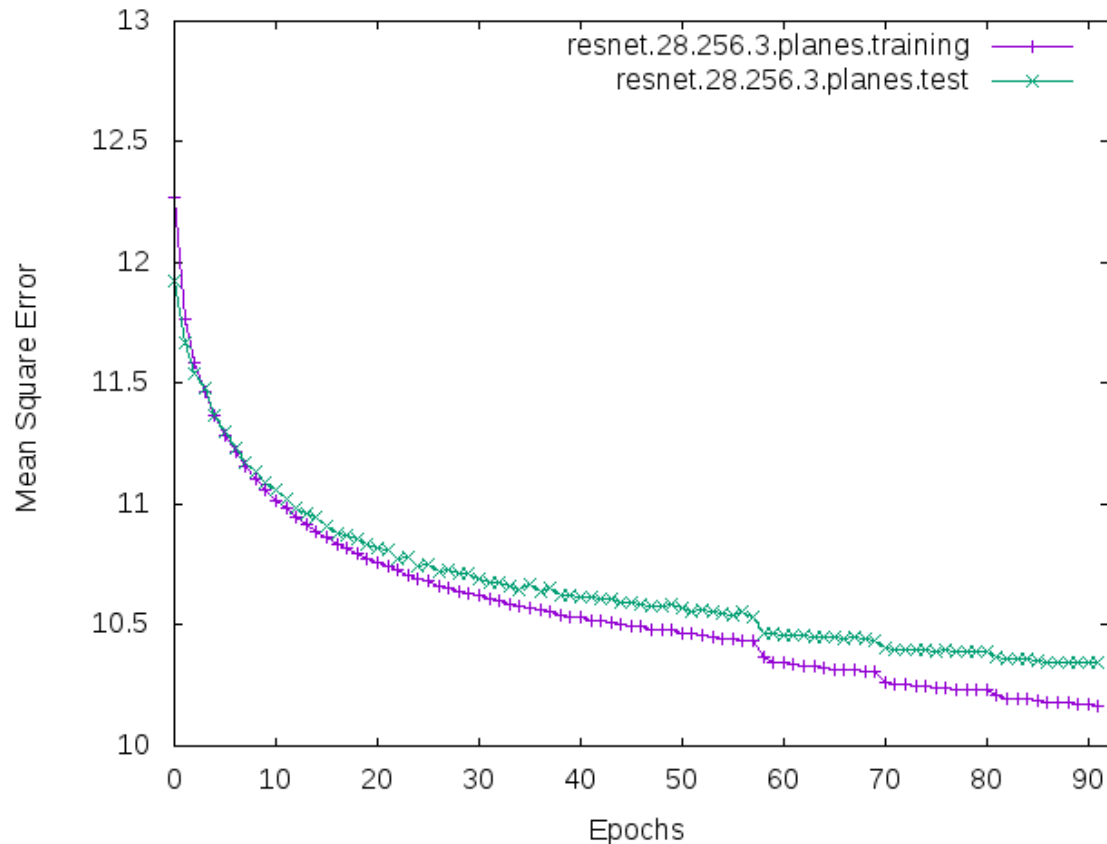
- Evolution of the errors for three output planes

Golois Layer
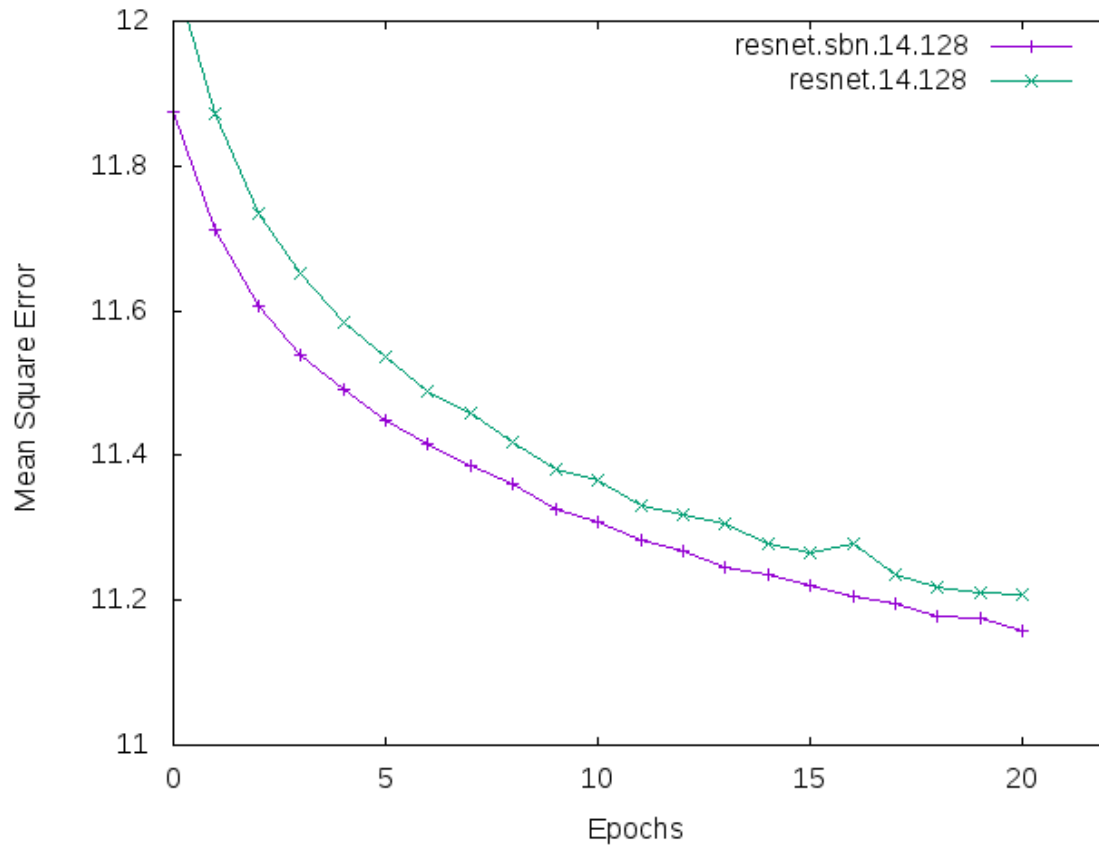
- Evolution of the error for resnet and Golois layers
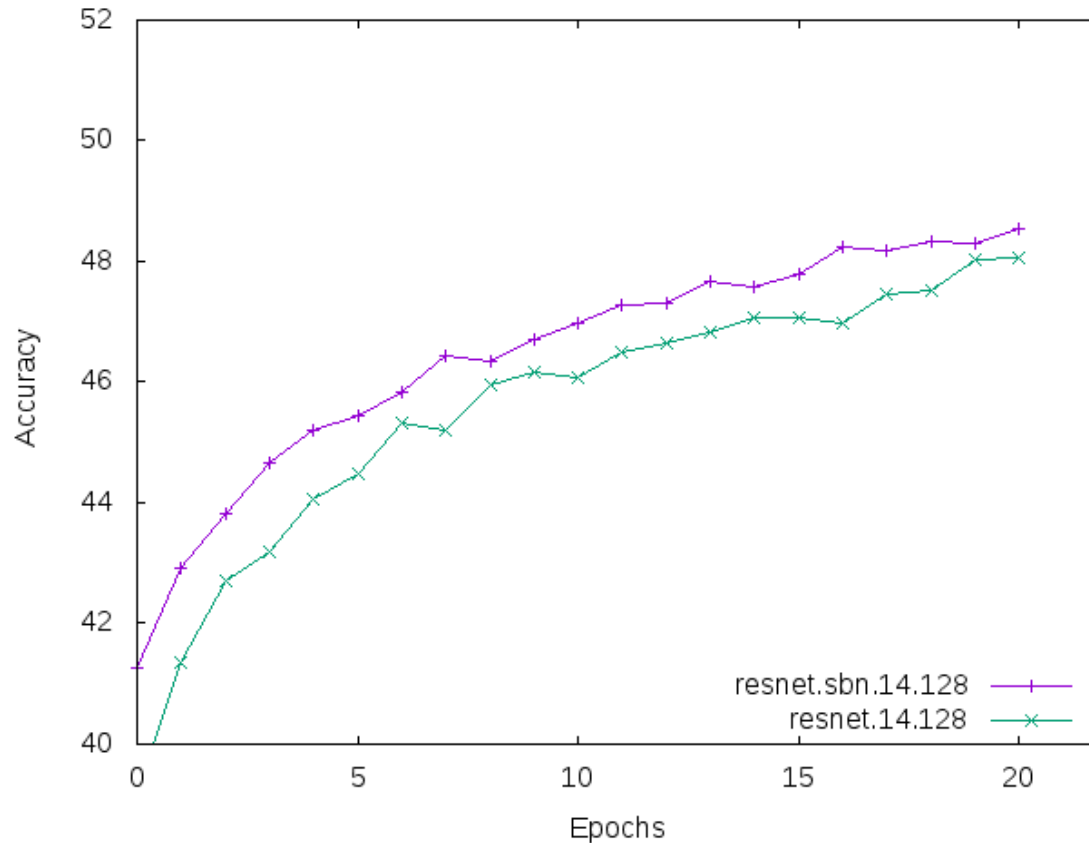
Golois Layer

- Evolution of the accuracy for resnet and Golois layers

Golois Layer

- Evolution of the error for different layers

Golois Layer

- Multiple Output Planes improves the generalization of residual networks.

- Spatial Batch Normalization helps train the residual network faster and obtain better accuracy.

Value Network

- The Golois policy network played 1,600,000 games against itself.

- 20 layers residual value network trained on these games.

- Parallel PUCT with policy and value networks.

- No playouts.

- Using this value network Golois reached 8d.

- ELF network => 9d.

AlphaGo Zero

AlphaGo Zero

AlphaGo Zero learns to play Go from scratch playing against itself.

After 40 days of self play it surpasses AlphaGo Master.

Nature, 18 october 2017.

It uses the raw representation of the board as input, even liberties are not used.

It has 15 input planes, 7 for the previous Black stones, 7 for the previous White Stones and 1 plane for the color to play.
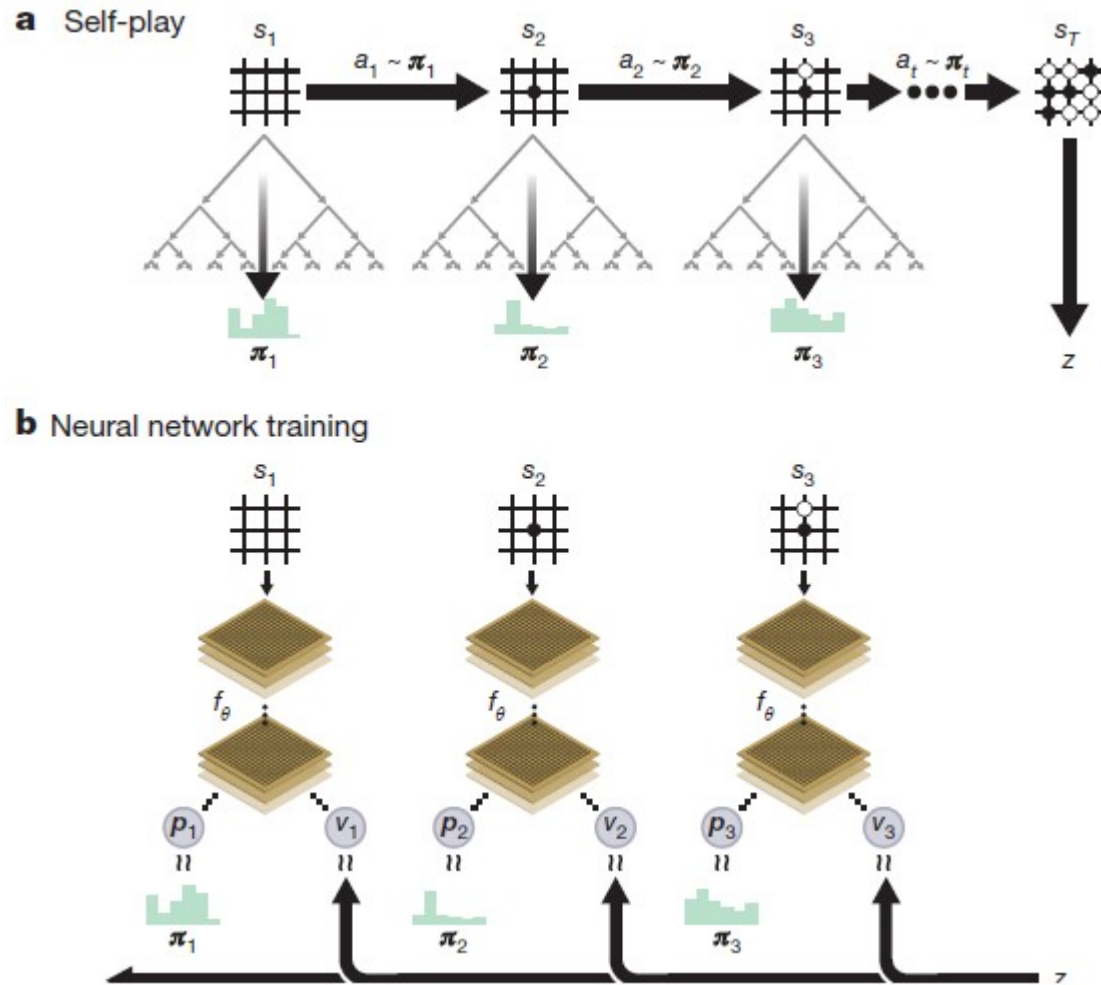
AlphaGo Zero

- It plays against itself using PUCT and 1,600 tree descent

$$U(s, a) = c_{\text{puct}} P(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)}$$

- It uses a residual neural network with two heads.

- One head is the policy, the other head is the value.
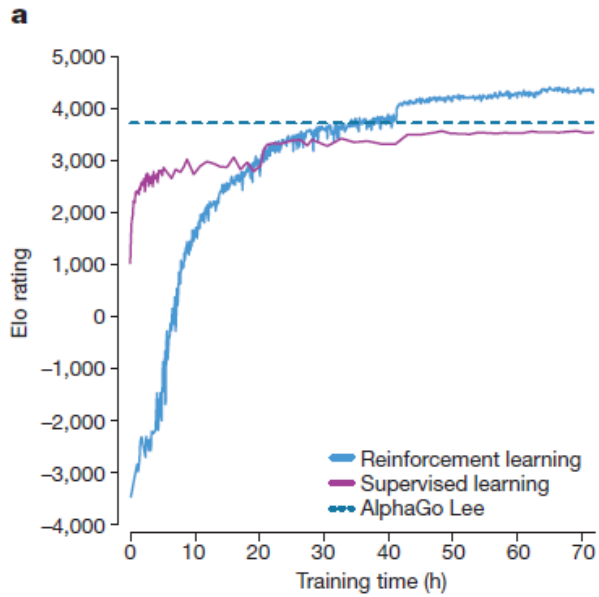
AlphaGo Zero

AlphaGo Zero

- After 4.9 million games against itself a 20 residual blocks neural network reaches the level of AlphaGo Lee (100-0).

- 3 days of self play on the machines of DeepMind.

- Comparison : Golois searches 1,600 nodes in 10 seconds on a 4 GPU machine.

- It would take Golois 466 years to play 4.9 million such games.
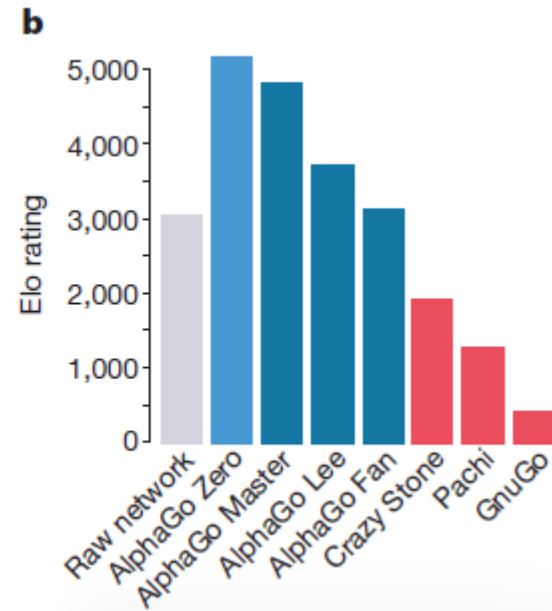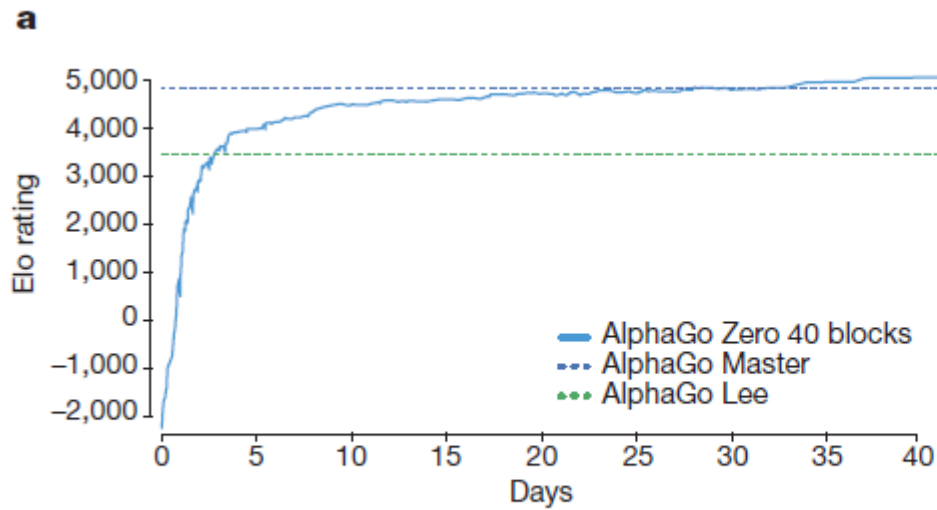
AlphaGo Zero
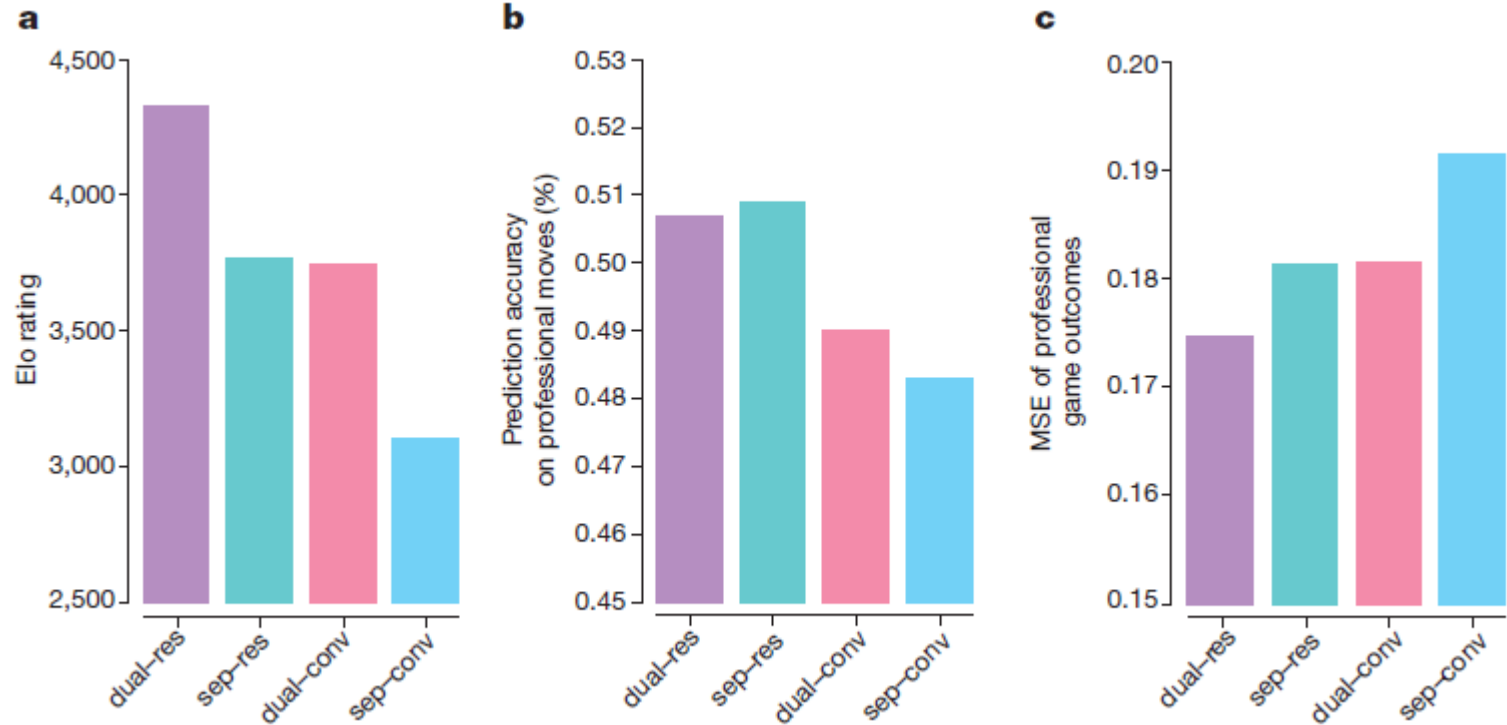
AlphaGo Zero
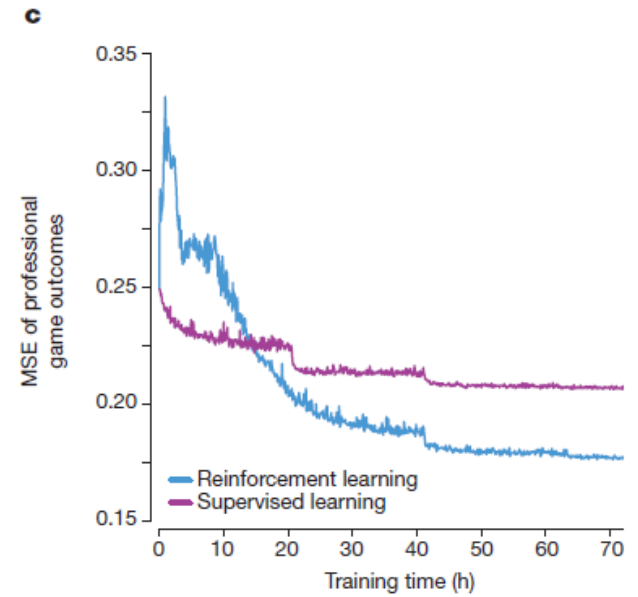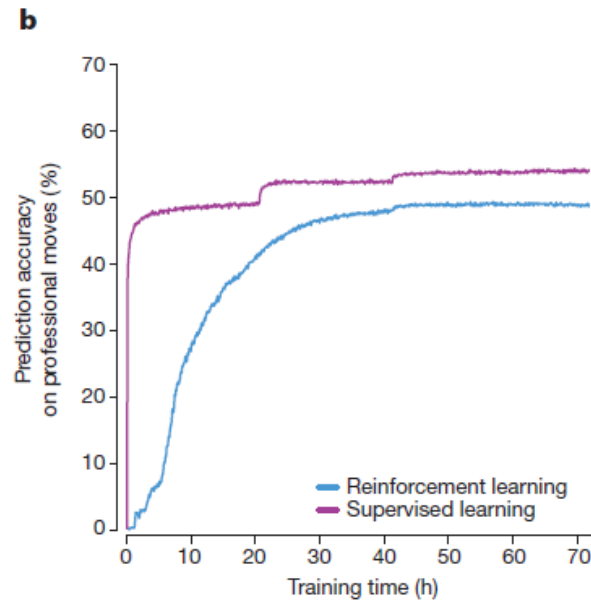
AlphaGo Zero

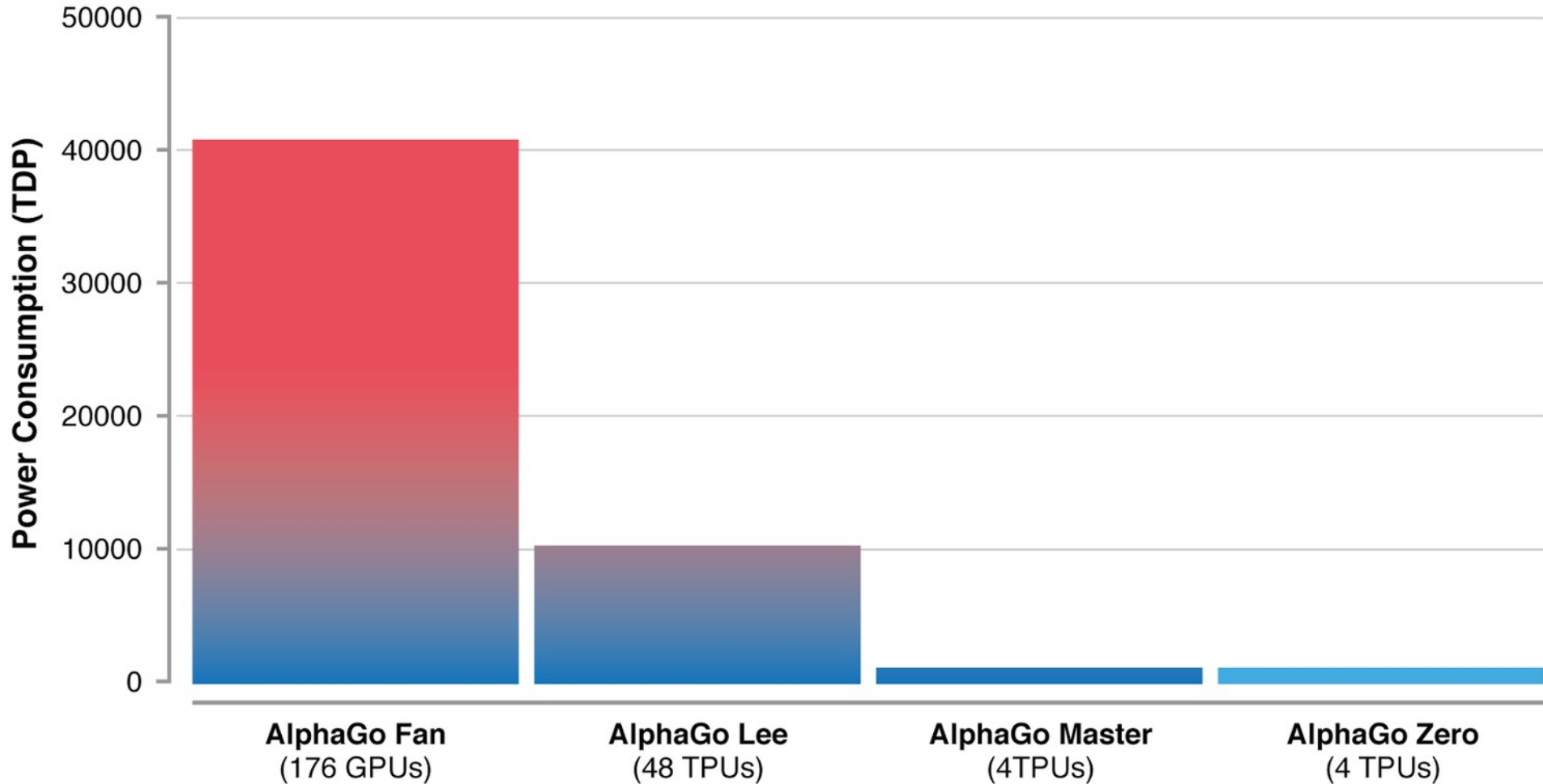- They used a longer experiment with a deeper network.
- 40 residual blocks.
- 40 days of self play on the machines of DeepMind.
- In the end it beats Master 89-11.

AlphaGo Zero

AlphaGo Zero

AlphaGo Zero

- AlphaGo Zero uses 40 residual blocks instead of 20 blocks for AlphaGo Master.

- With 20 blocks learning stalls after 3 days.

- Master with 40 blocks better than AlphaGo Zero?

Alpha Zero

Alpha Zero

- Arxiv, 5 december 2017.

- Deep reinforcement learning similar to AlphaGo Zero.

- Same algorithm applied to two other games :
  Chess and Shogi.

- Learning from scratch without prior knowledge.

Alpha Zero

- Alpha Zero surpasse Stockfish at Chess after 4 hours of self-play.

- Alpha Zero surpasses Elmo at shogi after 2 hours of self play.

Program	Chess	Shogi	Go
AlphaZero	80k	40k	16k
Stockfish	70,000k		
Elmo		35,000k	

Table S4: Evaluation speed (positions/second) of *AlphaZero*, *Stockfish*, and *Elmo* in chess, shogi and Go.

Alpha Zero

- 5 000 first generation TPU for training.

- 4 TPU for playing.

	Chess	Shogi	Go
Mini-batches	700k	700k	700k
Training Time	9h	12h	34h
Training Games	44 million	24 million	21 million
Thinking Time	800 sims	800 sims	800 sims
	40 ms	80 ms	200 ms

Table S3: Selected statistics of *AlphaZero* training in Chess, Shogi and Go.

Mu Zero

Mu Zero

- Arxiv, december 2019.

- Similar to Alpha Zero without knowing the rules of the games.

- Atari, Go, Chess and Shogi.

- Learning from scratch without prior knowledge.

Polygames

Polygames

- Alpha Zero approach for many games.

- A common interface to all the games.

- Fully convolutional network, average pooling…

- Pytorch and C++.

- Open source !

Conclusion

- AlphaGo : supervised learning and self play.

- Golois : residual networks and Spatial Batch Normalization improve learning.

- AlphaGo Zero : reinforcement learning from self play with MCTS. Raw inputs. Residual networks and combined policy and value network. Better than Master.

- Alpha Zero : Go, Chess and Shogi.

- Mu Zero : Atari, Go, Chess and Shogi.

- Polygames : many games.

Alpha Zero

Alpha Zero

- Define a network that takes as input the Breakthrough board and gives as output the policy and the value for the board.

- Bias the MCTS with policy and value using PUCT.

- Make the network play games and record the results of the Monte Carlo and the result of the games.

- Train the network on the results of the games.

- Iterate.

Alpha Zero

- The network takes 41 inputs with values 0 or 1, 20 inputs for black pawns, 20 inputs for white pawns and one input for the color to play.

- Option: also use previous boards as inputs.

- The network has 60 outputs for the policy head (3 possible moves for each cell), and 1 output for the value head.

- The architecture of the network can be completely connected as a starting point.

- Option : convolutional network, residual network.

Alpha Zero

1) Define the network

2) Implement the PUCT algorithm using the network. Use the same network for black and white, rotate the board for white so that moves are always forward.

3) Make the algorithm play against itself.

4) Record the Monte Carlo distributions and the result of self played games.

5) Train the network on the recorded data.

Projet Python

- Transformer une position de breakthrough 5x5 en trois matrices 5x5 de 0 et de 1 (Noir/Blanc/Vide).

- Faire deux réseaux convolutifs (blanc et noir) avec 76 sorties (75 coups possibles + évaluation) et ces trois matrices en entrée.

- Utiliser les réseaux dans PUCT pour politique et évaluation.

- Faire jouer à PUCT >100 parties contre lui même.

- Mémoriser pour chaque position un vecteur de 76 réels entre 0 et 1 (une fréquence pour chaque code de coup entre 0 et 75, code = $3*(5*x+y) + 0$, 1 ou 2) et un réel (1.0 si blanc a gagné, 0.0 sinon).

- Entraîner les deux réseaux convolutifs pour retrouver les fréquences et le résultat de la partie en sortie pour chaque position en entrée.

- Itérer.

Monte Carlo Search with Imperfect Information

Information Set MCTS

- Flat Monte Carlo Search gives good results for Phantom Go.

- Information Set MCTS.

- Card games.

Counter Factual Regret Minimization

- Poker : Libratus (CMU), DeepStack (UofA).

- Approximation of the Nash Equilibrium.

- There are about 320 trillion "information sets" in heads-up limit hold'em.

- What the algorithm does is look at all strategies that do not include a move, and count how much we "regret" having excluded the move from our mix.

- Combination with neural networks.

- Better than top professional players.

αμ

- Bridge
- Generate a set of possible worlds.
- Solve each world exactly
- Search multiple moves ahead
- Strategy Fusion => joint search
- Non Locality => Pareto fronts

PIMC

For all possible moves

　For all possible worlds

　　Exactly solve the world

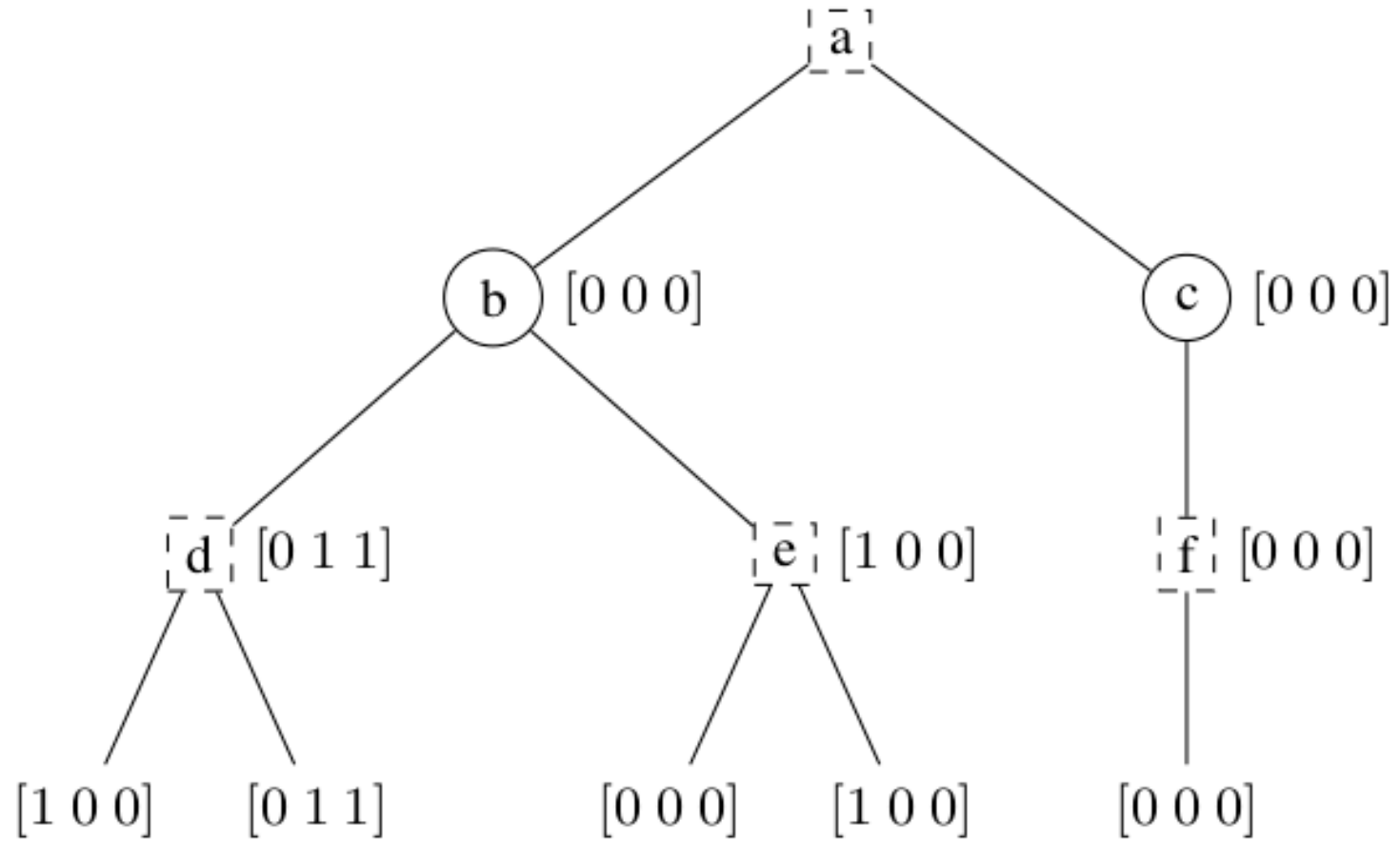Play the move winning in the most worlds

Strategy Fusion

- Problem = PIMC can play different moves in different worlds.

- Whereas the player cannot distinguish between the different worlds.

♠KJT7
♡AKQ
♢AKQ
♣xxx

N

S

♠A986
♡xxx
♢xxx
♣AKQ

Non Locality

Pareto Fronts

- A Pareto Front is a set of vectors.

- It maintains the set of vectors that are not dominated by other vectors.

- Consider the Pareto front {[1 0 0], [0 1 1]}.

- If the vector [0 0 1] is a candidate for entering the front, then the front stays unchanged since [0 0 1] is dominated by [0 1 1].

- If we add the vector [1 1 0] then the vector [1 0 0] is removed from the front since it is dominated by [1 1 0], and then [1 1 0] is inserted in the front. The new front becomes {[1 1 0], [0 1 1]}.

- It is useful to compare Pareto fronts.

- A Pareto front P1 dominates or is equal to a Pareto front P2 iff $\forall v \in P2$, $\exists v' \in P1$ such that (v' dominates v) or v'=v.

AlphaMu

- At Max nodes each possible move returns a Pareto front.

- The overall Pareto front is the union of all the Pareto fronts of the moves.

- The idea is to keep all the possible options for Max, i.e. Max has the choice between all the vectors of the overall Pareto front.
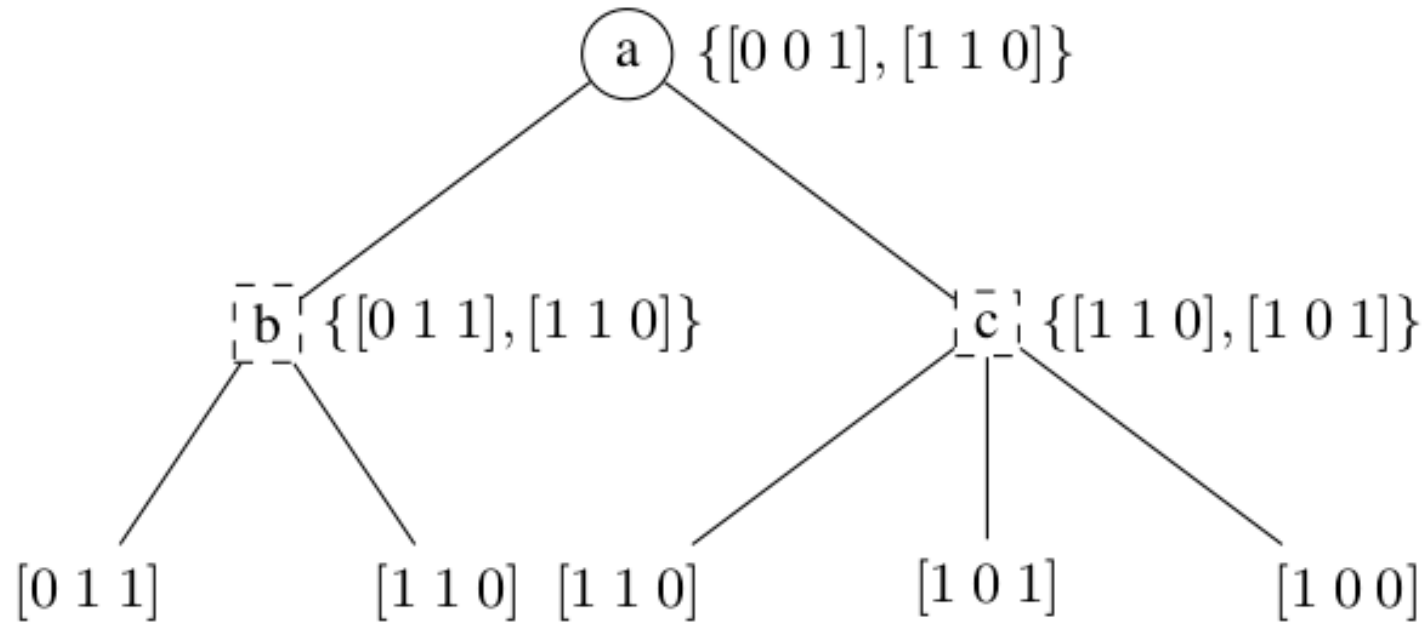
AlphaMu

- At Min nodes, the Min players can choose different moves in different possible worlds.

- They take the minimum outcome over all the possible moves for a possible world.

- When they can choose between two vectors they take for each index the minimum between the two values at this index of the two vectors.
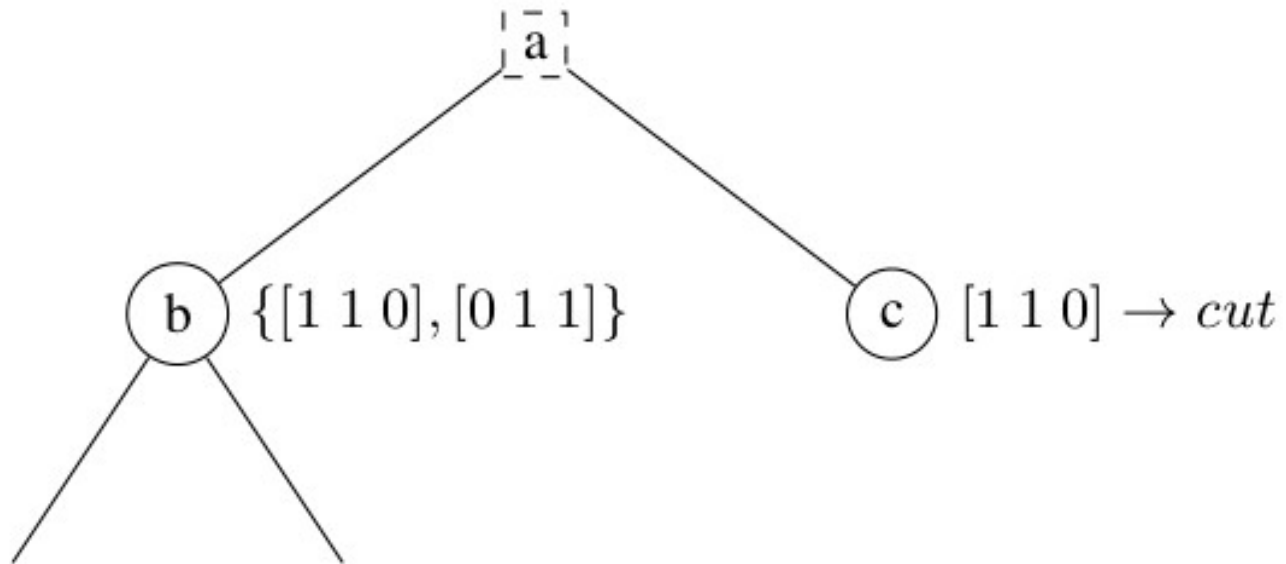
AlphaMu

- When Min moves lead to Pareto fronts, the Max player can choose any member of the Pareto front.

- For two possible moves of Min, the Max player can also choose any combination of a vector in the Pareto front of the first move and of a vector in the Pareto front of the second move.

- Compute all the combinations of the vectors in the Pareto fronts of all the Min moves.

- For each combination the minimum outcome is kept so as to produce a unique vector.

- Then this vector is inserted in the Pareto front of the Min node.

Product of Pareto Fronts at Min nodes

a $\{[0\ 0\ 1],[1\ 1\ 0]\}$

b $\{[0\ 1\ 1],[1\ 1\ 0]\}$

c $\{[1\ 1\ 0],[1\ 0\ 1]\}$

$[0\ 1\ 1]$ $[1\ 1\ 0]$ $[1\ 1\ 0]$ $[1\ 0\ 1]$ $[1\ 0\ 0]$

The Early Cut

The Root Cut

- If a move at the root of αμ for M Max moves gives the same probability of winning than the best move of the previous iteration of iterative deepening for M-1 Max moves, the search can safely be stopped since it is not possible to find a better move.

- A deeper search will always return a worse probability than the previous search because of strategy fusion.

- Therefore if the probability is equal to the one of the best move of the previous shallower search the probability cannot be improved and a better move cannot be found so it is safe to cut.

Experimental Results

- Comparison of the average time per move of different configurations of αμ on deals with 52 cards for the 3NT contract.

Cards	M	Worlds	TT	R	E	Time
52	1	20				0.118
52	2	20	n	n	n	1.054
52	2	20	y	y	n	0.512
52	2	20	y	n	y	0.503
52	2	20	y	y	y	0.433
52	3	20	n	n	n	10.276
52	3	20	y	y	n	3.891
52	3	20	y	n	y	1.950
52	3	20	y	y	y	1.176

Experimental Results

- Comparison of αμ versus PIMC for the 7NT contract, playing 10 000 games.

Cards	M	Worlds	$\neq$ results	Winrate	σ
52	2	20	283	0.643	0.0285
52	3	20	333	0.673	0.0257
52	4	20	374	0.679	0.0241
52	2	40	324	0.630	0.0268
52	3	40	347	0.637	0.0258
52	4	40	368	0.655	0.0248

AlphaMu

- AlphaMu solves de strategy fusion and the non locality problems of PIMC up to a given depth.

- It maintains Pareto Fronts in its search tree.

- It improves on PIMC for the 7NT contract of Bridge.

Conclusion

Monte Carlo Search is a simple algorithm that gives state of the art results for multiple problems:

- Games
- Puzzles
- Discovery of formulas
- RNA Inverse Folding
- Snake in the box
- Pancake
- Logistics
- Multiple Sequence Alignement