ゲーム木探索の最適制御:
将棋における局面評価の機械学習

東北大学大学院理学研究科
保木邦仁

コンピュータ将棋プログラム Bonanza

- Bonanza 製品版を販売
- 大和証券杯特別対局
  新聞一面カラー
  全国版テレビニュース
- NHK衛星第2「運命の一手」
  1時間に及ぶドキュメンタリー
- 角川書店新書
  ボナンザVS勝負脳

一般の方々にアピール

何故こんな事になってしまったのだろう。

- 強い将棋プログラムを無料で公開
- プログラムの動作がユニーク

Bonanza の歴史

1997年5月	Deep Blue がチェスの世界チャンピオンを破る
2004年夏	コンピュータチェスに関する文献との出会い 全幅探索で将棋プログラム作成を決意
2005年冬	本職の仕事内容（化学反応制御）の知識を将棋の局面評価学習に応用
2005年6月	フリーソフト Bonanza を公開
2006年5月	第16回世界コンピュータ選手権で優勝
2006年11月	清水上徹アマ竜王・加藤幸男アマ名人と対局
2007年3月	大和証券杯特別対局で渡辺明竜王相手に健闘
2007年5月	第17回世界コンピュータ選手権で4位

Singular Extensions: Adding Selectivity to Brute-Force Searching

Thomas Anantharaman, Murray S. Campbell and Feng-hsiung Hsu

Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213, USA

ABSTRACT

Brute-force alpha-beta search of games trees has proven relatively effective in numerous domains. In order to further improve performance, many brute-force game-playing programs have used the technique of selective deepening, searching more deeply on lines of play identified as important. Typically these extensions are based on static, domain-dependent knowledge. This paper describes a modification of brute-force search, singular extensions, that allows extensions to be identified in a dynamic, domain-independent, low-overhead manner. Singular extensions, when implemented in a chess-playing program, resulted in significant performance improvements.

1. Introduction

Brute-force alpha-beta search has come to hold a dominating position in games such as chess, checkers and Othello. Nevertheless, the brute-force approach has shown an inability, particularly in chess, to calculate the kinds of long forcing sequences that human masters are capable of discovering. One method of attacking this problem involves iterative incremental tree growth methods [2, 11, 12, 14], in which nodes are selected for expansion in such a way as to converge on a value or best move. In spite of their intuitive appeal, these methods have produced only limited successes (e.g. [12, 15] in chess tactics, [9] in Othello), due mainly to the large space and time overheads involved.

Another possibility is that of adding selectivity to the extremely efficient brute-force alpha-beta scheme. Static knowledge-based forward pruning is one method of creating selectivity, but it runs the risk of missing key moves and can safely increase the maximum search depth only slightly, without regards to how forceful the variations are. Null-move pruning [1] is less risky, but the potential increase in search depth is not large [5]. A more promising approach is the idea of *selective extensions*, also known as *selective deepening*, extending the search

Artificial Intelligence 43 (1990) 99–109

Checkers Is Solved

Jonathan Schaeffer,* Neil Burch, Yngvi Björnsson,† Akihiro Kishimoto,‡ Martin Müller, Robert Lake, Paul Lu, Steve Sutphen

The game of checkers has roughly 500 billion billion possible positions (5×10^{20}). The task of solving the game, determining the final result in a game with no mistakes made by either player, is daunting. Since 1989, almost continuously, dozens of computers have been working on solving checkers, applying state-of-the-art artificial intelligence techniques to the proving process. This paper announces that checkers is now solved: Perfect play by both sides leads to a draw. This is the most challenging popular game to be solved to date, roughly one million times as complex as Connect Four. Artificial intelligence technology has been used to generate strong heuristic-based game-playing programs, such as Deep Blue for chess. Solving a game takes this to the next level by replacing the heuristics with perfection.

Since Claude Shannon's seminal paper on the structure of a chess-playing program in 1950 (*1*), artificial intelligence researchers have developed programs capable of challenging and defeating the strongest human players in the world. Superhuman-strength programs exist for popular games such as chess [Deep Fritz (*2*)], checkers [Chinook (*3*)], Othello [Logistello (*4*)], and Scrabble [Maven (*5*)]. However strong these programs are, they are not perfect. Perfection implies solving a game—determining the final result (game-theoretic value) when neither player makes a mistake. There are three levels of solving a game (*6*). For the lowest level, ultra-weakly solved, the perfect-play result, but not a strategy for achieving that value, is known [e.g., in Hex the first player wins, but for large board sizes the winning strategy is not known (*7*)]. For weakly solved games, both the result and a strategy for achieving it from the start of the game are known [e.g., in Go Moku the first player wins and a program can demonstrate the win (*6*)]. Strongly solved games have the result computed for all possible positions that can arise in the game [e.g., Awari (*8*)].

Checkers (8 × 8 draughts) is a popular game enjoyed by millions of people worldwide, with many annual tournaments and a series of competitions that determine the world champion. There are numerous variants of the game played around the world. The game that is popular in North America and the (former) British Commonwealth has pieces (checkers) moving forward one square diagonally, kings moving forward or backward one square diagonally, and a forced-capture rule [see supporting online material (SOM) text].

The effort to solve checkers began in 1989, and the computations needed to achieve that result have been running almost continuously since then. At the peak in 1992, more than 200 processors were devoted to the problem simultaneously. The end result is one of the longest running computations completed to date.

With this paper, we announce that checkers has been weakly solved. From the starting position (Fig. 1, top), we have a computational proof that checkers is a draw. The proof consists of an explicit strategy that never loses—the program can achieve at least a draw against any opponent, playing either the black or white pieces. That checkers is a draw is not a surprise; grandmaster players have conjectured this for decades.

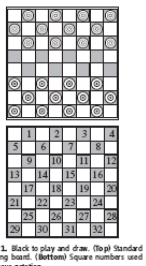
The checkers result pushes the boundary of artificial intelligence (AI). In the early days of AI research, the easiest path to achieving high performance was believed to be emulating the human approach. This was fraught with difficulty, especially the problems of capturing and encoding human knowledge. Human-like strategies are not necessarily the best computational strategies. Perhaps the biggest contribution of applying AI technology to developing game-playing programs was the realization that a search-intensive ("brute-force") approach could produce high-quality performance using minimal application-dependent knowledge. Over the past two decades, powerful search techniques have been developed and successfully applied to problems such as optimization, planning, and bioinformatics. The checkers proof extends this approach by developing a program that has little need for application-dependent knowledge and is almost completely reliant on search. With advanced AI algorithms and improved hardware (faster processors, larger memories, and larger disks), it has become possible to push the limits on the type and size of problems that can be solved. Even so, the checkers search space (5×10^{20}) represents a daunting challenge for today's technology.

Computer proofs in areas other than games have been done numerous times. Perhaps the best known is the four-color theorem (*9*). This deceptively simple conjecture—that given an arbitrary map with countries, you need at most four different colors to guarantee that no two adjoining countries have the same color—has been extremely difficult to prove analytically. In 1976, a computational proof was demonstrated. Despite the convincing result, some mathematicians were skeptical, distrusting proofs that had not been verified using human-derived theorems. Although important components of the checkers proof have been independently verified, there may be skeptics.

This article describes the background behind the effort to solve checkers, the methods used for achieving the result, an argument that the result is correct, and the implications of this research. The computer proof is online (*10*).

Background. The development of a strong checkers program began in the 1950s with Arthur Samuel's pioneering work in machine learning. In 1963, his program played a match against a capable player, winning a single game. This result was heralded as a triumph for the fledgling field of AI. Over time, the result was exaggerated, resulting in claims that checkers was now "solved" (*3*).

The Chinook project began in 1989 with the goal of building a program capable of challenging the world checkers champion. In 1990, Chinook earned the right to play for the World Championship. In 1992, World Champion Marion Tinsley narrowly defeated Chinook in the title match. In the 1994 rematch, Tinsley withdrew part way due to illness. He passed away eight months later. By 1996 Chinook was much



	1	2	3	4	
5		6		7	8
	9	10	11		12
13		14		15	16
	17		18		19	20
21		22		23		24
	25		26		27		28
29		30		31		32	

Fig. 1. Black to play and draw. (**Top**) Standard starting board. (**Bottom**) Square numbers used for move notation.

Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada.
*To whom correspondence should be addressed. E-mail: jonathan@cs.ualberta.ca
†Present address: Department of Computer Science, Reykjavik University, Reykjavik, Kringlan 1, IS-103, Iceland.
‡Present address: Department of Media Architecture, Future University, Hakodate, 116-2 Kamedanakano-cho Hakodate Hokkaido, 041-8655, Japan.

ゲームの場合の数

• チェッカー	10の30乗	コンピュータ
• オセロ	10の60乗	コンピュータ
• チェス	10の120乗	コンピュータ
• 将棋	10の220乗	アマ5段強
• 囲碁	10の360乗	アマ1級

必勝法が求まるのはチェッカーまで

オセロ・チェスは力任せの探索（全幅探索）と，
簡単な評価関数で人間よりは強くなる

Bonanza の特徴

1．全幅探索

- 主にコンピュータチェスで使用される手法
- ルールで許される手全てを考慮する
- 広く浅い読みを行う
- ゲームの知識に基づく戦略の選択は一切行わない

力任せの探索は簡単・高性能！

Minimax 法 ―――――――――――――――― $(80)^n$

Minimax 法＋beta cut ―――――――――――― $\left(\sqrt{80}\right)^n = (8.9)^n$

Minimax 法＋beta cut＋null move pruning や hash cut $\left(\frac{1}{4}\sqrt{80}\right)^n = (2.23)^n$

Minimax 法＋beta cut＋null move pruning や hash cut ＋Futility pruning $\frac{1}{4}\left(\frac{1}{4}\sqrt{80}\right)^n = \frac{1}{4}(2.23)^n$

１秒に50万局面探索 ⟶ １秒で18手先まで読める

実際は静止探索や move reordering の不備により効率が落ちる.
思考時間は序盤 3^n, 終盤 5^n に比例する程度

2. 局面評価の機械学習

最適制御理論

最大（小）化問題として力学系の制御方針を推論する

- 化学反応を制御する
- 最小燃料のロケット弾道
- 工場の消費電力
- 池の魚に与える餌

将棋プログラムの最適制御

プロ棋士と同様の棋譜を残すように
プログラムの動作を制御

数理的なモデルで将棋をとらえる

局面評価の機械学習

- TD-Gammon (G. Tesauro)
  思考部の学習
  Temporal difference + neuron network
- Logistello (M. Buro)
  辺のパターンの重み学習
  最小二乗法
- GPS 将棋 (金子ら)
  序盤の駒組みを棋譜から学習
  親子，兄弟モデルを用いた線形回帰

最適制御理論

最大（小）化問題として力学系の制御方針を推論する

- ラグランジュ形式の解析力学
- パルス整形による化学反応制御
- 最小燃料のロケット弾道
- 工場の消費電力
- 池の魚に与える餌

$$J = \int_0^T l\left(x,u,t\right)dt$$

t: 時間に関する数（離散も可）
$x(t)$: 系の状態
u: 制御変数

t を手数，$x(t)$ を minimax 探索の指し手で発展していく局面，u を特徴ベクトルとみなし，最適制御理論に基づいた将棋プログラムの機械学習をおこなう

Minimax 探索結果の最適制御

最適制御法に基づき，棋譜の指し手と minimax 探索が良く一致する特徴ベクトル v を求める

指し手の一致度を測る目的関数 J' を以下のように設計する

$$J'(P_0, P_1, \mathrm{K}, P_{N-1}, \boldsymbol{v}) = \sum_{i=0}^{N-1} l(P_i, \boldsymbol{v})$$

P_i: サンプルされた棋譜中の一局面

$l(P_i,v)$: 全合法手の評価値の違いの度合いを測る

$$l(P, \boldsymbol{v}) = \sum_{m=1}^{M} T\left[\xi(p_m, \boldsymbol{v}) - \xi(p_{m=0}, \boldsymbol{v})\right]$$

p_m: 局面 P を合法手 m で進めた子局面

M: 局面 P での合法手の数

$m = 0$: 棋譜中で実際に指された手

$\xi(p_m,v)$: minimax 探索の評価値

$T(x)$: 評価値の差を，棋譜の指し手との一致度に変換する関数

$T(x)$の関数形と役割 1

一価の単調増加関数．$|x|$ が大きい領域で傾きが小さく，$x = 0$ 付近で傾きが大きくなる 1 階微分可能なもの． $x = 0$， $x < 0$， $x > 0$ の意味は…

$T(x)$ を階段型関数にとると，目的関数 J' は「サンプルされた局面中，棋譜で指された手よりも良く判断してしまった合法手の数」となる．

強いプレイヤーと同じ手を指す minimax 関数の設計

↓

目的関数に停留値を与える特徴ベクトル v の求解

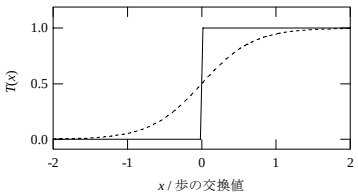


図 1 ：$T(x)$ の関数形．（実線）階段型関数
（破線）計算で実際に用いられたもの

$T(x)$の関数形と役割２

$T(x)$ が傾きを持つ領域の幅は，目的関数が指し手の善悪判断を行う分解能に相当

・分解能が良すぎると…

目的関数の滑らかさが失われる.

傾きを持つ領域内にあるサンプル中の合法手が減少し，学習データが不足.

・分解能が悪すぎると…

目的関数の大きさと，強い人の指し手との一致度の関係が失われる.

悪い手をさらに悪く，良い手をさらに良く評価する方針で最適化されてしまう.
このように，minimax 探索結果を変えない調整は適当ではない.

Minimax 探索の深さが十分ではなく，駒の損得・詰みが全く認識できない局面も学習データとして使ってしまう.

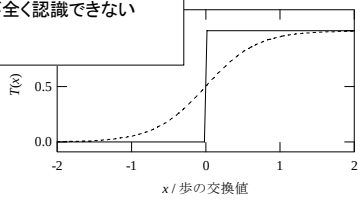


図１：$T(x)$ の関数形．（実線）階段型関数
（破線）計算で実際に用いられたもの

拘束条件の導入

自明解 $v = 0$ や駒割り等が定数倍変化した別解の除去

$$J''(P_0,\mathrm{K},\boldsymbol{v}) = J'(P_0,\mathrm{K},\boldsymbol{v}) + \lambda\left[M_1(\boldsymbol{v}) - M_0\right]$$

λ はラグランジュの未定乗数．$M_1(\boldsymbol{v})$ は駒割りに関する特徴ベクトル要素の大きさ等に相当し，これを定数 M_0 に拘束する.

ラクランジュ未定乗数 λ を導入する利点

p 個の変数に対する q 個の拘束条件から $\mathrm{p}-\mathrm{q}$ 個の独立な変数を求めることは可能だが，この作業は一般に困難

λ に適当な値を設定し，目的関数の導関数が 0 になる p 個の変数を求めると，独立な変数の組を求めることなく拘束条件を課すことが可能．

ペナルティーの導入

- 駒割りの占める割合を出来るだけ高く保つ
- 過学習を回避
- 目的関数の極小点を減らす

最終的な目的関数の表式

$$J(P_0,\mathrm{K},\boldsymbol{v}) = \sum_{i=0}^{N-1} l(P_i,\boldsymbol{v}) + \lambda\left[M_1(\boldsymbol{v}) - M_0\right] + wM_2(\boldsymbol{v})$$

w はペナルティーの重みを決めるパラメタ,
$M_2(v)$ は特徴ベクトル v の大きさを表す関数

最適化の数値的手法 1

ベクトルの要素数が非常に多いため，目的関数の勾配を求めて最適化を行う

$$\nabla_{\boldsymbol{v}} J(P_0,\mathrm{K},\boldsymbol{v}) = \sum_{i=0}^{N-1}\sum_{m=1}^{M_i-1} \frac{dT(x)}{dx} \nabla_{\boldsymbol{v}}\left[f\left(p_{i,m}^{\mathrm{leaf}},\boldsymbol{v}\right) - f\left(p_{i,m=0}^{\mathrm{leaf}},\boldsymbol{v}\right)\right]$$

$$+\lambda\nabla_{\boldsymbol{v}} M_1(\boldsymbol{v}) + w\nabla_{\boldsymbol{v}} M_2(\boldsymbol{v})$$

minimax 探索の結果としての最善応手列が $\boldsymbol{v}$ 近傍で単一と仮定し以下の関係を用いた

$$\nabla_{\boldsymbol{v}}\xi\left(p_{i,m},\boldsymbol{v}\right) = \nabla_{\boldsymbol{v}} f\left(p_{i,m}^{\mathrm{leaf}},\boldsymbol{v}\right)$$

興味深いことに，TD-leaf でも…

最適化の数値的手法２

目的関数が十分滑らかではない

- v を更新すると最善応手系列が変わる
- $T(x)$ の幅の中にあるサンプル数が少ない

よって，２次収束の性質をもつ手法は上手く働かない

$$v_l^{\text{new}} = v_l^{\text{old}} - h\,\text{sign}\left[\frac{\partial J(P_0,\mathrm{K},\boldsymbol{v})}{\partial v_l}\right]$$

- 初期特徴ベクトル要素と h を整数にとる
- h は初めは粗く，徐々に小さく

Bonanza 探索アルゴリズム概要1

通常の探索

基本的に全幅探索＋静止探索．用いる枝刈りは

- Beta cut
- Null move pruning
- Futility pruning
- 2，8段目の香，飛角不成りは考えない

- Bitboard
- 反復深化
- Aspiration search．ウインドウ幅は歩の交換値2つぶん
- PV search
- 延長王手 (1手)，one reply (0.5手)，リキャプチャー (0.5手)
- 指し手の順序並び替え
  - Transposition table （静止探索では用いない）
  - 再帰的反復深化（PV node で hash move が手に入らない場合）
  - 一手で詰む王手
  - Static Exchange Evaluation (SEE)
  - Killer moves
  - History heuristics
- 専用の詰み探索ルーチンは使わない．

将棋に応用された Futility pruning

チェスプログラムで広く利用される枝刈の手法
全幅探索＋静止探索のアルゴリズムにおける，末端付近の性質を利用

局面 P で指し手 m を指して静止探索を行う
以下の条件を満たす指して m は，安全に枝刈りされる

Futility pruning の条件

$$M(P) + M\mathrm{cap}(m) + V\mathrm{max} <= \mathrm{alpha}$$

M(P)	P の駒割
Mcap(m)	m による駒割の変動
Vmax	静的評価関数における駒割以外の寄与最大値

将棋の場合

チェスと比較して Vmax の値が大きくなり，この手法を直接用いると効率よく枝刈りされない
そこで，一手駒を動かすことによる局面評価値の変化が Vmax ほど大きくならない性質に着目

Futility Pruning

E(P) + Vdiff(m) <= alpha

Vdiff(m) = Mcap(m) + Mpiece-king(m) + Vmax(m)

Mpiece-king(m)	王以外の駒が移動した時，その駒と王の位置関係
Vmax(m)	駒の移動の種類（王の移動・王以外の移動）に対応した定数

Bonanza 探索アルゴリズム概要2

静止探索

手番を持つ側は手を指すか stand pat を返すか選択

静止探索を開始してから深さ７段目まで SEE の下で駒損しない以下の手を探索
- 取る手
- 成る手
- 王の移動
- 一手詰みの王手

８段目以降は歩を取る成らない手を除いた駒損しない駒を取る手を生成

SEE による指し手の順序並び替え

Bonanza 探索アルゴリズム概要3

静的評価関数で考慮する局面の特徴

- 駒割
- 王と他の駒2つの位置
- 王と王に隣接した味方の駒とその他の味方の駒3つの位置
- 隣接しあった駒2つの位置関係
- 竜馬飛角桂香の利き上にいる駒の種類
- 竜馬飛角香が動ける枡の数
- pin analysis. ピンされている駒の種類方向王との距離を考慮
- 角と同じ色の枡にいる味方の歩の数
- 歩桂銀が前進できるか
- 竜飛香の前・後の歩
- 王の周囲25枡の利きの配置

プロ棋士の棋譜3万局と将棋クラブ24の棋譜3万局（主に入玉）
を用いて静的評価関数のパラメタ約1万を調整

$\xi(p, v)$ には一手読み＋静止探索の Bonanza を使用

Bonanza 探索アルゴリズム概要4

- 拘束条件は飛角金銀桂香歩の駒割りの和
- ペナルティーは以下のように課す

$$M_2(\boldsymbol{v}) = \sum_l A_l(\boldsymbol{v}) v_l^2$$

$$A_l(\boldsymbol{v}) = \sum_{i=0}^{N-1} \sum_{m=1}^{M_i-1} \left| \frac{dT(x)}{dx} \frac{\partial}{\partial v_l} \left[f\left(p_{i,m}^{\text{leaf}}, \boldsymbol{v}\right) - f\left(p_{i,m=0}^{\text{leaf}}, \boldsymbol{v}\right) \right] \right|$$

出現頻度の高い特徴には強いペナルティー

例） ８八王ー金の位置に対する得点

ペナルティーなし	A_l なし	A_l あり
1八金 -200 3八金 -150	1八金 -150 3八金 -149	1八金 -150 3八金 -100

結果

駒割（交換値）

歩	と	香	桂	成香	成桂	成銀	銀	金	角	馬	飛	竜
106	272	279	304	323	363	415	428	527	617	698	700	854

持ち駒の数

歩：	27	33	21	6	-8	-17	-23…	
香：	28	39	51	63				
桂：	22	12	-15	-48				
銀：	37	28	-2	-51				
金：	31	21	-4	-39				
角：	28	9						
飛：	59	45						

角・馬の動ける升の数

角：	-55	-25	-7	0	8	14	6	9
馬：	-28	-12	-3	1	8	10	16	11

香・飛・竜縦方向に動ける升の数

香：	x	-7	2	6	18	25	27	24
飛：	-61	-43	-22	-9	-2	9	12	17
竜：	-45	-26	-17	-13	-1	1	10	8

飛・竜横方向に動ける升の数

飛：	-72	-47	-18	4	10	28	21	27
竜：	-35	-23	-16	-10	-4	4	11	12

王が8八にいる時の味方の金・と

-67	-60	-45	-27	-30	-51	-61	-73	-67
-57	-59	-28	-4	6	-13	-48	-82	-62
-18	-26	-26	0	14	-5	-27	-51	-40
-57	-35	-20	3	-7	-8	-15	-49	-64
-55	-36	-13	-18	-13	-23	-17	-35	-77
-57	-16	-5	-8	-12	-25	-37	-54	-68
-34	-3	13	19	-25	-39	-55	-73	-92
-97	K	30	-16	-6	-51	-36	-85	-150
-40	34	-49	23	-40	-17	-109	-123	-118

王が9九にいる時の味方の金・と

-69	-45	-17	8	-13	-38	-70	-79	-75
-21	-55	11	29	22	-12	-55	-80	-75
-12	-29	-8	28	37	0	-36	-56	-65
14	-35	-7	33	9	-5	-33	-56	-83
-81	-49	-9	-11	-14	-25	-39	-56	-101
-59	-25	2	-28	-37	-57	-63	-56	-72
14	36	17	-19	-49	-74	-80	-108	-75
45	27	18	1	-32	-91	-123	-127	-138
K	40	31	0	-14	-56	-139	-133	-116

王が8八にいる時の敵の金・と

-106	-25	-5	2	-42	-11	-20	-41	-67
-53	-7	10	-4	5	-3	8	-18	-72
-62	-26	-18	9	-21	4	-11	-28	-85
-50	-14	9	-9	-9	-20	-13	-28	-86
-12	24	25	4	-20	-27	-30	-47	-63
110	186	144	49	2	-29	-28	-36	-40
450	450	450	149	27	-12	-22	-25	-34
450	K	450	156	15	-24	-40	-31	-44
112	450	212	63	-19	-52	-69	-59	-72

王が9九にいる時の敵の金・と

-52	-8	15	7	-47	-34	-32	-57	-177
-54	14	30	5	-2	-16	6	-49	-98
-80	-11	-1	12	-17	-1	-22	-31	-79
-58	-9	-8	-10	-25	-30	-28	-40	-130
-26	-7	-6	-8	-32	-46	-42	-85	-182
128	120	35	-15	-7	-57	-56	-62	-116
320	271	206	65	11	-18	-45	-36	-68
318	444	214	106	14	-31	-47	-55	-77
K	448	207	51	-8	-61	-91	-99	-96

王が8八にいる時の味方の歩

x	x	x	x	x	x	x	x	x
-1	-39	-44	-52	-36	-49	-45	-67	-47
16	-31	-42	-13	0	-3	-32	-30	5
21	21	15	21	25	21	14	23	14
2	-12	1	4	4	-6	-1	-5	4
7	-12	2	11	0	-5	0	-8	2
-5	7	3	0	-3	-3	-6	3	2
62	K	48	10	-3	-19	-34	-27	-79
37	-97	15	13	-5	-23	-31	-24	-60

王が9九にいる時の味方の歩

x	x	x	x	x	x	x	x	x
-48	-86	-48	11	-7	-40	-35	-49	-35
28	14	-16	12	19	0	-60	-25	-10
58	29	39	26	38	22	8	24	-1
-7	-6	7	2	-5	-6	-5	-5	-4
-5	-13	0	3	0	-11	4	-16	-1
23	9	2	2	-8	2	-8	-1	8
72	-3	40	26	-10	-37	-49	-40	-99
K	36	67	33	-10	-39	-49	-56	-90

王が8八にいる時の敵の歩

-48	-111	-57	-38	-21	-25	-29	-60	-56
8	-52	-41	-6	-3	-2	-24	-24	2
-8	-3	-7	-7	-2	-1	-4	1	1
3	-9	-3	-1	-1	-1	4	-13	1
19	4	4	15	-3	-2	0	-11	-19
86	156	82	31	27	11	3	17	-20
115	0	77	50	4	-13	-18	-21	-80
83	K	84	-19	-4	-39	-25	-29	-86
x	x	x	x	x	x	x	x	x

王が9九にいる時の敵の歩

-101	-97	-49	-4	0	-11	-26	-48	-69
-24	-81	-25	-2	7	-2	-21	-32	-27
-30	-9	-13	-16	-10	-5	-10	2	3
-18	-5	1	-8	-4	-12	-9	-24	-8
1	8	-4	0	-3	-12	-19	-20	-38
72	108	31	44	15	10	-13	7	-76
200	199	76	53	3	-16	-35	-43	-136
0	200	111	36	-4	-54	-32	-35	-87
K	x	x	x	x	x	x	x	x

88王の時の味方の駒２つの位置

金の位置の得点

+8	-1	+1
王	銀	-33
+5	-14	+1

-7	+1	+1
王	金	+1
-2	+9	-96

歩の位置の得点

+1	+1	-2
王	銀	-16
+72	+19	+52

+1	+1	-1
王	金	+2
-107	-6	-42

78王の時の味方の駒２つの位置

金の位置の得点

+4	+6	+23
王	銀	-12
+18	+4	+33

-1	+9	+18
王	金	+10
+2	-18	-28

歩の位置の得点

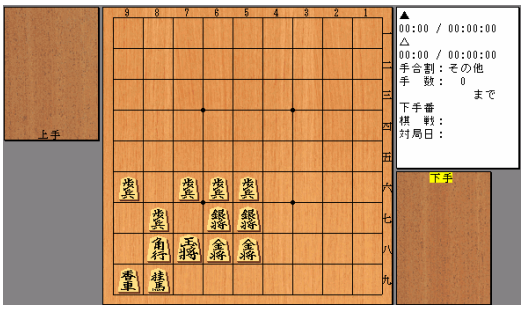
+0	+0	+0
王	銀	+4
+32	+21	+32

+12	+6	+4
王	金	+12
+25	+4	-42

問題点

王が6一にいる時の敵の飛

-69,	58	350	0	2	176	64	107	-119
126,	132	350	-80	92	150	181	-15	172
97,	48	101	-146	57	-141	0	128	56
-222	-137	90	-74	75	99	-135	-129	-400
-359	-112	94	-32	-225	15	-303	-281	-56
312	-315	-128	-400	344	-382	39	153	-303
-231	-400	-400	-37	-77	-400	-400	-400	39
-2	-92	66	15	105	-369	-400	-205	-95
77	-200	-155	-217	-81	51	-187	-212	-63



プロ棋士と Bonanza の思考内容

【▽４四飛まで】

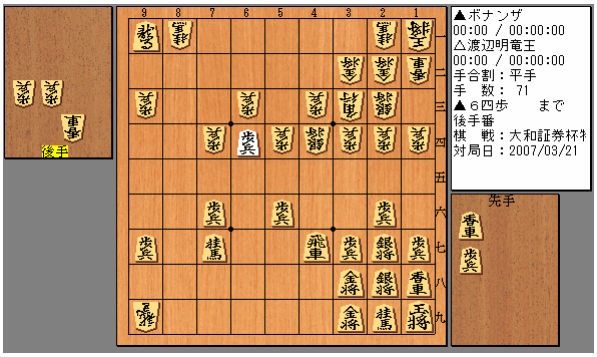
指し手	探索ノード数	割合
▲７七歩	17,330,794	17.8%
▲２六飛	41,907,150	43.1%
▲７九玉	13,377,128	13.8%
▲２五飛	5,513,094	5.7%
▲９五歩	3,494,148	3.6%
他の手合計	11,878,409	16.2%

図の局面におけるBonanzaの思考．基準深さ12，全探索ノード数約１億，最善手は▲７七歩（後手歩１枚有利）．

プロ棋士は▲7七歩，▲4五歩，▲9五歩に着目

伊藤毅志，松原仁，ライエル・グリンベルゲン
将棋の認知科学的研究，コンピュータ将棋の進歩5
松原仁編，共立出版，2005年

対竜王戦，中盤から終盤へ



対竜王戦，投了図