Learning To Play Chess Using Temporal Differences

Der Vortrag wird von Pham Thi Thu Trang gehalten

Einleitung

- TD- learning ist zuerst von Samuel (1959) entwickelt und später von Sutton (1988) mit TD(λ) erweitert und formalisiert worden.
- Es ist ein elegantes technisches Verfahren.
- Wahrscheinlich am meisten erfolgreich geworden durch das spielstarke „Tesauro‘s TD-Gammon“.
- Das Ziel wird sein, TD (λ) durch Modifizierungen gegenüber TD-learning zu entwickeln.

Der TD(λ) Algorithmus und seine Verwendung im Spiel

- Möglichkeit wählen : $p(x_t,x_{t+1},a)$
- Die erwartete Belohnung für jeden Zustand x von S (alle möglichen Brett- Positionen) ist gegeben durch : $J^*(x)=E_{x_N|x}\, r(x_N)$, wobei die Erwartung im Bezug auf die Übergangsmöglichkeiten $p(x_t,x_{t+1},a(x_t))$ ist und im Bezug auf die Aktion $a(x_t)$ wird, wenn der Agent seine Aktionen stochastisch wählt.

Annäherung von J*

- Benutzung einer parametrisierten Funktionsklasse:

  $J^{\sim} : S * R^k \rightarrow R$

- TD kombiniert mit dem Übergang $x_t \rightarrow x_{t+1}$:

  $d_t = J^{\sim}(x_{t+1},w) - J^{\sim}(x_t,w)$

- Die richtige Auswertungsfunktion hat die Eigenschaften: $E_{x_{t+1}|x}[J^{\sim}(x_{t+1},w) - J^{\sim}(x_t,w) = 0$

- Am Ende von TD ist folgendes erfüllt :

  $d_{N-1} = J^{\sim}(x_n,w) - J^{\sim}(x_{n-1},w) = r(x_n) - J^{\sim}(x_{n-1},w)$

Der TD(λ) Algorithmus aktualisiert den Parametervektor

- Am Ende vom Spiel ist w entsprechend:
  $w=w+\alpha\sum_{t=1} \nabla J^{\sim}(x_t, w)[\sum_{j=1} \lambda^{j-1} d_t]$
- Für $\lambda=0$: $w=w+\alpha\sum_{t=1} \nabla J^{\sim}(x_t, w)d_t$
  $w=w+\alpha\sum_{t=1} \nabla J^{\sim}(x_t, w) [J^{\sim}(x_{t+1},w) - J^{\sim}(x_t,w)]$
- $\lambda=1$: $w=w+\alpha\sum_{t=1} \nabla J^{\sim}(x_t, w) [r(x_N) - J^{\sim}(x_t,w)$
- Der TD(λ) Algorithmus konvergiert zu einem näherungsweise optimalen Parametervektor.

Minimax search und TD(λ)

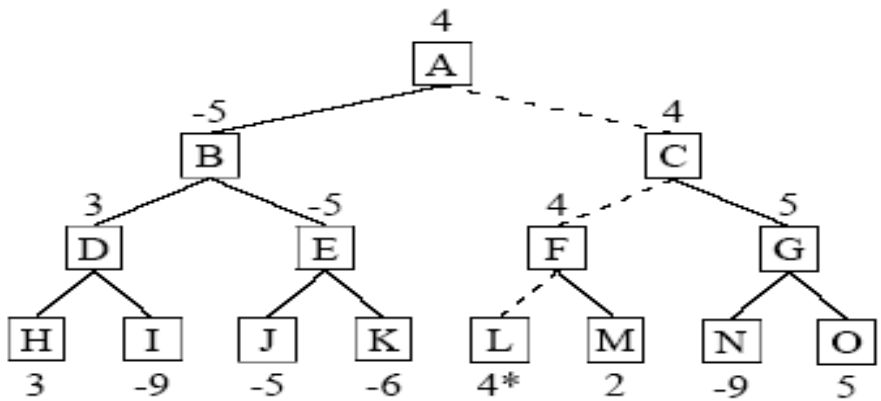

Figure 1. Full breadth, 3-ply search tree illustrating the minimax rule for propagating values. Each of the leaf nodes (H–O) is given a score by the evaluation function, $\tilde{J}(\cdot, w)$. These scores are then propagated back up the tree by assigning to each opponent's internal node the minimum of its children's values, and to each of our internal nodes the maximum of its children's values. The principle variation is then the sequence of best moves for either side starting from the root node, and this is illustrated by a dashed line in the figure. Note that the score at the root node A is the evaluation of the leaf node (L) of the principal variation. As there are no ties between any siblings, the derivative of A's score with respect to the parameters w is just $\nabla \tilde{J}(L, w)$.

Nicht eindeutige „Principal Variation“ (PV)

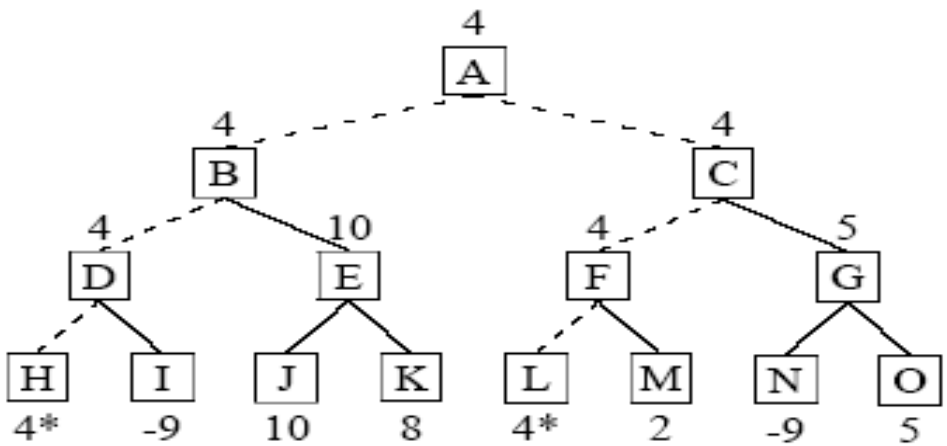

Figure 2. A search tree with a non-unique principal variation (PV). In this case the derivative of the root node A with respect to the parameters of the leaf-node evaluation function is multi-valued, either $\nabla \tilde{J}(H, w)$ or $\nabla \tilde{J}(L, w)$. Except for transpositions (in which case H and L are identical and the derivative is single-valued anyway), such "collisions" are likely to be extremely rare, so in TDLEAF(λ) we ignore them by choosing a leaf node arbitrarily from the available candidates.

TDLeaf(λ) und Schach

Zusammenfassend gesagt wird das Ergebnis der Versuche beschrieben, in denen der Algorithmus die Gewichte der linearen Bewertungsfunktion in dem Schachprogramm „KnightCap“ im Training bestimmt hat.

TDLeaf(λ) Algorithmus

Let $\tilde{J}(\cdot, w)$ be a class of evaluation functions parameterized by $w \in \mathbb{R}^k$. Let $x_1, \ldots, x_N$ be N positions that occurred during the course of a game, with $r(x_N)$ the outcome of the game. For notational convenience set $\tilde{J}(x_N, w) := r(x_N)$.

1. For each state x_i, compute $\tilde{J}_d(x_i, w)$ by performing minimax search to depth d from x_i and using $\tilde{J}(\cdot, w)$ to score the leaf nodes. Note that d may vary from position to position.

2. Let x_i^l denote the leaf node of the principle variation starting at x_i. If there is more than one principal variation, choose a leaf node from the available candidates at random. Note that

$$\tilde{J}_d(x_i, w) = \tilde{J}(x_i^l, w). \tag{9}$$

3. For $t = 1, \ldots, N-1$, compute the temporal differences:

$$d_t := \tilde{J}(x_{t+1}^l, w) - \tilde{J}(x_t^l, w). \tag{10}$$

4. Update w according to the TDLEAF(λ) formula:

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t^l, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]. \tag{11}$$

Figure 3. The TDLEAF(λ) algorithm

Die Experimente mit KnightCap

- Nehme KnightCap‘s Auswertungsfunktion und setze alle materiellen Parameter auf Null.
- Figuren werden durch einen Standartwert initialisiert:1 für Pawn, 4 für Knight, 4 für Bishop, 6 für Rook und 12 für Queen. Diese Parameter waren die Einstellung von „KnightCap“ auf freien Internet Schach -Servern sowohl gegen Menschen als auch gegen Computer.
- Nach 25 Spielen hat das Programm die Spielstärke eines mittelmäßigen Schachspielers.

Kleine Modifizierung zum Algorithmus

- $J^{\sim}(x_i^l,w)$ konvergiert bei der Auswertung zwischen –1 und 1: $v_i^l=\tanh[\beta J^{\sim}(x_i^l,w)]$; β war so angesetzt, dass der Wert von $\tanh[\beta J^{\sim}(x_i^l,w)] = 0.25$ äquivalent zur materiellen Überlegenheit der „Pawns“ ist.
- $TD(d_t=v^l_{t+1}-v_t)$ wurde im folgenden Verlauf geändert.
- Der Wert eines „Pawn“ wurde fest durch seinen initialen Wert bestimmt.

Schätzungen von KnightCap

- In 300 Spielen nimmt KnightCap‘s Stärke um 2150 Punkte zu (zum Vergleich mit der Stufe des meisterhaften menschlichen Spielers)
- Die handbestimmten Gewichte sind fast genauso gut im Vergleich mit „erlernten“ Gewichten.

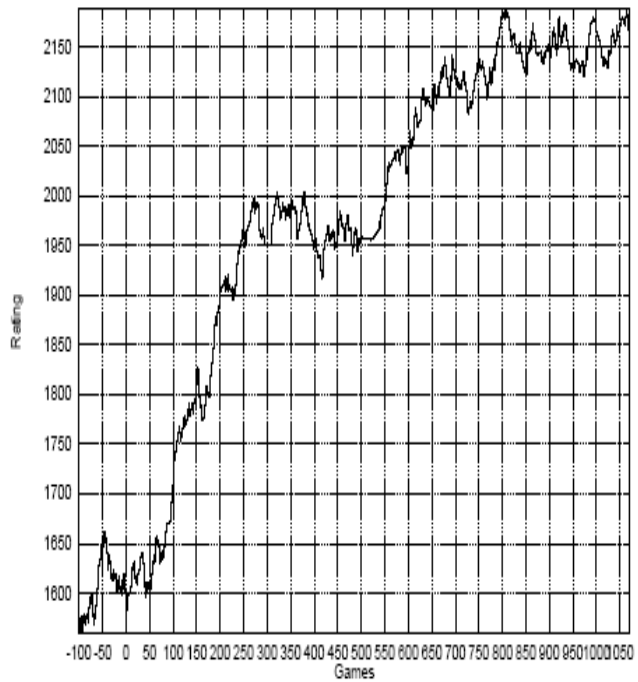

Figure 4. KnightCap’s rating as a function of games played (second experiment). Learning was turned on at game 0.

Programmstruktur von KnightCap

1. Board Representation
2. Search Algorithmus
3. Null Moves
4. Search Extensions
5. Asymmetries
6. Transposition Table
7. Move Ordering
8. Parallel Search
9. Evaluation Function
10. Modification for TDLeaf(λ)

Shogi

- Shogi ist der japanische Name für eine Art Schach.Es gehört zur gleichen Familie der Spiele wie Western -Schach oder chinesisches Schach.
- In Shogi werden die gefangenen Figuren nicht vom Spiel entfernt, sondern vom fangenden Spieler behalten und können später im Spiel wieder eingesetzt werden.
- Diese Spielregel lässt den Suchbaum in die Breite explodieren.

Shogi-Playing Engine(1)

- Die Experimente benutzen eine Suchmaschine, die von einem regulären Schachprogramm abgeleitet worden ist.
- Die Suchmaschine verwendet iterative Tiefensuche ‚alpha-beta Beschneidung und eine verstärkte Hintergrundsuche für die aktuelle Spielposition.
- Begrenzung der Hintergrundsuche auf 6 Spielpositionen.
- „Null-Move“ Baum-Beschneidung wird benutzt, um die Größe des Suchenbaums zu reduzieren.

Shogi-Playing Engine(2)

- Die Suche wird effizienter gestaltet, in dem man eine Austauschtabelle verwendet.
- Die Bewertungsfunktion an den Blättern des Hintergrundsuchbaums benutzt nur feste Ergebnisse mit zufälliger Zugwahl an der Wurzel.
- Die erlernten Werte der 13. Spielfiguren werden von der Auswertungsfunktion behandelt.
- Gleichgewicht der Position = Summe aller Figurenwerte (inklusive der Gefangenen) - Figurenwerte des Gegners.

Die Experimente mit Shogi

- In allen Spielen wird pro Position nur bis Tiefe 3 im Suchbaum ausgewertet.
- Um wiederholte Spielpositionen zu vermeiden, wird die Zugliste zufällig gewählt.
- Bei jedem Spiel wird der ermittelte Wert der Suche nach jeder Bewegung gesichert und entspricht der zugehörigen Position.
- Das Lernen der Werte für die Figuren geschieht ganz zufällig durch gegen sich selbst spielen.

Weight Traces

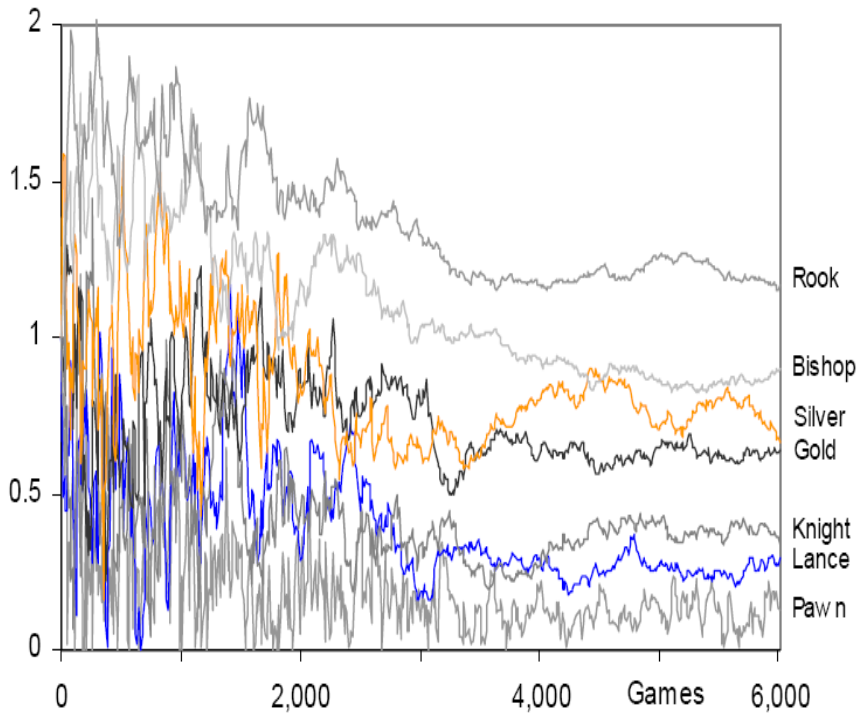

Fig. 2. Typical weight traces (main pieces)

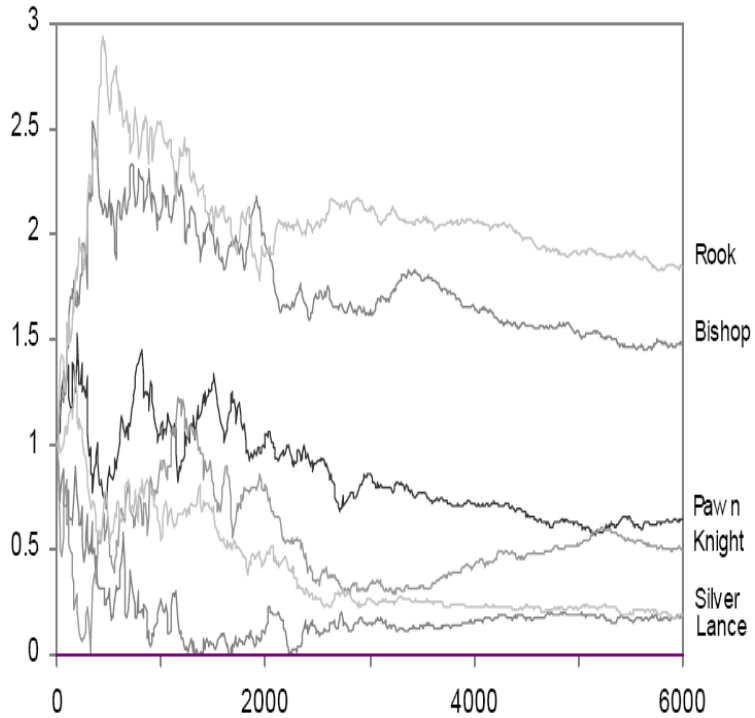

Fig. 3. Typical weight traces (promoted pieces)

Main Piece Value

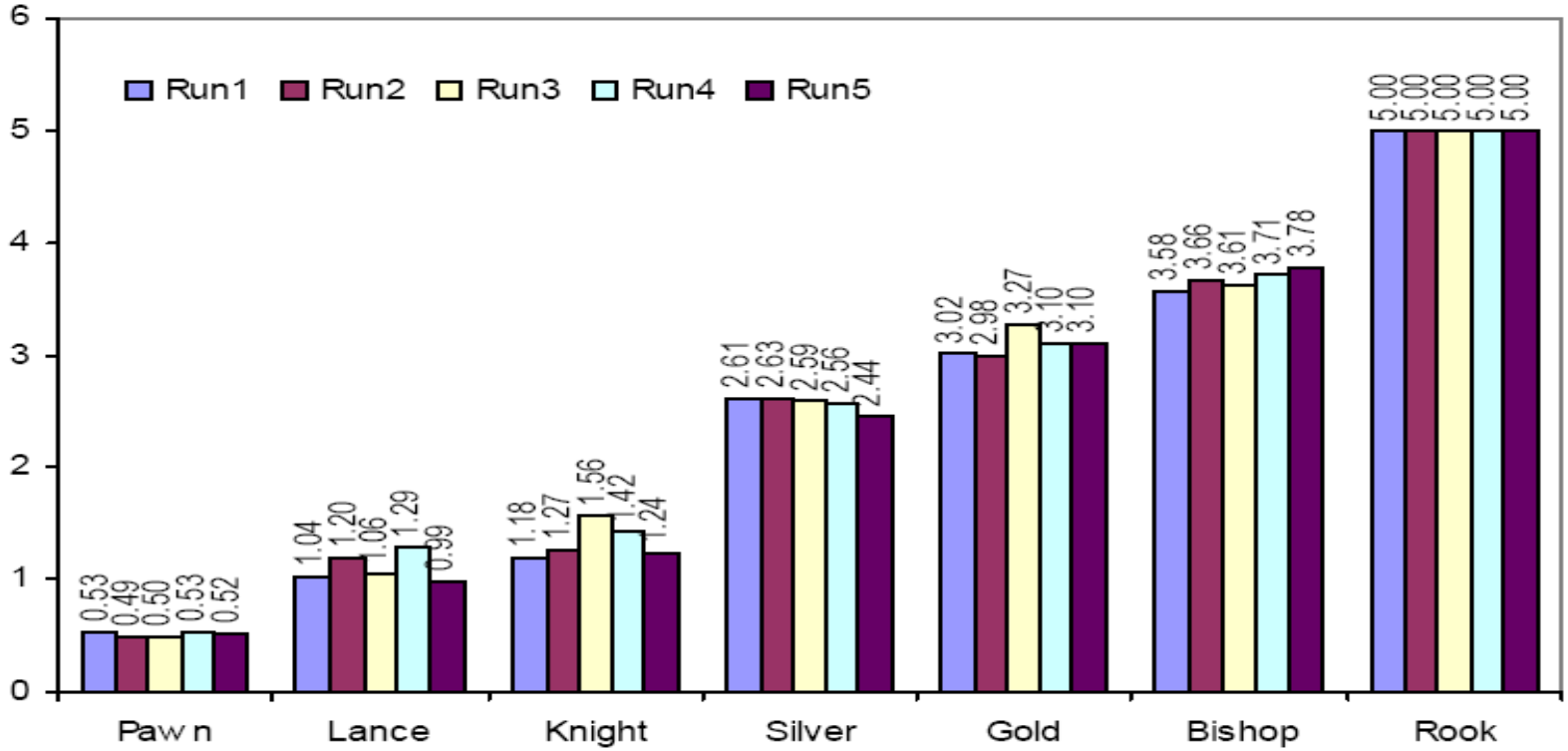

Fig. 4. Normalised learnt values for 5 runs (main pieces)

Promoted Piece Value

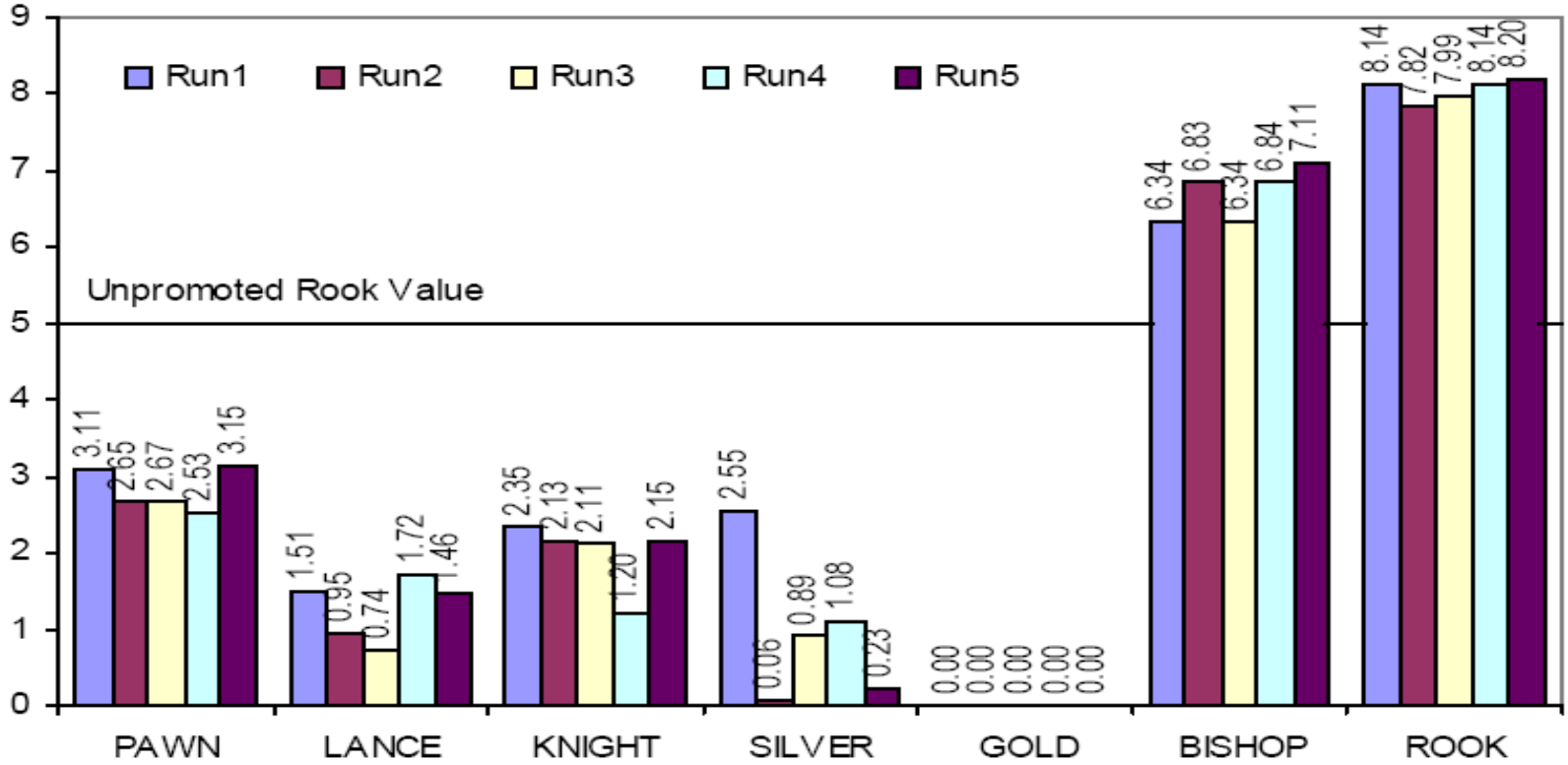

Fig. 5. Normalised learnt values for 5 runs (promoted pieces)

Matches to Test the Learnt Value

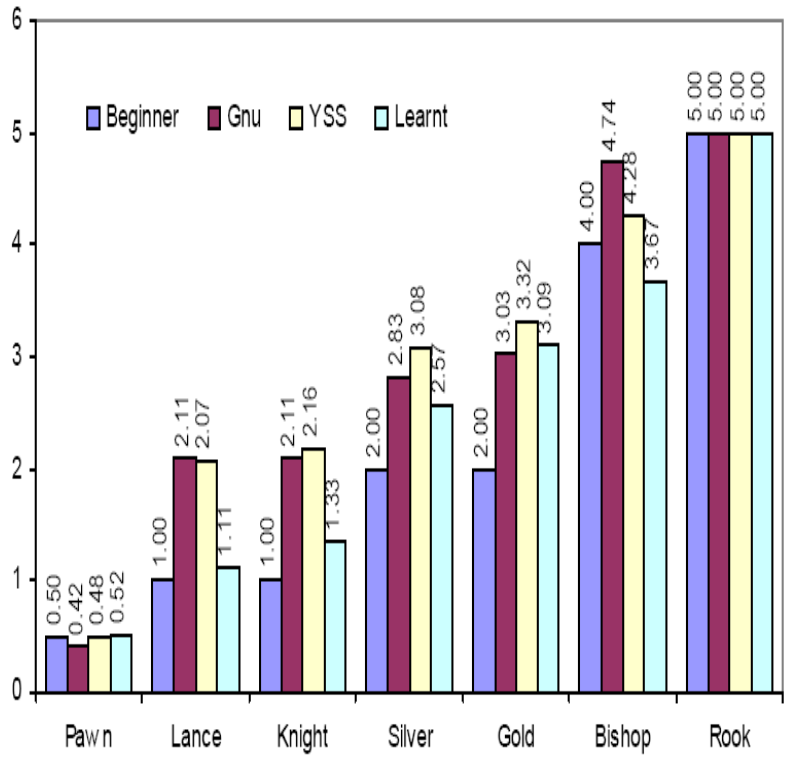

Fig. 6. Value sets tested in match play (main pieces)

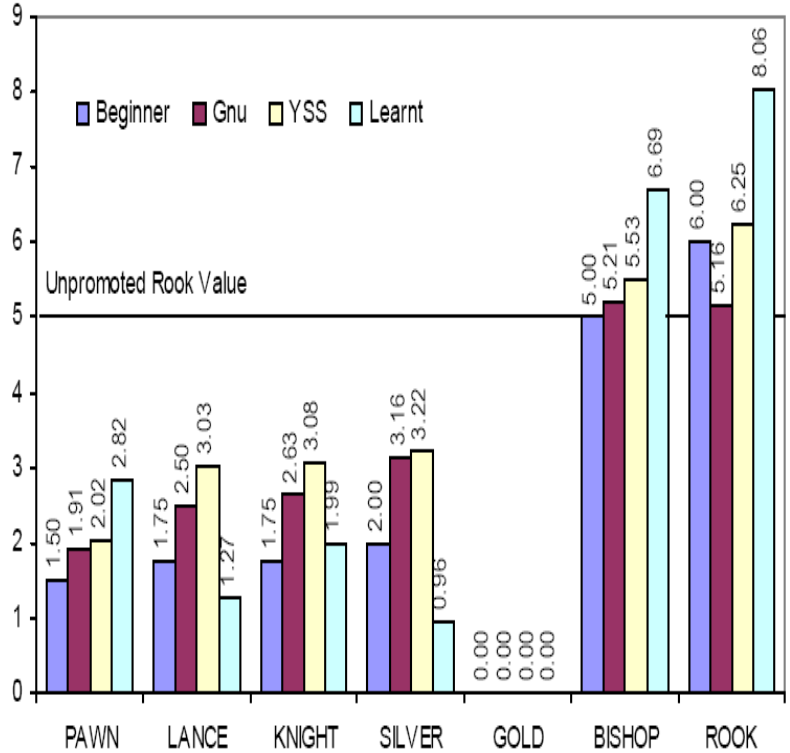

Fig. 7. Values sets tested in match play (promoted pieces)

Das war‘s

Danke für Ihre Aufmerksamkeit