Symbolisches Lernen in Go

Seminar Knowledge Engineering
und Lernen in Spielen, SS 04

Frank Steinmann

Motivation (1)

- Was kann gelernt werden?
- Globaler Ansatz:
  - eine Funktion
    f: Stellungen x Züge -> Belohnung
  - gut für kleine Brettgrößen
  - beim 19x19 Brett spezifische Methoden für verschiedene Subprobleme besser

Motivation (2)

- Suchstrategien verbessern
  - Programme verwenden oft lokale Suche, um ein bestimmtes Ziel in einem lokalen Teil des Spielbretts zu erreichen
  - typischerweise Verwendung von Alpha-Beta-Suche
  - Alpha-Beta-Suche profitiert von guter Sortierung der Züge
  - Lernen einer Heuristik zum Sortieren der Züge
  - Ermitteln der „Temperatur“ einer Stellung zur Vermeidung des Horizont-Effekts

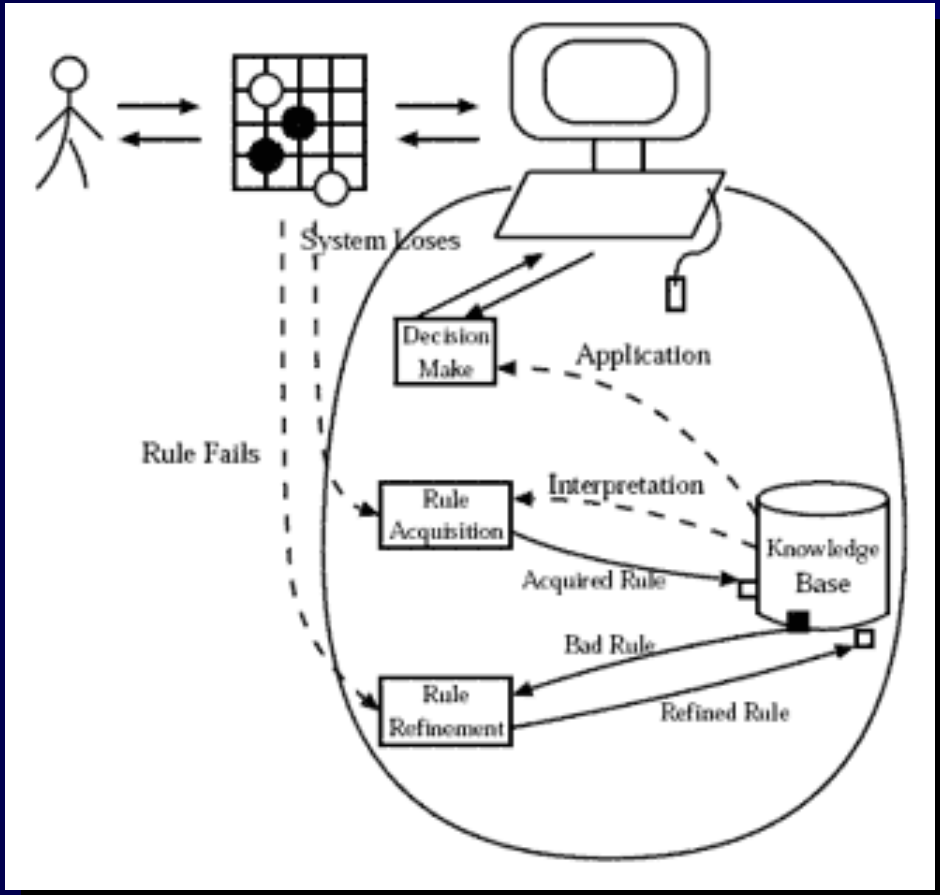
Ein deduktiver Ansatz (1)

- Ziel: striktes Wissen lernen
- verschiedene Regeln werden aus einem einzelnen Trainingsbeispiel gelernt
- vereinfachte Spielregeln: wer zuerst gegnerische Steine fängt, gewinnt

Ein deduktiver Ansatz (2)

- zwei Arten von Regeln
  - Basic Rules: Hintergrundwissen (Definition von Begriffen wie „fangen“, „atari“ etc.)
  - Forcing Rules: Zugregeln, die zum Sieg führen
- das System spielt (gegen irgendwen) und versucht, Forcing Rules zu lernen

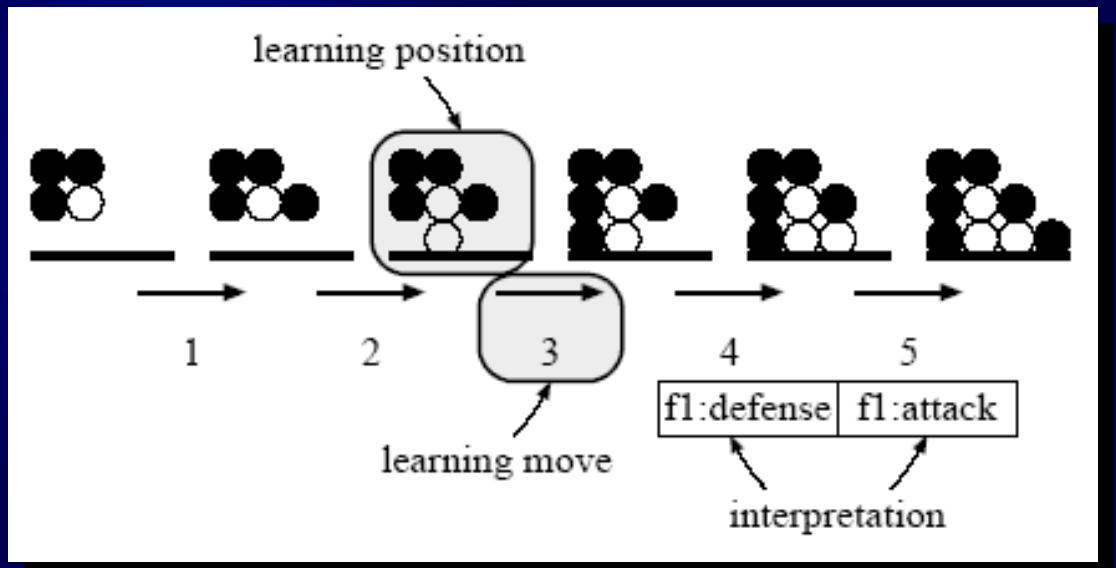
Das System im Überblick



- Decision Maker
  - wendet Regeln an
- Rule Acquisition
  - lernt Regeln
- Rule Refinement
  - verfeinert Regeln
- Knowledge Base
  - Aufbewahrung der Regeln

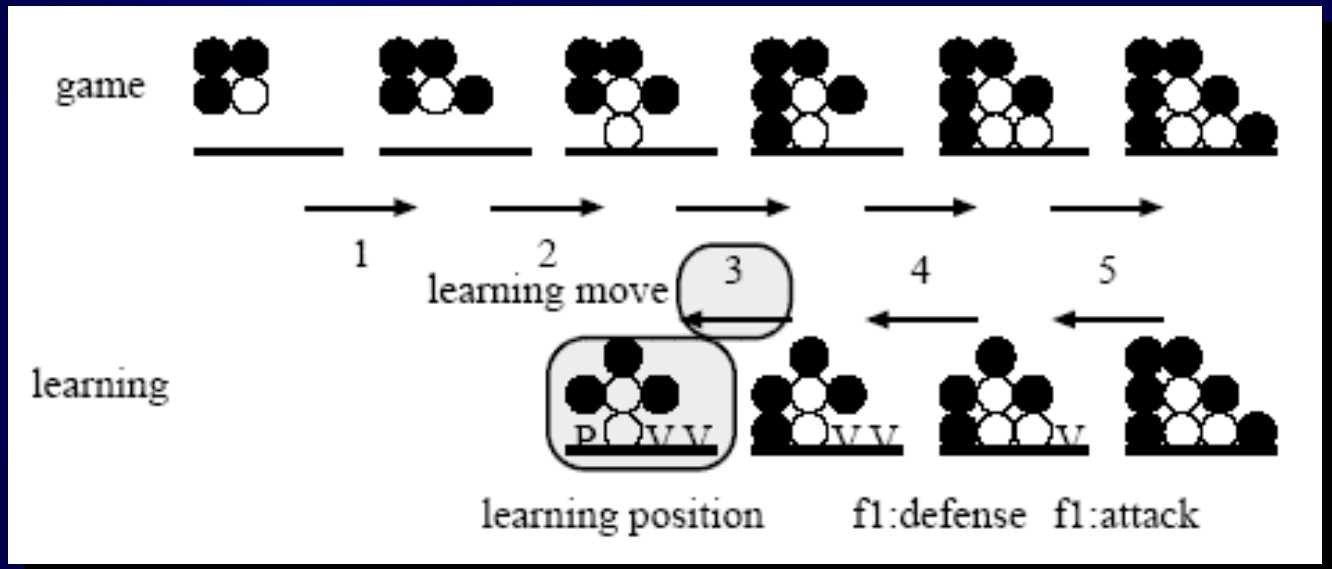
Rule Acquisition (1)

- Schritt 1:
  - versuche, die gemachten Züge mit Forcing Rules zu interpretieren, beginnend beim letzten



Rule Acquisition (2)

- Schritt 2:
  - Identifiziere für den zu lernenden Zug die relevanten Teile des Spielbretts



Rule Acquisition (3)

- Schritt 3:
  - Generalisierung der Stellung und des Zuges durch Änderung der Koordinaten von Konstanten zu Variablen

Ein evolutionärer Ansatz

- Ziel: nützliche (heuristische) Regeln lernen, die auf die gegebenen Trainingsdaten passen
- Regeln = *Individuen*
- Trainingsbeispiele = *Nahrung*
- jede Regel hat einen Aktivierungswert

Entwicklung von Regeln (1)

- Anfangszustand: keine Regeln
- Regeln, die auf ein Trainingsbeispiel passen, erhalten *Nahrung*, Aktivierungswert erhöht sich
- passt keine Regel, so wird eine neue Regel erzeugt, die auf das Beispiel passt und genau eine Bedingung enthält

Entwicklung von Regeln (2)

- Überschreitet ein Aktivierungswert einen bestimmten Grenzwert, wird die Regel in zwei geteilt:
  1. Originale Regel
  2. Komplexere Regel
- die komplexere Regel entsteht durch das Hinzufügen einer Bedingung
- Aktivierungswert wird auf beide Regeln aufgeteilt

Entwicklung von Regeln (3)

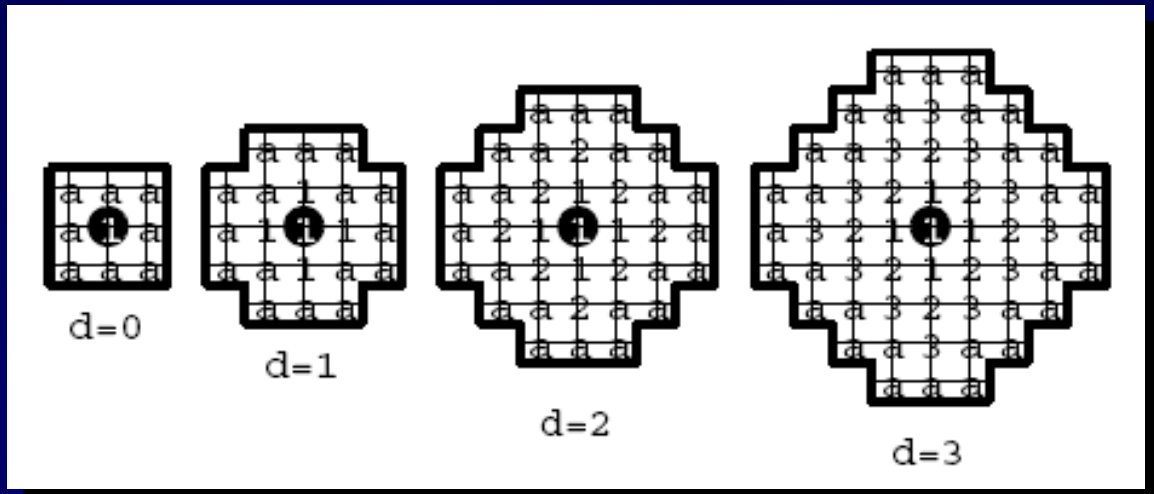
- In jedem Schritt wird den Regeln ein bestimmter Wert vom Aktivierungswert subtrahiert
- Regeln mit Aktivierungswert 0 sterben
- passen mehrere Regeln auf ein Beispiel, wird die *Nahrung* unter den Regeln aufgeteilt, für die unter den passenden Regeln keine speziellere existiert

Vergleichsalgorithmen

- Zwei Algorithmen, die sich durch die Struktur der Regeln unterscheiden:
  - Fixed Algorithm
  - Semi-Fixed Algorithm

Fixed Algorithm

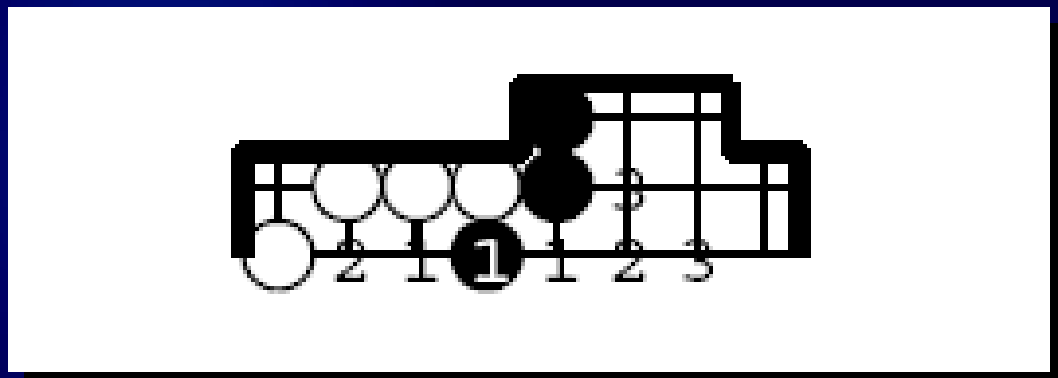
- Patterns einer festgelegten Form, variable Größe



- Lernen: Für jedes Beispiel werden Patterns jeder Größe abgespeichert

Semi-Fixed Algorithm

- Wie Fixed Algorithm, mit dem Unterschied, dass man von der Mitte aus maximal bis zum ersten nicht leeren Feld schaut



Regelauswahl

- Es werden meist mehrere Regeln auf eine Stellung passen
- Welche Regel bestimmt den Zug?
- Zwei Algorithmen:
  - Priority Assignment
  - Probability of Rule Accuracy

Priority Assignment

- Ziel: Gewichtung der Regeln
- Initialisierung: Alle Regeln erhalten Gewicht 200
- Für jeden möglichen Zug m: Gewichte der Regeln, die m vorschlagen, aufsummieren
- Züge in absteigender Reihenfolge nach Punkten sortieren
- Gewichte aller Regeln, die Züge vorschlagen, welche höher rangieren als der korrekte Zug, um 1 reduzieren
- Gewichte aller Regeln, die den korrekten Zug vorschlagen, um S/n erhöhen
- S = Summe der subtrahierten Gewichte
- n = Anzahl der Regeln, die den korrekten Zug vorschlagen

Probability of Rule Accuracy

- Ziel: Wahrscheinlichkeit ermitteln, mit der eine Regel den korrekten Zug liefert
- match : IF-Teil trifft zu
- hit : IF-Teil und THEN-Teil treffen zu
- Korrektheit für Regel i:

  a_i = Anzahl hits / Anzahl matches
- Wahrscheinlichkeit für einen Zug m:

  $A_m = 1 - \prod_{i \in R_m} (1 - a_i)$
- wobei R_m = Menge der Regeln, die m vorschlagen

Lernerfolg in Tsume-Go (1)

- 1039 Tsume-Go Probleme und ihre Antworten (insgesamt 3993 Züge) als Trainingsdaten
- 1. Schritt: Patterns lernen
- 2. Schritt: Patterns verfeinern
- zum Testen 3 Klassen von Problemen
  - einfache Probleme (100 Bsp.)
  - Probleme für 3 Dan (100 Bsp.)
  - Probleme für 5 Dan (100 Bsp.)

Lernerfolg in Tsume-Go (2)

- Gelöste Probleme in Prozent:

		first step		
		Flexible	Semi-Fixed	Fixed
second step	weights	31.0	13.3	11.0
	probabilities	25.0	15.0	13.3

- Wie hoch wird der korrekte Zug bewertet?

Rank	Basic	3 Dan	5 Dan	Average
1st	36 (36)	31 (31)	26 (26)	31 (31)
2nd	15 (51)	26 (57)	20 (46)	20 (51)
3rd	12 (63)	10 (67)	9 (55)	11 (62)
4th	10 (73)	7 (74)	14 (69)	10 (72)
5th	6 (79)	6 (80)	8 (77)	7 (79)

TILDE (1)

- TILDE = **T**op Down **I**nduction of **L**ogical **De**cision Trees
- Ziel: Lernen eine Heuristik, die Züge bewertet
- Ansatz: propositionale Repräsentationen (Attribut - Wert) können komplizierte Konzepte so nicht gut darstellen
- Besser relationale Repräsentation:
  - **BOARD(X, Y, GroupID)**: ordnet jedem Feld eine Gruppe zu
  - **GROUP(GroupID, Color)**: Gruppe von Steinen oder einzelnes leeres Feld
  - **LINK(Group, Adjacent_Group)**: benachbarte Gruppen

TILDE (2)

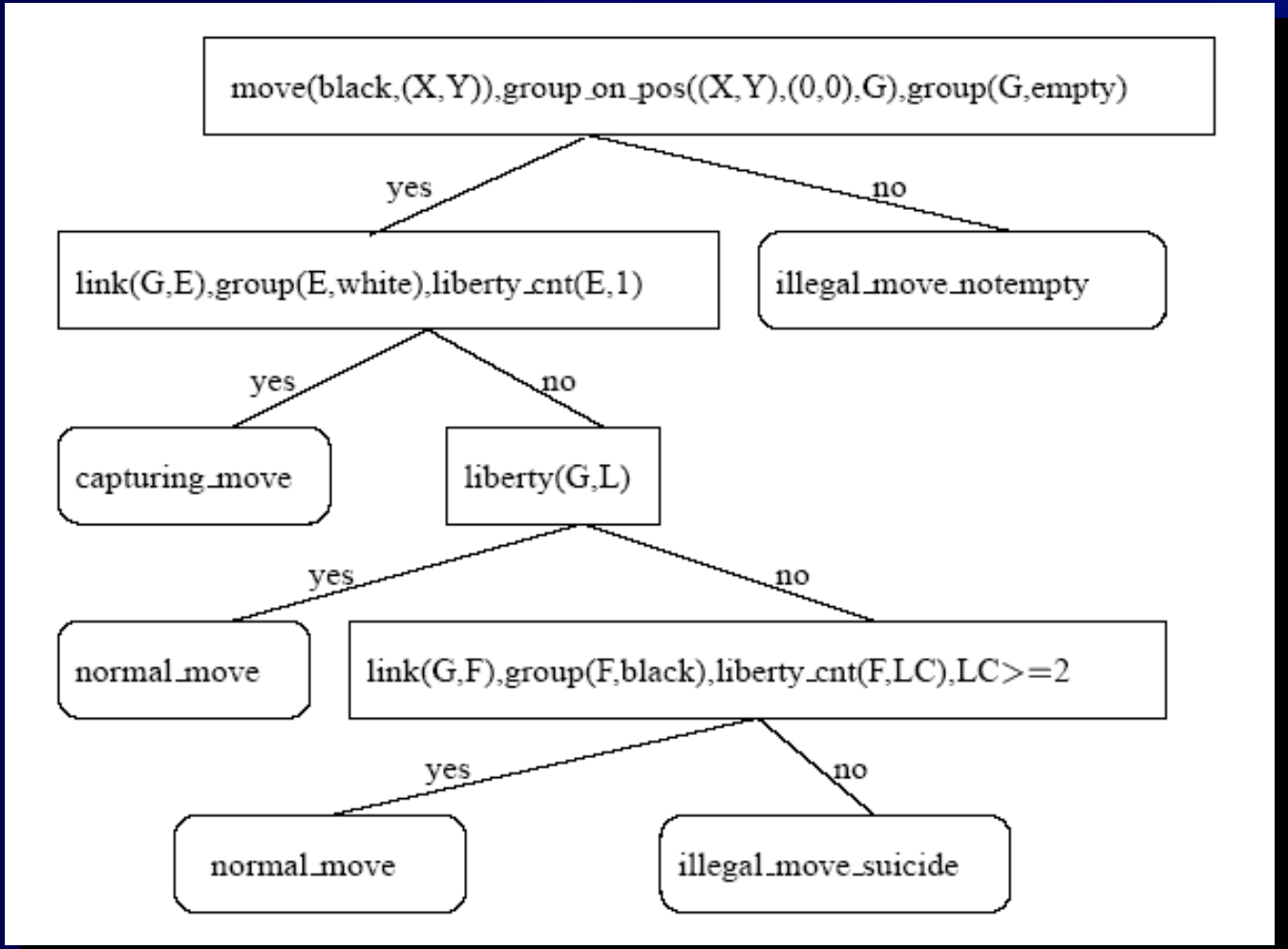
- Erzeugung weiterer Konzepte (Hintergrundwissen) mit Hilfe der gegebenen Relationen
- Beispiel:

```
liberty(Group,Liberty) :-
    link(Group,Liberty),
    group(Liberty,empty).
```

TILDE (3)

- TILDE ist eine Erweiterung des C4.5 Algorithmus
- Prädikatenlogik in den Tests in den Knoten
- Regression Mode zum Vorhersagen von reellen Zahlen anstelle von Klassen

Beispiel: Entscheidungsbaum



Lösen von Tsume-Go Problemen (1)

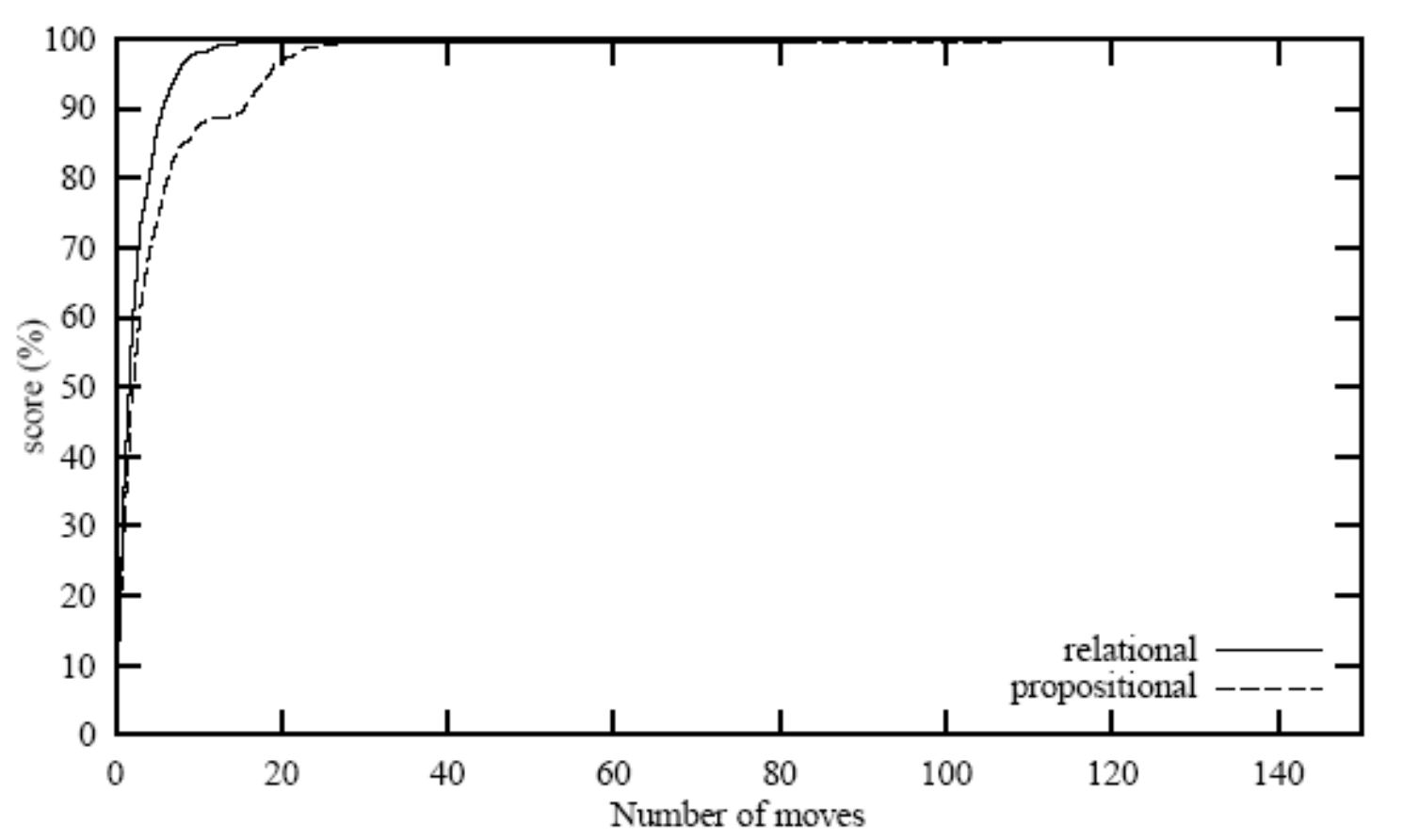
- 3600 Tsume-Go Probleme, davon 2600 zum Lernen, 1000 zum Testen
- Probleme können mehrere Lösungen haben, die geordnet sind nach ihrer Qualität
- Beste Lösung erhält Wert 1, zweitbeste 0.9 usw., Wert 0 für falsche Züge
- TILDE versucht, im Regression Mode diese Werte zu lernen
- Blätter des Entscheidungsbaums enthalten Mittelwert der Trainingsbeispiele

Lösen von Tsume-Go Problemen (2)

- Zwei Konfigurationen
  - relational (mit Hintergrundwissen)
  - propositional (nur Verwendung der Relation exists)

```
exists(RelativeCoordinate,Color) :-
    move(Side,Move),
    group_on_pos(Move,RelativeCoordinate,Group),
    group(Group,Color).
```

Ergebnisse (1)



Ergebnisse (2)

- Abschneiden der Algorithmen im Vergleich:

System	Testset(size)	1	2	3	4	5
Propositional Decision Tree	GoTools(1000)	29%	49%	61%	69%	73%
Relational Decision Tree	GoTools(1000)	35%	58%	73%	79%	87%
Flexible rules with Weights	Basic(100)	36%	51%	63%	73%	79%
Flexible rules with Weights	3 dan(100)	31%	57%	57%	74%	80%
Flexible rules with Weights	5 dan(100)	26%	46%	55%	69%	77%

Bewertung und Ausblick

- Die vorgestellten Algorithmen sind einigermaßen erfolgreich bei Tsume-Go
- Suche kann nicht ersetzt, aber unterstützt werden (durch gute Sortierung und/oder Vorauswahl der Züge)
- Nächster Schritt: Einsatz im richtigen Spiel (oder zumindest Teilen davon, z.B. Eröffnung, Endspiel)
- Schwierigkeiten: Strategisches Spiel, besonders auf dem großen Brett
- Besser geeignet zum Lösen lokaler Probleme

Quellen

- J. Ramon und H. Blockeel, *A survey of the application of machine learning to the game of go*, Proceedings of the First International Conference on Baduk (Sang-Dae Hahn, ed.), pp.1-10, 2001

- J. Ramon, T.Francis und H. Blockeel, *Learning a Tsume-Go Heuristik with Tilde*, Computers and Games, CG2000, Revised Papers (Marsland, T.A. and Frank, I., eds.), vol. 2063, Lecture Notes in Computer Science, pp.151-169, 2001

- Takuya Kojima und Atsushi Yoshikawa, *Knowledge acquisition from game records*, Prodeedings of the ICML-99 on Machine Learning in Game Playing, Bled, 1999