詰将棋における評価関数の自動獲得に向けて
– 評価関数中の論理式の効率的評価手法 –

金子 知適　　山口 和紀　　川合 慧
東京大学大学院 総合文化研究科 広域科学専攻
kaneko@graco.c.u-tokyo.ac.jp

概 要

将棋をはじめ大規模なゲームにおいて，評価関数を自動獲得する汎用的な手法はまだ確立されていない。著者らは評価関数中のゲーム固有の概念を論理式により表現し，その論理式を自動生成することにより評価関数を自動獲得することに取り組んでいる。しかしながら，この枠組みには，論理式を用いた局面の評価が非常に遅いため学習に必要な十分な量のデータが扱えないという問題点があった。この問題を解決するために著者らは前論文 [10] において，演繹データベースにおけるルールの評価手法にもとづき，論理式の評価を関係代数の計算として効率化する技術を提案した。本稿ではこの手法を詰め将棋に適用し，プロトタイプシステムによる実験で約 60 倍の高速化が確認されたことを報告する。

Toward Automatic Construction of Evaluation Function in Tsume-Shogi: An Efficient Evaluation Method for Logical Features

Tomoyuki KANEKO　　Kazunori YAMAGUCHI　　Satoru KAWAI
Graduate School of Arts and Sciences
The University of Tokyo
3-8-1 Komaba, Meguro-ku, Tokyo, 153-8902, JAPAN
kaneko@graco.c.u-tokyo.ac.jp

For mechanical acquisition of features for game state evaluation, researches[3][11] suggest that logical feature, that is the feature represented in logical formula, is promising. However, the evaluation of logical feature by Prolog based on the SLD-resolution is quite slow prohibiting it from being applied to more advanced games. This paper shows that our evaluation method proposed in [10] makes the evaluation approximately 60 times faster than the naive method when applied to Tsume-Shogi problems.

1　はじめに

本稿では局面に対して，その局面の“良さ”を数値化する関数を**評価関数**と呼び，評価関数中で，局面に対して評価値 (数値) を返すサブモジュールを feature と呼ぶ。評価関数は，複数の feature

```
f2(A):-owns(x,A). % PIECES FOR BLACK
```

図 1: Featureの例

の評価値を線形結合などのアルゴリズムで結合して局面の評価値とする。featureの例としてはオセロの「黒石の数」があげられる。この様にfeatureはゲーム固有の概念を表す部分とも言える。

従来の評価関数のパラメータ学習ではfeatureは人間が与えたものに固定されていたが，Zenithシステム [3] では，ゲームの遊び方と対戦による棋譜から，featureを自動的に生成し，評価関数の獲得・改良に成功している。これはfeatureの表現に論理式を採用したことによるものである。

しかしながら，featureの表現に論理式を採用したモデルでは[1]，局面評価が遅いという問題点がある。著者らは，この問題を解決するための新しいfeatureの評価方法を提案している [10]。本稿は第2節でfeature評価の問題点について説明し，第3節で著者らが提案した方法の概要を述べる。そして，第4節で詰め将棋にその手法を適用した結果を報告し，第5節でまとめを行う。

2 論理式の評価に関する問題

2.1 論理式を用いたfeature

featureの表現として論理式を用いるモデルはZenithシステム [3] で提案され，Metagame[4] でも使われている。このモデルでは，featureは「変数」と「論理式」から構成され,「featureの評価値」は論理式を満たす解集合 (その論理式を真とするような，変数に対する束縛) の大きさと定義される。以後，このモデルにおけるfeatureを単にfeatureと呼び，解集合の数え上げを「featureによる局面評価」と呼ぶ。論理式の中で使われる述語は，domain theory(ゲームの遊び方) で定義されるものと，駒の配置など局面そのものを表す述語がある。

Prologの記法で書いたオセロ [3] のfeatureの例を図1に示す[2]。この例では，Aが変数で，owns(x,A)が論理式である。domain theoryで，xは黒番プレイヤーで，owns(P,S)はオセロでプレイヤーPがマスSに石をおいていることを示す述語であると定義しておく。オセロの初期局面ではこのfeatureの評価値は2である。

2.2 Prologによる局面評価

ZenithやMetagameでは，featureによる局面評価にPrologを用いているが，Prologの処理系が単純にSLD-resolutionによりfeatureの評価を行うと，冗長な解集合の数え上げが頻繁に発生する。この冗長性は次の3点である。

1. 単独のルールの評価において，連言の解集合を求める場合，左にある項の異なる束縛を試す際に，右の項の計算が以前と同じ束縛になる場合があり冗長である。
2. 評価関数が局面を評価する際には多数のfeatureの評価を行うが，featureの論理式には共通する述語が数多くある。述語の解集合は同一の局面であれば同一であり，各featureの評価値

[1]他のfeatureの表現としては，例えばオセロの図形的パターン (Glem [1])，バックギャモンの升目そのもの [6] などがある。論理式の表現はより一般的であり，高度な学習が可能であると期待される。

[2]文献 [3] では f2(Num):-count([A],(owns(x,A)),Num). とfeatureの評価値を与える形式で記述されているが，本稿ではcountを除いてfeatureの内容のみを表す。

表 1: Prologで局面を評価した場合の冗長な解集合の計算 (1局面あたりの平均)

述語	bs	legal_move	span
呼び出し回数	1393.582	15.132	248640.576
独立な束縛	130	2	4380.806

の計算のたびに解集合を求めるのは冗長である。

3. domain theoryで定義されるルールには局面に依存するものとしないものがあるが，ゲームを通じて解が変化しない述語について，featureによる局面評価のたびに解集合を求めるのは冗長である。

1. 及び 2. の冗長性について，オセロの126個のfeatureと500局面を用いて実験的に測定した。局面に依存する3つの述語について，述語が呼び出された回数と，異なる束縛による呼び出しの回数を表1に示す。この結果から，述語bsは平均約1400回呼び出されているが，そのうち異なる束縛は平均130通りしかなく，10倍以上の冗長性があることが分かる。

3 演繹データベースによる局面評価

著者らは，演繹データベースのルールの評価方法を用いて，関係代数[7][8]によりfeatureの評価を行う手法を提案した[10]。以下，この手法をDDBと呼ぶことにする。DDBでは次の手順で局面を評価する。

1. 与えられた局面について，全てのルールについて解集合に相当する関係を求める。(3.1節参照)
2. 各featureに対して次の手順で値を評価する。
   (a) featureの論理式を関係代数式に翻訳する。(3.2節参照)
   (b) 翻訳された関係代数式の表す関係の大きさを計算する。(3.3節参照)

この手法では，手順1. でdomain theoryで定義される全てのルールに対応する解集合をあらかじめ求めて，以後それを用いるため，前節であげたような冗長な計算は発生しない。

3.1 演繹データベースによる解集合の計算

演繹データベースは3要素 $\langle F, R, C\rangle$ で構成される。F はPrologの事実の集合に対応し，R はPrologのルールの集合に相当する。C はintegrity constraintsであるがここでは考えない。演繹データベースでは，事実やルールに対する解集合を「関係」(relation)と呼び，質問に対して関係を返す。事実からルールにより導出される関係を求めるには，関係代数[7][8]を用いる。

ここでは，局面として与えられた事実をもとに，domain theoryで定義されたルールについて解集合を求め，関係に格納する。関係を求める手法には，top-down evaluation[8]を採用した。各ルールの関係を効率的に求めるために，ルール間の依存関係を調べ基本的なルールから先に関係を求める。同時に，後で行う自然接合のための索引を作成する。この時，解が局面に依存しないルールについてはこの関係と索引を局面にまたがって再利用することが可能である。

3.2 featureの論理式の関係代数式への変換

前節で計算した各述語に対応する関係を用いて，featureの局面評価を関係代数の演算により行う。featureの評価値はその変数と論理式に対する解集合の大きさであるから，その解集合に対応する関係の大きさ[3]を求めれば良い。この関係は，featureをルールとみなしてtop-down evaluationにより直接求めることができるが，ここでは次の節で述べる効率化を行うため関係代数式への変換と関係代数式の計算を分けて考える。

featureに対する関係を求める関係代数式は，論理式の各項に対する関係を自然接合し，featureの変数に射影するものである。項に対応する関係とは，簡単のためfunction symbolを用いない場合を考えると，項に対応するルールまたは事実に対応する関係から，項の引数に表れる定数に関する選択を施したものである[4]。

例として，fは以下の様に定義されたfeatureであるとする。Prolog同様，引数の中で大文字(A, B, C, D)は変数を，小文字(o, x)は定数を表すとする[5]。

```
f(A,B,C) :- span(A,B,C,o), neighbor(B,C,D), owns(x,D).
```

この時，f(A,B,C)に対応する関係は，span, neighbor, ownsに対応する関係，$\underline{\mathtt{span}}$, $\underline{\mathtt{neighbor}}$, $\underline{\mathtt{owns}}$を用いて，次の関係代数式から求めることができる。ここで，πは射影，σは選択，$\bowtie$は自然接合である。また関係の右肩の記号は属性を表し，X^nは匿名の属性を表す。

$$\pi_{A,B,C}(\sigma_{X^1=o}(\underline{\mathtt{span}}^{A,B,C,X^1}) \bowtie \underline{\mathtt{neighbor}}^{B,C,D} \bowtie \sigma_{X^2=x}(\underline{\mathtt{owns}}^{X^2,D})).$$

3.3 関係代数式の計算

前節で変換した関係代数式の表す関係を計算し，その大きさをfeatureの評価値とする。この時，式にあらわれる全ての関係の大きさが既知であることと，最終的に必要な情報は関係そのものではなくその大きさであることから，次の2点の工夫が可能である[10]。

- 自然接合の並び替え
- 共通属性のない自然接合の省略

4 詰め将棋への適用

4.1 詰め将棋の特徴

詰め将棋をゲームの題材として性能を測定した。前論文[10]ですでに効果が確認されているオセロと比較して，詰め将棋はゲームのルールが複雑でdomain theoryが大きいという特徴がある。両者におけるdomain theoryの比較を表2に示す。オセロのdomain theoryは文献[3]のもののうちfeatureによる局面評価に関する部分を抜粋した。詰め将棋のdomain theoryは新たに書き下した。

[3]ここでは関係の要素数(タプル数)を「関係の大きさ」と呼ぶ。

[4]function symbolを使う場合には，項の表す事実またはルールの関係から，ATOVアルゴリズム[8]により求める。

[5]このfeatureは文献[3]のオセロのdomain theoryを用いて生成したもので，oは白石，xは黒石，spanは連続する石の並び，neighborはマスの隣接関係を表している。

表 2: 詰め将棋とオセロの domain theory の比較

	ルールの数	事実の数	atom の数
詰め将棋	115	867	114
オセロ	20	527	74

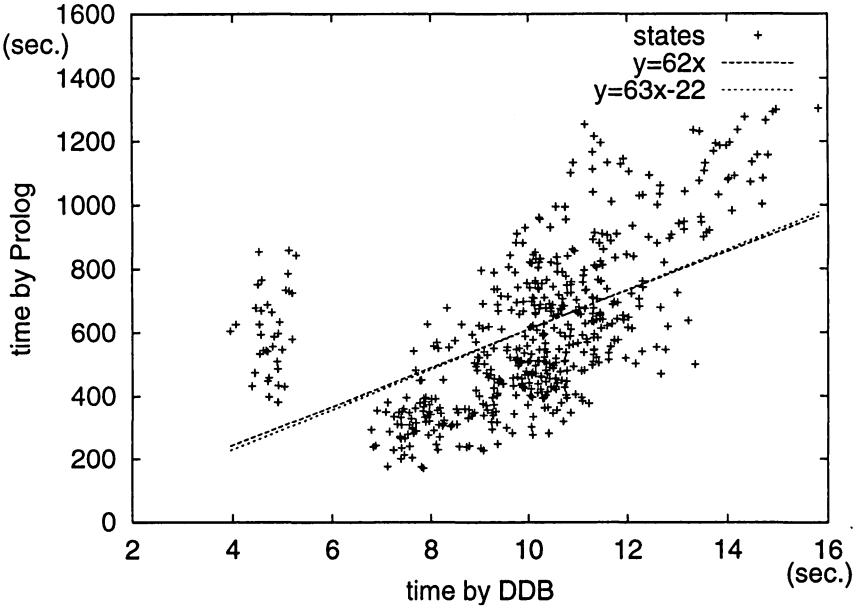

図 2: 局面評価にかかる時間の比較 (DDB vs. Prolog)
ゲーム:詰め将棋

4.2 性能測定

プロトタイプシステムにより DDB の評価を行った。C++[6] で実装した DDB と，Prolog[7] によるプログラムの局面評価にかかる時間の比較を図 2 に示す[8]。図の各点は局面の評価にかかった時間を表す。横軸が DDB，縦軸が Prolog での所要時間である。

評価する feature は，Zenith の手法で生成した 508 個の feature であり，1 つの feature は平均 7.0 個の項を含む。局面は詰め将棋の問題 20 題分から生成した 553 局面について測定した。図中の点線は，$y = ax$ 及び $y = ax + b$ について回帰を行ったものである。この図から，平均して約 60 倍の効率向上が確認された。

オセロについて同様の実験をした結果を図 3 に示す。評価する feature は，Zenith の手法で生成した 126 個の feature であり，1 つの feature は平均 3.9 個の項を含む。局面はシステムと人間との対戦 39 試合から生成した 10385 局面を用いた。

[6] gcc 2.95.1.
[7] SWI-Prolog version 2.9.10, available at ftp://swi.psy.uva.nl/pub/SWI-Prolog/
[8] CPU は Pentium Pro 200MHz, メモリーは 160MB の PC を用いた。

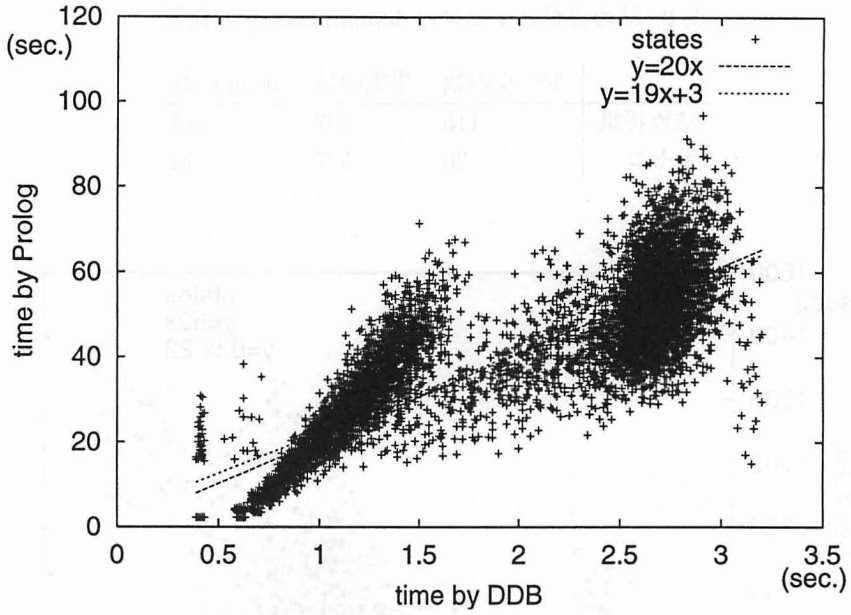

図 3: 局面評価にかかる時間の比較 (DDB vs. Prolog)
ゲーム:オセロ (文献 [10] の図 3 を C++の実装で再実験したもの)

詰め将棋では局面数が少ないが，全体的な傾向としては，オセロと同様に数 10 倍程度の高速化の効果があると言えるであろう。詰め将棋の方がオセロの場合より効果が高くなっているのは，使用した feature の数が多いため Prolog において冗長な計算の割合がより大きくなったためと思われる。

各局面について，ルールに対する関係の生成 (3.1 節参照) にかかった時間を図 4 に示す。横軸が評価全体にかかった時間，縦軸が関係の生成のみに要した時間である。オセロについて同じ内容を測定した結果を図 5 に示す。この結果から，詰め将棋の場合でもオセロの場合と同様に関係代数式の生成にかかるオーバーヘッドは 1 割程度に抑えられていることが分かる。局面に対して関係を求める時間は，オセロでは平均して 0.2 秒程度であったが，詰め将棋の場合 1 秒程度かかっている。この差は domain theory の大きさの差を表していると考えられ，間接的にゲームの難しさを示していると思われる。

5 まとめ

本稿では，feature を関係代数式に変換して評価する手法 DDB を詰め将棋に適用し，SLD-resolution を使った Prolog による局面評価に対して 60 倍程度の高速化が達成されることを示した。これにより詰め将棋の評価関数の自動獲得に於て大量の訓練例を扱うことが可能となり，よりよい評価関数を獲得することが期待できる。また，この結果とオセロに適用した結果をあわせて，DDB が特定のゲームによらず，有効な手法であることが確認された。

冗長な再計算を抑制する最も単純な方法は，一度計算した結果を表に記憶し 2 回目以降は表引

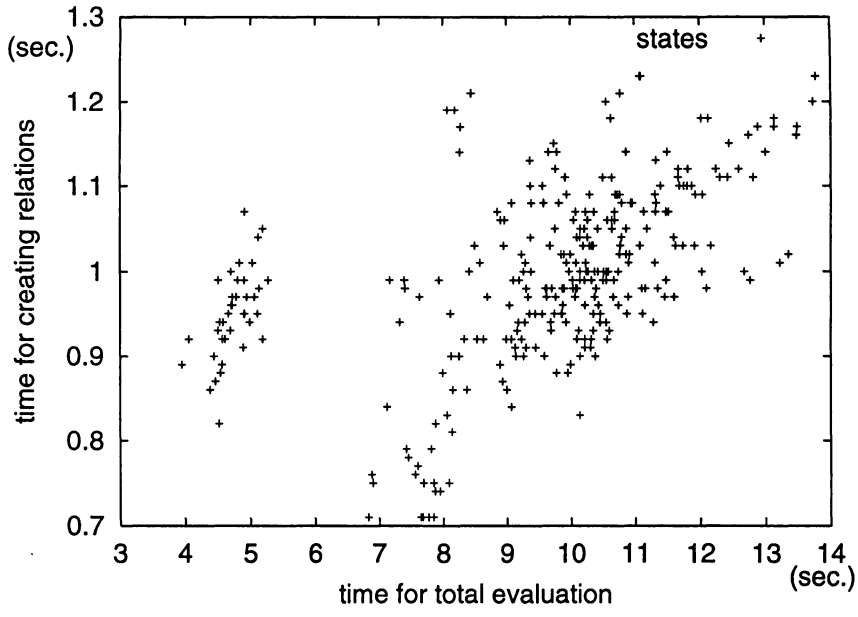

図 4: 関係代数式の生成にかかる時間と全体の評価時間との比較
ゲーム:詰め将棋

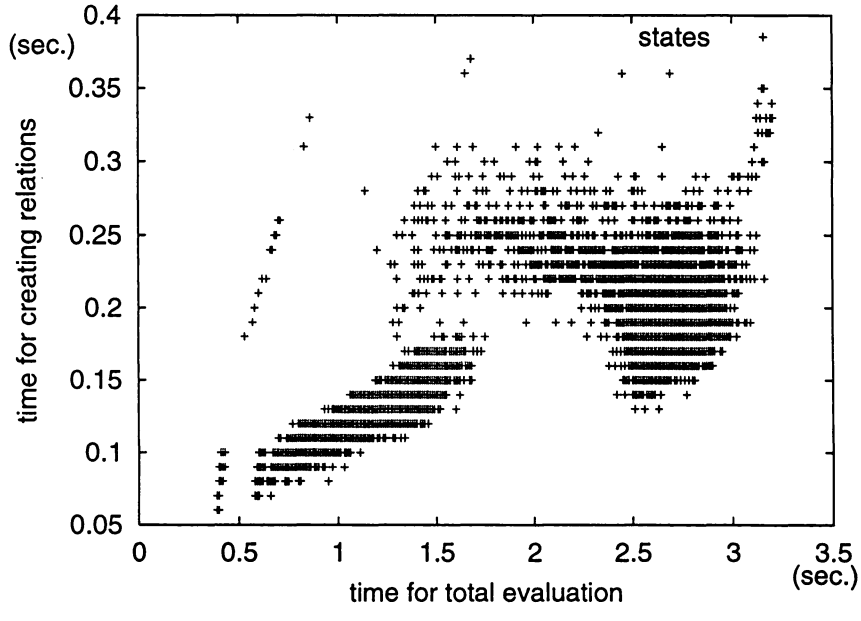

図 5: 関係代数式の生成にかかる時間と全体の評価時間との比較
ゲーム:オセロ (文献 [10] の図 4 を C++の実装で再実験したもの)

きにより再計算を省略すること (いわゆる memoization) である [5]。これに関しては，解集合をまとめて扱うという点と，3.3 節のような工夫が可能な点で，Prolog に memoization を加えたものよりもよりも DDB を用いるほうが有利だと考えられる。

一方 DDB では論理式を関係代数式に変換し解を求めるため，いわゆるメタ論理述語が使えないという制限がある。しかし feature の性質からそのような述語を使うことは考えにくく，事実，先行研究でも使われてないため，この制限は問題にならない考えている。

今後の課題であるが，論理式を用いない評価関数と比較すると，効率はなお改善の余地がある。現在，feature に表れる述語の folding による自然接合の省略や，view update[9] による局面間の類似性の利用を検討中である。一般に探索に於ては連続して評価する局面は類似性が高いため，view update の利用は有効だと思われる。

参考文献

[1] M. Buro. From simple features to sophisticated evaluation functions. In *Proceedings of the First International Conference on Computers and Games*. Springer-Verlag, 1998.

[2] Tom E. Fawcett and Paul E. Utgoff. Automatic feature generation for problem solving systems. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Conference on Machine Learning*, pp. 144–153. Morgan Kaufmann, 1992.

[3] Tom Elliott Fawcett. *Feature Discovery for Problem Solving Systems*. PhD thesis, Department of Computer Science, University of Massachusetts, 1993.

[4] Barney Darryl Pell. *Strategy Generation and Evaluation for Meta-Game Playing*. PhD thesis, University of Cambridge, 1993.

[5] Konstantinos Sagonas, Terrance Swift, and David S. Warren. Xsb as an efficient deductive database engine. *ACM SIGMOD*, Vol. 5, pp. 442–453, 1994.

[6] Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, Vol. 8, pp. 257–278, 1992.

[7] Jeffrey D. Ullman. *Prinsiples of Database and Knowledge-Base Systems, Volume I:Classical Database Systems*. Computer Science Press, Maryland, 1988.

[8] Jeffrey D. Ullman. *Prinsiples of Database and Knowledge-Base Systems, Volume II:The New Technologies*. Computer Science Press, Maryland, 1989.

[9] Toni Urpí and Antoni Olivé. A method for change computation in deductive databases. In *Proceedings of the 18th International Conference on VLDB*, pp. 225–237, 1992.

[10] 金子知適, 山口和紀, 川合慧. 関係代数を用いた feature 中の論理式の効率的評価方法. 情報処理学会研究報告 99-GI-1, pp. 7–14, 1999.

[11] 金子知適, 川合慧. 特徴集合の進化による評価関数の自動生成. 日本ソフトウェア科学会第 15 回大会論文集, pp. 409–412, 1998.