数当てゲーム MOO の最小質問戦略と最強戦略

An optimal MOO strategy

東京大学工学部 田中哲朗
Tetsuro Tanaka, Faculty of Engineering, University of Tokyo
tanaka@ipl.t.u-tokyo.ac.jp

概 要

数当てゲーム MOO の固定戦略のうち質問回数の平均を最小にする戦略 (最小質問戦略) を完全探索 (exhaustive search) によって求めた. 類似のゲームであるマスターマインドの最小質問戦略は [1] で報告されているが, 探索空間のより大きい MOO に関して求めたのは本研究が初めてと思われる.

MOO を対戦ゲームとしてとらえた場合には, 最小質問戦略が最強の固定戦略とは限らない. そこで, 最小質問戦略と対戦した時の勝率が最大である固定戦略を求め, 更にこの戦略と対戦して勝率が 0.5 を超える固定戦略がないことを確かめた. これによって, MOO が最強の固定戦略が存在するゲームであることがわかった.

1 MOO のルール

MOO は, Hit & Blow, Cow & Bull などさまざまな名で呼ばれ, 親しまれてきた数当てゲームである[1]. ゲームのルールは以下のようになっている.

- 2 人の対戦者が互いに異なる数字からなる 4 桁の数 (以下 MOO 数という) を紙に書いて, 両方の見えないところにしまっておく. “0586” のように最初の桁に 0 が来ても良い.
- 対戦者は相手の書いた数を当てるために, 交互に質問として MOO 数を提示する. 質問に対して出題した側は紙に書いた MOO 数との bull (場所も数字も同じもの), cow(数字は同じだが場所が異なるもの) の数を答える. 2 つの MOO 数から bull, cow の数を計算する操作を以下では MOO 積という. 相手の書いた MOO 数を当てると 4 bull となる.
- 4 bull となるまでの, 質問回数が少い方が勝ちとなる. 質問回数が等しい場合には引き分けとなる.

[1] 同名の MUD(Multi User Dungeon) ゲームもある.

質問者, 解答者を固定して片側だけから見たゲームの進行例を図 1 に示す.

正解	3951	回数	質問	応答
		1	0123	2C
		2	1245	2C
		3	2671	1B
		4	2850	1B
		5	9351	2B2C
		6	3951	4B

図 1: ゲームの進行例

MOO と良く似たゲームにマスターマインドというゲームがある. マスターマインドは数字 (色) の重複を許しているという点, 色の数が通常は 6 色と少ない点が異なっている. MOO はイギリスに由来するゲームでアメリカではマスターマインドの方が盛んだが, 計算機屋の中では MOO も知られていて, BSD 系の Unix には標準で MOO の出題プログラムが入っているものもある.

2 従来の研究

ランダムな MOO 数を生成し, 人間の質問に対して bull 数, cow 数を答える MOO 出題プログラムは早くから作られていた. [2] に 1971 年当時のケンブリッジ大のシステムが紹介されている. プレイヤーの平均質問回数によって順位をつけ, ハイスコア表示をするもので, ユーザの間で大流行し, 乱数生成アルゴリズムを解析して, 次の出題を予測するものすら現れたとある.

MOO 出題プログラムは, プログラミングの良い練習問題でもあり, [3] のようにプログラミングシンポジウムの課題プログラムになったこともあるし, [4] のように shell で書いた例もある.

一方, MOO をコンピュータにプレイさせる研究も [2] に紹介されている. MOO 数は全部で 5040 (${}_{10}P_4$) 個ある. いくつかの質問をした後で条件を満たす MOO 数の集合が求められるが, その集合に対して, 5040 個の MOO 数中の最良の質問をどう求めるかが強い MOO プレイプログラムを作るためのポイントである.

MOO 数の集合 π に対して, 質問として MOO 数 T を選んだ時, 応答は 4B, 3B, 2B2C, 2B1C, 2B, 1B3C, 1B2C, 1B1C, 1B, 4C, 3C, 2C, 1C, 0C の 14 種類があり, それぞれに応じて π は $t_1, ..., t_{14}$ に分割される. この分割を評価する関数 $f(T)$ を作って, それを最小化する T を質問として選ぶというプログラムがまず考えられる.

J. Larmouth は, 大きさ n の集合に対する質問回数が $\mathbf{log}(n)$ と見積もれることを利用して,

$$f(T) = \sum_{i\in[1,14],t_i\neq 0} |t_i|\mathbf{log}(|t_i|)(-2log2 \ \ \mathbf{if} \ \ T \in \pi)$$

という関数を選び, 平均質問回数 5.24 を達成した. また, B. Landy は

$$f(T) = max(|t_i|)$$

$$f(T) = \sum_{i=1}^{14} |t_i|\mathbf{log}(1+|t_i|)$$

$$f(T) = \sum_{i=1}^{14} |T_i|F(|t_i|) \ \ \mathbf{where} \ \ F(n) \ \ \mathbf{is} \ \ \mathbf{the} \ \ \mathbf{solution} \ \ \mathbf{of} \ \ \ x^x = n$$

などを提案している.

1987 年に bit のナノピコ教室の課題としてこの問題が出された [5]. この時の解答も $f(T)$ を作って, それぞれの集合に対する最良質問を選ぶものが主流だった. 優勝者のプログラムは, B. Landy の式を改良した式を用いて, 平均質問回数 5.22(総質問数 26347) を達成している.

3 完全探索による最小質問戦略の計算

$f(T)$ を計算する際に, 分割された集合 $t_1, ..., t_{14}$ のそれぞれについて 5040 個の質問をした際の集合の分割まで考えると, 計算量は増えるけれども, より良い質問を選ぶことができる. これは, 深さ 2 のゲーム木の探索問題と考えることができる.

これを更に発展させると, 最初の質問をおこなう前の 5040 個の MOO 数の集合に対して, 深さ無制限で各集合が大きさ 1 以下になるまでゲーム木の探索をおこなえば, 平均質問回数を最小とすることができることがわかる. 一旦, この探索を実行して各局面での最良質問を表としてもっておけば, ゲームをプレイするプログラムは非常に単純になる.

[2] でも, この完全探索による最小質問表の作成の可能性は示唆されているが, 当時の状況では

> This is far too expensive on computation.

であった.

その後の計算機の進歩によりマスターマインドに関する最小質問戦略は 完全探索を用いて求められた [1]. しかし, マスターマインドは各局面での質問の数が $6^4 = 1296$ なのに対して, MOO では ${}_{10}P_4 = 5040$ であり, 探索木のの

枝の数が 4 倍ほどあり, 平均質問回数が大きいことから, 探索木をより深くまで探索する必要があり, より計算時間がかかると考えられる.

本研究では,

1. 高速化に適したデータ構造の採用
2. 対称な質問の排除
3. 前処理により良い推測関数を求める

という点に特に注意を払って, プログラミングをおこなった. その結果, ワークステーション (Sparc Station 20) を使用して, 前処理に 60 時間, 最小質問戦略の計算に 30 時間程度の時間で最小質問戦略 (総質問数 26274) を求めることができた. 以下ではその方法の詳細をのべる.

4 探索の高速化

4.1 データの表現

MOO 数は, 26 ビットの整数で表現する. 下位 16 ビットは, 4 ビットで 10 進 1 桁を表現する packed decimal をとる. 上位 10 ビットは, 下位から順に 0-9 のそれぞれの数が MOO 数に含まれている時に 1 をセットしておく.

このように表現された MOO 数 a, b の bull 数を求めるには, `a ^ b` (^は排他的論理和) の下位 16 ビットを取って 10 進で 0 の桁数を数えれば良い. これには, 64K バイトの大きさのテーブルを用意しておけば十分である. cow 数を求めるには`a & b` の上位 10 ビットの 1 がセットされているビットの数を数えて, これから bull 数を引けば良い. これは, 1K バイトの大きさのテーブルで済む. このデータ表現に基づく MOO 積の計算は以下のようになる.

```
typedef int Question2;
int moo(Question2 q1,Question2 q2)
{
  int bull,cow;

  bull=bulltable[(q1^q2)&0xffff];
  cow=cowtable[(q1&q2)>>16];
  return(moo_product[bull][cow]);
}
```

SS20 上で実行時間を調べたところ, ナイーブなものと比較すると 3 倍程度の速度が出ていることがわかった. 5040x5040 の大きさのテーブルを作れば, テーブルの参照は 1 回で済むが, テーブル 1 つに 25M エントリ (1 エントリを 4 ビットで表現すれば 12.5M バイト) を必要とするので今回は見送った.

4.2 同値な質問の排除

集合に対する最良質問を考えるのに 5040 通りのすべてについて計算するのは無駄なことも多い．5040 通りの質問の中で同値な質問は行わないようにチェックをするようにした．以下のような置換の繰り返しによって到達しうる質問は同値な質問とする．

- 集合の中に出てこない数字は置換可能.

  ([1234,3456,1256] という集合では質問における [0,7,8,9] は同じとみなせる)

- これまでの質問に出てこなかった数字は置換可能.

  (1 回目の質問が 0123 で 2 回目の質問が 1234 でその結果得られた集合に対する質問では，[5,6,7,8,9] は同じと見なせる)

- これまでの質問列が, 位置, 数字の置換で自分自身に等しくなる置換が存在する場合は, 新たな質問に関しても同じ置換が可能.

同値な質問の中で, もっとも小さい数のみを残して他の質問は除去する．このチェックは以下のようにおこなう.

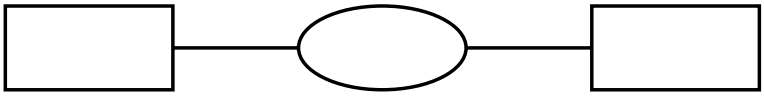
1. 質問が同値な数字の集合 $\{a_1, a_2, ..., a_n\}(a_1 < a_2 < ... < a_n)$ の要素を含む場合は, 上の桁から順に $a_1, a_2, ..$ のように現れているかをチェック.
2. これまでの質問列で, 場所, 数字の置換をほどこした結果同じ質問列になるような置換が存在する時 (総数はたかだか $4! = 24$), 質問に対して同様の置換をほどこした数の中で最小かどうかをチェック.

同値な質問の除去により, 1 回目の質問は “0123” に固定して考えることができる．同様に 2 回目の質問は 4567, 0456, 4056, 0145, 0415, 4501, 1045, 1405, 0124, 0142, 0214, 0241, 1204, 1240, 1023, 1032, 0231, 1230) の 19 通りに絞ることができる．3 回目, 4 回目の質問ではここまで効果はないが, 平均質問数は 1900 となり, 5040 と比較するとかなり減っている.

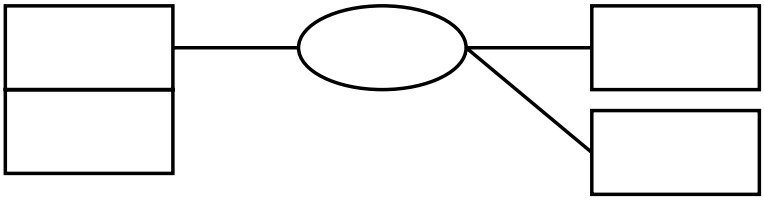
4.3 要素数 1,2,3 の集合の計算

要素数が 1,2,3 の集合に関しては, より高速に最良質問を求めることができる．それぞれについて以下に記す.

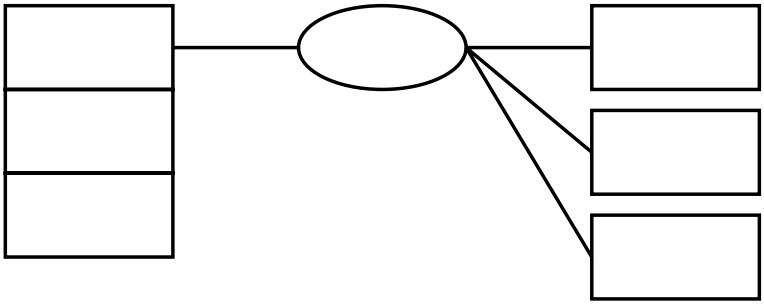
- 要素数が 1 の集合に対する最良質問はその要素そのものである．この時総質問回数は 1 になる．



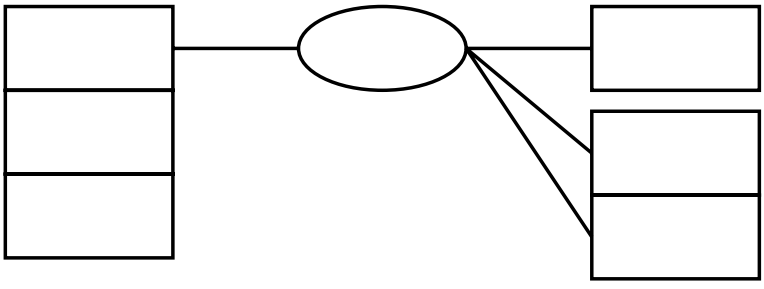
- 要素数 2 の集合の場合は 2 つのうちのどちらを選んで質問しても総質問は 3 になる．これが最良質問である．



- 要素数 3 の集合の場合は，2 通りの場合がある．
  3 つの要素の中の 1 つを質問としたときの総質問回数が 5 となる時はそれが最良質問になる．



  3 つとも総質問回数が 6 になるとしてもそれ以外の質問をした時の総質問回数は 6 を下回ることはないので 3 つのうちのどれかを質問するのが最良質問になる．



4.4 枝刈り

完全探索を実行するにしても，明かに最適でないような枝を深く探索するのは時間の無駄である．そこで，見込みのない枝の探索をしないように，枝刈りを行なった．

ある集合の最小総質問回数を計算する際，その局面でいくつかの質問をしたときの総質問回数が既に計算できていたとする．他の質問をした場合について計算する際，その時点での最小総質問回数よりも小さい総質問回数が得られるのでなければ計算する意味がない．

ある質問をした時の総質問回数は，その質問によって得られた部分集合の予想総質問回数の和と等しいかより大きい．したがって，その和よりも小さい値が既に計算されていたら, その質問は最良質問ではないことがわかる．

この枝刈を効率よく行なうため，5040 通りの質問は，その質問をしたときの部分集合の予想総質問回数の和によってソートしてその値が小さい枝から順に探索していくようにする．このソートには heap sort を用いる．取り出す要素数が小さいこのような目的には heap sort は有効である．

集合の要素数が決まった時に，要素数から総質問回数の下限が計算できる．たとえば, 要素数 n が $n \leq 14$ の時は, 最良の場合でも 1 回の質問であたる要素は 1 つで, 2 回の質問であたる要素は $n-1$ 個なので, 総質問回数の下限は, $1+2(n-1)=2n-1$ となる．

総質問回数の下限を良い精度で求めることは, 枝刈りを有効に働かせるための重要なポイントとなる．今回は, 集合の要素数だけでなく集合中に出て来る数字の種類 (4-10) も用いて総質問回数の下限を求めることにした．

これを求めるために, $n(4 \leq n \leq 10)$ 種類の数字を使ったすべての MOO 数の集合に対して, 深さ m までの要素 1 以上のノード数が最大となる戦略を求める探索をおこなうプログラムを作成した．10 種類の数字に関しては, 対称性を生かして, 深さ 3 でも 80 秒程度で求まったが, 9 種類の数字に対して深さ 3 の探索をおこなったところ, 59 時間程度かかった．結果を表 1 に示す．

表 1 から, たとえば数字の種類が 10 種類の時の総質問数の下限を表 2 のようにすることができる．

5 最小質問戦略の計算

以上の方針にしたがって, 最小質問戦略を求めるプログラムを作成した．ソースファイルは C 言語で 800 行ほどのプログラムである．1 回目の質問に対して, 14 の集合にわかれるので, 引数として番号 (0-13) を与えて, 各グループに関する計算をさせるようになっている．これで, 14 の集合の計算を別のワークステーションでおこなわせる形の並列性が得られるが, 各集合の計算時間が大きく異なるためあまり意味はなかったかもしれない．

表 1: 深さ n までの最大ノード数

種類	深さ 1	深さ 2	深さ 3
4	4	12	24
5	8	45	109
6	11	78	276
7	13	101	494
8	14	114	674
9	14	122	783
10	14	127	864

表 2: 10 種の数字からなる集合の総質問数の下限

要素の数 (n)	予想総質問回数
$1 \leq n \leq 14$	$2n - 1$
$15 \leq n \leq 127$	$3n - 15$
$128 \leq n \leq 864$	$4n - 142$
$865 \leq n$	$5n - 1006$

C コンパイラは gcc を用い, SS20 上で実行した. その結果, 総計算時間が 27 時間程度で総質問数 26274(平均質問回数 5.213) という最小質問戦略が求められた. 表 3 にその概要を示す.

最小質問戦略をそのままの形で提示すると, ページ数を超えてしまうので概略を伝えるため, 表 4 に, 5040 通りの MOO 数に対する質問回数の分布を示す.

6 最強質問戦略の計算

平均質問回数を小さくすることが強い MOO プログラムを作ることにつながることは確かだが, 平均質問回数が最小だからといって固定戦略を取るプログラムの中で最強であると言えるだろうか? これはもちろん言えない. MOO の対戦において, 結果は勝ち, 負け, 引き分けの 3 種類だけであって, 1 手差の勝ちも 2 手差の勝ちも価値は同じである.

最小質問戦略が最強の固定戦略かどうかを調べるために, 最小質問戦略と対戦した時の勝率が最大となるような戦略を求めた. ある質問回数で当てた際の最小質問戦略に対する勝率の期待値 (引き分けは 0.5 勝とする) を γ_n とした時, $\Gamma_n = \gamma_n \times 5040 \times 2 - 5040$ をその質問回数の評価値とする. この変

表 3: 最小質問戦略の結果

1 回目の応答	2 回目の質問	要素数	総質問数	計算時間 (時分秒)
4B		1	0	0.0
2B2C	0132	6	15	0.0
1B3C	0134	8	22	0.1
4C	1230	9	23	0.1
3B	0245	24	73	0.5
2B1C	0145	72	240	2.1
2B	0245	180	659	2:08.3
1B2C	0245	216	804	1:48.3
3C	1435	264	1004	5:00.5
0C	4567	360	1446	2.1
1B	0456	480	1913	2:52.5
1B1C	0245	720	2992	9:28.6
2C	1245	1260	5548	2:12:29.7
1C	1456	1440	6495	24:24:47.2

表 4: 最小質問戦略の分布

質問回数	1	2	3	4	5	6	7
度数	1	7	63	697	2424	1774	74

形によって整数演算だけで計算ができる. これを表 5 にしめす.

表 5: 評価関数

質問回数	1	2	3	4	5	6	7	8
勝ち	5039	5032	4969	4272	1848	74	0	0
負け	0	1	8	71	768	3192	4966	5040
引き分け	1	7	63	697	2424	1774	74	0
評価値 (Γ_n)	5039	5041	4961	4201	1080	-3118	-4966	-5040

最小質問戦略を求める手直しして, 評価値が最大になるような戦略 (以下, 最高勝率戦略) を見つけるプログラムを作った. この実行は 58 時間ほどかかった. 最小質問戦略を求める場合と比較して, 評価値を乗算で求める必要があったことと, 枝刈りがそれほど効かなかったため時間がかかっている. この概要を表 6 に示す.

表 6: 最高勝率戦略の概要

1 回目の応答	2 回目の質問	要素数	総質問数	計算時間 (時分秒)
4B		1	0	0.0
2B2C	0214	6	16	0.5
1B3C	0134	8	22	0.5
4C	1034	9	24	0.5
3B	0456	24	74	1.2
2B1C	0145	72	240	17.4
2B	0456	180	661	4:01.8
1B2C	0245	216	804	3:48.9
3C	1435	264	1004	7:29.5
0C	4567	360	1449	10.2
1B	0456	480	1915	38:18.1
1B1C	0145	720	2996	1:46:15.2
2C	1245	1260	5555	16:23:42.8
1C	1456	1440	6512	38:56:42.9

これは, 平均質問回数が 5.221 回 (総質問回数 26312) であり, 最小質問戦略ではないが, 最小質問戦略に対する勝率は, 50.20823 % である.

最小質問戦略に対して, 勝率が 50 %を超える最高勝率戦略があることがわ

かったとして, 最高勝率戦略に対する勝率が 50%を超える戦略があるのではないかという可能性もある. 戦略 A, B, C があったとして, $A > B, B > C, C > A$ のようなジャンケンのような構造になっていて, 最強の戦略はないという予想も立てられる.

しかし, この予想は実験によって否定された. 最高勝率戦略に対する勝率が最大となる戦略を再び 52 時間かけて求めたところ, 最高勝率戦略に対して勝率を最大にする戦略は最高勝率戦略で, 勝率は 0.5 を超えないことがわかったからである. したがって, 最高勝率戦略は最強戦略となっていることがわかった.

最小質問戦略と最強戦略の分布を並べたのが表 7 である. 最強戦略は平均値が大きい代わりに 5 回以内に当てる場合が多くなっていることがわかる.

表 7: 最小質問戦略と最強戦略の分布の比較

質問回数	1	2	3	4	5	6	7	8
最小質問戦略	1	7	63	697	2424	1774	74	0
最強戦略	1	4	47	688	2531	1628	141	0

7 検証

プログラムは C 言語で, 850 行ほどで, 他に枝刈りパラメータの決定プログラムが 650 行ほどある. プログラムは何度もチェックしたが, プログラムにバグがある可能性は否定できない. プログラム及び得られた戦略は, WWW 上 (`http://www.ipl.t.u-tokyo.ac.jp/~tanaka/moo/moo.html`) で公開するので, バグを見つけた人がいたら教えて欲しい. ただし, Knuth が TeX に対してしているように, バグレポートに対して賞金を出すことは考えていない.

得られた結果が, 正しい戦略になっていることは, 戦略の出力を独立に書かれた Lisp プログラムでチェックして確かめられたが, 最小質問戦略であるかどうかをチェックは難しい. 円周率の計算では別の公式を使った別プログラムで計算して結果を比較するという手法をもちいるとのことだが, ここで用いた枝刈り等のテクニックを用いずに, 許容できる計算時間で求めることができるかどうかは分からない.

8 まとめ

MOO の固定戦略の最小質問戦略と最強戦略を求めた. ただし, 最強戦略といっても固定戦略の中で最強なのであって, 相手の手の進み具合を見て手を

変更する動的な戦略ならば, 最強戦略に対して勝ち越すことも可能であろう. たとえば,

1. 先攻のプレイヤーが当てた後は, 後攻は解の集合から手を選ぶ

   最小質問戦略, 最強戦略では, 評価値を最大にするために解の集合に含まれない MOO 数を提示することがあるが, 先攻が当ててしまった場合には解の集合の中から選ばないと, 必ず負けてしまう.

2. 相手の戦略を予想して, 早い段階で正解までの質問回数を予測し, こちらの戦略を決定する.

といった動的戦略が考えられる. ただし, こちらが正解までの質問を n 回と判断したかを相手が察知してしまうと, 自分の戦略で n 回で正解になる集合に解が含まれると判断して, より早く数を当ててしまうかもしれない. 更に, それを見越して戦略を変えることも考えられる. まだまだ, このゲームの奥は深いかもしれない.

参考文献

[1] Kenji Koyama, Tony W.Lay: An optimal Mastermind Strategy, J. Recreational Mathematics, Vol. 25, No. 4, pp. 251-256 (1994).

[2] $\aleph_0$: *Computer Recreations.* SOFTWARE, Vol. 1, No. 2, pp. 201-204 (1971).

[3] “プログラミング宿題問題解答 MOO”, よいプログラムを作るにはシンポジウム報告集, pp. 93-125 (1979).

[4] srekcah@sra.junet: “自由きままに Shell プログラミング (2)”, ユニックスマガジン, Vol.1, No.2, pp. 80-83 (1986).

[5] 吉川竹四郎: “ナノピコ教室解答”, bit, Vol.19, No.7 (1987).

[6] Donald E.Knuth: The Computer as Master Mind, J. Recreational Mathematics Vol. 9, No. 1, pp. 1-6 (1976-77).