Strategies for playing MOO, or “Bulls and Cows”

http://www.jfwaf.com/Bulls%20and%20Cows.pdf

John Francis
johnf@panix.com

A discussion of the game of Bulls and Cows (and similar games), showing how to derive an optimal strategy for playing the most common forms of the game.

Revision History

January 2010	Initial Version

Table of Contents

Introduction

Bulls and Cows is an old number-guessing game for two players.

Each player thinks of a (usually 4-digit) secret number with no repeated digits.

The other player then attempts to guess the number. For each guess the player is told the number of digits that match and are in the correct place (bulls), and also the number of digits that are in both numbers but in different positions (cows).

For example: if the secret number is 2170, and the guess is 0123, the response would be 1 bull and two cows (the bull being "1", and the cows "0" and "2").

The most common form of the game is played with 4-digit numbers, but other pattern lengths have been used, together with variations in the number of different symbols. Examples can be found in the game *Mastermind®*[1] (or *Master Mind*) which uses coloured pegs rather than digits. The number of distinct colours is typically either 6 or 8, and the number of pegs either 4 or 5. Mastermind is usually played without the restriction that all peg colours in a pattern be distinct.

History

Bulls and Cows has been played as a paper-and-pencil game for a century or more. I first played a computer version in 1968 on Titan, the Cambridge University Atlas, where the game was known as "MOO". At that time the top place in the ladder of competitors was held by a computer program, written by John Larmouth, which used a simple predictor algorithm to generate a good strategy (except for the occasional brief period when one of the computer science students found a way to defeat the file protection on the league table, but that's a topic for another time). While the strategy chosen by Dr. Larmouth's program was very good, it wasn't optimal, and at the time finding the optimal strategy was an unsolved problem. Over the next decade I returned to this question from time to time, and before long I had a strategy which I believed to be optimal, although still unable to prove it. I eventually succeeded in a complete analysis of the "Bulls and Cows" game tree in 1978 or 1979 (using around two months of CPU time on a DECSYSTEM-2050), and confirmed my belief that my earlier strategy was, indeed, optimal. Since then I've continued to refine the predictors used in the tree pruning algorithm, and computers have got a lot faster, so the complete analysis now runs in a matter of minutes, not months.

[1] A registered trademark of Pressman Toy Corporation, used by agreement with Invicta Toys and Games, Ltd., UK

Earlier Works

The first computer implementations of Bulls and Cows were the MOO program written by Dr. Frank King at Cambridge in 1968, a version for the TSS/8 time sharing system written by J.S. Felton, and a 1970 version written for the Multics system at MIT by Jerrold Grochow. The game was introduced to a wider audience in the "Computer Recreations" column of the Apr-Jun 1971 issue of "*Software, Practice & Experience*"[1]. This article described several strategies for playing MOO – the one mentioned earlier used by Dr. Larmouth in his league-leading MOO-playing program, and three other strategies investigated by Mr. B. Landy (although the estimated performance given for the third of these algorithms is a little on the optimistic side). A later article[2] in *Software* by Grochow concentrates on the problem of protecting the league table, with no discussion of the game itself.

In 1976 Donald E. Knuth published a short article[3] in the *Journal of Recreational Mathematics* showing that the game of Mastermind could be solved in at most five guesses (and including a description of a strategy to do so), but noting that the minimum expected game length was unknown. A strategy with a slightly better expected game length was published by R.W. Irving in 1978[4], but it was not until 1993 that Kenji Koyama and Tony W. Lai published their paper[5] detailing both an optimal strategy (which solved all but one case in five guesses) and a strategy taking at most five guesses with only a minimal increase in expected game length.

Very little seems to have been written on MOO or Bulls and Cows in the period after the original 1971 *Software* article up until 1996, when Tetsuro Tanaka of the University of Tokyo presented a paper[6] at the Game Programming Workshop describing a strategy with an expected game length of 5.213 (26274/5040)[2]. In this article Tanaka also noted that a strategy which produces a minimum expected game length is not the optimal strategy for head-to-head play between two human opponents when a win by any number of moves scores the same as a win by a single move. He provided a strategy with a slightly higher expected game length of 5.224 (26312/5040) but which will win in head-to-head competition against the minimum-expected-game-length strategy.

[2] This matches the value that I had found.

Terminology

In the subsequent discussion the following terminology will be used. All formulae will be given for the general case, but specific examples will be for the traditional “Bulls and Cows” game (using 4-digit decimal numbers with no repeated digits)

$\boldsymbol{p}$	The number of places in the pattern being guessed
$\boldsymbol{d}$	The number of different symbols (or colours) available
$\boldsymbol{u}$	The number of distinct symbols used in all previous guesses
$^{\boldsymbol{n}}\boldsymbol{P}_{\boldsymbol{r}}$	Number of permutations of $\boldsymbol{r}$ objects chosen from $\boldsymbol{n}$ possibilities
$^{\boldsymbol{n}}\boldsymbol{C}_{\boldsymbol{r}}$	Number of combinations of $\boldsymbol{r}$ objects chosen from $\boldsymbol{n}$ possibilities
$\boldsymbol{r}$	The number of different responses that are possible (see below)

Any potential guess partitions the solution space into different ***response classes***. The maximum number of different responses to any guess is (p + 1)•(p + 2)/2 – 1 (the ‘-1’ coming from the fact that it is impossible to have a response that indicates (p – 1) Bulls and one Cow; if all p symbols in a guess are in the answer, and (p – 1) of them are in the correct place, then by the pigeonhole principle the final symbol has to go in the one remaining position, and thus must also be in the correct place)

Overview

The game tree is analysed using a best-first search algorithm with alpha-beta tree pruning. An estimate of the outcome is made for each potential guess, and this is used to determine the order in which the subtrees are evaluated. The efficiency of this approach is extremely dependent on the quality of the heuristic used to predict the outcome of a particular guess; the more precise the estimate, the more likely it is that an optimal subtree will be evaluated at an early stage, and the larger is the number of subtrees that can be eliminated without requiring any further evaluation.

The normal definition of an optimal strategy is one that minimises the expected number of guesses to discover the secret pattern. A slightly different problem is to find a strategy that minimises the maximum number of guesses taken. Much of the analysis remains the same for either definition; any differences will be noted at the appropriate point in the following discussions.

Equivalences

An easy way to reduce the branching of the game tree, at least for the first few levels, is to group potential guesses into equivalence classes. The simplest of these equivalences is to treat two sub-trees as equivalent if one can be derived from the other by a systematic renaming of the different symbols. For the initial try this means that we only need to consider one possible guess (0123 for Bulls and Cows). On subsequent tries we only need to consider one guess for each of the possible ways of filling ***n*** places using symbols from the ***u*** distinct symbols used so far; all the ways of filling the remaining (***p*** – ***n***) places from the (***d*** – ***u***) new symbols are equivalent. This gives the following formula in the general case:

$$\sum_{n=\max(0,p-(d-u))}^{n=p} {}^{u}P_{n} \cdot {}^{p}C_{n} = \sum \frac{u!}{(u-n)!} \cdot \frac{p!}{(p-n)! \cdot n!}$$

For the specific example of Bulls and Cows the number of distinguishable tries for each of the possible values of ***u*** is shown in the following table:

$$\begin{aligned} {}^{4}P_{0} \cdot {}^{4}C_{0} &= 1 \cdot 1 = 1 \\ {}^{4}P_{1} \cdot {}^{4}C_{1} &= 4 \cdot 4 = 16 \\ {}^{4}P_{2} \cdot {}^{4}C_{2} &= 12 \cdot 6 = 72 \\ {}^{4}P_{3} \cdot {}^{4}C_{3} &= 24 \cdot 4 = 96 \\ {}^{4}P_{4} \cdot {}^{4}C_{4} &= 24 \cdot 1 = 24 \end{aligned}$$

====
209

$$\begin{aligned} {}^{5}P_{0} \cdot {}^{4}C_{0} &= 1 \cdot 1 = 1 \\ {}^{5}P_{1} \cdot {}^{4}C_{1} &= 5 \cdot 4 = 20 \\ {}^{5}P_{2} \cdot {}^{4}C_{2} &= 20 \cdot 6 = 120 \\ {}^{5}P_{3} \cdot {}^{4}C_{3} &= 60 \cdot 4 = 240 \\ {}^{5}P_{4} \cdot {}^{4}C_{4} &= 120 \cdot 1 = 120 \end{aligned}$$

====
501

$$\begin{aligned} {}^{6}P_{0} \cdot {}^{4}C_{0} &= 1 \cdot 1 = 1 \\ {}^{6}P_{1} \cdot {}^{4}C_{1} &= 6 \cdot 4 = 24 \\ {}^{6}P_{2} \cdot {}^{4}C_{2} &= 30 \cdot 6 = 180 \\ {}^{6}P_{3} \cdot {}^{4}C_{3} &= 120 \cdot 4 = 480 \\ {}^{6}P_{4} \cdot {}^{4}C_{4} &= 360 \cdot 1 = 360 \end{aligned}$$

====
1045

$$\begin{aligned} {}^{7}P_{1} \cdot {}^{4}C_{1} &= 7 \cdot 4 = 28 \\ {}^{7}P_{2} \cdot {}^{4}C_{2} &= 42 \cdot 6 = 252 \\ {}^{7}P_{3} \cdot {}^{4}C_{3} &= 210 \cdot 4 = 840 \\ {}^{7}P_{4} \cdot {}^{4}C_{4} &= 840 \cdot 1 = 840 \end{aligned}$$

====
1960

$$\begin{aligned} {}^{8}P_{2} \cdot {}^{4}C_{2} &= 56 \cdot 6 = 336 \\ {}^{8}P_{3} \cdot {}^{4}C_{3} &= 336 \cdot 4 = 1344 \\ {}^{8}P_{4} \cdot {}^{4}C_{4} &= 1680 \cdot 1 = 1680 \end{aligned}$$

====
3360

$$\begin{aligned} {}^{9}P_{3} \cdot {}^{4}C_{3} &= 504 \cdot 4 = 2016 \\ {}^{9}P_{4} \cdot {}^{4}C_{4} &= 3024 \cdot 1 = 3024 \end{aligned}$$

====
5040

$${}^{10}P_{4} \cdot {}^{4}C_{4} = 5040 \cdot 1 = 5040$$

====
5040

There is one other optimisation that is worthwhile at earlier levels of the game tree. Consider permutations of the order in which places are filled (with an associated renaming of symbols) which are consistent with all prior guesses. This defines an equivalence relationship over the list of potential guesses at the next level. For Bulls and Cows this gives 20 equivalence classes at the second level:

Cycle Diagram	**Guess**	**Response**	**Before**	**After**	**Count 1**	**Count 2**
(•)(•)(•)(•)	0123	BBBB			1	1
(•)(•)(• •)	0132	BBCC	209	67	6	6
(•)(• • •)	0231	BCCC	209	75	8	8
(• •)(• •)	1032	CCCC	209	39	3	3
(• • • •)	1230	CCCC	209	59	6	6
(•)(•)(•)(• ◦)	0124	BBB	501	107	4	24
(•)(•)(• • ◦)	0134	BBC	501	270	12	72
(•)(• •)(• ◦)	0214	BCC	501	270	12	72
(•)(• • • ◦)	0234	BCC	501	—	24	144
(• •)(• • ◦)	1034	CCC	501	270	12	72
(• • •)(• ◦)	1204	CCC	501	175	8	48
(• • • • ◦)	1234	CCC	501	—	24	144
(•)(•)(• ◦)(• ◦)	0145	BB	1045	295	6	180
(•)(• • ◦)(• ◦)	0245	BC	1045	—	24	720
(• •)(• ◦)(• ◦)	1045	CC	1045	295	6	180
(• • • ◦)(• ◦)	1245	CC	1045	—	24	720
(• • ◦)(• • ◦)	1435	CC	1045	541	12	360
(•)(• ◦)(• ◦)(• ◦)	0456	B	1960	373	4	480
(• • ◦)(• ◦)(• ◦)	1456	C	1960	1012	12	1440
(• ◦)(• ◦)(• ◦)(• ◦)	4567	NONE	3360	218	1	360
					209	5040

The first few rows in this table show the different ways of partitioning the ***p*** places (represented by • glyphs) into cycles. Subsequent rows are created by extending a cycle to include a selection from the pool of unused symbols (represented by the glyph ◦), subject to the restriction that any cycle can include at most one ◦ entry.

A representative guess for each equivalence class is shown, together with the response class to an initial try of 0123 that each guess falls into (note that there are several cases where two different cycle diagrams map to the same response class). Next is the count of distinguishable and equivalence-reduced third-level guesses.

The final two columns are purely informational, and show the number of examples of each equivalence class which can be found in the shortest list of distinguishable guesses (209 entries), and in the full list of all possible guesses (5040 entries).

Estimators

The hardest part of the problem is to come up with a good estimator that predicts what the cost will be for any particular guess, at a computational burden lower than examining the entire game tree from that point on. The estimator must provide a strict lower bound on the cost; this is what will be used to prune away branches of the game tree that can not contribute to the optimal strategy.

The simplest estimator derives from the observation that the fan-out at any level of the game tree is limited to ***r***, the number of different possible responses to a guess. This means that at most $(r-1)^{n-1}$ patterns can be identified on the nth guess.

For Bulls and Cows the number of different responses is 14. This means that for the first four levels the maximum number of solved patterns is 1, 13, 169 and 2197, leaving at least 2660 patterns that will require five or more guesses. This gives a lower bound of five for the maximum game length, and an expected game length of at least (1•1 + 13•2 + 169•3 + 2197•4 + 2660•5)/5040 = 22622/5040 = 4.4885

The next improvement comes from looking at the second level of the game tree. For each of the possible response classes to the original guess consider each of the distinguishable second guesses (one for each of the cycle equivalence classes). The maximum theoretical number of responses, ***r***, will only be attained in a small number of cases. There are several factors which contribute to this. One is that the maximum number of responses can only be obtained from a guess which is itself a member of the remaining solution set; any other guess will produce at most (r – 1) different responses (the "All Bulls" response being excluded). But getting even this many responses is unlikely; most guesses will produce far less than the maximum. For any given response class to the initial guess, examining the number of different responses to each potential second guess allows us to calculate the maximum number of patterns that can be solved on the second and third tries combined. This allows calculation of an overall maximum for each initial response class.

There is one further refinement if we are attempting to minimise the expected (rather than maximum) game length. An impossible guess producing a fan-out of ***f*** can solve up to f patterns at the next level. But a possible guess with a fan-out of only f-1 produces a slightly better expected game length for up to f-1 possibilities (the guess itself, plus up to f-2 possibilities at the next level), and matches the expected game length if there are f possibilities remaining. This means we should prefer a possible guess with a fan-out of f-1 to an impossible guess with fan-out f.

In much the same way we can look at all the potential third guesses to determine $F_2(r_1, c_{12}, r_2)$, the maximum fan-out that can be attained at the next level of the tree. Here r_1 and r_2 are the responses to the first and second guesses, while c_{12} is the equivalence class relationship between the first and second guesses. There are a few ways to avoid an exhaustive search to calculate this value; if we can put an upper bound on the fan-out that can be attained, we can terminate the search as soon as that value is achieved. One such limit is that nodes with only a small number of possible solutions, $\boldsymbol{n}$, can obviously not have a fan-out greater than $\boldsymbol{n}$. In particular a node with only two possibilities remaining will obviously achieve this maximum fan-out by trying either of the possible solutions. Another limit comes from the fact that if the maximum fan-out for the initial response class r_1 is $F_1(r_1)$, then no subset of this response class will be able to achieve a fan-out greater than $F_1(r_1)$ (and, similarly, a response r_2 limits the possible fan-out to $F_1(r_2)$).

Note that the equivalence class relationship is symmetric, so $c_{12} \equiv c_{21}$. This, and the fact that the contents of a response class is independent of the order in which guesses were made, shows F_2 is also symmetric: $F_2(r_1, c_{12}, r_2) = F_2(r_2, c_{21}, r_1)$.

So far we have found the maximum possible fan-out after the first three guesses, which is enough to determine the number of patterns that can be solved in at most four guesses. Up to this point all calculations have been exact, but to continue this to the next level would entail a significant amount of calculation. But even without exact calculation we can come up with a reasonably good estimate of the possible fan-out at the next level. Once again we start with the list of potential third-level guesses that was used in the determination of $F_2(r_1, c_{12}, r_2)$. Rather than simply looking at the fan-out generated by each guess, though, we now examine each of the response classes, and calculate an upper bound on the fan-out that is possible within that response class. The size of each response class gives one of the limits. The fan-out can also not exceed $F_2(r_1, c_{12}, r_2)$, and symmetry considerations show that the fan-out is similarly bounded by $F_2(r_1, c_{13}, r_3)$ and $F_2(r_2, c_{23}, r_3)$. Satisfying all these constraints automatically satisfies those from $F_1(r_1)$, $F_1(r_2)$, and $F_1(r_3)$.

We can calculate an upper bound on the next level fan-out for any guess by combining the fan-out limits for each of the response classes. Taking the maximum of this over all the third-level guesses gives an upper bound on the fan-out that could be attained at the fourth level. Combining this with the results of the earlier calculations is just what is needed to compute an estimator function $\boldsymbol{E}(r_1, c_{12}, r_2)$.

Performing these calculations for the case of minimising the expected game length for Bulls and Cows produces the following table[3]:

		1	**2**	**3**	**4**	**5**	**6**	
BBBB	1	1						1
BBCC	6		1	2	2	1		21
BCCC	8		1	2	3	2		30
CCCC	9		1	3	4	1		32
BBB	24		1	5	10	8		97
BBC	72		1	8	32	29	2	311
BCC	216		1	10	58	141	6	1005
CCC	264		1	10	68	178	7	1236
BB	180		1	10	51	118		826
BC	720		1	13	102	459	145	3614
CC	1260		1	13	110	604	532	6693
B	480		1	12	83	308	76	2366
C	1440		1	13	106	594	726	7791
NONE	360		1	10	65	214	70	1782
	5040	1	13	111	694	2657	1564	25805

The entries in the columns headed **1 – 4** are the (exact) upper limits on the possible number of solutions achievable at that level. Column **5** is the approximate bound calculated as described in the previous section, and column **6** shows the number of solutions remaining after allowing for those accounted for in the previous columns. The final column shows the total number of guesses taken finding these solutions.

From this we see that the expected game length must be at least 5.12 (25805/5040). As this is a strict lower bound this means that the minimum game length must be at least six for some solutions, which places a lower bound on the maximum game length required by any strategy. Since some heuristic strategies can be shown to require no more than seven moves this reduces the problem of finding the maximum game length to the one of determining whether there is a strategy which will never take more than six moves. In fact it is a relatively simple task[4] to show there is no such strategy, so a limit of seven moves is the best that can be achieved.

[3] For a more detailed description of the calculations see Appendix A

[4] One method would be by performing an exhaustive search, always looking at the largest response class first.

Heuristic Strategies

The 1971 article in *Software* mentions several heuristic strategies that performed as well as all but the best human players. The most impressive of these (achieving a mean game length that only differs from that of an optimal strategy by roughly one part in two hundred) is the one described below that was used by Dr. J. Larmouth, but three additional strategies evaluated by B. Landy are also mentioned.

The July 2001 issue of the International Computer Games Association Journal was mostly given over to articles on Ken Thompson's contributions to computer chess. One of the articles in that issue, written by D.M. Ritchie[7], mentioned in passing a very early Unix program that Ken Thompson had created to investigate a strategy for playing MOO. The brief description in that article states the strategy chose a next guess that would provide the maximum number of different responses.

A more detailed description of many of these and of several other strategies, together with the mean and maximum game lengths that each strategy achieves, can be found in a paper written by Andy Pepperdine[8]

Dr. Larmouth's Strategy

This, like many of the heuristic strategies, picks the next guess by looking at some function of the sizes of the different response classes generated by each guess, and choosing the guess that gives the minimum (or maximum) value of the function. The function to be minimised in this case, based on information theory, is

$$\sum_{|t_r| \neq 0} |t_r| \bullet \log(|t_r|) \; - \; 2 \bullet \log(2) \bullet |t_{BBBB}|$$

where $|t_r|$ is the size of response class r, and the 2•log(2) term is a correction for the case when the guess being evaluated is one of the possible solutions remaining.

It is unclear what method was used as a tiebreaker when multiple guesses produced the same value. The simplest choice (just pick the first one) can be shown to have an expected game length of 5.24 (26408/5040)[5], and to solve all but one pattern using at most seven guesses. It is very hard to improve on this – even a perfect choice of tiebreaker ("God's algorithm") could achieve no better than a total cost of 26406 guesses (which can be done with no pattern requiring more than seven).

[5] In the original 1971 paper this is stated to have required 'a large amount of computation' on one of the fastest supercomputers of the day; some forty years on the calculations take less than a second on my notebook computer!

An Optimal Strategy

The fundamental problem to be solved in finding an optimal strategy for playing Bulls and Cows can be described as follows:

> *Given a set of potential solutions, and a list of guesses to be considered, find the guess from that list which yields the "best" result (in some sense) when evaluated over the set of potential solutions.*

The usual definition of "best" (or optimal) is one that minimises the expected game length, although other game variations discussed later use different definitions.

The simplest way to solve this problem is by using a divide-and-conquer approach. Applying each guess to the list of potential solutions divides the solution space into disjoint subsets – one for each possible response. Combining optimal solutions to each of the sub-problems then provides an optimal solution for a particular guess, which in turn leads to an optimal solution to the original problem.

There are several refinements that are needed in order to turn this description of the algorithm into a practical way of solving the problem. The simplest of these is one to guarantee that the algorithm will eventually terminate. While this can be done by the common-sense approach of not repeating a guess that has already been tried, a slightly better technique is to reject any guess that does not split the solution list into two or more parts.

A practical algorithm must not only terminate – it must do so in a reasonable amount of time. An exhaustive search of the game tree to a depth of six or seven levels, considering each of the 5000 or so potential guesses for each response class, is computationally infeasible; some way must be found to reduce the search space. One good way to do this at the early levels of the game tree is to use the methods discussed in the section on equivalences. All guesses at level one are effectively equivalent, and there are only twenty distinguishable guesses at level two, so these two levels alone provide better than a million-fold speed improvement. Continuing these techniques to level three still provides a significant benefit, but beyond that point the diminishing returns, coupled with the additional overhead, make keeping track of the equivalences from column permutations less attractive, although it is still worth taking into consideration those derived from simply renaming symbols.

In addition to looking for techniques to speed up searches by avoiding duplication of effort, we can also look for ways to eliminate the need to investigate some parts of the search tree. For this we can use the estimators that were introduced earlier, together with some limit on the number of guesses allowed.

When one of the list of candidate guesses at level n is applied to the set of potential solutions it splits the set into several subsets. The estimator functions $E(r_i, c_{ij}, r_j)$ can be used to compute a lower bound on the number of guesses needed to solve any of these subsets; this number must be greater than or equal to the largest of the estimates for all i, j with $1 \leq i < j \leq n$. Taking the sum of the bounds for each of the subsets enables us to calculate a lower bound on the number of guesses needed from this point on after trying that particular candidate.

Computing this maximum is an $O(n^2)$ function of the level. A good approximation is to take the maximum from the estimator function used at the previous level and just those estimators $E(r_i, c_{in}, r_n)$, and to remember which estimator was used. This is only an O(n) calculation, but is likely to come very close to the exact result. In fact for levels up to level three it will be exact, and for higher levels the number of solutions is small enough that differences in estimators rarely affect the result.

There are several ways to use this estimate. The most straightforward is to compare it to the limit on the number of guesses; if it is larger then this candidate guess can be eliminated immediately, and we can proceed to the next guess from the list.

An alternative approach is to first calculate this estimate for each of the candidate guesses before examining any guess in detail, and to sort the candidate list so that guesses with a lower estimate are processed earlier. While this does mean that all the estimates must be calculated before completely analysing any guess, the cost of doing so is generally more than recouped by not spending any time investigating guesses with an estimated cost greater than that achieved by the optimal solution.

There is another technique that can be employed based upon the cost estimate that was made at the previous level. If this is less than or equal to twice the number of patterns in the set of potential solutions (so the expected game length from this point on is less than or equal to two) then the members of the set of solutions should be tried as guesses first. This will be a much shorter list than the full list of guesses – often by a factor of 100 (or even more) and so can be tested much faster. Not only that – if we find a solution with an expected game length of at most two there is no need to look at the full list of guesses, as no improvement is possible.

The next step (unless the candidate guess is eliminated based on the estimated cost) is to proceed to the investigation of each of the subsets of the solution space. There are a couple of minor preparatory tasks that should be performed at this time; one is to identify the list of candidate guesses to be used at the next level (either one of the special-case lists for the second and third guess, or the general case list based on the number of different symbols used in all the earlier guesses). Another task is to calculate the difference between the total estimated cost of this guess and the limit on the cost allowed (henceforth referred to as the *slack* in the cost constraint).

The subsets are now processed in turn. The order in which this processing is done can have a noticeable effect on the run time of the algorithm. Generally speaking smaller subsets require less time than larger ones, while searches with less slack run faster (sometimes much faster) than those with more slack. This suggests that a good approach might be to process the subsets in increasing order of size.

A subset of size one requires no further processing – the only remaining solution is already identified. Larger subsets can be examined recursively, with the limit on the allowed cost set to the sum of the slack and the estimated cost for this subset.

Before processing a subset, though, we need to check for a maximum search depth. The estimator function gives us a limit on the number of patterns that can be solved in the next three moves, which gives a lower limit on the achievable game length. If this is too large then we can give up on this subset without further processing.

If no solution for a subset can be found there is no solution possible starting from this candidate guess, and so we can immediately move on to the next candidate. Otherwise the difference between the cost of the solution and the estimated cost for the subset is calculated, and subtracted from the current value for the slack. (It is this reduction in slack that speeds up the analysis of later subsets, which is why it is generally advisable to delay processing of larger subsets for as long as possible).

If there are no more subsets left to examine then we have found a solution to the current bounded search problem, although not necessarily an optimal solution. The limit on the number of guesses allowed can now be reduced to one less than the cost of the solution just found (which will in turn reduce the slack for all future investigations), and processing can continue with the next candidate guess.

Eventually all guesses will have been either processed or rejected. If any solutions were found then the most recent one is an optimal solution; if not, then no solution is possible that satisfies the initial conditions.

Results

The search for an optimal “Bulls and Cows” strategy produces a decision tree that uses a total of 26274 moves to solve all 5040 patterns, for an average game length of 5.213 (a value independently confirmed by the results that Tanaka found), and with no pattern requiring more than seven guesses. This is doubly optimal; it simultaneously achieves the optimum value for both the mean and the maximum game length. The solution is by no means unique; there are many such trees that have the same average and maximum game length, often with different number of patterns solved in a given number of guesses. The following table shows examples:

Strategy Name	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Total**	**Nodes**	**Probes**
Fast Strategy	1	7	62	718	2403	1757	92		26274	1562	86
Slow Strategy	1	7	61	692	2445	1755	79		26274	1524	128
Tanaka’s Result	1	7	63	697	2424	1774	74		26274	—	—
Larmouth - One	1	7	70	642	2410	1806	103	1	26408	1651	223
Larmouth - All	1	7	69	651	2396	1813	102		26406	1657	217
Possibles Only	1	13	109	637	2125	1924	231		26688	1845	0
Head-to-Head	1	4	47	688	2531	1628	141		26312	—	—

For each strategy this shows the number of patterns solved in 1, 2, ... 8 guesses, and also the total number of guesses required to identify all 5040 possible patterns. The final two columns show the number of decision points in the strategy tree (nodes where more information is needed to choose between multiple patterns), and the number of such nodes where the guess to be tried is not a possible solution.

The first three strategies in the table are all optimal; the primary solution found by my program, another one found (as a check) with the shortcut to preferentially guess possible solutions disabled, and the strategy described by Tanaka in 1996.

The table also contains details for some non-optimal strategies; one that uses the heuristic algorithm devised by Dr. Larmouth, a variation on that strategy that picks the best amongst all equal predictions, and a strategy constrained to only make guesses which are themselves possible solutions.

The final row shows values for Tanaka’s strategy for playing head-to-head games. Further discussion of this strategy will be provided in a later version of this paper.

Performance

The complete analysis of Bulls and Cows, including all pre-processing, takes just under 45 minutes when run on my notebook computer (a 2GHz Intel Core 2 Duo). This compares quite favourably with Tanaka's 1996 results of around 90 hours on a SUN SparcStation 20 (a machine with 1/60 of the performance of my notebook). Much of the difference comes from using a lookup table to find the response code of two patterns, rather than calculating it every time - a technique considered by Tanaka, but rejected because of the excessive (for the time) memory demands of a table with over 25 million entries.

When compared to the performance of my 1978 code on the DECSYSTEM-2050 (which had perhaps 1/3000 of the CPU integer performance of my notebook PC) that 45 minutes doesn't look quite so good – that would scale to around three months of 1978 CPU time, while to the best of my recollection that first analysis only took 75% of that, despite running a somewhat less sophisticated version of the software. I can think of several factors that might contribute to this. One is that while CPU speed has improved by a factor of 3000, memory access speed hasn't done quite as well (achieving perhaps only a factor of 1000) and the large memory footprint and mostly linear memory access patterns don't cache very well. Another factor is that the 1978 program was hand-optimised assembly code (including using platform-specific tricks like executing inner loops out of the fast registers). Another possibility is that my CPU speed calculations are incorrect; another set of figures suggests that I may be overstating speed gains by a factor of two or more. And finally there's always the possibility that my memory is at fault, and that the two+ months that I remember was just the final chunk that completed the analysis.

Statistics

In the course of the strategy search some 6 million positions in the game tree were investigated. At each of these positions an average of around 500 guesses were applied to the potential solutions in a response class (with an average size of 10), which works out to approximately 30 billion response calculations. This takes roughly 1/3 of the total CPU time of 2700 seconds, or around 30 nsec to determine each response, store it in the solution table, and increment the appropriate counter. The other 2/3 of the CPU time is almost entirely spent in selecting estimator functions, calculating cost estimate values, and sorting lists of patterns or guesses.

Variations

A strategy than yields the minimum expected game length is appropriate when one is playing either against a computer, or head-to-head against a human opponent when the number of points scored in any round varies based on the difference in the number of guesses taken (this is the scoring method in used in *Mastermind*). As Tanaka observed in his 1996 paper a different strategy is required when a win only scores a single point no matter how many (or how few) moves are taken.

All discussions to this point have assumed that play is against an honest opponent (the *codemaker*, to use the *Mastermind* terminology) who picks a pattern at random at the beginning of the game. Dennis Ritchie, in an email follow-up[9] to a posting I had made on the Usenet news group alt.folklore.computers, proposed a couple of other ways that a codemaker could act. One suggestion was that instead of picking a secret pattern purely at random the codemaker could note any regularities in the play of his opponent (such as always guessing “0123” initially, with predictable second guesses after each response), and use that information to choose patterns that would take more than the average number of moves to identify. An easy way to counter this approach is to permute the order of both the places and the symbols.

Another suggestion was that instead of choosing an initial pattern the codemaker simply gives replies that, while consistent, make the game last as long as possible. This is maximising what Conway[10] refers to as the *Remoteness* of a game position (a concept originally described by Steinhaus[11]). The way to counter this is to use a strategy that is guaranteed to identify any pattern using at most ***n*** moves. There is then nothing that the codemaker can do to force any game to last longer than ***n***, and it may be possible for an opponent to force the game to finish earlier.

Yet another variation is the problem of how many guesses are required to identify the secret pattern if all guesses are to be made before any information is revealed.

Conclusions

Full computer analysis of the game of Bulls and Cows was an intractable problem when I first encountered the game in 1968. With the vast increases in computer power since then it is now possible to determine an optimal strategy in less than an hour, but only by taking advantage of several optimisation techniques – exhaustive examination of the entire game tree is still beyond the reach of a naive approach.

I believe that the game presents an interesting problem for those interested in the study of recreational mathematics and computing – small enough to be solvable, but large enough to be challenging.

References

[1] $\aleph_0$, Computer Recreations, Software – Practice & Experience 1.2, 201-204 (Apr-Jun 1971)

[2] J.M. Grochow, MOO in Multics, Software – Practice & Experience Vol. 2, 303-304 (1972)

[3] D.E. Knuth, The Computer as Master Mind, J. Recreational Mathematics Vol. 9 (1976/7)

[4] R.W. Irving, Towards an Optimal Mastermind Strategy, J. Recreational Mathematics (1978)

[5] K. Koyama & T. Lai, An Optimal Mastermind Strategy, J. Recreational Mathematics (1993)

[6] Tetsuro Tanaka, An Optimal MOO Strategy, Game Programming Workshop - Japan (1996)

[7] D.M. Ritchie, "Ken, Unix, and Games", ICGA Journal 24 No. 2 (June 2001)

[8] A. Pepperdine, The Game of MOO, http://www.pepsplace.org.uk/Trivia/Trivia.html (2007)

[9] D.M. Ritchie, private communication (July 2001)

[10] J.H. Conway, On Numbers and Games, L.M.S. Monograph 6, Academic Press (1976)

[11] H. Steinhaus, Definicje potrzebne do teorji gry i pościgu, Myśl. Akad. Lwów 1(1), (1925)
Translated as Definitions for a theory of games and pursuit, Naval Research Logistics 7 (1960)

[12]

Appendix A – A Worked Example of Estimator Calculations

The following table shows how a game of Bulls and Cows can continue after an initial response to 0123 of BBB The number in each non-empty cell shows how many of the 24 remaining possibilities give a particular response to the potential second guess shown at the head of the column.

	Second guess after an initial response to 0123 of BBB (24 remaining possibilities)																		
	0132	0231	1032	1230	0124	0134	0214	0234	1034	1204	1234	0145	0245	1045	1245	1435	0456	1456	4567
BBBB					1														
BBCC						1	1												
BCCC								1		1									
CCCC									1		1								
BBB					5	1						2							
BBC	12				3	5		1				2	1						
BCC	12	18				2	7	6	1		1		3	2	1				
CCC		6	24	24			1	1	7	8	7			2	3	4			
BB					15	5						8	1				3		
BC						10	10	10				4	9		1	2	6	2	
CC							5	5	15	15	15		2	12	11	10		7	
B												8	4				9	1	4
C													4	8	8	8	3	11	12
NONE																	3	3	8
	2	2	1	1	1+3	6	5	6	4	3	4	5	7	4	5	4	5	5	3

There are several interesting things to note in this table. One is that there are many empty cells (or, alternatively, that many guesses generate less than 14 responses). While this is partly due to the fact that there are only a small number of possible solutions in this equivalence class (24, in this case), other equivalence classes show similar results. Another noteworthy point is that some of the potential guesses (specifically 1032 and 1230) provide no additional information, as they produce no further subdivision of the solution set into two or more different response classes.

But the most significant part of this table is the final row, which shows how many different responses are generated by each of the distinguishable second guesses. Of particular interest is the fact that the maximum fan-out of 7 comes from a guess which is not itself in the solution set. This means that it is possible to correctly identify more patterns on the third guess by trying an impossible second guess