Alexey Slovesnov

email slovesnov@yandex.ru

site http://bulls-cows.sourceforge.net

Search of optimal algorithms for bulls and cows game.

Abstract.

The article concerns two optimization ways of bulls and cows game. The first one is minimizing amount of numbers which computer can guess for exactly seven turns, all others should be guessed for up to six turns. The second way is minimizing average amount of turns for guess arbitrary secret number (minimizing average game length). During of search several intermediate algorithms were created. For some of them trees are built and created web application for online playing between computer and man.

At the moment author searches for servers to finish calculations and finding the exact estimation. If you want to help the project please write to the author.

1st of November 2011

second edition

address of last edition of article
http://bulls-cows.sourceforge.net/bullscows.pdf

Contents

Introduction.

Optimizing of bulls-cows game traditionally has two directions.

- First direction. Minimizing of average amount of turns to guess arbitrary secret number. Since this problem is solved (see [1] and [2]), average game length is 26274/5040=5.2131, then it's sufficient to find algorithm with such average game length.
- Second direction. It's known that there is no algorithm which can guess each secret number using six or less turns. At the same time there are algorithms which can guess each number using up to seven moves. So the second directions is to minimize amount of numbers which algorithm can guess using exactly seven turns.

Searching of minimal average game length or minimal amount of numbers which computer can guess using exactly seven moves are very difficult problems. Let's try roughly estimate number of operations for exhaustive search. It's obviously that we can use number 0123 as first turn without loss of generality. Moves from second to sixth can be on of 5039 possibilities each (number 0123 not used). After every move we can get up to 14 responses, for one of them (four bulls and zero cows) we don't need further computation. Let suppose that we can reduce average number of turns in four times, and we can reduce average number of responses in four times as well. So we have estimation for amount of nodes for exhaustive search $5039^5 \times (\frac{13}{16})^6 \approx 9.3 \times 10^{17}$. This estimation looks like huge even for modern computers.

The article consists of several parts. In first part we introduce some terms and give mathematical theory. The second part describes the algorithms. The third one describes base nodes of search algorithms. The fourth one describes components of project. In fifth part results of algorithms for minimizing amount of numbers, which can be guessed for seven turns, are showed. In sixth part algorithm will be extended for minimizing not only amount of numbers which computer can guess for exactly seven turns, but for minimizing amount of numbers which computer can guess using exactly six or fewer turns. Seventh part has results of minimizing average game length. Eighth part describes of trees building. They will be use for web application where computer can guess numbers. Ninth part concerns of calculations times.

1 Theory.

1.1 Notation.

Response on some number denote as *„amount of bulls.amount of cows“*. For example, 2.1 means that we got answer two bulls and one cow.

Secret number t with response r name as *turn* or *move* and note as (t, r). Sometimes *turn* will be written without brackets and comma - 5678 0.1.

Set of all secret numbers note as $\Omega = (0123, 0124 \dots 9876)$.

Set of all possible answers note as $R = (0.0, 0.1 \dots 4.0)$.

Sequence of turns note as $(t_1, r_1) \dots (t_n, r_n)$.

Numbers which can be guessed for exactly k attempts are called as „$k-th$“.

Minimal amount of turns for all algorithms, which can be guessed using exactly k attempts after sequence $(t_1, r_1) \dots (t_n, r_n)$ note as $\Phi_k[(t_1, r_1) \dots (t_n, r_n)]$.

Minimal average game length for all algorithms, after sequence of turns $(t_1, r_1) \dots (t_n, r_n)$ note as $\Delta[(t_1, r_1) \dots (t_n, r_n)]$. Suppose that after sequence of turns $(t_1, r_1) \dots (t_n, r_n)$ left m secret numbers. To work with integer values we'll use analogous function, which equals sum of number of turns to guess recent secret numbers $\Lambda[(t_1, r_1) \dots (t_n, r_n)] = m \times \Delta[(t_1, r_1) \dots (t_n, r_n)]$.

Consider sequence of turns $(t_1, r_1) \dots (t_n, r_n)$ and number p. Sum of amount of turns which can be guessed using exactly k tries, after turns $(t_1, r_1) \dots (t_n, r_n)$ and $(n+1)$-th turn p with all possible responses note as $\Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, *)]$.

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, *)] = \sum_{r \in R} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, r)] \tag{1}$$

Analogously,

$$\Lambda[(t_1, r_1) \dots (t_n, r_n)(p, *)] = \sum_{r \in R} \Lambda[(t_1, r_1) \dots (t_n, r_n)(p, r)] \tag{2}$$

Numbers consist of four digits. Every digit has its own place from 1 to 4. First digit has place 1, second one 2, etc. Every digit can be from 0 to 9. Denote set of all transformations of digits from 0 to 9 and places from 1 to 4 as Ψ. Items of set Ψ denote as $\psi \in \Psi$.

The result of transformation ψ on number t denote as $\psi(t)$.

The result of transformation ψ on set of numbers $T = \{t_1 \dots t_n\}$ denote as set of numbers $\{\psi(t_1) \dots \psi(t_n)\}$. $\psi(T) = \{\psi(t_1) \dots \psi(t_n)\}$.

Denote cardinality of set S as $|S|$. For example, $|\Psi| = 10! \times 4! = 87\,091\,200$.

Empty set - $\varnothing$.

The goal of article is to find $\Phi_7[\varnothing]$ and $\Lambda[\varnothing]$.

1.2 Transformations.

It's possible to do any redesignation of digits and positions, which in fact does not change anything, so the following is true:

Lemma 1 *Let we have sequence of turns $(t_1, r_1) \dots (t_n, r_n)$, and some transformation ψ, then*

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)] = \Phi_k[(\psi(t_1), r_1) \dots (\psi(t_n), r_n)]$$

$$\Lambda[(t_1, r_1) \dots (t_n, r_n)] = \Lambda(\psi(t_1), r_1) \dots (\psi(t_n), r_n)]$$

Lemma 2 *For every two numbers p_1 and p_2 there is exists transformation ψ such as $\psi(p_1) = p_2$ and $\psi(p_2) = p_1$.*

Proof. By writing computer program which looks over all possible numbers $p_1 \in \Omega$ and $p_2 \in \Omega$ and all transformations $\psi \in \Psi$, we can make sure that it's true.

Example. Consider two numbers 0123 and 4051. Find transformation ψ, such that $\psi(0123) = 4051, \psi(4051) = 0123$.

Solution. Consider transformation ψ_1, which swaps the digits $0 \leftrightarrow 1$, $2 \leftrightarrow 5, 3 \leftrightarrow 4$. Number 0123 goes to number $\psi_1(0123) = 1054$, and number 4051 goes to $\psi_1(4051) = 3120$. Now consider place transformation ψ_2 $1 \leftrightarrow 4$, digit from first place goes to fourth place and vice versa. By applying transformation ψ_2 we got $\psi_2(\psi_1(0123)) = \psi_2(1054) = 4051$. Similar $\psi_2(\psi_1(4051)) = \psi_2(3120) = 0123$. Thus the composition $\psi = \psi_2 \circ \psi_1$ yields the desired result.

Lemma 3 *For every two numbers p_1 and p_2 and every two responses r_1 and r_2*

$$\Phi_k[(p_1, r_1)(p_2, r_2)] = \Phi_k[(p_1, r_2)(p_2, r_1)]$$

$$\Phi_k[(p_1, r_1)(p_2, *)] = \Phi_k[(p_2, r_1)(p_1, *)]$$

Proof. From lemma 2 we know that, exists transformation ψ, such that $\psi(p_1) = p_2, \psi(p_2) = p_1$. Thus, using lemma 1, we have:

$$\Phi_k[(p_1, r_1)(p_2, r_2)] = \Phi_k[(\psi(p_1), r_1)(\psi(p_2), r_2)] = \Phi_k[(p_2, r_1)(p_1, r_2)]$$

Since the order of moves is not important, it's possible to interchange moves 1 and 2, so

$$\Phi_k[(p_2, r_1)(p_1, r_2)] = \Phi_k[(p_1, r_2)(p_2, r_1)]$$

So first was proved. Similarly,

$$\Phi_k[(p_1, r_1)(p_2, *)] = \sum_r \Phi_k[(p_1, r_1)(p_2, r)] = \sum_r \Phi_k[(p_2, r_1)(p_1, r)] = \Phi_k[(p_2, r_1)(p_1, *)]$$

Assertion was proved.

Note. The same is true for Λ function.

Lemma 4 *For every number p and every response r.*

$$\Phi_k[(0123, r)] = \Phi_k[(p, r)]$$

$$\Phi_k[(0123, *)] = \Phi_k[(p, *)]$$

Proof. Consider number 0123 and some number p. Then from lemma 2 we got that there is transformation ψ, such that $\psi(0123) = p$. If we take for lemma 1 sequence with one move $(0123, r)$, with some response r, then $\Phi_k[(0123, r)] = \Phi_k[(\psi(0123), r)] = \Phi_k[(p, r)]$. As well it's obvious that $\Phi_k[(0123, *)] = \Phi_k[(p, *)]$ because

$$\Phi_k[(0123, *)] = \sum_r \Phi_k[(0123, r)] = \sum_r \Phi_k[(\psi(0123), r)] = \sum_r \Phi_k[(p, r)] = \Phi_k[(p, *)]$$

Assertion was proved.

Note. The same is true for Λ function.

1.3 The equivalence classes.

We call two numbers p and q are equivalent relative to sequence of turns $(t_1, r_1)\dots(t_n, r_n)$ $(p \sim q)$ if $\exists\psi : \psi(p) = q$ and $\psi(t_i) = t_i, 1 \le i \le n$.

Lemma 5 *The relation $p \sim q$ is equivalence relation.*

Proof.

1. Lets prove that $a \sim a$ (reflexivity).

   It's obvious. It suffices to take the identity transformation.

2. If $a \sim b$, then $b \sim a$ (symmetry).

   If $a \sim b$, then $\exists\psi : \psi(a) = b, \psi(t_i) = t_i$. Every transformation has inverse transformation $\psi^{-1} : \psi^{-1}(b) = a, \psi^{-1}(t_i) = t_i$. Thus the symmetry is proved.

3. If $a \sim b$ and $b \sim c$, then $a \sim c$ (transitivity).

   If $a \sim b$, then $\exists\psi_1 : \psi_1(a) = b, \psi_1(t_i) = t_i$. And if $b \sim c$, then $\exists\psi_2 : \psi_2(b) = c, \psi_2(t_i) = t_i$. It's obvious that $\psi_2 \circ \psi_1(a) = \psi_2(b) = c$ and $\psi_2 \circ \psi_1(t_i) = \psi_2(t_i) = t_i$. Thus the transitivity is proved.

Lemma 6 *If two numbers p and q belongs one equivalence class, for sequence $(t_1, r_1)\cdots(t_n, r_n)$ then $\forall k$, $\forall r$*

$$\Phi_k[(t_1, r_1)\cdots(t_n, r_n)(p, r)] = \Phi_k[(t_1, r_1)\cdots(t_n, r_n)(q, r)]$$

$$\Phi_k[(t_1, r_1)\cdots(t_n, r_n)(p, *)] = \Phi_k[(t_1, r_1)\cdots(t_n, r_n)(q, *)]$$

Proof. If $p \sim q$, then $\exists\psi : \psi(p) = q, \psi(t_i) = t_i$. Using lemma 1, we got

$$\Phi_k[(t_1, r_1)\cdots(t_n, r_n)(p, r)] = \Phi_k[(\psi(t_1), r_1)\cdots(\psi(t_n), r_n)(\psi(p), r)] =$$

$$= \Phi_k[(t_1, r_1)\cdots(t_n, r_n)(q, r)]$$

Analogously,

$$\Phi_k[(t_1, r_1)\cdots(t_n, r_n)(p, *)] = \sum_r \Phi_k[(t_1, r_1)\cdots(t_n, r_n)(p, r)] =$$

$$= \sum_r \Phi_k[(\psi(t_1), r_1)\cdots(\psi(t_n), r_n)(\psi(p), r)] = \sum_r \Phi_k[(t_1, r_1)\cdots(t_n, r_n)(q, r)] =$$

$$= \Phi_k[(t_1, r_1)\cdots(t_n, r_n)(q, *)]$$

Assertion was proved.

Note. The same is true for Λ function.

1.4 Set of first moves.

Lemma 7 *For first move is enough to consider only the number 0123, the rest can be discarded.*

$$\Phi_k[\varnothing] = \Phi_k[(0123, *)]$$

Proof. To find $\Phi_k[\varnothing]$ we need to look over all possible numbers from Ω as first turn and take minimum estimation

$$\Phi_k[\varnothing] = \min_{p\in\Omega} \Phi_k[(p, *)]$$

From lemma 4 follow that for every number p: $\Phi_k[(p, *)] = \Phi_k[(0123, *)]$, so

$$\Phi_k[\varnothing] = \min_{p\in\Omega} \Phi_k[(p, *)] = \Phi_k[(0123, *)]$$

Assertion was proved.

Note. The same is true for Λ function.

1.5 Set of second moves.

Lemma 8 *If first turn is 0123, then it is sufficient to use only numbers from set S_2 as second turn.*

$$S_2 = (0124, 0132, 0134, 0145, 0214, 0231, 0234, 0245, 0456,$$
$$1032, 1034, 1045, 1204, 1230, 1234, 1245, 1435, 1456, 4567)$$

or

$$\Phi_k[(0123, r)] = \min_{p\in S_2} \Phi_k[(0123, r)(p, *)]$$

Proof. Consider two numbers p and q. If exists transformation ψ such that, $\psi(0123) = 0123, \psi(p) = q$, then

$$\Phi_k[(0123, r)(p, *)] = \Phi_k[(\psi(0123), r)(\psi(p), *)] = \Phi_k[(0123, r)(q, *)]$$

Similarly, if exists transformation ϕ such that, $\phi(0123) = q$, $\phi(p) = 0123$, then

$$\Phi_k[(0123, r)(p, *)] = \Phi_k[(\phi(0123), r)(\phi(p), *)] =$$

$$= \Phi_k[(q, r)(0123, *)] = \sum_{e\in R} \Phi_k[(q, r)(0123, e)]$$

Using lemma 3, we can see that

$$\sum_{e\in R} \Phi_k[(q, r)(0123, e)] = \sum_{e\in R} \Phi_k[(0123, r)(q, e)] = \Phi_k[(0123, r)(q, *)]$$

That is from two possible second turns p and q, it's sufficient consider only number p if we can find transposition ψ or ϕ.

We realize, with a computer program, the following algorithm.

Lets initialize the empty set S. After that we do a cycle for all numbers $p \in \Omega$ and goes over all moves $q \in S$. If doesn't exist transformation ψ such as $\psi(0123) = 0123, \psi(p) = q$ or $\psi(0123) = q, \psi(p) = 0123$, then insert number p into set S. At the end remove number 0123 from set S because this turn has already been made. As the result we got set $S_2 = S \backslash (0123)$.

Assertion was proved.

Note. The same is true for Λ function.

Now instead of brute force computation for second turn, when we look over 5040 possibilities, it is enough to examine only $|S_2| = 19$ moves.

1.6 Set of third moves.

After a similar reasoning for the third move, we get the following assertion:

Lemma 9 *Consider every number $s \in S_2$, and two numbers p and q, then if exists ψ, such that*

$$\psi(0123) = 0123,\ \psi(s) = s,\ \psi(p) = q,$$

then

$$\Phi_k[(0123, r_1)(s, r_2)(p, r_3)] = \Phi_k[(0123, r_1)(s, r_2)(q, r_3)]$$

Similarly with building set of second turns for every number s from set S_2 it is possible to construct set of third turns $S_3(s)$. We remove number s itself from set $S_3(s)$ at the end of calculations.

All sets $S_3(s)$ have 7072 items, it means that we got about 7072/19=372.2 numbers for every item of set S_2. Now we need to look over only for 7072 possibilities (for turns from one to three), instead of $5039^2 = 2.5 \times 10^7$ possibilities.

Note. The same is true for Λ function.

1.7 Subsets estimation.

Lemma 10 *$\forall R' \subset R$ and every sequence of turns $(t_1, r_1) \dots (t_n, r_n)$, and number t next assertion is true*

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, *)] \geq \sum_{r \in R'} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)]$$

in particular

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, *)] \geq \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)]\ \ \forall r \in R$$

Proof. From formula 1 we got that

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, *)] = \sum_{r \in R} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)] =$$

$$= \sum_{r \in R'} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)] + \sum_{r \notin R'} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)] \geq$$

$$\geq \sum_{r \in R'} \Phi_k[(t_1, r_1) \dots (t_n, r_n)(t, r)]$$

Assertion was proved.

Note. The same is true for Λ function.

1.8 Minimal estimation.

Minimizing amount of 7th. If we find move which has estimation equals 0 then it is best estimation and we don't need to do further search.

Minimizing average game length. Let cardinality of set of recent numbers is $|S|$ and we do k-th turn. If some move splits set of recent numbers to subsets with at most one item then it's best turn (all others can be pruned) and sum of turns equals $\Lambda = k + (|S| - 1)(k + 1) = |S|(k + 1) - 1$.

1.9 Fast estimation.

Minimizing amount of 7th. Let the sequence of turns $(t_1, r_1) \dots (t_n, r_n)$ have been made and we count 7th. Denote $S = (s_1, s_2 \dots)$ as set of numbers, which satisfies all of the turns from the sequence. In some cases when set S consists of low amount of items, we can immediately obtain the estimation.

If we do the seventh turn then

$$\Phi_7[(t_1, r_1) \dots (t_n, r_n)] = \begin{cases} 1 & \text{if } |S| = 1 \\ 5040 & \text{if } |S| > 1 \end{cases}$$

If we do the sixth turn then

$$\Phi_7[(t_1, r_1) \dots (t_n, r_n)] = \begin{cases} & \text{if } |S| = 1 \text{ or } |S| = 2 \text{ or} \\ |S| - 1 & |S| = 3 \text{ and for one turn } s_i \text{ two} \\ & \text{others give different responses.} \end{cases}$$

If we do turn from first to fifth

$$\Phi_7[(t_1, r_1) \dots (t_n, r_n)] = \begin{cases} & \text{if } |S| = 1 \text{ or } |S| = 2 \text{ or} \\ 0 & |S| = 3 \text{ and for one turn } s_i \text{ two} \\ & \text{others give different responses.} \end{cases}$$

Later we will count not only 7th but 6th, 5th etc. So the above formula can be extended.

If we do k-th turn.

$$\Phi_k[(t_1, r_1) \dots (t_n, r_n)] = \begin{cases} 1 & \text{if } |S| = 1 \\ 5040 & \text{if } |S| > 1 \end{cases}$$

If we do $(k-1)$-th turn.

$$\Phi_k[(t_1,r_1)\dots(t_n,r_n)] = \begin{cases} & \text{if } |S|=1 \text{ or } |S|=2 \text{ or} \\ |S|-1 & |S|=3 \text{ and for one turn } s_i \text{ two} \\ & \text{others give different responses.} \end{cases}$$

If we do turn from first to $(k-2)$-th.

$$\Phi_k[(t_1,r_1)\dots(t_n,r_n)] = \begin{cases} & \text{if } |S|=1 \text{ or } |S|=2 \text{ or} \\ 0 & |S|=3 \text{ and for one turn } s_i \text{ two} \\ & \text{others give different responses.} \end{cases}$$

Minimizing average game length. If we do k-th turn and $|S|=1$ or $|S|=2$ or $|S|=3$ and for one turn s_i two others give different responses, then

$$\Lambda[(t_1,r_1)\dots(t_n,r_n)] = k + (|S|-1)(k+1) = |S|(k+1)-1$$

1.10 Cutoffs for turns 4-6.

For turns 1-3 we built the sets which can strictly accelerate the algorithm. Analogously we can build sets for turns 4-6, but unfortunately they need a lot of operating memory. Now we describe how to prune some of the numbers for turns from four to six.

Note. In practice all of next prunes were used only for solving sequence (0123,0.1)(1245,0.0) by the exact algorithm.

1.10.1 Absent digits.

Let we do some sequence of turns $(t_1,r_1)\dots(t_n,r_n)$. Denote $S=(s_1,s_2\dots)$ set of numbers which satisfy all of the turns from sequence. Assume that all numbers $s_1,s_2\dots$ do not contain some digits $D=(d_1,d_2\dots)$. Suppose that absent digits are in ascending order $d_1<d_2<\dots$ Consider some number p, which has one or more absent digits. If number p, has only absent digits then it can be pruned, because it does give nothing. If number p has from one to three absent digits, then they should go in strict ascending order $d_1,d_2\dots$, otherwise this number can be pruned. Let us illustrate this by example.

Let the sequence of turns is (0123,0.1)(1245,0.0). All of the numbers from set S don't contain digits $D=(1,2,4,5)$. Consider the number 4058. It has two absent digits 4 and 5. This number can be pruned because the number 1028 is equivalent to the number 4058.

1.10.2 Uncalled digits.

Analogous with absent digits we can use uncalled digits. Let we have some sequence of turns $(t_1,r_1)\dots(t_n,r_n)$ and $A=(a_1,a_2\dots)$ is set of uncalled digits. Assume that uncalled digits are in ascending order, $a_1<a_2<\dots$ It's possible

to prune numbers which have absent digits, which are not ordered in ascending order. Consider the example.

Let we made sequence of turns (0123,0.1)(1245,0.2). Set of uncalled digits is $A = (6, 7, 8, 9)$. Consider the number 7601. It's obviously that it can be pruned, because of the number 6701, which is equivalent to it. Analogously, the number 7123, can be pruned because of the number 6123.

Note. The only distinction from absent digit is that it's not possible to prune all of the numbers which have only uncalled digits. From all of the numbers with uncalled digits only we should consider number 6789.

1.10.3 Numbers which split set of recent numbers into one group.

Assume that we do sequence of turns $(t_1, r_1) \dots (t_n, r_n)$. It's obvious that if some number splits set of recent numbers into only one group then this turn is not best and can be pruned.

1.11 For several turns we got the same response.

Consider sequence of turns $(t_1, r_1) \cdots (t_n, r_n)$. Split all numbers $t_1 \cdots t_n$ on groups with same response. Note $T_{b.c}$ as set of numbers with same response $b.c$, where b is amount of bulls and c is amount of cows.

Example. Let we have sequence of three turns (0123,0.1)(1234,0.1)(5678,0.2), then $T_{0.1} = \{0123, 1234\}, T_{0.2} = \{5678\}$. We can present the sequence using alternate form $(T_{0.1}, 0.1)(T_{0.2}, 0.2)$.

Lemma 11 *Let we have the sequence of turns $(t_1, r_1) \cdots (t_n, r_n)$, which has alternative form $(T_{b_1.c_1}, b_1.c_1) \dots (T_{b_l,c_l}, b_l.c_l)$, and two numbers p and q. Let we have transformation ψ such that all set of numbers $T_{b.c}$ with same responses, transform to itself. $\forall b, \forall c \quad \psi(T_{b.c}) = T_{b.c}$ and number p moves to number q $\psi(p) = q$. Then for each response r and each k*

$$\Phi_k[(t_1, r_1) \cdots (t_n, r_n)(p, r)] = \Phi_k[(t_1, r_1) \cdots (t_n, r_n)(q, r)]$$

Proof. Using lemma 1 and that order of turns is not important we got

$$\Phi_k[(t_1, r_1) \cdots (t_n, r_n)(p, r)] = \Phi_k[(T_{b_1.c_1}, b_1.c_1) \dots (T_{b_l,c_l}, b_l.c_l)(p, r)] =$$

$$= \Phi_k[(\psi(T_{b_1.c_1}), b_1.c_1) \dots (\psi(T_{b_l,c_l}), b_l.c_l)(\psi(p), r)] =$$

$$= \Phi_k[(T_{b_1.c_1}, b_1.c_1) \dots (T_{b_l,c_l}, b_l.c_l)(q, r)] = \Phi_k[(t_1, r_1) \cdots (t_n, r_n)(q, r)]$$

Assertion was proved.

Note. The same is true for Λ function.

Lemma 11 is extension of lemma 1, because numbers t_i don't have to move to itself under transformation ψ. It's sufficient only that all sets with same response move to themselves. Using lemma 11, it's possible to reduce cardinality of sets of third turns when we got the same response on first and second turns.

2 Algorithms.

2.1 The exact algorithm.

The exact algorithm always gives exact estimation. It has two arguments set of numbers S, which satisfy all of the previous moves, and upper estimate β. About estimate β is written in the section about alpha-beta pruning. Algorithm looks over all possible numbers $t \in \Omega$. After each of them set S will be split on subsets with same responses $S_1(t) \dots S_k(t)$, now we need to find estimate for all this subsets, recursively calling the same algorithm. The estimate of turn t will be a sum of estimates of all subsets $S_1(t) \dots S_k(t)$. Minimal estimate for all numbers will be the exact estimation of set S.

2.1.1 Last layer algorithm.

Since we know that sixth turn is last turn it is possible to strictly accelerate algorithm because we should do such turn t that after it all subsets $S_i(t)$ consist of only one item or empty. So, if for some i $|S_i(t)| > 1$ then we go to the next turn. If amount of resent numbers more than 14, then we couldn't solve all numbers using seven or fewer turns, so we can immediately return maximum estimate. If amount of resent numbers equals 14, then we should search using only numbers from set S.

This algorithm always get the exact estimate and uses for all algorithms.

Minimizing 7th. At first looks over only numbers from set S. If such turn splits S on groups with only one item $|S_1(t)| = \dots = |S_n(t)| = 1$, then it is best turn and estimate equals $|S| - 1$.

After that we should look over all other turns. If all groups consist of only one item then it is best turn and amount of 7th equals $|S|$.

If we still couldn't find move which split set S on groups with only one item then algorithm returns estimate 5040, which is worse than any other estimate.

Minimizing average game length. At first looks over only numbers from set S. If such turn splits S on groups with only one item, then it is best turn and estimate equals $\Lambda = 6 + 7(|S| - 1) = 7|S| - 1$.

After that we should look over all other turns. If all groups consist of only one item then it is best turn and amount of 7th equals $\Lambda = 7|S|$.

If we still couldn't find move which split set S on groups with only one item then algorithm returns estimate equals 5040×7, which is worse than any other estimate.

2.2 Heuristic algorithms.

Heuristic algorithms not always give exact estimation, but they are much faster than the exact algorithm. Therefore, we will successively narrow the range of them use, gradually moving to the exact algorithm. For search of minimizing

amount of 7th crush algorithm is better, while Larmouth's algorithm is better for minimizing average game length.

2.2.1 Crush algorithm.

Let we have some set of numbers $S = (s_1, s_2 \dots)$, satisfying all of the previous moves.

We iterate through all possible moves t. Each of them split set S into subsets with the same responses $S_1(t) \dots S_k(t)$. Denote $n_i(t) = |S_i(t)|$ ($\sum n_i(t) = |S|$). We suppose that all subsets $S_i(t)$ sorted by its size, that is $n_1(t) \geq n_2(t) \geq \dots$

Suppose that for the move t we got subsets with sizes $n_1(t), n_2(t) \dots$, and for the move p we got subsets with sizes $n_1(p), n_2(p) \dots$ Assume that move t is better, than move p if $n_1(t) < n_1(p)$ or $n_1(t) = n_1(p)$ and $n_2(t) < n_2(p)$ etc. For example turn with subsets sizes $18, 18, 17 \dots$ is better than turn with subsets sizes $18, 18, 18 \dots$ Crush algorithm selects move which splits set S on smaller parts than all other moves.

Additionally we'll use next rules.

- If found turn t, such that $n_1(t) = 1$, then
  1. if $t \in S$ then it's best turn.
  2. if $t \notin S$, then further search looks over of turns only from S.
- From two turns $t \in S$ with subsets sizes $n_1(t) \dots n_k(t)$ and $p \notin S$ with subsets sizes $n_1(p) \dots n_k(p)$ such that $n_1(t) = n_1(p) \dots n_k(t) = n_k(p)$, we'll select turn t as the best. This means that from moves with same subsets sizes we prefer turns from set S.

2.2.2 Larmouth's algorithm.

Larmouth's algorithm is used from minimizing game length.

We iterate through all possible moves t. Each of them split set S into subsets with the same responses $S_1(t) \dots S_k(t)$. Denote $n_i(t) = |S_i(t)|$ ($\sum n_i(t) = |S|$).

Denote belong function

$$IN(t, S) = \begin{cases} 1 & \text{if } t \in S \\ 0 & \text{if } t \notin S \end{cases}$$

The best turn is turn which minimizes function

$$F(t) = \sum_{n_i(t) > 1} n_i(t) \log(n_i(t)) - 2 \log 2 \times IN(t, S)$$

2.3 Accelerations of algorithm.

Analogously with last layer algorithm it's possible to immediate return maximum estimation on pre last layer if amount of recent numbers is greater than $1 + 13 + 13^2$.

Before to do a search for all possibilities we'll search only from recent numbers, recursively call search algorithm. If estimate e which we get is less than upper estimate β then it's possible to lower upper estimate $\beta = e$.

2.4 Notation of algorithms.

Further we'll use next names of algorithms.

Minimizing 7th.

1. crush35 - crush algorithm is used for turns 3-5, for all others exact algorithm is used.
2. crush45 - crush algorithm is used for turns 4, 5 only.
3. crush5 - crush algorithm is used for turn 5 only.
4. exact - only exact algorithm is used.

Minimizing average game length.

1. avg35 - Larmouth algorithm is used for turns 3-5, for all others exact algorithm is used.
2. avg45 - Larmouth algorithm is used for turns 4, 5 only.
3. avg5 - Larmouth algorithm is used for turn 5 only.

Note. For all of the algorithms for sixth turn the exact algorithm is used.

2.5 Algorithm extension for counting 6th, 5th etc.

Note. This section concerns only minimizing 7th algorithms.

Below we'll build trees for different algorithms for online play with man. Quite often we can see that amount of 7th equals to zero. In this case it's possible to find turn which minimizes amount of 6th. If we can find turn that minimizes amount of 6th to zero, then we minimize amount of 5th etc.

The search algorithm is written such that it can minimize not only 7th but and 6th, 5th etc. It uses set of recent numbers S, upper estimate β and two additional parameters,

count - means what do we need to count: 7th, 6th...

mde - shows max depth to which we want to use the exact algorithm, (for the last layer the exact algorithm is always used). If we count 7th then for sixth

turn we use last layer algorithm. If we want to count 6th then for fifth turn we use last layer algorithm etc.

Note. If algorithm couldn't solve all of the numbers for needed amount of turns then it returns maximum estimate.

3 Algorithm components.

3.1 Alpha-beta pruning.

Alpha-beta pruning is widely used for various games. This greatly speeds up the search, by the way the estimate is left exact. More information can be found on the Internet.

3.2 Sets of the second, third turns and fixed the first one.

Using researches we can use only 0123 as the first turn, numbers from set $s \in S_2$ as the second turn and numbers from sets $S_3(s), s \in S_2$ as the third turn.

3.3 Sorting of second and third turns.

It's well known that looking over „good" turns before „bad" ones accelerates algorithm, so, while we create sets for the second and third turns S_2 and $S_3(s_2)$, reorder them in the inverse order.

3.4 Sorting of other turns.

Turns 4-6 will also reorder in inverse order.

3.5 The response table.

Since one often needs to use response function which count amount of bulls and cows between two numbers, we allocate memory for char array. The size equals 5040×5040. We fill the array with responses before the search. The array size is about 24.2 MB.

3.6 The mechanism of the tasks.

For problems which take a lot of time the mechanism of the tasks was created. At the beginning we should create file jobs.txt. Each line of this file has one problem with its own parameters. One can see example of the jobs.txt file below.

```
solve 0123(0,1)+0145(0,1) 7 4 8 #
solve 0123(0,1)+0134(0,1) 7 4 28 #
solve 0123(0,2)+1234(0,1)+4532(0,1) 7 5 3 #
```

At the beginning one can see keyword "solve" which means that this problem still need to be solved. During of working different processes take tasks from file and change keyword "solve" to another keyword "taken" to show to other processes that this problem already taken for solution. Next the sequence of turns with responses is written. Finally one can see three parameters of algorithm.

- *count* - shows what we need to count: 7 - 7th, 6 - 6th etc.

  Note. This parameter is not used for minimizing game length.

- *mde* - defines from which turn program will use heuristic algorithm: 2 - from third, 3 - from fourth ...

- β - upper estimate beta.

The mechanism of the tasks allows simultaneously solve several problems with many threads. It is created such that it's automatically resume the solving after computer restarts. Thus the problems can be solved several days.

If it's necessary to solve several problems with independent results then it'll be better place longer counting problems at the beginning of the list. It allows to make maximum loading of computer because when all of the threads will finish except one, then it'll be better if last thread will finish as soon as possible.

4 Projects description.

Project includes several applications. All of them is written under c++ except one, which is written under JavaScript and uses for online game between computer and man.

4.1 Console application "executor".

The application uses for tasks calculation which placed in jobs.txt file.

4.2 Windows service.

The service runs several threads of executor.

Calculations for crush5, avg5, exact algorithms take very long time. So we need to use system service of windows which automatically starts upon computer starts. Since the computer for calculations has four cores the service runs four thread with low priority, to not hinder of other programs.

4.3 Windows application - bcw.

Helper routines.

- Creating jobs and gather data for transition algorithm (see 5.2).

- Running of crush35, crush45, avg35, avg45 algorithms.

- Building of trees.
- Loading tree from file, calculation of its statistics and creation tree initialization string for JavaScript.
- Building html file for tree browsing using jQuery.
- Solving (0123,0.1)(1245,0.0) sequence with the exact algorithm for minimizing 7th.

4.4 Console application "shutdown".

The application turns off the computer in special time.

Due to the fact that some problems solving too long, and at night, electricity is about three times cheaper than during the day, the computer many times counted the problems at night. This application automatically turns off the computer when electricity supply meter switches to the daytime rate.

4.5 Tree application (written on JavaScript).

Online game between computer and man.

- Loading tree and counting of its statistics.
- Game with man using storing tree.

5 Minimizing 7th results.

5.1 Crush35 and crush45 algorithms.

Since crush algorithm is much faster the the exact one at first we run crush35 algorithm. Then we run crush45 algorithm only for cases where estimate of 7th is greater than zero with upper estimates which we got from crush35 algorithm. Further we'll do the same for crush5 and exact algorithms. Amount of 7th is

showed in next table.

amount of numbers	first turn	crush35	crush45
1440	0123 0.1	76	53
1260	0123 0.2	22	10
720	0123 1.1	1	0
480	0123 1.0	0	-
360	0123 0.0	0	-
264	0123 0.3	0	-
216	0123 1.2	0	-
180	0123 2.0	0	-
72	0123 2.1	0	-
24	0123 3.0	0	-
9	0123 0.4	0	-
8	0123 1.3	0	-
6	0123 2.2	0	-
1	0123 4.0	0	-
total 5040		99	63

There is no response on very first turn 0123 for which crush35 algorithm returns estimate equals 5040. That means that the algorithm solve all secret numbers using up to seven turns.

5.2 Transition from one algorithm to another.

Lets split receiving of estimations for crush5 and exact algorithms into two phases. At first using the results of algorithm crush45 we got estimations for crush5 then using the results of crush5 we'll move to exact algorithm. Calculations for algorithm crush5 and for exact algorithm take a long time. So solving of this algorithms splits in several steps.

Lets call the algorithm for which we already have estimates as "prev", and one to which we go as "next".

Steps 1-5. Estimation of best turns which we got from prev algorithm..

1. For prev algorithm one find best turn t_1 for very first turn (0123,0.1).
2. Consider all of the responses r_1 such that amount of 7th with prev algorithm is greater than zero $\Phi_7^{prev}[(0123, 0.1)(t_1, r_1)] > 0$.
3. For such responses r_1 count amount of 7th with next algorithm $e_1(r_1) = \Phi_7^{next}[(0123, 0.1)(t_1, r_1)]$ and use as upper estimate β of prev algorithm $\beta = \Phi_7^{prev}[(0123, 0.1)(t_1, r_1)]$.
4. Note $E_1 = \sum_{r_1} e_1(r_1)$

5. Apply steps 1-4 for very first turn (0123,0.2), we'll got best turn t_2 and its estimation E_2. So, using steps 1-5, for the best turns of prev algorithm, we found estimates with next algorithm for this turns.

Step 6. Building table for sequences (0123,0.1)(t,0.2).

6. Run next algorithm for all second turns t except $(t_1, t_2, 0132, 0231, 1032, 1230)$, for sequences (0123,0.1)(t,0.2). For turns t_1, t_2 we already have the estimations. As well we suppose that numbers $0132, 0231, 1032, 1230$ are not the best. We run next algorithm for them at the end. For next algorithm we'll use upper estimate $\beta = \min(E_1, \Phi_7^{prev}[(0123, 0.1)(t, 0.2)])$. Note this estimate as $e_{12}(t) = \Phi_7^{next}[(0123, 0.1)(t, 0.2)]$. Sort moves by ascending of $e_{12}(t)$ and write them into table. Add numbers t_1 and t_2 with its estimates to table.

Note. By sorting numbers in step 6 we in fact sort numbers or problems by time of calculation in descending order. Thus we accelerate the calculations (see 3.6).
Note. By building of the table for turns (0123,0.1)(t,0.2), we can use it to find optimal turns and estimates for the first turns (0123,0.1) and (0123,0.2) (see lemma 3).
Steps 7-8. Searching for amount of 7th for first turn (0123,0.2).

7. Consider turn t with minimal estimation $e_{12}(t)$. For this turn consider all different responses r, for which number of 7th using next algorithm for sequence $(0123, 0.2)(t, r)$ is greater than zero. To accelerate computing we'll use prev algorithm estimations. Denote E_2^* as sum of all such estimations which is estimation of sequence $(0123, 0.2)(t, *)$ by next algorithm. If $E_2^* < E_2$ then decrease the best estimation $E_2 = E_2^*$.

8. For numbers t which have $e_{12}(t) < E_2$ do analogous routine and potentially decrease E_2. The result estimate E_2 is the best estimation of next algorithm for very first turn (0123, 0.2).

Step 9. Searching amount of 7th for the first turn (0123,0.1).

9. Now we should do the same for the first turn (0123,0.1).

Step 10. Check recent numbers.

10. Make sure than turns $t = (0132, 0231, 1032, 1230)$ are not the best. For them run next algorithm for sequences $(0123, 0.1)(t, 0.1)$ and $(0123, 0.2)(t, 0.2)$ with upper estimates $\beta = E_1$ and $\beta = E_2$ respectively.

Now we make transitions from crush45 algorithm to crush5.

5.3 Crush5 algorithm.

5.3.1 Estimations for the best turns (steps 1-5).

Consider the best number for first turn (0123,0.1), which we got by crush45 algorithm. It's number 1245. We can get several responses after it, but only three of them have amount of 7th greater than zero. For this cases run crush5 algorithm.

turn 1	turn 2	crush45	crush5
0123 0.1	1245 0.1	41	38
0123 0.1	1245 0.2	10	7
0123 0.1	1245 0.0	2	1
total		53	46

Analogously, if first turn is (0123,0.2), then best turn is 1435.

turn 1	turn 2	crush45	crush5
0123 0.2	1435 0.1	9	8
0123 0.2	1435 0.2	1	0
total		10	8

Now we got start estimations on maximum amount of 7th. After the first turn (0123,0.1) the estimate is $E_1 = 46$. And after the first turn (0123,0.2) the estimate is $E_2 = 8$.

5.3.2 Building of the table (step 6).

t	(0123,0.1)(t,0.2)
1234	3
1034	4
1245	7
1204	7
1435	8
1045	9
1456	10
0234	10
0245	15
0214	17
0134	18
0456	30
0145	39
0124	41
4567	≥ 46

5.3.3 The best turn after the first turn (0123,0.2) (steps 7-8).

t	(0123,0.1)(t,0.2)	(0123,0.2)(t,0.2)	$E_2 = 8$
1234	3	1	$E_2 = 4$
1034	4	×	
1245	7	×	
1204	7	×	
1435	8	0	
1045	9	×	
1456	10	×	
0234	10	×	
0245	15	×	
0214	17	×	
0134	18	×	
0456	30	×	
0145	39	×	
0124	41	×	
4567	≥ 46	×	

Consider the second turn 1234. Only for responses 0.1 and 0.2 amount of 7th is greater than zero.

turn 1	turn 2	crush45	crush5
0123 0.2	1234 0.1	-	3
0123 0.2	1234 0.2	-	1
0123 0.2	1234 1.1	1	0
total			4

So if the second turn is 1234, then amount of 7th equals 4. From table we can see that we don't need to consider all recent numbers because amount of 7th for them is always greater or equal then four. Thus the best second turn for crush5 algorithm is 1234, and amount of 7th equals 4.

5.3.4 The best turn after (0123,0.1) (step 9).

t	(0123,0.1)(t,0.2)	(0123,0.1)(t,0.1)	$E_1 = 46$
1234	3	≥ 43	$E_1 \geq 46$
1034	4	≥ 42	$E_1 \geq 46$
1245	7	38	$E_1 = 46$
1204	7	≥ 39	$E_1 \geq 46$
1435	8	≥ 38	$E_1 \geq 46$
1045	9	≥ 37	$E_1 \geq 46$
1456	10	≥ 36	$E_1 \geq 46$
0234	10	≥ 36	$E_1 \geq 46$
0245	15	≥ 31	$E_1 \geq 46$
0214	17	≥ 29	$E_1 \geq 46$
0134	18	≥ 28	$E_1 \geq 46$
0456	30	≥ 16	$E_1 \geq 46$
0145	39	≥ 7	$E_1 \geq 46$
0124	41	≥ 5	$E_1 \geq 46$
4567	≥ 46	$\times$	

Consider the second turn 1245 which we got as the best turn of crush45 algorithm. After it no one number gives the better result.

5.3.5 Checking numbers with digits 0-3 (step 10).

At the end we should make sure that numbers with digits from 0 to 3 only are not the best turns.

turn 1	turn 2	crush5
0123 0.1	0132 0.1	≥ 46
0123 0.1	0231 0.1	≥ 46
0123 0.1	1032 0.1	≥ 46
0123 0.1	1230 0.1	≥ 46

turn 1	turn 2	crush5
0123 0.2	0132 0.2	≥ 4
0123 0.2	0231 0.2	≥ 4
0123 0.2	1032 0.2	≥ 4
0123 0.2	1230 0.2	≥ 4

5.3.6 The results of crush5 algorithm.

After the first turn (0123,0.1) the best second turn is 1245 and amount of 7th is 46. After the first turn (0123,0.2) the best second turn is 1234 and amount of 7th is 4.

5.4 The exact algorithm.

Unfortunately the exact algorithm take a long time. At the moment one searches servers to get the exact estimation.

5.5 Comparison table of different algorithms.

turn 1	crush35	crush45	crush5
0123 0.1	76	53	46
0123 0.2	22	10	4
0123 1.1	1	0	0
total 7th	99	63	50

6 Minimizing amount of 6th, 5th etc.

We considered only responses on first turn 0123 which give amount of 7th greater than zero. It is only two responses 0.1 and 0.2. For all others we can minimize amount of 6th. If minimal amount of 6th is zero then we can minimize amount of 5th etc. Problem of minimizing 6th is much more easier because we should search with lower depth. Minimizing table is showed below where first response 4.0 was omitted.

turn 1	7th	6th	5th	4th
0123 0.1	≤ 46			
0123 0.2	≤ 4			
0123 1.1	0	213		
0123 1.0	0	88		
0123 0.0	0	84		
0123 0.3	0	20		
0123 1.2	0	8		
0123 2.0	0	4		
0123 2.1	0	0	28	
0123 3.0	0	0	2	
0123 0.4	0	0	0	6
0123 1.3	0	0	1	
0123 2.2	0	0	0	4

7 Minimizing average game length results.

The problem of minimizing average game length is much harder than 7th minimizing, but we know the exact estimation for it (see [1] and [2]). So it's sufficient to find algorithm with optimal estimation. At first run avg35 and avg45 algorithms. After that run very fast algorithm which is not use heuristic algorithms, but tries only recent numbers for turns 4-6 ("fso" column).

Write the exact algorithm results from [2] to the "exact" column of below table. Let's compare minimal estimation of avg45 and fso algorithms with the exact algorithm results. We can see that optimal estimate is not achieved for

answers 0.1, 0.2 and 1.1. So, we need to run avg5 algorithm only for that cases.

turn 1	avg35	avg45	fso	min(avg45,fso)	exact	avg5
0123 0.0	1808	1807	1806	1806	1806	1806
0123 0.1	8009	7951	7942	7942	7935	7935
0123 0.2	6828	6817	6818	6817	6808	6808
0123 0.3	1284	1268	1269	1268	1268	1268
0123 0.4	32	32	32	32	32	32
0123 1.0	2400	2394	2393	2393	2393	2393
0123 1.1	3731	3717	3716	3716	3712	3712
0123 1.2	1031	1020	1020	1020	1020	1020
0123 1.3	30	30	30	30	30	30
0123 2.0	845	839	839	839	839	839
0123 2.1	314	312	312	312	312	312
0123 2.2	21	21	21	21	21	21
0123 3.0	97	97	97	97	97	97
0123 4.0	1	1	1	1	1	1
total turns	26 431	26 306	26 296	26 294	26 274	26 274
average length	5.2442	5.2194	5.2175	5.2171	5.2131	5.2131

From the resulting table we see that avg5 algorithm gives the exact estimation. Thus, the minimum possible length of the game is reached.

8 Tree building.

To create program which guesses secret numbers and not use long calculations we need to preliminary build the tree and store it to file, then load the tree in separate program and use it without any calculations.

Minimizing 7th. During of tree building we'll minimize not only 7th. If amount of 7th for some node equals to zero we'll minimize 6th. If exists turn which have amount of 6th equals to zero, then we'll minimize 5th etc. To minimize 6th, 5th we use the exact algorithm. To minimize 7th for first turns (0123,0.1) and (0123,0.2), we'll use one of the algorithms crush45, crush5. Lets call such algorithms as optimized ones and note them as crush45o, crush5o. The algorithms have the same amount of 7th with its not optimized copies but they strictly reduce amount of 6th, 5th etc. To accelerate construction of the tree we use preliminary counted first three turns.

During of tree building for every node we store additional parameter $n-th$, which means from what we need start to count for its children. For example if $n-th=5$ then all of the children start to count amount of 5th. As well we store estimation itself e, which helps for tree building. For example if $n-th=5$ and $e=30$, then sum of 5th for all children equals to 30.

Minimizing average game length. During of tree building for every node we store estimate e, which helps for tree building. For example if $e = 30$, then sum of estimations of all children equals to 30. We use preliminary counted first three turns also.

8.1 Realizing of algorithm on c++.

For tree checking at first program on c++ is used. It loaded tree and run for all secret numbers until it get response 4.0. After it program gathers statistics how many numbers were guessed using exact seven, six etc. turns, and how many totally turns is needed to guess secret numbers.

The program constructs html file for tree browsing. The example you can see here - `http://bulls-cows.sourceforge.net/crush5oTree.html`.

8.2 Realizing of algorithm on JavaScript.

After successful checking of built trees using program on c++ we can create web implementation of algorithm. In order not to load server we create program on JavaScript.

JavaScript language does not use pointers. So for every tree node we assign unique identifier *id*. Instead of children's pointers we'll use them *id*-s.

The goal is to make size of script as small as possible, so we'll use next rules

1. The tree is given by one string. The symbol „!“ is used, as nodes separator.

2. We store all of the items in sequence, that is the first item has $id = 0$, the second one $id = 1$ etc. Thus, we don't need to use *id* as parameter. All what we need is index of best number (from 0 to 5039) and array of *id*-s of children for all possible responses.

3. Every tree node has children. Lets enumerate all responses. The total amount of responses is 14, but we will not use response 4.0, so we'll use 13 *id*-s of children.

4. We write all of the parameters using symbols with ASCII codes from 35 to 127, without symbol „\“(which has ASCII code 92), because it needs to use two symbols to pass it to JavaScript. So we use 92-th system of numeration. The symbols which have ASCII codes greater than 127 are not used because they are recognized differently by different browsers. The maximum number what we need to pass is lower than 92×92, so for passing every number we need up to two bytes.

5. If node has no children then array of *id*-s of children is not passed as parameter.

6. It's possible to pass array of children *id*-s by two ways. The first one is to pass all 13 children *id*-s (two bytes for every *id*). The second one pass pairs „array_index (one byte from 0 to 13) and index_value (2 bytes)“.

Amount of bytes using second type is a multiple of three. At the same time amount of bytes using first type equals 26. Thus, it's always possible to define what type of passing is used. During node storing we'll use type of passing which has the shorter string.

By using all of this optimizations the size of tree string reduces from 222 KB to 31.6 KB.

9 Calculation times.

9.1 Crush35 and crush45 algorithms.

Sample row

```
crush35 (0,1) max=5040 e=76 time 00:00:01 best=1456
```

means that it was ran crush35 algorithm after turn (0123,0.1) with upper estimate 5040. The result is new estimate 76 with best number 1456. The calculations take one second.

```
solving 0123(*.*) count 7th
-----------------
crush35 (0.0) max=5040 e= 0 time 00:00:00 best=4567
crush35 (0.1) max=5040 e=76 time 00:00:01 best=1456
crush35 (0.2) max=5040 e=22 time 00:00:01 best=1234
crush35 (0.3) max=5040 e= 0 time 00:00:00 best=4567
crush35 (0.4) max=5040 e= 0 time 00:00:00 best=1456
crush35 (1.0) max=5040 e= 0 time 00:00:00 best=0134
crush35 (1.1) max=5040 e= 1 time 00:00:00 best=1456
crush35 (1.2) max=5040 e= 0 time 00:00:00 best=4567
crush35 (1.3) max=5040 e= 0 time 00:00:00 best=1456
crush35 (2.0) max=5040 e= 0 time 00:00:00 best=4567
crush35 (2.1) max=5040 e= 0 time 00:00:00 best=4567
crush35 (2.2) max=5040 e= 0 time 00:00:00 best=1456
crush35 (3.0) max=5040 e= 0 time 00:00:00 best=4567
total 99 time 00:00:03
-----------------
crush45 (0.1) max=  76 e=53 time 00:05:31 best=1245
crush45 (0.2) max=  22 e=10 time 00:02:31 best=1435
crush45 (1.1) max=   1 e= 0 time 00:00:04 best=1456
total 63 time 00:08:07
```

After using of crush35 algorithm, only for cases where amount of 7th is greater than zero, we run crush45 algorithm, with upper estimate which we got from crush35 algorithm.

9.2 Crush5 algorithm.

Crush5 algorithm ran using tasks mechanism so there are no data about time of calculation.

9.3 Avg35 and avg45 algorithms.

Sample row

```
avg35 (0.0) max=**** e=1808 time 00:00:01 best=4567
```

means that it was ran avg35 algorithm after turn (0123,0.0) with maximum upper estimate. The result is new estimate 1808 with best number 4567. The calculations take one second.

```
solving 0123(*.*)
-----------------
avg35 (0.0) max=**** e=1808 time 00:00:01 best=4567
avg35 (0.1) max=**** e=8009 time 00:00:02 best=1245
avg35 (0.2) max=**** e=6828 time 00:00:05 best=1435
avg35 (0.3) max=**** e=1284 time 00:00:01 best=1245
avg35 (0.4) max=**** e=  32 time 00:00:00 best=1230
avg35 (1.0) max=**** e=2400 time 00:00:01 best=0456
avg35 (1.1) max=**** e=3731 time 00:00:03 best=0145
avg35 (1.2) max=**** e=1031 time 00:00:00 best=0245
avg35 (1.3) max=**** e=  30 time 00:00:00 best=1234
avg35 (2.0) max=**** e= 845 time 00:00:00 best=0456
avg35 (2.1) max=**** e= 314 time 00:00:00 best=0245
avg35 (2.2) max=**** e=  21 time 00:00:00 best=0132
avg35 (3.0) max=**** e=  97 time 00:00:00 best=0245
avg35 (4.0) max=**** e=   1 time 00:00:00 best=0123
total 26 431     avg=5.2442 time 00:00:17
-----------------
avg45 (0.0) max=1808 e=1807 time 00:08:23 best=4567
avg45 (0.1) max=8009 e=7951 time 00:29:51 best=1456
avg45 (0.2) max=6828 e=6817 time 00:32:22 best=1435
avg45 (0.3) max=1284 e=1268 time 00:06:30 best=1435
avg45 (0.4) max=  32 e=  32 time 00:00:00 best=-001
avg45 (1.0) max=2400 e=2394 time 00:11:44 best=0456
avg45 (1.1) max=3731 e=3717 time 00:19:45 best=0245
avg45 (1.2) max=1031 e=1020 time 00:05:22 best=0245
avg45 (1.3) max=  30 e=  30 time 00:00:01 best=-001
avg45 (2.0) max= 845 e= 839 time 00:03:48 best=0245
avg45 (2.1) max= 314 e= 312 time 00:01:16 best=0145
avg45 (2.2) max=  21 e=  21 time 00:00:00 best=-001
avg45 (3.0) max=  97 e=  97 time 00:00:15 best=-001
avg45 (4.0) max=   1 e=   1 time 00:00:00 best=0123
total 26 306     avg=5.2194 time 01:59:22
```

9.4 Avg5 algorithm.

Avg5 algorithm ran using tasks mechanism so there are no data about time of calculation.

10 Edition history.

- First edition 25th of December 2008 (only in Russian)
- Second edition 1st of November 2011
  - Article was strongly extended and supplemented.
  - Article was translated from Russian to English.
  - The site of project was created.

References

[1] John Francis, Strategies for playing MOO, or Bulls and Cows.
`http://www.jfwaf.com/Bulls%20and%20Cows.pdf`

[2] Tetsuro Tanaka, An Optimal MOO Strategy, Game Programming Workshop - Japan.