Alexey Slovesnov

email slovesnov@yandex.ru

site http://slovesnov.users.sf.net/?bullscows,english

Optimal algorithms for mastermind and bulls-cows games.

Abstract.

The article concerns two optimization criteria of mastermind and bulls-cows games. The first one is minimizing average amount of turns to guess arbitrary secret number (minimizing average game length). It will be proved that there is no algorithm which can guess all numbers for up to six turns for bulls and cows game, but there are algorithms which can guess all of the numbers for up to seven turns. Also it will be proved that there is no algorithm which can guess all numbers for up to four turns for mastermind game, but there are algorithms which can guess all of the numbers for up to five turns. So the second optimization criterion is minimizing amount of numbers which algorithm guesses using exactly seven turns for bulls and cows game and exactly five turns for mastermind, all others should be guessed for lower number of turns. For all of optimal algorithms trees with statistics are built. Web application is created where computer can guess numbers using one of the optimal algorithms.

30 January 2017

Edition 3.1

address of last edition of article
http://slovesnov.users.sf.net/bullscows/bullscows.pdf

Contents

1 Introduction. **2**

2 Theory. **3**
2.1 Notation. 3
2.2 Transformations. 4
2.3 The equivalence classes. 5
2.4 Sets of first-third moves. 6
2.5 Algorithms accelerations. 7
2.6 Unused cutoffs. 12

3 Algorithms. **14**
3.1 The exact algorithm. 14
3.2 Heuristic algorithms. 14
3.3 Notation of algorithms. 15
3.4 Extension for counting 6-movers, 5-movers etc. 16

4 Projects description. **16**
4.1 Windows application - bcw. 16
4.2 Javascript application. 16
4.3 The mechanism of the tasks [only bulls and cows]. 17
4.4 Problems executor [only bulls and cows]. 17
4.5 Windows service [only bulls and cows]. 17

5 Results. **18**
5.1 Minimal amount of numbers. 18
5.2 Minimizing average game length. 25

6 Tree building. **27**
6.1 Tree building on c++. 27
6.2 Tree building on javascript. 28

7 Edition history. **30**

References. **30**

1 Introduction.

An secret number for bulls-cows game can be four-place decimal with different digits from 0 to 9. Total amount of secret numbers is $10!/6! = 5040$. For mastermind secret number is four-place digital with digits from 0 to 5. Digits can repeat. So total amount of secret numbers is $6^4 = 1296$. In common case such games have three parameters - number of positions, number of different digits and repeatability (1 - yes, 0 -no). From this point of view bulls-cows game has parameters (4, 10, 0), mastermind has parameters (4, 6, 1). Optimization of two game types has two directions.

Minimal average game length. Minimizing of average amount of turns to guess arbitrary secret number. Since this problem is solved for bulls-cows (see [1] and [2]), average game length is 26274/5040=5.21, then it's sufficient to find algorithm with such average game length. Also such problem is solved for mastermind game (see [3]). Minimal average game length is 5625/1296=4.34 if number of turns is arbitrary (six turns is enough) and 5626/1296=4.34 is we can do maximum five turns. For this optimization type we construct three optimal algorithms one for bulls-cows game and two for mastermind game.

Minimal amount of numbers. It's known for bulls-cows game that there is no algorithm which can guess each secret number using six or less turns. Also it'll be proven here. At the same time there are algorithms which can guess each number using up to seven moves. So the problem is to minimize amount of numbers which algorithm can guess using exactly seven turns. Similarly for mastermind game there is no algorithm which can guess any secret number for up to four turns, but there are algorithms which can do this using maximum five turns. There are no articles about this criterion were found in the internet so we need to do exhaustive search. For bulls and cows we'll prove that minimal amount of numbers which algorithm can guess using exactly seven turns is 50, all others are guessed for up to six turns. For mastermind we'll prove that minimal amount of numbers which algorithm can guess using exactly five turns is 539. Also it proves that there is no algorithm which can guess any number for up to six turns for bulls and cows and there is no algorithm which can guess any number for up to four turns for mastermind.

Searching of minimal average game length or minimal amount of numbers are very difficult problems. Let's try roughly estimate number of operations for exhaustive search for bulls and cows. It's obviously that we can use number 0123 as first turn without loss of generality. Turns from second to sixth can be one of 5039 possibilities each (number 0123 is not used). After for every turn we can get up to 14 responses, for one of them (four bulls and zero cows) we don't need to do 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 the estimation for amount of nodes for exhaustive search is $\left(\frac{13}{4}\right)^6 \cdot \left(\frac{5039}{4}\right)^5 \approx 3.7 \times 10^{18}$. This estimation looks like huge even for

modern computers.

The article consists of several parts: notation and mathematical theory, algorithms description and components of program, results and trees of optimal algorithms with statistics. The trees will be used for web application where computer can guess numbers.

2 Theory.

2.1 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 for bulls and cows note as $\Omega^b = (0123, 0124 \dots 9876)$. For mastermind $\Omega^m = (0000, 0001 \dots 5555)$. If set Ω is used without upper index then it means both game types.

Set of all possible answers note as $R = (0.0, 0.1 \dots 4.0)$. It's the same set for both game types.

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

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

Minimal amount of numbers which can be guessed using exactly k attempts, for all algorithms, after sequence $T = (t_1, r_1) \dots (t_n, r_n)$ note as $\Phi_k[T] = \Phi_k[(t_1, r_1) \dots (t_n, r_n)]$. If Φ is used without superscript then it means both game types.

Minimal average game length for all algorithms, after sequence of turns T note as $\Delta[T]$. Suppose that after sequence of turns T 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 $\Lambda[T] = m \times \Delta[T]$.

Consider sequence of turns $T = (t_1, r_1) \dots (t_n, r_n)$ 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 $\Phi_k[T(p, *)] = \Phi_k[(t_1, r_1) \dots (t_n, r_n)(p, *)]$.

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

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 position from 1 to 4. First digit has position 1, second one 2, etc. Every digit can be from 0 to 9 for bulls and cows and from 0 to 5 for mastermind. Denote set of all transformations of digits and positions 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 (number of elements) of set S as $|S|$. For example, $|\Omega^m| = 6^4 = 1296$.

Empty set - $\varnothing$.

The goal of article is to find $\Phi_7^b[\varnothing]$ for bulls-cows and $\Phi_5^m[\varnothing]$ for mastermind, and construct algorithms with optimal estimations. Since it's known that $\Lambda^b[\varnothing] = 26274$, $\Lambda^m[\varnothing] = 5625$ and $\Lambda^m[\varnothing] = 5626$ if maximum five turns are allowed, then we only need to construct such algorithms.

2.2 Transformations.

It's possible to do any permutation 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 *[bulls and cows only] For every two numbers $t_1 \in \Omega^b$ and $t_2 \in \Omega^b$ there is exists transformation ψ such as $\psi(t_1) = t_2$ and $\psi(t_2) = t_1$.*

Proof. By writing computer program which looks over all possible numbers $t_1 \in \Omega^b$ and $t_2 \in \Omega^b$ 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 transforms to number $\psi_1(0123) = 1054$, and number 4051 transforms 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 *[bulls and cows only] For every two numbers t_1 and t_2 and every two responses r_1 and r_2*

$$\Phi_k^b[(t_1, r_1)(t_2, r_2)] = \Phi_k^b[(t_1, r_2)(t_2, r_1)]$$

$$\Phi_k^b[(t_1, r_1)(t_2, *)] = \Phi_k^b[(t_2, r_1)(t_1, *)]$$

Proof. From lemma 2 we know that, exists transformation ψ, such that $\psi(t_1) = t_2, \psi(t_2) = t_1$. Thus, using lemma 1, we have:

$$\Phi_k^b[(t_1, r_1)(t_2, r_2)] = \Phi_k^b[(\psi(t_1), r_1)(\psi(t_2), r_2)] = \Phi_k^b[(t_2, r_1)(t_1, r_2)]$$

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

$$\Phi_k^b[(t_2, r_1)(t_1, r_2)] = \Phi_k^b[(t_1, r_2)(t_2, r_1)]$$

So first was proved. Similarly,

$$\Phi_k^b[(t_1, r_1)(t_2, *)] = \sum_r \Phi_k^b[(t_1, r_1)(t_2, r)] = \sum_r \Phi_k^b[(t_2, r_1)(t_1, r)] = \Phi_k^b[(t_2, r_1)(t_1, *)]$$

Assertion was proved.

Note. The same is true for Λ^b function.

2.3 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 4 *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 5 *If two numbers p and q belongs to one equivalence class, for sequence $(t_1, r_1) \cdots (t_n, r_n)$ then $\forall k$, $\forall r_1 \dots r_n$, $\forall r$*

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

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

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.

Values of Φ_k and Λ for some sequence $t_1 \dots t_n$ are the same for all equivalent numbers, so for exhaust search it's sufficient to explore only one number for each equivalence class. Let's call such set as equivalence set for sequence $t_1 \dots t_n$.

Algorithm of construction of equivalence set for sequence $t_1 \dots t_n$. Preset set $S = (t_1 \dots t_n)$, then explore all numbers $p \in \Omega$. If there is no exists transformation ψ such that, $\psi(t_i) = t_i \ \forall i$ and $\psi(p) \in S$, then add p to set S. At the end remove numbers $t_1 \dots t_n$ from set S because this numbers are from previous moves and will not be best. So we got equivalence set S for sequence $t_1 \dots t_n$.

2.4 Sets of first-third moves.

2.4.1 Set of first moves.

Consider empty sequence of turns $\varnothing$ and build equivalence set for it. For bulls and cows this set has only one item $S_1^b = (0123)$ and five items for mastermind $S_1^m = (0000, 0001, 0011, 0012, 0123)$. So next assertion is true.

Lemma 6 *For the first turn it's sufficient search only numbers from set S_1.*

$$\Phi_k[\varnothing] = \min_{s_1 \in S_1} \Phi_k[(s_1, *)] \quad \Lambda[\varnothing] = \min_{s_1 \in S_1} \Lambda[(s_1, *)]$$

2.4.2 Set of second moves.

Consider item from set of first moves $s_1 \in S_1$ and construct equivalence set for it $S_2(s_1)$ for sequence s_1. It's sufficient to explore only moves from $S_2(s_1)$ after first turn s_1. For bulls and cows we got

$$S_2^b(0123) = (0124, 0132, 0134, 0145, 0214, 0231, 0234, 0245, 0456,$$

$$1032, 1034, 1045, 1204, 1230, 1234, 1245, 1435, 1456, 4567)$$

For mastermind we got five sets $S_2^m(s_1), s_1 \in S_1^m$ which have totally 284 items. So next assertion is true.

Lemma 7 *If the first turn was $s_1 \in S_1$, then it's sufficient to search only items from set $S_2(s_1)$.*

$$\Phi_k[(s_1,r)] = \min_{s_2 \in S_2(s_1)} \Phi_k[(s_1,r)(s_2,*)] \quad \Lambda[(s_1,r)] = \min_{s_2 \in S_2(t)} \Lambda[(s_1,r)(s_2,*)] \quad \forall r$$

2.4.3 Set of third moves.

Similarly we can got set of third moves. Amount of such sets for mastermind game is too big, so we use them only for bulls and cows. All of the sets $S_3^b(s_1,s_2)$ for bulls and cows totally include 7072 items. Now for first three moves we need to search only 7072 numbers, instead of $5039^2 \approx 2.5 \times 10^7$.

2.5 Algorithms accelerations.

2.5.1 Order of numbers.

Search algorithm is working faster when it explores best turns at first. Later we'll construct trees for optimal algorithms. Most of the best turns satisfy of all previous moves. It's true for 95-98% of cases. So at first better to explore turns which satisfy of all previous moves, then all others.

2.5.2 Subsets estimation.

Lemma 8 *$\forall R' \subset R$ and any sequence of turns $T = (t_1,r_1)\dots(t_n,r_n)$, and number t next assertion is true*

$$\Phi_k[T(t,*)] \geq \sum_{r \in R'} \Phi_k[T(t,r)]$$

particularly

$$\Phi_k[T(t,*)] \geq \Phi_k[T(t,r)] \ \ \forall r \in R$$

Proof. Using formula 1 we got that

$$\Phi_k[T(t,*)] = \sum_{r \in R} \Phi_k[T(t,r)] = \sum_{r \in R'} \Phi_k[T(t,r)] + \sum_{r \notin R'} \Phi_k[T(t,r)] \geq \sum_{r \in R'} \Phi_k[T(t,r)]$$

Assertion was proved.

Note. The same is true for Λ function.

Let we have sequence $T = (t_1,r_1)\dots(t_n,r_n)$, upper bound β and some number t. Let for some subset of responses $R' \subset R$ we have that $\sum\limits_{r \in R'} \Phi_k[T(t,r)] \geq \beta$, then this number is not best turn and can be pruned.

2.5.3 Minimal estimation.

Minimal amount of numbers. 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 number 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 $\Lambda = k + (|S| - 1)(k + 1) = |S|(k + 1) - 1$.

2.5.4 Fast estimation.

Let the sequence of turns $T = (t_1, r_1) \dots (t_n, r_n)$ have been made. Denote $S(T) = (s_1, s_2 \dots)$ as set of numbers, which satisfies all of the turns from the sequence. In some cases when set S is small, we can immediately obtain the estimation.

Minimal amount of numbers. If we do the seventh turn for bulls-cows and fifth turn for mastermind then

$$\Phi_7^b[T] = \Phi_5^m[T] = \begin{cases} 1 & \text{if } |S(T)| = 1 \\ 5040 & \text{if } |S(T)| > 1 \end{cases}$$

If we do the sixth turn for bulls-cows and fourth turn for mastermind then

$$\Phi_7^b[T] = \Phi_5^m[T] = \begin{cases} & \text{if } |S(T)| = 1 \text{ or } |S(T)| = 2 \text{ or} \\ |S(T)| - 1 & |S(T)| = 3 \text{ and for one of items } s_i \\ & \text{two others give different responses} \end{cases}$$

If we do turn from first to fifth for bulls-cows and from first to third for mastermind then

$$\Phi_7^b[T] = \Phi_5^m[T] = \begin{cases} & \text{if } |S(T)| = 1 \text{ or } |S(T)| = 2 \text{ or} \\ 0 & |S(T)| = 3 \text{ and for one turn } s_i \text{ two} \\ & \text{others give different responses} \end{cases}$$

Later we will count not only 7-movers but 6-movers, 5-movers etc for bulls-cows and 4-movers, 3-movers etc for mastermind (see. 3.4). So the above formula can be extended.

If we do k-th turn.

$$\Phi_k^b[T] = \Phi_k^m[T] = \begin{cases} 1 & \text{if } |S(T)| = 1 \\ 5040 & \text{if } |S(T)| > 1 \end{cases}$$

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

$$\Phi_k^b[T] = \Phi_k^m[T] = \begin{cases} & \text{if } |S(T)| = 1 \text{ or } |S(T)| = 2 \text{ or} \\ |S(T)| - 1 & |S(T)| = 3 \text{ and for one turn } s_i \text{ two} \\ & \text{others give different responses.} \end{cases}$$

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

$$\Phi_k^b[T] = \Phi_k^m[T] = \begin{cases} & \text{if } |S(T)| = 1 \text{ or } |S(T)| = 2 \text{ or} \\ 0 & |S(T)| = 3 \text{ and for one turn } s_i \text{ two} \\ & \text{others give different responses.} \end{cases}$$

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

$$\Lambda[T] = k + (|S(T)| - 1)(k+1) = |S(T)|(k+1) - 1$$

2.5.5 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.

2.5.6 Penultimate turn algorithm.

Penultimate turn should be such turn t that after it all of subsets $S_i(t)$ consist of only one item or empty. If for some i $|S_i(t)| > 1$ then we go to the next turn. Total number of different responses is 14. So if amount of resent numbers is more than 14, then we couldn't solve all of them using certain number of turns, so we can immediately return maximum estimate. If amount of resent numbers equals to 14, then we should search using only recent numbers. This algorithm always get the exact estimate and is used for all algorithms.

Minimal amount of numbers. 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 estimate is $|S|$. If we still couldn't find move which splits set S on groups with only one item then algorithm returns worst estimate.

Minimal average game length. Suppose that we do k-th turn. For example we do sixth turn for bulls-cows and count 7-movers. At first looks over only numbers from set S. If number splits S on groups with only one item, then it is best turn and estimate equals $k + (|S| - 1)(k+1) = (k+1)|S| - 1$. After that we should look over all other turns. If all groups consist of only one item then it is best turn and estimate equals $(k+1)|S|$. If we still couldn't find move which splits set S on groups with only one item then algorithm returns worst estimate.

2.5.7 Cutoffs of last but two turn.

Minimal amount of numbers. Similarly with penultimate turn algorithm it's possible to immediate return maximum estimation for last but two turn if amount of recent numbers is greater than $1 + 13 + 13^2 = 183$. If amount of recent numbers equals 183 then it's sufficient to explore only recent numbers. We can reduce number 183 for bulls and cows game. The first turn can split recent number for maximum 14 groups. For one of them with response 4.0 we don't need further searching. For answer 2.2 we can get maximum 6 numbers, for response 1.3 - 8, for 0.4 - 9. For rest $13-3 = 10$ groups we can get maximum 13 numbers. So we can guess maximum 1 number for one turn, maximum 13 numbers for two turns, and maximum $10 \cdot 13 + 6 + 8 + 9 = 153$ for three turns. So totally we have $1 + 13 + 153 = 167$ turns.

Fast cutoffs. Let we have upper estimation β, and number t which splits recent numbers on sets $S_1 \cdots S_n$. Assume that for some of them $S_i, 1 \leq i \leq m$ we already get estimations e_i. Sets S_i will be estimated by penultimate turn algorithm and sometimes turn t can be pruned.

Minimal amount of numbers. Minimal estimate of set S_i by penultimate turn algorithm equals $|S_i| - 1$. So minimal estimation of turn t, after estimation of first m sets, equals $E(m) = \sum\limits_{i=1}^{m} e_i + \sum\limits_{i=m+1}^{n} (|S_i|-1)$. If $E(m) \geq \beta$, then turn t can be pruned. If we count one more estimation e_{m+1}, we can count $E(m+1)$ and potentially prune the turn.

Average game length. Assume that penultimate turn has number $k-1$. Minimal estimation of set S_i by penultimate turn algorithm equals $(k+1)|S_i| - 1$. So minimal estimation of turn t, after estimation of first m sets, equals $E(m) = \sum\limits_{i=1}^{m} e_i + \sum\limits_{i=m+1}^{n} ((k+1)|S_i|-1)$ If $E(m) \geq \beta$, then turn t can be pruned. If we count one more estimation e_{m+1}, we can count $E(m+1)$ and potentially skip the turn.

2.5.8 Equivalent splits.

Consider two numbers p and q. Let both of them are recent numbers or both of them are not recent numbers. Assume that p and q split recent numbers set for the same amount of subsets $S_{p_1} \ldots S_{p_n}$ and $S_{q_1} \ldots S_{q_n}$. Take minimal number mp_i from set S_{p_i}. Order sets S_{p_i} by them minimal numbers $mp_i < mp_{i+1}$. The same do for S_{q_i}. If set S_{p_i} equals S_{q_i} for all i, then numbers p and q give the same splits and one of them can be pruned.

It's obvious that all of the sets with only one item has the same estimation. The same is true for sets with two items and for sets with three items such that for one item two others have different responses. So for number p, we to count three numbers: amount of sets with size one n_{p_1}, amount of sets with size two

n_{p_2}, and amount of special sets with size three n_{p_3}; and all other sets $S_{p_1} \dots S_{p_n}$. Now for comparing p and q we need to compare n_{p_i} and n_{q_i}, next other sets. In this case amount of pruned turns increases because it's not important which numbers have sets with size 1, 2 and special sets with size 3.

The total number of splits equals $|\Omega|$ multiples by depth count. It's not big number so we can store all of splits. For searching equivalent splits we need to go over maximum $|\Omega|$ of splits, which takes a long time. To reduce it set hash code for each split and go over only for splits with same hash code.

2.5.9 Hashing.

Let we need to estimate some set S. It's possible that search algorithm is already estimate this set earlier. If store set and estimation then it's possible to not estimate it and just return stored estimation. Consider set $\Omega = (\omega_1 \cdots \omega_{|\Omega|})$. Subset $S \subset \Omega$ can be set by bit sequence with length $|\Omega|$, i-th bit of sequence equals one if $\omega_i \in S$. The total amount of sequences is huge $2^{|\Omega|}$, so we store only some of them. The algorithms have next options (see 3).

- Starting from some number of turn *mde* (except penultimate and last turn), we use heuristic algorithm. This parameter defines algorithm type.
- Only recent numbers are explored.

For fast searching of set S in hash table we use the same idea which was used for equivalent splits (hash code of sets).

Hash table item. To store set S with estimation, we use:

- bit sequence with length $|\Omega|$
- algorithm type - *mde*
- one bit *heuristic*. Bis is set if we use heuristic estimation.
- one bit $fromSetOnly$. Bit is set if we explore turns only from set S
- number of current turn n
- estimation flag (the exact estimation or $\geq \beta$)
- estimation of set or upper bound β (depends on estimation flag)
- the best turn (for tree building)

Hash table checking. If set S is found in hash table that is bit sequence, algorithm type *mde*, *heuristic* and $fromSetOnly$ bits are the same, then return value depends on estimation flag.

Estimation flag - the exact estimation. It's possible three options.

1. Number of current turn equals to number of turn from has table. In this case estimation from hash table returns.
2. Number of turn n is less than number of turn from hash table n_{hash}. For minimal amount of turns criterion zero estimation returns. For average game length $e_{hash} - |S|(n_{hash} - n)$ returns.
3. Number of turn n is greater than number of turn from hash table n_{hash}. The worst estimation returns because we couldn't guess all recent number for certain amount of turns.

Estimation flag - $\geq \beta$. It means that estimation of set S is $\geq \beta$. It's possible three options.

1. Number of current turn equals to number of turn from has table and $\beta \leq \beta_{hash}$. In this case β returns.
2. Number of turn n is less than number of turn from hash table n_{hash}. Only for average game length criterion if $\beta \leq e_{hash} - |S|(n_{hash} - n)$ then β returns.
3. Number of turn n is greater than number of turn from hash table n_{hash} and estimation from hash table is greater than zero. The worst estimation returns because we couldn't guess all recent number for certain amount of turns.

2.5.10 Response table.

We often need to count amount of bulls and cows between two numbers. To not count response every time we allocate char array with size $|\Omega|^2$ and fill it with responses before the search. The response table strongly accelerates the search.

2.6 Unused cutoffs.

2.6.1 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, otherwise this number can be pruned. Consider the example. Let the sequence of turns for bulls and cows game 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.

2.6.2 Uncalled digits.

Similarly reasoning can be used for uncalled digits. Let we have some sequence of turns $(t_1, r_1) \dots (t_n, r_n)$ and $D = (d_1, d_2 \dots)$ is set of uncalled digits. Assume that uncalled digits are in ascending order, $d_1 < d_2 < \dots$ It's possible to prune numbers which have absent digits, which are not ordered in ascending order. Consider the example. Let for bulls and cows game we made sequence of turns (0123, 0.1)(1245, 0.2). Set of uncalled digits is $D = (6, 7, 8, 9)$. Consider the number 7601. It can be pruned, because of the equivalent number 6701.

Note. The only distinction from absent digits 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 in the example we should consider number 6789.

2.6.3 Same response for several turns.

Consider sequence of turns $(t_1, r_1) \cdots (t_n, r_n)$. Split all of the 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. Consider sequence of 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 alternative form $(T_{0.1}, 0.1)(T_{0.2}, 0.2)$.

Lemma 9 *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 it's exist transformation ψ such that all sets of numbers $T_{b.c}$ with same responses, transform to itself. $\forall b, \forall c \ \ \psi(T_{b.c}) = T_{b.c}$ and number p transforms 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 9 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 9, it's possible to reduce cardinality of sets of third turns when we got the same response on first two turns.

3 Algorithms.

3.1 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. 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.

3.2 Heuristic algorithms.

Heuristic algorithms not always give exact estimation, but they are much faster than the exact algorithm. We will successively narrow usage of them, gradually moving to the exact algorithm. For searching of minimal amount of numbers we'll use crush algorithm. This algorithm was created by author of this paper. For minimizing of average game length we'll use Larmouth's algorithm. Description of algorithm could be found in the internet. Each of this heuristic algorithms could be used for both optimization criteria but crush algorithm gives better results for searching minimal amount of numbers, while Larmouth's algorithm is better for average game length.

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 in descending order, that is $n_1(t) \geq n_2(t) \geq \dots$

Suppose that for 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.

Larmouth's algorithm. 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 $n_i(t) = |S_i(t)|$.

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

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

3.3 Notation of algorithms.

Further we'll use next names of algorithms.

Minimal amount of numbers.

bulls and cows

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.

mastermind

1. crush3 - crush algorithm is used for turn 3 only.
2. exact - only exact algorithm is used.

Minimal average game length.

bulls and cows

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.

mastermind

1. crush3 - Larmouth algorithm is used for turn 3 only.
2. exact - only exact algorithm is used.

3.4 Extension for counting 6-movers, 5-movers etc.

Consider minimizing amount of numbers for bulls and cows game. Below we'll build trees for different algorithms. Quite often we can see that amount of 7-movers equals to zero. In this case it's possible to find turn which minimizes amount of 6-movers. If we can find turn that minimizes amount of 6-movers to zero, then we minimize amount of 5-movers etc. Search algorithm can minimize not only 7-movers but 6-movers, 5-movers etc. It uses additional parameter *depthCount* which means what do we need to count: 7-movers, 6-movers...

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

Note. The same is done for mastermind game.

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, building trees and count statistics of them.

4.1 Windows application - bcw.

Helper routines.

- Running of crush35, crush45, avg35, avg45 algorithms for bulls and cows.
- Running all of algorithms for mastermind.
- Create sets of equivalence for turns 1-3 for bulls-cows and for turns 1-2 for mastermind.
- Building trees and storing them to text files.
- Loading tree from file, calculation of its statistics and creation tree string for javascript.

4.2 Javascript application.

Online game using web application, building trees and count statistics.

- Building html file for tree browsing using jQuery.
- Loading tree and counting of its statistics.
- Game with man using storing tree.

4.3 The mechanism of the tasks [only bulls and cows].

For problems which take a lot of time special mechanism is created. At the beginning we should create file jobs.txt. Each line of this file has one problem with 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 needs to be solved. Different processes take tasks from file and change keyword *solve* to another keyword *taken* to show to other processes that this problem is already taken. Next the sequence of turns with responses is written. Finally one can see three parameters of algorithm.

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

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

- *mde* - defines from which turn program heuristic algorithm is used: 2 - from third, 3 - from fourth... For penultimate and last turn the exact algorithm is always used.

- 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's better to 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. So it's better to sort problems by decreasing of time of calculation.

4.4 Problems executor [only bulls and cows].

The console application uses for jobs which placed in jobs.txt file.

4.5 Windows service [only bulls and cows].

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 other programs.

5 Results.

5.1 Minimal amount of numbers.

5.1.1 Bulls and cows.

Crush35 and crush45 algorithms. Since crush algorithm is much faster then the exact one at first we run crush35 algorithm. Then we run crush45 algorithm only for cases where number of 7-movers 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 7-movers is showed below.

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. It means that the algorithm solve all secret numbers using up to seven turns.

Transition from one algorithm to another. Lets split of solving by 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 crush5 and the exact algorithm takes a long time. So split solution 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 find best turn t_1 for very first turn (0123, 0.1).

2. Consider all of the responses r_1 such that amount of 7-movers 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 7-movers using next algorithm $e_1(r_1) = \Phi_7^{next}[(0123, 0.1)(t_1, r_1)]$ and use as upper estimate β result 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 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 estimations 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 4.3).

Note. We can use table for turns (0123, 0.1)(t, 0.2) 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 7-movers 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 7-movers 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 7-movers 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 transition from crush45 algorithm to crush5.

Crush5 algorithm.

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 for three of them amount of 7-movers is 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 the 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 7-movers. After the first turn (0123, 0.1) the estimate is $E_1 = 46$. And after the first turn (0123, 0.2) the estimate is $E_2 = 8$.

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

The best turn after (0123, 0.2) (steps 7-8). Consider the second turn 1234. Only for responses 0.1 and 0.2 amount of 7-movers 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 7-movers equals 4. From table we can see that we don't need to consider all recent numbers because amount of 7-movers for them is always greater or equal then four. Thus the best second turn for crush5 algorithm is 1234, and amount of 7-movers equals 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.

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.

turns one and two	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

turns one and two	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

Crush5 algorithm results. After the first turn (0123, 0.1) the best second turn is 1245 and amount of 7-movers is 46. After the first turn (0123, 0.2) the best second turn is 1234 and amount of 7-movers is 4.

The exact algorithm.

Estimations for the best turns (steps 1-5). Consider the best number for first turn (0123, 0.1), which we got by crush5 algorithm. It's number 1245. We can get several responses after it, but only for three of them amount of 7-movers is greater than zero. For this cases run exact algorithm.

turn 1	turn 2	crush5	exact
0123 0.1	1245 0.1	38	38
0123 0.1	1245 0.2	7	7
0123 0.1	1245 0.0	1	1
total		46	46

Analogously, if first turn is (0123, 0.2), then the best turn is 1234.

turn 1	turn 2	crush5	exact
0123 0.2	1234 0.1	3	3
0123 0.2	1234 0.2	1	1
total		4	4

Now we got start estimations on maximum amount of 7-movers. After the first turn (0123, 0.1) the estimate is $E_1 = 46$. And after the first turn (0123, 0.2) the estimate is $E_2 = 4$.

Steps 6-9. The difference between exact and crush5 algorithms one can see in box frames.

t	(0123, 0.1)(t, 0.2)	(0123, 0.1)(t, 0.1)	$E_1 = 46$	(0123, 0.2)(t, 0.2)
1234	3	≥ 43	$E_1 \geq 46$	1
1034	4	≥ 42	$E_1 \geq 46$	$\times$
1204	7	≥ 39	$E_1 \geq 46$	$\times$
1245	7	38	$E_1 = 46$	$\times$
1435	7	≥ 39	$E_1 \geq 46$	$\times$
1045	9	36	?	$\times$
0234	10	≥ 36	$E_1 \geq 46$	$\times$
1456	10	≥ 36	$E_1 \geq 46$	$\times$
0245	15	≥ 31	$E_1 \geq 46$	$\times$
0214	16	≥ 30	$E_1 \geq 46$	$\times$
0134	18	≥ 28	$E_1 \geq 46$	$\times$
0456	30	≥ 16	$E_1 \geq 46$	$\times$
0124	38	≥ 8	$E_1 \geq 46$	$\times$
0145	38	≥ 8	$E_1 \geq 46$	$\times$
4567	≥ 46	$\times$	$E_1 \geq 46$	$\times$

We need to explore number 1045. Consider all of the answers r for which estimation of sequence (0123, 0.1)(1045, r) by crush5 algorithm is greater than zero. We got three answers, for two of them 0.1 and 0.2 we already have estimations (see table below). Now count estimate of sequence (0123, 0.1)(1045, 0.0) by the exact algorithm. We can see that number 1045 is also the best turn, after first turn (0123, 0.1). So the exact algorithm gives the same estimate with crush5.

turn 1	turn 2	crush5	exact
0123 0.1	1045 0.1	≥ 37	36
0123 0.1	1045 0.2	9	9
0123 0.1	1045 0.0	1	1
total		≥ 47	46

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.

turns one and two	exact
(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

turns one and two	exact
(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

Comparison table of different algorithms.

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

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

turn 1	7-movers	6-movers	5-movers	4-movers
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

5.1.2 Mastermind.

Mastermind game is simpler than bulls and cows. So we can do search for crush3 and exact algorithms for all possible first turns. The result table with amount of 5-movers is showed below. Dash symbol means that it's not possible to solve all secret numbers for up to five turns for this first turn.

turn 1	crush3	exact
0000	-	-
0001	-	695
0011	651	608
0012	569	539
0123	-	583

So we can see that very first turn 0012 is best turn. Minimal amount of 5-movers equals to 539. The summary table for amount of 5-movers after first turn for different responses is showed below.

turn 1	crush3	exact
0012 0.1	165	159
0012 1.1	128	121
0012 0.2	124	116
0012 1.0	88	80
0012 2.0	31	30
0012 1.2	19	19
0012 0.0	12	12
0012 0.3	1	1
0012 2.1	1	1
0012 0.4	0	-
0012 1.3	0	-
0012 2.2	0	-
0012 3.0	0	-
0012 4.0	0	-
total	569	539

5.2 Minimizing average game length.

5.2.1 Bulls and cows.

The problem of minimizing average game length is much harder than 7-movers 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

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

5.2.2 Mastermind.

Maximum five turns. Mastermind game is simpler than bulls and cows. So we can do search for avg3 and exact algorithms for all possible first turns if we can do maximum five turns. The result table is showed below. Dash symbol means that it's not possible to solve all secret numbers for up to five turns for such first turn.

turn 1	avg3	exact
0000	-	-
0001	-	5808
0011	5716	5702
0012	5638	5626
0123	-	5676

So we can see that very first turn 0012 is best turn. So we got first optimal algorithm with minimal average game length if it's possible to do only up to five turns. The summary table for game length after first turn for different responses is below.

turn 1	avg3	exact
0012 0.1	1253	1251
0012 1.1	1034	1029
0012 0.2	995	994
0012 1.0	804	800
0012 2.0	445	445
0012 1.2	347	347
0012 0.0	327	327
0012 0.3	171	171
0012 2.1	154	154
0012 3.0	76	76
0012 2.2	15	15
0012 1.3	11	11
0012 0.4	5	5
0012 4.0	1	1
total	5638	5626

Any number of turns. If we can do any number of turns then the best average game length is 5625/1296=4.03 (see [3]). Now we have algorithm with game length 5626/1296, the best turn is 0012. So we need to reduce estimate by one. Suppose that the best turn for optimal algorithm is still the same. We can expect that estimation could be reduced for the answers with longest average game length. From upper table for avg3 and exact algorithms we can see that such responses are 0.1, 1.1, 0.2 and 1.0. Run search for the first turns (0012, 0.1), (0012, 1.1), (0012, 0.2), (0012, 1.0). For first turn (0012, 1.1) the estimation became lower 1028/1296. So we have achieved minimal average game length 5625/1296. It's interesting that in this case for new and old algorithms the first and the second turns are the same.

6 Tree building.

6.1 Tree building on c++.

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

6.1.1 Minimal amount of numbers.

Bulls and cows. During of tree building we'll minimize not only 7-movers. If amount of 7-movers for some node equals to zero we'll minimize 6-movers. If exists turn which has amount of 6-movers equals to zero, then we'll minimize 5-movers etc. To minimize 6-movers, 5-movers the exact algorithm is always

used. To minimize 7-movers for first turns 0123 0.1 and 0123 0.2, we'll use the exact algorithm.

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

Mastermind. Similar for mastermind game if number of 5-movers equals to zero then we minimize amount of 4-movers etc. In contrast to bulls and cows where we used first three turns here only first two is used.

6.1.2 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. The same with minimal amount of numbers we use preliminary counted first three turns for bulls-cows and first two turns for mastermind.

6.2 Tree building on javascript.

For online game, gathering statistics and build html trees we use program on javascript. To set tree node we need its turn and children pointers for all possible answers except answer 4.0. Javascript language has no pointers. For every tree node we set identifier *id*. Instead of children pointers we use identifiers of them. For every node we set recent numbers array. Tree building algorithm automatically calculates children identifiers using recent numbers and turn of node.

Consider nodes for which one of recent numbers splits recent numbers set by subsets with maximum size 1. For this special nodes turn is not passed, because it can be calculated. Nodes with one or two recent numbers automatically satisfies this condition. Candidates to special nodes can be recent numbers sets which have size ≤ 14 (14 - number of different answers). Below you can see the table with special nodes for all optimal algorithms.

algorithm	total nodes	special nodes	recent nodes %
crushBullsCows	5269	4859	410 8%
avgBullsCows	5117	4711	406 8%
crushMastermind	1349	1259	90 7%
avgMastermind5	1319	1235	84 6%
avgMastermind	1316	1234	82 6%

Additionally it's possible to not pass turn for nodes which can be split on subsets with size 1 not only by turns from recent numbers but any turn. There

are about 1-2% of such nodes. This idea is slowdown javascript tree building so it's not used.

The whole tree is given by one string, which is list of recent 13% node turns. String consists of 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. So we use 92-th system of numeration. The symbols which have ASCII codes greater than 127 are not used because they are differently recognized by different browsers. The turn is a number from 0 to 5039, which is lower than 92·92, so to pass turn we need two symbols or two bytes.

Tree node processing. Algorithm is working such that recent numbers array S is set by parent. So recent numbers is set before node processing. If S is special one, then we calculate turn of node. Otherwise turn t is taken from tree string. Split recent numbers set S on subsets with same answers $S_{b_1c_1} \dots S_{b_nc_n}$ for turn t (except answer 4.0). Set number $5b + c$ for answer $(b\ c)$ to order sets $S_{b_ic_i}$. Now go over sets $S_{b_ic_i}$ using order. We add new nodes to the end of tree with recent numbers sets $S_{b_ic_i}$. New items are children of current node, for answers $(b_i\ c_i)$.

Tree building. At the beginning create tree with root node only. The turn for this node is the first turn of tree string. Recent numbers set is all numbers from Ω. After proceeding root node we get some new nodes which need to process. The algorithm is working until some of the nodes is unprocessed. Let's take number of non special nodes for which we need pass best turn and count string length for all algorithms. We can see that we need just $(410 + 406 + 90 + 84 + 82) \times 2 = 2144$ bytes.

7 Edition history.

Edition 3.1 30 January 2017

- Changed tree building algorithm for javascript.

Edition 3.0 2 November 2014

- Bulls and cows game is solved for minimize amount of turns criterion.
- Solution of mastermind game for two optimization criteria is added.
- Tree building and tree statistics is added for mastermind game.
- Cutoffs of penultimate layer is added.
- Equivalent splits are added.
- Hashing is added.
- Html tree building and tree statistics moved to javascript.

Edition 2.1 5 November 2013 (small changes)

Edition 2.0 1 November 2011

- Article was strongly extended and supplemented.
- Article was translated from Russian to English.
- The site of project was created.

Edition 1.0 25 December 2008 (only in Russian)

References

[1] John Francis, Strategies for playing MOO, or Bulls and Cows.
`http://slovesnov.users.sourceforge.net/bullscows/bulls_and_cows.pdf`
Original URL `http://www.jfwaf.com/Bulls%20and%20Cows.pdf`

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

[3] Kenji Koyama, Tony W. Lai, An Optimal Mastermind Strategy. Journal of Recreational Mathematics, 1993.