KAUNAS UNIVERSITY OF TECHNOLOGY
INFORMATICS FACULTY
SOFTWARE ENGINEERING DEPARTMENT

Ramūnas Zavistanavičius

Nonogram solving algorithms analysis and implementation for augmented reality system

Master thesis

Advisor
Dr. Andrej Ušaniov

Kaunas, 2012

KAUNAS UNIVERSITY OF TECHNOLOGY
INFORMATICS FACULTY
SOFTWARE ENGINEERING DEPARTMENT

Ramūnas Zavistanavičius

Nonogram solving algorithms analysis and implementation for augmented reality system

Master thesis

Reviewer
Dr. Sigitas Drąsutis
2012-05-28

Advisor
Dr. Andrej Ušaniov
2012-05-28

Author
IFM-0/2 gr. stud.
Ramūnas Zavistanavičius
2012-05-28

Kaunas, 2012

CONTENTS

FIGURES

CHARTS

TABLES

Santrauka

Papildyta realybė

Papildyta realybė yra inovatyvus kompiuterinės grafikos ir realaus pasaulio vaizdų panaudojimas, sukuriantis naują vaizdą. Papildyta realybė sklandžiai integruoja technologijas realiame pasaulyje, leisdama natūraliai naudotis naujausiomis technologijomis.

Nonogramos

Nonogramos, dar žinomos kaip Japoniški gryžiažodžiai, yra loginis galvosūkis. Norint išspręsti šį galvosūkį lentelės langeliai turi būti nuspalvinti arba palikti tušti priklausomai nuo skaičių, esančių lentelės šone ir viršuje, taip atskleidžiant paslėptą paveikslėlį. Šio tipo galvosūkiuose skaičiai parodo kiek vientisų užspalvintų langelių yra duotoje eilutėje arba stulpelyje.

Darbo tikslas

Šio darbo tikslas yra suprogramuoti ir palyginti nonogramų sprendimo algoritmus ir nustatyti, kuris iš jų yra tinkamesnis naudoti išmaniuosiuose telefonuose su Android operacine sistema.

Abstract

Augmented reality

Augmented reality (AR) is an innovative use of computer graphics in combination with real world data to create a new kind of video image. AR seamlessly integrates technology with the real world, allowing for a naturally enhanced computing.

Nonogram

Nonograms, also known as Paint by Numbers or Griddlers are logic puzzles in which cells in a grid have to be colored or left blank according to numbers given at the side of the grid to reveal a hidden picture. In this puzzle type, the numbers measure how many unbroken lines of filled-in squares there are in any given row or column.

Aim

Aim of this project is to implement and compare nonogram solution algorithms and find out which algorithm is suitable for mobile device running Android operating system.

1. Introduction

1.2. Purpose

1.2.1. Problem

The purpose of this work is implement nonogram solving algorithms and compare speed and memory usage. Algorithms have to solve puzzles in reasonable speed. According to Jakob Nielsen [1], one second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Ten seconds is about the limit for keeping the user's attention focused on the dialog. For augmented reality this number is even smaller. Also we have to keep in mind memory usage, because there are limited resources on mobile devices.

1.2.2. Nonograms

Nonograms are a popular kind of puzzle whose name varies from country to country, including Paint by Numbers and Griddlers. The goal is to fill cells of a grid in a way that contiguous blocks of the same color satisfy the clues, or restrictions, of each line or column.

According to Wikipedia [2], these kind of puzzles were created in 1987 by Non Ishida, a Japanese graphics editor, and Tetsuya Nishio, a professional Japanese puzzler, at the same time and with no relation whatsoever. Soon after, nonograms started appearing in Japanese puzzle magazines and later as electronic games. Today, magazines with nonogram puzzles are published in several countries and are available as electronic games in a variety of platforms.

Ueda e Negao prove in [3] that the nonogram problem is NP-Complete.

1.2.2.1. Black and white Nonograms

In black and white nonograms the clues indicate the sequence of contiguous blocks of cells to be filled (e.g. the clue 3,1,2 indicates that there is a block of 3 contiguous cells, followed by a sequence of one or more empty cells, then a block of one cell filled, followed by another sequence of one or more empty cells, finally followed by a sequence of two filled cells in that row or column). Figure 1 shows an example of a black and white nonogram (unsolved, to the left, solved, to the right).

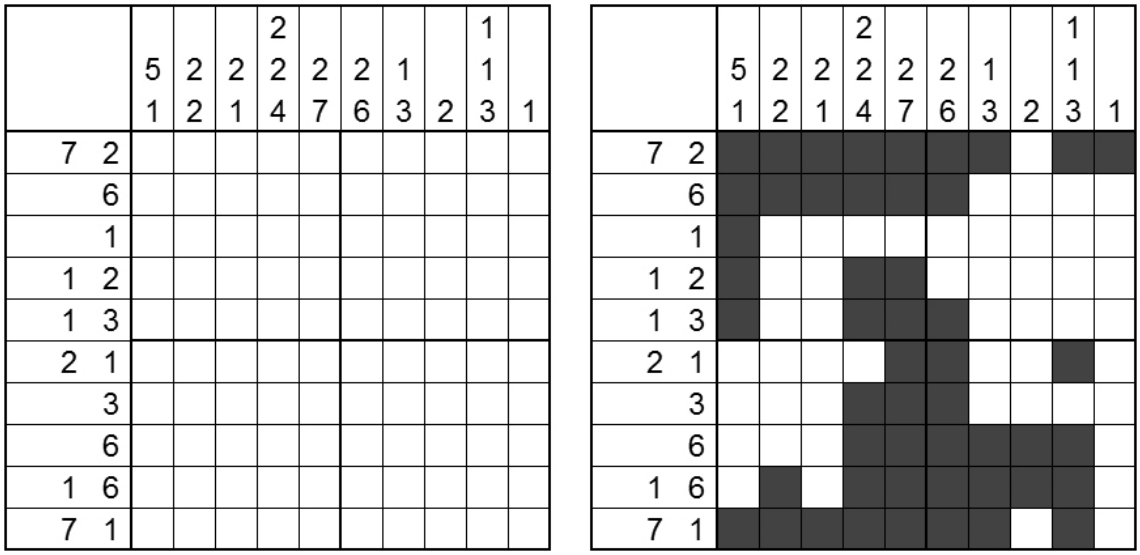


Figure 1 Black and white nonogram example (unsolved: left, solved: right)

Known approaches to solving black and white nonograms are the depth-first search (bruteforce) one, the iterative one, the ILP one by Bosch [4] and a genetic algorithm by Wouter Wiggers [5].

2. Analysis

2.1. Nonograms

In the previous chapter a brief description of nonograms was presented. In this one a more detailed explanation about nonograms is shown.

Nonograms are a popular kind of puzzle, whose name varies from country to country, including Paint by Numbers and Griddlers. The goal is to fill cells of a grid in a way that contiguous blocks of the same color satisfy the clues, or restrictions, of each line or column.

According to Wikipedia [2], this kind of puzzle was created in 1987 by Non Ishida, a Japanese graphics editor, and Tetsuya Nishio, a professional Japanese puzzler, at the same time and with no relation whatsoever. Soon after, nonograms started appearing in Japanese puzzle magazines and later as electronic games. Today, magazines with nonogram puzzles are published in several countries and are available as electronic games in a variety of platforms.

The most common nonograms are black and white, but they exist also in colors. In fact, black and white nonograms are a specialization of colored nonograms, i.e., are two colored nonograms.

Also there is a different kind of nonogram - called triddlers - in which cells are triangles. In this kind of puzzles we have three sets of clues instead of only two. These puzzles can also exist in multiple colors.

Ueda e Negao proves in [3] that the nonogram problem is NP-Complete.

2.2. Black and white Nonograms

In black and white nonograms the clues indicate the sequence of contiguous blocks of cells to be filled (e.g. the clue 3, 1, 2 indicates that there is a block of 3 contiguous cells, followed by a sequence of one or more empty cells, then a block of one cell filled, followed by another sequence of one or more empty cells, finally followed by a sequence of two filled cells in that row or column). Figure 1 shows an example of a black and white nonogram (unsolved, to the left, solved, to the right).

In order to solve this kind of puzzle it is necessary to determine which cells will be filled (black) and which will be empty (white). Determining which cells will be empty is as important as determining which will be filled because the former will help

delimiting the solutions for the blocks of each line or column. Simpler puzzles, like the one shown in figure 2.10, can usually be solved by applying the following methods to each line at a time.

Known approaches to solving black and white nonograms are the depth-first search (brute force) one, the iterative one, the ILP one by Bosch [4] and a genetic algorithm by Wouter Wiggers [5].

Simpler puzzles, like the one shown in Figure 2, can usually be solved by applying the following methods to each line at a time.

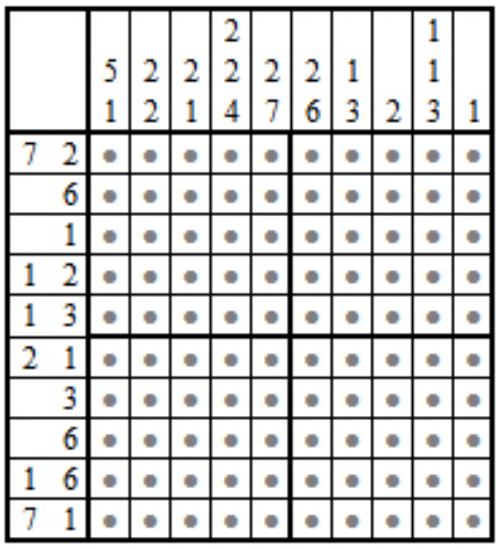


Figure 2 Black and white nonogram

2.2.1. Simple boxes

At the beginning of the solution, when there are no filled cells, for each block $b_i \in \{b_1,...,b_B$ in each row, the space available $S(b_i)$ for it is determined, assuming that the remaining blocks are moved closer to the extremities of the grid as possible (previous blocks to the left and subsequent block to the right). b_i represents a set of filled cells in sequence (vector). The value for $S(b_i)$ can be calculated using equation

$$S(b_i) = L - B + 1 - \sum_{k \neq i}^{B} T(b_k)$$

L represents the size of the line, B represents the number of blocks on the line and $T(b_i)$ represents the size of b_i .

It is also possible to know for each block what is the potential first cell that can occupied through equation

$$b_i[1] = \{b_{i-1}[1] + T(b_{i-1}) + 1), \qquad i > 1$$

$b_i[1]$ is block's b_i first cell position in the grid.

Within this set of cells it is possible to determine which subset is actually filled by analyzing the extremities of the solution, i.e., sliding the block as far to the left as possible and then as far to the right as possible and checking which cells are common to both solutions. In this way, equation

$$T(s_i) = 2t(b_i) - S(b_i)$$

gives the size of this sub-block, where $T(s_i)$ is the size of the sub-block s_i that can be determined for block b_i

In the same way, it is possible to obtain the first cell (consequently the remaining) of this sub-block through equation

$$s_i[1] = b_i[1] + S(b_i) - T(b_i), \qquad T(s_i) > 0$$

$s_i[1]$ is the position of the first cell of sub-block s_i.

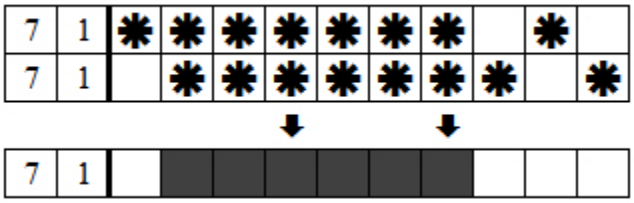


Figure 3 Example for the method Simple boxes in black and white nonograms

As an example, for the 10th line of the puzzle shown in Figure 2, L = 10, B = 2, $T(b_1) = 7$ and $T(b_2) = 1$. Therefore the space available for the first block is $S(b_1) = 10 - 2 + 1 - 1 = 8$ and $S(b_2) = 10 - 2 + 1 - 7 = 2$. The leftmost indexes each can occupy are $b_1[1] = 1$ and $b_2[1] = 1 + 7 + 1 = 9$.

As for the sub-blocks of cells that can be filled at this point, $T(s_1) = 2 \times 7 - 8 = 6$ and $T(s_2) = 2 \times 1 - 2 = 0$, i.e., it is not possible to fill, for now, any cell in respect to the second block, but it is possible to fill six cells with respect to the first one. It is yet to determine the starting cell of the first and second sub-blocks: $s_1[1] = 1 + 8 - 7 = 2$, i.e., it is possible to fill, at this point, cells 2 through 7 of that line.

Figure 3, from line 10 of the puzzle shown in Figure 2, exemplifies this method for a size 10 line with two blocks of sizes 7 and 1.

2.2.2. Simple spaces

This method consists of determining spaces by searching for cells that are out of range of any possible blocks of boxes. For example, considering a row of ten cells with boxes in the fourth and ninth cell and with clues of *3* and *1*, the block bound to the clue *3* will spread through the fourth cell and clue *1* will be at the ninth cell.

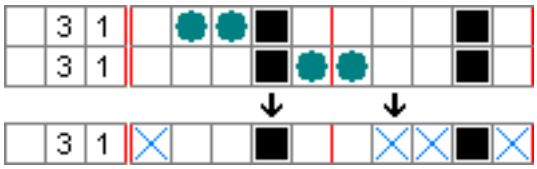


Figure 4 Simple spaces example on white and black nonogram

First, the clue *1* is complete and there will be a space at each side of the bound block.
Second, the clue *3* can only spread somewhere between the second cell and the sixth cell, because it always has to include the fourth cell; however, this may leave cells that may not be boxes in any case, i.e. the first and the seventh.
Note: In this example all blocks are accounted for; this is not always the case. The player must be careful for there may be clues or blocks that are not bound to each other yet.

2.2.3. Forcing

In this method, the significance of the spaces will be shown. A space placed somewhere in the middle of an uncompleted row may force a large block to one side or the other. Also, a gap that is too small for any possible block may be filled with spaces.

Figure 5 Forcing spaces example on white and black nonogram

For example, considering a row of ten cells with spaces in the fifth and seventh cells and with clues of *3* and *2*:

- the clue of *3* would be forced to the left, because it could not fit anywhere else.
- the empty gap on the sixth cell is too small to accommodate clues like *2* or *3* and may be filled with spaces.

- finally, the clue of *2* will spread through the ninth cell according to method *Simple Boxes* above.

2.2.4. Glue

Sometimes, there is a box near the border that is not farther from the border than the length of the first clue. In this case, the first clue will spread through that box and will be forced outward from the border.

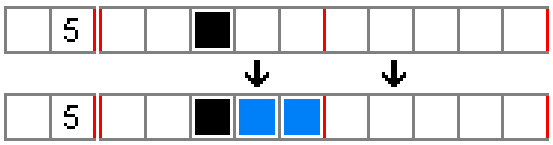


Figure 6 Glue example on white and black nonogram

For example, considering a row of ten cells with a box in the third cell and with a clue of *5*, the clue of *5* will spread through the third cell and will continue to the fifth cell because of the border.

Note: This method may also work in the middle of a row, further away from the borders.

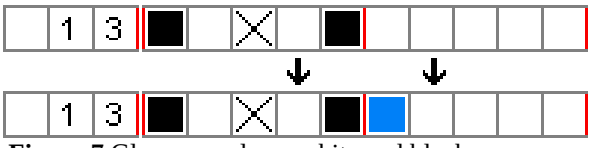


Figure 7 Glue example on white and black nonogram

- A space may act as a border, if the first clue is forced to the right of that space.
- The *first* clue may also be preceded by some other clues, if all the clues are already bound to the left of the forcing space.

2.2.5. Joining and splitting

Boxes closer to each other may be sometimes joined together into one block or split by a space into several blocks. When there are two blocks with an empty cell between, this cell:

- will be a space if joining the two blocks by a box would produce a too large block;
- will be a box if splitting the two blocks by a space would produce a too small block that does not have enough free cells remaining.

For example, considering a row of fifteen cells with boxes in the third, fourth, sixth, seventh, eleventh and thirteenth cell and with clues of *5*, *2* and *2*:

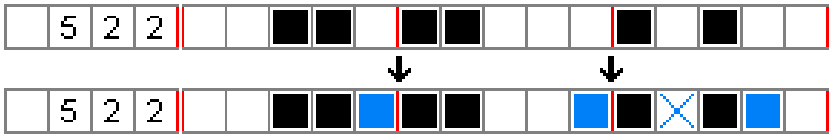


Figure 8 Joining and splitting example on white and black nonogram

- the clue of *5* will join the first two blocks by a box into one large block, because a space would produce a block of only 4 boxes that is not enough there;
- and the clues of *2* will split the last two blocks by a space, because a box would produce a block of 3 continuous boxes, which is not allowed there.
- *Note: The illustration picture also shows how the clues of* 2 *will be further completed. This is, however, not part of the* Joining and splitting *technique, but the* Glue *technique described above.*

2.2.6. Mercury

Mercury is a special case of *Simple spaces* technique. Its name comes from the way mercury pulls back from the sides of a container.

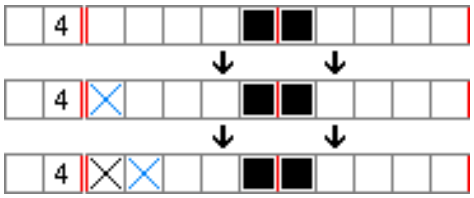


Figure 9 Mercury example on white and black nonogram

If there is a box in a row that is in the same distance from the border as the length of the first clue, the first cell will be a space. This is because the first clue would not fit to the left of the box. It will have to spread through that box, leaving the first cell behind. Furthermore, when the box is actually a block of more boxes to the right, there will be more spaces at the beginning of the row, determined by using this method several times.

2.2.7. Contradictions

Some more difficult puzzles may also require advanced reasoning. When all simple methods above are exhausted, searching for contradictions may help. It is wise to use a

pencil (or other color) for that in order to be able to undo the last changes. The procedure includes:

1) Trying an empty cell to be a box (or then a space).
2) Using all available methods to solve as much as possible.
3) If an error is found, the tried cell will not be the box for sure. It will be a space (or a box, if space was tried).

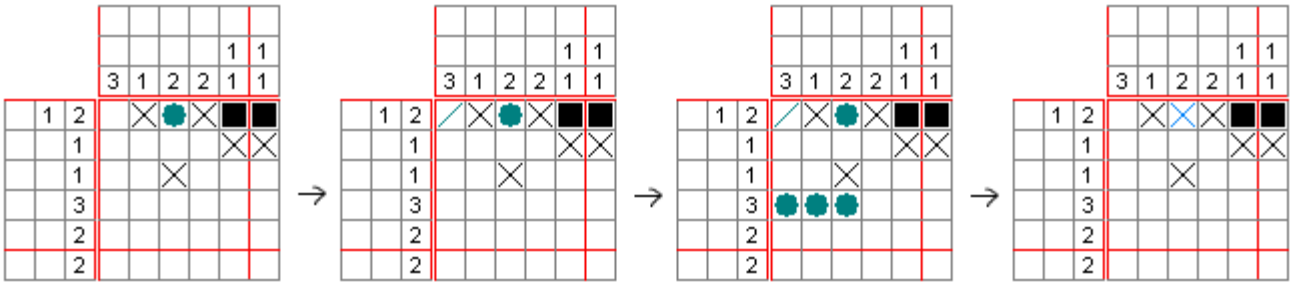


Figure 10 Contradiction example on white and black nonogram

In this example a box is tried in the first row, which leads to a space at the beginning of that row. The space then *forces* a box in the first column, which *glues* to a block of three boxes in the fourth row. However, that is wrong because the third column does not allow any boxes there, which leads to a conclusion that the tried cell must not be a box, so it must be a space.

The problem of this method is that there is no quick way to tell which empty cell to try first. Usually only a few cells lead to any progress, and the other cells lead to dead ends. Most worthy cells to start with may be:

- cells that have many non-empty neighbors;
- cells that are close to the borders or close to the blocks of spaces;
- cells that are within rows that consist of more non-empty cells.

2.3. Approaches to solving Nonograms

In the previous section was explained how simpler puzzles can be solved by looking at each line at a time and applying one or more methods to color cells or mark them as spaces. For more complex puzzles we can reach a state where we cannot fill more unknown cells by applying those methods. At that point we have to try and guess a value (color or space) for a cell and then reapply the aforementioned methods to try to

reach a solution or a contradiction. Eventually we will reach another state where another guess must be made to continue to try to solve the puzzle, and so on. If a contradiction is reached, then the value we chose for a determined cell is wrong. In black and white puzzles this means that the cell will have the opposite value (empty if the chosen value was filled, filled otherwise). These more complex puzzles are usually difficult to solve by a human.

2.3.1. Depth-first search (brute-force)

This approach tries all possible combinations for the set of blocks of each line. For example, for a size 10 line, belonging to a black and white nonogram, with two blocks of sizes 5 and 1, we would have 10 possibilities only for that line, as shown in Figure 11.

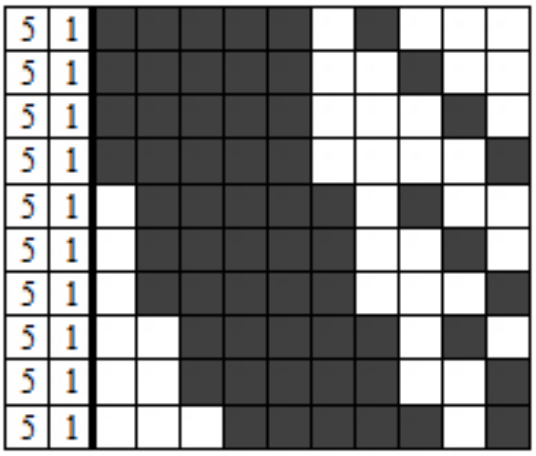


Figure 11 Depth-first search all possibilities for a row

2.3.2. Constraint programming

Constraint programming is a programming paradigm wherein relations between variables are stated in the form of constraints. Constraints differ from the common primitives of imperative programming languages in that they do not specify a step or sequence of steps to execute, but rather the properties of a solution to be found. More information about constraint programming will be in Analysis section.

2.4. Constraints

A constraint is simply a logical relation among several unknowns (or variables), each taking a value in a given domain. A constraint thus restricts the possible values that

variables can take, it represents some partial information about the variables of interest. For instance, the circle is inside the square relates two objects without precisely specifying their positions, i.e., their coordinates. Now, one may move the square or the circle and he or she is still able to maintain the relation between these two objects. Also, one may want to add another object, say a triangle, and to introduce another constraint, say the square is to the left of the triangle. From the user (human) point of view, everything remains absolutely transparent. Constraints naturally meet several properties:

1) constraints may specify partial information, i.e. constraint need not uniquely specify the values of its variables;
2) constraints are non-directional, typically a constraint on (say) two variables X; Y can be used to infer a constraint on X given a constraint on Y and vice versa;
3) constraints are declarative, i.e. they specify what relationship must hold without specifying a computational procedure to enforce that relationship;
4) constraints are additive, i.e. the order of imposition of constraints does not matter, all that matters at the end is that the conjunction of constraints is in effect;
5) constraints are rarely independent, typically constraints in the constraint store share variables.

Constraints arise naturally in most areas of human endeavor. The three angles of a triangle sum to 180 degrees, the sum of the currents floating into a node must equal zero, the position of the scroller in the window scrollbar must reflect the visible part of the underlying document, these are some examples of constraints which appear in the real world. Thus, constraints are a natural medium for people to express problems in many fields.

I use CHOCO constraint programming library, which helps me to implement algorithm efficiently.

2.4.1. Constraints programming

Constraint programming is the study of computational systems based on constraints. The idea of constraint programming is to solve problems by stating constraints (conditions, properties) which must be satisfied by the solution.

Work in this area can be tracked back to research in Artificial Intelligence and Computer Graphics in the sixties and seventies. Only in the last decade, however, has there emerged a growing realization that these ideas provide the basis for a powerful

approach to programming, modeling and problem solving and that different efforts to exploit these ideas can be united under a common conceptual and practical framework, constraint programming.

2.4.2. Modeling with Constraint programming

The formulation and the resolution of combinatorial problems are the two main goals of the constraint programming domain. This is an essential way to solve many interesting industrial problems such as scheduling, planning or design of timetables, puzzles. The main interest of constraint programming is to propose to the user to model a problem without being interested in the way the problem is solved.

2.4.3. The constraint satisfaction problem

Constraint programming allows solving combinatorial problems modeled by a Constraint Satisfaction Problem (CSP). Formally, a CSP is defined by a triplet (X, D, and C):

- **Variables**: $X = \{X_1, X_2, \ldots, X_n\}$ is the set of variables of the problem.
- **Domains**: D is a function which associates to each variable X_i its domain $D(X_i)$, i.e. the set of possible values that can be assigned to X_i . The domain of a variable is usually a finite set of integers: $D(X_i) \subset Z$ (integer variable). But a domain can also be continuous ($D(X_i) \subseteq R$ for a real variable) or made of discrete set values ($D(X_i) \subseteq P(Z)$ for a set variable).
- **Constraints:** $C = \{C_1, C_2, ..., C_m\}$ is the set of constraints. A constraint C_j is a relation defined on a subset $X^j = \{X_1^j, X_2^j, ..., X_{n^j}^j\} \subseteq X$ of variables which restricts the possible tuples of values $(v_1, ..., v_{n^j})$ for these variables: $(v_1, ..., v_{n^j}) \in C_j \cap (D(X_1^j) \times D(X_2^j) \times ... \times D(X_{n^j}^j))$.

  Such a relation can be defined explicitly (ex: $(X_1, X_2) \in \{(0,1),(1,0)\}$) or implicitly (ex: $X_1 + X_2 \leq 1$).

Solving a CSP is to find a tuple $v = (v_1, ..., v_n) \in D(X)$ on the set of variables which satisfies all the constraints:

$$v = (v_1, ..., v_{n^j}) \in C_j, \forall j \in \{1, ..., m\}.$$

For optimization problems, one needs to define an objective function $f : D(X) \to R$. An optimal solution is then a solution tuple of the CSP that minimizes (or maximizes) function *f*.

2.5. Depth-first search

Solving the Japanese nonogram using a depth first search algorithm is very straight forward. One takes the first row of the puzzle and generates all the different positions for that row. In Figure 12 this is done for the first row of that puzzle.

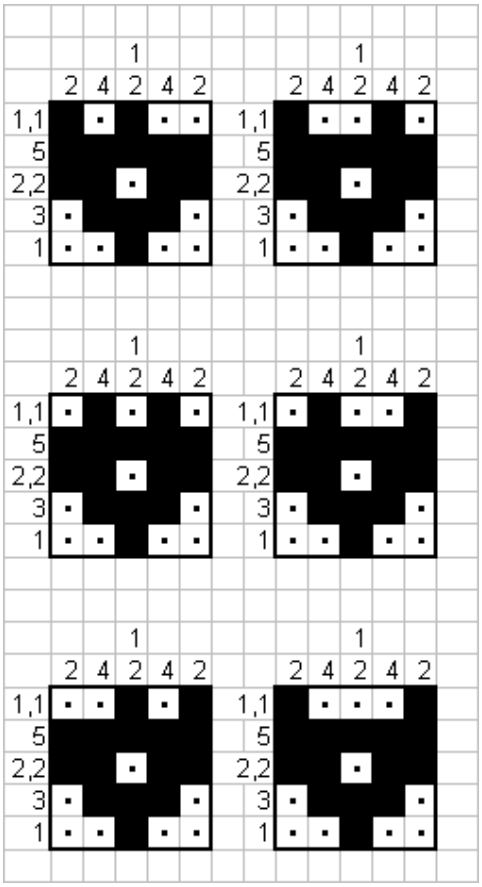


Figure 12 Results of Depth-first search

When all the possible positions for the first row have been generated the second row generates a new position and the process starts all over again until all possible solutions of the nonogram have been generated. It is called a depth first search because all the possible positions for the first row are generated first. The different rows in the puzzle together create a solution. Each different solution is then checked for correctness using the columns of the puzzle as a reference. When every column of the Japanese puzzle is correct the puzzle is solved. This is because each generated position of a row is already correct. In Figure 13 this process is shown for the different positions of the first row.

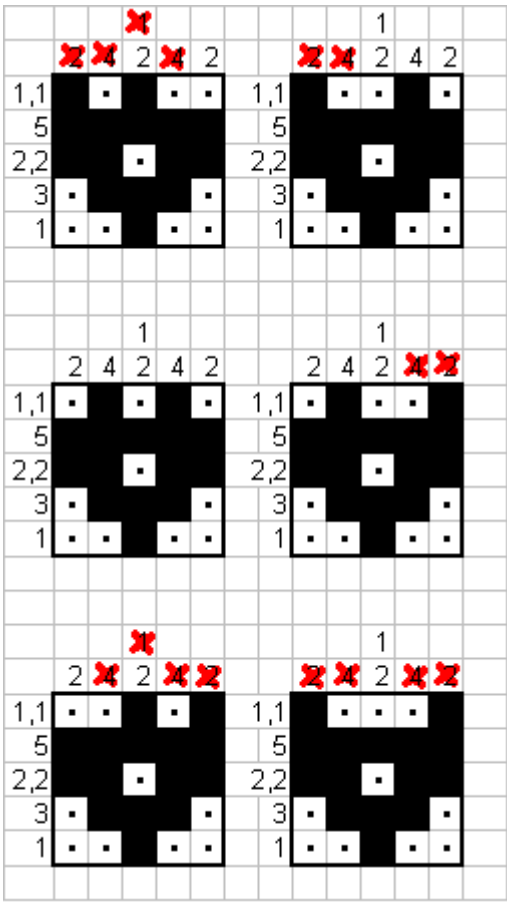


Figure 13 Solution`s verification

The third solution in Figure 13 is the correct one. The solution of the puzzle in Figure 13 is found quickly because the positions of the other rows were already correct. The depth first search algorithm searches for the correct solution of the puzzle and returns the correct one if it exists and the number of checks that were necessary to find it. A check is defined as the process of evaluating a proposed solution of the puzzle. Because the steps the depth first search algorithm takes are irreversible and the state space is finite (there are only a finite number of permutations of the rows) a solution will be found if it exists. Although the method of generating the state space during the depth first search and the evaluation function are both very fast the overall performance of this algorithm is very poor. This is because there are so many possible states that need to be checked.

2.6. Backtracking

The simplest algorithm for solving puzzles is backtracking, which traverses the search graph in a depth-first manner. It is often assumed that the variables are examined in a fixed ordering. The backtracking algorithm maintains and operates on a partial solution

that denotes a state in the algorithm's search space. Backtracking has three phases: a forward phase in which the next variable in the ordering is selected and called current; a phase in which current partial solution is extended by assigning a consistent value to the current variable, if one exists; and a backward phase in which, when no consistent value exists for the current variable, focus returns to the variable prior to the current variable.

Backtracking

Input: A constraint network *R* and an ordering of the variables $d = \{x_1,..., x_n\}$.

Output: Either a solution if one exists or a decision that the network is inconsistent.

1) (Initialize) *cur = 0*
2) (Step forward) If x_{cur} is the last variable the all variables have value assignments – exist with this solution. Otherwise *cur = cur + 1.* Set $D'_{cur} = D_{cur}$.
3) (Choose a value) Select a value $a \in D'_{cur}$ that is consistent with all previously instantiated variables:
   a) If $D'_{cur} = 0$ (x_{cur} is a dead-end), go to step 4.
   b) Select *a* from D'_{cur} and remove it from D'_{cur}.
   c) For each constraint defined on x_1 through x_{cur} test whether it is violated by $\vec{a}_{cur-1}$ and $x_{cur} = 0$. If it is, go to step 3a.
   d) Instantiate $x_{cur} = a$ and go to step 1.
4) (Backtrack step) If x_{cur} is the first variable, exit with "inconsistent". Otherwise, set $cur = cur - 1$. Go to step 2.

Figure 14 Backtracking algorithm

Figure 14 describes a basic backtracking algorithm. In addition to its fixed value domain D_i, each variable x_i maintains a mutable value domain D'_i such that $D'_i \subseteq D_i$. D'_i holds the subset of D_i that has not yet been examined under the current instantiation. Initially, all variables are uninstantiated.

We denote by $\vec{a}_i$ the subtuple of consecutive values $(a_1,..., a_i)$ for a given ordering of the variables $x_1,..., x_i$. We denote by $\vec{a}$ an arbitrary subtuple of values. Step 3c in algorithm (Figure 14) is implemented by performing consistency checks between

$x_{cur} = a$ and all past assignments $x_i = a_i$, $1 \le i < cur$. The algorithm tests whether the tuple $\vec{a}_{cur-1}$, if extended by $x_{cur} = a$ is consistent. If the constraints are binary, the algorithm tests whether the pairs $(x_i = a_i, x_{cur} = a)$ are allowed by the binary constraints $R_{i,cur}$. Because consistency checking is performed frequently, a count of the number of consistency checks is a common measure of the overall costs of the algorithm.

Backtracking usually suffers from thrashing, namely, rediscovering the same inconsistencies and same partial successes during search. Efficient cures for such behavior in all cases are unlikely, since the problem is NP hard. However, there are some simple heuristics that can provide convenient remedies in a large number of cases. Some of these heuristics involve limited exploration of the future search so as to increase the chance of a good current decision, while others involve a limited amount of learning, which entails exploiting information already collected during search.

2.7. Other methods

In this section other methods for solving nonograms, implemented and tested on personal computers, will be reviewed.

2.7.1. Genetic algorithms

A genetic algorithm uses biological-derived techniques such as inheritance, natural selection, recombination and mutation.

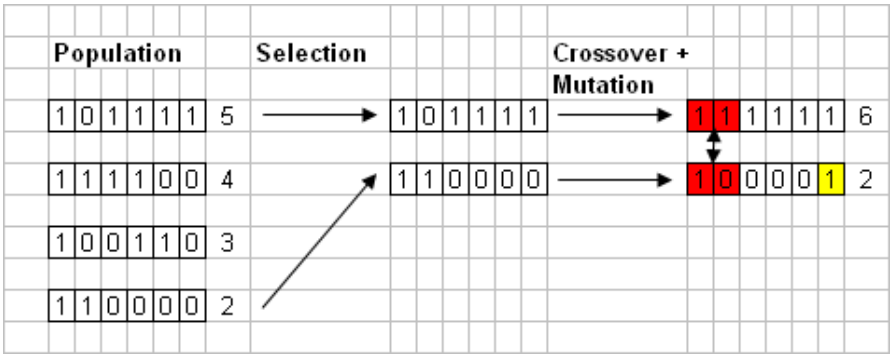


Figure 15 Example of genetic algorithm

The genetic algorithm that is used to solve nonograms works with the three operators: selection, crossover and mutation [6]. The functioning of theese three is shown in Figure 15. These operators work on the date sets of the algorithm, which are also called

chromosome. Another important part of the algorithm is the fitness function. This function calculates the fitness of a chromosome using the genes of the chromosome. The goal of a genetic algorithm is to optimize this fitness function and find individual that has the „best“ fitness value. More information about genetic algorithms can be found in [7].

The performance of the genetic algorithm was measured by everaging the number of avaluations of the first three succesful runs. The results of the performace tests are favoring the genetic algorithm. If we compare 10x10 sized nonogram, the depth-firsts search algorithms much more avaluations compared to genetic algorithm. However, the depth first search algorithm outperforms the genetic algorithm when solving small nonograms. Also there are situations when genetic algorithms get stuck. This is possible to avoid by detecting that the population of the genetic algorithm no longer evolves [5].

2.7.2. Backtracking modifications

There are modifications and heuristics for simple depth-first search backtracking algorithms, that helps to solve different nonograms faster.

Backjumping is one of the primary tools for redusing backtracking‘s unfortunate tendency to rediscover the same dead-ends [8]. We can sinesse rediscovering the same deadends by identifying the culprit variable responsible for the dead-end and then jumping back immediatly to reinstantiate the culprit variable, instead of repeatedly instantiating the chronologically previous variable. Identification of a coulprit variable in backtracking is based on the notion of conflict sets [8].

There also is the Gashnick‘s backjumping [9]. This type of backjumping records some information while generating next backtracking step and uses this information to determine the dead-end‘s culprit variable. The algorithm uses a marking technique whereby each variable maintais a pointer to the latest predesesor found incompatible with any of the variable‘s values.

These backtracking modifcations helps to solve some nongrams faster, but with other puzzles it can be even slower, because overall nonograms solution is NP complexity.

2.7.3. Results

Accordint to Jan Wolter survey [10], fastest algorithms are written in C programming language and tested on Unix operating system. In Table 1 is shown two fastest nonogram solving algoritms. Tests were done with 5000 nonogram, size of 30x30. Those two algorithms solved most of those puzzles in inpressive time – 1 second or less.

Table 1 Two fastest nonogram solving algorithms (30x30 puzzles)

Solver	**Percent solved in under a second**	**Percent solved in under 0.1 second**
Jan Wolter's pbnsolve Program	96.4%	81%
Kuang-che Wu‘s Naughty	94.9%	87%

3. Project

Nonogram Solver is an augmented reality system, which uses augmented reality to provide nonogram solution. All user has to do is point mobile device at nonogram, take picture and solution is displayed.

Nonogram solver can detect nonogram objects (numerals, grid), solve nonogram and show augmented solution for user (Figure 16).

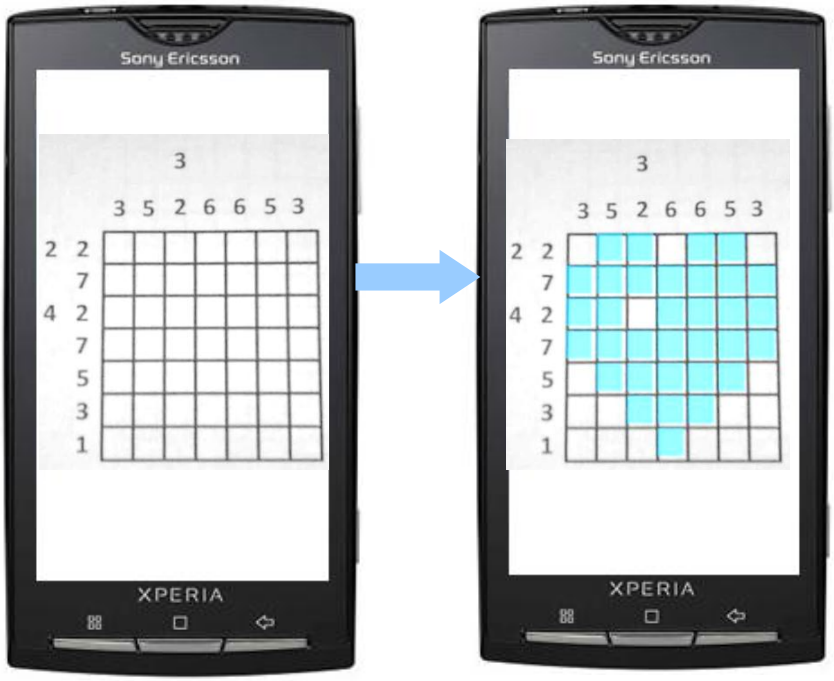


Figure 16 Principle of nonogram solver (left: unsolved nonogram, right: solved nonogram)

3.1. The purpose of the system

Mobile phones, which were not long ago "brick-like" devices limited to phone calls, have evolved into smart phones, with increased storage, communication and computational resources. After analyzing worldwide smart phones market, we concluded that smart phones with Android OS become more and more popular.

According to Gartner analysis [11] in Figure 17 Android OS made an impressive 1200% gain in devices shipped, and shot them all the way from 3.5% to 25.5% market share.

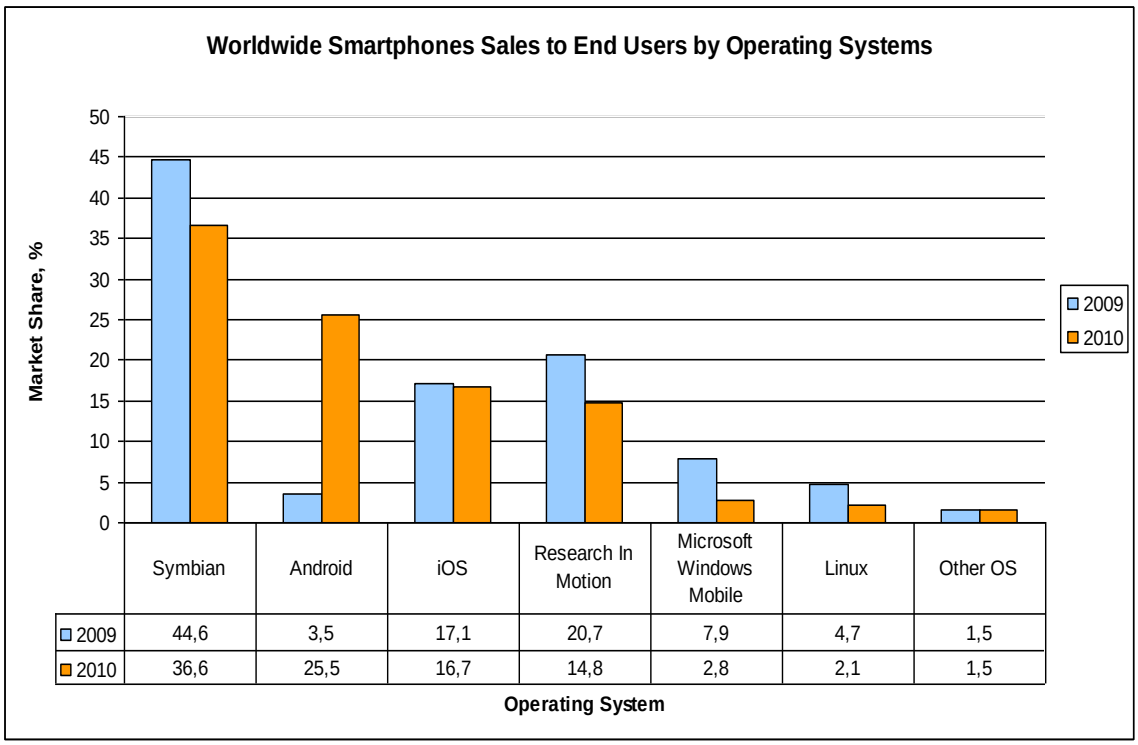


Figure 17 Worldwide smart phones sales in 2009-2010

In 2011 according to Millennial Android operating system reached 50 % of market share of smartphones operating systems (Figure 18).

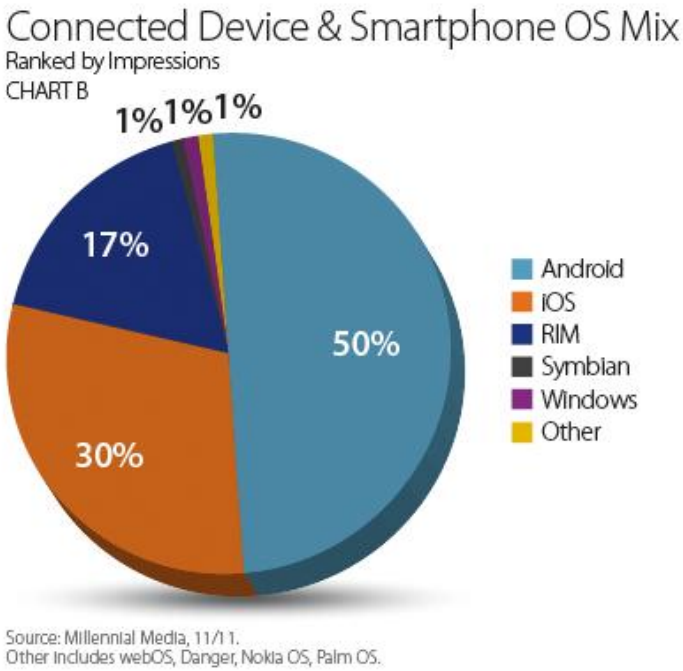
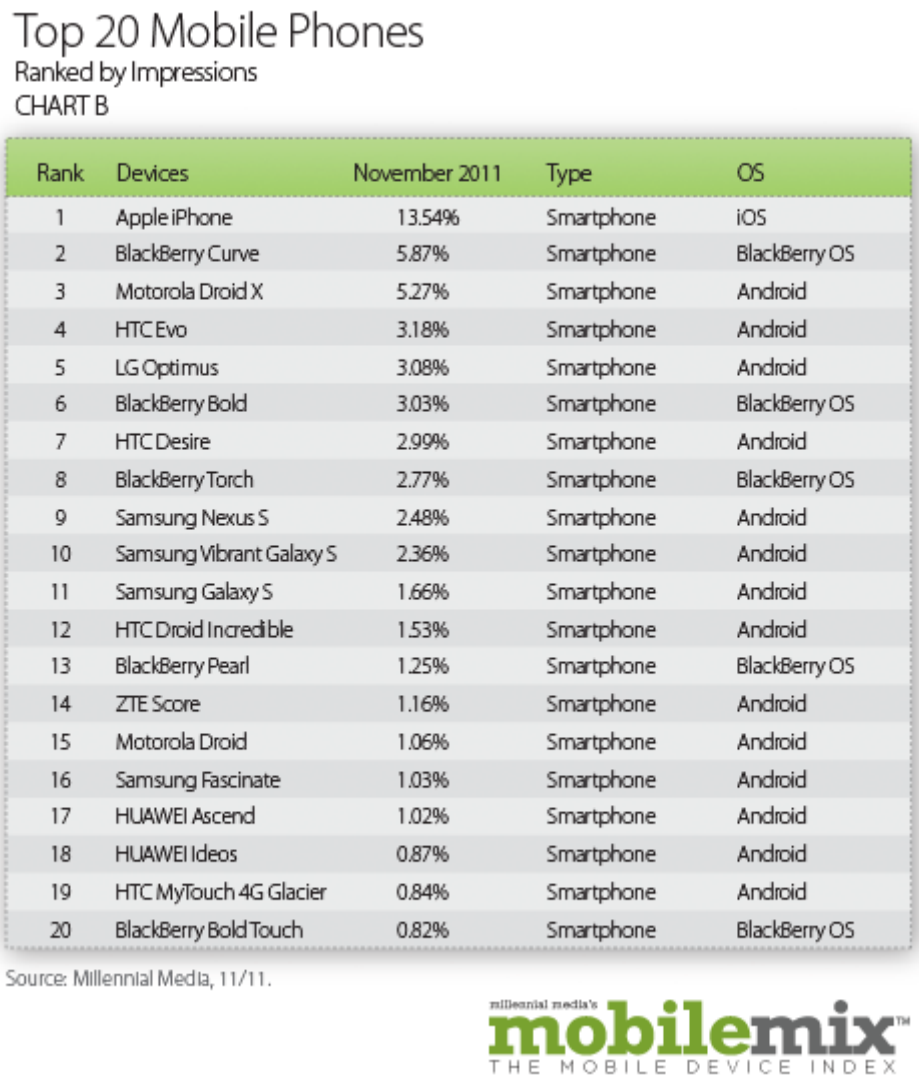


Figure 18 Market share of smartphones operating systems

Because of such rapid mobile phones evolution, augmented reality finds its way into smart phones market, transforming it into interactive personal digital assistants (PDA). Furthermore 14 out of 20 the most popular smartphones run on Android operating system (Figure 19).



Top 20 Mobile Phones
Ranked by Impressions
CHART B

Rank	Devices	November 2011	Type	OS
1	Apple iPhone	13.54%	Smartphone	iOS
2	BlackBerry Curve	5.87%	Smartphone	BlackBerry OS
3	Motorola Droid X	5.27%	Smartphone	Android
4	HTC Evo	3.18%	Smartphone	Android
5	LG Optimus	3.08%	Smartphone	Android
6	BlackBerry Bold	3.03%	Smartphone	BlackBerry OS
7	HTC Desire	2.99%	Smartphone	Android
8	BlackBerry Torch	2.77%	Smartphone	BlackBerry OS
9	Samsung Nexus S	2.48%	Smartphone	Android
10	Samsung Vibrant Galaxy S	2.36%	Smartphone	Android
11	Samsung Galaxy S	1.66%	Smartphone	Android
12	HTC Droid Incredible	1.53%	Smartphone	Android
13	BlackBerry Pearl	1.25%	Smartphone	BlackBerry OS
14	ZTE Score	1.16%	Smartphone	Android
15	Motorola Droid	1.06%	Smartphone	Android
16	Samsung Fascinate	1.03%	Smartphone	Android
17	HUAWEI Ascend	1.02%	Smartphone	Android
18	HUAWEI Ideos	0.87%	Smartphone	Android
19	HTC MyTouch 4G Glacier	0.84%	Smartphone	Android
20	BlackBerry Bold Touch	0.82%	Smartphone	BlackBerry OS

Source: Millennial Media, 11/11.

millennial media's mobilemix™
THE MOBILE DEVICE INDEX

Figure 19 The most popular smartphones

That is why “Nonogram solver” was created for smartphones with Android OS.

3.2. Similar nonograms products

Nonogram (Dmitry Mikhailenko) – simple application for solving given puzzle. Can randomly generate nonograms. Application cannot solve nonograms, only check answer.

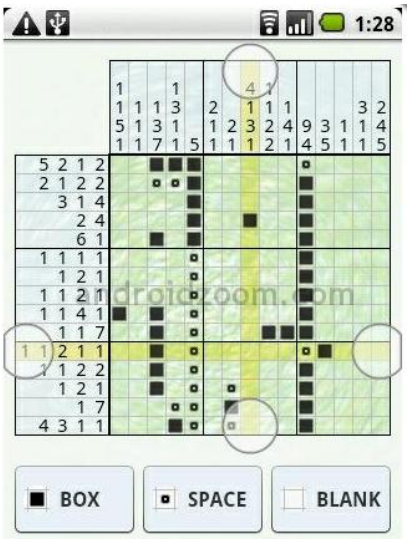


Figure 20 Dmitry Mikhailenko application‘s demonstration

Nonogram (Shcheglov Maksym) – application, which allows users to solve predefined nonograms. Nonogram size varies from 5x5 to 10x15, threre are leaderboard table, multiple undos, puzzles saving for later. Application cannot solve nonograms, only check answer.

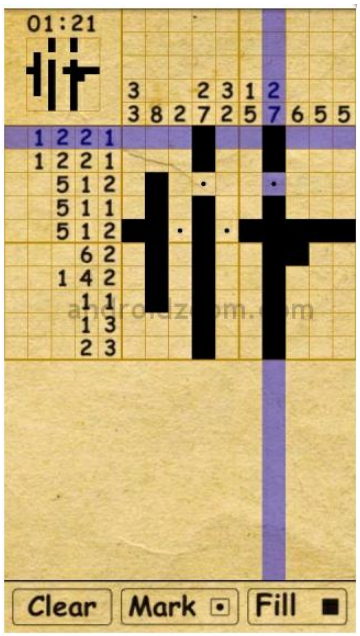


Figure 21 Shcheglov Maksym application's demonstration

Nonogram Nexus (Shai Shapira) - application, which allows users to solve predefined nonograms. Can randomly generate nonograms, maximus size is 999x999. pplication cannot solve nonograms, only check answer.

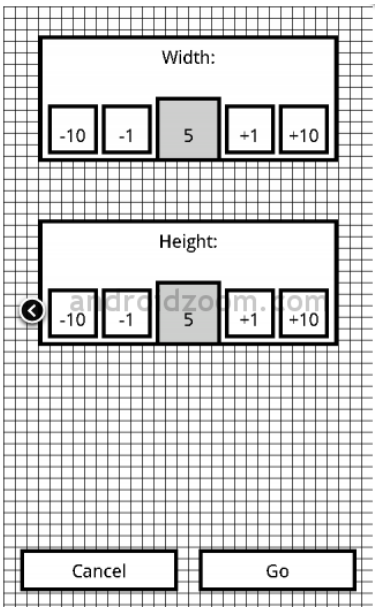


Figure 22 Shai Shapira application's demonstration

PIX Nonogram (RedRabbit Interactive) – application has predefined list of nonograms, which user can solve. This list gets regular updates. Application cannot solve nonograms, only check answer.

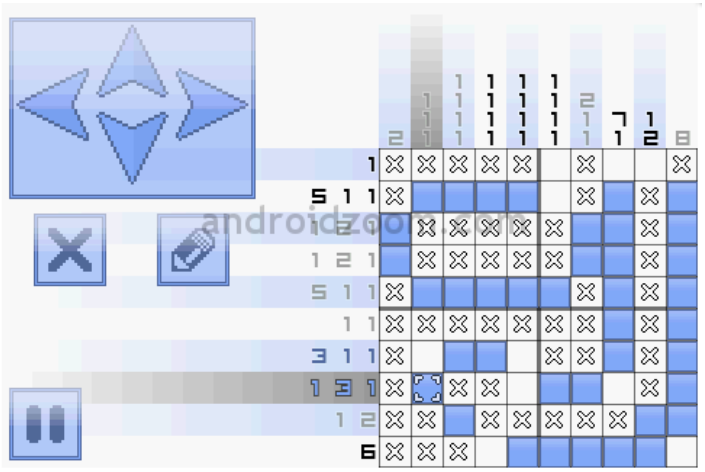


Figure 23 RedRabbit interactive application's demonstration

Nonogram 30x30 (cosotto) – has 100 predefined 30x30 size nonograms. Allows users to solve them and check answer. Application cannot solve nonograms, only check answer. Requires internet connection.

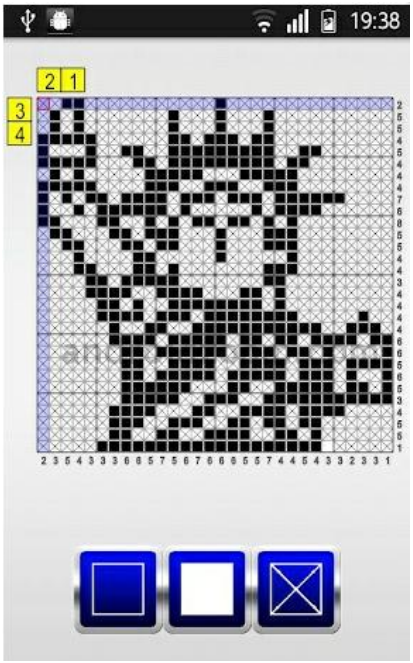


Figure 24 Cosotto application's demonstration

Pixler – Nonogram Puzzle (Melvin Apps) – application, having predefined set of nonogram. Size varies from 5x5 to 20x20. Has colored nonograms and can randomly generate solvable puzzles.

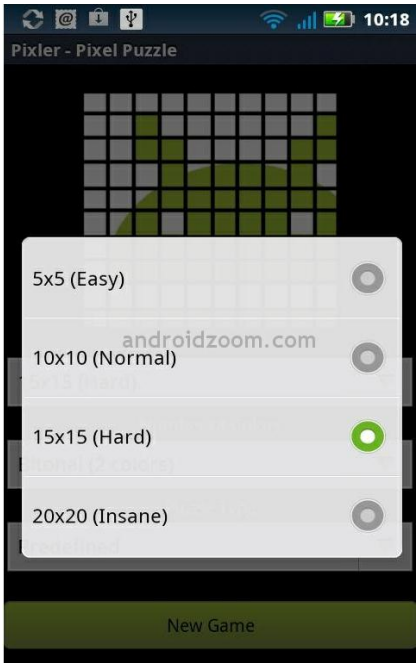


Figure 25 Melvin Apps application's demonstration

Table 2 Comparison of nonogram application

Author	Supported phones	Price	Size	Solve puzzle
Dmitry Mikhailenko	Most Android Phone	Free	0.54 MB	No
Shcheglov Maksym	Most Android Phone	Free	1.42 MB	No
Shai Shapira	Most Android Phone	Free	0.2 MB	No
RedRabbit Interactive	Most Android Phone	$1.99	0.83 MB	No
cosotto	Most Android Phone	Free	0.81 MB	No
Melvin Apps	Most Android Phone	$1.29	0.34 MB	No

3.3. Similar AR products

Layar - first AR system designed for Android OS. It displays real time digital information on top of reality in the camera screen of the mobile phone. While looking through the phone's camera lens, a user can see houses for sale, popular bars and shops, tourist information of the area, play a live game, etc.

Figure 26 Layar demonstration

WikiTude Drive - first augmented reality turn-by-turn navigation application for Android smart phones. Application uses phone's camera and GPS receiver in tandem, layering selected route over a live view of what's ahead of the car.

Figure 27 WikiTude Drive demonstration

TagWhat - basically a social networking application that uses augmented reality. It lets users to tag whatever they see in front of them using tag feature. People visiting those tagged places will see the details while pointing android phone to places in front.

Figure 28 TagWhat demonstration

Space InvadAR – a vision based game that uses AR. While pointing camera towards a high resolution image, application loads the game.

Figure 29 Space InvadAR demonstration

Sudoku grab – Sudoku puzzles solver. User just needs to point phone's camera to Sudoku puzzle and it gets solved. Solution is overlaid on real world Sudoku puzzle.

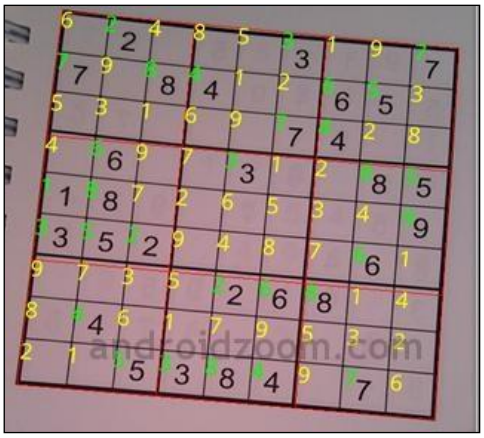


Figure 30 Sudoku Grab example

Table 3 Comparison of augmented reality application

	Invadar	**TagWhat**	**Wikitude Drive**	**Layar**	**Sudoku Grab**
Supported phones	HTC Desire, Nexus 1	Most of mobile devices with Android OS	HTC Motorola Droid Samsung galaxy, Huawei RBM2, Nexus 1	Most of mobile devices with Android OS	Most of mobile devices with Android OS
Price	$25	Free	Free	Free + paid layers	$1
Size	3 MB	0.6 MB	2 MB	2 MB	1 MB
Release date	2010 08 08	2010 09 13	2009 10 28	2009 10 29	201015

3.4. Nonogram solver implementation

3.4.1. Use cases

This chapter provides Nonogram Solver use cases' description.

System use cases are displayed in use case diagram (Figure 31)

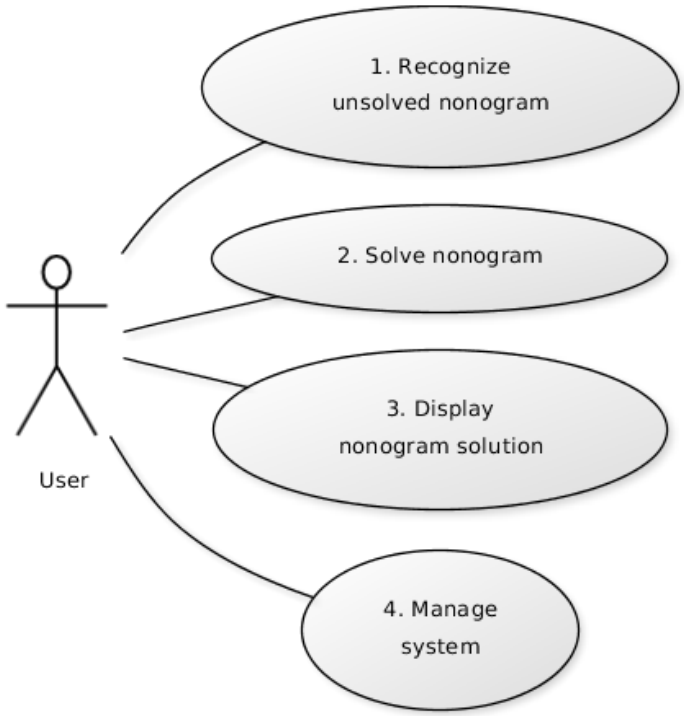


Figure 31. System's use case diagram

1. USE CASE: Recognize unsolved nonogram **User/Actor name:** User **Description:** Recognizes supplied nonogram and converts it to format recognized by the system. In this case matrix will be used. **Conditions before:** No nonogram was supplied as input data. **Invoke conditions:** User starts nonogram solving by pressing 'START' button. System begins nonogram search. **Conditions after:** Nonogram solving can be initiated.

2. USE CASE: Solve nonogram

User/Actor name: User

Description: Solves supplied nonogram with one of nonogram solution algorithms.

Conditions before: User starts nonogram solving by pressing 'START' button.

Invoke conditions: Nonogram was recognized in video feed or picture.

Conditions after: Solution can be displayed for user.

3. USE CASE: Display nonogram solution

User/Actor name: User

Description: Augments video feed or picture with nonogram solution.

Conditions before: Nonogram was solved.

Invoke conditions: Nonogram solution was prepared for displaying.

Conditions after: Results can be saved.

4. USE CASE: Manage system

User/Actor name: User

Description: User can view help, solving logs. When nonogram solving is in progress user can terminate it.

Conditions before: None.

Invoke conditions: User selects one of following actions: view help, show logs, terminate solving.

Conditions after: None.

3.4.2. Functions

Basic functional requirements list for Nonogram Solver:

1. System should recognize nonogram grids with different dimensions.
2. System should recognize numerals 1, 2, 3, 4, 5, 6, 7, 8 and 9.
3. System should display ■ (Square) symbol for filled squares.
4. System should notify user if there is no objects to recognize.
5. System should recognize objects from pictures.
6. System should allow solving process termination.

In the following tables each requirement is specified.

Table 4 Functional requirement no. 1 detailed description

Requirement number: **1**	Requirement type: **1**	Use case number: **1**
Description: **System should recognize nonograms with different dimensions**		
Rationale: **To be able solve different level nonograms.**		
Source: **Andrej Ušaniov**		
Fit criterion: **Different dimensions of nonograms will be recognized.**		
Customer satisfaction: **5**	Customer dissatisfaction: **2**	
Dependencies: **All requirements using nonograms structure and dimensions**		Conflicts: **None**
Supporting materials: **None**		
History: **Created March 10th, 2011**		

Table 5 Functional requirement no. 2 detailed description

Requirement number: **2**	Requirement type: **1**	Use case number: **1**
Description: **System should recognize numerals 1, 2, 3, 4, 5, 6, 7, 8 and 9**		
Rationale: **Nonogram puzzle consist of 1, 2, 3, 4, 5, 6, 7, 8, 9 numerals. So they should be recognized by the system.**		
Source: **Andrej Ušaniov**		
Fit criterion: **1, 2, 3, 4, 5, 6, 7, 8, 9 numerals will be recognized and used by the system. Numbers will be recognized from video or photo.**		
Customer satisfaction: **3**	Customer dissatisfaction: **2**	
Dependencies: **All requirements where recognized nonogram data is used.**		Conflicts: **None**
Supporting materials: **None**		
History: **Created March 10th, 2011**		

Table 6 Functional requirement no. 3 detailed description

Requirement number: **3**	Requirement type: **1**	Use case number: **3**
Description: **System should display ■ (Black Square) symbol for filled squares**		
Rationale: **Nonogram solution should be visible to the user.**		
Source: **Andrej Ušaniov**		
Fit criterion: **Nonogram solution will be displayed as ■ (Black Square) symbols.**		
Customer satisfaction: **5**	Customer dissatisfaction: **1**	
Dependencies: **None**		Conflicts: **None**
Supporting materials: **None**		
History: **Created March 10th, 2011**		

Table 7 Functional requirement no. 4 detailed description

Requirement number: **4**	Requirement type: **1**	Use case number: **1, 2**
Description: **System should notify user if there is no objects to recognize**		
Rationale: **User should be informed in case there is no data feed or data feed is faulty**		
Source: **Andrej Ušaniov**		
Fit criterion: **System will notify user if there is no objects to recognize by displaying warning window.**		
Customer satisfaction: **4**	Customer dissatisfaction: **2**	
Dependencies: **All requirements using input data (nonograms)**		Conflicts: **None**
Supporting materials: **None**		
History: **Created March 10th, 2011**		

Table 8 Functional requirement no. 5 detailed description

Requirement number: **5**	Requirement type: **1**	Use case number: **1**
Description: **System should recognize objects from pictures**		
Rationale: **There may be a need to use earlier saved picture of nonogram for solving**		
Source: **Andrej Ušaniov**		
Fit criterion: **System will recognize objects from earlier saved pictures of nonograms. Nonogram will be saved in text file.**		
Customer satisfaction: **4**	Customer dissatisfaction: **2**	
Dependencies: **All requirements using input data (nonograms)**		Conflicts: **None**
Supporting materials: **None**		
History: **Created March 10th, 2011**		

Table 9 Functional requirement no. 6 detailed description

Requirement number: **6**	Requirement type: **1**	Use case number: **4**
Description: **System should allow solving process termination**		
Rationale: **If nonogram solving takes too much time (or other reasons), there should be an opportunity to terminate it.**		
Source: **Andrej Ušaniov**		
Fit criterion: **System will allow nonogram solving termination by pressing <Cancel> button.**		
Customer satisfaction: **4**	Customer dissatisfaction: **2**	
Dependencies: **None**		Conflicts: **None**
Supporting materials: **None**		
History: **Created March 10th, 2011**		

3.5. Architecture

Nonogram solver has 4 basic packages (Figure 32).

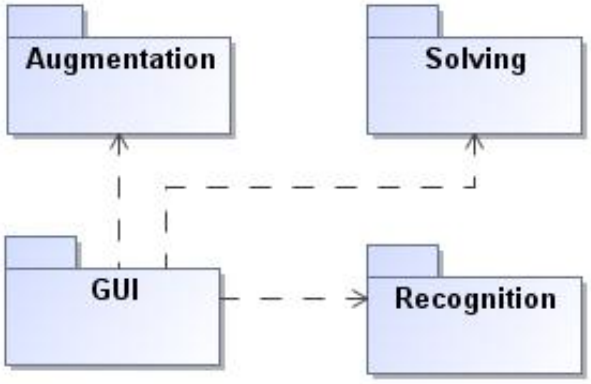


Figure 32. Packages of Nonogram solver

Recognition

Package is used for images recognition. Nonogram is split into different parts then these parts are converted to useful data for nonogram solving and augmentation.

Solving

Package is responsible for nonogram solving

Augmentation

Package is responsible for picture augmentation

GUI

Package is responsible for all graphic windows used in the system, such as Main Window, Help, and Settings. Also it saves results and statistics information of the application

4. Research

During master's study course augmented reality system "Nonogram solver" was implemented with one nonogram solving algorithm: depth-first search. To increase solving performance Constraint programming approach was used.

4.1. Problem

Algorithms have to solve puzzles in reasonable speed. According to Jakob Nielsen [1], one second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Ten seconds is about the limit for keeping the user's attention focused on the dialog. For augmented reality this number is even smaller. Also we have to keep in mind memory usage, because there are limited resources on mobile devices.

4.2. Goals

Goals for nonogram solving algorithms analysis and implementation:

1) Implement nonogram solving algorithms;
2) Measure solution time for implemented algorithms;
3) Compare solution time;
4) Suggest method for faster nonogram solution;

4.3. Suggested implementation

To increase speed of solution, Constraint programming approach was selected. This methodology should increase solution speed with bigger than 15x15 size puzzles. In chapter 2.4 is explained how this algorithm works.

Because we have limited resources in smartphones, other solution can be sending nonogram to the serve, solving it and getting answer, as in Figure 33.

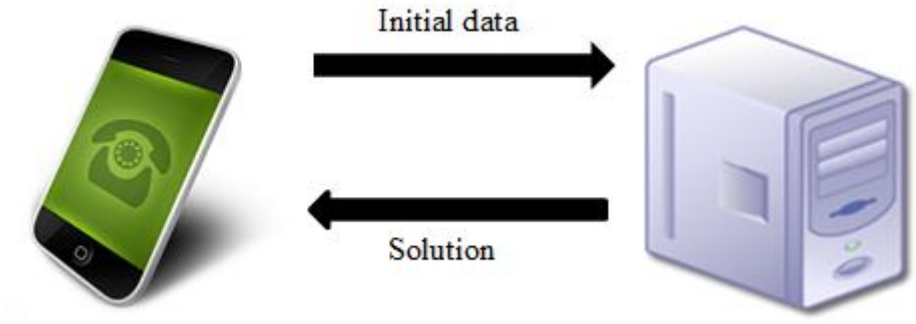


Figure 33 Nonogram's solution on the server

5. Experimentation

5.1. Nonograms for experimentation

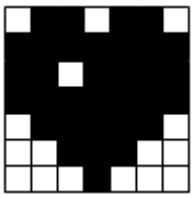


Figure 34 7x7 nonogram "Heart"

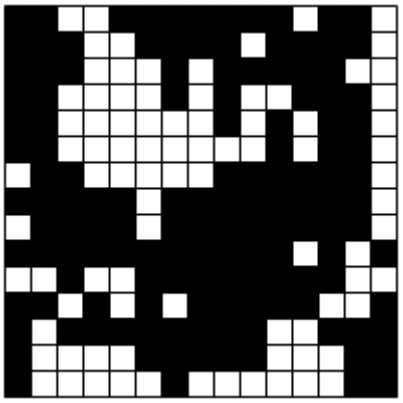


Figure 35 15x15 nonogram "Clutter"

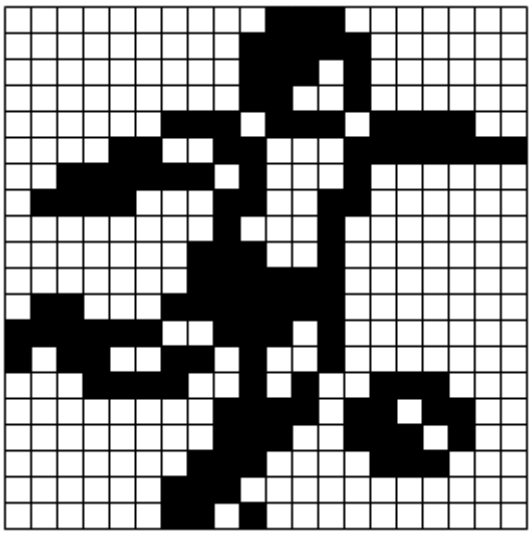


Figure 36 20x20 nonogram "Football player"

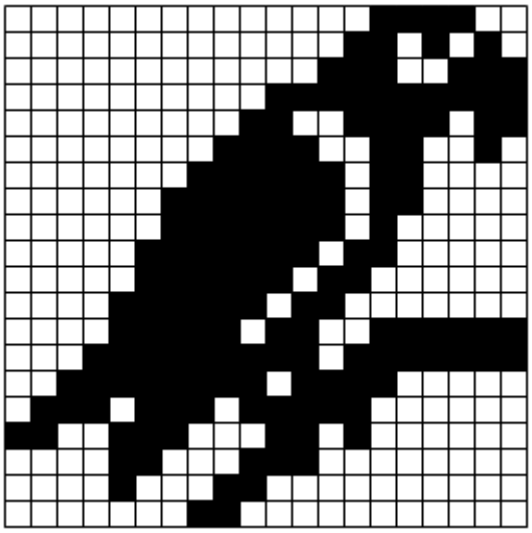


Figure 37 20x20 nonogram "Parrot"

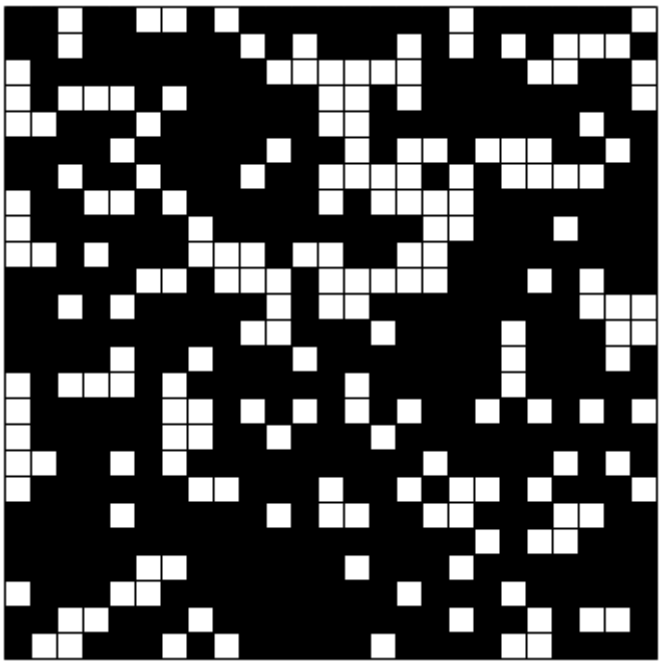


Figure 38 25x25 nonogram "Clutter 2"

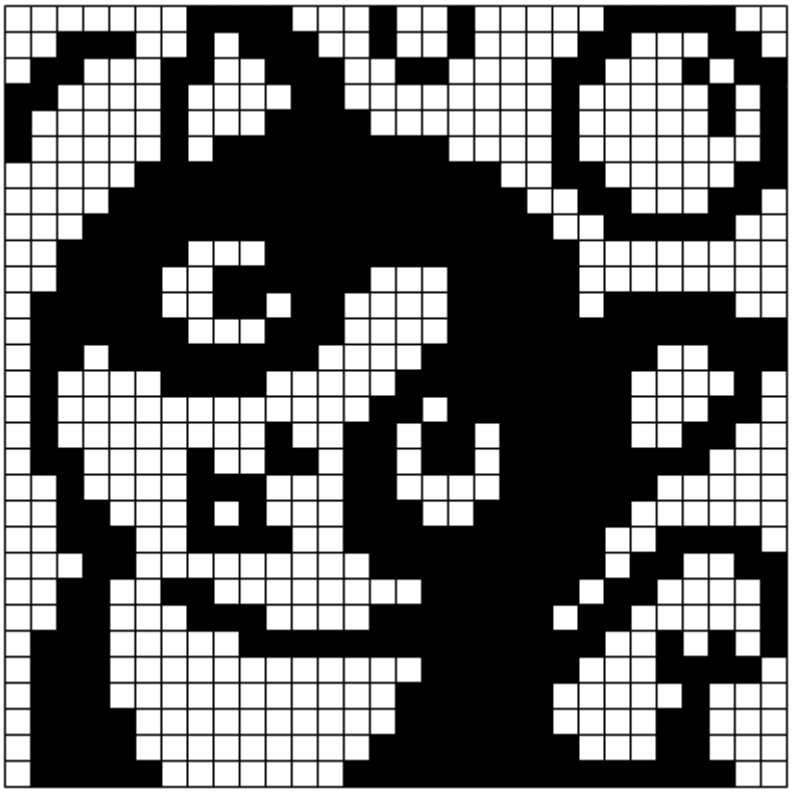


Figure 39 30x30 nonogram "Cat"

5.2. Result

Testing was done with 5 different dimensions nonograms. Exemplary nonograms are shown in Figure 34, Figure 35, Figure 36, Figure 37, Figure 38 Figure 39.

Software was tested on Sony Ericsson Xperia X10 smartphone (384 MB of RAM and 1 GHz CPU) and personal computer (4 GB of RAM and 3.2 GHz CPU).

In Chart 1 and Chart 2 are displayed execution times of both algorithms on smartphone and PC. As we can see “Constraint programming” (CP) implementation’s time is gradually increasing according to a size of puzzle. Opposite situation is with “Backtracking” implementation, where solving time increases very fast, and 20x20 nonogram is a limit for this implementation.

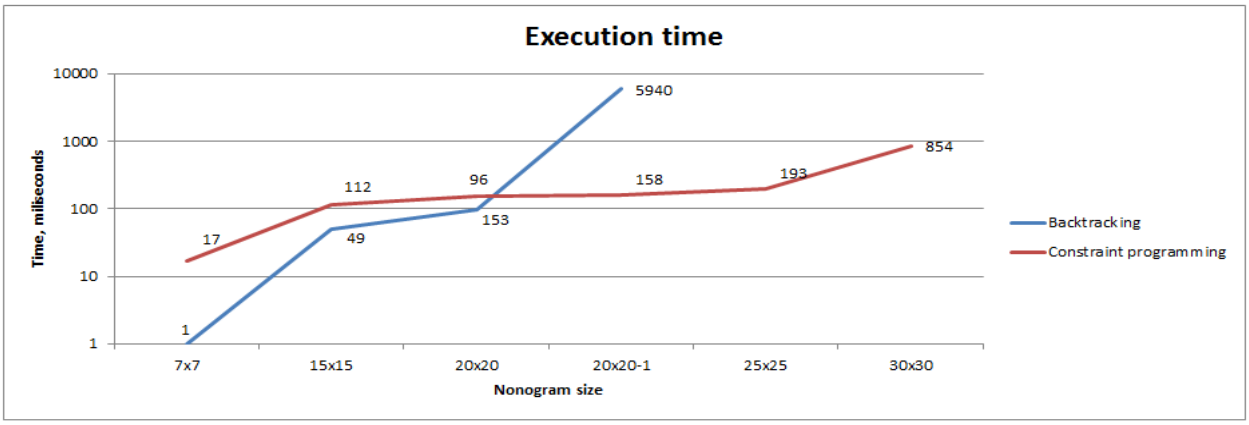


Chart 1 Algorithms execution times on Personal Computer

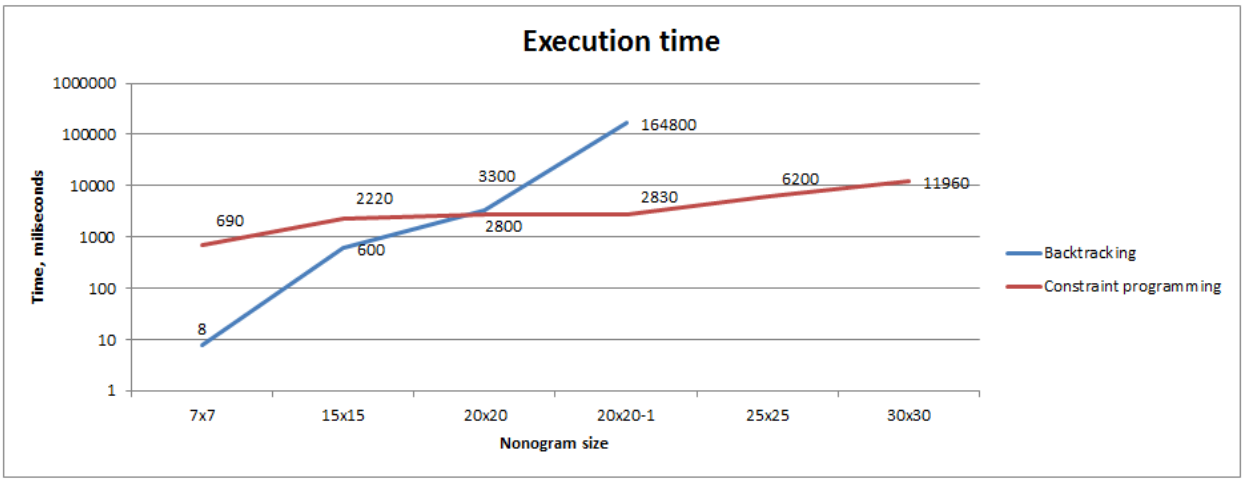


Chart 2 Algorithms execution times on Mobile device

Also we can see, that time for solving same size nonograms can be completely different. In this case, two different 20x20 nonograms are solved in 3300 and 164800 milliseconds on smartphone, that's because of possible paths, that algorithm can take. As we can see in Chart 1 and Chart 2 execution times on smartphone and PC differs significantly because of computational power of devices. For best outcome both nonogram solving algorithms should be combined and all computation should be moved to remote server.

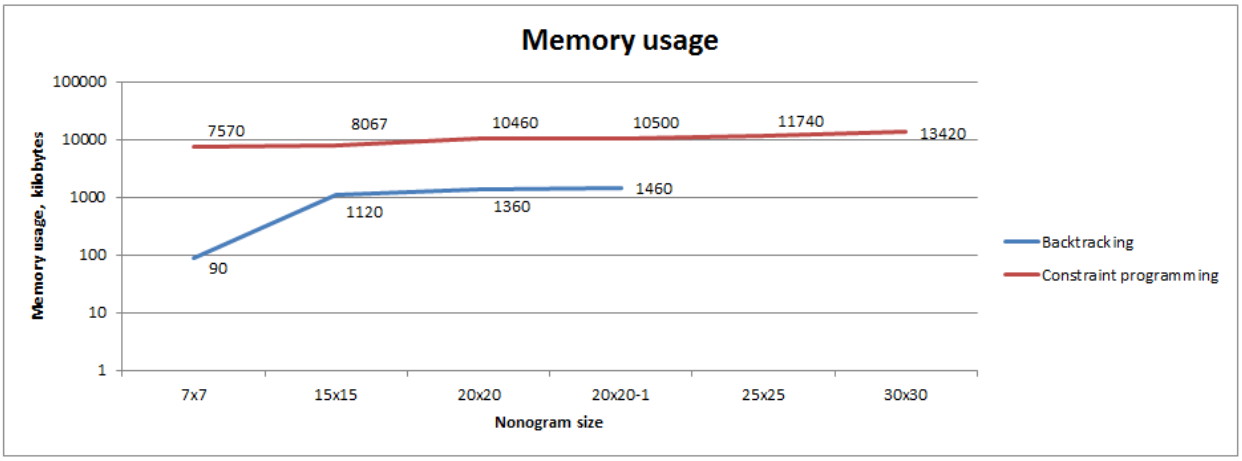


Chart 3 Algorithms memory usage

In Chart 3 are displayed memory usages of both algorithms on smartphone. “Constraint programming” implementation uses more memory than “Backtracking” implementation with every type of nonogram, because each time we have to initialize DFA (deterministic finite automaton) variable. Memory usage difference varies from 5 to 10 times.

6. Conclusions

1. Analysis of the literature shows, that nonograms solution is NP-complete problem and different nonograms of the same size can have very different solution times.
2. Experiment shown, that backtracking depth-first search algorithm is faster than Constraint programming implementation with nonograms which are less than 20x20 size.
3. To get best performance both algorithms should be used – backtracking implementation for small nonograms and Constraint programming for 20x20 size and bigger nonograms.
4. After evaluation and analysis moving all computation to remote server is suggested.

7. Bibliography

[1] **J. Nielsen.** Usability Engineering. 1993.

[2] **Nonogram, Wikipedia.** *http://en.wikipedia.org/wiki/Nonogram.*

[3] **U. Nobuhisa, N. Tadaaki.** Np-completeness results for nonogram via parsimonious reductions. Technical Report TR96-0008, Tokyo Institute of Technology (Titech). *http://www.cs.titech.ac.jp/~tr/reports/.* [May 1996]

[4] **A. B. Robert.** Painting by numbers. Optima, (65):16–17, May 2001. *Also available http://www.oberlin.edu/math/faculty/bosch/pbn.ps.*

[5] **W. Wiggers.** A comparison of a genetic algorithm and a depth first search algorithm applied to japanese nonograms. Paper, Faculty of EECMS, University of Twente, 2004.

[6] **D. E. Goldberg.** Genetic and evolutionary algorithms come of age, *Communications of the ACM, 37:113-119.* 1994.

[7] **D. E. GoldBerg.** Genetic algorithms in Search, Optimization and Machine Learning. *Addison-Wesley.* 1989.

[8] **R. Dechter, D. Frost.** Backtracking algorithms for constraint satisfaction problem s - a tutorial survey, *University of California.* 1998.

[9] **J. Gaschnig.** Performance measurement and analysis of search algorithms. *Technical report CMU-CS-79-124, Carnegie Mellon University.* 1979.

[10] **J. Wolter.** Survey of Paint-by-Number Puzzle Solvers, *http://webpbn.com/survey/.* 2011.

[11] **B. Tudor, C. Pettey.** Gartner Says Worldwide Mobile Phone Sales Grew 35 Percent in Third Quarter 2010. s.l. : Gartner, 2010.

[12] **W. Wiggers.** A comparison of a genetic algorithm and a depth first search algorithm applied to Japanese nonograms. *Twente : Twente student conference on IT,* 2004.

[13] **M. Scheinerman.** Exploring Augmented Reality. *Haverford : Haverford College,* 2009.

[14] **T. Reicher.** A Framework for Dynamically Adaptable Augmented Reality Systems. *Munich : Institute of computer science at the Technical University of Munich,* 2004.

[15] **I. E. Sutherland.** A head-mounted three dimensional display. s.*l. : AFIPS Fall Joint Computer Conference,* 1968.

[16] **R. Raskar.** Spatially Augmented Reality, First International *Workshop on Augmented Reality.* 1998.

[17] **D. Wagner, D. Schmalstieg.** Handheld Augmented Reality Displays. *s.l. : Virtual Reality Conference, 2006.*

8. Acronyms

AR – augmented reality

HCI – human-computer interaction

NP - non-deterministic polynomial-time

PDA – personal digital assistant

OS – operating system

GUI – graphical user interface

CSP – constraint satisfaction problem

CP – constraint programming

DFA - deterministic finite automaton

PC – personal computer