Katholieke
Universiteit
Leuven

**Department of
Computer Science**

PROJECT APLAI

Report

Advanced Programming Languages for AI (H02A8A)

Bavo Goosens (r0297884)
Sander Geijsen (r0304675)

Academic year 2014–2015

Abstract

The goal of this project was to learn AI constraint programming languages and to compare them on a few example problems. As such the assignment was to develop and discuss two problems(Sudoku and Battleship Solitaire) in ECLiPSe CLP and CHR or Jess. In this report we present our viewpoints and implementations for these two problems. We motivate why we chose CHR or Jess. Finally, we discuss the results of the experiments we ran on our programs.

Contents

1 Introduction

In the following sections we will first introduce the problem domain and the constraints we used to solve the problem. Some important questions we encountered while developing our solutions will be detailed in the first part of each problem domain discussion. The second part will explain the setup and conclusions of the experiments for each problem.

2 Sudoku

Sudoku is a puzzle where the goal is to fill in the gaps with the numbers 1 to 9 according to a set of rules. Each value needs to be represented once in each row, column and block. The problem is documented as NP-Complete. In this section we will first introduce new viewpoints and discuss everything related to these new ways of looking at the problem. After that, we will discuss our choices and problems encountered during the implementation in two programming languages. This leads into a discussion about our experiments. We will conclude with a discussion about the drawn conclusions.

2.1 Viewpoints and Programs

2.1.1 New ViewPoints

We implemented two different viewpoints. In the first one we assume the values (1 to 9) are the variables who have a domain from 1 to 81 indicating at which position they are located. The constraint for this viewpoint are largely the same as the standard viewpoint: a variable can not be located twice, but should be present once, in the same row, column and block. Every variable needs a cardinality of 9 to ensure that it is represented in each column, row and block. As we will see later this viewpoint is quite slow and we have opted to not translate it into CHR. The second viewpoint considers a full 9 by 9 matrix associated with each value from the standard viewpoint. This gives us a total of $81 * 9$ variables with a domain from 0 to 1, where a one indicates that this value is present at this index in the full solution of the Sudoku puzzle.

2.1.2 Alternative Viewpoint considered

Beside these two viewpoints, we also implemented a third viewpoint. We consider again a grid from the standard 9x9 viewpoint. We label each square from 1 to 81. Another 9x9 grid represents the location of the values corresponding to the row number. e.g. row 1 contains the value 6,19, . . . This means that in the original grid, there stands a 1 on the locations 6,19, . . . The constraints on these domain variables are the following.

- row: The quotient of the integer division of the location and 9 must be different for all values on the same row of the second grid.
- column: the modulo of the division of the location and 9 must be different for all values on the same row of the second grid.
- block: the result of the following equation must be different for each value on every row of the second grid. The division used is the interger division.

$$(3 * ((XValueOfGrid - 1)/3)) + ((YValueOfGrid - 1)/3) + 1,$$

We did not use this viewpoint any further because we noticed that is was very slow. We ran it on different puzzles and noticed that it took over 1000 seconds for every puzzle. Therefore we implemented a third viewpoint discussed above.

2.1.3 Criteria

A viewpoint is good if it finds a solution for every puzzle relatively fast. This means that we will compare different viewpoints to each other to see which is faster. To achieve this, a viewpoint will have to show certain characteristics. Small domain sizes that limit the amount of backtracks necessary to find a solution. Strong constraint rules that limit the domain sizes. Finally a small number of domain variables. A trade-off between these three has to be made to find a fast viewpoint.

2.1.4 Channeling

We have written some basic channeling which solely consists of transforming the data from one viewpoint into an other. We did not ran experiments on channeling constraints of our two viewpoints.

2.1.5 Programming Language discussion

ECLiPSe

Sudoku is a constraint satisfaction problem. We can limit for the standard viewpoint the domain of the variables to 0..9 and define constraints that represent the Sudoku puzzle. ECLiPSe will narrow down the Domain options by using arc-consistency techniques. This involves selecting the variables of each domain variable that satisfy the constraints. Backtracking will provide a solution if multiple options remain possible.

CHR

CHR differs form ECLiPSe because there is no arc-consistency and backtracking. We use posElement and element as constraints to represent the sudoku puzzle. Each of these constraints contains a position and a value/values. We define constraint handling rules on these constraints to represent the sudoku puzzle. These rules will reduce the possible numbers of possible elements for a position. When no soultion can be found by reducing the domains, we choose a value of a domain of a position. By doing this, we possible fire the constraint handling rules which would reduce the number of possible solutions. We repeat this process until a solution is found. It is possible to implement different search methods by selecting different values from a domain or by choosing different domains first.

2.2 Experiments

2.2.1 Explaining Experiments

In this section we describe which experiments we ran on the implementation in ECLiPSe and CHR. For viewpoint 1 and 3 we tried different search methods. These search methods are tested.

- SD/Dmin : Smallest Domain and smallest value
- SD/Dmax : Smallest domain and biggest value
- LD/Dmin : Largest domain and smallest value
- LD/DMax : Largest domain and largest value

2.2.2 Discussion results of experiments

In this section we discuss the results from our experiments of Sudoku. We ran four different search methods on viewpoint 1 and 2 of ECLiPSe and on viewpoint 1 of CHR. The results for viewpoint 1 of ECLiPSe is shown in table 1. Table 2 shows the results for viewpoint 3 of ECLiPSe. Table **??** shows the results for viewpoint 1 of CHR. From these tables we can conclude that our second viewpoint is the fastest. This was expected because the domain sizes of viewpoint 2 are much. There are more domain variables than viewpoint 1 but for every domain variable that is grounded in the search method, 24 other domain variables are limited. We also notice that the number of backtracks for viewpoint 2 is much lower than viewpoint 1. Here also the small domain sizes are the reason for this behaviour. Figure 1, 2, 3 and 6 also show these results.

We also show the Results of CHR viewpoint 1 in Figure **??** and figure **??**

2.2.3 Difficult puzzles

We didn't look for a reason why certain puzzles were slower than others.

2.2.4 Conclusion Sudoku

We conclude this section with an overview of our obtained results. We observed that viewpoint 3 is faster than viewpoint 1. Also the ECLiPSe implementation of viewpoint 1 is faster than CHR.

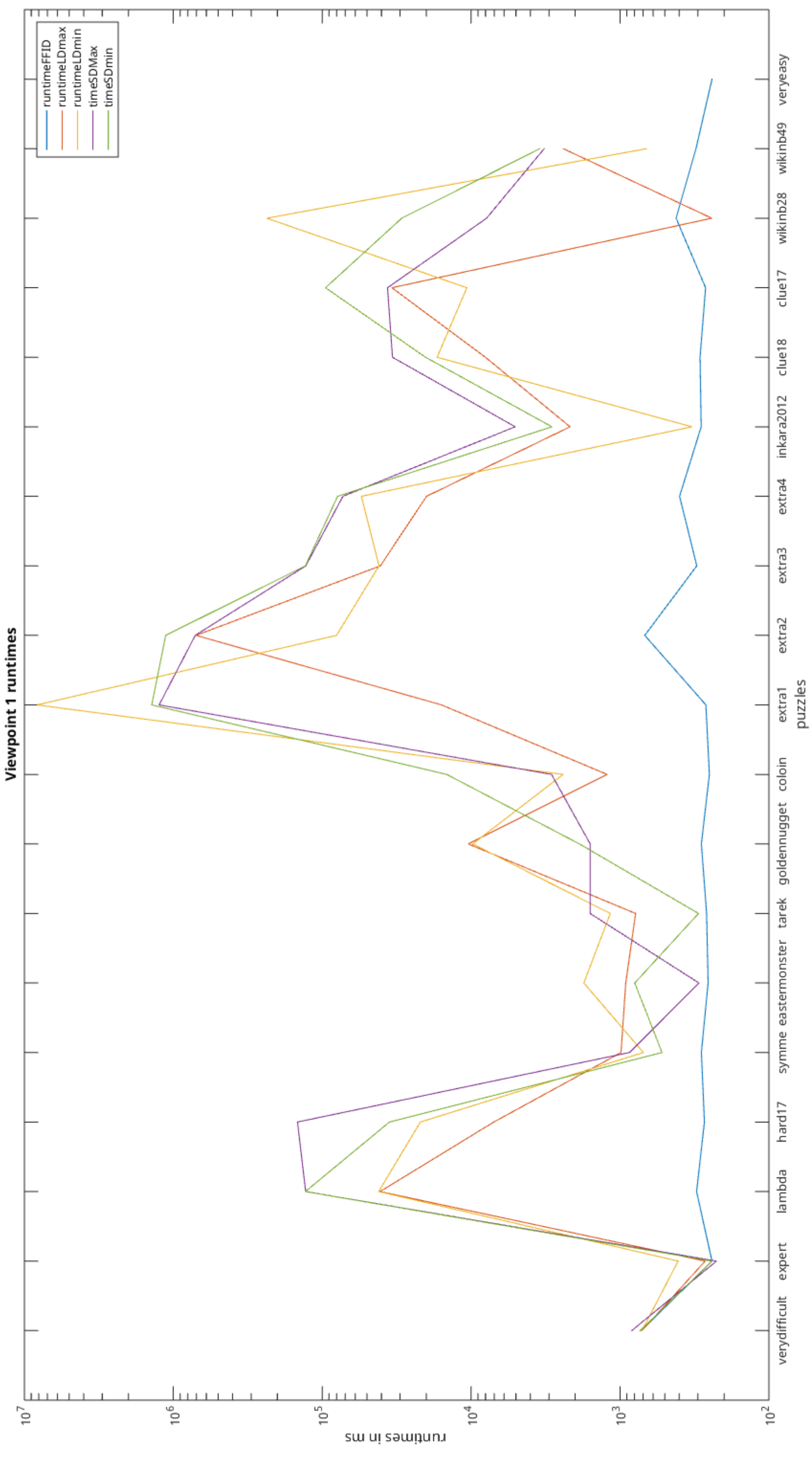

Figure 1: Time for each puzzle viewpoint 1.

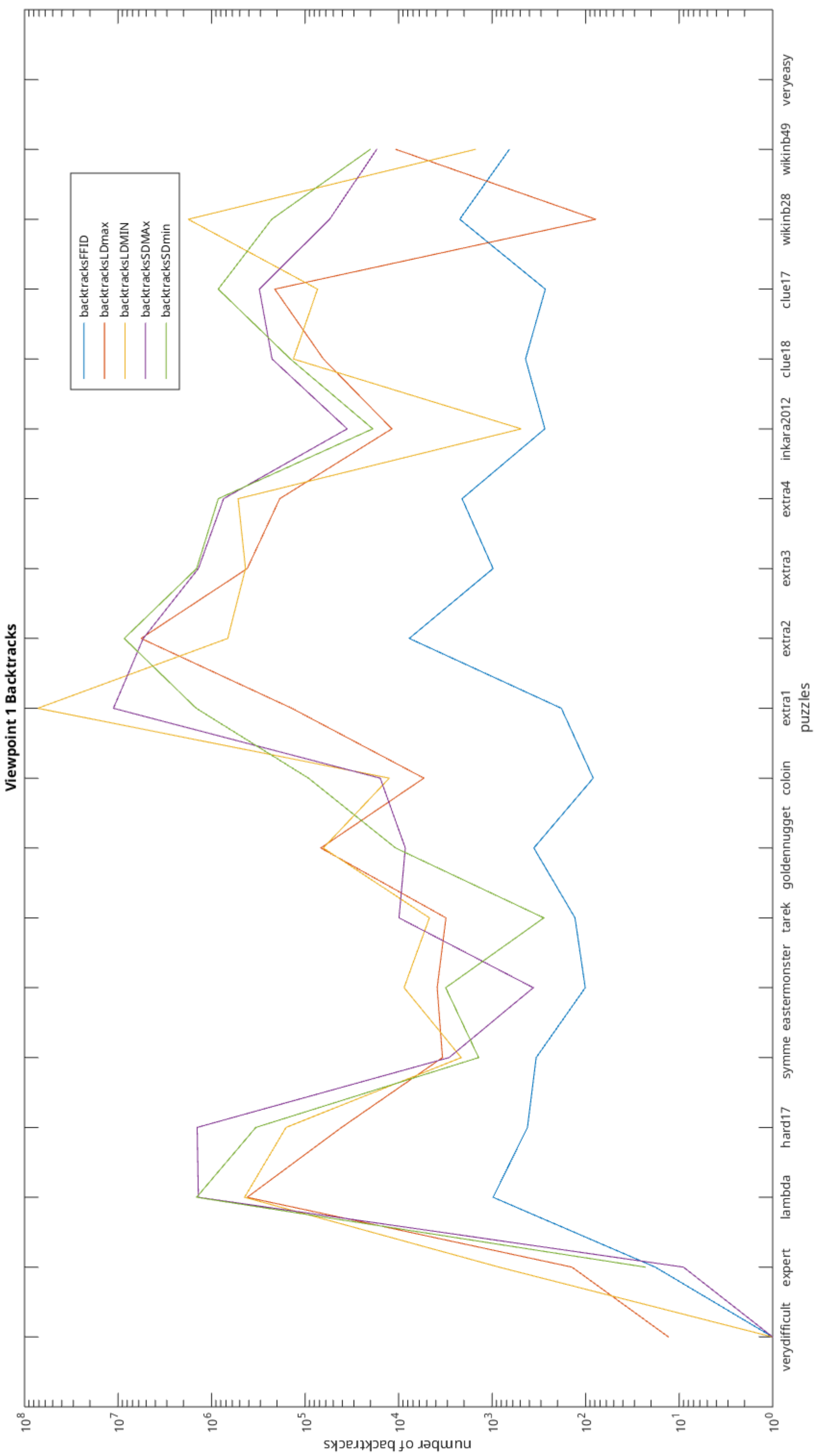

Figure 2: Amount of backtracks for each puzzle viewpoint 1.

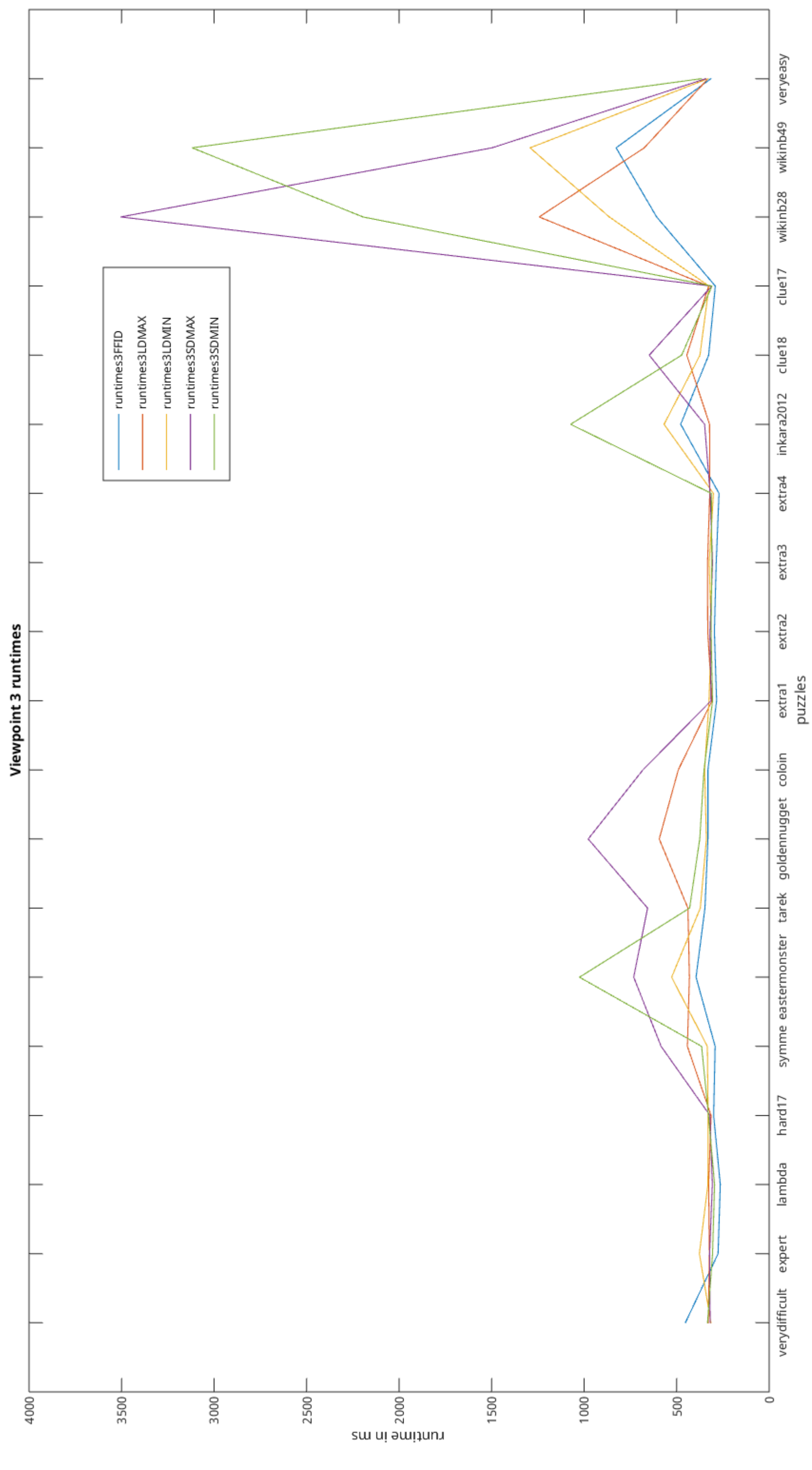

Figure 3: Time for each puzzle viewpoint 3.

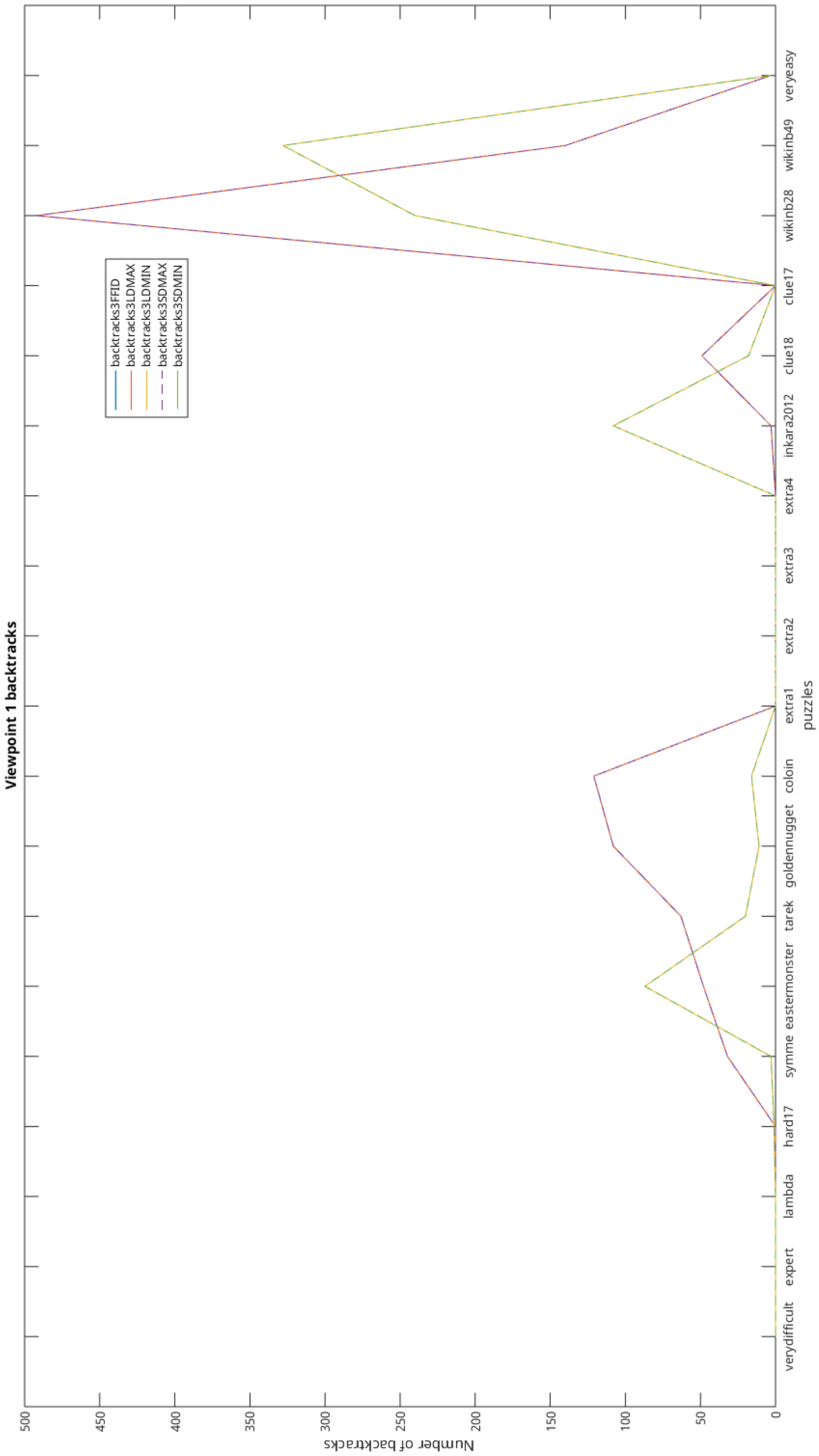

Figure 4: Amount of backtracks for each puzzle viewpoint 3.

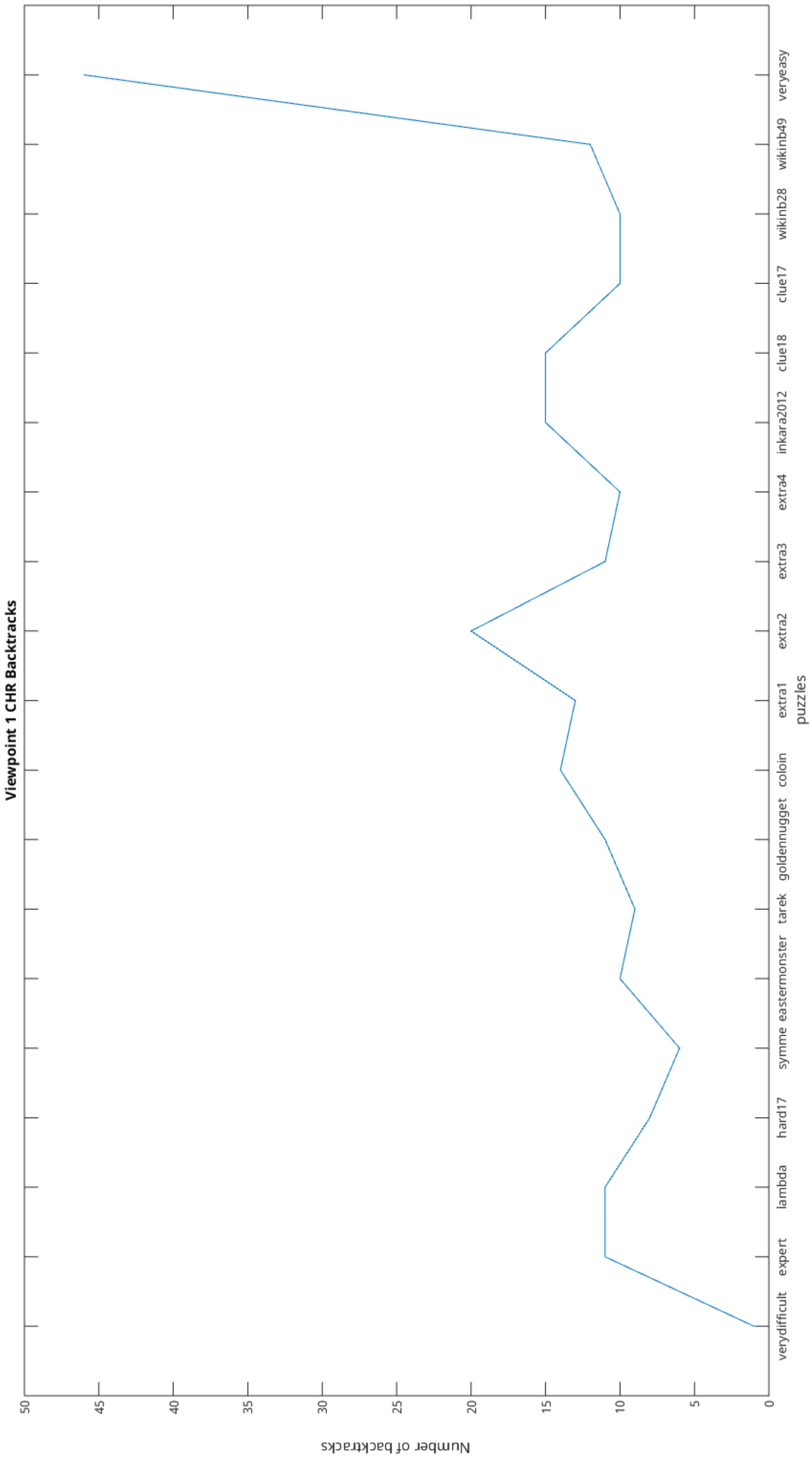

Figure 5: Amount of backtracks for each puzzle viewpoint 3.

Puzzle	SD/Dmin		SD/Dmax		LD/Dmin		LD/Dmax		FF/ID	
	Backtracks	Time	backtracks	Time	backtracks	Time	backtracks	Time	backtracks	Time
verydifficult	0	732	1	834	1	712	13	711	1	718
expert	23	241	9	225	842	407	140	268	18	241
lambda	1445509	129738	1384771	129174	443847	41747	415545	40866	977	306
hard17	336280	35367	1426165	146510	159819	21907	40751	7044	419	271
symme	1394	522	2885	864	2141	697	3396	984	338	284
eastermonster	3139	796	363	295	8741	1749	3869	915	101	256
$\text{tarek}_0 52$	280	297	9902	1587	4687	1162	3116	785	130	262
goldennugget	10762	1872	8480	1585	63837	9755	67910	10445	358	284
coloin	92069	14568	15732	2884	12575	2409	5370	1221	83	251
extra1	1458995	1395583	11207724	1244966	72133768	8222636	138624	15811	182	265
extra2	8590409	1125894	5377367	712429	671855	80158	5673058	703203	7690	682
extra3	1445509	129747	1384771	128930	433847	41507	415545	40641	977	305
extra4	847969	78842	743931	72664	520302	54575	186785	19941	2097	398
inkara2012	18889	2865	35494	5058	489	328	11812	2164	273	285
clue18	143683	20221	225631	33732	133368	16925	63701	8015	439	290
clue17	848842	95013	308434	36418	73689	10683	211022	34038	270	266
$\text{sudowiki}_n b28$	227497	29237	54737	7875	1775791	235391	78	242	2221	420
$\text{sudowiki}_n b49$	20151	3458	17140	3231	1521	665	10648	2401	655	309
veryeasy	/	/	/	/	/	/	/	/	0	241

Table 1: Computation time and number of backtracks viewpoint 1 ECLiPSe for every Sudoku puzzle

Puzzle	SD/Dmin		SD/Dmax		LD/Dmin		LD/Dmax		FF/ID	
	Backtracks	Time	backtracks	Time	backtracks	Time	backtracks	Time	backtracks	Time
verydifficult	0	333	0	319	0	315	0	329	0	453
expert	0	306	0	322	0	377	0	323	0	276
lambda	0	295	0	307	0	331	0	327	0	264
hard17	1	328	0	320	1	330	0	314	1	300
symme	3	365	32	584	3	334	32	442	3	292
eastermonster	87	1025	48	732	87	527	48	431	87	395
tarek$_0$52	20	430	63	657	20	372	63	439	20	347
goldennugget	11	375	108	978	11	339	108	594	11	331
coloin	16	352	121	682	16	349	121	491	16	331
extra1	0	306	0	314	0	325	0	309	0	284
extra2	0	314	0	320	0	313	0	332	0	296
extra3	0	310	0	306	0	324	0	333	0	286
extra4	0	312	0	319	0	301	0	321	0	271
inkara2012	108	1073	3	349	108	568	3	322	108	478
clue18	18	473	49	648	18	374	49	445	18	327
clue17	0	309	0	310	0	325	0	327	0	291
sudowiki$_n b$28	240	2195	492	3505	240	864	492	1243	240	608
sudowiki$_n b$49	328	3118	140	1498	328	1292	140	675	328	827
veryeasy	3	367	3	342	3	331	3	330	3	315

Table 2: Computation time and number of backtracks viewpoint 2 ECLiPSe for every Sudoku puzzle

Figure 6: Amount of backtracks for each puzzle viewpoint 3.

Puzzle	SD/FF	
	Backtracks	Time
verydifficult	1	473
expert	11	671
lambda	11	4271
hard17	8	2016
symme	6	1307
eastermonster	10	1488
tarek$_0$52	9	4081
goldennugget	11	3302
coloin	14	1607
extra1	13	4251
extra2	20	18592
extra3	11	4302
extra4	10	6218
inkara2012	15	6829
clue18	15	3326
clue17	10	8760
sudowiki$_n b$28	10	17830
sudowiki$_n b$49	12	2777
veryeasy	46	551

Table 3: Computation time and number of backtracks viewpoint 1 CHR for every Sudoku puzzle

3 Solitaire Battleship

In battleship a fleet of ships are hidden on a 10x10 grid. The player needs to find a positioning of the ships on this grid that complies with all the hints and rules which are provided as part of the problem. To complete the problem, the grid must contain the following ships. A battle cruiser that takes four squares in length, two cruisers that take three places each, three destroyers that take 2 places each and finally four submarines that take one place each. These ships can only be positioned horizontally or vertically. Two ships cannot occupy adjacent grid squares, not even diagonally.

3.1 Viewpoint and Programs

We implemented a basic solver in ECLiPSe. The input for this problem is a list of hints, a list that represents how many ship parts are located on a row and a list that represents how many ship parts are located on a column. These hints are used to limit the domain of certain variables. After that, ECLiPSe uses arc-consistency and backtracking to find a solution that satisfies all constraints. We introduce the following domain variables and constraints to represent the battleship solitaire puzzle.

3.1.1 Domain Variables

We represent a grid as a 12x12 matrix for convenience of some constraints that we will explain later. We do this by adding a row at the top and bottom. We do the same for the columns. We use two of these grids. In the first grid S we present if there is a ship part located. In the second grid T we present what kind of ship part is located on that position. We use 0 for water, 1 for sub-marine, 2 for cruisers, 3 for destroyers and 4 for battleships. We introduce also four ladder grids. These grids represent the possible ship part locations of a ship. These grids are also 12x12 but contain a array of three elements.

3.1.2 Constraints

In this section we describe the different constraints used in ECLiPSe and CHR. For each constraint we discuss its purpose and if it is a active or passive constraint.

Border Constraint

The first constraint says that all border elements must be the same. This applies to grid S and T. Formally we notate this as

$$s_{0j} = s_{12j} = s_{i0} = s_{i12} = 0, \forall i, j, 0 \leq i, j \leq 12$$

$$t_{0j} = t_{12j} = t_{i0} = t_{i12} = 0, \forall i, j, 0 \leq i, j \leq 12$$

Cardinality Constraint

The second constraint says that the number of squares that are occupied by submarines is 4, four destroyers 6, for cruisers 6 and four battleships 4. This constraints limits the amount of ships that can be placed on the board. More formally we notate this as

$$|\{t_{ij}|t_{ij} = k, 1 \leq i, j \leq 10| = l$$

where $l = 4, 6, 6, 4$ when $k = 1, 2, 3, 4$

Tally Constraint

This constraint says the number of occupied squares in a row or column must be equal to the given number. These numbers are provided like we described above. More formally we notate this as

$$\sum_{j=2}^{11} s_{ij} = R_{i-1}$$

where R_i is the number of occupied squares for row i.

$$\sum_{i=2}^{11} s_{ij} = C_{j-1}$$

where R_j is the number of occupied squares for column j.

Occupied Constraint

This constraint says that when a square is occupied, the squares that are the cornering the occupied squares are water. More formally we notate this as

$$if\, s_{ij} = 1 \quad then \quad s_{i-1j-1} = s_{i-1j+1} = s_{i+1j-1} = s_{i+1j+1} = 0$$

We represent the grid as an 12x12 matrix. We used this representation because now we don't have to differentiate between elements that are located on the border of the original 10x10 grid.

Channelling Constraints

This constraint declares the relation between the S and T grid. When a S square is occupied, the corresponding T square must be occupied. More formally we notate this as

$$s_{ij} = (t_{ij} > 0),\ \forall i, j,\ 1 \leq i, j \leq 12$$

Ladder Constraint

This constraints says that there is a run of occupied squares from e.g. (i, j) to $(i, j + k)$. We introduced four ladder grids. One R for runs to the right, one L for left, one U for up and finally one D for runs of occupied squares downwards. for each direction we hold a list of 3 arrays to check how long the ship is that occupies the squares.
The constraints on the ladder variables in the grid R are the following

$$r_{ij1} = 1 \quad iff \quad s_{ij} = 1 \quad and \quad s_{i,j+1} = 1$$

$$r_{ijk} = 1 \quad iff \quad r_{ij,k-1} = 1 \quad and \quad s_{i,j+k} = 1 \quad for \quad 2 \leq k \leq 3 \quad and \quad j+k-1 \leq 12$$

The constraints on the ladder variables in the grid L are the following

$$l_{ij1} = 1 \quad iff \quad s_{ij} = 1 \quad and \quad s_{i,j-1} = 1$$

$$l_{ijk} = 1 \quad iff \quad l_{ij,k-1} = 1 \quad and \quad s_{i,j-k} = 1 \quad for \quad 2 \leq k \leq 3 \quad and \quad j-k-1 > 0$$

The constraints on the ladder variables in the grid U are the following

$$u_{ij1} = 1 \quad iff \quad s_{ij} = 1 \quad and \quad s_{i-1,j} = 1$$

$$u_{ijk} = 1 \quad iff \quad u_{ij,k-1} = 1 \quad and \quad s_{i-k,,j} = 1 \quad for \quad 2 \leq k \leq 3 \quad and \quad i-k-1 > 0$$

The constraints on the ladder variables in the grid D are the following

$$d_{ij1} = 1 \quad iff \quad s_{ij} = 1 \quad and \quad s_{i+1,j} = 1$$

$$d_{ijk} = 1 \quad iff \quad d_{ij,k-1} = 1 \quad and \quad s_{i+k,,j} = 1 \quad for \quad 2 \leq k \leq 4 \quad and \quad i+k-1 \leq 12$$

For each position (i,j) we have four arrays $R[i,j,1..3]$, $L[i,j,1..3]$, $U[i,j,1..3]$ and $D[i,j,1..3]$ that represent possible runs of occupied squares in each direction. The constraint ensuring the correct value of t_{ij} is the following.

$$t_{ij} = max(\sum_{k=1}^{4} r_{ijk} + \sum_{j=1}^{4} l_{ijk}\,, \sum_{k=1}^{4} u_{ijk} + \sum_{k=1}^{4} d_{ijk})$$

Hint Constraints

These hints represent the constraints that are forced to the square at position (i,j). There are seven different kind of hints that come with a position. water, circle, left, right, top, bottom and middle. A water hint represents a square that contains water. A circle hint represents a square that contains a sub-marine. Left, right, top and bottom hints represents squares that contain a ship part and that there is at least 1 ship part next to it. For left this means that the position $(i,j+1)$ contains a ship part. We formally describe what happens for each kind of hint below

1. water: $s_{ij} = 0$
2. circle: $s_{i-1,j} = s_{i+1,j} = s_{i,j-1} = s_{i,j+1}$
3. left: $s_{i-1,j} = s_{i+1,j} = s_{i,j-1} = 0\,, s_{i,j+1} = 1$
4. right:$s_{i-1,j} = s_{i+1,j} = s_{i,j+1} = 0\,, s_{i,j-1} = 1$
5. top: $s_{i-1,j} = s_{i,j-1} = s_{i,j-1} = 0\,, s_{i+1,j} = 1$
6. bottom: $s_{i+1,j} = s_{i,j-1} = s_{i,j-1} = 0\,, s_{i-1,j} = 1$
7. middle:

$$s_{i-1,j} = s_{i+1,j} = 1 \quad and \quad s_{i,j-1} = s_{i,j+1} = 0 \quad or \quad s_{i-1,j} = s_{i+1,j} = 0 \quad and \quad s_{i,j-1} = s_{i,j+1} = 1$$

3.1.3 Symmetrical solutions

The constraints above allows ECLiPSe to find a solution for the battleship solitaire puzzle. It does not find the correct amount of possible different solutions. We did not implement extra constraints that would allow us to check for symmetrical solutions.

Extra constraint

We introduce an extra constraint to make sure that occupied T squares have the same value if two or more squares are next to each other(excluding diagonally) and contain ship parts. More formally we notate this as

$$if\, s_{ij} = 1 \quad then \quad t_{i-1,j} = t_{ij} \quad or \quad t_{i-1,j} = 0 \quad \forall i,j \; 2 \leq i,j \leq 11$$

$$if\, s_{ij} = 1 \quad then \quad t_{i+1,j} = t_{ij} \quad or \quad t_{i+1,j} = 0 \quad \forall i,j \; 2 \leq i,j \leq 11$$

$$if\, s_{ij} = 1 \quad then \quad t_{i,j-1} = t_{ij} \quad or \quad t_{i,j-1} = 0 \quad \forall i,j \; 2 \leq i,j \leq 11$$

$$if\, s_{ij} = 1 \quad then \quad t_{i,j+1} = t_{ij} \quad or \quad t_{i,j+1} = 0 \quad \forall i,j \; 2 \leq i,j \leq 11$$

3.1.4 CHR

We implemented part of the program in CHR. We didn't finish the program in time because we got stuck of different constraints that involved the occurrences of values(Tally constraint, Cardinality constraint).

3.2 Experiments

3.2.1 ECLiPSe

We ran experiments on our ECLiPSe implementation. We used 4 different search methods. The results are show in table 4, figure 7 and figure 8

3.2.2 CHR

We did not run experiments on our CHR implementation because we did not fully implement all constraints.

Puzzle	SD/Dmin		SD/Dmax		LD/Dmin		LD/Dmax		FF/ID	
	Backtracks	Time	backtracks	Time	backtracks	Time	backtracks	Time	backtracks	Time
15	1003	4023	7124	32652	2623	6483	279	2725	795	4489
17	1	349	136	1209	20	456	32	556	1	425
41	1	336	1776	4848	115	567	104	843	1	379
42	6	390	710	2233	44	493	174	887	6	405
61	103	583	1139	3313	491	1234	171	845	90	639
62	2581	10579	1997	11605	17681	50727	492	2191	2500	9873
67	17	366	8	359	53	476	3	410	15	392
68	377	2005	290	1113	2888	6714	45	544	278	1655
70	44	576	5950	23994	123	694	992	6199	37	606
102	79	431	57	486	395	841	80	620	51	516
112	353	1317	1447	6396	966	2161	225	1544	238	1449
118	1344	7180	452	4956	11077	42057	48	713	1184	6510
133	131	994	3507	16292	2389	6613	214	1352	128	1159
161	0	343	239	1971	0	345	49	732	0	394
173	30	468	6704	16974	2462	5285	192	1252	22	472
177	730	2295	5535	17171	3998	6629	760	3209	557	2287
182	3555	14016	78434	337219	15959	34498	17290	66557	2953	13139
187	25	490	30	670	98	779	4	458	21	510
189	1528	7953	4539	29984	8034	24892	828	4136	1401	6416
204	793	2542	217	1396	2235	6112	60	548	503	2436
222	119	850	536	1941	819	2347	101	793	95	805
233	9	342	95	658	110	460	13	439	7	407
271	2670	10037	9090	28538	12481	25458	1343	5556	2032	9151
280	88	565	3228	11350	149	611	1118	3761	66	541
281	429	1332	1742	4002	1533	2684	392	1330	395	1386
284	22	461	1075	7491	314	1456	101	871	18	463

Table 4: Computation time and number of backtracks ECLiPSe battleship solitaire

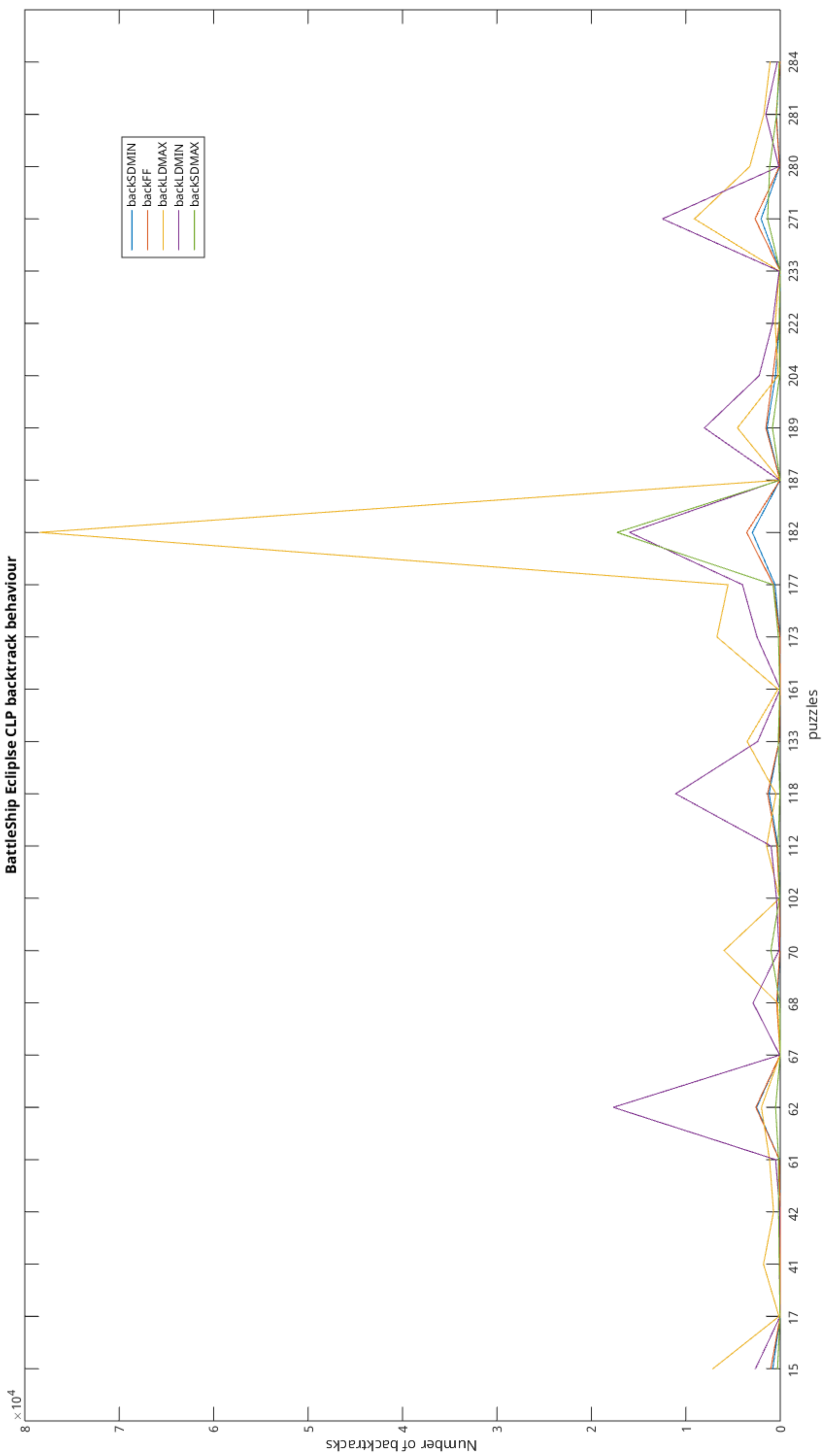

Figure 7: Amount of backtracks for each battleship puzzle.

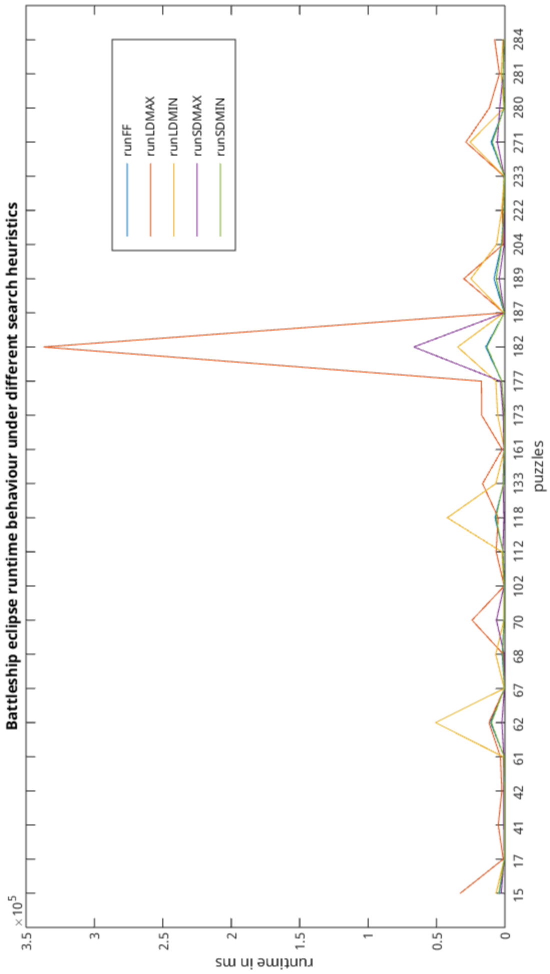

Figure 8: Amount of backtracks for each battleship puzzle.

4 Conclusion

As last section of this document we summarize our project into strong and weak points.

4.1 Strong Points

- Sudoku Eclipse viewpoint 1
- Sudoku Eclipse viewpoint 2
- Sudoku CHR viewpoint 1
- BattleShip Eclipse viewpoint.
- Experiments Sudoku

Weak Points

- CHR Battleship not completed
- CHR Sudoku viewpoint2 not completed
- No symmetry detection
- Undetailed Report