Set Partitioning Problem

Basic Individual Project
Algorithms and Computability

Patryk Walczak

October 2020

Contents

1 Introduction

The document includes two solutions of the set partitioning problem. The first one is the optimal solution using the brute-force approach and the second solution is quasi-optimal. In my case (level 3) both solutions have to meet the assumption - the number of partitioned sets is greater or equal 2. Moreover I have to define one additional fitness function, which is described below.

In each solution I assumed that the set is provided form the text file, and the number (r) of divided subsets is inputted by user. In description of the solution I omitted that aspects and I assumed that the set and the numbers are known. Of course, implementation contains such operations as reading from file.

2 Problem description

For a given set $S = \{a_1, a_2, ..., a_n\}$ find the partition of the set where:

$$w : S \to N_+ = \{1, 2, ...\}$$

$$S = S_1 \cup S_2 \cup ... \cup Sr(\forall 1 \leq i, j \leq r, i \neq j, S_i \cap S_j = \emptyset)$$

Created subsets are pairwise disjoint.
Moreover the split into subsets must be performed via a minimization of the following function

$$max\{w(S_1), w(S_2), ..., w(S_r)\} - min\{w(S_1), w(S_2), ..., w(S_r)\} \to minimal \qquad (1)$$

where:

$$w(S) = \sum_{i=1}^{n} w(a_i)$$

$w(S_i)$ - sum of weights of the elements from S_i

My additional fitness function is:

$$\sum_{i=1}^{r} \left| w(S_i) - \frac{w(S)}{r} \right| \to minimal \qquad (2)$$

My additional fitness function helps in finding the solution with the most even distribution of weights. In my solutions it is considered only for two results with the same minimal difference of weights.

3 Exact solution

3.1 Exact solution description

The first exact solution is using brute-force algorithm. All configurations are considered - maximum and minimal weight and value of the second fitness function is computed for every way of dividing the set into r subsets. It is known that such solution cannot be optimal, but such approach assure that the result is correct.

3.2 Exact solution algorithm

Algorithm 1 Main function

Returns: The list of r subsets which sufficient the both fitness functions

```
1:  Read data-set to S
2:  Read number of partitions to r
3:  Sub = AllSubsets(S, r)
4:  for each element ∈ Sub do
5:      weight - difference of weight between set with the maximum and minimum weight
6:      absweight - the computation of the second fitness function
7:      if weight < w then
8:          w = weight
9:          a = absweight
10:         resultSets = element
11:     else
12:         if weight = w & absweight < a then
13:             w = weight
14:             a = absweight
15:             resultSets = element
16:         end if
17:     end if
18: end for
19: return resultSets
```

My exact solution starts in the main function, where data-set and number of partition are read (lines 1-2).Then with *AllSubsets* algorithm, it obtains all possible divisions data set into r subsets. Next, check the results in the for each loop, and if the result is better then currently saved, save the one as the best solution. After iterating all partitions the most sufficient result is returned.

Algorithm 2 $AllSubsets(S, r)$

Parameters: S - a set, r - number of partitions
Returns: All possible divisions of the set S into r subsets

```
1: if r = 1 then
2:     return S
3: end if
4: if r = length(S) then
5:     result = divide S into r one elements subsets
6:     return result
7: end if
8: a_1 = first element of S
9: remove a_1 from S
10: A = AllSubsets(S, r)
11: resultSets is a list of sets of subsets
12: for each sets ∈ A do
13:     for each set ∈ sets do
14:         Add to set element a_1
15:         Add set to resultSets
16:     end for
17: end for
18: B = AllSubsets(S, r − 1)
19: for each sets ∈ B do
20:     Add to sets_1 set with one element a_1
21:     Add to resultSets set sets_1
22: end for
23: return resultSets
```

The *AllSubset* is the most important algorithm in the exact solution. The function returns the list of all possible divisions input set into r subsets. It is a recursive function so firstly there are two stop conditions.

First, check if the set should be divided by one (lines 1-3) and obviously there is only one such partition - the set as it is. Then next stop condition, while there is set with n elements and r is equal n then it is known there exists one way - divide all elements of the input set into r one-element subset.

After removing the first element from the input set that there exist two recursive calls. One with the decreased set and the same parameter r (line 10). So if there are just r subsets this means that the one removed preciously element has to be an element of one of the already returned sets. The loops (lines 12-17) generates all such solutions and save them result list.

At the end the second recursive call is performed but this time with the decreased

set and r-1 parameter (lines 18). The function returns r-1 subsets, so there sill is one needed. In such way, the removed first element has to create a subset with one element itself. The loop (lines 19-22) generate all combinations of such result.

After the steps, all possible divisions of input data-set into r subset are returned as a list structure.

3.3 Exact solution - correctness proof

The algorithm generates all possible divisions the data-set into r subset. Every combination is checked so to prove the algorithm it is sufficient to prove stop conditions. There are two recursive calls and two conditions are checking at the start (if r is 1 or length of input set is equal r). In each run the function by itself the set is decreased by 1, in the second call additionally, r is also decreased. After the finite number of calls, the set size is equal to either 1 or r, which triggers the conditions in lines 1-3 or 4-7. Therefore, the algorithm has STOP property. It generates all possible solutions, so after checking every one as the best result, it finds the most optimal solution for the given fitness function.

3.4 Exact solution - theoretical complexity analysis

For the exact solution is sufficient to compute the upper bound of the algorithm. The definition of the upper bound is as follows:

$$\text{g is an upper bound of f(denoted by f} = \text{O(g))} \Leftrightarrow \exists_{c>0}\exists_{n_0 \in N}\forall_{n>n0} f(n) \leq c * g(n))$$

Below algorithms contain the time and space complexity on the right-hand side. In the computation, it is assumed that the set has n as the cardinality and r is the number of partitions.

Algorithm 3 Main function — *Time* *Space*

Returns: The list of r subsets which sufficient the both fitness functions

1: Read data-set to S
2: Read number of partitions to r
3: $Sub = AllSubsets(S, r)$ — $T(n,r)$ $S(n,r)$
4: **for each** $element \in \mathcal{S}ub$ **do** — $T(n,r)$ $S(n,r)$
5: $weight$ - difference of weight between set with the maximum and minimum weight — $n*r$ $n*r$
6: $absweight$ - the computation of the second fitness function — $n*r$ $n*r$
7: **if** $weight < w$ **then**
8: $w = weight$
9: $a = absweight$
10: $resultSets = element$
11: **else**
12: **if** $weight = w \ \& \ absweight < a$ **then**
13: $w = weight$
14: $a = absweight$
15: $resultSets = element$
16: **end if**
17: **end if**
18: **end for**
19: **return** $resultSets$

The main function iterate on the result of $AllSubsets(S, r)$ and finds the best solution. So the most important computations are present in the called function.

Algorithm 4 $AllSubsets(S, r)$ — *Time* — *Space*

Parameters: S - a set, r - number of partitions
Returns: All possible divisions of the set S into r subsets

1:	**if** $r = 1$ **then**		
2:	**return** S	1	n
3:	**end if**		
4:	**if** $r = length(S)$ **then**		
5:	$result$ = divide S into r one elements subsets	n	n
6:	**return** $result$		
7:	**end if**		
8:	a_1 = first element of S	1	1
9:	remove a_1 from S		
10:	$A = AllSubsets(S, r)$	$T(n-1, r)$	$S(n-1, r)$
11:	$resultSets$ is a list of sets of subsets		
12:	**for each** $sets \in \mathcal{A}$ **do**	$T(n-1, r)$	$S(n-1, r)$
13:	**for each** $set \in sets$ **do**	r	$S(n-1, r)$
14:	Add to set element a_1	1	1
15:	Add set to $resultSets$		
16:	**end for**		
17:	**end for**		
18:	$B = AllSubsets(S, r-1)$	$T(n-1, r-1)$	$S(n-1, r-1)$
19:	**for each** $sets \in \mathcal{B}$ **do**	$T(n-1, r-1)$	$S(n-1, r-1)$
20:	Add to $sets$ set with one element a_1	1	1
21:	Add to $resultSets$ set $sets$		
22:	**end for**		
23:	**return** $resultSets$		

When r is equal 1 the complexity time complexity is 1 and space is n because its return the input set. When r is equal n then algorithm create n disjoint subsets and return them. In the case time and space complexity is n.

It is easier to use two formulas in rest computation of complexity. Let's denote

$$A(n, r)$$

- number of subsets of an n-element divided on r subsets

$$B(n, r)$$

- complexity of rest $AllSubsets(n, r)$

So

$$A(n, r) = A(n-1, r) * r + A(n-1, r-1)$$

And

$$B(n,r) = B(n-1,r) + A(n-1,r) * r + B(n-1,r-1) + A(n-1,r-1) = \\ = B(n-1,r) + B(n-1,r-1) + A(n,r)$$

After summarizing and computing those all equation the result of complexity is

$$T(n,r) = \mathcal{O}(r^n)$$

4 Quasi-optimal (heuristic) solution

4.1 Heuristic solution description

The second solution is quasi-optimal oriented. I was supposed to create heuristic solution. Here the way of finding best result and number of considered options are changed. The algorithm takes as input the set and the number of expected subsets. It checks the value of the fitness function (any from the two previously mentioned). The solution uses "greedy algorithm".

4.2 Heuristic solution algorithm

Algorithm 5 Main function

Returns: The list of r subsets which sufficient the both fitness functions

1: Read dataset to S
2: Read number of partitions to r
3: Sort set S from the biggest weight to the smallest one
4: **for each** $element \in \mathcal{S}$ **do**
5: $\quad Set = MinimalSubset(resultSets)$
6: $\quad$ Add to Set element.
7: **end for**
8: **return** $resultSets$

Firstly, (after reading the inputs and initialization of variables) the algorithm sorts the data set (line 3) from the biggest weight element to the smallest. Then it iterates on the set choosing to which subset add the element. The examination function is described below.

Algorithm 6 $MinimalSubset(sets)$

Parameters: sets - list of sets
Returns: the set with the minimal weight from the sets

1: $minW$ = min weight form $sets$
2: $result$ = select from $sets$ where sum of weight is = $minW$
3: **if** $result$ has more than one element **then**
4: $\quad result$ = first element of $results$
5: **end if**
6: **return** $result$

In the simple function, the set with the smallest current weight is chosen. Then if there is more than one, the function returns the first set (the order is not significant). The biggest element is added to such chosen subset until all elements of the initial data set are added to some subset.

4.3 Heuristic solution - correctness proof

In the simple function, the set with the smallest current weight is chosen. Then if there is more than one, the function returns the first set (the order is not significant). The biggest element is added to such chosen subset until all elements of the initial data set are added to some subset. The algorithm bases on the main idea of the greedy algorithms. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem. The point is to obtain such a combination of subsets with the smallest possible difference between weights of the subset with the biggest and the smallest weight. Additionally, my algorithm finds the solution with the evenest distribution of weights. So in each step, the best option is to add the biggest-weighting element to the subset with currently the smallest sum of weights. Moreover, if we have more than one such set there is no difference which one is chosen until we add the second fitness function (eq.2). With that assumption, the best to add the current element is the smallest set. Such a procedure always finds the best adding option in each step, which provides the solution.

4.4 Heuristic solution - theoretical complexity analysis

In the case of heuristic solution, it is sufficient to compute lower bound of complexity. The definition of the lower

$$\text{g is an lower bound of f(denoted by f} = \Omega(g)) \Leftrightarrow \exists_{c>0}\exists_{n_0 \in N}\forall_{n>n0} c * g(n) \leq f(n)$$

Below algorithms contain the time and space complexity on the right-hand side. In the computation, it is assumed that the set has n as the cardinality and r is the number of partitions.

Algorithm 7 $MinimalSubset(sets)$ — *Time* *Space*

Parameters: sets - list of sets
Returns: the set with the minimal weight from the sets

```
1: minW = min weight form sets
2: result = select from sets where sum of weight is = minW                r  r
3: if result has more than one element then
4:     result = first element of results
5: end if
6: return result
```

In the *MinimalSubset* I assumed that the finding the minimal value for set the algorithm uses the best searching algorithm with cardinality as the complexity.

Algorithm 8 Main function — *Time* *Space*

Returns: The list of r subsets which sufficient the both fitness functions

1:	Read dataset to S		
2:	Read number of partitions to r		
3:	Sort set S from the biggest weight to the smallest one	$n * log(n)$	$log(n)$
4:	**for each** $element \in \mathcal{S}$ **do**	n	n
5:	$Set = MinimalSubset(resultSets)$	r	r
6:	Add to Set element.		
7:	**end for**		
8:	**return** $resultSets$		

As previous, it is assumed that the algorithm uses the best sort function with n*log(n) time complexity. Then it iterates on the n elements and for each find the best subset. With all the information the lower time complexity of the algorithm denoted as function T is:

$$T(n, r) = n * log(n) + n * r$$

And the lower space complexity is:

$$S(n, r) = log(n) + n * r$$

5 Implementation

5.1 Instruction

In my program there exits "InputData" structure and it contains:

- Number of partitions
- Chosen fitness function (first or second in case of optimal solution)
- Chosen solution (exact of heuristic)
- The initial set of weights

At the start, there is a question to a user to provide data from a file or type them in the console. If the first option is chosen then the program reads the file "set.txt" in localization of program (exe file). The file has to contain suitable data.

Example:
2
o
1
5

or/and

2
o
1
1,2,3,4,5

Such structure can be copied as many times as the user wants and:

- first line contains the number of partitions
- second line contains solution (o for optimal and h for quasi-optimal)
- third line contains the number of fitness function (1 or 2)
- fourth line contains set or number of elements in the set (in the second case the weights are randomly chosen)

If a user chooses the type option on the console there to appear the options of a solution, fitness function and space to provide a number of partitions and set.

5.2 Compilation

The whole project was created in Visual Studio environment and it was used to compile the program. The easiest way is to open the project (.sln file) and from context menu choose "build" button. After that program is built and it can be run by clicking "run" button or execute the created exe file (`AaC_Project.exe`) in "`.\AaC_Project\bin\Debug`" or "`.\AaC_Project\bin\x64\Debug`" localization. It depends on what type of complication is chosen. I preferred the "x64" type of compilation.

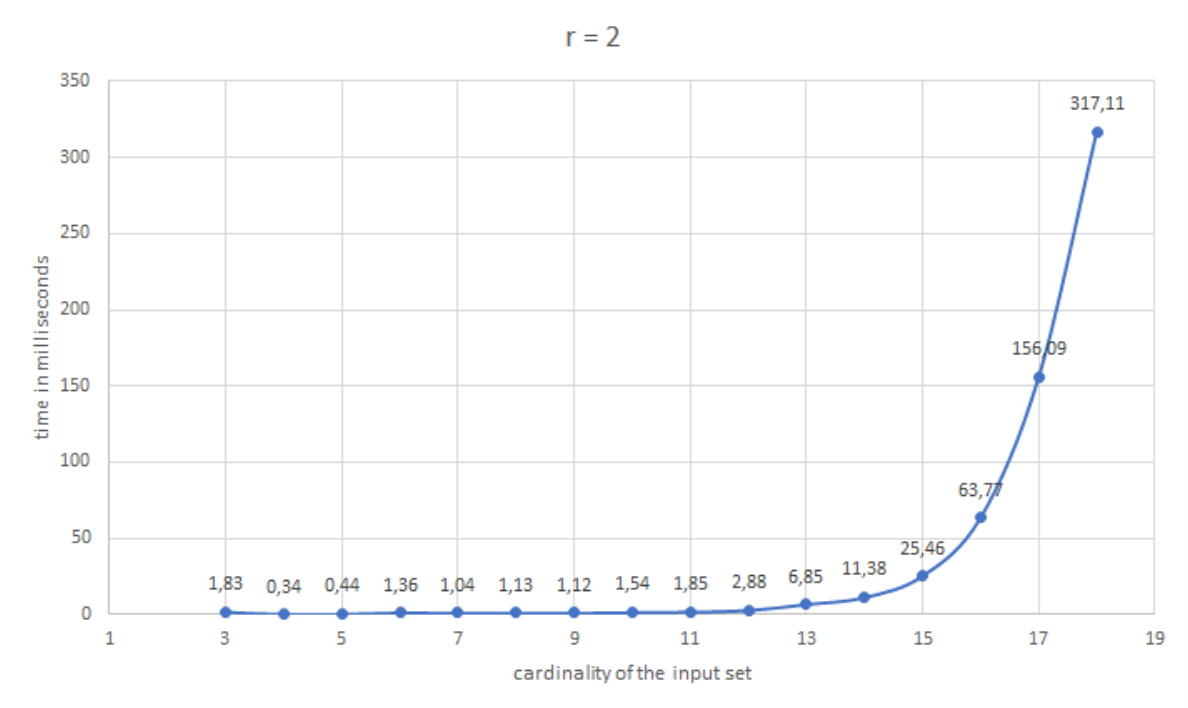
5.3 Exact solution results

For the exact solution, I prepared two types of test. First with the number of partition equal 2 and second equal 4. With the parameters, I generated sets with cardinality from 3 to 18 (each time new sets to have different results). Then the program was run with the initial data 10 times and with the additional function save the results to "results.txt" file. After that, I compute the average of the time results and prepared the graph. Of course, the test was run 10 times for each combination of fitness function and number of the partition.

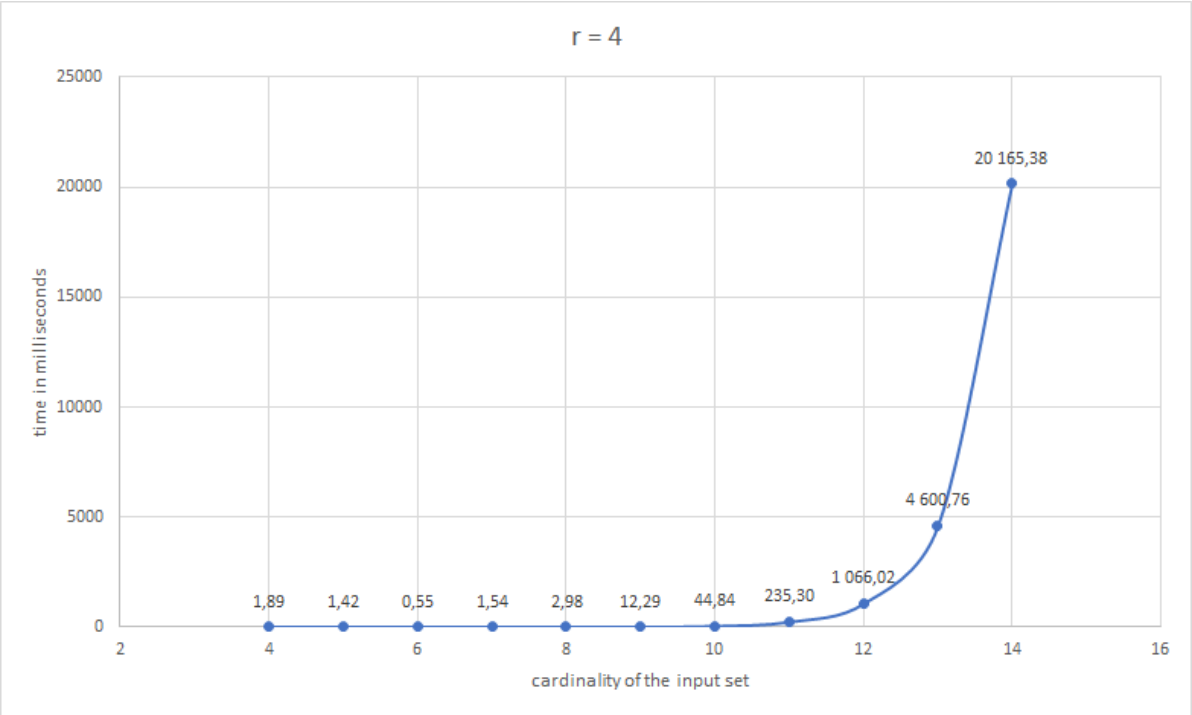
Every results are presented in form of graph where in title is defined the number of partitions, on the y-axis is the time in ms and on the x-axis is the cardinality of input set.

5.3.1 First Fitness Function

The first with number of partitions equal 2 gives the results



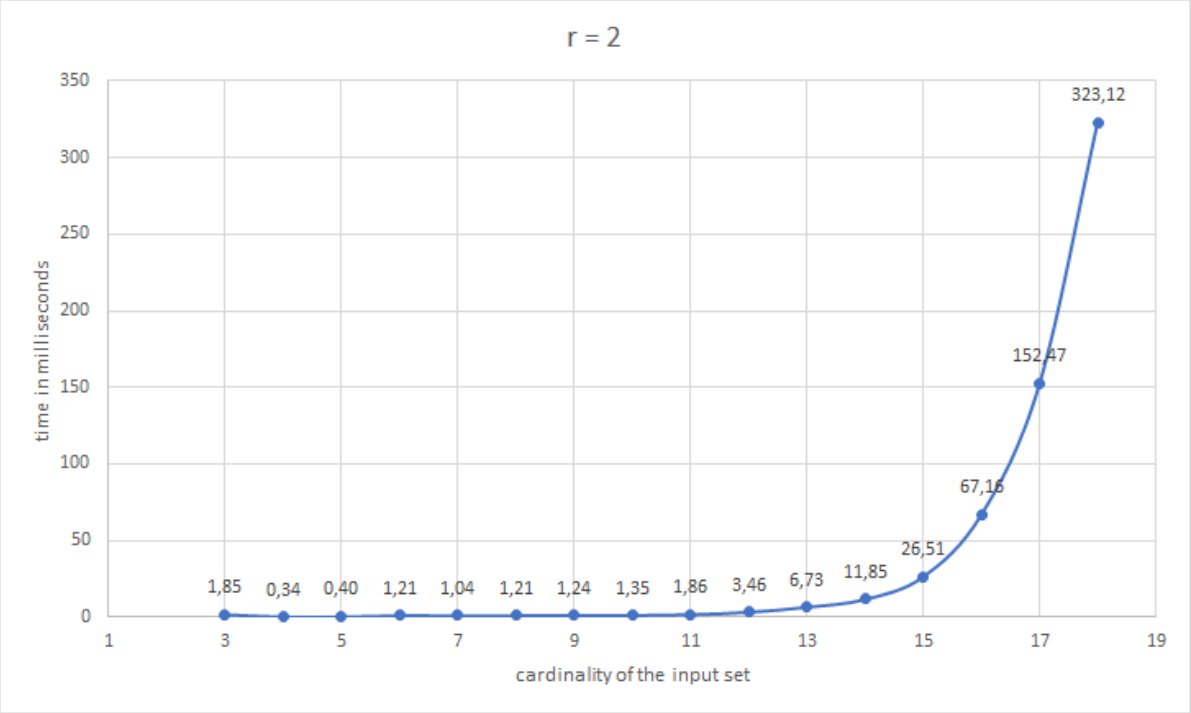
And the second with the number of partitions equal 4 gives



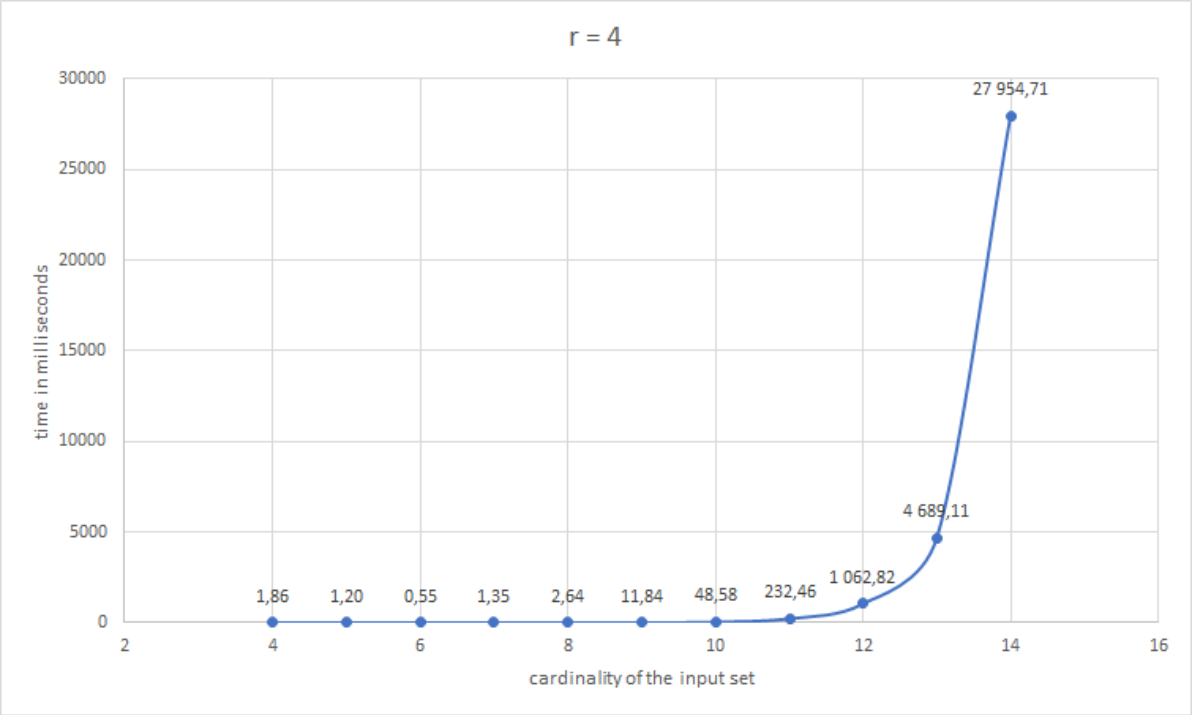
The down axis shows the number of elements in the set, and after those test, there is known that the number of partitions increases the time significantly.

5.3.2 Second Fitness Function

Now, look at the results for the second fitness function.


r = 2
time in milliseconds
cardinality of the input set
350
300
250
200
150
100
50
0
1
3
5
7
9
11
13
15
17
19
1,85
0,34
0,40
1,21
1,04
1,21
1,24
1,35
1,86
3,46
6,73
11,85
26,51
67,16
152,47
323,12

And


r = 4
time in milliseconds
cardinality of the input set
30000
25000
20000
15000
10000
5000
0
2
4
6
8
10
12
14
16
1,86
1,20
0,55
1,35
2,64
11,84
48,58
232,46
1 062,82
4 689,11
27 954,71

5.3.3 Conclusion

The test was performed only for small numbers because as it is written in complexity section the time is growing exponentially. That means the testing higher number for an optimal solution takes a lot of time. But all the test proves that the complexity is exponential and it is shown on all the graphs.

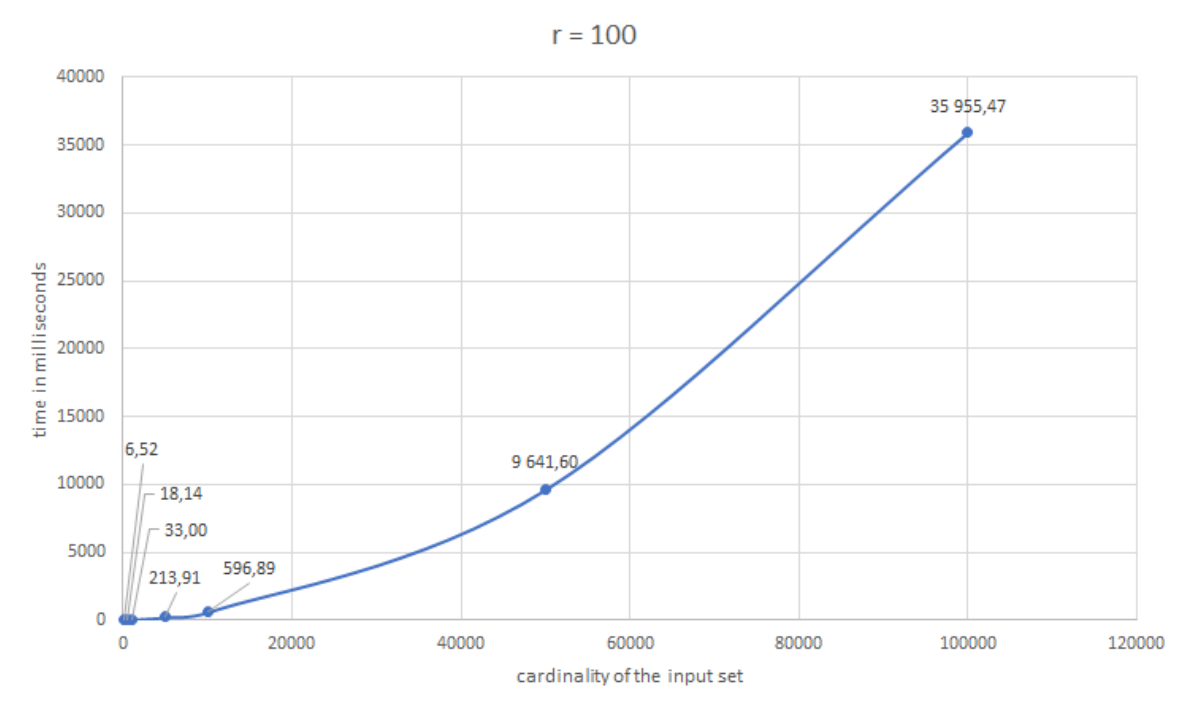
5.4 Quasi-optimal (heuristic) solution results

In the case of heuristic solutions the test are similar to the optimal solution. Here I prepared two types of test - one for number of partition equal 100 and second equal 1000. Moreover, the program was run 10 times and the results shown below are the average of the all results. Also there were used the new additional function which saves the results to file.

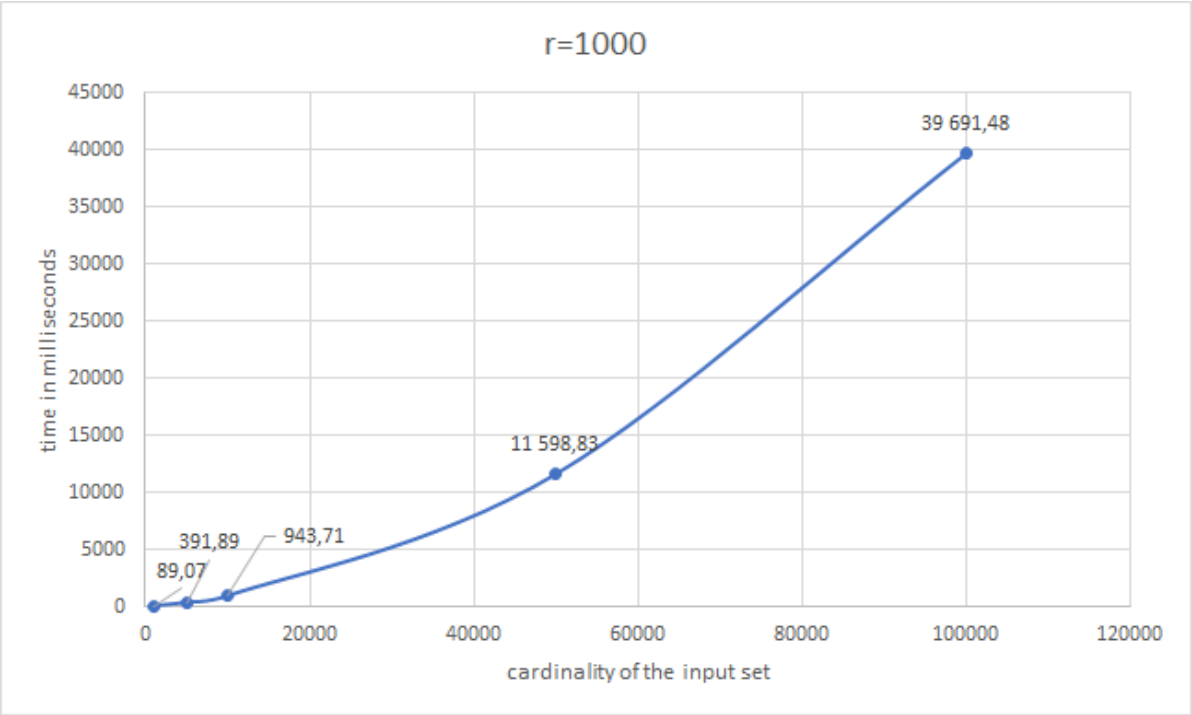
Every results are presented in form of graph where in title is defined the number of partitions, on the y-axis is the time in ms and on the x-axis is the cardinality of input set.

5.4.1 Tests

The first test show the results for number of partitions equal 100.



And the second test show the results for 1000 number of partitions.



5.4.2 Conclusion

In the case of heuristic solution, the cardinality of the set is higher than in optimal solution because the solution is faster and complexity is no exponential. The nearly linear complexity is shown on the upper graphs.
In the comparison of the solutions the difference is shown in complexity section but with the graphical results is proven that the time of computing is much smaller in the heuristic solution.

6 Version list

6.1 Implementation

The program was implemented exactly as it was specified in the documentation on the first checkpoint.

6.2 Changes

Date	Version	Comment
20.10.2020	STAGE_1.0	Sections 1-4
03.11.2020	STAGE_2.0	Complexity sections and preparing implementation
10.11.2020	STAGE_3.0	Improvement of implementation and new results sections
15.11.2020	STAGE_3.1	Improvement of test description, tests and addition of statement about implementation