LOG 734 Heuristics in Analytics

Assignment 2

Students:

191360	Alexandr Reznik
191363	Andrey Belski
191362	Ivan Terekh

1. Introduction

In this assignment, we should design and implement an metaheuristic algorithm for the multidimensional knapsack problem (MKP).

The problem is described in the section 2. Problem definition.

Some heuristics implemented during the previous assignments are presented in section 3. Heuristics from previous assignments .

Two different metaheuristics were implemented and tested during the work on the assignment.

Tabu search is described in section 4. Tabu Search.

GRASP is described in section 5. GRASP .

The final choice is the metaheuristic algorithm based on the heuristic from the assignment 2 mixed with GRASP.

Its results are presented in the section 6. Results.

Problem definition, pseudocode for common heuristic algorithms and inspiration are taken from LOG 734 lectures and lecturer.

Programing language used for the implementation is c++.

2. Problem definition

Multidimensional knapsack problem can be formulated as:

$$max \sum_{j \in V} c_j x_j$$

$$\sum_{j \in V} a_{ij} x_j \leq b_i, \qquad i \in M$$

$$x_j \in \{1,0\}, \qquad j \in V$$

Where $V = \{1, \dots, n\}$ is a set of different items.

$M = \{1, \dots, m\}$ is a set of dimensions.

Item $j \in V$ has a given size $a_{ij} \geq 0$ in dimension $i \in M$. (r[i][j] in the code)

There is a fixed capacity $b_i > 0, i \in M$.

Item $j \in V$ has a given profit $c_j \geq 0$. (p[j] in the code)

x_j denotes if item j is selected or not.

3. Heuristics from previous assignments

Three construction heuristic algorithms were created previously. All of them are very similar and follow the same pseudocode. The only difference is line 4.

1: $V^{\#} \coloneqq \{1, \dots, n\}$

2: $g_i \coloneqq 0 \; \forall \; \boldsymbol{i} \in \{\boldsymbol{1}, \dots, \boldsymbol{m}\}$

3: while $V^{\#} \neq \emptyset$ do

4: calculate $w_j \; \forall \; \boldsymbol{j} \in \boldsymbol{V^{\#}}$

5: Choose a variable $j^* = \arg\max_{\mathrm{j} \in \mathrm{V}^{\#}} \{\frac{c_j}{w_j}\}$

6: **if** $\exists \; \boldsymbol{i} \in \{\boldsymbol{1}, \dots, \boldsymbol{m}\}$: $g_i + a_{ij^*} > b_i$ **then**

7: $x_{j^*} \coloneqq 0$

8: **else**

9: $x_{j^*} \coloneqq 1$

10: $g_i \coloneqq g_i + a_{ij^*} \; \forall \; \boldsymbol{i} \in \{\boldsymbol{1}, \dots, \boldsymbol{m}\}$

11: **end if**

12: $V^{\#} \coloneqq V^{\#} \setminus \{j^*\}$

13: **end while**

14: Output $f = \sum_{j \in \{1,\dots,n\}} c_j * x_j$

In the code each heuristic is presented as a function `solveConstruction(…);` with the different last argument. Line 4 for different heuristics is the following:

Static, normalized:

$w_j = \sum_{i \in \{1,\dots,m\}} \frac{a_{ij}}{b_i} \; \forall \; \boldsymbol{j} \in \boldsymbol{V^{\#}}$**.**

Dynamic, normalized:

$\sum_{i \in \{1,\dots,m\}} \frac{a_{ij}}{(b_i - g_i)} \; \forall \; \boldsymbol{j} \in \boldsymbol{V^{\#}}$.

Dynamic, L2-normalized:

$w_j = \sqrt{\sum_{i \in \{1,\dots,m\}} \left(\frac{a_{ij}}{(b_i - g_i)}\right)^2} \; \forall \; \boldsymbol{j} \in \boldsymbol{V^{\#}}$.

As for improvement heuristics there were used several neighborhood operators such as:

1. Flip: $N^1(x) = \{x' : \sum_{j=1}^{n} |x_j - x'_j| = 1\}$

2. Double-flip: $N^2(x) = \{x' : \sum_{j=1}^{n} |x_j - x'_j| = 2\}$
3. Triple-flip: $N^3(x) = \{x' : \sum_{j=1}^{n} |x_j - x'_j| = 3, -3 < \sum_{j=1}^{n} (x_j - x'_j) < 3\}$
4. Quadruple-flip: $N^4(x) = \{x' : \sum_{j=1}^{n} |x_j - x'_j| = 4, -4 < \sum_{j=1}^{n} (x_j - x'_j) < 4\}$

Jagged local search with different neighborhood operators follows the next pseudocode:

1: input: initial solution, x

2: input: neighborhood operator, N

3: $x_{prev} := x$

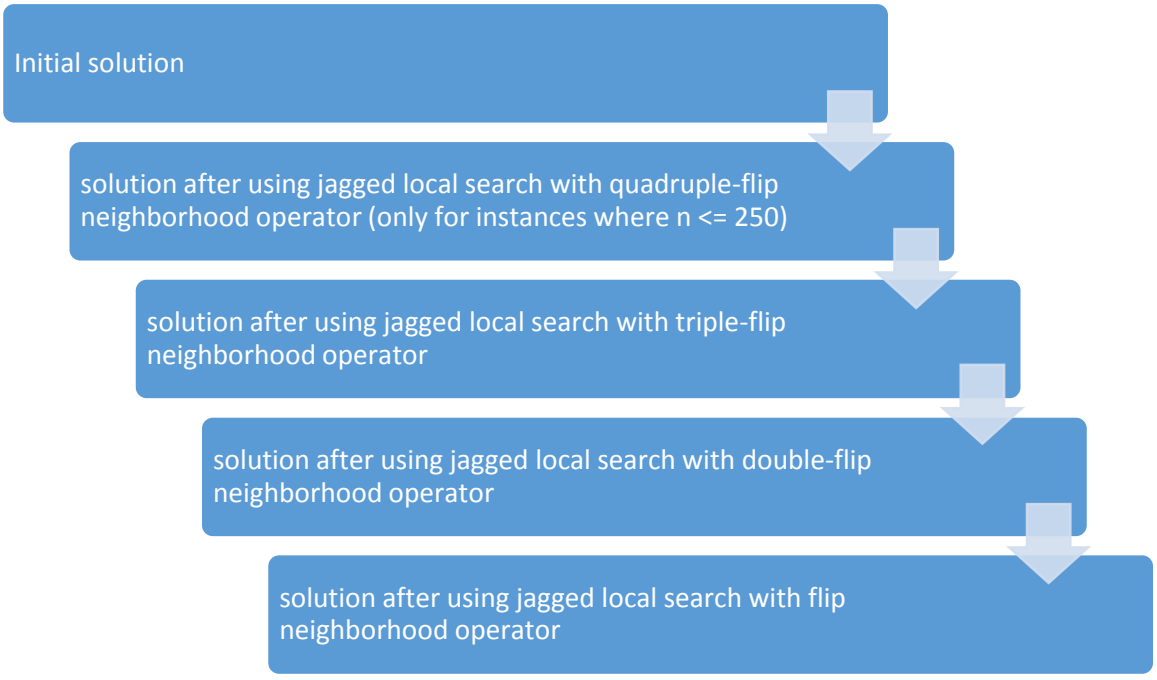
3: **while** there is a $x' \in N(x)$ that is better than x_{prev} **do**

4: Choose the best neighbor $x' \in N(x)$ better than x_{prev}, update $x_{prev} = x, x := x'$.

5: **end while**

- We call 1-search local search with N^1.
- 2j-search is the jagged local search with N^2.
- 3j-search is the jagged local search with N^3.
- 4j-search is the jagged local search with N^4.

The final improvement heuristics is (4j, n<= 250)-3j-2j-1j, which means:



One more search approach was implemented:

1: input: initial solution, x

2: input: neighborhood operators, N^{small}, N^{large}

3: Improve x with jagged local search with N^{small}.

4: **while** there is a $x' \in N^{large}(x)$ that is better than x **do**

5: Choose the best neighbor $x' \in N(x)$ better than x, update $x := x'$.

6: Improve x with jagged local search with N^{small}.

7: **end while**

- We call 24-search previously described search approach with $N^4\ as\ N^{large}$ and $N^2\ as\ N^{small}$.
- 23-search is the previously described search approach with $N^3\ as\ N^{large}$ and $N^2\ as\ N^{small}$.

4. Tabu search

We decided to select Tabu search because we created quite good heuristic algorithms and we thought that it could be beneficial when creating improvement-based metaheuristic. Choosing from simulated annealing and tabu search we decided to use tabu search because it is considered to be a better algorithm in general according to the lectures.

Basic pseudocode is the following:

1: current ← a starting solution

2: Initialize tabu memory

3: **while** stopping criterion not met **do**

4: Find a list of candidate moves, a subset of $N(current)$ according to tabu

5: Select the best solution, s, in the candidate list that maximizes an extended cost function

6: Update tabu memory, perform the move: current ← s

7: **end while**

At first we implemented tabu search with the double-flip neighborhood operator:

$$N^2(x) = \left\{ x' : \sum_{j=1}^{n} |x_j - x'_j| = 2 \right\}$$

We try to find a solution going only in feasible solutions. Tabu attribute is the simplest one. If the variable was flipped on the iteration then it could not be flipped during several next iterations. As the tabu tenure several approaches were tested.

Static tabu tenure tests:

tenure/test	7	9	sqrt(n)	sqrt(n)/2
100-5-01	24282	24242	24242	24326
100-5-02	24225	24225	24225	24225
100-5-03	23449	23449	23449	23523
100-5-04	23313	23391	23313	23434
100-5-05	23826	22787	23826	23947
250-10-01	58776	58840	58776	58842
250-10-02	58328	58328	58328	58409
250-10-03	57634	57595	57514	57662
250-10-04	60624	60624	60624	60624
250-10-05	57643	57643	57643	57654
500-30-01	115141	114135	115141	115141
500-30-02	114043	114043	114043	114043
500-30-03	115327	115829	115617	115678
500-30-04	113999	114505	114505	114505
500-30-05	115686	115617	115326	115584

We can see, that the sqrt(n) is a bit better than others. We concluded that big tenure is not mandatory for this problem and switched to the random tenure testing.

For all the tests above and below 14 was used as the random seed. Alternatively, later 1479 because we had changed it and forgot to change it back. 14 was chosen because the sum of the numbers of letters in Alex, Ivan and Andrey is 14. 1479 was used just because we typed 79. Anyway, we assume that the choice of the seed is somehow critical.

After some testing, we decided that random number from 1 to 10 is a good choice for the tabu tenure. The difference between different numbers was not very high.

But the results were not better than our results from the previous assignment. We decided to implement something else. After a bit of googling it appeared that usually people use flip neighborhood operator for tabu search. We implemented it and added infeasible solutions to our search space. Also we implemented NB aspiration criterium (Revoke tabu for a move that leads to a new best solution).

Evaluating of the solution now became

$$cost = \sum_{j \in V} c_j x_j - \alpha \sum_{i \in M} \sum_{j \in V} (a_{ij} x_j - b_j)$$

Where alpha can be changed.

test	alpha
100-5-01	1
100-5-02	1
100-5-03	1
100-5-04	2
100-5-05	1
250-10-01	10
250-10-02	12
250-10-03	2
250-10-04	2
250-10-05	1
500-30-01	6
500-30-02	6
500-30-03	2
500-30-04	1
500-30-05	2

In the table you can see the best values of alpha for different problems (values from 1 to 12 were tested). Somewhere it is beneficial to use only feasible solutions, somewhere it is not. We did not noticed a pattern here also.

Here are the comparison of double-flip tabu search and single-flip tabu search starting from 3 different construction heuristics (described in the previous section)

	double-flip	single-flip
100-5-01	24381	24381
100-5-02	24274	24274
100-5-03	23551	23538
100-5-04	23456	23479

100-5-05	23991	23966
250-10-01	58918	58471
250-10-02	58452	58161
250-10-03	57924	57381
250-10-04	60715	60605
250-10-05	57805	57643
500-30-01	115265	114878
500-30-02	114075	113941
500-30-03	116002	115340
500-30-04	114687	114111
500-30-05	115765	115340

Double-flip tabu search seems better but it is not better than our previous assignment heuristics.

Perhaps we have not put enough efforts in improving the tabu search.

We decided to try something else.

5. GRASP

We decided to use tabu search because we had good improvement heuristics but it appeared that we didn't use them at all in tabu search. We simply created a new one. And for using our improvement heuristics we decided to implement a construction based metaheuristic that can provide solutions which can be improved using our improvement heuristics. GRASP is easier to implement so we have chosen it from the construction based metaheuristics.

The pseudocode of the construction part of GRASP is the following:

1: $V^{\#} \coloneqq \{1, \dots, n\}$

2: $g_i \coloneqq 0 \; \forall \; \boldsymbol{i} \in \{\mathbf{1}, \dots, \boldsymbol{m}\}$

3: while $V^{\#} \neq \emptyset$ do

4: calculate $w_j \; \forall \; \boldsymbol{j} \in \boldsymbol{V^{\#}}$

5: Find a reduced candidate list, $L^{RC} \subseteq V^{\#}$ consisting of α% best ranked variables according to $\frac{c_j}{w_j}$.

6: Select randomly $j^* \in L^{RC}$

7: **if** $\exists \; \boldsymbol{i} \in \{\mathbf{1}, \dots, \boldsymbol{m}\}$: $g_i + a_{ij^*} > b_i$ **then**

8: $x_{j^*} \coloneqq 0$

9: **else**

10: $x_{j^*} \coloneqq 1$

11: $g_i \coloneqq g_i + a_{ij^*} \; \forall \; \boldsymbol{i} \in \{\mathbf{1}, \dots, \boldsymbol{m}\}$

12: **end if**

13: $V^{\#} \coloneqq V^{\#} \setminus \{j^*\}$

14: **end while**

15: Output $f = \sum_{j \in \{1,\dots,n\}} c_j * x_j$

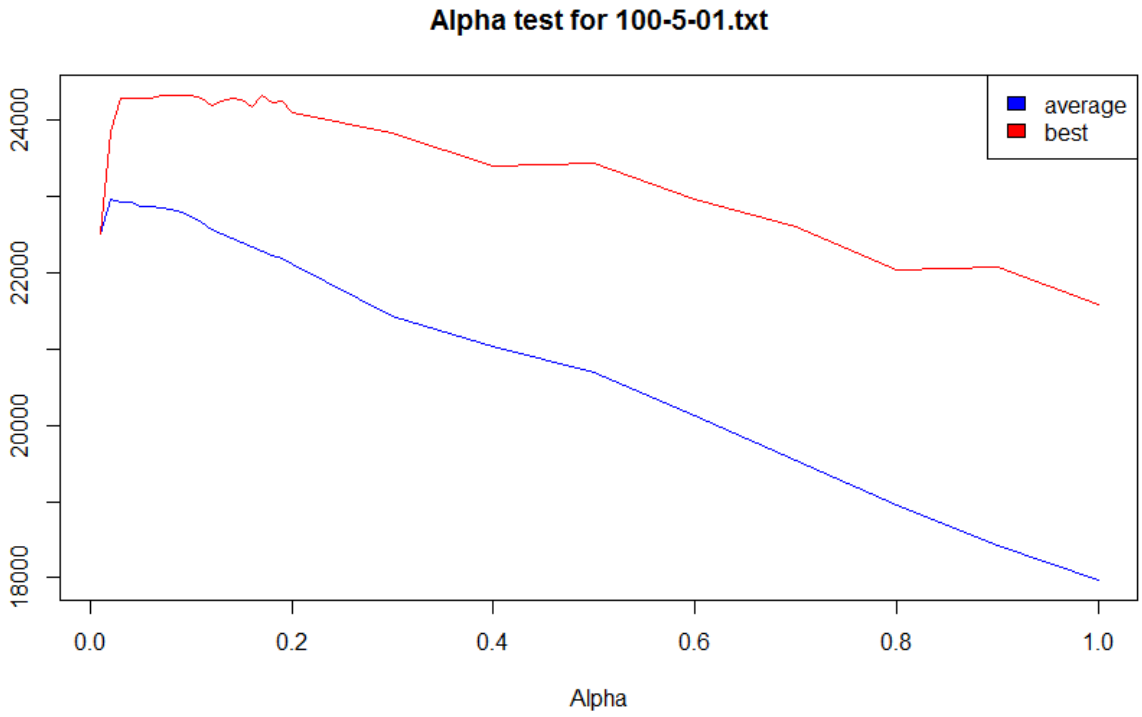
Line 4 of the pseudocode can be changed to one of the approaches described in section 3 (Static, normalized; Dynamic, normalized; Dynamic, L2-normalized).

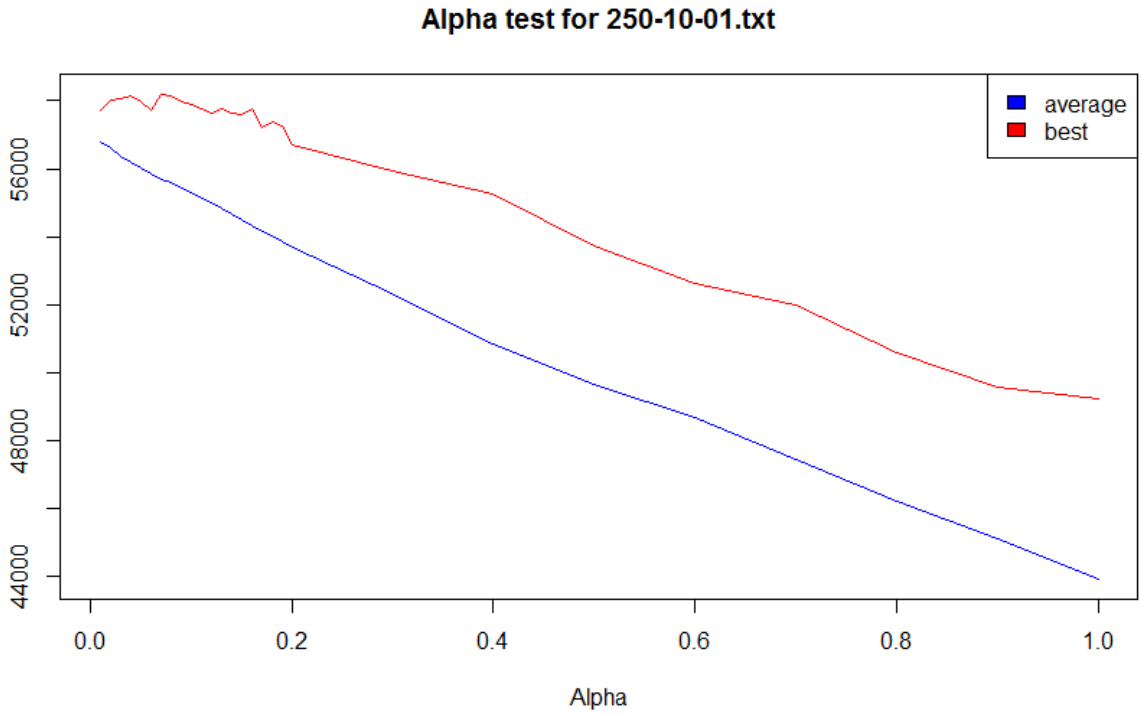
We have tested GRASP with different alpha values (20 seconds for each alpha value from the next table. Static, normalized rate.)

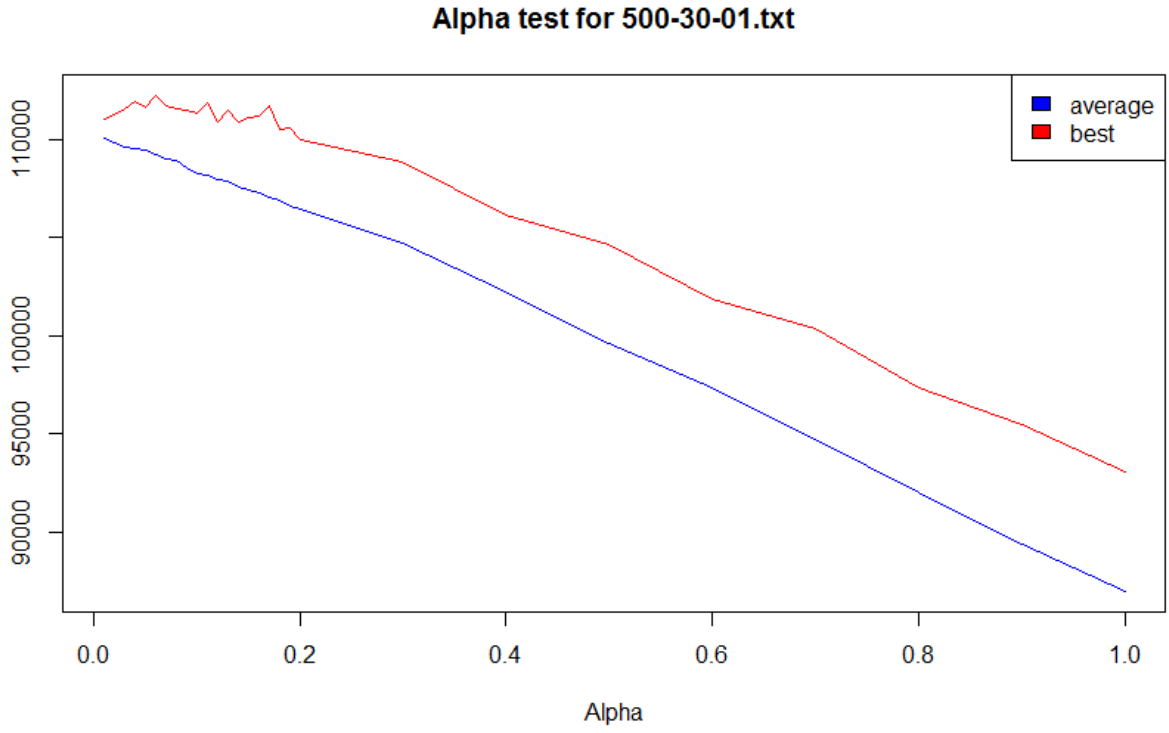
0,01
0,02
0,03
0,04
0,05

0,06
0,07
0,08
0,09
0,1
0,11
0,12
0,13
0,14
0,15
0,16
0,17
0,18
0,19
0,2
0,3
0,4
0,5
0,6
0,7
0,8
0,9
1

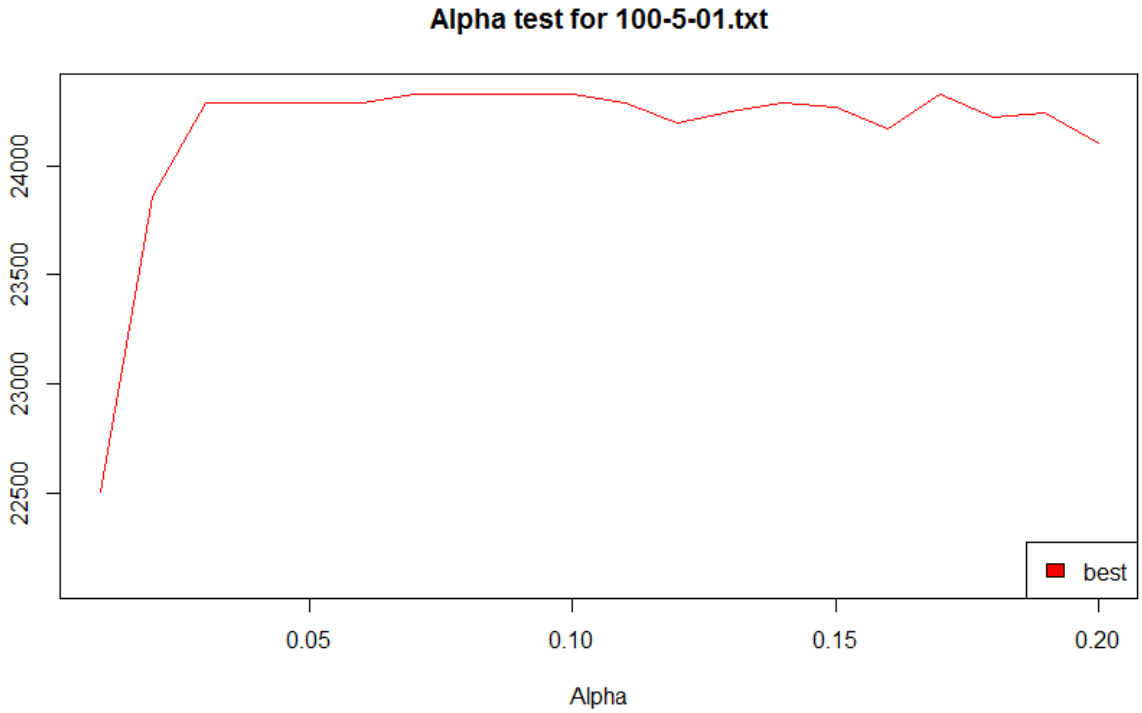
The results for the best and the average value are the following:

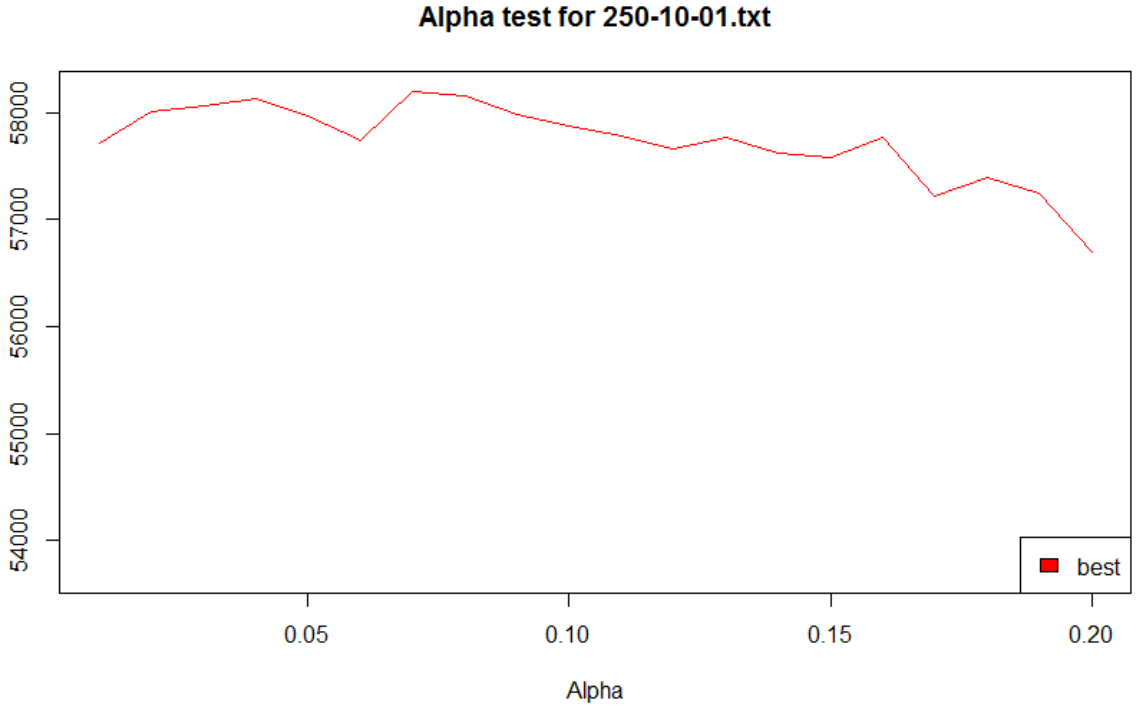


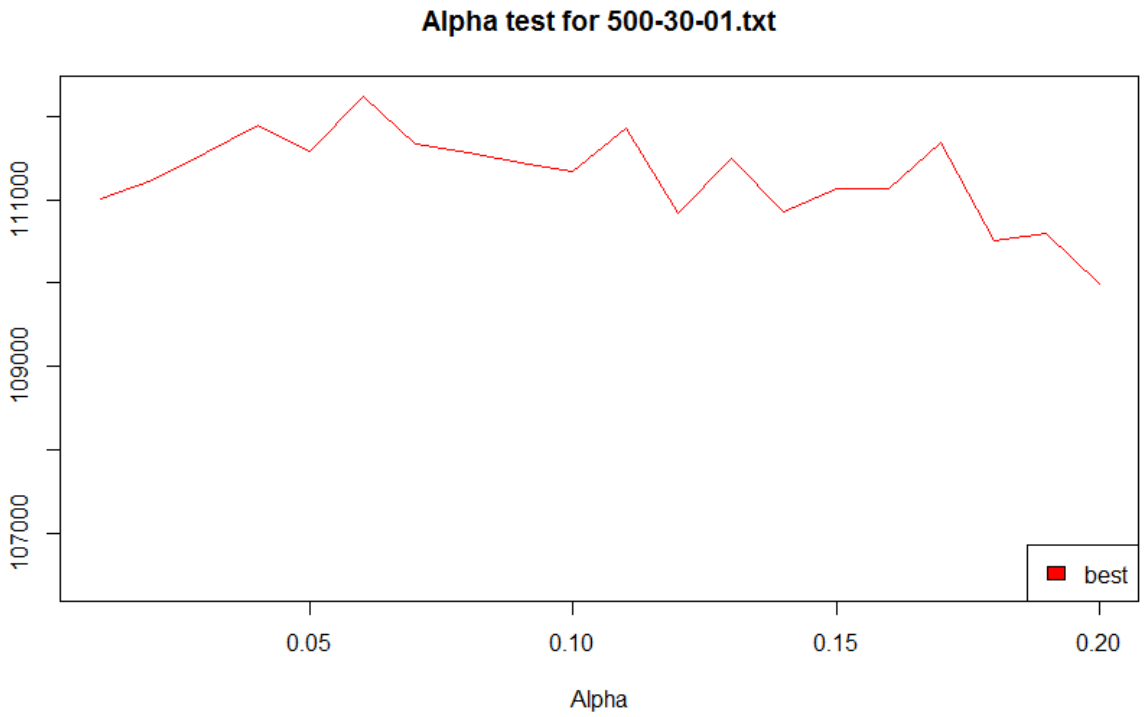




We assumed that alpha value should be selected from the interval (0, 0.2). The following plots show the best value found during the same tests but a bit closer.


Alpha test for 100-5-01.txt
24000
23500
23000
22500
0.05
0.10
0.15
0.20
Alpha
best


Alpha test for 250-10-01.txt
58000
57000
56000
55000
54000
0.05
0.10
0.15
0.20
Alpha
best



Once again we can't say that one value is the best for all instances but we can see that usually values from the interval [0.05, 0.1] provide good results. We decided to choose 0.07 as alpha value. It was changed several times for testing but the deviation wasn't very high.

Testing different rates(easy is static, normalized; standard is dynamic, normalized; norm is dynamic, L2-normalized):

easy	standard	norm	easy	standard	norm
24329	24381	24373	22856,7	23750,2	23507,9
24274	24274	24274	23389,7	23652,3	23374
23538	23538	23523	22573,8	23015,2	22891
22732	23484	23332	19547,7	22828,4	22627,6
23818	23991	23991	22252	23302,4	23313,8
57791	58690	58736	55662,7	57557,2	57597,2
58076	58513	58183	56217	57608,5	57238,3
57154	57727	57489	54975,5	56644	56331,9
60300	60608	60398	58020,8	59634,5	59343,1
56018	57776	57554	52670,7	56427,4	56590,3
111634	113630	113694	109016	112076	112511
112174	113002	113148	109792	111648	111807
113228	114335	114828	110470	112950	113542
110855	113221	113248	107098	111388	112025
111462	113861	114489	108706	112234	113219

It seems that dynamic, normalized rate is better for small problems (n <= 250), while dynamic, L2-normalized rate is good for larger problems (n = 500). But we need to improve the output of GRASP with the improvement heuristics. The improvement heuristic we implemented during the work on the assignment 2 was able to improve the output of GRASP with dynamic, normalized rate better than the output of GRASP with dynamic, L2-normalized rate. That is why dynamic, normalized rate is used for GRASP in the final heuristics.

To make our results at least as good as in assignment 2 we decided to leave the heuristic from asigment 2 and add GRASP for the free time. The heuristics from assignment 2 works not more than 10 seconds and the limit is 60 seconds. That means we have 50 seconds for GRASP.

The pseudocode for the heuristic from assignment 2 with added GRASP:

1: solution1 := Construction with static, normalized rate

2: Improve solution1 using the method (4j <= 100)-3j-2j-1j.

3: solution2 := Construction with dynamic, normalized rate

4: Improve solution2 using the method (4j <= 250)-3j-2j-1j.

5: solution3 := Construction with dynamic, L2-normalized rate

6: Improve solution3 using the method (4j <= 100)-3j-2j-1j.

7: solution4 := The best from obtained with GRASP

8: Improve solution4 using the method (4j <= 100)-3j-2j-1j.

8: result := the best solution among solution1, solution2, solution3, solution 4.

We tested this metaheuristic using different options for GRASP. Firstly, we tested different alpha values but the deviation from the results obtained with alpha was very low. And we failed to find a better value for alpha.

Secondly we tested different options of improving GRASP solutions. The main idea is that we can run GRASP for some time, obtain many solutions and then improve only the best of them.

The options we tested are:

- Improve only one result of GRASP
- Improve 5 results of GRASP
- Improve 7 results of grasp

We tested this options with the L2-normalized rate and decided that this rate doesn't work well enough.

Here are the results:

Improve best grasp	Improve 5 best	Improve 7 best
24373	24373	24373
24274	24274	24274
23538	23538	23538
23325	23325	23350
23991	23991	23991
58710	58785	58915
58355	58571	58527
57710	57710	57786
60446	60547	60575
57839	57839	57839
114302	114536	114694
113284	113709	113709
115319	115319	115319

113732	114036	114045
115246	115246	115246

It's better to run grasp many times for smaller period of time and then improve the results. We started testing of running grasp for a fixed period of time, improving the best solution obtained and then repeating this procedure while we have time left in a minute.

We also tried faster improvement for instances where n = 250. We refused to use 4j part of the improvement for them so that we can run more iterations of GRASP for a period+Improvement, but the results did not become any better. We came back to using 4j-search for instances where n=250. We tested running GRASP for different periods before improving:

3 sec	1 sec	0.5 sec	0.1 sec	1 iteration
24381	24381	24381	24381	24381
24274	24274	24274	24274	24274
23538	23551	23551	23551	23538
23477	23534	23534	23534	23534
23991	23991	23991	23991	23991
59021	59041	59041	59041	59021
58629	58629	58629	58629	58629
57821	57888	57888	57888	57954
60875	60875	60875	60875	60875
57801	57946	57946	57946	57830
115154	115154	115154	115154	115154
114246	114246	114246	114246	114246
116228	116228	116228	116228	116228
114645	114645	114645	114645	114645
115928	115928	115928	115928	115928

We decided that running GRASP for 1 second and then improving is good enough.

We also tried improving GRASP with tabu search but it wasn't a good idea ☺.

The final pseudocode of the metaheuristic is:

1: solution1 := Construction with static, normalized rate

2: Improve solution1 using the method (4j <= 100)-3j-2j-1j.

3: solution2 := Construction with dynamic, normalized rate

4: Improve solution2 using the method (4j <= 250)-3j-2j-1j.

5: solution3 := Construction with dynamic, L2-normalized rate

6: Improve solution3 using the method (4j <= 100)-3j-2j-1j.

7: solution4 := initial solution

8: **while** total running time < 50 seconds **do**

9: Run GRASP for 1 second and select the best solution x

10: Improve x using the method (4j <= 100)-3j-2j-1j.

11: **if** x is better than solution4: solution4 <= x

12: **end while**

13: result := the best solution among solution1, solution2, solution3, solution4.

6. Results

Assignment 2	Assignment 3	Difference
24381	24381	0
24274	24274	0
23496	23551	55
23389	23534	145
23991	23991	0
59021	59041	20
58629	58629	0
57821	57888	67
60875	60875	0
57801	57946	145
115154	115154	0
114246	114246	0
116228	116228	0
114645	114645	0
115928	115928	0

We were able to improve only some results using GRASP. With the different random seed some other solutions can be improved. The extending of running time could be beneficial for our metaheuristic.

The times in the table above are obtained using processor “Intel Core i7-4720HQ CPU @ 2.60 GHz”. Random seed id 1479.