Multidimensional Knapsack Problem

Ubeda A. Keneth[1]

[1]Université libre de Bruxelles, Belgium

kubedaar@ulb.ac.be

1. Introduction

The main idea of this experiment is trying to solve the multidimensional knapsack problem applying a few different constructive heuristics and some improvements based on the constructive heuristics as initial solution. First of all the MKP is going to be briefly described.
The 0–1 Knapsack problem (KP) consists of placing a finite number of item in a knapsack, in which each item has an associated value and volume; the solution should not exceed the capacity of the knapsack and most important obtain the highest possible profit. Therefore the MKP is based in the same idea as KP but allows to have more than one knapsack within the problem. This means that the MKP consists on n items, m knapsack. Due the multidimensional property the problem needs n * m constraints being careful of the knapsacks capacity. So the restriction for one item to be part of the solution is to not violate any constraint. An important point is the fact that the constraints are constructed based on the capacities and the weight or volume the item takes on the knapsack. One last important detail about the MKP is its objective function.

$$max \sum_{i=1}^{n} P_i \chi_i \tag{1}$$

Going back to the experiment, there are two things to measure throughout the performance of the heuristics presented in this report: relative percentage deviation respect to the best know values and the time each algorithm takes getting a solution. Due to the complexity of the problem it is important to find a trade off between profit and time because often a better solution quality means lots of computation time. Regarding the improvements the impact of the initial solution over the improvement is going to be discussed.

The algorithms have been developed using Java 1.8, in the Heuristics more detail is going to be given. And for the performance tests Java Microbenchmark Harness (JMH) has been used, also more detial is going to be given in the Experiment section.

2. Heuristics

Three constructive heuristics (CH) which are Random, Greedy and Toyoda have been implemented as well as three different improvements which are First Improvement (FI), Best Improvement (BI) and Variable Neighborhood Descent (VND) based on the First improvement. The CH starts with an empty solution and outputs a feasible solution. The improvements starts with an initial solution which in this case will be the first three CH. Doing the concerned combinations 12 different methods will be discussed.

2.1. Constructive Heuristics

The pseudo code for the CH algorithms are explained below.

2.1.1. Random

The Random CH starts with an empty solution, then items are shuffled, finally it tries to add one by one item and it only adds the ones which don't violate any constraint.

```

public RandomSolution getFeasibleSolution() {
        int v[] = createShuffled();
        for (int i = 0; i < super.getpIns().getItems(); i++) {
            super.checkBeforeAddItem(v[i]);
        }
        return this;
}
```

2.1.2. Greedy

The Greedy CH differs from the Random in the fact that rather than shuffle the items, it sorts the items based on the profit, from the most profit to the lowest, and tries to add item by item and it only adds the ones which don't violate any constraint.

```

public GreedySolution getFeasibleSolution() {
    int v[] = this.sortByProfit();
    for (int i = 0; i < super.getpIns().getItems(); i++) {
        super.checkBeforeAddItem(v[i]);
    }
     return this;
}
```

2.1.3. Toyoda

Toyoda is an algorithm more complex algorithm compared to the previous ones, it has to perform more steps. The first one is to normalize the constraints in the 0-1 range, the calculate U the vector of resources used so far, subsequently calculate the vector V which is the pseudo-utility, then it tries to add item by item until it is not possible anymore to add items.

```

public ToyodaSolution getFeasibleSolution() {
    this.normalizeConstraints();
    boolean again = true;
    for (int i = 0; again; i++) {
        // step 4 calculate "u"
        this.computeResourcesUsed(i);
        // step 5 calculate "v"
        // step 6 if first iteration  v = a
        this.computeV(i);
        // step 7 sort non-selected items accorging to the pseudoutility
        this.sortAndComputePseudoUtility();
        // step 8 scan the list until you can add one item
        // step 9 when you add one item, stop the cycle,
        // update and repeat teh process until you cannot add any new item
        again = this.addItem();
    }
    return this;
}
```

2.2. Improvements

The pseudo code for the improvements algorithms are explained below.

2.2.1. First Improvement

The first improvement attempt to improves a initial solution, in this case three different ones. The important points to mention is the fact that for this experiments the initial solution is shuffled and the items not in the solution are sorted depending on the initial solution, Random: sorted in random order; Greedy: sorted from the highest profit to the lowest; Toyoda: sorted based on the pseudo-utility. The principal characteristic of this algorithm is that after each movement it verifies whether there is or not an improvement, If there is it changes the initial solution and starts over again until there isn't anymore and improvement.

```

public Solution getImprovedSolution() {
    while (super.improved) {
        super.improved = false;
        // shuffle items in s
        super.shuffleInitialSolution();
        // sort list of non-inserted items
        super.sortNonInsertedInSolution();
        for (int i = 0; i < super.lkInitialSolution.size(); i++) {
            this.tmpSol = super.initSolution.copy();
            // remove k items from shuffled s
            this.tmpSol.removeItem(super.lkInitialSolution.get(i));
            // try to add non-selected items, one by one, from the sorted list
            super.tmpPqueue = new PriorityQueue<>(super.sortedNonInSolution);
            while (!tmpPqueue.isEmpty()) {
                // check feasibility
                this.tmpSol.checkBeforeAddItem(tmpPqueue.poll().getKey());
            }
            // check improvement
            if (this.tmpSol.getValue() > super.solution.getValue()) {
```

```java
                super.solution = this.tmpSol.copy();
                super.initSolution = super.solution.copy();// Apply move
                super.improved = true;
                break;
            }
        }
    }
    return super.solution;
}
```

2.2.2. Best Improvement

The first steps for the BI are the same as FI the only difference between these two algorithms is the moment they changed the initial solution to start over again. This algorithm rather than accept an improvement when it finds one it memorize the new solution and changes after finish to explore all the neighborhood. Due to this the algorithm can gives better quality solution but less efficient.

```java

public Solution getImprovedSolution() {
    while (super.improved) {
        super.improved = false;
        // shuffle items in s
        super.shuffleInitialSolution();
        // sort list of non-inserted items
        super.sortNonInsertedInSolution();
        for (int i = 0; i < super.lkInitialSolution.size(); i++) {
            this.tmpSol = super.initSolution.copy();
            // remove k=1 items from shuffled s
            this.tmpSol.removeItem(super.lkInitialSolution.get(i));
            // try to add non-selected items, one by one, from the sorted list
            super.tmpPqueue = new PriorityQueue<>(super.sortedNonInSolution);
            while (!tmpPqueue.isEmpty()) {
                // check feasibility
                this.tmpSol.checkBeforeAddItem(tmpPqueue.poll().getKey());
            }
            // check improvement
            if (this.tmpSol.getValue() > super.solution.getValue()) {
                super.solution = this.tmpSol.copy();
                super.improved = true;
            }
        }
        super.initSolution = super.solution.copy();// Apply Move
    }
    return super.solution;
}
```

2.2.3. VND First Improvement

This VND is based on the First improvement but per each movement this algorithm is going to try remove from the solution not only one item but 1, 2 and 3, all combinations are possible. It will change from 1 to 2 and from 2 to 3 when the actual k stops to improve. This increase of the search space yield very good quality answers but once the items in the solution increases, this algorithm can takes long time of computations due to try all possible over k elements.

```java
public Solution getImprovedSolution() {
    for (int ki = 0; ki < this.k; ki++) { // k is here
        super.improved = true;
        while (super.improved) {
            super.improved = false;
            // shuffle items in s
            super.shuffleInitialSolution();
            // sort list of non-inserted items
            super.sortNonInsertedInSolution();
            // get all possible combinations
            this.getAllPossibleCombinations(
                    super.lkInitialSolution.size(), ki + 1, lkInitialSolution
            );
            for (int i = 0; i < this.allPossibleCombinations.size(); i++) {
                this.tmpSol = super.initSolution.copy();
                // remove k=1,2,3 items from shuffled s
                for (int j = 0; j < this.allPossibleCombinations.get(i).size(); j++) {
                    this.tmpSol.removeItem(
                            this.allPossibleCombinations.get(i).get(j)
                    );
                }
                // try to add non-selected items, one by one, from the sorted list
                super.tmpPqueue = new PriorityQueue<>(super.sortedNonInSolution);
                while (!tmpPqueue.isEmpty()) {
                    // check feasibility
                    this.tmpSol.checkBeforeAddItem(tmpPqueue.poll().getKey());
                }
                // check improvement
                if (this.tmpSol.getValue() > super.solution.getValue()) {
                    super.solution = this.tmpSol.copy();
                    super.initSolution = super.solution.copy();// Apply Move
                    super.improved = true;
                    break;
                }
            }
        }
    }
    return super.solution;
}
```

3. Experiment and results using JMH benchmark

JMH was used to measure the computation time taken by the different algorithms. JMH is an OpenJDK project that aims to facilitate setting up a benchmark environment for Java performance tests. the tests have been automatized using JMH and it is important to mention that because the Random CH is an non deterministic algorithm the results showed below is the result of 15 trials using different randoms seeds.
The other paremeter measured is the solution quality, for this the Relative percentage deviation has been used.

$$\triangle_{ki} = 100 \cdot \frac{profit_{ki} - bestknown_i}{bestknown_i} \tag{2}$$

Table 1. Computer specifications

Type	Compute optimized AWS EC2 mc5.large
CPU	Intel Xeon Platinum 8124M 2 cores
VCPU	2
Clock speed	3 GHz
RAM	4GB
OS	TheAmazon Linux AMI OS
Java	Java 1.8

These are the random seed used for the 15 executions for the Random based heuristics. and the seed for the shuffle step in the improvements.

```
    private final static Long IMPROVEMENT_SEED = 4619L;
    private final static Long RANDOM_SEEDS[] = {
        7440L, 1437L, 1279L, 1263L,
        1872L, 1252L, 1394L, 1015L,
        1009L, 3882L, 1130L, 4602L,
        1047L, 1353L, 2952L
    };
```

One more important point to mention is the instances description. For this experiment 60 different instances were provided, these instances are divided in two groups: 10 knapsack and 100 items (10x100) and 10 knapsacks and 250 items (10x250). Each group will be discussed independently.

3.1. Constructive Heuristics

In order to compare the CH a Wilcoxon test with Bonferroni correction has been done also there are summary tables showing the global results of the tests.

3.1.1. 10x100 Instances

Table 2. Average Results Constructive Heuristics 10x100 Instance group

	t (ms) avg	Δavg
Random	0.0281	18.31
Greedy	0.0353	9.44
Toyoda	0.5296	3.56

This table shows that in average the Toyoda has obtained better quality solutions but also has taken the most time.

Table 3. Constructive Heuristics 10x100 Instance group

	Random		Greedy		Toyoda	
Instance	**t (ms)**	**Δavg**	**t (ms)**	**Δ**	**t (ms)**	**Δ**
OR10x100-0.25_1	**0.0296**	24.70	0.0425	14.13	1.3686	**4.80**
OR10x100-0.25_2	**0.0296**	28.52	0.0377	23.90	1.0461	**6.12**
OR10x100-0.25_3	**0.0296**	27.92	0.0363	21.88	0.6714	**7.62**
OR10x100-0.25_4	**0.0284**	24.58	0.0361	9.66	0.4192	**8.96**
OR10x100-0.25_5	**0.0270**	27.30	0.0371	11.45	0.4584	**8.28**
OR10x100-0.25_6	**0.0265**	23.94	0.0346	14.87	0.4244	**3.41**
OR10x100-0.25_7	**0.0284**	26.45	0.0347	14.65	0.4183	**6.39**
OR10x100-0.25_8	**0.0279**	27.40	0.0344	12.71	0.4222	**6.07**
OR10x100-0.25_9	**0.0271**	27.02	0.0346	14.08	0.5345	**5.85**
OR10x100-0.25_10	**0.0284**	26.16	0.0349	19.58	0.4363	**2.92**
OR10x100-0.50_1	**0.0278**	18.26	0.0349	7.77	0.5445	**2.74**
OR10x100-0.50_2	**0.0278**	16.93	0.0350	6.38	0.6018	**2.77**
OR10x100-0.50_3	**0.0277**	17.79	0.0349	9.65	0.4765	**1.80**
OR10x100-0.50_4	**0.0284**	18.46	0.0345	8.90	0.5709	**4.21**
OR10x100-0.50_5	**0.0277**	18.14	0.0351	9.43	0.4661	**2.40**
OR10x100-0.50_6	**0.0267**	17.75	0.0349	10.27	0.4740	**2.70**
OR10x100-0.50_7	**0.0282**	18.29	0.0347	6.77	0.4519	**1.51**
OR10x100-0.50_8	**0.0281**	17.90	0.0349	4.82	0.4365	**3.35**
OR10x100-0.50_9	**0.0290**	20.17	0.0348	9.28	0.4445	**1.47**
OR10x100-0.50_10	**0.0275**	16.84	0.0347	8.54	0.4218	**2.85**
OR10x100-0.75_1	**0.0277**	10.63	0.0346	4.73	0.6244	**1.87**
OR10x100-0.75_2	**0.0270**	10.77	0.0347	5.37	0.5616	**2.44**
OR10x100-0.75_3	0.0280	10.51	0.0346	4.40	0.4583	**2.00**
OR10x100-0.75_4	0.0286	9.29	0.0349	**3.65**	0.4555	4.41
OR10x100-0.75_5	0.0277	11.70	0.0347	3.38	0.4497	**0.98**
OR10x100-0.75_6	0.0274	10.06	0.0348	4.57	0.4469	**1.41**
OR10x100-0.75_7	0.0279	10.37	0.0344	4.63	0.4569	**1.61**
OR10x100-0.75_8	0.0292	9.74	0.0349	6.01	0.4532	**1.46**
OR10x100-0.75_9	0.0294	10.80	0.0345	4.53	0.4399	**2.01**
OR10x100-0.75_10	0.0294	11.01	0.0356	3.17	0.4542	**2.28**

It is easy to see in this table the behavior of the three different algorithms across all the instances highlighting the lowest deviation values and the lowest computation time values which corresponds to Toyoda and Random respectively. Also it is possible to see that the Greedy deviation values are lower than the Random deviation values.

A Wilcoxon Signed-ranks test indicates that Toyoda get a lower deviation compared to greedy and random. At the same time Random Greedy also report a significant difference between them, being Greedy who reports lower deviation values compared with Random. As the Box plot shows where the distributions of Random and Greedy overlaps a little, while Toyoda is clearly lower than Greedy.

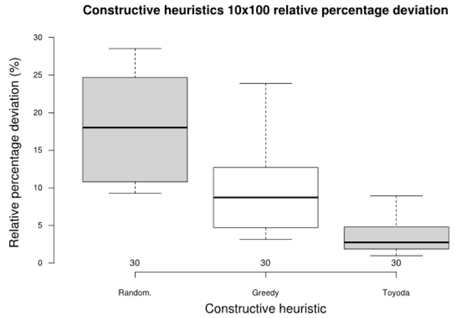


Figure 1. Percentage Deviation

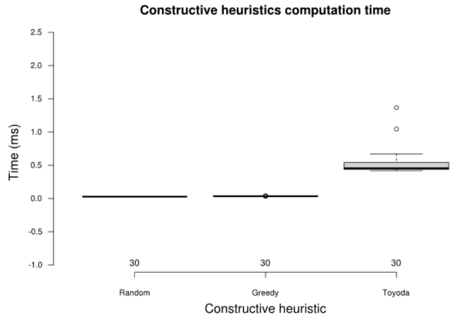


Figure 2. Computation time

A Wilcoxon Signed-ranks test indicates that Toyoda get a lower higher computation time compared to Greedy and Random. At the same time Random Greedy also report a significant difference between them. As the Box plot shows where the distributions of Random is clearly above Greedy and Random

3.1.2. 10x250 Instances

Table 4. Average Results Constructive Heuristics 10x250 Instance group

	t (ms) avg	∆avg
Random	0.0748	17.57
Greedy	0.0959	7.97
Toyoda	1.2392	2.67

This table shows that in average the Toyoda has obtained better quality solutions but also has taken the most time.

Table 5. Constructive Heuristics 10x250 Instance group

	Random		Greedy		Toyoda	
Instance	**t (ms)**	**∆avg**	**t (ms)**	**∆**	**t (ms)**	**∆**
OR10x250-0.25_1	**0.0739**	24.72	0.0963	14.25	1.1509	**3.56**
OR10x250-0.25_2	**0.0737**	25.77	0.0961	10.96	1.1420	**2.76**
OR10x250-0.25_3	**0.0738**	26.37	0.1059	17.06	1.1559	**4.24**
OR10x250-0.25_4	**0.0739**	25.54	0.0959	11.77	1.1628	**2.24**
OR10x250-0.25_5	**0.0739**	25.58	0.0971	13.41	1.1553	**6.32**
OR10x250-0.25_6	**0.0740**	24.96	0.0960	11.52	1.1608	**5.76**
OR10x250-0.25_7	**0.0740**	24.48	0.0959	12.57	1.1652	**3.69**
OR10x250-0.25_8	**0.0740**	26.48	0.0967	17.26	1.1569	**4.21**
OR10x250-0.25_9	**0.0742**	27.39	0.0954	11.82	1.1514	**3.96**
OR10x250-0.25_10	**0.0741**	25.44	0.0950	14.78	1.1838	**2.39**
OR10x250-0.50_1	**0.0743**	16.38	0.0972	6.84	1.2313	**3.06**
OR10x250-0.50_2	**0.0754**	17.13	0.0961	5.54	1.2473	**3.06**
OR10x250-0.50_3	**0.0746**	17.56	0.0965	7.09	1.2404	**1.83**
OR10x250-0.50_4	**0.0746**	16.87	0.0956	9.79	1.2435	**4.88**
OR10x250-0.50_5	**0.0744**	16.32	0.0948	6.22	1.2384	**2.30**
OR10x250-0.50_6	**0.0745**	17.68	0.0959	8.80	1.2561	**3.51**
OR10x250-0.50_7	**0.0747**	18.32	0.0947	6.98	1.2436	**2.16**
OR10x250-0.50_8	**0.0747**	17.53	0.0943	6.76	1.2364	**2.68**
OR10x250-0.50_9	**0.0755**	17.61	0.0955	7.09	1.3543	**3.38**
OR10x250-0.50_10	**0.0757**	17.66	0.0958	5.99	1.2434	**2.45**
OR10x250-0.75_1	**0.0751**	9.43	0.0944	3.48	1.3054	**0.99**
OR10x250-0.75_2	**0.0751**	10.01	0.0950	3.68	1.3083	**1.58**
OR10x250-0.75_3	**0.0751**	9.70	0.0958	3.53	1.2796	**1.48**
OR10x250-0.75_4	**0.0750**	9.90	0.0942	4.12	1.3022	**1.09**
OR10x250-0.75_5	**0.0754**	9.80	0.0959	3.41	1.3059	**1.44**
OR10x250-0.75_6	**0.0759**	9.71	0.0955	2.96	1.3130	**0.70**
OR10x250-0.75_7	**0.0756**	9.44	0.0952	3.08	1.3045	**1.12**
OR10x250-0.75_8	**0.0765**	9.63	0.0942	3.15	1.3107	**1.44**
OR10x250-0.75_9	**0.0758**	9.89	0.0943	2.82	1.2932	**0.84**
OR10x250-0.75_10	**0.0762**	9.86	0.0961	2.28	1.3331	**1.11**

It is easy to see in this table the behavior of the three different algorithms across all the instances highlighting the lowest deviation values and the lowest computation time values which corresponds to Toyoda and Random respectively. Also it is possible to see that the Greedy deviation values are lower than the Random deviation values.

A Wilcoxon Signed-ranks test indicates that Toyoda get a lower deviation compared to greedy and random. At the same time Random Greedy also report a significant difference between them, being Greedy who reports lower deviation values compared with Random. As the Box plot shows where the distributions of Random and Greedy overlaps a little, while Toyoda is clearly lower than Greedy.

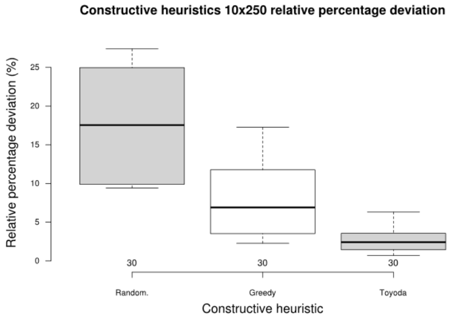


Figure 3. Percentage Deviation

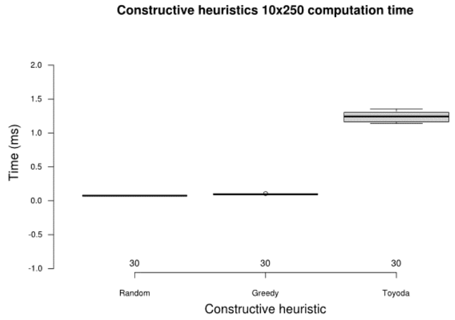


Figure 4. Computation time

A Wilcoxon Signed-ranks test indicates that Toyoda get a lower higher computation time compared to Greedy and Random. At the same time Random Greedy also report a significant difference between them. As the Box plot shows where the distributions of Random is clearly above Greedy and Random

3.2. Improvements 10x100 instances

Table 6. Average results 10x100 instance group

	Frist Improvement		Best Improvement		VND First Improvement	
Initial Solution	**t (ms) avg**	**∆avg**	**t (ms) avg**	**∆avg**	**t (ms) avg**	**∆avg**
Random	3.2848	5.4865	9.3484	5.4378	33649.7945	2.6689
Greedy	1.5491	4.5022	3.1126	4.3740	21814.7773	1.6431
Toyoda	2.2412	1.5185	4.0010	1.0817	50521.8906	0.5432

In this table it is possible to compare among the three different improvements. It shows that the FI reports the lower average computation time values. BI reports also lower values than VND, but the computation time of VND compared with FI and BI is way higher. For the deviation Toyoda is showing lower average values while there isn't a big difference between the average values between FI and BI.

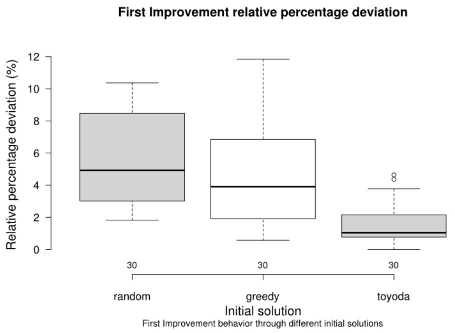
Table 7. Improvements 10x100 instance group

	First Improvement						Best Improvement						VND First Improvement					
	Random		Greedy		Toyoda		Random		Greedy		Toyoda		Random		Greedy		Toyoda	
Instance	**t (ms)**	**∆avg**	**t (ms)**	**∆**	**t (ms)**	**∆**	**t (ms)**	**∆avg**	**t (ms)**	**∆**	**t (ms)**	**∆**	**t (ms)**	**∆avg**	**t (ms)**	**∆**	**t (ms)**	**∆**
OR10x100-0.25_1	3.25	9.00	**1.07**	8.10	3.88	**0.00**	7.41	8.64	2.43	8.17	3.71	1.87	111.72	4.21	279.73	1.52	275.91	**0.00**
OR10x100-0.25_2	3.18	7.69	2.77	6.28	2.75	2.93	7.95	9.03	**2.31**	8.85	3.75	**0.54**	113.18	4.31	74.27	6.10	250.80	1.48
OR10x100-0.25_3	3.00	9.82	**1.33**	9.19	1.78	4.36	7.14	8.46	1.63	12.94	4.52	1.55	105.61	5.39	47.64	3.38	227.47	**0.62**
OR10x100-0.25_4	2.95	9.14	**0.64**	7.35	1.17	4.64	6.73	8.18	1.18	6.45	5.68	2.56	101.15	4.41	78.02	3.16	197.17	**0.65**
OR10x100-0.25_5	3.13	10.37	1.68	7.90	1.23	3.77	7.32	9.55	**1.15**	8.55	3.70	2.65	111.10	5.19	69.07	2.18	244.50	**2.12**
OR10x100-0.25_6	2.87	8.48	**1.32**	6.90	1.99	2.11	6.52	8.18	1.72	7.92	4.69	1.74	97.55	4.48	127.07	2.99	167.85	**1.37**
OR10x100-0.25_7	3.08	9.04	**1.75**	6.85	1.82	2.64	7.43	8.03	4.17	3.31	2.25	2.35	115.01	5.39	175.44	1.66	280.80	**1.15**
OR10x100-0.25_8	2.89	9.65	**1.63**	6.41	2.14	2.16	6.52	8.91	3.01	4.17	2.23	3.19	103.05	4.80	58.30	1.94	375.80	**0.72**
OR10x100-0.25_9	3.16	9.00	1.60	8.30	**1.26**	2.74	7.46	9.06	2.37	6.12	6.88	1.72	107.88	4.23	65.22	3.34	359.81	**0.59**
OR10x100-0.25_10	3.17	8.05	**1.10**	11.84	1.28	2.88	7.72	9.09	2.22	12.52	3.96	1.20	98.88	5.33	67.76	3.17	312.70	**1.04**
OR10x100-0.50_1	3.98	5.53	2.28	2.69	**2.06**	1.40	12.10	6.02	3.40	4.23	4.13	1.19	4047.79	1.98	1376.36	1.74	4632.48	**0.35**
OR10x100-0.50_2	4.38	4.08	**1.25**	3.22	3.48	1.03	13.27	4.96	3.39	2.24	4.75	0.78	2571.14	2.11	2119.83	0.83	5167.70	**0.58**
OR10x100-0.50_3	3.95	5.50	**2.46**	4.12	3.12	1.17	10.90	6.09	3.34	5.17	2.75	1.00	3654.69	2.47	4531.73	0.98	4957.47	**0.73**
OR10x100-0.50_4	4.62	3.74	**1.52**	6.41	5.18	1.43	14.94	3.96	2.54	6.32	6.77	**0.40**	3567.86	2.25	1968.57	3.41	10300.80	0.57
OR10x100-0.50_5	4.13	4.65	**1.09**	4.12	2.60	**0.45**	12.97	4.47	6.80	1.70	4.14	**0.45**	4004.88	2.37	8563.77	0.55	3193.61	**0.45**
OR10x100-0.50_6	4.28	4.76	1.91	5.38	**1.65**	1.84	12.34	4.18	3.34	5.84	4.11	1.41	3825.00	2.21	3816.31	1.90	13756.11	**0.22**
OR10x100-0.50_7	4.16	4.60	**1.07**	3.70	3.78	0.98	13.08	4.70	3.35	3.03	4.08	1.02	3364.68	2.53	3048.86	1.30	6392.79	**0.25**
OR10x100-0.50_8	4.36	5.59	**0.93**	3.64	2.41	1.06	12.80	4.96	2.51	2.55	5.54	0.67	3663.90	2.20	1470.44	1.50	4591.58	**0.34**
OR10x100-0.50_9	3.98	5.08	2.46	4.13	**2.11**	0.60	11.82	5.96	4.17	3.87	2.76	0.80	3519.55	2.53	3835.14	1.57	6132.29	**0.14**
OR10x100-0.50_10	3.81	5.52	**1.68**	3.58	3.05	0.84	11.13	5.62	9.20	2.58	4.13	1.03	3933.72	2.56	2686.11	1.19	6581.31	**0.60**
OR10x100-0.75_1	2.67	1.96	1.48	1.63	**1.01**	0.81	8.29	2.60	2.06	1.39	3.72	**0.33**	123196.29	0.78	47650.39	0.60	280364.20	**0.33**
OR10x100-0.75_2	2.68	2.39	**1.52**	1.91	1.90	0.77	8.70	2.50	2.84	2.44	2.95	0.60	102785.55	0.79	47042.04	**0.54**	86363.72	0.55
OR10x100-0.75_3	2.36	3.01	2.28	0.90	**2.13**	0.36	7.24	2.41	4.81	1.63	4.74	0.37	95849.22	0.88	208464.15	0.50	148431.35	**0.27**
OR10x100-0.75_4	2.52	2.51	1.41	1.33	**1.15**	0.81	7.76	2.44	2.08	1.25	4.07	0.34	95573.72	1.31	42576.53	0.25	176541.43	**0.12**
OR10x100-0.75_5	2.79	2.66	**0.88**	2.07	1.87	0.77	9.49	2.27	1.37	1.31	3.08	0.41	71817.60	0.94	42470.62	0.33	120964.13	**0.22**
OR10x100-0.75_6	2.51	2.70	**0.92**	2.38	1.84	0.79	7.73	2.76	3.53	2.37	3.76	0.62	96969.06	0.86	47250.25	0.43	76045.54	**0.26**
OR10x100-0.75_7	2.51	3.21	**0.99**	1.86	2.58	0.44	7.53	2.53	2.81	1.22	1.91	0.50	83084.13	0.99	36179.49	1.09	140459.36	**0.38**
OR10x100-0.75_8	2.56	3.16	1.86	1.45	**1.70**	0.38	7.58	3.08	3.52	1.67	3.95	0.35	104980.26	0.87	43329.47	0.26	91304.34	**0.12**
OR10x100-0.75_9	2.79	1.85	**1.95**	0.85	1.95	1.21	9.26	2.13	3.46	1.02	5.43	0.43	105553.23	0.83	53143.05	0.47	225281.35	**0.00**
OR10x100-0.75_10	2.85	1.82	**1.64**	0.57	2.36	0.21	9.32	2.36	2.67	0.39	1.89	0.38	92466.44	0.84	51877.70	0.39	101502.36	**0.08**

It is possible to compare the performance among the three improvements and their initial solutions across all the instances. The lowest deviation values and lowest computation values are highlighted showing that the lowest time values are mostly in the FI. However the lower deviation values are mostly found in VND.

Table 8. Fraction of 10x100 instances that FI improves from initial solution

				First Improvement		
	Random	Greedy	Toyoda	Random	Greedy	Toyoda
Instance	**Δavg**	**Δ**	**Δ**	**Δ avg**	**Δ**	**Δ**
OR10x100-0.25_1	24.70	14.13	4.80	**9.00**	**8.10**	**0.00**
OR10x100-0.25_2	28.52	23.90	6.12	**7.69**	**6.28**	**2.93**
OR10x100-0.25_3	27.92	21.88	7.62	**9.82**	**9.19**	**4.36**
OR10x100-0.25_4	24.58	9.66	8.96	**9.14**	**7.35**	**4.64**
OR10x100-0.25_5	27.30	11.45	8.28	**10.37**	**7.90**	**3.77**
OR10x100-0.25_6	23.94	14.87	3.41	**8.48**	**6.90**	**2.11**
OR10x100-0.25_7	26.45	14.65	6.39	**9.04**	**6.85**	**2.64**
OR10x100-0.25_8	27.40	12.71	6.07	**9.65**	**6.41**	**2.16**
OR10x100-0.25_9	27.02	14.08	5.85	**9.00**	**8.30**	**2.74**
OR10x100-0.25_10	26.16	19.58	2.92	**8.05**	**11.84**	**2.88**
OR10x100-0.50_1	18.26	7.77	2.74	**5.53**	**2.69**	**1.40**
OR10x100-0.50_2	16.93	6.38	2.77	**4.08**	**3.22**	**1.03**
OR10x100-0.50_3	17.79	9.65	1.80	**5.50**	**4.12**	**1.17**
OR10x100-0.50_4	18.46	8.90	4.21	**3.74**	**6.41**	**1.43**
OR10x100-0.50_5	18.14	9.43	2.40	**4.65**	**4.12**	**0.45**
OR10x100-0.50_6	17.75	10.27	2.70	**4.76**	**5.38**	**1.84**
OR10x100-0.50_7	18.29	6.77	1.51	**4.60**	**3.70**	**0.98**
OR10x100-0.50_8	17.90	4.82	3.35	**5.59**	**3.64**	**1.06**
OR10x100-0.50_9	20.17	9.28	1.47	**5.08**	**4.13**	**0.60**
OR10x100-0.50_10	16.84	8.54	2.85	**5.52**	**3.58**	**0.84**
OR10x100-0.75_1	10.63	4.73	1.87	**1.96**	**1.63**	**0.81**
OR10x100-0.75_2	10.77	5.37	2.44	**2.39**	**1.91**	**0.77**
OR10x100-0.75_3	10.51	4.40	2.00	**3.01**	**0.90**	**0.36**
OR10x100-0.75_4	9.29	3.65	4.41	**2.51**	**1.33**	**0.81**
OR10x100-0.75_5	11.70	3.38	0.98	**2.66**	**2.07**	**0.77**
OR10x100-0.75_6	10.06	4.57	1.41	**2.70**	**2.38**	**0.79**
OR10x100-0.75_7	10.37	4.63	1.61	**3.21**	**1.86**	**0.44**
OR10x100-0.75_8	9.74	6.01	1.46	**3.16**	**1.45**	**0.38**
OR10x100-0.75_9	10.80	4.53	2.01	**1.85**	**0.85**	**1.21**
OR10x100-0.75_10	11.01	3.17	2.28	**1.82**	**0.57**	**0.21**



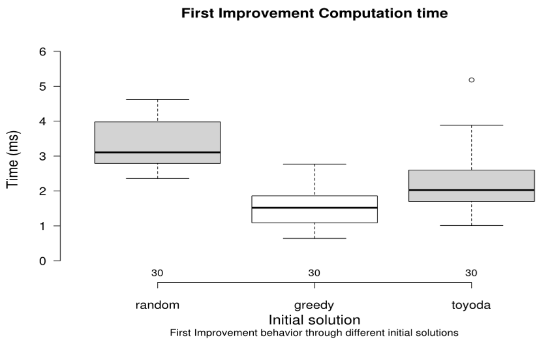


Figure 5. Percentage deviation

Figure 6. Computation time

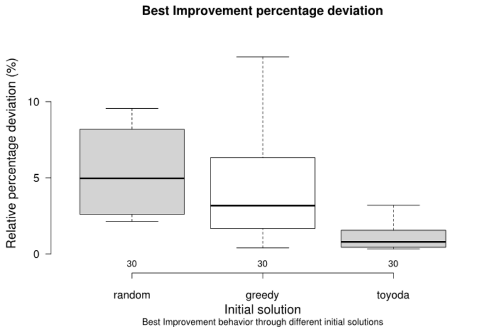
The Table above shows that all initial solutions have been improved applying the FI across all the instances.

A Wilcoxon Signed-ranks test indicates that when applying First improvement Toyoda get a lower deviation compared to greedy and random. At the same time Random and Greedy don't report a significant difference between them As the Box plot shows where the distributions of Random and Greedy overlaps , while Toyoda is clearly lower.

The test also indicates that applying First improvement among Toyoda, Random and Greedy, Greedy obtains the lower computation time values followed by Toyoda and Random with the higher values.

Table 9. Fraction of 10x100 instances that BI improves from initial solution

				Best Improvement		
	Random	Greedy	Toyoda	Random	Greedy	Toyoda
Instance	**Δavg**	**Δ**	**Δ**	**Δ avg**	**Δ**	**Δ**
OR10x100-0.25_1	24.70	14.13	4.80	**8.64**	**8.17**	**1.87**
OR10x100-0.25_2	28.52	23.90	6.12	**9.03**	**8.85**	**0.54**
OR10x100-0.25_3	27.92	21.88	7.62	**8.46**	**12.94**	**1.55**
OR10x100-0.25_4	24.58	9.66	8.96	**8.18**	**6.45**	**2.56**
OR10x100-0.25_5	27.30	11.45	8.28	**9.55**	**8.55**	**2.65**
OR10x100-0.25_6	23.94	14.87	3.41	**8.18**	**7.92**	**1.74**
OR10x100-0.25_7	26.45	14.65	6.39	**8.03**	**3.31**	**2.35**
OR10x100-0.25_8	27.40	12.71	6.07	**8.91**	**4.17**	**3.19**
OR10x100-0.25_9	27.02	14.08	5.85	**9.06**	**6.12**	**1.72**
OR10x100-0.25_10	26.16	19.58	2.92	**9.09**	**12.52**	**1.20**
OR10x100-0.50_1	18.26	7.77	2.74	**6.02**	**4.23**	**1.19**
OR10x100-0.50_2	16.93	6.38	2.77	**4.96**	**2.24**	**0.78**
OR10x100-0.50_3	17.79	9.65	1.80	**6.09**	**5.17**	**1.00**
OR10x100-0.50_4	18.46	8.90	4.21	**3.96**	**6.32**	**0.40**
OR10x100-0.50_5	18.14	9.43	2.40	**4.47**	**1.70**	**0.45**
OR10x100-0.50_6	17.75	10.27	2.70	**4.18**	**5.84**	**1.41**
OR10x100-0.50_7	18.29	6.77	1.51	**4.70**	**3.03**	**1.02**
OR10x100-0.50_8	17.90	4.82	3.35	**4.96**	**2.55**	**0.67**
OR10x100-0.50_9	20.17	9.28	1.47	**5.96**	**3.87**	**0.80**
OR10x100-0.50_10	16.84	8.54	2.85	**5.62**	**2.58**	**1.03**
OR10x100-0.75_1	10.63	4.73	1.87	**2.60**	**1.39**	**0.33**
OR10x100-0.75_2	10.77	5.37	2.44	**2.50**	**2.44**	**0.60**
OR10x100-0.75_3	10.51	4.40	2.00	**2.41**	**1.63**	**0.37**
OR10x100-0.75_4	9.29	3.65	4.41	**2.44**	**1.25**	**0.34**
OR10x100-0.75_5	11.70	3.38	0.98	**2.27**	**1.31**	**0.41**
OR10x100-0.75_6	10.06	4.57	1.41	**2.76**	**2.37**	**0.62**
OR10x100-0.75_7	10.37	4.63	1.61	**2.53**	**1.22**	**0.50**
OR10x100-0.75_8	9.74	6.01	1.46	**3.08**	**1.67**	**0.35**
OR10x100-0.75_9	10.80	4.53	2.01	**2.13**	**1.02**	**0.43**
OR10x100-0.75_10	11.01	3.17	2.28	**2.36**	**0.39**	**0.38**



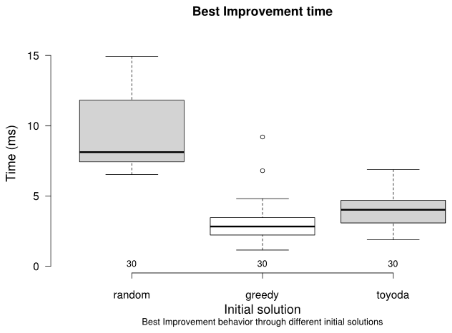


Figure 7. Percentage deviation

Figure 8. Computation time

The Table above shows that all initial solutions have been improved applying the BI across all the instances.

A Wilcoxon Signed-ranks test indicates that when applying Best improvement Toyoda gets a lower deviation compared to Greedy and Random. At the same time Random and Greedy don't report a significant difference between them as the Box plot shows where the distributions of Random and Greedy overlaps , while Toyoda is clearly lower.

The test also indicates that applying Best improvement among Toyoda, Random and Greedy, Greedy obtains the lower computation time values followed by Toyoda and Random with the higher values.

Table 10. Fraction of 10x100 instances that VND FI improves from FI

	First Improvement			VND First Improvement		
	Random	Greedy	Toyoda	Random	Greedy	Toyoda
Instance	avg	Δ	Δ	Δ avg	Δ	Δ
OR10x100-0.25_1	9.00	8.10	0.00	**4.21**	**1.52**	**0.00**
OR10x100-0.25_2	7.69	6.28	2.93	**4.31**	**6.10**	**1.48**
OR10x100-0.25_3	9.82	9.19	4.36	**5.39**	**3.38**	**0.62**
OR10x100-0.25_4	9.14	7.35	4.64	**4.41**	**3.16**	**0.65**
OR10x100-0.25_5	10.37	7.90	3.77	**5.19**	**2.18**	**2.12**
OR10x100-0.25_6	8.48	6.90	2.11	**4.48**	**2.99**	**1.37**
OR10x100-0.25_7	9.04	6.85	2.64	**5.39**	**1.66**	**1.15**
OR10x100-0.25_8	9.65	6.41	2.16	**4.80**	**1.94**	**0.72**
OR10x100-0.25_9	9.00	8.30	2.74	**4.23**	**3.34**	**0.59**
OR10x100-0.25_10	8.05	11.84	2.88	**5.33**	**3.17**	**1.04**
OR10x100-0.50_1	5.53	2.69	1.40	**1.98**	**1.74**	**0.35**
OR10x100-0.50_2	4.08	3.22	1.03	**2.11**	**0.83**	**0.58**
OR10x100-0.50_3	5.50	4.12	1.17	**2.47**	**0.98**	**0.73**
OR10x100-0.50_4	3.74	6.41	1.43	**2.25**	**3.41**	**0.57**
OR10x100-0.50_5	4.65	4.12	0.45	**2.37**	**0.55**	**0.45**
OR10x100-0.50_6	4.76	5.38	1.84	**2.21**	**1.90**	**0.22**
OR10x100-0.50_7	4.60	3.70	0.98	**2.53**	**1.30**	**0.25**
OR10x100-0.50_8	5.59	3.64	1.06	**2.20**	**1.50**	**0.34**
OR10x100-0.50_9	5.08	4.13	0.60	**2.53**	**1.57**	**0.14**
OR10x100-0.50_10	5.52	3.58	0.84	**2.56**	**1.19**	**0.60**
OR10x100-0.75_1	1.96	1.63	0.81	**0.78**	**0.60**	**0.33**
OR10x100-0.75_2	2.39	1.91	0.77	**0.79**	**0.54**	**0.55**
OR10x100-0.75_3	3.01	0.90	0.36	**0.88**	**0.50**	**0.27**
OR10x100-0.75_4	2.51	1.33	0.81	**1.31**	**0.25**	**0.12**
OR10x100-0.75_5	2.66	2.07	0.77	**0.94**	**0.33**	**0.22**
OR10x100-0.75_6	2.70	2.38	0.79	**0.86**	**0.43**	**0.26**
OR10x100-0.75_7	3.21	1.86	0.44	**0.99**	**1.09**	**0.38**
OR10x100-0.75_8	3.16	1.45	0.38	**0.87**	**0.26**	**0.12**
OR10x100-0.75_9	1.85	0.85	1.21	**0.83**	**0.47**	**0.00**
OR10x100-0.75_10	1.82	0.57	0.21	**0.84**	**0.39**	**0.08**

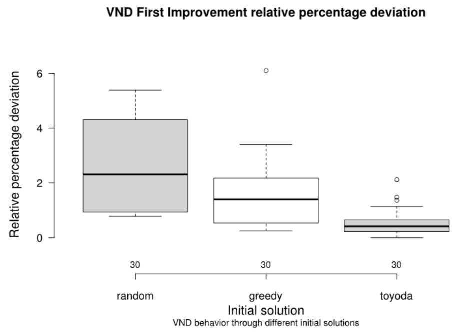


Figure 9. Percentage deviation

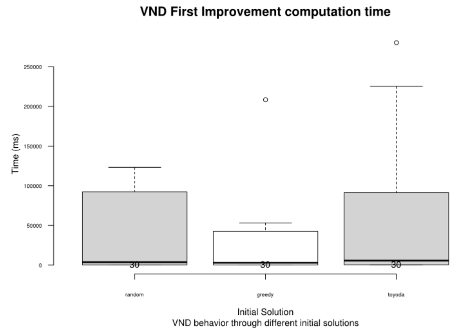


Figure 10. Computation time

The Table above shows that all the initial solutions have been improved applying VND improvement across all the instances.

A Wilcoxon Signed-ranks test indicates that when applying VND FI improvement Toyoda gets a lower deviation compared to Greedy and Random. At the same time Random and Greedy reports a significant difference between them being Greedy who reports lower values than Random , while Toyoda is clearly lower.

The test also indicates that applying VND FI improvement among Toyoda, Random and Greedy, there isn't a significant difference in the computation time.

3.3. Improvements 10x250 instances

Table 11. Average results 10x250 instance group

	Frist Improvement		Best Improvement		VND First Improvement	
Initial Solution	**t (ms) avg**	**∆avg**	**t (ms) avg**	**∆avg**	**t (ms) avg**	**∆avg**
Random	30.9870	4.3879	147.3161	4.5324	401924.2841	3.4886
Greedy	12.6425	4.2438	33.1510	4.0001	1290885.9511	1.8018
Toyoda	20.3591	0.9311	50.1991	0.7284	2319715.9324	0.3960

In this table it is possible to compare among the three different improvements. It shows that the FI reports the lower average computation time values. BI reports also lower values than VND, but the computation time of VND compared with FI and BI is way higher.
For the deviation Toyoda is showing lower average values while there isn't a big difference between the average values between FI and BI.
An important observation is that the highest deviation that Toyoda obtained (FI), is lower than the lower Random and Greedy deviations values obtained with VND.

Table 12. Improvements 10x250 instance group

	First Improvement						**Best Improvement**						**VND First Improvement**					
Initial Solution	Random		Greedy		Toyoda		Random		Greedy		Toyoda		Random		Greedy		Toyoda	
Instance	**t (ms)**	**∆avg**	**t (ms)**	**∆**	**t (ms)**	**∆**	**t (ms)**	**∆avg**	**t (ms)**	**∆**	**t (ms)**	**∆**	**t (ms)**	**∆avg**	**t (ms)**	**∆**	**t (ms)**	**∆**
OR10x250-0.25_1	33.65	6.12	11.73	9.07	20.35	2.44	131.2	6.24	19.77	9.28	25.64	2.05	44739.09	3.59	56664.47	2.35	126550.09	**0.44**
OR10x250-0.25_2	29.29	7.42	14.62	8.30	**13.22**	0.79	114.6	8.91	24.08	7.02	34.45	0.45	89815.15	3.50	54165.46	2.06	50107.17	**0.33**
OR10x250-0.25_3	32.98	7.85	**8.53**	8.47	21.40	1.04	122.6	7.78	21.97	8.86	76.75	1.16	90385.28	3.47	54494.81	2.76	45238.75	**0.55**
OR10x250-0.25_4	34.49	6.26	**7.20**	8.28	21.24	0.96	140.2	5.73	28.94	7.78	51.28	0.31	60615.99	3.22	72660.70	1.85	98323.97	**0.30**
OR10x250-0.25_5	32.54	6.18	**6.48**	8.84	16.92	2.04	128.5	6.79	32.47	5.64	57.90	2.01	78600.01	3.44	12579.22	3.45	54399.87	**0.78**
OR10x250-0.25_6	32.30	7.37	**10.97**	7.77	28.17	2.65	124.7	6.92	12.17	8.69	68.91	1.67	50259.22	3.56	15198.56	4.12	48250.24	**0.63**
OR10x250-0.25_7	30.72	7.47	15.19	5.59	16.66	1.40	109.3	7.27	**7.90**	8.90	42.32	1.21	65664.45	3.44	12043.01	4.04	67475.92	**0.28**
OR10x250-0.25_8	33.78	5.26	**11.51**	7.80	32.27	1.38	136.5	5.59	36.41	7.27	60.44	1.67	62689.71	3.06	16271.79	3.20	123103.99	**0.83**
OR10x250-0.25_9	34.73	7.56	**5.51**	7.80	15.76	0.84	124.3	7.65	20.52	6.78	33.65	0.98	83640.02	3.80	18091.31	2.25	55227.74	**0.30**
OR10x250-0.25_10	30.92	7.11	15.01	9.13	**12.43**	0.70	116.8	8.10	24.22	7.72	34.50	0.63	86455.74	3.80	38746.38	2.87	64083.38	**0.32**
OR10x250-0.50_1	36.73	3.74	**9.38**	3.08	19.37	1.59	183.6	4.72	44.97	2.74	110.30	0.58	2671827	1.51	7324704.13	0.46	3097737.87	**0.39**
OR10x250-0.50_2	39.61	4.23	16.97	2.53	**12.58**	1.59	198.1	4.56	28.00	2.68	60.79	0.64	1438400	1.65	1848399.39	0.90	3217549.50	**0.36**
OR10x250-0.50_3	40.35	4.11	13.52	3.43	**13.00**	0.49	202.9	4.26	44.89	2.57	30.81	0.39			1489156.39	1.03	4314538.73	**0.28**
OR10x250-0.50_4	39.34	3.27	26.77	3.16	**21.05**	0.70	188.7	4.33	44.44	4.00	70.10	0.39			1672588.06	1.04	5614099.36	**0.29**
OR10x250-0.50_5	37.31	4.36	**20.33**	3.44	39.46	0.50	192.5	4.21	89.27	2.72	48.18	0.70			1100209.79	2.31	1990461.00	**0.24**
OR10x250-0.50_6	38.49	4.07	**6.85**	4.51	38.58	1.06	212.7	3.63	44.72	3.26	111.02	0.53			1724390.67	0.66	6460403.26	**0.28**
OR10x250-0.50_7	36.35	4.29	**7.99**	4.39	35.29	0.81	194.5	4.77	44.53	3.09	50.76	0.69			1918550.83	0.74	2155586.41	**0.33**
OR10x250-0.50_8	37.67	4.39	**7.53**	4.29	33.70	0.75	193.8	4.62	44.44	2.34	51.22	1.03			1451121.14	1.41	2580113.89	**0.32**
OR10x250-0.50_9	37.86	4.64	**16.83**	2.30	43.16	0.81	186.4	4.83	55.58	2.02	30.12	1.30			4282731.50	0.50	5778856.67	**0.40**
OR10x250-0.50_10	39.92	3.82	**8.97**	2.91	25.82	0.81	215.5	3.94	28.05	2.84	65.38	0.51			2654951.44	0.84	10452210.84	**0.28**
OR10x250-0.75_1	21.38	2.51	25.59	0.80	**10.27**	0.43	117.4	2.00	39.03	0.89	21.18	0.42			60276050.25	**0.41**		
OR10x250-0.75_2	22.77	2.25	10.51	1.26	**10.02**	0.55	125.4	2.40	39.25	1.24	41.84	0.46			42066749.67	**0.37**		
OR10x250-0.75_3	23.02	1.99	22.32	0.70	**11.40**	0.48	130.3	1.52	30.05	1.40	67.84	**0.14**						
OR10x250-0.75_4	21.94	2.14	15.68	1.81	**10.90**	0.42	116.7	2.17	31.45	2.04	41.55	**0.22**						
OR10x250-0.75_5	23.55	2.04	**10.05**	1.35	11.02	0.51	130.3	1.84	30.45	1.49	40.15	**0.30**						
OR10x250-0.75_6	21.14	2.19	**8.98**	1.20	16.05	0.24	116.3	2.24	17.56	1.44	33.66	**0.16**						
OR10x250-0.75_7	22.66	2.21	**10.25**	1.23	14.97	0.53	125	1.89	34.37	1.13	34.37	**0.26**						
OR10x250-0.75_8	21.89	2.20	**12.17**	1.04	13.94	0.62	114.3	2.15	22.16	1.55	29.98	**0.48**						
OR10x250-0.75_9	20.95	2.72	16.71	0.89	14.21	0.41	105.6	2.92	44.00	0.67	41.16	**0.20**						
OR10x250-0.75_10	21.28	1.86	**5.16**	1.97	17.56	0.38	120.7	1.97	8.83	1.97	39.75	**0.33**						

It is possible to compare the performance among the three improvements and their initial solutions across all the instances. The lowest deviation values and lowest computation values are highlighted showing that the lowest time values are mostly in the FI. However the lower deviation values are mostly found in VND. The table is not complete due the huge amount of time the VND takes. The further instance reached in each initial solutions are: Random (0.50_2,23.97 minutes,average of 15), Greedy (0.75_2, 11.68 hrs) and Toyoda (0.50_10,2.90 hrs). As is easy to see with VND when the items in the solution increase the amount of time grows exponentially.

Table 13. Fraction of 10x250 instances that FI improves from initial solution

				First Improvement		
	Random	Greedy	Toyoda	Random	Greedy	Toyoda
Instance	∆avg	∆	∆	∆ avg	∆	∆
OR10x250-0.25_1	24.72	14.25	3.56	**6.12**	**9.07**	**2.44**
OR10x250-0.25_2	25.77	10.96	2.76	**7.42**	**8.30**	**0.79**
OR10x250-0.25_3	26.37	17.06	4.24	**7.85**	**8.47**	**1.04**
OR10x250-0.25_4	25.54	11.77	2.24	**6.26**	**8.28**	**0.96**
OR10x250-0.25_5	25.58	13.41	6.32	**6.18**	**8.84**	**2.04**
OR10x250-0.25_6	24.96	11.52	5.76	**7.37**	**7.77**	**2.65**
OR10x250-0.25_7	24.48	12.57	3.69	**7.47**	**5.59**	**1.40**
OR10x250-0.25_8	26.48	17.26	4.21	**5.26**	**7.80**	**1.38**
OR10x250-0.25_9	27.39	11.82	3.96	**7.56**	**7.80**	**0.84**
OR10x250-0.25_10	25.44	14.78	2.39	**7.11**	**9.13**	**0.70**
OR10x250-0.50_1	16.38	6.84	3.06	**3.74**	**3.08**	**1.59**
OR10x250-0.50_2	17.13	5.54	3.06	**4.23**	**2.53**	**1.59**
OR10x250-0.50_3	17.56	7.09	1.83	**4.11**	**3.43**	**0.49**
OR10x250-0.50_4	16.87	9.79	4.88	**3.27**	**3.16**	**0.70**
OR10x250-0.50_5	16.32	6.22	2.30	**4.36**	**3.44**	**0.50**
OR10x250-0.50_6	17.68	8.80	3.51	**4.07**	**4.51**	**1.06**
OR10x250-0.50_7	18.32	6.98	2.16	**4.29**	**4.39**	**0.81**
OR10x250-0.50_8	17.53	6.76	2.68	**4.39**	**4.29**	**0.75**
OR10x250-0.50_9	17.61	7.09	3.38	**4.64**	**2.30**	**0.81**
OR10x250-0.50_10	17.66	5.99	2.45	**3.82**	**2.91**	**0.81**
OR10x250-0.75_1	9.43	3.48	0.99	**2.51**	**0.80**	**0.43**
OR10x250-0.75_2	10.01	3.68	1.58	**2.25**	**1.26**	**0.55**
OR10x250-0.75_3	9.70	3.53	1.48	**1.99**	**0.70**	**0.48**
OR10x250-0.75_4	9.90	4.12	1.09	**2.14**	**1.81**	**0.42**
OR10x250-0.75_5	9.80	3.41	1.44	**2.04**	**1.35**	**0.51**
OR10x250-0.75_6	9.71	2.96	0.70	**2.19**	**1.20**	**0.24**
OR10x250-0.75_7	9.44	3.08	1.12	**2.21**	**1.23**	**0.53**
OR10x250-0.75_8	9.63	3.15	1.44	**2.20**	**1.04**	**0.62**
OR10x250-0.75_9	9.89	2.82	0.84	**2.72**	**0.89**	**0.41**
OR10x250-0.75_10	9.86	2.28	1.11	**1.86**	**1.97**	**0.38**

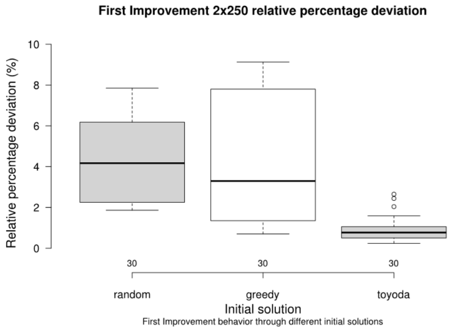


Figure 11. Percentage deviation

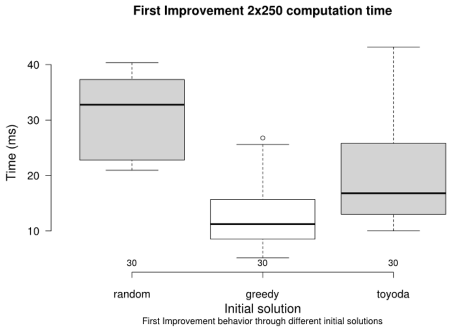


Figure 12. Computation time

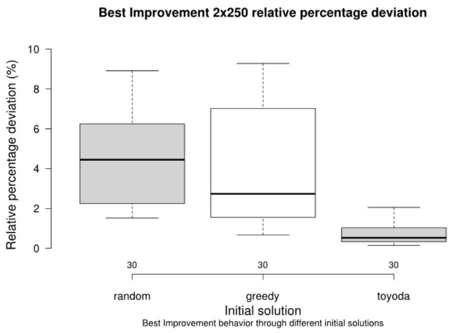
The Table above shows that all the initial solutions have been improved applying the FI improvements across all the instances.

A Wilcoxon Signed-ranks test indicates that when applying FI improvement Toyoda gets a lower deviation compared to Greedy and Random. At the same time Random and Greedy don't report a significant difference between them As the Box plot shows where the distributions of Random and Greedy overlaps , while Toyoda is clearly lower.

The test also indicates that applying First improvement among Toyoda, Random and Greedy, Greedy obtains the lower computation time values followed by Toyoda and Random with the highest values. computation time.

Table 14. Fraction of 10x250 instances that BI improves from initial solution

				Best Improvement		
	Random	Greedy	Toyoda	Random	Greedy	Toyoda
Instance	**∆avg**	**∆**	**∆**	**∆ avg**	**∆**	**∆**
OR10x250-0.25_1	24.72	14.25	3.56	**6.24**	**9.28**	**2.05**
OR10x250-0.25_2	25.77	10.96	2.76	**8.91**	**7.02**	**0.45**
OR10x250-0.25_3	26.37	17.06	4.24	**7.78**	**8.86**	**1.16**
OR10x250-0.25_4	25.54	11.77	2.24	**5.73**	**7.78**	**0.31**
OR10x250-0.25_5	25.58	13.41	6.32	**6.79**	**5.64**	**2.01**
OR10x250-0.25_6	24.96	11.52	5.76	**6.92**	**8.69**	**1.67**
OR10x250-0.25_7	24.48	12.57	3.69	**7.27**	**8.90**	**1.21**
OR10x250-0.25_8	26.48	17.26	4.21	**5.59**	**7.27**	**1.67**
OR10x250-0.25_9	27.39	11.82	3.96	**7.65**	**6.78**	**0.98**
OR10x250-0.25_10	25.44	14.78	2.39	**8.10**	**7.72**	**0.63**
OR10x250-0.50_1	16.38	6.84	3.06	**4.72**	**2.74**	**0.58**
OR10x250-0.50_2	17.13	5.54	3.06	**4.56**	**2.68**	**0.64**
OR10x250-0.50_3	17.56	7.09	1.83	**4.26**	**2.57**	**0.39**
OR10x250-0.50_4	16.87	9.79	4.88	**4.33**	**4.00**	**0.39**
OR10x250-0.50_5	16.32	6.22	2.30	**4.21**	**2.72**	**0.70**
OR10x250-0.50_6	17.68	8.80	3.51	**3.63**	**3.26**	**0.53**
OR10x250-0.50_7	18.32	6.98	2.16	**4.77**	**3.09**	**0.69**
OR10x250-0.50_8	17.53	6.76	2.68	**4.62**	**2.34**	**1.03**
OR10x250-0.50_9	17.61	7.09	3.38	**4.83**	**2.02**	**1.30**
OR10x250-0.50_10	17.66	5.99	2.45	**3.94**	**2.84**	**0.51**
OR10x250-0.75_1	9.43	3.48	0.99	**2.00**	**0.89**	**0.42**
OR10x250-0.75_2	10.01	3.68	1.58	**2.40**	**1.24**	**0.46**
OR10x250-0.75_3	9.70	3.53	1.48	**1.52**	**1.40**	**0.14**
OR10x250-0.75_4	9.90	4.12	1.09	**2.17**	**2.04**	**0.22**
OR10x250-0.75_5	9.80	3.41	1.44	**1.84**	**1.49**	**0.30**
OR10x250-0.75_6	9.71	2.96	0.70	**2.24**	**1.44**	**0.16**
OR10x250-0.75_7	9.44	3.08	1.12	**1.89**	**1.13**	**0.26**
OR10x250-0.75_8	9.63	3.15	1.44	**2.15**	**1.55**	**0.48**
OR10x250-0.75_9	9.89	2.82	0.84	**2.92**	**0.67**	**0.20**
OR10x250-0.75_10	9.86	2.28	1.11	**1.97**	**1.97**	**0.33**



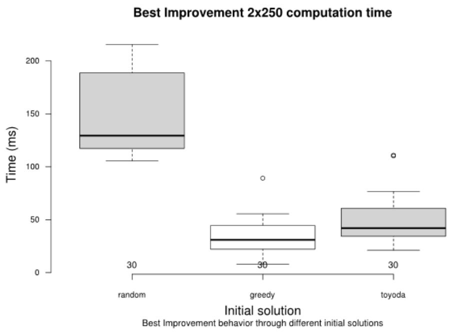


Figure 13. Percentage deviation

Figure 14. Computation time

The Table above shows that all the initial solutions have been improved applying the BI improvements across all the instances.

A Wilcoxon Signed-ranks test indicates that when applying BI improvement Toyoda gets a lower deviation compared to Greedy and Random. At the same time Random and Greedy don't report a significant difference between them As the Box plot shows where the distributions of Random and Greedy overlaps , while Toyoda is clearly lower.

The test also indicates that applying Best improvement among Toyoda, Random and Greedy, Greedy obtains the lower computation time values followed by Toyoda and Random with the highest values.

Table 15. Fraction of 10x250 instances that VND FI improves from FI

	First Improvement			VND First Improvement		
	Random	Greedy	Toyoda	Random	Greedy	Toyoda
Instance	Δavg	Δ	Δ	Δ avg	Δ	Δ
OR10x250-0.25_1	6.12	9.07	2.44	**3.59**	**2.35**	**0.44**
OR10x250-0.25_2	7.42	8.30	0.79	**3.50**	**2.06**	**0.33**
OR10x250-0.25_3	7.85	8.47	1.04	**3.47**	**2.76**	**0.55**
OR10x250-0.25_4	6.26	8.28	0.96	**3.22**	**1.85**	**0.30**
OR10x250-0.25_5	6.18	8.84	2.04	**3.44**	**3.45**	**0.78**
OR10x250-0.25_6	7.37	7.77	2.65	**3.56**	**4.12**	**0.63**
OR10x250-0.25_7	7.47	5.59	1.40	**3.44**	**4.04**	**0.28**
OR10x250-0.25_8	5.26	7.80	1.38	**3.06**	**3.20**	**0.83**
OR10x250-0.25_9	7.56	7.80	0.84	**3.80**	**2.25**	**0.30**
OR10x250-0.25_10	7.11	9.13	0.70	**3.80**	**2.87**	**0.32**
OR10x250-0.50_1	3.74	3.08	1.59	**1.51**	**0.46**	**0.39**
OR10x250-0.50_2	4.23	2.53	1.59	**1.65**	**0.90**	**0.36**
OR10x250-0.50_3	4.11	3.43	0.49		**1.03**	**0.28**
OR10x250-0.50_4	3.27	3.16	0.70		**1.04**	**0.29**
OR10x250-0.50_5	4.36	3.44	0.50		**2.31**	**0.24**
OR10x250-0.50_6	4.07	4.51	1.06		**0.66**	**0.28**
OR10x250-0.50_7	4.29	4.39	0.81		**0.74**	**0.33**
OR10x250-0.50_8	4.39	4.29	0.75		**1.41**	**0.32**
OR10x250-0.50_9	4.64	2.30	0.81		**0.50**	**0.40**
OR10x250-0.50_10	3.82	2.91	0.81		**0.84**	**0.28**
OR10x250-0.75_1	2.51	0.80	0.43		**0.41**	
OR10x250-0.75_2	2.25	1.26	0.55		**0.37**	
OR10x250-0.75_3	1.99	0.70	0.48			
OR10x250-0.75_4	2.14	1.81	0.42			
OR10x250-0.75_5	2.04	1.35	0.51			
OR10x250-0.75_6	2.19	1.20	0.24			
OR10x250-0.75_7	2.21	1.23	0.53			
OR10x250-0.75_8	2.20	1.04	0.62			
OR10x250-0.75_9	2.72	0.89	0.41			
OR10x250-0.75_10	1.86	1.97	0.38			

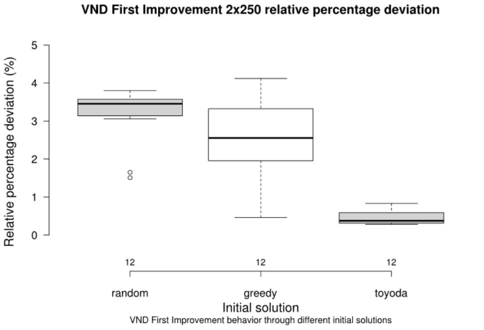


Figure 15. Percentage deviation

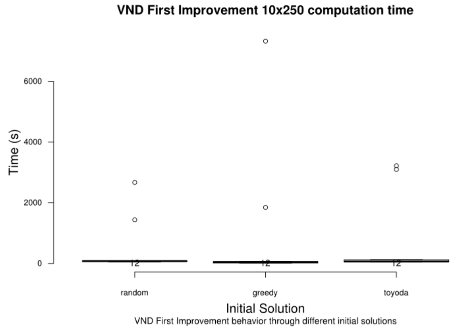


Figure 16. Computation time

The Table above shows that all the initial solutions have been improved applying the VND improvements across all the instances.

A Wilcoxon Signed-ranks test indicates that when applying VND improvement Toyoda gets a lower deviation compared to Greedy and Random. At the same time Random and Greedy don't report a significant difference between them As the Box plot shows where the distributions of Random and Greedy overlaps , while Toyoda is clearly lower.

The test also indicates that applying VND FI improvement among Toyoda, Random and Greedy, there isn't a significant difference in the computation time.

4. Conclusion

The cost of get a better quality solution always means more computation time, as it was observed across the analysis when from the different initial solutions an improvement is applied it always improved, however the computation time also increases. But comparing among the the different improvements it is easy to see that Toyoda improves faster, so if either FI or BI is applied to a Toyoda initial solution this yields better quality solutions than greedy or Random initial solutions in a VND which amount of computation time is considerably bigger than FI and BI.

The results also indicates that VND always improves the solution quality but the computation time increases faster since the algorithm demands to try all possible combinations at the time of removing items from the initial solution, making insignificant the comparison of the computation time between the algorithm starting with the different initial solutions.