HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

GROUP 18

REPORT

TOPIC 17: BIN PACKING LOWER, UPPER BOUND

IT3052E

Lecturer : Bui Quoc Trung

Student group

#Name	Student code
1.Nguyen Huu Nam	20210630
2.Tran Thi Hien	20214896
3.Nguyen Van Quoc	20214926

Ha Noi, year 2023

I. Problem description

There are N customers 1 ,2, …, N such that customers i requests to purchase an amount of goods $d(i)$ and the total cost of goods is $c(i)$. There are K delivery vehicles 1, 2, …, K, in which vehicle k has the minimum and maximum load of $c_1(k)$ and $c_2(k)$, respectively. Calculate the delivery method for the customer such that:

• Each customer can only receive delivery by at most 1 vehicle.

• The total amount of cargo on each vehicle must be within the minimum and maximum vehicle capacity.

• The total cost of goods received by the served customers is maximized.

II. Problem modeling

Denotation:

- X is a matrix of size $N \times K, X(i,j) = 1$ if the i^{th} item is placed on the j^{th} car, 0 otherwise (domain: $\{0;\ 1\}$).
- M: the total value of items on the cars.
- $W(k)$: the total weight of items on the k^{th} car.

Constraints:

- (1): $\sum_1^K X(n,j) \leq 1 \quad \forall n \in [1,N]$: each item can only be assigned to at most 1 car.
- (2): $c_1(k) \leq W(k) = \sum_1^N X(i,\ k) \times d(i) \leq c_2(k)\ \forall k \in [1,K]$: the total weight of items on each car is between the lower and upper bound of capacity.
- $M = \sum_1^K \sum_1^N X(i,j) \times c(i)$: total value of items on cars.

Objective:

Maximize M

III. Data generation

The generating algorithm for each file:

- Randomize the number of items (N) based on the number of cars (approximately K^2)
- Randomize the weight and profit of each value (ranging from 1 to 10).
- Randomly assign the lower and upper bound for capacities of each car.

Note that the probability distribution of all random variables in this algorithm is uniform.

IV. Detail of solutions

We applied the following methods in our solution:

- Backtracking
- Local search (hill climbing)
- Iterated local search
- MIP model
- CP model

V.1. Data structures used to represent input and output data:

- The input data:

N, K: integers representing the number of items and cars

items: list containing lists of weight and value of each item

cars: list containing lists of lower bound and upper bound for capacity of each car

- The output data:

best_value: integer representing the maximum profit

best_sol: a list of N intergers ranging from 0 to K indicating which item is assigned to which car (Ex: [1,0,1,2] means that item 1 and 3 is assigned to car 1, item 4 is assigned to car 2 while item 2 isn't assigned to any car)

V.2. Backtracking method

Backtracking is a brute force algorithm. It is also a depth first search because it will explore a branch until it finds a solution or hits a dead end (a state that the total weight in a car exceeds the upper bound of its capacity) before backtracking.

Below are the steps to solve the problem using Backtracking:

1. Set the initial state as an empty solution.
2. Initialize the best solution value to zero.
3. Generate all possible candidate solutions by recursively exploring all the remaining items that have not been assigned to any bin.
4. Prune the search tree by checking if the current weight of each car is less than the upper bound of its capacity. If it is not, backtrack and discard the current candidate solution.
5. For each candidate solution, check if it satisfies the lower and upper bound constraints of each bin. If it does, compute the objective function value and compare it to the best solution value. If it is better, update the best solution value and best solution.
6. Repeat steps 3-5 until all candidate solutions have been explored.
7. Return the best solution value and the best solution.

Theoretically, Backtracking is guaranteed to solve every problem without time limitation. However, due to the complexity scale of the problem, it will take huge amount of time to solve nonsimple cases using Backtracking.

V.3. Local search method (steepest hill climbing)

Local search is a heuristic optimization technique used for solving problems by iteratively improving a solution within a neighborhood of the current solution. The idea is to start with an initial solution and repeatedly make small changes to it in the hope of finding a better solution. Here, we decided to use the Steepest hill climbing algorithm, which is to evaluate all neighboring solutions and choose the one that leads to the highest improvement in the objective function. This approach ensures that the algorithm moves in the steepest possible direction towards the optimal solution at each step, hence the name "steepest" hill climbing.

Below are the steps to solve the problem using Local search:

1. Generate an initial state randomly in the form of a list (the same data structure as the list used to show the solution in the ouput)

2. Calculate the objective value of the initial solution. If it does not meet the constraints for the lower and upper bound of capacities, return 0.

3. Generate all the possible neighbors of the current solution by increasing each element in the list by 1 (if the element is equal to K, change it to 0)

4. Calculate the objective value of each neighbor.

5. Select the neighbor with the highest objective value as the next solution.

6. If the objective value of the next solution is better than the current solution, set the current solution to the next solution. Otherwise, terminate the algorithm and return the current solution as the best solution found.

7. Repeat steps 3-6 until a stopping criterion is met. (no better solution can be found in the neighborhood)

Note that since there is a constraint for the lower bound, there is no guarantee that the randomly generated state and its neighborhood are feasible solutions. Hence, we improved upon Local search by incorporating the next method.

V.4. Iterated local search method

Iterated Local Search (ILS) is a metaheuristic optimization algorithm that combines elements of local search and metaheuristics to solve optimization problems. The main idea behind ILS is to use the strengths of local search algorithms, which are good at finding local optima, and metaheuristics, which are good at escaping local optima and finding good solutions in a large search space. By combining the two, ILS is able to find better solutions than either method alone.

Below are the steps to solve the problem using Iterated local search:

1. Generate an initial state randomly.

2. Perform local search (steepest hill climbing) from the current solution to find a better one.

3. Update the best solution and the best solution value. Generate a new initial state for the next iterations.

4. Repeat steps 2 and 3 until a stopping criterion is met. (reaching a maximum amount of iterations).

Although Iterated local search is still a stochastic algorithm and may not find an optimal solution (or even a feasible one) in some cases, it is guaranteed to perform much better in comparision with the previous Local search method.

V.5. MIP model and CP model

We used OR-Tools, a library developed by Google, to implement these two programs.

MIP model is used for optimization problems where some or all of the variables are restricted to take integer values. It is a linear programming model with additional constraints on the variables to take only integer values. OR-Tools provides different MIP solvers, and the one that we used is SCIP.

CP model, on the other hand, is used for combinatorial optimization problems, where the goal is to find a feasible solution that satisfies a set of constraints. It allows variables to take discrete values and supports many constraint types, including global constraints. OR-Tools provides a CP solver based on constraint propagation and local search.

With the problem modeling from previous, below are the general steps to solve the problem using MIP model and CP model:

1. Import the libraries
2. Create the data
3. Declare the solver object
4. Create the variables
5. Define the constraints
6. Define the objective function
7. Solve the problem

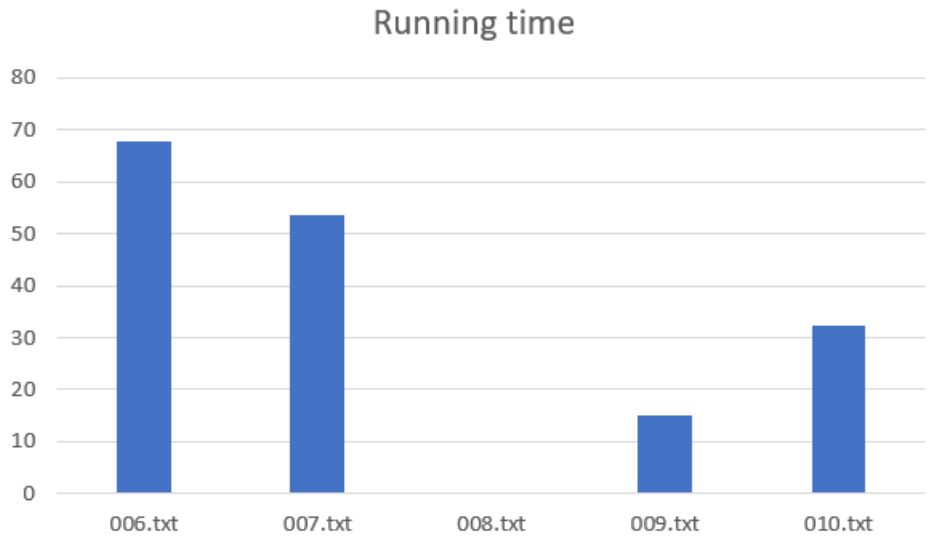
V. Analysis, findings, conclusions

V.1. Backtracking method

Below are running time for easy test cases:

K = 2:

K = 3:



For cases from $K = 4$ and above, the running time exceed the time limit of 1 hour.

V.2. Local search method

As we explained above, the Local search algorithm seems to have poor performance with larger test cases, hence we decided to run it 10 times for each of the first 20 test cases to test its performance.

The algorithm comes up with a legitimate solution for 27/200 cases, only 12/27 of those are optimal solutions. => Success rate are 6%, which is much worse than our expectation.

V.3. Iterated local search method

We run the Iterated local search algorithm with 1000 iterations for each test cases.

As a result, it comes up with a legitmate solution for 19/36 test cases, 14/19 of those are optimal solutions. Out of the 16 hard and very hard cases, it can only find a legitimate solution for 1 case (not an optimal solution).

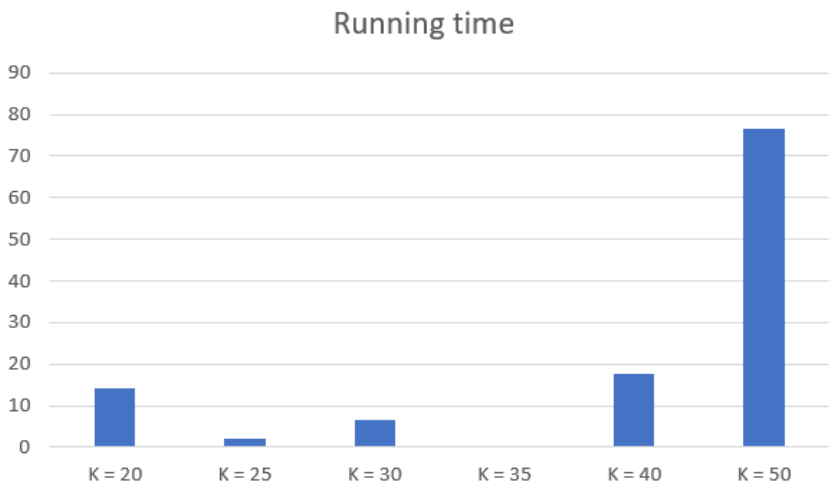
Because the algorithm can't find a feasible solution for most of the complex test cases, there is not enough information to conclude about its running time.

V.4. MIP model

We run the MIP model with a time limitation of 1 hour and it can solve 35/36 test cases.

The objective values of solved instances are excellent in general and the running time for each of the first 30 test cases is less than 1 second.

Below is the running time for very hard test cases:

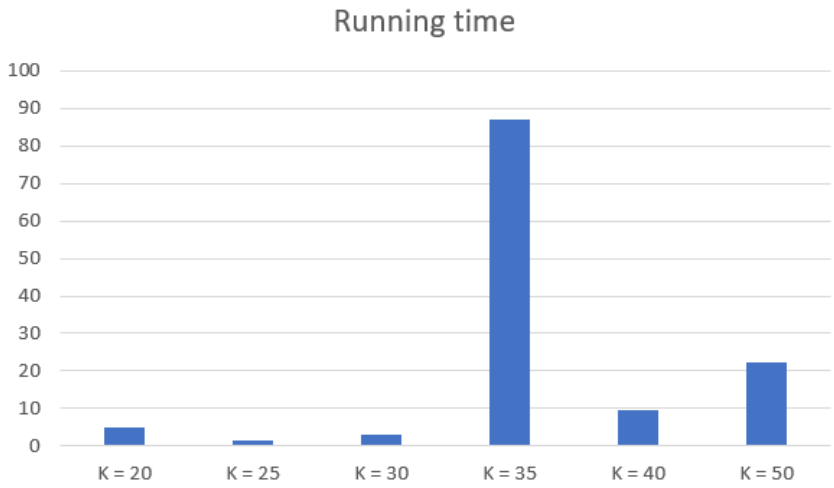


V.5. CP model

We run the CP model with a time limitation of 1 hour and it can solve 36/36 test cases.

The objective values of solved instances are excellent in general and the running time for each of the first 30 test cases is less than 1 second.

Below is the running time for very hard test cases:



V.6. Final results

- Running time: (in second)

Test_set	Backtracking	Iterated Local Search	MIP model	CP model
001.txt	0.0012	0.0238	0.0063	0.0050
002.txt	0.0090	0.0402	0.0088	0.0057
003.txt	0.0073	0.0429	0.0059	0.0056
004.txt	0.0096	0.0057	0.0085	0.0054
005.txt	0.0049	0.0001	0.0060	0.0055
006.txt	67.659	0.0016	0.0066	0.0072
007.txt	53.648	0.1070	0.0261	0.0112

008.txt	0.3497	0.0121	0.0101	0.0054
009.txt	15.179	0.0932	0.0068	0.0108
010.txt	32.317	0.0956	0.0180	0.0146
011.txt		0.0409	0.0080	0.0099
012.txt		0.1147	0.0078	0.0106
013.txt		0.0235	0.0074	0.0089
014.txt		0.0005	0.0089	0.0113
015.txt		0.0006	0.0074	0.0100
016.txt		NaN	0.0394	0.0301
017.txt		0.4889	0.0112	0.0100
018.txt		NaN	0.0108	0.0168
019.txt		0.2729	0.0098	0.0173
020.txt		0.3469	0.2440	0.0188
021.txt		NaN	0.0429	0.0151
022.txt		NaN	0.0391	0.0331
023.txt		NaN	0.1648	0.1198
024.txt		3.3455	0.0330	0.0887
025.txt		NaN	0.1006	0.1197
026.txt		NaN	0.0631	0.1205
027.txt		NaN	0.6059	0.6759
028.txt		NaN	0.9449	0.6126
029.txt		NaN	1.2699	0.8902
030.txt		NaN	1.8228	0.9159
031.txt		NaN	14.320	5.1177
032.txt		NaN	1.9396	1.6489

033.txt		NaN	6.4179	3.0960
034.txt		NaN	NaN	86.863
035.txt		NaN	17.684	9.6655
036.txt		NaN	76.702	22.287

- Objective values:

Test_set	Backtracking	Iterated local Search	MIP model	CP model
001.txt	17	17	17	17
002.txt	33	33	33	33
003.txt	35	35	35	35
004.txt	35	35	35	35
005.txt	37	37	37	37
006.txt	62	62	62	62
007.txt	74	74	74	74
008.txt	23	23	23	23
009.txt	44	42	44	44
010.txt	68	68	68	68
011.txt		100	100	100
012.txt		106	106	106
013.txt		70	70	70
014.txt		84	84	84
015.txt		76	76	76
016.txt		0	125	125
017.txt		156	160	160

018.txt		0	125	125
019.txt		118	123	123
020.txt		118	130	130
021.txt		0	184	184
022.txt		0	269	269
023.txt		0	361	361
024.txt		365	410	410
025.txt		0	513	513
026.txt		0	646	646
027.txt		0	820	820
028.txt		0	993	993
029.txt		0	1180	1180
030.txt		0	1218	1218
031.txt		0	2290	2290
032.txt		0	3496	3496
033.txt		0	5082	5082
034.txt		0	0	6853
035.txt		0	8625	8625
036.txt		0	13788	13789

V.7. Conclusions

- Backtracking: not impractical
- Local search: very low efficiency
- Iterated local search: moderate efficiency (for easy and medium cases), low efficiency (for hard cases)
- MIP model: high efficiency
- CP model: high efficiency (slightly better than MIP model)

VI. Proposal on future work

VI.1. Improve the current algorithms

Backtracking:

- Improve the pruning technique.
- Change the order of states that the algorithm visit.

Local search and iterated local search:

- Apply heuristics to generate better initial state.
- Improve upon constructing the state's neighborhood.
- Improve the legitimate checking function.

VI.2. Investigate further into generating and working with large test sets

As shown in the test results, the case with K = 35 seems to be much more complex than the case with K = 40 and K = 50. Hence, we decided to generate serveral test case with K = 50 (not documented), and surprisingly, there are 2 out of 10 cases which take less time than some of the simpler cases while the remaining 8 cases take a huge amount of time. The reason behind this is probably the random nature of the generated data.

VI.3. Incorporate other algorithms

We have come up with basic pseudo code for some other algorithms (greedy algorithm, genetic algorithms), however, we failed to implement them and decided to go with the 5 current approaches instead.

VII. Member contribution

Nguyễn Hữu Nam	Trần Thị Hiền	Nguyễn Văn Quốc
Leader Backtracking Local search Iterated local search MIP and CP model (50%) Generating data Report (50%)	MIP and CP model (50%) Collecting data Slides	Data analysing Report (50%)

VIII. Disclaimer

Since we misintepreted the main purpose of the presentation and also worried about the 15 – minute limitation, we decided to focused on the wrong part, which is the final result, and combined the algorithm and the data analysing part together. We sincerely apologize to you and we really hope that you could consider forgiving us for our mistake.

All the source code, slides and final report can be found here.