



PROJECT

Fundamentals of Optimization



TABLE OF CONTENTS



Problem description

introduction and overview of the problem

01

03

Proposed methods

about proposed solution

Modelling

how you write down everything that you want to say that you can't say in the source code

02

04

Experiments

experiments

05

Conclusion

final conclusion



Bin packing lower, upper bound

Tran Le My Linh 20210535

Nguyen Ba Thiem 20214931

Phan Dinh Truong 20214937

Nguyen Minh Tuan 20214940



01

Problem description



IDEA FOR THE PROBLEM

TRUCK DATA

Quantity K trucks

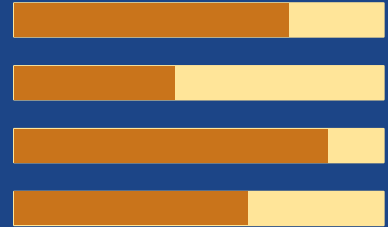
Load limit max, min

CUSTOMER DATA

Quantity N customers

Order quantities of goods

Value Total value



GOALS

- ✓ each customer takes goods from only 1 truck
- ✓ total amount of goods/truck in range [min,max load]
- ✓ total values of delivered goods must be maximized



02

MODELLING



└ FIRST ANNOUNCEMENT ─



$d[1],$
 $d[2]$
 \dots
 $d[N]$

goods_quantity
 $= [d[1], d[2], d[3], \dots, d[N]]$

the vector containing the quantity of
goods that customers ordered



$c[1],$
 $c[2]$
 \dots
 $c[N]$

values
 $= [c[1], c[2], c[3], \dots, c[N]]$

the vector containing goods' values

└ SECOND ANNOUNCEMENT ─



Load of each truck

weighted sum of goods' quantities
indicating either that truck contains
goods for that particular customer.

Variables

$weight[1:K, 1:N]$: $weight[i, j]$
packages of customer j
are contained on truck j

$weight[i, j] == 1$
truck i contains packages
of customer j

$weight[i, j] != 1$
truck i doesn't contains packages
of customer j

THIRD ANNOUNCEMENT

Constraints



For each truck i

load must be between lower bound
and upper bound

$$c1[i] \leq \text{weight}[i,:]^T \times \text{goods_quantity} \leq c2[i]$$



For each customer j

his/her packages can only be
delivered on one single truck

$$\text{sum}(\text{weight}[i, j]) \leq 1 \text{ for } i \text{ in } [1, K]$$



LAST ANNOUNCEMENT



Objective function to maximize

Totals values of
delivered packages

sum(weight x values)



03

PROPOSED METHODS



└ PROPOSED ALGORITHMS ─



Greedy



Local search

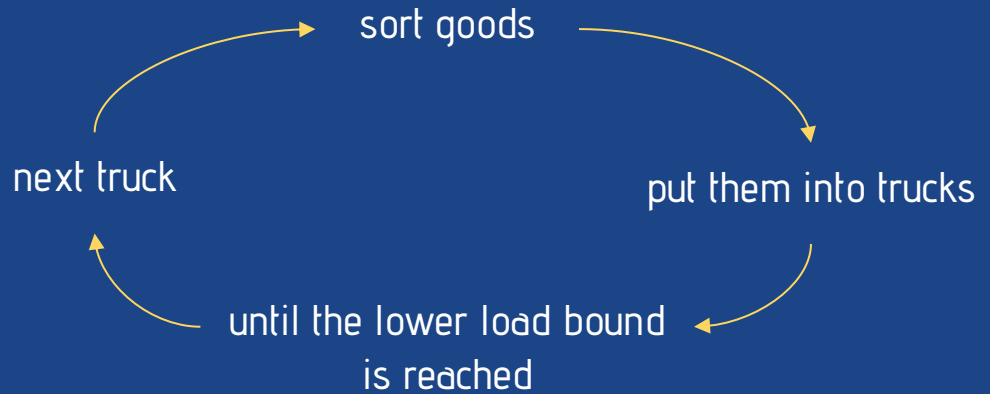


Ortools ILP



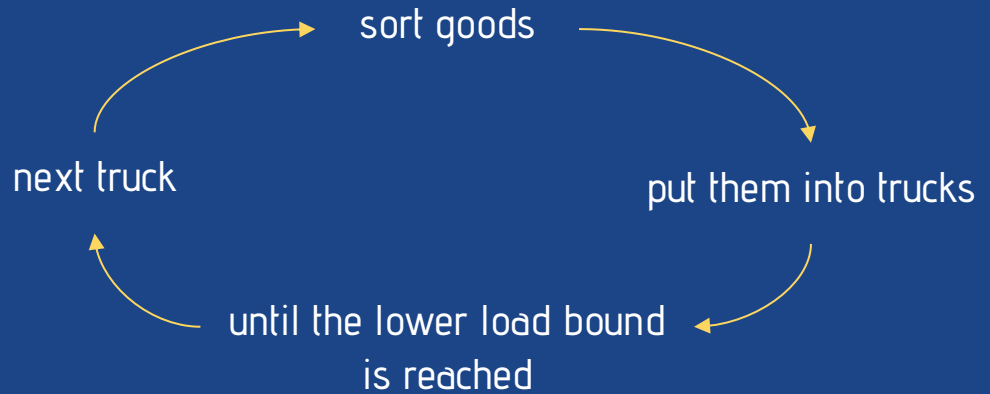
Ortools CP-SAT

└ PROPOSED ALGORITHMS ─



▲ Next, we pack (goods) into trucks until the trucks are full or there is no package left. ▼

└ PROPOSED ALGORITHMS ─

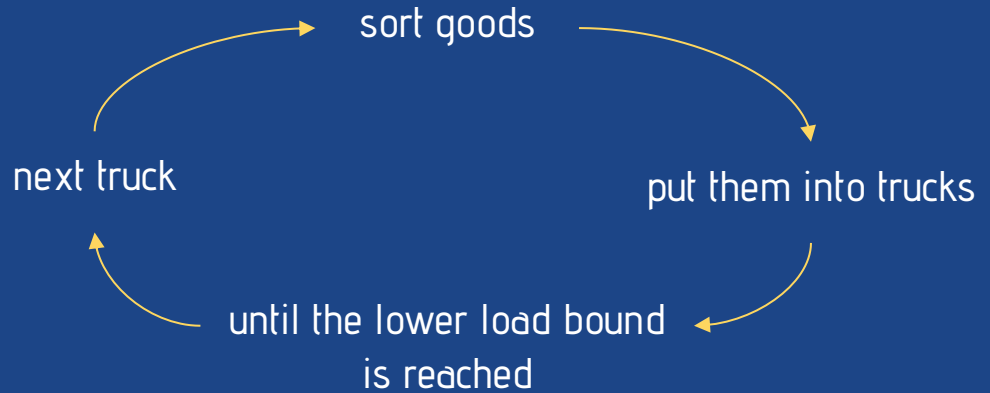


▲ PROBLEM

in which way
should we sort packages and trucks?



└ PROPOSED ALGORITHMS ─



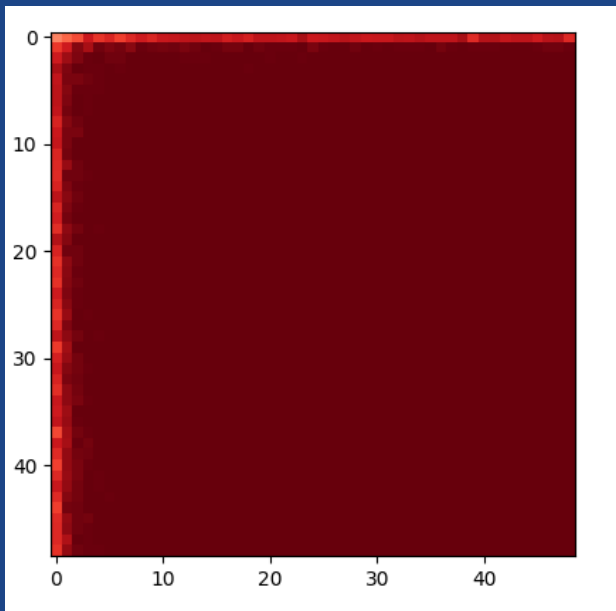
▲ PROBLEM ▼

with packages sorting,
3 ways to sort: their quantities, values
and by their efficiency

✚ SORTING ✚



efficiency = value / quantity



**Most of the time, we don't have a
feasible solution.
Otherwise, the algorithm fails.**



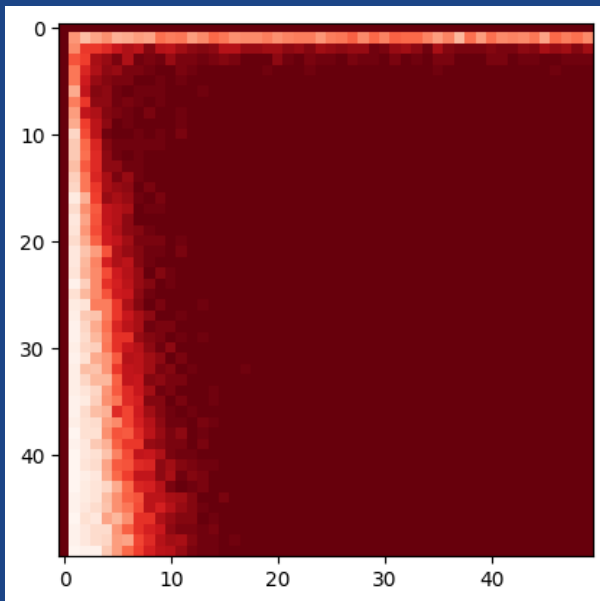
Sort by efficiency



SORTING



prioritizing the values of goods
instead of efficiency



Test A: 99.62% goods delivered
92.89% total values

Test B: 90.54% goods delivered
99.75% total values

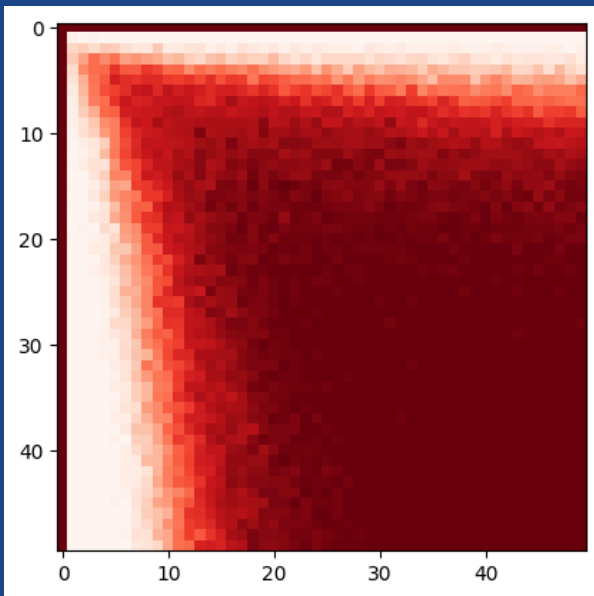


Sort by value

✚ SORTING ✚



we think that sorting by the quantity of goods
might be a better bet



our algorithm gives us
a feasible solution every time

Test A: 93.97% goods delivered
93.97% total values

Test B: 97.59% goods delivered
97.59% total values



Sort by quantity

✚ SORTING ✚



lower
bound

First, we scale both values and quantities to a range of `[min_importance, 1]`, here `min_importance (> 0)`

define importance of a package

$\text{importance} = \text{value}^{\text{order}} \times \text{quantity}$

order: scale down the importance of value comparing to quantity
we choose `order = 2`

Test A: 92.38% packages delivered
94.17% value rate

Test B: 97.98% packages delivered
98.66% value rate



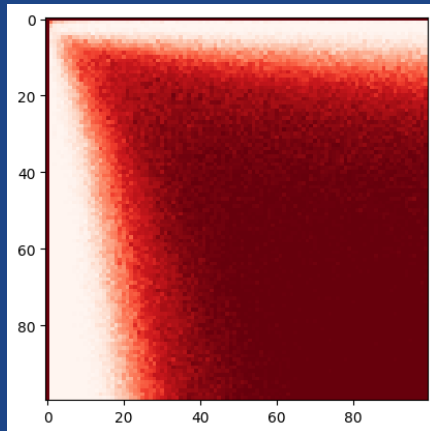
Sort by a combination of value and quantity

✚ SORTING ✚



we sort those approaches by their rates of values delivered, try to use the one with the highest rate and check if it fails or not. If it fails, the next best method will be used.

Method	Test A (N, K < 100)			Test B (N = 10^5 , K = 2×10^2)		
Result	Value rate (%)	Deliver rate (%)	Failing rate (%)	Value rate (%)	Deliver rate (%)	Failing rate (%)
Efficiency	88.38	86.57	93.62	0	0	100
Value	92.89	90.54	74.55	99.75	99.62	0
Importance	94.17	92.38	61.94	98.66	97.98	0
Quantity	93.97	93.97	29.69	97.59	97.59	0
Combine	96.78	96.05	29.26	99.75	99.62	0



efficiency, value, importance and quantity. ▼

Combining different ways of sorting



TIME COMPLEXITY



Python default sorting algorithm is Timsort (a combination of Merge sort and Insertion sort),

→ sorting time complexity: $O(N \log N)$

Adding goods to trucks requires 2 nested for-loops, each has a time complexity of $O(NK)$

→ Adding goods time complexity: $O(NK)$

→ Time complexity: $O(NK)$

In reality, greedy algorithm (if possible) instantly give us a solution in test A and takes less than a few seconds in test B.

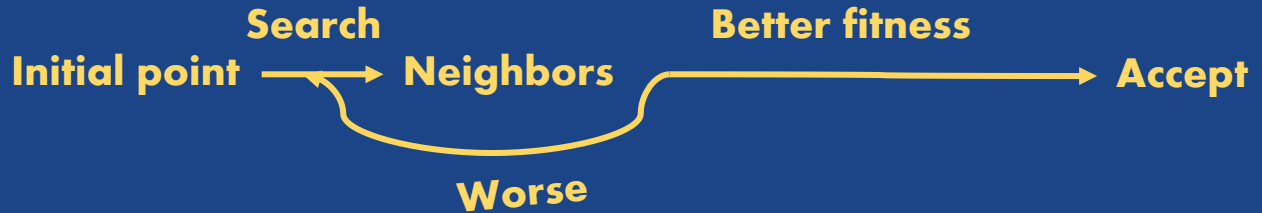


└ PROPOSED ALGORITHMS ─



Using greedy to find a solution having the high value of objective function which could not satisfy all constraints, then find its neighbor which satisfies all constraints to set initial solution for local search algorithm

└ Hill climbing ─



Starting with sufficient point then find around itself the better solution until reach the time limit.

Hill Climbing is stuck in local maximum, plateau, or shoulder.



└ Simulated annealing ─



└ Simulated annealing ─



As recommended, temperature, cooling rate, time limit are 1000, 0.7, 60s respectively in default . This method chooses neighbor randomly at each temperature, and accept it with a probability

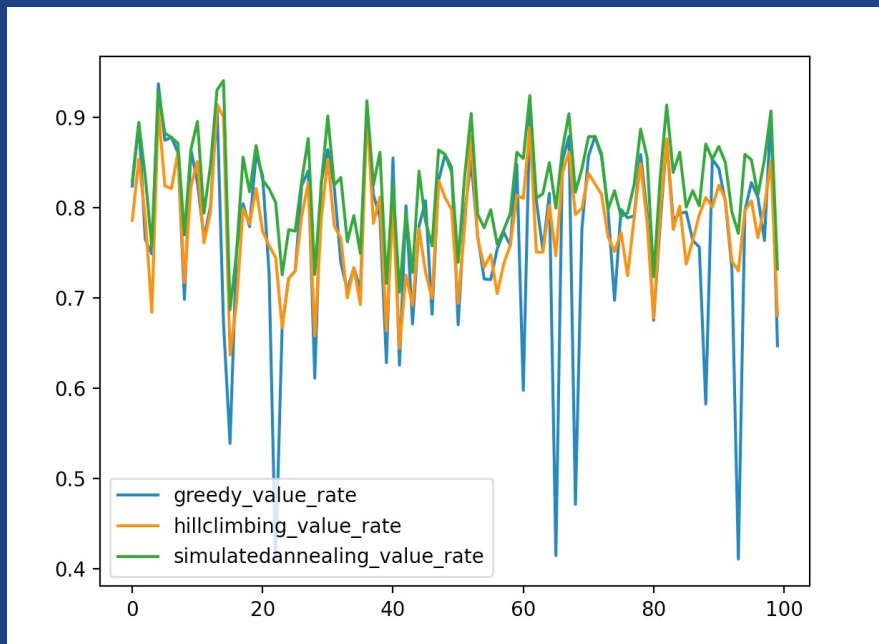
$$p = e^{\frac{\Delta E}{k.T}}$$

In this case, k is average of all delta E of temperature reduction. If this method reach time limit, it returns the best feasible solution.





QUICK TEST



N=50, K=2



└ PROPOSED ALGORITHMS ─



The Strengths and Weaknesses of this method

Strengths: This method gives us quite fast and accurate result when the number of trucks and customers is small. The only job is to define the variables and its constraints, the rest is for the library.

Weaknesses: For large datasets, that means OR-Tools takes a long time to run. We can find many other libraries or other method to support.

└ PROPOSED ALGORITHMS ─



CP-SAT is another solver for linear integer programming which implements local search and meta-heuristics on top of a Constraint Programming solver.

Its performance is much higher than SCIP which was used above.

Moreover, we add an option to feed the solver with a initial solution from greedy algorithm, which improves the time required to find some first feasible solutions.
with $N = 10000$ and $K = 50$ with a variable time limit on Google Colab

Time limit	Without initial solution		With initial solution	
	Value rate (%)	Deliver rate (%)	Value rate (%)	Deliver rate (%)
30	TIMED OUT	TIMED OUT	98.83	98.46
90	91.09	90.70	98.86	98.82
120	96.82	95.59	99.06	98.98



04

EXPERIMENTS

N = 10, K = 2					N = 10, K = 5				
	Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)		Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)
Greedy	74.28	71.91	6.0	0.0002	Greedy	85.29	84.46	35.0	0.004
Hill Climbing	81.01	77.90	0.00	60.000	Hill Climbing	93.35	91.90	0.00	60.000
Simulated Annealing	81.01	78.00	0.00	60.000	Simulated Annealing	93.23	91.70	0.00	60.000
CP SAT	81.01	78.00	0.00	0.006	CP SAT	93.49	92.00	0.00	0.021
ILP (SCIP)	81.01	77.90	0.00	0.009	ILP (SCIP)	93.49	92.00	0.00	0.008
CP SAT + Greedy	81.01	77.90	0.00	0.005	CP SAT + Greedy	93.49	92.00	0.00	0.028

N = 50, K = 2					N = 50, K = 5				
	Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)		Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)
Greedy	76.99	71.24	0.00	0.0004	Greedy	85..79	83.12	0.00	0.005
Hill Climbing	77.89	74.80	0.00	60.000	Hill Climbing	91.43	88.36	0.00	60.000
Simulated Annealing	82.60	78.62	0.00	60.000	Simulated Annealing	92.12	89.06	0.00	60.000
CP SAT	82.92	79.32	0.00	1.450	CP SAT	93.44	91.22	0.00	22.445
ILP (SCIP)	82.92	79.26	0.00	0.039	ILP (SCIP)	93.47	91.22	0.00	1.383
CP SAT + Greedy	82.92	79.34	0.00	1.360	CP SAT + Greedy	93.47	91.22	0.00	22.396

Performance of Greedy, Hill Climbing, Simulated Annealing, CP SAT, ILP (SCIP) and CP SAT + Greedy									
N = 100, K = 2					N = 100, K = 5				
	Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)		Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)
Greedy	78.41	71.86	0.00	0.007	Greedy	89.69	85.74	0.00	0.007
Hill Climbing	79.07	76.15	0.00	60.000	Hill Climbing	92.50	90.11	0.00	60.000
Simulated Annealing	81.64	77.87	0.00	60.000	Simulated Annealing	92.37	90.14	0.00	60.000
CP SAT	83.11	79.25	0.00	0.171	CP SAT	93.20	93.20	0.00	11.606
ILP (SCIP)	83.11	79.27	0.00	0.098	ILP (SCIP)	93.20	93.20	0.00	1.940
CP SAT + Greedy	83.11	79.27	0.00	0.374	CP SAT + Greedy	93.20	93.20	0.00	11.973

N = 100, K = 10					N = 100, K = 50				
	Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)		Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)
Greedy	92.14	90.68	1.00	0.009	Greedy	100.00	100.00	98.00	0.006
CP SAT	97.20	96.03	0.00	2.183	CP SAT	100.00	100.00	15.00	12.801
ILP (SCIP)	97.20	96.03	0.00	6.492	ILP (SCIP)	100.00	100.00	39.00	32.892
CP SAT + Greedy	97.20	96.03	0.00	2.379	CP SAT + Greedy	100.00	100.00	13.00	11.895

N = 100, K = 100					N = 100, K = 500				
	Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)		Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)
Greedy	0.00	0.00	100.00	0.003	Greedy	0.00	0.00	100.00	0.012
CP SAT	100.00	100.00	61.00	25.848	CP SAT	100.00	100.00	10.00	24.180
ILP (SCIP)	100.00	100.00	74.00	15.450	ILP (SCIP)	100.00	100.00	26.00	22.583
CP SAT + Greedy	100.00	100.00	61.00	26.021	CP SAT + Greedy	100.00	100.00	10.00	23.803

N = 1000, K = 10					N = 1000, K = 50				
	Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)		Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)
Greedy	94.39	91.59	0.00	0.018	Greedy	99.32	99.02	0.00	0.013
CP SAT	96.37	94.82	0.00	19.061	CP SAT	99.75	99.62	0.00	28.603
ILP (SCIP)	96.17	94.69	0.00	39.874	ILP (SCIP)	99.32	99.33	16.22	59.861
CP SAT + Greedy	96.36	94.82	0.00	27.999	CP SAT + Greedy	99.75	99.62	0.00	24.803

N = 1000, K = 100					N = 1000, K = 500				
	Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)		Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)
Greedy	99.88	99.82	95.00	0.029	Greedy	0.00	0.00	100	0.177
CP SAT	99.94	99.94	42.00	30.433	CP SAT	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT
ILP (SCIP)	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT	ILP (SCIP)	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT
CP SAT + Greedy	99.93	99.93	41.00	29.586	CP SAT + Greedy	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT

N = 10000, K = 10					N = 10000, K = 50				
	Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)		Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)
Greedy	94.94	90.88	0.00	0.068	Greedy	98.85	98.56	0.00	0.076
CP SAT	96.12	94.60	0.00	TIMED OUT	CP SAT	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT
ILP (SCIP)	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT	ILP (SCIP)	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT
CP SAT + Greedy	96.22	94.65	0.00	TIMED OUT	CP SAT + Greedy	98.85	98.56	0.00	TIMED OUT

N = 10000, K = 100					N = 10000, K = 500				
	Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)		Value rate (%)	Deliver rate (%)	Failing rate (%)	Exec time (s)
Greedy	99.59	99.39	0.00	0.108	Greedy	0.00	0.00	100	0.642
CP SAT	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT	CP SAT	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT
ILP (SCIP)	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT	ILP (SCIP)	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT
CP SAT + Greedy	99.59	99.39	0.00	TIMED OUT	CP SAT + Greedy	TIMED OUT	TIMED OUT	TIMED OUT	TIMED OUT



05

CONCLUSION



✚ Conclusion ✚



SHORTED TIME: greedy algorithms

-> its failure rate is high when N and K are close to each other.

Simulated Annealing > Hill Climbing

-> the performance gap gets narrower when K increases

-> these algorithms are implemented in Python. Their performance could be better if implemented in other programming languages.



✚ Conclusion ✚



when N and K are small enough: **ILP method**
using **OR-TOOLS (SCIP solver)**

-> in the shortest time

when N and K are higher ($N > 100, K > 10$): **CP**
using **OR-TOOLS (CP-SAT solver)**

-> in the shorter time

when N and K are enormous, it might take a
long time: **CP** with an initial solution if Greedy
can provide us with a feasible solution.



✚ Conclusion ✚



methods using OR-Tools give better performance given the same time limit compared to others when N and K are large, Greedy algorithm is the recommended method: gives us a fairly good solution in just a few seconds - even when it fails, we didn't waste too much time.

Local search methods even though might give optimal or good feasible solutions, take too much time because of their programming language.



THANKS!

