Master Thesis

An Evolutionary Algorithm for Solving the Two-Dimensional Irregular Shape Packing Problem Combined with the Knapsack Problem

Albert Espín Román

Advisor: René Alquézar (UPC - FIB)

Master in Artificial Intelligence

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

UNIVERSITAT DE BARCELONA

UNIVERSITAT ROVIRA i VIRGILI

Outline

1. Introduction
2. Previous Work
3. Proposed Methods
4. Datasets
5. Experimental Methodology
6. Results and Discussion
7. Conclusions and Future Work

Introduction

- Knapsack Problem: place items defined by their **value** and **weight** in a shapeless container with a maximum weight limit (**capacity**). Maximize the value in the container.
- Packing Problem: place items defined by their **geometric shape** in a container of a certain shape without **intersections**. Maximize the number of items placed in the container (alternatively: minimize the area/volume of all items).
- **Joint Problem**: place items defined by their **geometric shape**, **value** and **weight** in a container of a certain shape. Maximize the value in the container, respecting two constraints:
  - The container's **capacity** cannot be exceeded.
  - Geometric **intersections** cannot occur.

Applicability

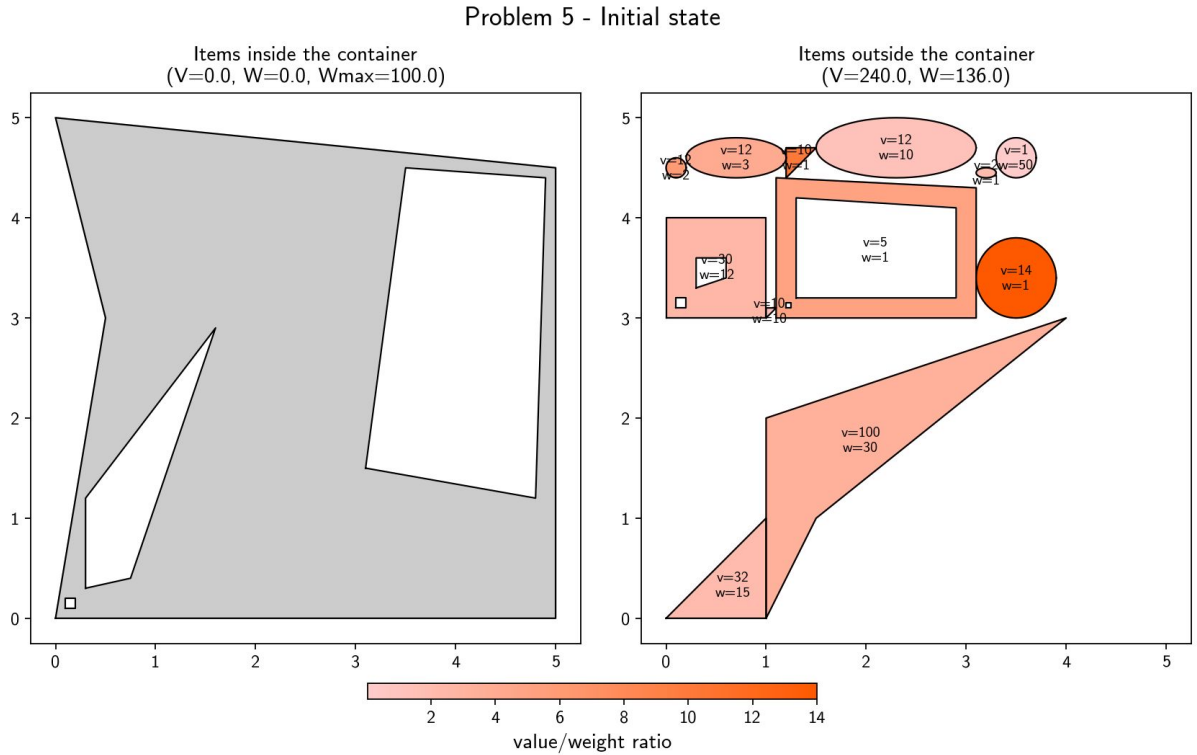
- The Packing Problem is widely applied in many industries.
- The features of the Knapsack Problem added by the Joint Problem can be useful:
  - Giving value to objects can help to **prioritize some placements**:
    - Goods that need to be delivered urgently (3D Packing in logistics).
    - Parts which are strategic or essential (2D Packing in high-precision tools).
  - Giving weight to items (and a limit to the container) can model **realistic scenarios**:
    - Transporting objects in a lorry is unfeasible if the weight is excessive (3D Packing in logistics).
    - Maximum time allowed to cut pieces (2D Packing for textile garments).

Goals of this Work

- Design, implement and evaluate (in quality and efficiency) algorithms that solve a flexible 2D variant of the Joint Problem:
  - 0/1 Knapsack scheme: items can be placed 0 or 1 times in the container, but cloned items can exist.
  - Both the items and the container can have different shapes:
    - Circles.
    - Ellipses.
    - Polygons (regular or irregular, concave or convex).
    - Multi-polygons: compound polygons with holes.
  - The methods should be abstract enough to facilitate their adaptation to the 3D variant, not planned for this work due to the expected long computation times (3D intersections).

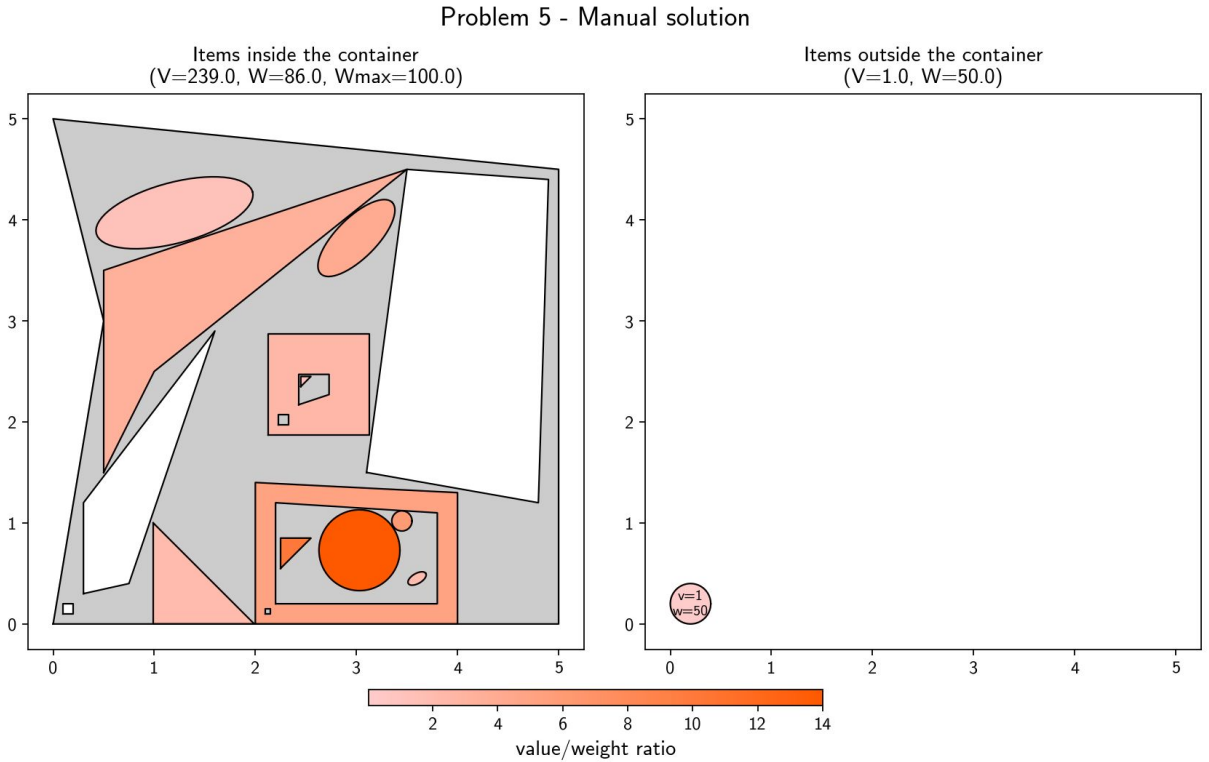
Example visualization

- Initial situation: the container is empty, with all items outside.



Example visualization

- Goal situation: the container is filled with objects, maximizing the value.



Previous Work

- Both the Packing Problem and the Knapsack Problem are optimization problems which have been traditionally solved with high quality using **evolutionary algorithms**: they escape local optima better than other approaches. [1]
- Since the Joint Problem is a combination of the previous problems, the main method of this work is also an evolutionary algorithm.
- Some existing evolutionary algorithms for the Knapsack Problem:
  - Represent chromosomes with bit strings (one bit per item: present or absent). [2]
  - Use a repair operator to remove the least profitable items when mutation or crossover leads to exceed the capacity. [3]
  - Use the container's value as the fitness function. [4]

Previous Work

- Some existing evolutionary algorithms for the Packing Problem:
  - **Discretize** all continuous variables: shapes (e.g. square compositions), positions (grid), and rotations (90 degrees), to have a finite search space and fast intersection checks. [5]
    - Drawback: loss of **precision** for non-trivial shapes, and limited positions and rotations ⇒ No discretizations in this work: a simplified (re)definition of the problem would be a risk for solution quality.
  - Use **multiple mutations**: place items, remove, move or rotate placed items. [5]
  - Define a crossover that **swaps items** in a random rectangular region [5], or preserves the common ones (if discretized the chromosome in a grid). [6]

Proposed Methods

- Three methods were developed in this work:
  - **Greedy** algorithm.
  - **Reversible** algorithm.
  - **Evolutionary** algorithm.
- The non-evolutionary methods were created to determine if simple techniques were sufficient to solve the Joint Problem.
- The **chromosome** used in the evolutionary algorithm is flexible enough to be adopted by the other two methods to represent a solution:
  - Hash table (**dictionary**) to map indices of placed items with their position and rotation.
- The rest of information of a problem is solution-independent so it resides in static data structures: the container's shape and capacity, plus the shape, value and weight of items.

Greedy algorithm

- Selects items stochastically, proportionally to a scoring function:

$$G(v, w, a) = \underbrace{K_v \cdot v}_{\text{reward items with high absolute value}} + \underbrace{(1 - K_v) \cdot (v \,/\, ((1 - K_a) \cdot w + K_a \cdot a))}_{\text{reward items whose value is high compared to their weight and area (penalize if they are high)}}$$

- Adds items to the container with random position and rotation.
- Can perform **early stopping** if:
  - No item (with feasible weight, respecting the limit) remains outside the container.
  - No item is successfully placed for many iterations (convergence assumption).
- As a last criterion, ends after a maximum number of iterations.

Reversible algorithm

- Selects items randomly, in a uniform way.
- Other than adding items, it can also (with less probability) remove objects or modify placements (randomly move or rotate), to **promote exploration**.
- Avoids losing long-term value by **reverting the solution** to the pre-removal state if a removal does not lead to higher value (via item additions) after some iterations.
- Uses the same termination criteria as the greedy algorithm.

Evolutionary algorithm – Main characteristics

- The **initial population** is generated by placing items in the container (at random positions and rotations) for some iterations (I_{max} = 300). To promote variability and **exploration**, all items should appear in some solutions, but some items are hard to place (e.g. large objects), so:
  - Each solution is first **specialized** in placing a specific item.
  - After that, items to place are **randomly selected**.
- Every generation, the **tournament strategy** is used to select the fittest individuals among small samples (pools) of the population (μ = 100 individuals) to become:
  - **Parents** of new offspring (λ = 200), generated using crossover and/or mutation.
  - **Survivors** of a generation (to restore the population size).
- Solution **feasibility** is always preserved: mutation and crossover operators are only confirmed when their solutions are valid (repair operations would be expensive).

Evolutionary algorithm – Main characteristics

- The **fitness** function returns the **container's value**. If there is a tie:
  - The chromosome with the least area used wins (area minimization). If there is a tie:
    - The individual whose items are in the smallest-area bounding rectangle wins (compaction).
- **Elitism** is used: the fittest individuals (E = 5) are preserved each generation, to avoid losing the best solutions (**exploitation**).
- The algorithm terminates after a fixed number of maximum generations (G_{max} = 30), but it can perform **early stopping** if the top fitness has not improved after some iterations (G_{conv} = 12), assuming **convergence**.

Evolutionary algorithm – Mutation

- The mutation operator performs **multiple iterations** ($I_{mut} = 5$), in which the following operations can be chosen (with different probability), selecting items randomly:
  - **Add** an item to the container. High probability ($P_a = 0.6$), since fitness (value) improves.
  - **Remove** an item from the container. Low probability ($P_r = 0.1$), since fitness decreases:
    - Only worth it when it frees space or weight for more profitable (future) items.
    - Each removal must be **compensated** by a later addition attempt in the same execution of the mutation operator.
  - **Modify** the position and/or rotation of a placed item, for exploration (probability: $P_m = 0.3$):
    - Random change.
    - Small change, i.e. small displacement or angle.
    - Swap of two items.
    - Move (in a random direction) or rotate (clockwise or counterclockwise) until the item would intersect with others or the container. Motivation: **compaction**.

Evolutionary algorithm – Crossover

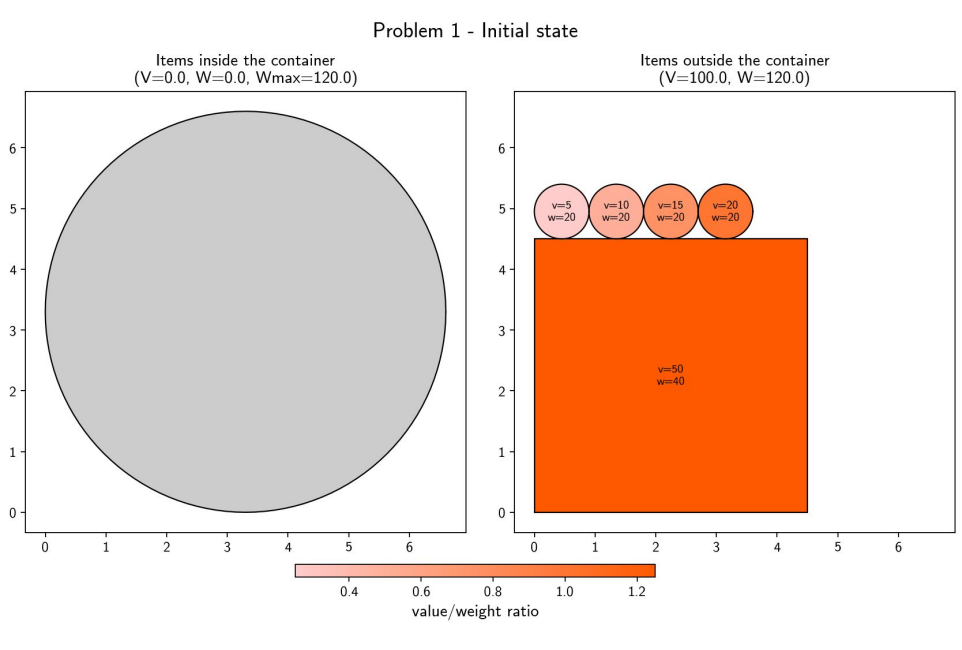
- Recombine two parents to create (at least) two offspring, following a process:
  1. Split the container in **two regions** (R1, R2), using either:
     a. Segment crossing the container.
     b. Shape (circle, ellipse or polygon): one region is inside, the other outside.
  2. Locate the **items completely contained** in each region for each parent.
  3. For offspring 1, copy the placements of items that parent 1 had in R1, and parent 2 in R2.
  4. For offspring 2, copy the placements of items that parent 1 had in R2, and parent 2 in R1.
  5. Discard duplicate items, so that they do not appear twice in an offspring.
  6. For items that were **intersecting** in the two regions of parents, try to place them in multiple orders in offspring (up to R = 5 permutations).
     a. Keep the two candidates with the highest fitness, and exceptionally more if their fitness is remarkable (better than a proportion C = 0.95 of the population, i.e. 95%).

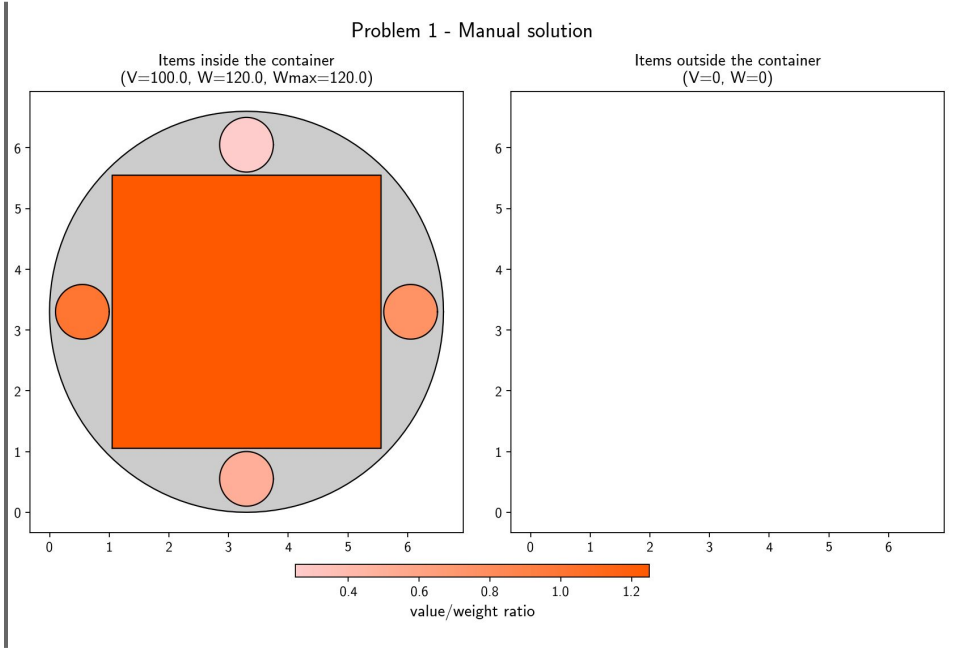
Joint Problem Dataset

- Set of **10 problems** created to test the algorithms that solve the Joint Problem, with manually obtained **optimal solutions**.
- Alternative: adapting a Packing dataset and imputing value and weights.
- However, a new dataset offered the possibility to face challenges specific of the Joint Problem.
- It was essential to represent a **diversity** of scenarios (e.g. difficulty levels), and a moderately high **variability** was achieved for many characteristics of the problems:

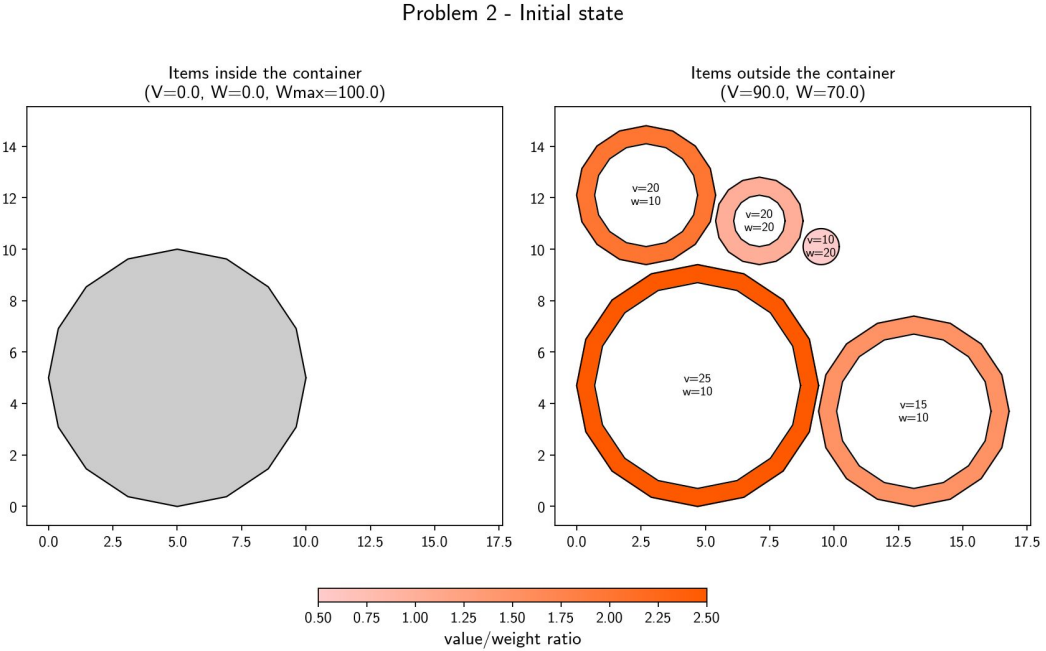
	Item num.	Opt. % item num. in cont.	Opt. % item value in cont.	Item weight % of max weight	Item area % of max area	Cont. weight saturation %	Cont. area saturation %
Min	5	6.67	51.55	70	22.88	70	16.54
Max	20	100	100	275	183.94	100	88.94
Mean	11.4	70.76	81.78	146.12	79.67	92.06	48.06
Std	5.02	33.85	17.15	60.44	49.46	9.67	24.13
Std/(max-min)%	33.44	36.27	35.39	29.48	30.71	32.23	33.33

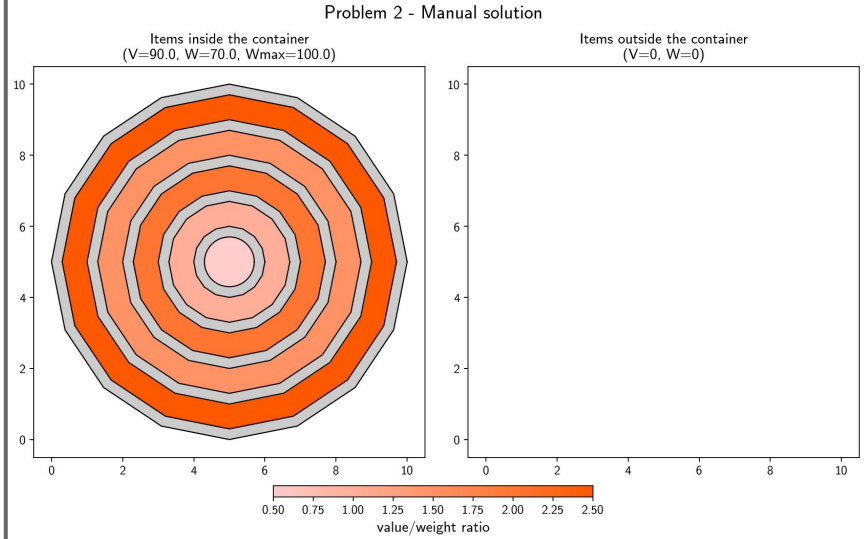
Problem 1


Problem 1 - Initial state
Items inside the container
(V=0.0, W=0.0, Wmax=120.0)
Items outside the container
(V=100.0, W=120.0)
v=5
w=20
v=10
w=20
v=15
w=20
v=20
w=20
v=50
w=40
value/weight ratio

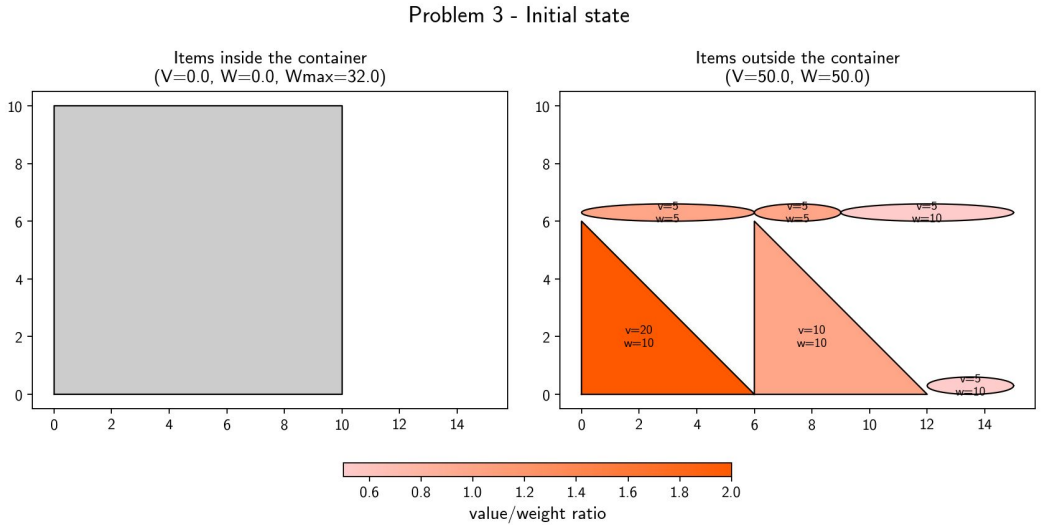

Problem 1 - Manual solution
Items inside the container
(V=100.0, W=120.0, Wmax=120.0)
Items outside the container
(V=0, W=0)
value/weight ratio

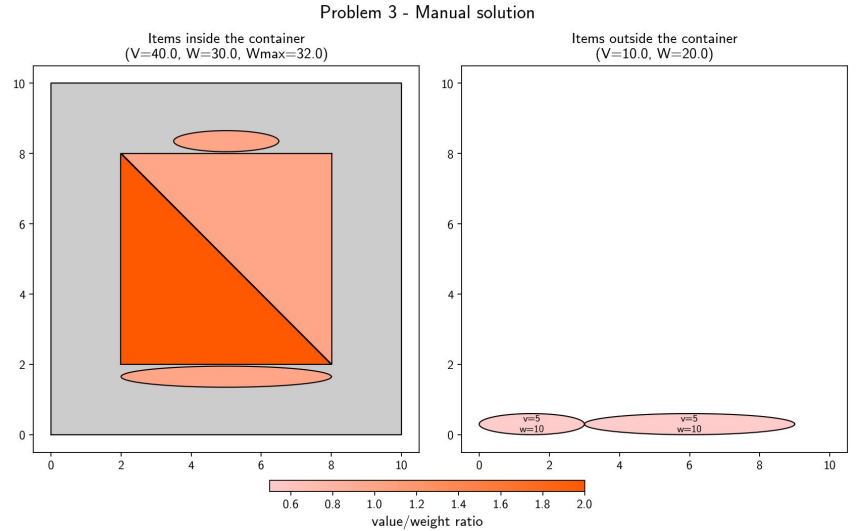
Problem 2


Problem 2 - Initial state
Items inside the container
(V=0.0, W=0.0, Wmax=100.0)
Items outside the container
(V=90.0, W=70.0)
v=20
w=10
v=20
w=20
v=10
w=20
v=25
w=10
v=15
w=10
value/weight ratio

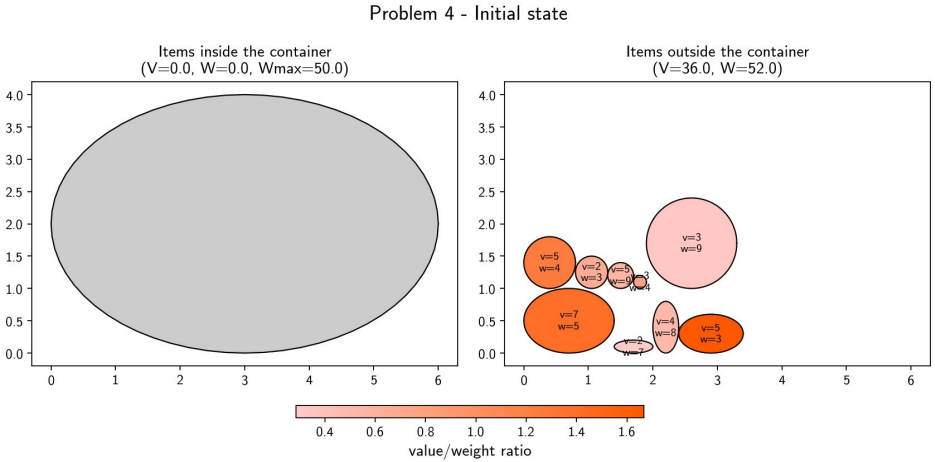

Problem 2 - Manual solution
Items inside the container
(V=90.0, W=70.0, Wmax=100.0)
Items outside the container
(V=0, W=0)
value/weight ratio

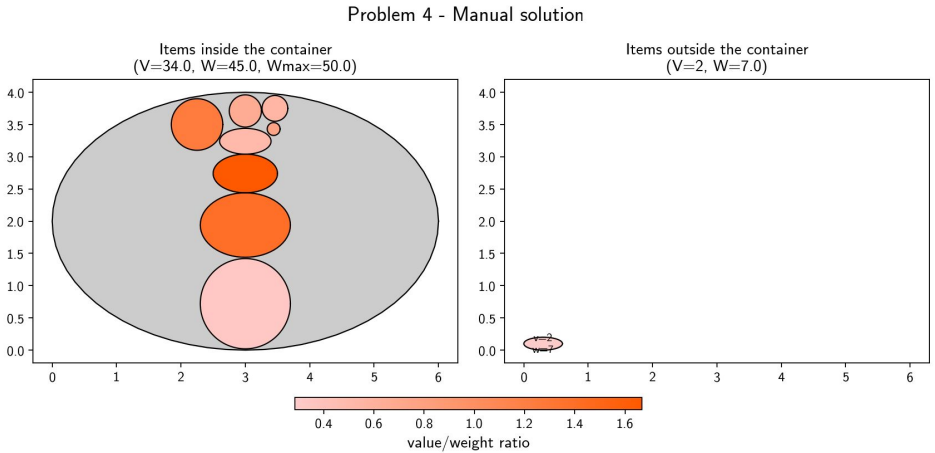
Problem 3


Problem 3 - Initial state
Items inside the container
(V=0.0, W=0.0, Wmax=32.0)
Items outside the container
(V=50.0, W=50.0)
v=5
w=5
v=5
w=5
v=5
w=10
v=20
w=10
v=10
w=10
v=5
w=10
value/weight ratio

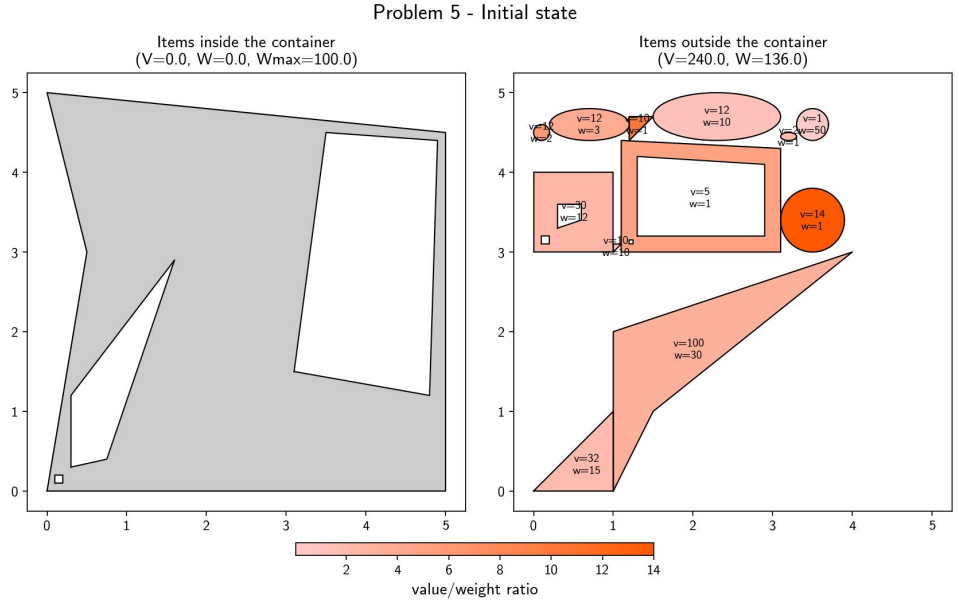

Problem 3 - Manual solution
Items inside the container
(V=40.0, W=30.0, Wmax=32.0)
Items outside the container
(V=10.0, W=20.0)
v=5
w=10
v=5
w=10
value/weight ratio

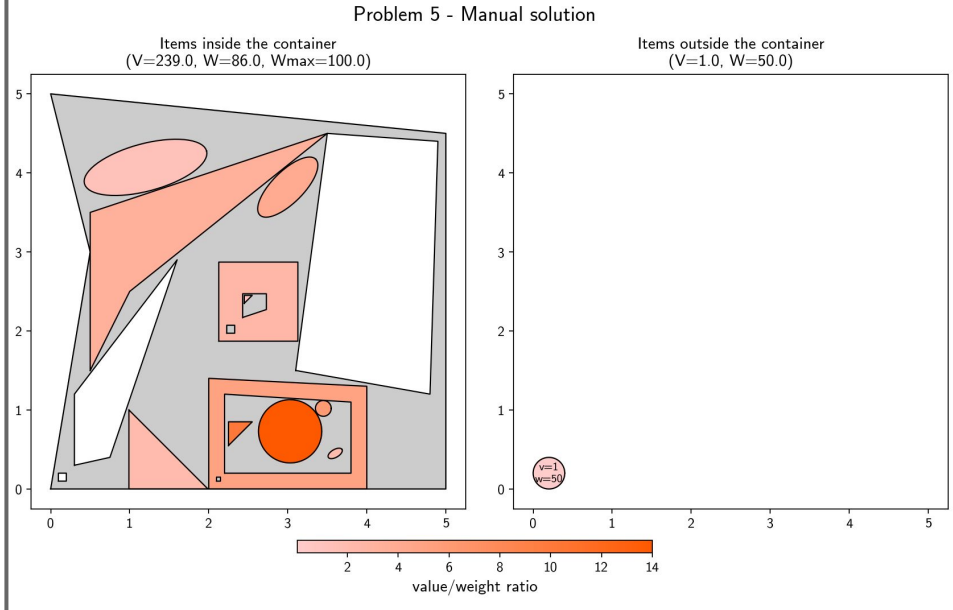
Problem 4


Problem 4 - Initial state
Items inside the container
(V=0.0, W=0.0, Wmax=50.0)
Items outside the container
(V=36.0, W=52.0)
v=5
w=4
v=2
w=3
v=3
w=9
v=7
w=5
v=4
w=8
v=5
w=3
value/weight ratio

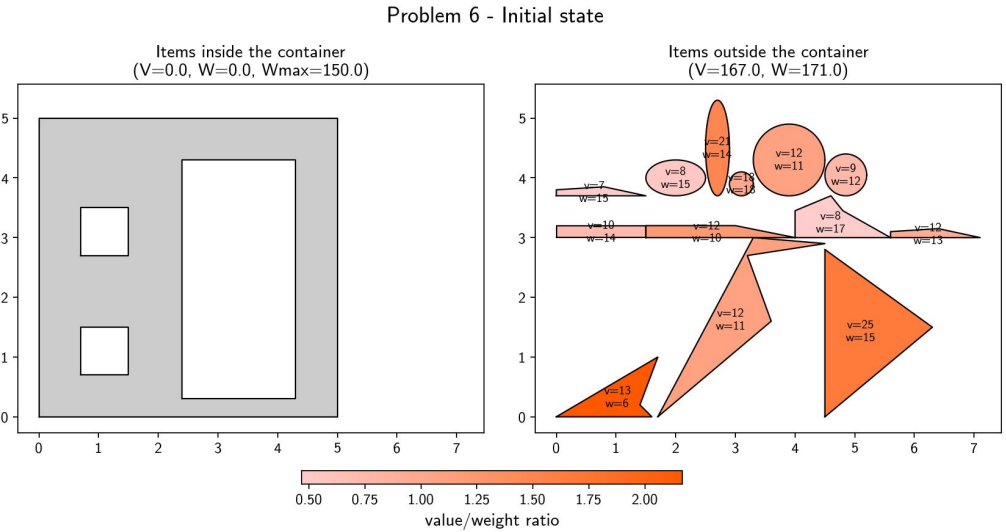

Problem 4 - Manual solution
Items inside the container
(V=34.0, W=45.0, Wmax=50.0)
Items outside the container
(V=2, W=7.0)
value/weight ratio

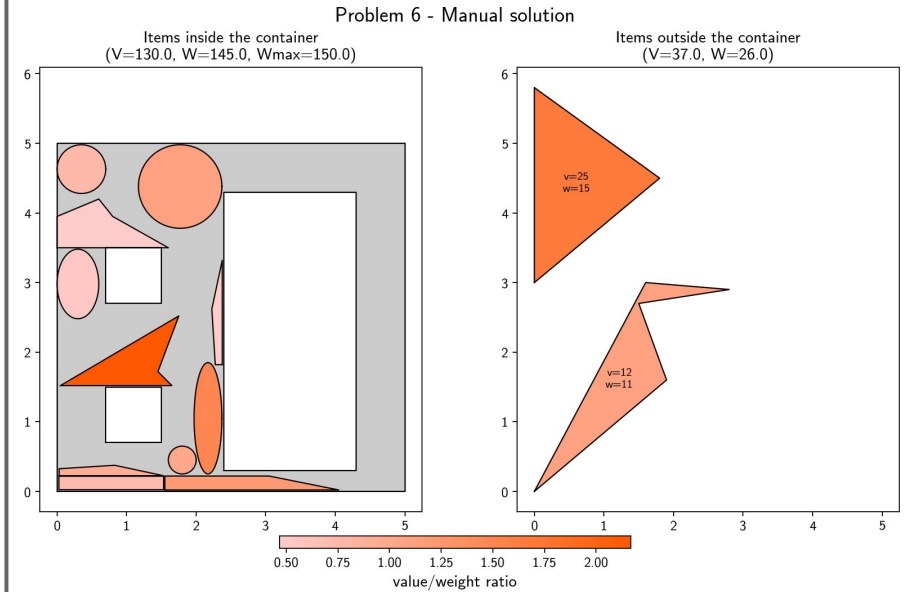
Problem 5


Problem 5 - Initial state
Items inside the container
(V=0.0, W=0.0, Wmax=100.0)
Items outside the container
(V=240.0, W=136.0)
v=12
w=3
v=12
w=10
v=1
w=50
v=5
w=1
v=30
w=12
v=14
w=1
v=100
w=30
v=32
w=15
value/weight ratio

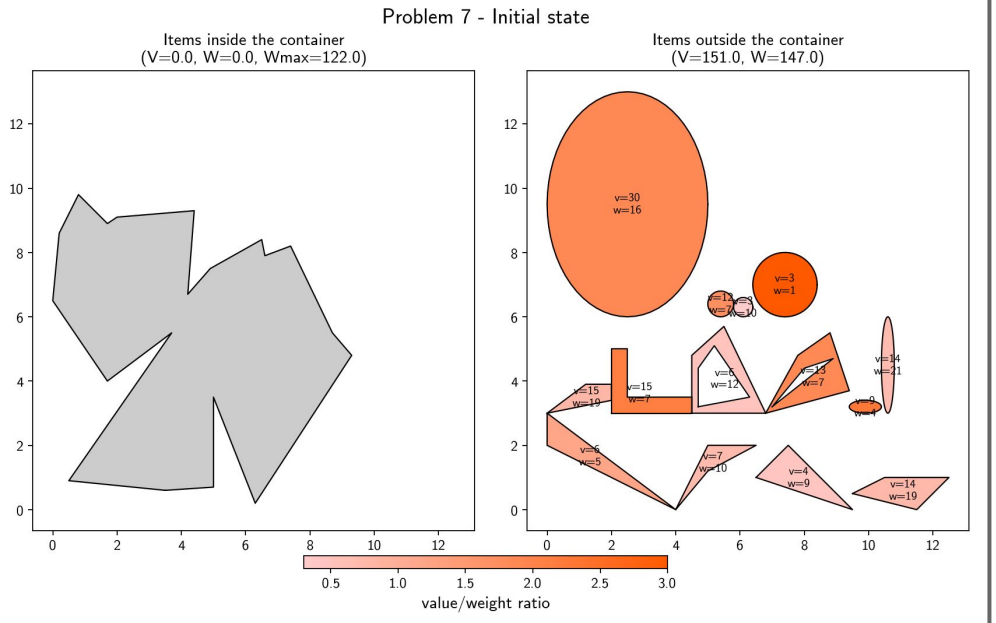

Problem 5 - Manual solution
Items inside the container
(V=239.0, W=86.0, Wmax=100.0)
Items outside the container
(V=1.0, W=50.0)
v=1
w=50
value/weight ratio

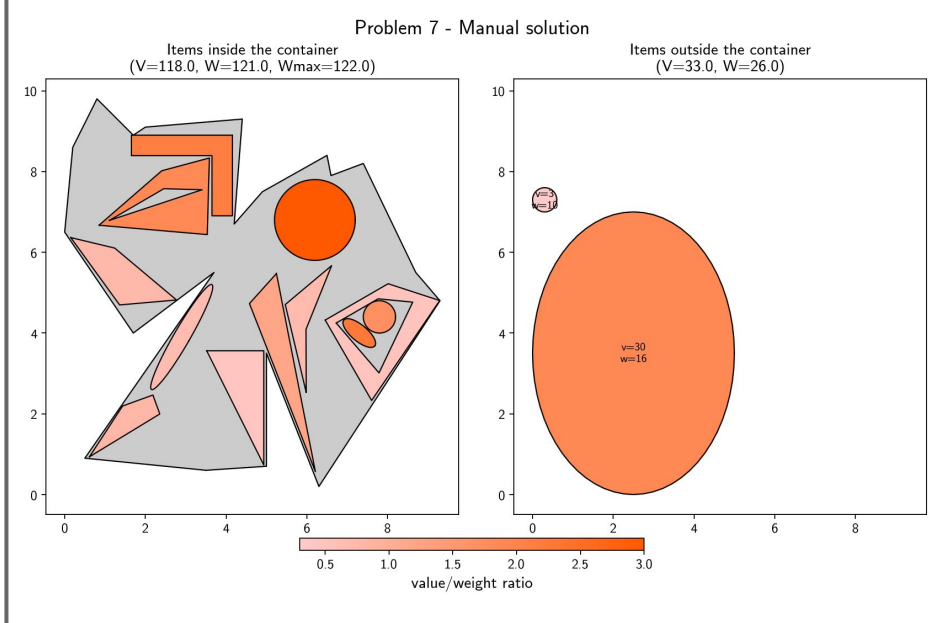
Problem 6


Problem 6 - Initial state
Items inside the container
(V=0.0, W=0.0, Wmax=150.0)
Items outside the container
(V=167.0, W=171.0)
v=21
w=14
v=12
w=11
v=9
w=12
v=8
w=15
v=7
w=15
v=8
w=17
v=10
w=14
v=12
w=10
v=12
w=13
v=12
w=11
v=25
w=15
v=13
w=6
0.50 0.75 1.00 1.25 1.50 1.75 2.00
value/weight ratio

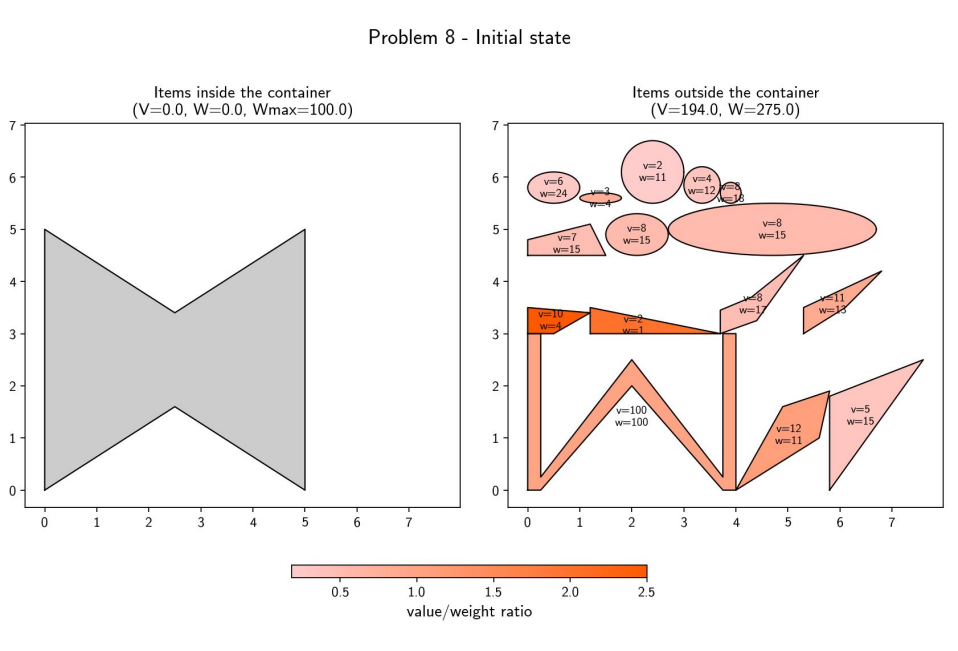

Problem 6 - Manual solution
Items inside the container
(V=130.0, W=145.0, Wmax=150.0)
Items outside the container
(V=37.0, W=26.0)
v=25
w=15
v=12
w=11
0.50 0.75 1.00 1.25 1.50 1.75 2.00
value/weight ratio

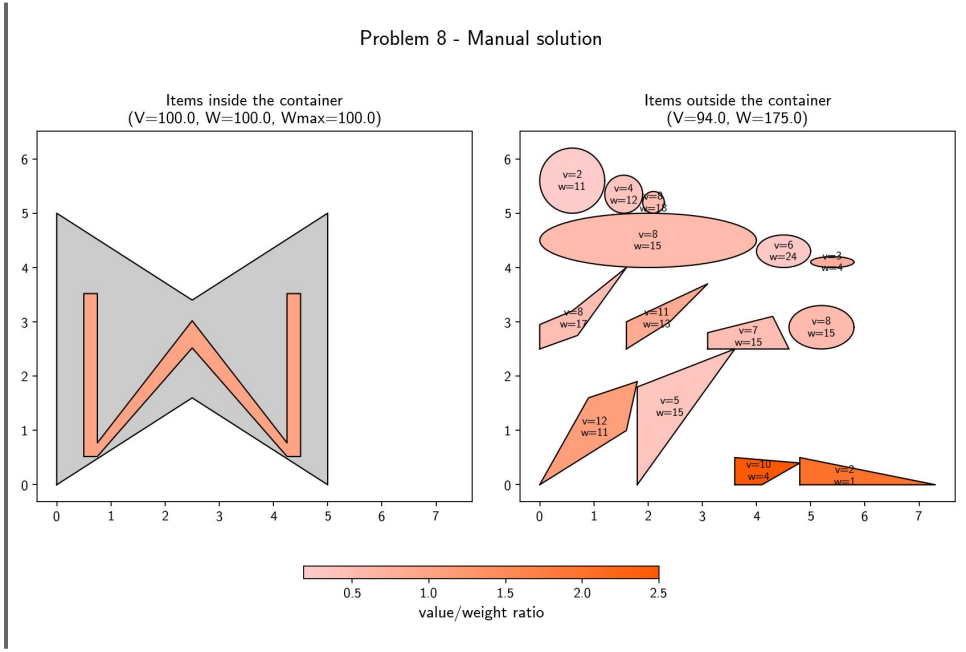
Problem 7


Problem 7 - Initial state
Items inside the container
(V=0.0, W=0.0, Wmax=122.0)
Items outside the container
(V=151.0, W=147.0)
v=30
w=16
v=3
w=1
v=15
w=10
v=15
w=7
v=6
w=12
v=13
w=7
v=14
w=21
v=9
w=4
v=6
w=5
v=7
w=10
v=4
w=9
v=14
w=19
value/weight ratio

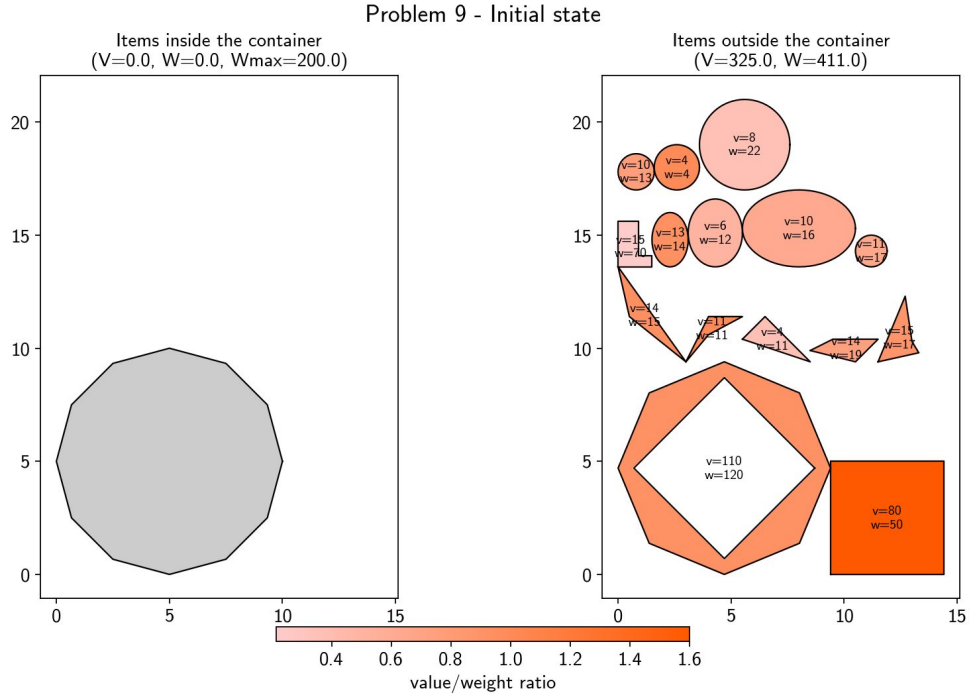

Problem 7 - Manual solution
Items inside the container
(V=118.0, W=121.0, Wmax=122.0)
Items outside the container
(V=33.0, W=26.0)
v=30
w=16
value/weight ratio

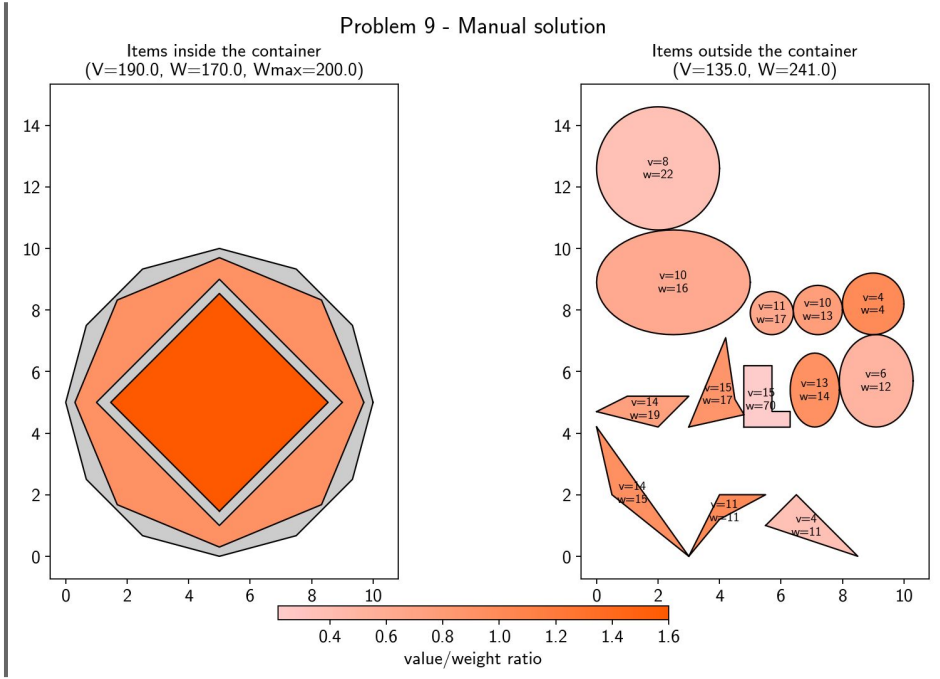
Problem 8


Problem 8 - Initial state
Items inside the container
(V=0.0, W=0.0, Wmax=100.0)
Items outside the container
(V=194.0, W=275.0)
v=6 w=24
v=3 w=4
v=2 w=11
v=4 w=12
v=8 w=18
v=7 w=15
v=8 w=15
v=8 w=15
v=10 w=4
v=2 w=1
v=8 w=17
v=11 w=13
v=100 w=100
v=12 w=11
v=5 w=15
value/weight ratio

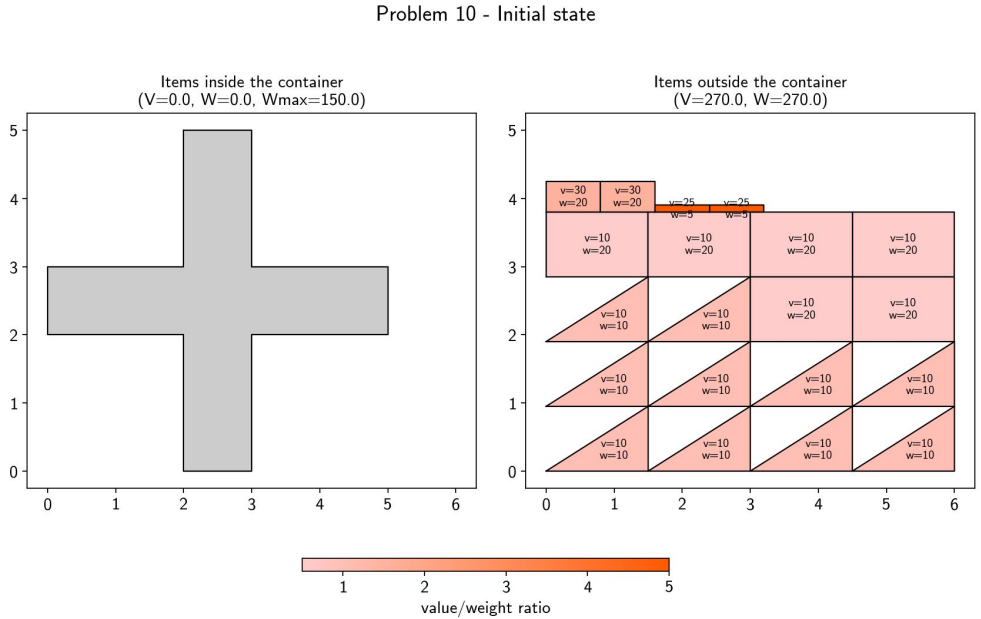

Problem 8 - Manual solution
Items inside the container
(V=100.0, W=100.0, Wmax=100.0)
Items outside the container
(V=94.0, W=175.0)
v=2 w=11
v=4 w=12
v=8 w=18
v=8 w=15
v=6 w=24
v=3 w=4
v=8 w=17
v=11 w=13
v=7 w=15
v=8 w=15
v=5 w=15
v=12 w=11
v=10 w=4
v=2 w=1
value/weight ratio

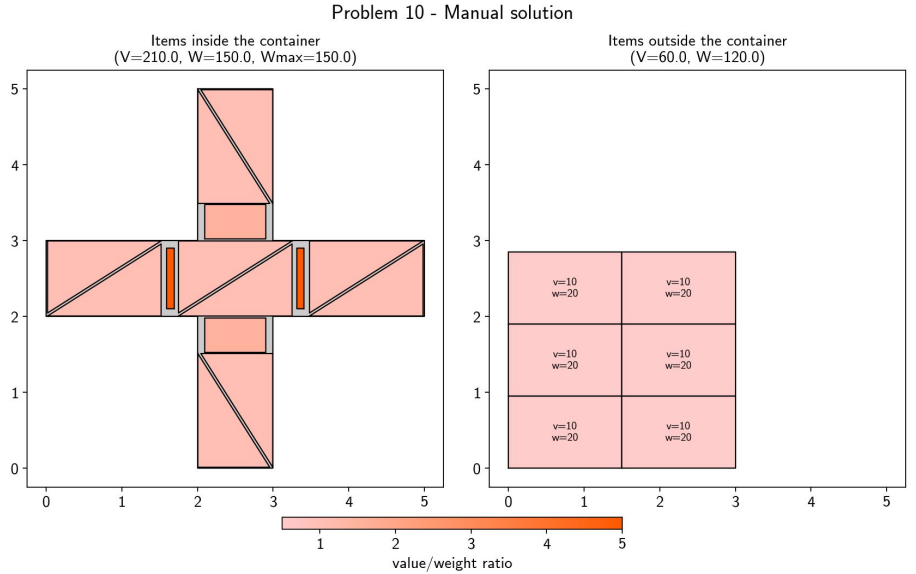
Problem 9


Problem 9 - Initial state
Items inside the container
(V=0.0, W=0.0, Wmax=200.0)
Items outside the container
(V=325.0, W=411.0)
v=8
w=22
v=10
w=13
v=4
w=4
v=15
w=70
v=13
w=14
v=6
w=12
v=10
w=16
v=14
w=15
v=11
w=11
v=4
w=11
v=14
w=10
v=15
w=17
v=110
w=120
v=80
w=50
value/weight ratio


Problem 9 - Manual solution
Items inside the container
(V=190.0, W=170.0, Wmax=200.0)
Items outside the container
(V=135.0, W=241.0)
v=8
w=22
v=10
w=16
v=11
w=17
v=10
w=13
v=4
w=4
v=15
w=17
v=15
w=70
v=13
w=14
v=6
w=12
v=14
w=19
v=14
w=15
v=11
w=11
v=4
w=11
value/weight ratio

Problem 10


Problem 10 - Initial state
Items inside the container
(V=0.0, W=0.0, Wmax=150.0)
Items outside the container
(V=270.0, W=270.0)
v=30 w=20
v=25 w=5
v=10 w=20
v=10 w=10
value/weight ratio


Problem 10 - Manual solution
Items inside the container
(V=210.0, W=150.0, Wmax=150.0)
Items outside the container
(V=60.0, W=120.0)
v=10 w=20
value/weight ratio

Experimental Methodology

- Two types of experiments were performed:
  - Parameter optimization.
  - Comparison of algorithms.
- In both cases, all algorithms were run **10 times** for **statistical significance**.
- The results (among different configurations or algorithms) were compared in terms of:
  - **Solution quality**, also compared to the manual optimal solutions.
  - **Execution time**, as a measure of efficiency.
- The experiments were run on the Joint Problem Dataset.
  - Ideally, it is advisable to perform parameter optimization and testing in different problems, but it was not done due to the long time of creating problems with manual solutions.
  - Additionally, the “optimized” parameters lead to similar results, and the changes are likely to be **universally beneficial** for problems in general (e.g. more iterations is better) so the risk of overspecialization seems very low.

Parameter Optimization

- Some parameters of the algorithms were optimized using **grid search**, and their effect in quality and time was studied.
- Unsurprising findings:
  - All algorithms perform better (yet slower) with more iterations (or generations), and the evolutionary algorithm also with greater population size and offspring size.
- Interesting findings for the evolutionary algorithm:
  - **Crossover does not improve** the results (nor make them worse).
    - Reason: the mutation operator is flexible enough to be able to generate any possible solution.
    - Reaction: for simplicity, crossover is not used in further experiments.
  - In mutation, **placement modifications** do not help to improve final solutions when present, but help to **converge faster** (so they are useful).
    - Reason: additions and removals alone can lead to any solution, but placement modifications are a shortcut for removing an item and then placing it back somewhere else.

Comparison of Algorithms – Results

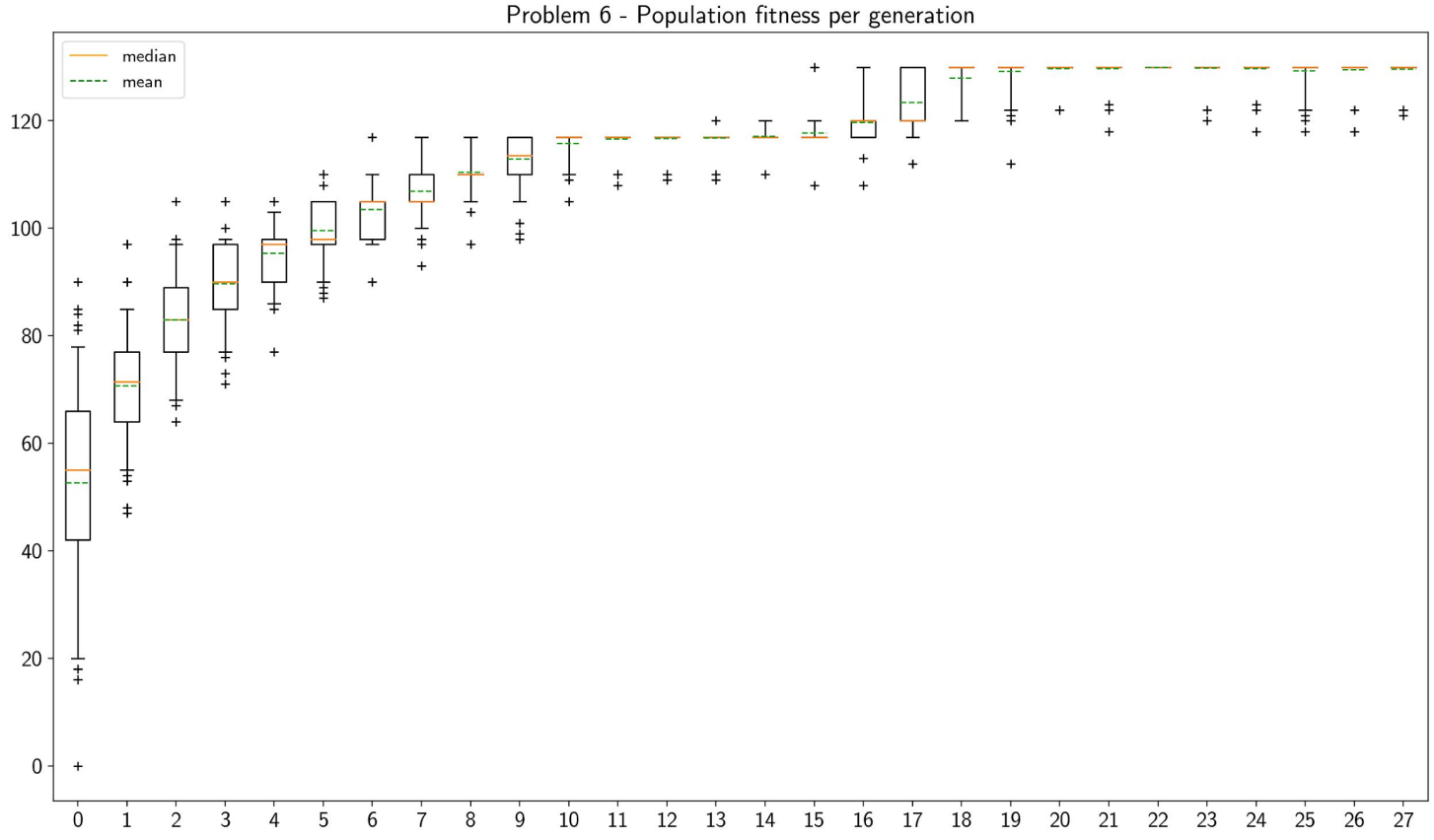
- The evolutionary algorithm obtains solutions of significantly **higher quality** than the other two.
- The evolutionary algorithm obtains **optimal** results in 7 of the 10 problems (at least once).

Algorithm	Greedy					Reversible					Evolutionary					Manual
Statistic	mean	std	min	med	max	mean	std	min	med	max	mean	std	min	med	max	optim.
Problem 1	50	0	50	50	50	50	0	50	50	50	**100**	0	**100**	**100**	**100**	100
Problem 2	50	10.49	30	50	75	50	3.16	45	50	55	**77.5**	6.42	**70**	**75**	**90**	90
Problem 3	23	5.1	20	20	35	22	5.1	15	20	35	**37.5**	2.5	**35**	**37.5**	**40**	40
Problem 4	32.7	0.78	31	33	**34**	30.1	1.81	29	29	**34**	**34**	0	**34**	**34**	**34**	34
Problem 5	130.1	4.97	123	134	134	111.6	11.73	103	105	139	**234**	0	**234**	**234**	**234**	239
Problem 6	104.9	6.41	97	105	118	110	10.1	97	109	**130**	**119.9**	5.77	**110**	**118**	**130**	130
Problem 7	110.3	6.39	96	113.5	115	103.4	6.89	91	105	115	**114.7**	1.95	**111**	**115**	**118**	118
Problem 8	53.8	5.93	45	55	60	49.9	7.63	42	48.5	65	**84.5**	15.5	**69**	**84.5**	**100**	100
Problem 9	109.3	2.61	105	111	113	104.4	5.62	94	103.5	113	**172.4**	4.43	**166**	**174**	**177**	190
Problem 10	136	4.9	**130**	**140**	140	132	8.72	120	130	150	**140**	6.32	**130**	**140**	**150**	210

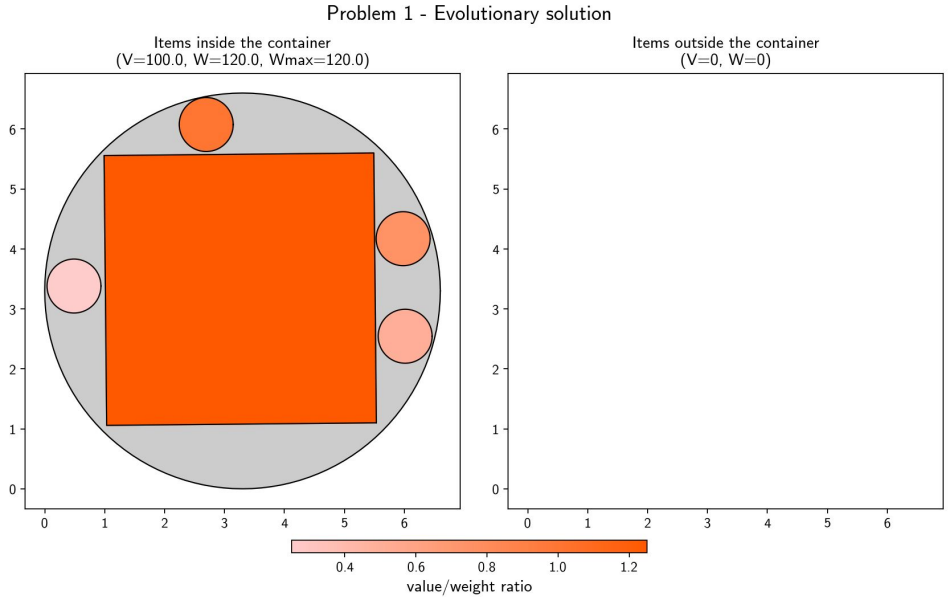
Results Discussion

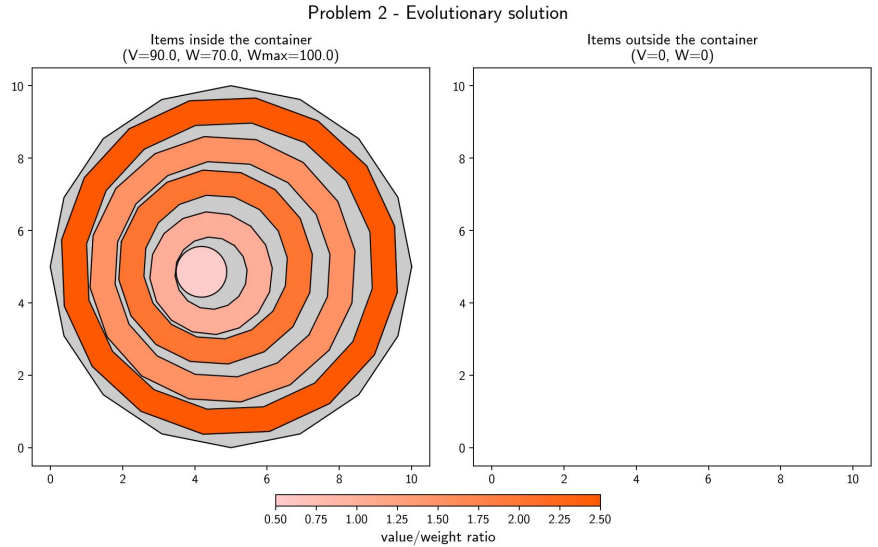
- The evolutionary algorithm is more effective because:
  - It creates, manages and updates **multiple solutions** (population) **at the same time**, while the greedy and reversible methods keep only one solution at a time.
    - Maintaining and applying operators to many solutions also makes the evolutionary algorithm significantly **slower**. Intersection checks are the slowest operations for all algorithms.
  - **Fitness-proportional** parent selection and population update promote the **refinement** of solutions, when mutation succeeds to place items (increasing value, i.e. fitness).
  - The best results are never lost, thanks to **elite keeping**.

Fitness evolution

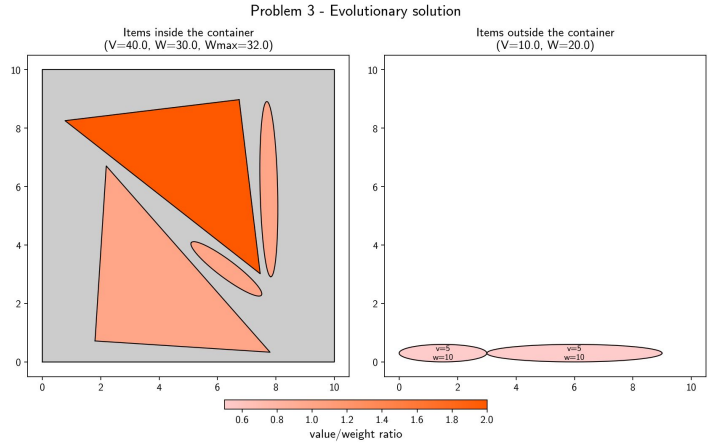


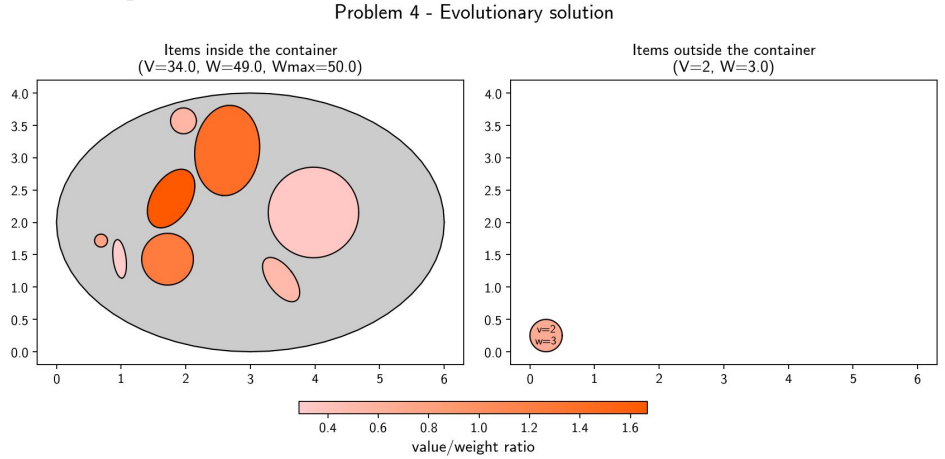
Evolutionary algorithm – Optimal solutions

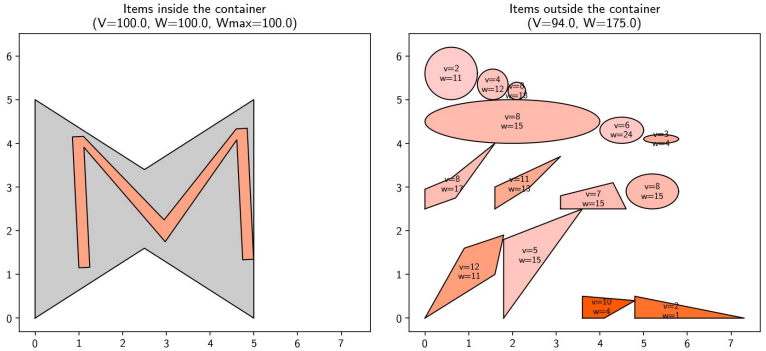




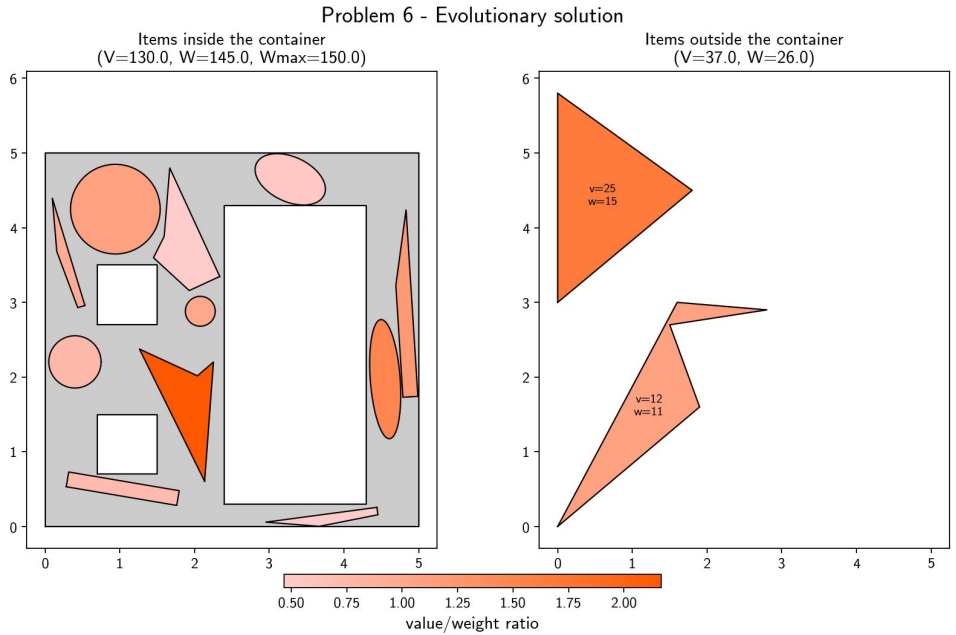
Evolutionary algorithm – Optimal solutions

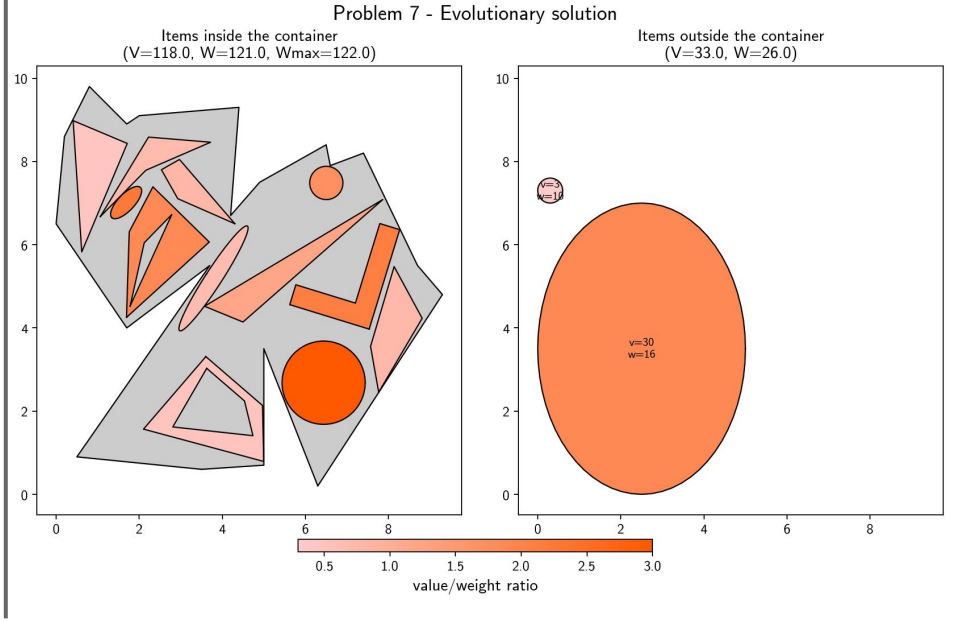






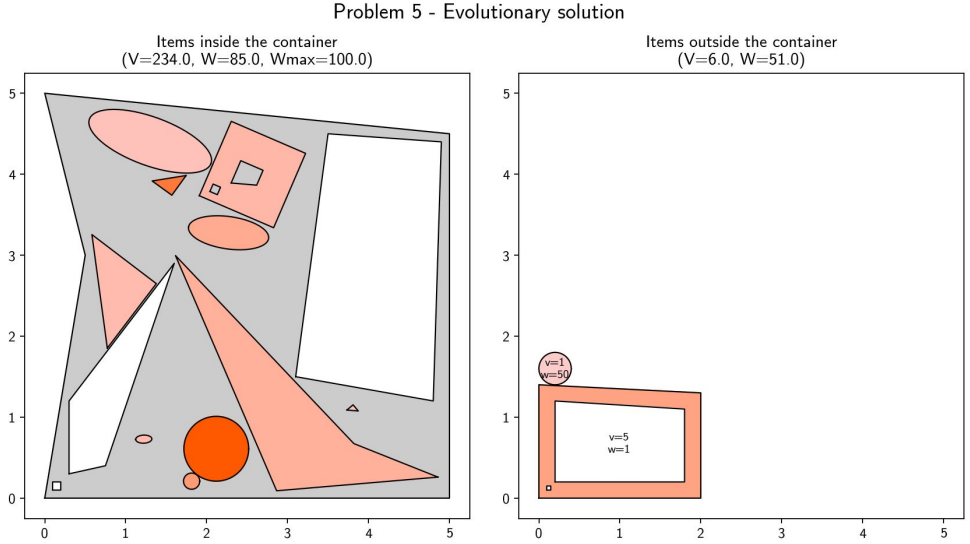
Evolutionary algorithm – Optimal solutions

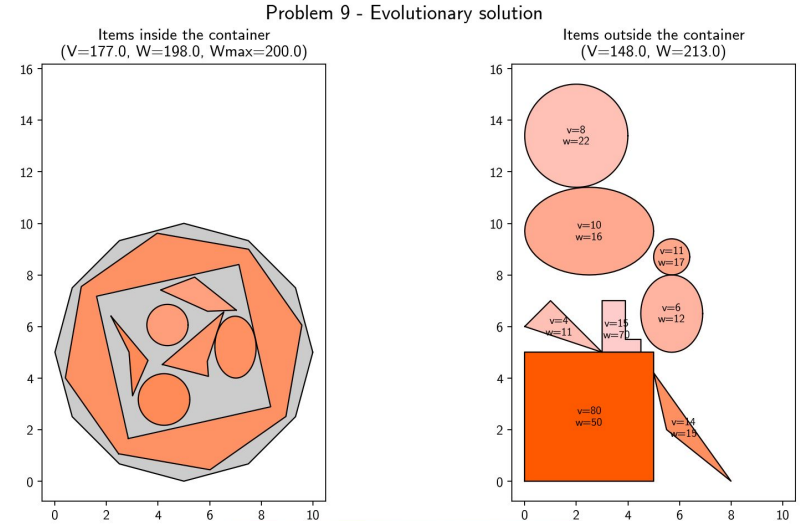




Evolutionary algorithm – Aspects to improve

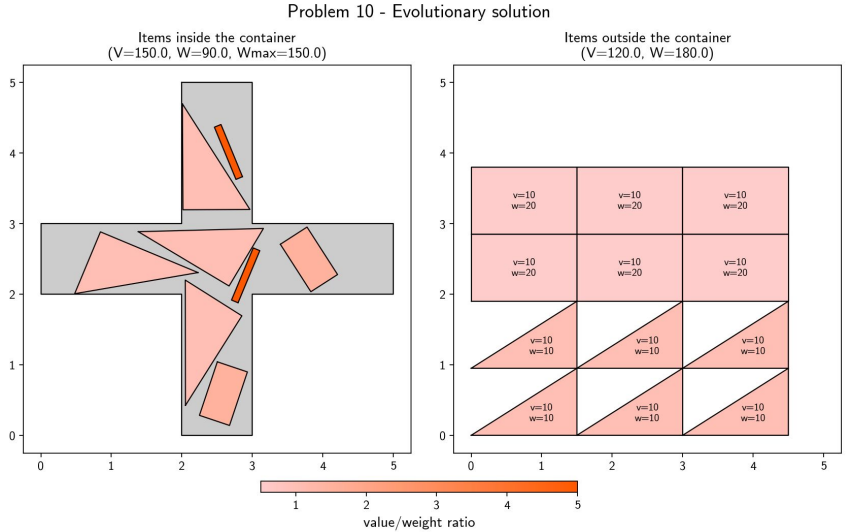
- Problems with more than one very large, **difficult-to-place item**:
  - If these items are not placed at start, it is very unlikely that later attempts will place them, once other items take part of the space.
  - When only one difficult object appears, the situation is solved thanks to the **item specialization** performed when generating the initial population.
  - Potential improvement: make specialization of **pairs of items** (or bigger sets), at least for large items.

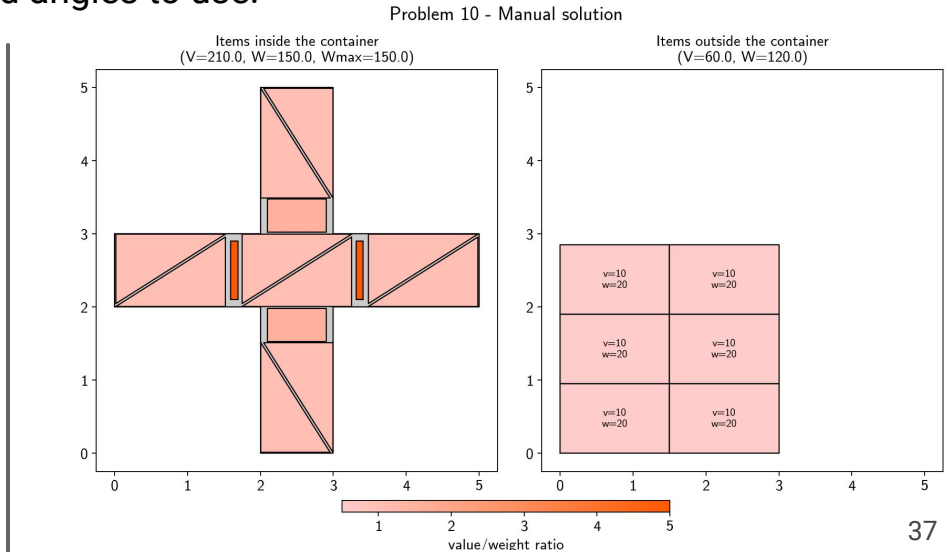




Evolutionary algorithm – Aspects to improve

- In the Joint Problem, the priority is to maximize the value, not to place items compactly according to some rules, e.g. keep items next to each other or next to the container bounds, if possible.
- However, problems with very little space in optimal solutions would need to **prioritize compaction**, adapting the cornering strategies used in some Packing algorithms [7].
- If a problem is known to require only **few possible rotations** (e.g. multiples of 90 degrees), the algorithm would be much more effective if told which limited angles to use.



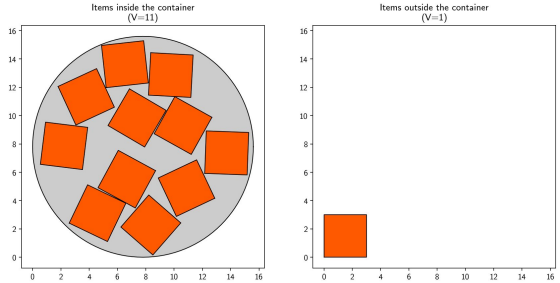


Packing Experiments and Results

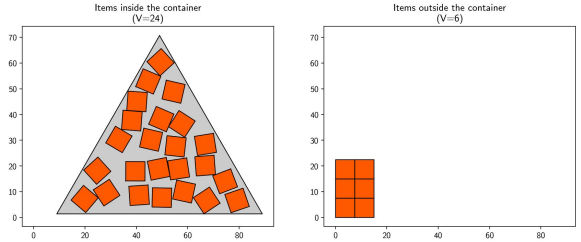
- To test its general applicability, the evolutionary algorithm was applied (without any modification) to a traditional **Packing problem**. Objective: maximize the number of items in the container, without value notions or weight limit.
- **Packing Dataset**: 9 problems with optimal solutions generated using Wolfram Alpha. [8]
- The evolutionary algorithm obtains optimal solutions in 5 of the 9 problems, and close-to-optimal ones in most other runs.
  - The results are satisfactory, knowing that it was not designed for the Packing Problem.
  - The new problems have simple shapes, so it is possible to restrict rotations:
    - Wolfram Alpha infers that mathematically, and it is faster and optimal.
    - If the **rotation restrictions** were specified in the evolutionary algorithm, it would also improve.
  - Wolfram Alpha is (currently) not capable of packing complex shapes, while the evolutionary algorithm supports irregular polygons, multi-polygons, etc.

Packing Solutions

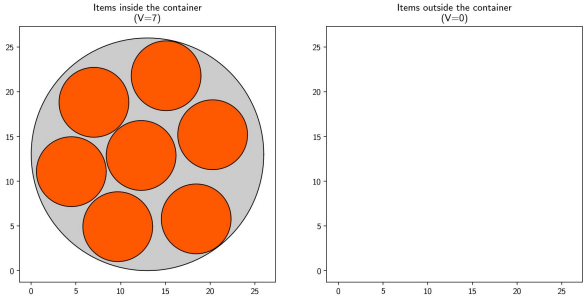
Squares in circle - Evolutionary solution



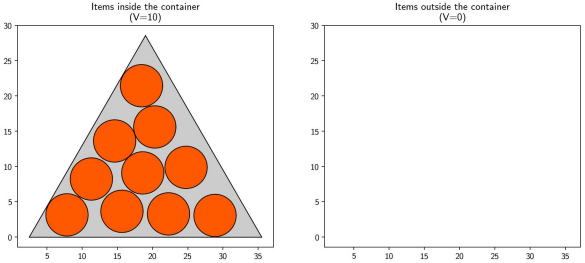
Squares in triangle - Evolutionary solution



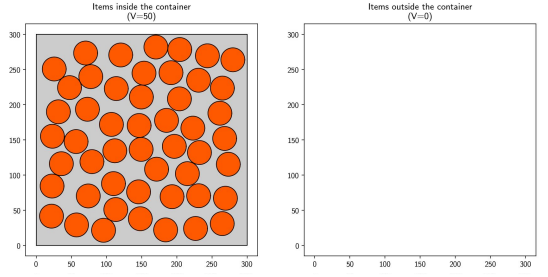
Circles in circle - Evolutionary solution



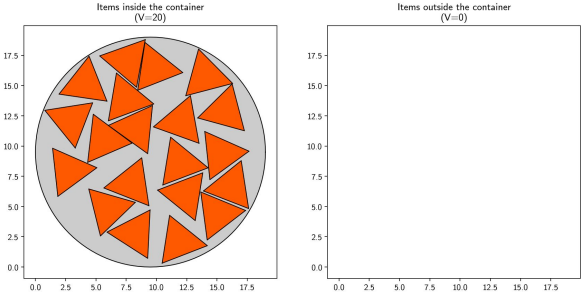
Circles in triangle - Evolutionary solution



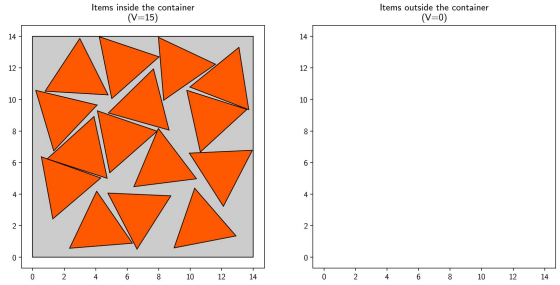
Circles in square - Evolutionary solution



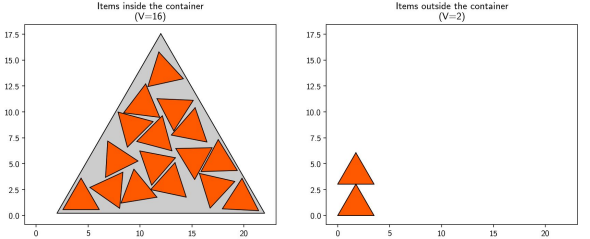
Triangles in circle - Evolutionary solution



Triangles in square - Evolutionary solution



Triangles in triangle - Evolutionary solution



Conclusions and Future Work

- The proposed evolutionary algorithm can solve the Joint Problem with **optimal** or **close-to-optimal solutions** in most of the tested problems, outperforming the alternative methods (greedy and reversible). The code is available in a repository to facilitate future research.
- The evolutionary algorithm can solve a traditional Packing Problem too (without changes).
- Hence, it should be possible to **adapt** the algorithm to the **3D Joint Problem**:
  - Replace 2D shapes with 3D ones (rectangular cuboids, spheres, meshes, etc.) and define their intersection checks.
  - Add domain-specific **constraints** when required:
    - Gravity and stability, e.g. in logistics, where objects cannot float and should not fall.
    - Fragility, e.g. when packing glass objects, that should not have heavy weight on top.
- Purely **mathematical methods** can be explored to **guarantee optimality**, but they are likely to be very difficult to define.
  - They are common in the Packing Problem (e.g. Wolfram Alpha [8]), but only for simple shapes, mainly circles, triangles, squares or rectangles.

References

- [1] SN Sivanandam and SN Deepa. Genetic algorithms. In Introduction to genetic algorithms, pages 15–37. Springer, 2008.
- [2] Zbigniew Michalewicz and Jaros law Arabas. Genetic algorithms for the 0/1 knapsack problem. In International Symposium on Methodologies for Intelligent Systems, pages 134–143. Springer, 1994.
- [3] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms a comparative case study. In International conference on parallel problem solving from nature, pages 292–301. Springer, 1998.
- [4] Arild Hoff, Arne Løkketangen, and Ingvar Mittet. Genetic algorithms for 0/1 multidimensional Knapsack problems. In Proceedings Norsk Informatikk Konferanse, pages 291–301. Citeseer, 1996.
- [5] Sakait Jain and Hae Chang Gea. Two-dimensional packing problems using genetic algorithms. Engineering with Computers, 14(3):206–213, 1998.
- [6] Ping Chen, Zhaohui Fu, Andrew Lim, and Brian Rodrigues. Two-dimensional packing for irregular shaped objects. In 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the, pages 10–pp. IEEE, 2003.
- [7] Eva Hopper. Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods. PhD thesis, University of Wales. Cardiff, 2000.
- [8] Inc. Wolfram Research. Geometric packing in 2d - wolfram—alpha examples, 2019. Accessed: 2019-12.

Thank you.

Do you have any questions?