Racel DAMECHLI
Clément CARON
Anahide CORLAY

ESILV
12/2023
DIA3-A4

AI Project – Biscuits Factory

TABLE DES MATIERES:

I – Description

Goals:

A biscuit factory aiming to maximize production and profit from a single roll of dough while considering the defects and different biscuit sizes and shapes. In other words, we would like to get:

- **Maximize Production**: Create various biscuits in different sizes and shapes from a single roll of dough to maximize production while ensuring the highest possible profit.
- **Optimize Placement**: Find the best arrangement of biscuits on the roll to maximize the value of the solution, which is the sum of the values of the individual biscuits placed on it.

To understand better the problem, we need to determine the constraints.

Constraints:

- **Roll of Dough**:
  - Predefined rectangular length referred to as 'LENGTH', representing a one-dimensional problem.
  - Contains irregularities referred to as defects, each with a position and a class (e.g., 'a', 'b', 'c', etc.).
- **Biscuits Production:**
  - Each biscuit has a specific size, value, and a threshold for the maximum number of defects of each class it can contain.
  - Biscuits can be produced an infinite number of times.
- **Solution Constraints**:
  - Biscuits must be placed at integer positions on the roll.
  - No overlap in biscuit placement.
  - Each biscuit must contain fewer or equal defects of each class than its threshold permits.
  - The sum of the sizes of the biscuits does not exceed the length of the roll.

The AI algorithm project aims to maximize biscuit production from a roll of dough by efficiently managing defects and ensuring optimal placement.

The project will involve defining and analyzing the production problem, formulating it into a computational model, developing and implementing solutions, followed by rigorous testing and optimization.

The defects.csv file plays a crucial role in informing the placement strategies to meet quality and efficiency goals.

- Defect Positions and Classes which lists each defect's position on the roll and its class.
- Defect Management in Solution: The CSV data will directly impact how biscuits are assigned on the dough roll, as each biscuit has a threshold for the maximum number of defects of each class it can contain

After describing the project and the different goals, we are going to depict our steps of implementation.

II – Implementation

In order to work on this project, we had to implement a few scripts to separate the different constraints and goals. We created “biscuits.py”, “bogo.py”, “backtracking.py”, “Greedy.py” and “Bee.py”. We used object-oriented programming to model the biscuits, defects, and the roll itself. The “bogo.py”, “backtracking.py”, “Greedy.py” and “Bee.py” are the two solving problem approaches. The librairy we used are numpy, csv, deepcopy and python-constraint. In the beginning we wanted to present only the performances of the greedy search and backtracking algorithms, as they’re used for constraint satisfaction problems. However, as we will discuss later, those two algorithms are deterministic. So, we wanted to see if by adding randomness to our optimization methods we could improve the solution. We chose to do the optimizing using the algorithm we worked on for the forum and an algorithm we designed.

Let’s describe those scripts:

Biscuits:

```
from csv import DictReader
from copy import deepcopy
import numpy as np
```

This is designed to simulate biscuit placement on a roll of dough in the presence of defects, aiming to maximize production based on the different constraints.

In Biscuits.py we find several functions and classes. The two classes are Roll and Biscuits. For the Biscuits it attributes a size and value and tolerance for defects for each biscuits permitting to know if the biscuits are valid or not. The Roll class helps to manage the placement of biscuits.

To help us place the biscuits in the easiest way possible on the roll we added important functions such as dict_sums_defects and fill_roll_random. The first will calculate the total number of each defect type within a given range which is crucial for determining if a biscuit can be placed in a specific section of the dough roll. The second will attempt to fill the roll with biscuits randomly, considering the size of the roll, the sizes of biscuits, and the placement constraints due to defects.

We separate two Biscuits versions. We had to modify the function to adapt our algorithm, and we decided to continue with the two versions. The Biscuits_clement works with Greedy and Backtracking algorithm, and Bisuits_racel works with ABC and Bogo algorithms.

Backtracking algorithm:

Imports:

```
from biscuits_clement import biscuit types, defects list, Roll
```

This code is an uninformed algorithm. It helps to figure out how to put as many biscuits as possible on a piece of dough to get the most value. We must make sure the biscuits don't overlap and that they fit well with any spots or defects on the dough. The method we use is backtracking. It tries every possible way to arrange the biscuits one by one. If we find out that the way we're trying doesn't work because the biscuits are overlapping or there are too many defects, we go back and try a different way.

Advantages of this approach:

- Complete
- Optimal
- Simple

In fact, this approach looks at every single possibility, so we know we aren't missing out on a better way to arrange the biscuits. It finds the best solution because it checks everything, it can find the very best way to arrange the biscuits, even though it might take a while.

Greedy research algorithm:

Imports :

```
from biscuits_clement import biscuit_types, defects_list, Roll
from constraint import Problem
```

This method is heuristic greedy research.

To start we had to download constraint library to define a problem where biscuits must be placed on a roll without overlapping and within defect tolerance.

In the same way as others, we added the different constraints, variables. We then did the greedy heuristic implementation. And finally, we calculated the values and total of biscuits.

Advantages of this approach:

- Efficiency:

The greedy heuristic is generally fast, as it makes local, optimal choices at each step, which can quickly lead to a good solution.

- Simplicity:

The approach is easier to understand and implement compared to some more complex optimization algorithms.

- Structured:

Using constraint programming provides a clear, structured way to define the problem, making the code potentially easier to adapt or extend.

Bogo algorithm:

Imports:

```
from biscuits_racel import Roll
```

This code snippet implements a simple, random approach to solving the biscuit placement problem using what's known as the "Bogo" algorithm. It was a first attempt to problem solving, it helped us to have a first view of a potential solution. The Bogo algorithm is incredibly straightforward. It provides a baseline for how well (or poorly) a purely random approach performs. This can be useful for comparison against more sophisticated methods.

ABC (artificial bee colony) algorithm:

Imports:

```
import numpy as np
from biscuits_racel import Roll, biscuit_types
from numpy.random import default_rng
```

We wanted for our second approach to use the Artificial Bees Colony (ABC) Algorithm we saw and implemented during the AI forum. We had a hard time to implement this algorithm into our specific biscuit problem, but we successfully solved the issues and got a result.

As a reminder the ABC algorithm works through detailed phases:

- Scouts Phase: Scout bees find new random solutions (new ways to place biscuits on the roll).
- Workers Phase: Worker bees exploit the current food sources (solutions) and try to find nearby, better solutions. If they can't improve for a certain number of tries, they abandon the food source and become scouts.
- Onlookers Phase: Onlooker bees watch the dances of the worker bees (which communicate the quality of the solutions) and choose food sources to exploit based on the quality.
- Iterative Improvement: The algorithm iteratively improves the solution by combining exploration (scouts finding new solutions) and exploitation (workers and onlookers improving existing solutions).

After a set number of iterations, the algorithm returns the best solution it found, which is the arrangement of biscuits on the roll with the highest total value.

The advantages of this approach:

- Exploration and Exploitation:

The ABC algorithm effectively balances exploring new solutions and exploiting known good ones, which can lead to finding better overall solutions than methods that do only one or the other.

- Flexibility:

It's relatively easy to adapt the ABC algorithm to different types of optimization problems, including the biscuit placement problem.

- Parallelizable:

Different bees can work simultaneously, making it suitable for parallel execution and potentially faster solutions.

III – Results and Conclusion

Results

Here are all the results of the number of biscuits on the roll and the total value of the best roll.

- **Backtracking and Greedy:**

```
In [24]: runfile('C:/Users/cleme/OneDrive/Bureau/Cours/A4/AI Algorithms/biscuits/backtracking.py', wdir='C:/Users/
cleme/OneDrive/Bureau/Cours/A4/AI Algorithms/biscuits')
Reloaded modules: biscuits
The total value of the best roll is: 643
Number of each biscuit type in the best arrangement:
<class 'biscuits.Biscuit'>: 65

In [25]: runfile('C:/Users/cleme/OneDrive/Bureau/Cours/A4/AI Algorithms/biscuits/griddy.py', wdir='C:/Users/cleme/
OneDrive/Bureau/Cours/A4/AI Algorithms/biscuits')
Reloaded modules: biscuits
The total value of the best roll is: 621
Number of each biscuit type in the best roll:
<class 'biscuits.Biscuit'>: 60
```

- **ABC:**

```
bees ×

/usr/bin/python3.11 bees.py
Best price is : 636, best number of biscuits is :147

Process finished with exit code 0
```

- **Bogo:**

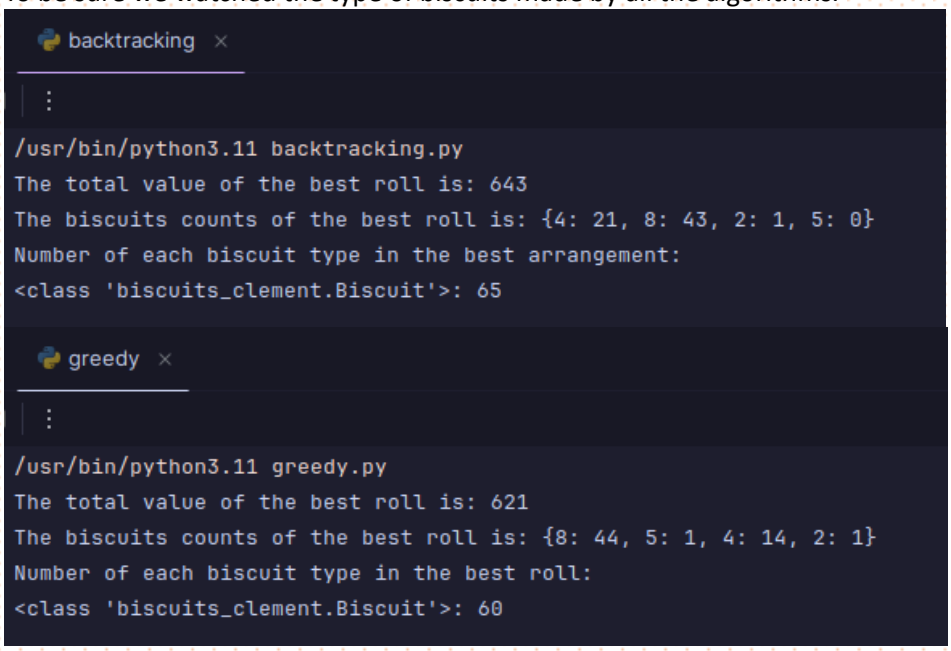
```
bogo ×

/usr/bin/python3.11 bogo.py
Best price is : 662, best number of biscuits is :133

Process finished with exit code 0
```

Conclusion

The Backtracking and Greedy seemed to give approximately the same results;
To be sure we watched the type of biscuits made by all the algorithms.



As we can see, even the type of biscuits used for the best roll are the same. However, the random algorithms tend to provide a better solution, with the drawback of time of execution, which is significantly longer.
In other words, the two algorithms we initially wanted to present were Backtracking and Greedy, but our implementations were deterministic, meaning they produced the same result with each execution, and they distributed the biscuits evenly across the roll. Therefore, we decided to experiment with Bogo search and the Artificial Bee Colony algorithm, as these last two algorithms are not deterministic due to their random factors.

Problems encountered

During this whole project we encountered a few issues. The first one was to adjust our ABC algorithm to the biscuits problem. We had to find a way to make it work well and this was a complex task. First, we had some results that were way too high (1400 total value for the best roll, although the maximum possible without checking for defects was 800). After long debugging sessions, we checked our functions and classes, and we succeeded in having a normal result. The issue was in the checking of biscuits constraints. Some biscuits would pass the checks when they were defective.

We also struggled to choose the best option upon the 4 methods used. The backtracking and greedy options seemed the best ones, we still found it more interesting and pertinent to take a non-deterministic yet precise approach such as ABC algorithm.

This project was very stimulating because it asked us to use all the tools we saw during this semester, and sometimes dig further and deeper.

We hope this project was as interesting to read as it was to do.

IV - Sources

https://en.wikipedia.org/wiki/Greedy_algorithm

https://www.programiz.com/dsa/greedy-algorithm

https://learning.devinci.fr/mod/forum/view.php?id=55615

https://learning.devinci.fr/mod/resource/view.php?id=38692

https://learning.devinci.fr/mod/resource/view.php?id=46636

https://jeffe.cs.illinois.edu/teaching/algorithms/book/02-backtracking.pdf

https://fr.wikipedia.org/wiki/Retour_sur_trace

https://www.scaler.com/topics/data-structures/backtracking-algorithm/

https://fr.wikipedia.org/wiki/Tri_stupide

https://www.geeksforgeeks.org/bogosort-permutation-sort/