Report LINGI2261: Assignment 4

Group N°29

Student1: Ben Haddou Mehdi

Student2: Hennebo Eliot

May 6, 2021

1 The bin packing problem (13 pts)

1. Formulate the bin packing problem as a Local Search problem (problem, cost function, feasible solutions, optimal solutions). **(1 pt)**

- Problem : We have a finite number of objects of integer size that we want to place in a finite number of bins of integer capacity. We want to find a way to place the objects in the bins so as to minimize the number of total bins needed to hold all the objects.
- Cost function : The function we are trying to minimize is $1 - (\frac{\sum_{i=1}^{n}(f_i/C)^2}{k})$, where k = number of bins, f_i = sum of all the pieces in bin i, and C = bin capacity.
- Feasible solutions : A solution is feasible if every item is in a bin and the fullness of every bin is lower or equals to its capacity.
- Optimal solutions : A solution is optimal if for each other solution of the problem, the optimal solution has a smaller fitness function.

2. You are given a template on Moodle: *binpacking.py*. Implement your own extension of the *Problem* class from `aima-python3`. Implement the `maxvalue` and `randomized maxvalue` strategies. To do so, you can get inspiration from the *randomwalk* function in *search.py*. Your program will be evaluated on 15 instances (during 1 minute) of which 5 are hidden. We expect you to solve at least 12 out the 15.

   (a) `maxvalue` chooses the best node (i.e., the node with minimum value) in the neighborhood, even if it degrades the quality of the current solution. The `maxvalue` strategy should be defined in a function called *maxvalue* with the following signature: `maxvalue(problem,limit=100,callback=None)`. **(2.5 pts)**

The program has been submitted and passed 14/15 instances.

(b) `randomized maxvalue` chooses the next node randomly among the 5 best neighbors (again, even if it degrades the quality of the current solution). The `randomized maxvalue` strategy should be defined in a function called *randomized_maxvalue* with the following signature: `randomized_maxvalue(problem,limit=100,callback=None)`. **(2.5 pts)**

The program has been submitted and passed 14/15 instances.

3. Compare the 2 strategies implemented in the previous question and `randomwalk` defined in *search.py* on the given bin packing instances. Report, in a table, the computation time, the value of the best solution (in term of fitness) and the number of steps needed to reach the best result. For the randomized max value and the random walk, each instance should be tested 10 times to reduce the effects of the randomness on the result. When multiple runs of the same instance are executed, report the mean of the quantities. **(3 pts)**

	Max value			Random max value			Random walk		
Inst.	T(ms)	Fit.	NS	T(ms)	Fit.	NS	T(ms)	Fit.	NS
i01	239	0.2205	9	1181	0.2205	47.4	758	0.3302	24.1
i02	269	0.2253	10	1181	0.2253	48.2	149	0.3000	4.7
i03	168	0.2311	7	951	0.2314	33.7	1183	0.3164	43.7
i04	156	0.2337	6	936	0.2341	37.5	343	0.3036	12.0
i05	45	0.2473	2	453	0.2473	18.7	56	0.2569	1.4
i06	171	0.2223	6	1355	0.2223	45.8	77	0.2599	2.3
i07	165	0.2334	6	1481	0.2333	58.8	169	0.2778	6.1
i08	105	0.2296	4	1079	0.2295	44.5	421	0.2944	15.9
i09	186	0.2114	7	1081	0.2114	43.3	166	0.2563	6.0
i10	179	0.2346	7	494	0.2345	20.5	102	0.2778	3.2

NS: Number of steps — **T**: Time — **Fit.**: Fitness

4. **(4 pts)** Answer the following questions:

   (a) What is the best strategy? **(1 pt)**

We can clearly say that the MaxValue and RandomizedMaxValue strategies provide better results than RandomWalk. Nevertheless in this exercise, the results are relatively similar for both main strategies, at least only in terms of fitness because the time and steps are less good in randomized.

(b) Why do you think the best strategy beats the other ones? **(1 pt)**

In this problem MaxValue is the best implementation of the hill-climbing algorithm in term of time and step, which is in constant search of the best solution, this results in the fact that it finds it faster because only a few steps are needed and it automatically tries to go to the most optimal solution at the risk of being blocked in a local optimum. The other strategies do not necessarily choose greedily the best neighbor at each step, which makes the number of steps and the time longer.

(c) What are the limitations of each strategy in terms of diversification and intensification? **(1 pt)**

MaxValue is focused on intensification, it will only look for the neighbor with the best fitness value without taking into account diversification, which can lead it to fall into a local optimum.
RandomizedValue is a balanced strategy because it couples intensification and diversification, by taking "one of the best neighbors" we leave ourselves a chance not to fall into a local optimum and especially to to potentially get out of it.
RandomWalk is totally focused on diversification, its lack of intensification makes it mediocre.

(c) What is the behavior of the different techniques when they fall in a local optimum? **(1 pt)**

MaxValue has no chance to get out of a local optimum since it will never choose a less satisfactory result than the one it has already reached.
RandomizedMaxValue is better suited to this kind of situation, it has a chance to get away with going to a neighbor on another hill, if the optimum local is not too pronounced and the neighbors are still diverse enough and not all stuck on the same hill.
RandomWalk has such a different behavior that it is hardly likely to be stuck in a local optimum, if this happens it will jump to another hill since it will not prioritize a neighbor in the optimum.

2 Propositional Logic (7 pts)

2.1 Models and Logical Connectives (1 pt)

Consider the vocabulary with four propositions A, B, C and D and the following sentences:

- $(\neg A \vee C) \wedge (\neg B \vee C)$
- $(C \Rightarrow \neg A) \wedge \neg(B \vee C)$
- $(\neg A \vee B) \wedge \neg(B \Rightarrow \neg C) \wedge \neg(\neg D \Rightarrow A)$

1. For each sentence, give its number of valid interpretations, i.e. the number of times the sentence is true (considering for each sentence **all the proposition variables** A, B, C and D). **(1 pt)**

We create the truth tables and count the number of rows returning true :

$(\neg A \vee C) \wedge (\neg B \vee C)$: 5
$(C \Rightarrow \neg A) \wedge \neg(B \vee C)$: 2
$(\neg A \vee B) \wedge \neg(B \Rightarrow \neg C) \wedge \neg(\neg D \Rightarrow A)$: 1

2.2 Color Grid Problem (6 pts)

1. Explain how you can express this problem with propositional logic. For each sentence, give its interpretation. **(2 pts)**

- For each cell at least one color must be used :
  $\bigwedge_{\substack{0\leq i<n \\ 0\leq j<n}} \left(\bigvee_{0\leq k<n} C_{i,j,k} \right)$
- Each cell implies that you cannot have a cell of the same color on the same row, or on the same column, or on the same top left diagonal, or on the same top right diagonal :
  $C_{i,j,k} \implies \bigwedge_{0<d<=i} \left(\neg C_{i-d,j,k} \wedge \neg C_{i,j-d,k} \wedge \neg C_{i-d,j-d,k} \wedge \neg C_{i-d,j+d,k} \right)$
- Each fixed input have must be included :
  $\bigwedge_{(i,j,k)\in \text{Points}} C_{i,j,k}$.

2. Translate your model into Conjunctive Normal Form (CNF). **(2 pts)**

$$\bigwedge_{\substack{0\leq i<n \\ 0\leq j<n \\ 0\leq k<n}} \left(\bigwedge_{0<d<=i} \underbrace{\left(\neg C_{i,j,k} \vee \neg C_{i-d,j,k}\right)}_{\text{row}} \wedge \underbrace{\left(\neg C_{i,j,k} \vee \neg C_{i,j-d,k}\right)}_{\text{column}} \wedge \underbrace{\left(\neg C_{i,j,k} \vee \neg C_{i-d,j-d,k}\right)}_{\text{top left diagonal}} \wedge \underbrace{\left(\neg C_{i,j,k} \vee \neg C_{i-d,j+d,k}\right)}_{\text{top right diagonal}} \right) \wedge \underbrace{\bigwedge_{\substack{0\leq i<n \\ 0\leq j<n}} \left(\bigvee_{0\leq k<n} C_{i,j,k} \right)}_{\text{at least one color}} \wedge \underbrace{\bigwedge_{(i,j,k)\in \text{Points}} C_{i,j,k}}_{\text{points}}$$

3. Modify the function `get_expression(size)` in `cgp_solver.py` such that it outputs a list of clauses modeling the color grid problem for the given input. Submit your code on INGInious inside the *Assignment4: Color Grid Problem* task. The file `cgp_solver.py` is the *only* file that you need to modify to solve this problem. Your program will be evaluated on 10 instances of which 5 are hidden. We expect you to solve at least 8 out the 10. **(2 pts)**

The program has been submitted and passed 10/10 instances.