Report LINGI1361: Assignment 4

Group N°160

Student1: Camille Caulier 24701800

Student2: Noah Meluzola 85302100

May 12, 2022

1 The cover vertex problem (13 pts)

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

- problem: we have an initial coverage with K nodes covering N edges. Our goal is to maximise N such that N equals the total amount of edges in the graph. This is done by finding permutations of K.
- cost function: number of edges taken (we aim to maximise the cost function).
- feasible solutions: all solutions are feasible as long as we have different nodes in the cover; The cover must be of size K (given in the instance) and be K different nodes (having double nodes would not make any sense).
- optimal solutions: all covers that take all edges are considered as optimal as they cover all the edges.

2. You are given a template on Moodle: *vertexcover.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 maximal 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)**

No comment

(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)**

No comments

3. Compare the 2 strategies implemented in the previous question and `randomwalk` defined in *search.py* on the given vertex cover instances. Report, in a table, the computation time, the value of the best solution 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(s)	Val	NS	T(s)	Val	NS	T(s)	Val	NS
i01	0.00093	7	1	0.0008749	7	4.6	0.0006106	6.9	15.4
i02	0.0163	16	2	0.0160946	16	6.6	0.009924	15.2	109.6
i03	0.0139	13	3	0.0139057	13	6.8	0.008352	12.2	41.8
i04	0.002698	4	0	0.0026130	4	3.1	0.0020891	4.0	4.3
i05	0.470	29	3	0.4735385	29	3.2	0.3006227	27.6	199.9
i06	0.2791	2	1	0.2849679	2	3.1	0.2594414	1.1	48.9
i07	0.22894	18	9	0.2332343	18	20.2	0.16913	16.0	100.0
i08	0.055	18	5	0.0543563	18	11.9	0.03293	15.4	132.8
i09	0.983	29	4	0.9880252	29	4.4	0.810971	26.4	218.2
i10	0.43973	24	6	0.4292162	24	12.1	0.36122	19.3	188.0

NS: Number of steps — **T**: Time — **Val**: Value of the best solution NOTE: for all strategies I gave them a max step value of nVertices * nEdges. In the table I recorded the step value of when the best value was found.

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

   (a) What is the best strategy? **(0.5 pt)**

From what can be noticed, the best strategy is the maxvalue as it finds the solution in less steps, and find most of the time the optimal solution (in inginious there was one instance where maxvalue was not able to find an opitmal solution but randommax was able to) but randommax value is able to guarantee a better chance of getting the optimal coverage. Thus randommax is better although it is slightly slower as to assure that we are not stuck in a local optimum.

(b) Why do you think the best strategy beats the other ones? What conclusions can be drawn on the vertex cover problem ? **(1.5 pt)**

The successor function gives a wide and broad selection of K*(N-K) nodes . This in turn, allows us quickly find with very few steps the optimal solution. Since there exist local optimums maxvalue could get stuck where as since randommax allows for diversity by taking the amongst the five best neighbours it allows for it to be direct yet at the same time easy to escape local maximums. Since maxvalue is too direct this can be a problem if a graph was created and given a bad initial coverage such that maxvalue would be stuck in a local maximum. On the other hand, randomwalk is too diverse to the point that we can most certainly say that we won't be able to find the solution unless we run an extremely large amount of step thus making it not a viable option.

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

For the maxvalue it is the most intense out of all the others, its only means of diversification is from its successors function where it produces K*(N-K) nodes. This has proven sufficient in solving the problem. The only caveat would be if there was a very well placed local maximum such that max value would loop due to the initial coverage and other factors. For randommax it diversifies itself by randomly choosing out of the best 5 neighbours, this allows to explore new paths and perhaps break itself out of a loop (if it is stuck in one). This has also proven efficient as it has solved an instance that the maxvalue was unable to do. The only problem is that it takes more steps and is a question of how likely we are of finding the optimal cover. Since the problem is straight forward this has not been a problem. For randomwalk, we completely rely on pure luck that we will stumble upon the correct solution since its diversification is too great that it is not guided at all to the solution

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

Maxvalue will continue to loop if the problem if the conditions are met. Other wise it will continue to wander around. For randommax it will loop out of local optimum, eventually if possible (if not a bigger choice will be needed to select from) and will continue in order to find another maximum. randomwalk will easily break out since it has a very high diversification but chances are that if it find a local maximum and take it as a solution rather than the optimal solution since it is a problem of probability IE what are the chances that we get the one of the opitmal solution(s) than a local optimum? what we know for sure is randomwalk will not get stuck but will continue to randomly wander. We can conclude that the randommax will be better suited for getting out of local optimums.

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:

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

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)**

$\neg(A \wedge B) \vee (\neg B \wedge C)$ 12 valid interpretations

$(\neg A \vee B) \Rightarrow C$ 10 valid interpretations

$(A \vee \neg B) \wedge (\neg B \Rightarrow \neg C) \wedge \neg(D \Rightarrow \neg A)$ 3 valid interpretations

2.2 N-Queens Problem (6 pts)

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

For the N queens problem we have five different global conditions; We can only have exactly one queen per row , exactly one queen per column, at most one queen per diagonal (black lines), at most one queen per diagonal (white), and lastly we must include the queens already placed. Considering the board to be NxN, for a row we obtain our first clause; $(C_{i,0} \vee ... \vee C_{i,N-1})$ this ensures that we have at least One queen for the row. Afterwards we need to ensure that we have at most one queen per column, thus we have $\wedge_{i=0}^{N-1} \wedge_{j=0}^{N-1} (C_{i,j} \Rightarrow \neg C_{i,j+1} \wedge ... \wedge C_{i,N-1})$. For the rows we obtain something very similar; $(C_{0,j} \vee ... \vee C_{N-1,j})$ this ensures that we have at least One queen for the column. To ensure that we have at most one queen per row, $\bigwedge_{j=0}^{N-1} \bigwedge_{i=0}^{N-1} (C_{i,j} \Rightarrow \neg C_{i+1,j} \wedge ... \wedge CN-1, j)$. For diagonals, we simply need to precise that we only need at most one queen and not "exactly one queen". For each diagonal k = i-j = -N+1 ... N-1 along the axis [0,0] to [N,N] if we separate k into two halves with the bottom half k = {-N-1,..,-1{ = i-j this gives us $\bigwedge_{k=0}^{N-2} \bigwedge_{i=k}^{N-2} \bigwedge_{j=i+1}^{N-1} (\neg C_{i,i-k} \vee \neg C_{j,j-k})$. For the second half k = {0,..,N-1{ = i-j this gives us $\bigwedge_{k=1}^{N-2} \bigwedge_{i=k}^{N-2} \bigwedge_{j=i+1}^{N-1} (\neg C_{i-k,i} \vee \neg C_{j-k,j})$. for diagonals in the other way, for k = i + j = 0,..,2N-2. Once again we separate into two halves, for first k = {0,..,n+1} we obtain $\bigwedge_{k=1}^{N-1} \bigwedge_{i=1}^{k-1} \bigwedge_{j=i+1}^{k} (\neg C_{i,k-i} \vee \neg C_{j,k-j})$. For the other half, k = {N+2,..,2N} $\bigwedge_{k=1}^{N-2} \bigwedge_{i=1}^{N-k-1} \bigwedge_{j=i+1}^{k} (\neg C_{i,N-i} \vee \neg C_{j,N-j})$ For the queens already placed we just need to add $\bigvee_{i=0}^{Nqueens} C_i$

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

For the rows, if we rewrite the two global clauses we obtain $\text{Clause}_1 = \bigwedge_{i=0}^{N-1} (C_{i,0} \vee ... \vee C_{i,N-1})$ and $\text{Clause}_2 = \bigwedge_{k=0}^{N-1} \bigwedge_{i=0}^{N-2} \bigwedge_{j=i+1}^{N-1} (C_{k,i} \vee Ck, j)$

For the columns we obtain $\text{Clause}_3 = \bigwedge_{i=0}^{N-1} (C_{0,i} \vee ... \vee C_{N-1,i})$ and $\text{Clause}_4 = \bigwedge_{k=0}^{N-1} \bigwedge_{i=0}^{N-2} \bigwedge_{j=i+1}^{N-1} (C_{i,k} \vee Cj, k)$

For diagonals, we obtain $\text{Clause}_5 = \bigwedge_{k=0}^{N-2} \bigwedge_{i=k}^{N-2} \bigwedge_{j=i+1}^{N-1} (\neg C_{i,i-k} \vee \neg C_{j,j-k})$. For the second half k = {0,..,N-1{ = i-j $\text{Clause}_6 = \bigwedge_{k=1}^{N-2} \bigwedge_{i=k}^{N-2} \bigwedge_{j=i+1}^{N-1} (\neg C_{i-k,i} \vee \neg C_{j-k,j})$. for diagonals in the other way, for k = i + j = 0,..,2N-2. For first k = {0,..,n+1} we obtain $\text{Clause}_7 \bigwedge_{k=1}^{N-1} \bigwedge_{i=1}^{k-1} \bigwedge_{j=i+1}^{k} (\neg C_{i,k-i} \vee \neg C_{j,k-j})$. For the other half, k = {N+2,..,2N} $\text{Clause}_8 = \bigwedge_{k=1}^{N-2} \bigwedge_{i=1}^{N-k-1} \bigwedge_{j=i+1}^{k} (\neg C_{i,N-i} \vee \neg C_{j,N-j})$ For the queens already placed we have $\text{Clause}_9 = \bigvee_{i=0}^{Nqueens} C_i$. With all the clauses defined our global cnf is $\bigwedge_{i=0}^{9} Clause_i$

3. Modify the function `get_expression(size)` in `queen_solver.py` such that it outputs a list of clauses modeling the n-queens problem for the given input. Submit your code on INGInious inside the *Assignment4: N-Queens Problem* task. The file `queen_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)**