Report LINGI1361: Assignment 4

Group N°012

Student1: Tristan Richard 15552000

Student2: /

May 8, 2023

1 The Atom Placement problem (13 pts)

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

problème : On a un graphe de V sites avec F liens entre ces sites, l'objectif est de placer les N atomes sur les sites de manière à minimiser l'énergie totale
fonction de cout: Somme des énergies d'interactions pour les F liens (on cherche à la minimiser)
solutions réalisables : Tous les atomes doivent être placés en gardant le nombre d'atome pour chaque type constant, un seul atome par site évidemment.
solutions optimal : la répartition des atomes qui minimise la cost fonction donc minimise l'énergie totale

2. You are given a template on Moodle: *atomplacement.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)`. **(2.5 pts)**

étant donné que l'on cherche à minimiser l'énergie totale, la fonction comporte une comparaison négative contrairement à un algo max value classique

(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)`. **(2.5 pts)**

meme commentaire que pour max value

3. Compare the 2 strategies implemented in the previous question and `randomwalk` defined in *search.py* on the given vertex cover instances. Run the strategies during 100 steps, and 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	0.311	151	5	0.312	141	15	0.022	324.1	55.7
i02	15.189	1281	17	14.683	1271	24.7	0.178	1968	40.6
i03	14.293	1128	22	14.476	1170	33.3	0.181	2111.8	37.1
i04	13.590	274	12	13.688	284	24.7	0.183	810.3	46.8
i05	0.346	304	4	0.35	305	11.6	0.022	477.4	41.9
i06	0.366	269	7	0.367	283	13.2	0.237	456.8	40.9
i07	1.789	244	14	1.798	249	22.9	0.055	564.4	54.8
i08	6.08	469	18	5.986	493	27.5	0.113	1047.2	47.3
i09	1.818	481	7	1.82	430	27.5	0.054	921.2	35.4
i10	5.783	2011	13	5.861	2001	23.9	0.111	2606.8	61.8

NS: Number of steps — **T**: Time — **Val**: Value of the best solution

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

   (a) In one sentence, what is the best strategy? **(0.5 pt)**

La meilleur stratégie est random max value

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

L'étendu des possibilités renvoyées par la fonction successor est très grande, ainsi un algo de mavalue pourrait rester bloquer sur un minimum local alors que la partie aléatoire de random value permet d'explorer plus de possibilité, c'est pour ca que la solution optimal est trouvé après un nombre de step plus élevé. Par contre random walk, a l'inverse est trop aléatoire et ne permet pas de converger vers une solution optimal.

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

max value : c'est la méthode la plus intense car elle prend la meilleur solution trouvé à chaque itération mais cette procédure reduit la diversification et peut entrainer des blocage sur un minimum local.
random max : cette méthode est le meilleur intermédiaire entre intensification et diversification car elle choisi un noeud aléatoire parmis les 5 meilleurs ce qui garentit une bonne diversifaction tout en restant performant.
random walk: la diversification est la plus élevée mais on ne trouve presque jamais la bonne solution.

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

max value : La méthode va rester bloquer autout de cette optimum local
random max : La méthode va sortir de cette optimum local pour en trouver un autre
random walk : La méthode ne va pas rester bloquer

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)**

1: 6 interprétations valides
2 : 5 interprétations valides
3 : 3 interprétations valides

2.2 Tapestry Problem (6 pts)

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

-Une seule forme par ligne: $(\sum_{j,b} C_{i,j,a,b} = 1) \wedge (\forall j, b, j' \neq j(\neg C_{i,j,a,b} \vee \neg C_{i,j',a,b}))$

-une seule forme par colonne: $\sum_{i,b} C_{ijab} = 1 \quad \wedge \quad \forall i, b, i' \neq i, (\neg C_{ijab} \vee \neg C_{i'jab})$

une couleur par ligne : $\sum_{j,a} C_{i,j,a,b} = 1 \quad \wedge \quad \forall j, a, j' \neq j, (\neg C_{i,j,a,b} \vee \neg C_{i,j',a',b})$

une couleur par colonne : $(\sum_{i,a} C_{ijab} = 1) \wedge (\forall i, a, i' \neq i(\neg C_{ijab} \vee \neg C_{ij'ab}))$

une seule combinaison de forme couleur : $(\neg C_{i,j,a,b} \vee \neg C_{i',j',b'})$

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

deja sous forme CNF

3. Modify the function `get_expression(size)` in `tapestry_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: Tapestry Problem* task. The file `tapestry_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 all the instances. **(2 pts)**