

Homework 2

Constrained Optimization and Local Search

Due: 6:00pm on 9/30/09

Problem 2.1

The $n \times m$ -queens problem is “played” on an $n \times m$ sized rectangular chess board. Given such a board, for arbitrary n and m , write an ILP to answer the following question: *what is the maximum number of queens that can be placed on the board such none is attacking any other?* Two sample configurations of queens on 6×9 boards appear in Figure 1—one legal and one illegal.

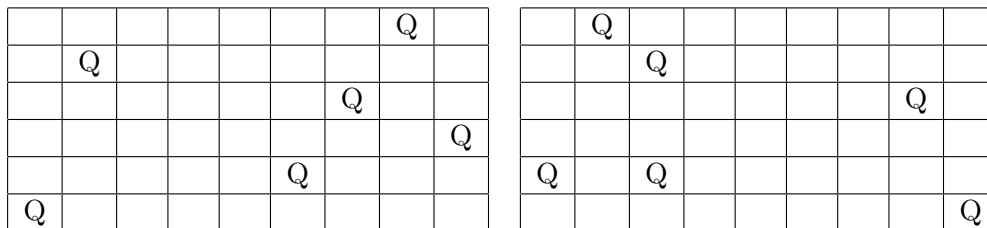


Figure 1: 6×9 -Queens. LHS: A valid configuration. RHS: An invalid configuration.

For those of you who are not familiar with the legal moves of chess pieces, a queen can move as many cells as she likes in any direction she likes—along a row, along a column, or along a diagonal. At the end of this writeup, there is a pointer to a resource with short descriptions of the legal moves in chess, which should provide all of the information about chess that is necessary to complete this assignment. Feel free to see a TA on hours if you have questions.

Hint: Here are some questions to think about while formulating these problems as integer programs:

- What function is being optimized (i.e. maximized or minimized)?
- What are the decision variables in this problem?
- What are the constraints on the decision variables?
e.g., Are they integer-valued or real-valued?
- How can the rules governing the movement of pieces be formulated as constraints?

Solution:

Care of Jimmy Kaplowitz

Let $q_{x,y}$ be a two-valued variable that is equal to 0 if there is no queen in the cell at row x and

column y of the $n \times m$ chess board and 1 if a queen is present there. x is an integer ranging from 1 to n , and y is an integer ranging from 1 to m . Then the goal is to maximise

$$\sum_{i=1}^n \sum_{j=1}^m q_{i,j}$$

under the following constraints:

$$\begin{aligned} \sum_{i=1}^n q_{i,j} &\leq 1 \quad \text{for all } j \in \{1, \dots, m\} \\ \sum_{j=1}^m q_{i,j} &\leq 1 \quad \text{for all } i \in \{1, \dots, n\} \\ \sum_{j=1}^{\min(n-i+1, m)} q_{(i+j-1), j} &\leq 1 \quad \text{for all } i \in \{1, \dots, n\} \\ \sum_{i=1}^{\min(n, m-j+1)} q_{i, (i+j-1)} &\leq 1 \quad \text{for all } j \in \{2, \dots, m\} \\ \sum_{j=1}^{\min(n-i+1, m)} q_{(i+j-1), (m-j-1)} &\leq 1 \quad \text{for all } i \in \{1, \dots, n\} \\ \sum_{i=1}^{\min(n, m-j+1)} q_{(n-i-1), (i+j-1)} &\leq 1 \quad \text{for all } j \in \{2, \dots, m\}. \end{aligned}$$

Extra Credit:

Write an ILP to answer the following question: *what is the maximum number of points that can be scored by placing queens, rooks, bishops, and knights on an $n \times m$ chess board such that none is attacking any other, where the pieces are valued at q, r, b , and k , respectively?* Typically, $q = 9$, $r = 5$, $b = 3$, and $k = 3$; however, your formulation should be general enough to handle arbitrary point values as inputs to the problem.

Problem 2.2

Given the following knapsack problem:

$$\begin{aligned} \max \quad & 8x_1 + 50x_2 + 49x_3 + 54x_4 \\ \text{s.t.} \quad & 8x_1 + 10x_2 + 10x_3 + 12x_4 \leq 30 \end{aligned}$$

Perform two complete Least Discrepancy Searches (also called Limited Discrepancy Searches), one for each of the conditions below. Before you begin, please see the additional handout on LDS available online.

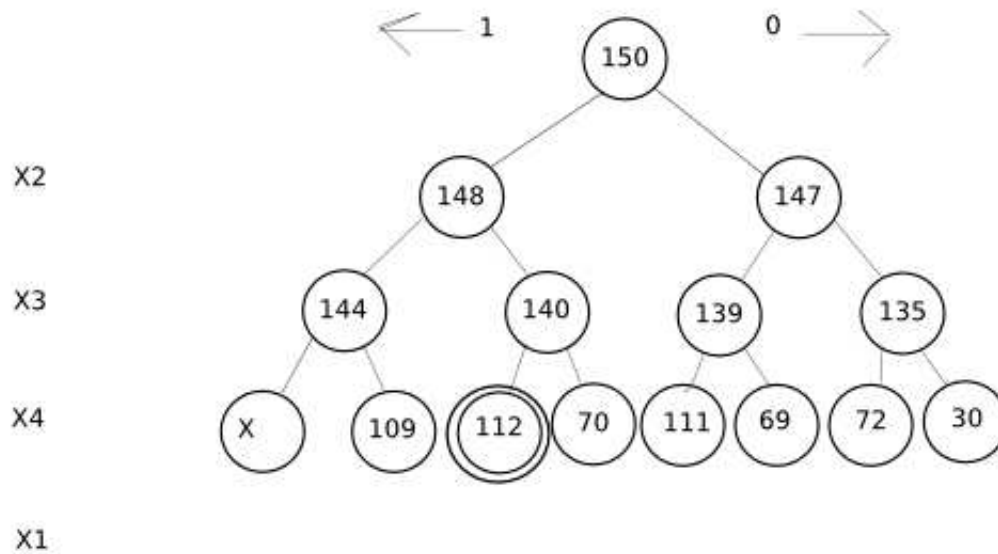
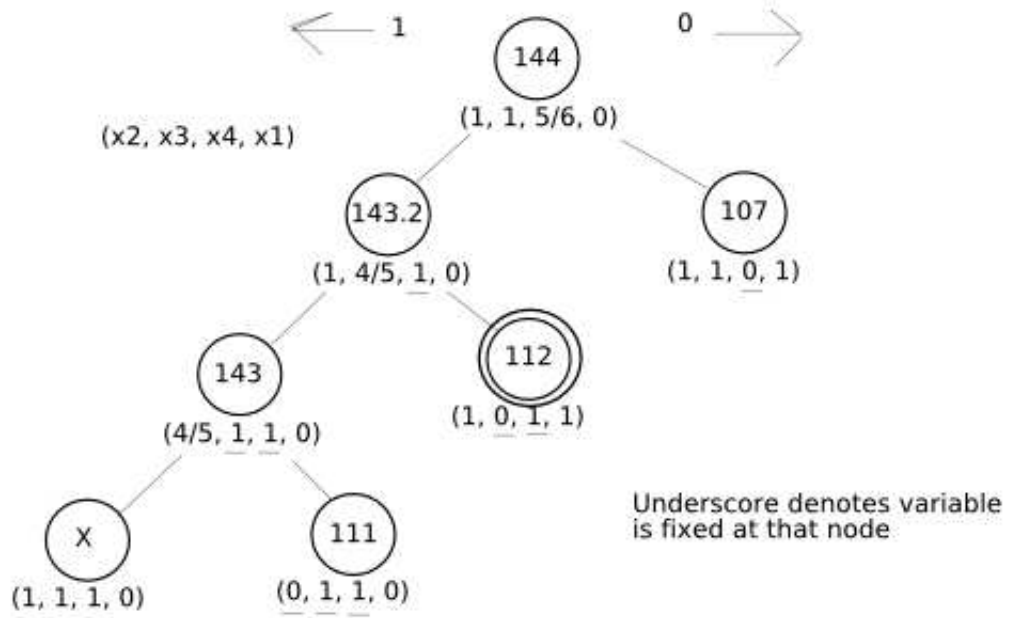
- a) The branching variable chosen is the fractional variable in the linear relaxation. As the heuristic, choose the child with the highest linear relaxation value. (Recall that the heuristic tells you which value to try first for a chosen branching variable.)
- b) The branching variable is the most efficient of the remaining variables that haven't been assigned a value, where efficiency is value divided by weight. Calculate the 'relaxation value' in a different way: find the most efficient unassigned item and calculate what the value would be if the knapsack were entirely filled with that item. For example, if items have efficiency (value / weight) $4/3$, $4/2$ and $1/2$ and the capacity is 5 then the relaxation value would be $5 \cdot 4/2 = 10$. (This will give you an upper bound on the possible value) As the heuristic, choose the child with the greatest relaxation value, as calculated in this way.

Your handin for each problem should consist of a tree where each node contains

- a. Which variables or fractions of variables are selected
- b. The total value of the current selection (where value is calculated as per the instructions for (a) and (b))

This can either be done by hand, or with a program like Dia, but be sure the end result is readable and clear. If your tree is getting unreasonably large, you are probably doing something wrong.

Solution:



Problem 2.3

There are many variations on the application of simulated annealing. The version discussed in class scanned all of the neighbors and always took an improving solution if one was found. Otherwise, the search assigned a probability to each neighbor corresponding to its decrement in performance. This can be formalized as follows:

Given a set X of states, an objective function $Obj(x)$ and a neighbors function $N(x)$: at each step, starting at state x , evaluate the objective function at all neighbors y_1, \dots, y_n . If any neighbor improves the value of the objective function, then move greedily to the best neighbor: set x equal to an $\arg \min_{y \in \{y_1, \dots, y_n\}} Obj(y)$. Otherwise, if no neighbor improves the value of the objective function, then stochastically move to some neighbor y with probability $p \sim e^{-\Delta(x,y)/T}$, for $T > 0$, where $\Delta(x, y) = Obj(y) - Obj(x)$. (The probabilities are normalized to sum to 1, so that the algorithm is guaranteed to accept some move.)

This approach however can be potentially very costly when working with large neighborhoods. Classic simulated annealing therefore evaluates each of the neighbors one at a time, always accepting an improving state, and otherwise making the transition with probability p . This algorithm can be formalized as follows:

SA(X, Obj, N, m, x, C)	
Inputs	local search problem number of iterations m random start state x cooling schedule C
Output	best state visited x^*
Initialize	$x^* = x, T$
for $i = 1$ to m	
a. for some $y \in N(x)$	
(a) compute $\Delta(x, y) = Obj(y) - Obj(x)$	
(b) if $\Delta(x, y) < 0$, then $x = y$ /* x is an improvement */	
(c) else $\text{rand}[0, 1] \leq e^{-\Delta(x,y)/T}$, then $x = y$	
(d) if $Obj(x) < Obj(x^*)$, $x^* = x$	
b. decay T according to schedule C	
return x^*	

Table 1: Classical Simulated Annealing.

For this problem, we ask you to compare these two algorithms.

- a. Describe a search space in which the in-class algorithm outperforms classic simulated

annealing. Use no more than five states.

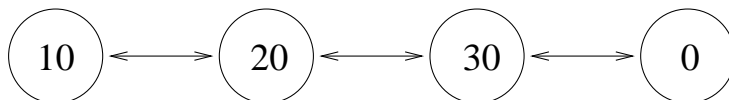
Solution:



This algorithm always finds heaven in the above search space, but simulated annealing might go to hell.

- b. Describe a search space in which classic simulated annealing outperforms the in-class algorithm. Use no more than five states.

Solution:



Starting at the state with the value 10, this algorithm forever oscillates between visiting states 10 and 20, and ultimately returns the local optimum 10. On the other hand, simulated annealing eventually finds the optimal state 0, since there is some positive probability of two consecutive exploratory moves.

- c. Describe the behavior of the in-class algorithm as $T \rightarrow \infty$ in terms of heuristics and/or related optimization algorithms.

Solution:

This algorithm mimics best-improvement search, if an improvement exists. If not, as $T \rightarrow \infty$, it behaves like a random walk.

- d. Describe the behavior of the in-class algorithm as $T \rightarrow 0$ in terms of heuristics and/or related optimization algorithms.

Solution:

As $T \rightarrow 0$, this algorithm behaves like the force-best-move heuristic.