Graphical Models Coursework 2

Razvan Valentin Marinescu
Student Number: 14060166
razvan.marinescu.14@ucl.ac.uk

Konstantinos Georgiadis
Student Number: 14110861
konstantinos.georgiadis.14@ucl.ac.uk

October 29, 2014

Contributions

The problem was solved slightly different by both of us (in an independent manner), but we got the same answer at the end. The only difference in our solutions was in how we looped over all the possible configurations of the battleships.

Battleships Problem

Without loss of generality, we can easily transform the given problem into the following equivalent problem. Find the pixel with the highest probability of containing a ship given the following constraints:

- ships cannot collide/overlap on the board
- one ship is oriented vertically, the other horizontally
- the ships cannot be placed on the blacklisted locations, given in the problem definition: [1, 10; 2, 2;3, 8; etc ..]

The first two requirements were already given, while the third one was incorporated given the extra information about the 10 unsuccessful misses. Assuming each pixel is a variable $x_{i,j}, i \in \{1, ..., 10\}, j \in \{1, ..., 10\}$ then if $x_{i,j} = 1$ it means that the pixel (i,j) is occupied by a battleship. If $x_{i,j} = 0$ then it is water. We want to calculate:

$$\arg\max_{i,j} \; p(x_{i,j} = 1 | x_{1,10} = 0, x_{2,2} = 0, ..., x_{9,9} = 0) =$$

$$\arg\max_{i,j} \; \frac{p(x_{i,j} = 1, x_{1,10} = 0, x_{2,2} = 0, ..., x_{9,9} = 0)}{p(x_{1,10} = 0, x_{2,2} = 0, ..., x_{9,9} = 0)}$$

However in the calculation of the above probabilities, when we will sum over the variables which are not considered known, we have two possibilities. Either the restrictions for the placement of the battleships is met and there is a uniform probability(say k), or the restrictions are not met and the probability is equal to zero. By summing over all the variables which are not known, we will basically have:

$$\arg\max_{i,j} \; \frac{p(x_{i,j} = 1, x_{1,10} = 0, x_{2,2} = 0, ..., x_{9,9} = 0)}{p(x_{1,10} = 0, x_{2,2} = 0, ..., x_{9,9} = 0)} =$$

$$\arg\max_{i,j} \; \frac{N_{i,j} * K}{N * K} =$$

$$\arg\max_{i,j} \; \frac{N_{i,j}}{N}$$

Where N is the number of all the acceptable battleship placements given the restrictions $x_{1,10} = 0, x_{2,2} = 0, ..., x_{9,9} = 0$ and $N_{i,j}$ is the number of all the acceptable battleship placements with the additional restriction that $x_{i,j} = 1$.
We did a brute-force approach by trying to go through every possible legal position of each ship on the map. Whenever we managed to find a valid configuration, we incremented a counter

for each of the occupied pixels. After all the possible configurations have been analysed and the counters of the respective pixels incremented, we normalise the counter matrix (divide each pixel by the sum of all the pixels in the matrix). However, there must be 10 pixels occupied on the board, so each pixel probability gets multiplied by 10. **The final result we get is that pixel at position (5,1) has the highest probability of being occupied, which is 0.2069**

For finding all the possible ways to position the boats on the board we do the following:

- for each legal position of the first boat
  - for each legal position of the second boat
    - make sure boats don't collide
    - increment counters on the grid of pixels.

The MATLAB code that implemented this brute-force approach is given below, with appropriate comments.

```
function prob120()

len = 10;
grid = zeros(len, len); % in each cell (i,j) contains how many possible scenarios exist where it is occupied by any of the boats

boatLen = 5;

illegal_locations = [1, 10; 2, 2;3, 8; 4, 4; 5, 6; 6, 5; 7, 4; 7, 7; 9, 2; 9, 9]';

% a few tests
testPoints = [ones(boatLen, 1) * 1, ones(boatLen, 1) * 6 + (0:boatLen-1)']';
assert(boat_illegal_location(testPoints, illegal_locations ) == 1);

testPoints = [ones(boatLen, 1) * 1, ones(boatLen, 1) * 5 + (0:boatLen-1)']';
assert(boat_illegal_location(testPoints, illegal_locations ) == 0);


% horizontal boat at position (iH,jH) occupies cells (iH, jH), (iH, jH+1), (iH, jH+2), (iH, jH+3) ...
% vertical boat at position (iV,jV) occupies cells (iV,jV), (iV+1,jV), (iV+2,jV) ...

% fix location of the horizontal ship
for iH=1:len
    for jH=1:(len-boatLen+1)
        % find cells the boat is occupying
        hBoatPoints = [ones(boatLen, 1) * iH, ones(boatLen, 1) * jH + (0:boatLen-1)']';

        % make sure boat is not on illegal location
        if (boat_illegal_location(hBoatPoints, illegal_locations))
            continue;
        end

        % fix location of the vertical ship
        for iV=1:(len-boatLen+1)
            for jV=1:len
                % find cells the boat is occupying
                vBoatPoints = [ones(boatLen, 1) * iV  + (0:boatLen-1)', ones(boatLen, 1) * jV]';

                % make sure boat is not on illegal location
```

```
                if (boat_illegal_location(vBoatPoints, illegal_locations))
                    continue;
                end

                % make sure boats don't collide
                if (boat_illegal_location(vBoatPoints, hBoatPoints))
                    %vBoatPoints
                    %hBoatPoints
                    continue;
                end

                % update grid counters
                %[hBoatPoints, vBoatPoints]
                grid = update_grid(grid, [vBoatPoints, hBoatPoints]);
            end
        end
    end
end

% normalise the grid
grid = grid ./ sum(grid(:));

% multiply everything by 10 because there are 10 pixels that are actually
% occupied on the board (5 pixels for each ship)
grid = grid .* 10;

grid

max = 0;
maxI = 1;
maxJ = 1;

% find the highest probability in the grid
for i=1:len
    for j=1:len
        if(grid(i,j) > max)
            maxI = i;
            maxJ = j;
            max = grid(i,j);
        end
    end
end

max
[maxI, maxJ]

end

function is_on_illegal_loc = boat_illegal_location(boatPoints, illegal_locations)

is_on_illegal_loc = 0;

for point=boatPoints
    for loc=illegal_locations
        if (any(point - loc) == 0)
            is_on_illegal_loc = 1;
            return;
        end
    end

end

end
```

```
105 function grid = update_grid(grid, boatPoints)

107 for p=boatPoints
        grid(p(1),p(2)) = grid(p(1),p(2)) + 1;
109 end

111 end
```