CS440 Fall 2015

Assignment 2: Constraint Satisfaction Problems and Games

Deadline: October 26, 11:59:59PM

As on Assignment 1, you have the option of working in groups of up to three people. You are free to either stay with the same team or pick a new one, but as before, three-unit students must work with three-unit students and four-unit students must work with four-unit students. For more detail, see **submission instructions**.

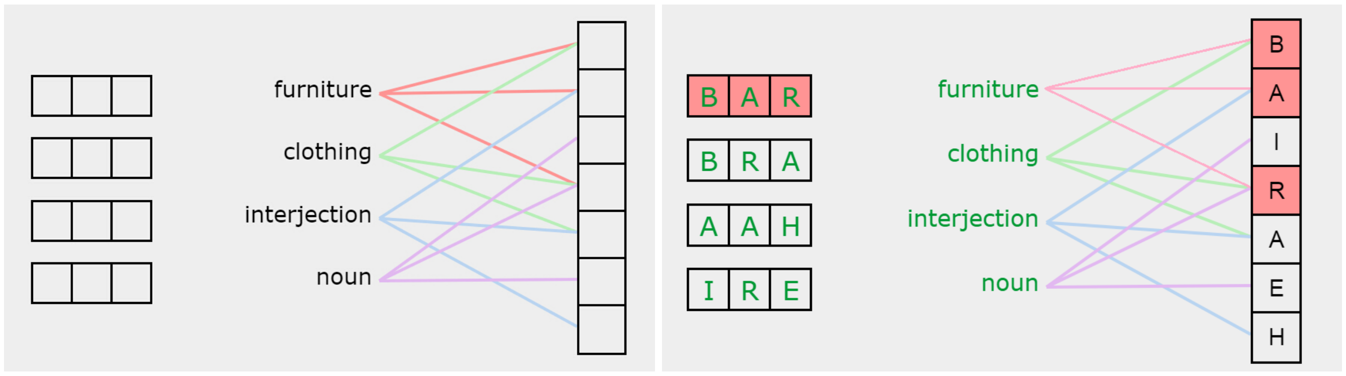
Contents

Part 1: Constraint Satisfaction Problems

1.1 Word Puzzles (for everybody)

Adapted by Qieyun Dai from Dana Moshkovitz and Pasin Manurangsi

In this problem, you are asked to solve word puzzles using backtracking search. In these puzzles, you have to fill in an array with letters such that certain subsets of the letters form words from a given category. An example of a word puzzle with one possible solution is given below.



In this example, the letters connected to red lines should form a word from the "furniture" category and similarly, the letters connected to the green lines should form a word belonging to the "clothing" category. All the words have a length of 3 and the list of candidate words for each category is given in this word list (you can also find a zip file here where each txt file corresponds to one category, and the words for this category are listed one per line).

Your goal is to implement backtracking search to find **all solutions** to the following instances (each puzzle may have multiple solutions):

puzzle 1

puzzle 2

puzzle 3

puzzle 4

puzzle 5

In each input file, the first line specifies the size of the result array, and the rest of the file lists the category names and indices of the result array that correspond to the word of that category.

You should implement two verisons of backtracking search for two different formulations:

(a) **letter-based assignment** where you fill in the array one letter at a time;

(b) **word-based assignment** where you fill in the array one word at a time.

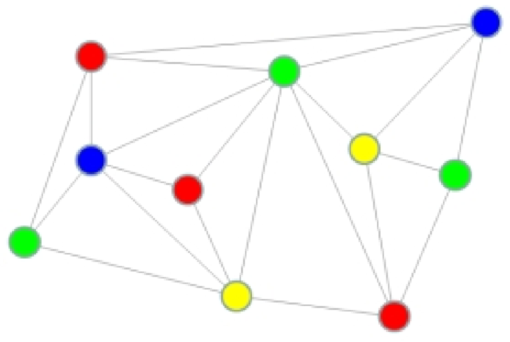
In your report, please include the following:

(1) For each formulation, specify the **variables**, **domains** and **constraints**. Briefly discuss what kind of constraint checking/inference you have implemented (if any) to make the search more efficient.

(2) For each example, list **all possible solutions** (there may be more than one).

(3) For each example and each formulation, show your search trace. (Note that if you do a consistency check before each attempted assignment, the search space shouldn't be too large.) Here is an example letter-based trace and word-based trace. In the first line of the trace file, indicate your assignment order (you're free to choose whatever order you like). List one search path per line. The path is terminated either when a solution is found (please list the found solution) or no valid assignment exists and the search needs to backtrack.

1.2 Map coloring (for four-unit students)



(Based on question 6.10 in 3rd edition of the textbook.)

Generate random instances of map-coloring problems as follows: scatter N points on the unit square; select a point X at random, connect X by a line segment to the nearest point Y such that X is not already connected to Y and the segment crosses no other segment (see, e.g., here how to test for segment-segment intersection); repeat the previous step until no more connections are possible. The points represent regions on the map and the lines connect neighbors. Now try to color each map with four colors using backtracking search (a) without forward checking, using random variable and value assignment order; and (b) with forward checking and any other variable and value ordering heuristics that can increase search efficiency. For (b), briefly discuss what you implemented.

For each N, generate several random problem instances, and try to make N as large as you can manage. On average, how many constraints (edges) do your map coloring instances have for each N? For each variant of backtracking search, report the average number of variable assignments attempted as a function of N and average running time as a function of N (plots are very welcome here).

Tips

- For segment-segment intersection code, see, e.g. here.
- Feel free to generate random map coloring problems using an alternative method, for example, using a Delaunay triangulation of a set of random points.
- For bonus points, attempt to solve this problem by local search.

Part 2: War Game

Adapted from www.cool-ai.com with help from Michael Sittig, Codruta Girlea, and Jason Cho

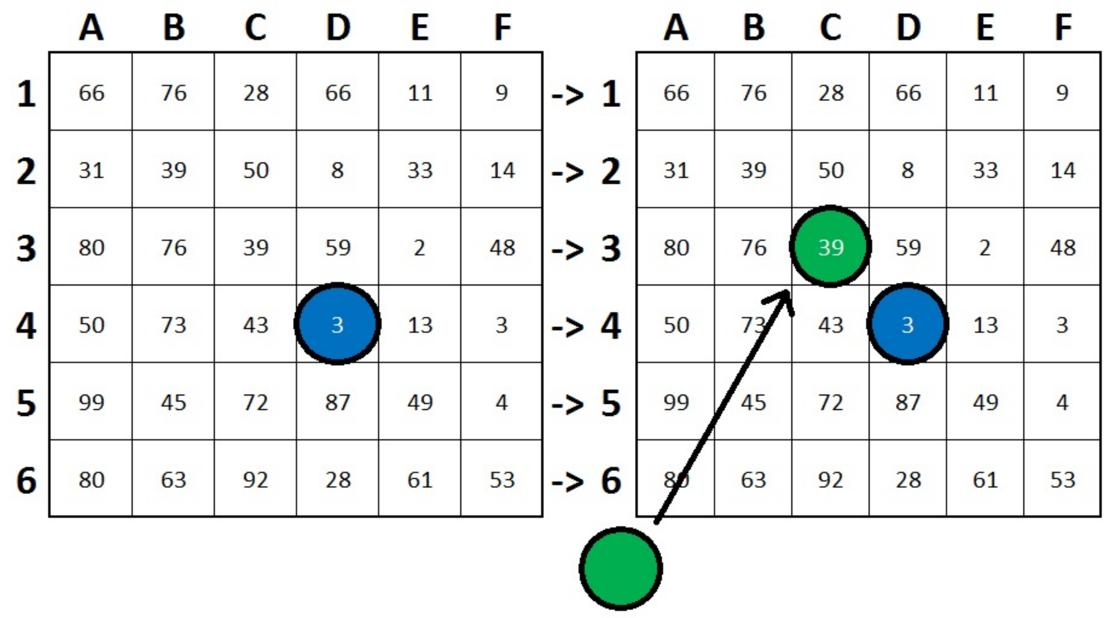
The goal of this part of the assignment is to implement an agent to play a simple "warfare" game.

Rules of the game

- The game board is a 6x6 grid representing a city.
- Each square has a fixed point value between 1 and 99.
- There are two players, "blue" and "green". Each player takes turns: blue moves first, then green, then blue, etc.
- The object of the game is to be the player in the end with the largest total value of squares in their possession. That is, one wants to capture the squares worth the most points.
- The game ends when all the squares are occupied by all players since no more moves are left.
- Movement is always vertical and horizontal but never diagonal.
- Pieces can be conquered in the vertical and horizontal direction, but never the diagonal direction.
- The values of the squares can be changed for each game, but remain constant within a game.
- In each turn, a player can make one of two moves:

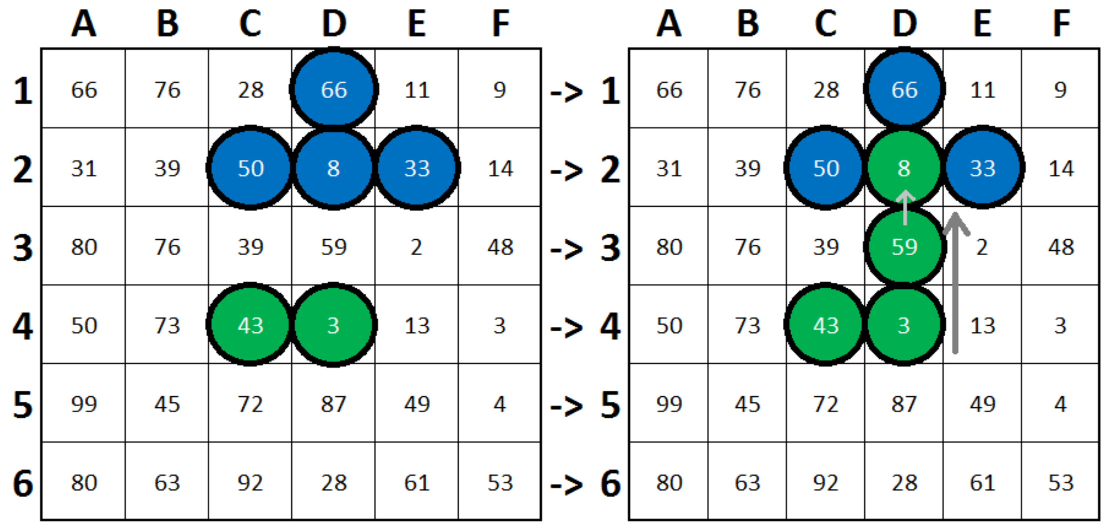
  Commando Para Drop: You can take any open space on the board with a Para Drop. This will create a new piece on the board. This move can be made as many times as one wants to during the game, but only once per turn. A Commando Para Drop cannot conquer any pieces. It simply allows one to arbitrarily place a piece on any unoccupied square on the board. Once you have done a Para Drop, your turn is complete.

  The image below illustrates a Commando Para Drop. In this case, green drops a new piece on square [C,3]. This square is worth 39, which is a higher number, meaning that it contains some juicy oil wells or other important resources. After that, the score is green 39 : blue 3. A Commando Para Drop could have been carried out on any squares except for [D,4] since blue already occupies it.

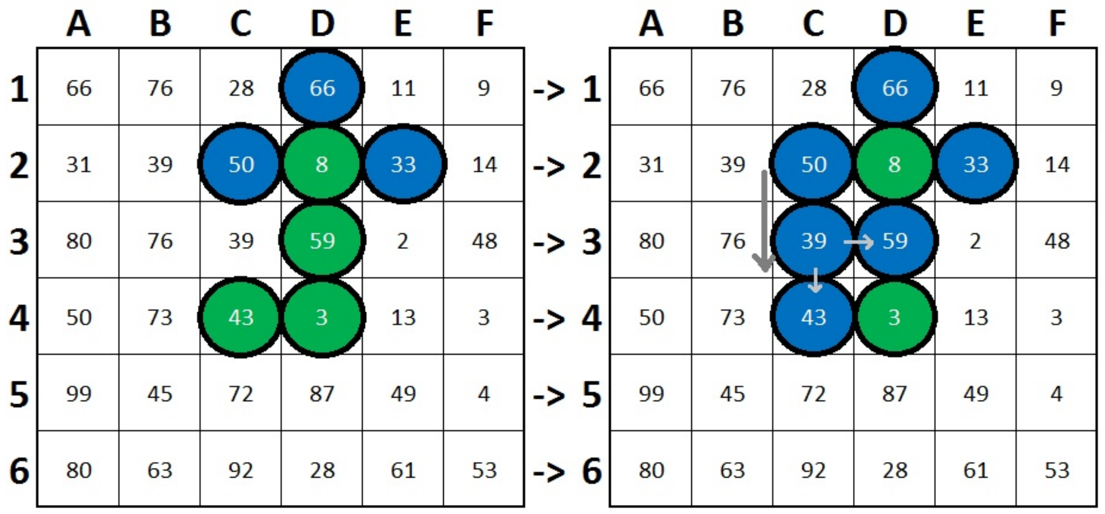


M1 Death Blitz: From any space you occupy on the board, you can take the one next to it (up, down, left, right, but not diagonally) if it is unoccupied. The space you originally held is still occupied. Thus, you get to create a new piece in the blitzed square. Any enemy touching the square you have taken is conquered and that square is turned to your side (you turn its piece to your side). An M1 Death Blitz can be done even if it will not conquer another piece. Once you have made this move, your turn is over.

The image below illustrates an M1 Death Blitz. Green blitzes the piece in [D,4] to [D,3]. This conquers the blue piece in [D,2] since it is touching the new green piece in [D,3]. A blitz always creates a new piece and always moves one square, but it does not conquer another piece unless it is touching it. Thus, another valid move might have been for [D,4] to have blitzed [E,4]. Then the green player would own [D,4] and [E,4] but would have conquered none of blue's pieces. Note, the score before the blitz was green 46 : blue 157 but afterwards is green 113 : blue 149.



Here is another illustration:



Here blue blitzes [C,3] from [C,2]. In the process green's pieces at [D,3] and [C,4] are conquered since they touch [C,3]. Notice that in its next move, green will not be able to conquer any of blue's pieces and only the piece at [D,4] would be able to execute an M1 Death Blitz since [D,2] has no neighboring unoccupied squares.

You can assume that blitzing is mandatory: i.e., if you put a piece adjacent to pieces you already own and you have the opportunity to "convert" neighboring enemy pieces, you have to take it.

2.1 Minimax and alpha-beta agents (for everybody)

Your task is to implement agents to play the above game, one using **minimax search** and one using **alpha-beta search**. Your program should use depth-limited search with an evaluation function -- which you, of course, need to design yourself and explain in the report. Try to determine the maximum depth to which it is feasible for you to do the search (for alpha-beta pruning, this depth should be larger than for minimax). The worst-case number of leaf nodes for a tree with a depth of three in this game is roughly 42,840. Thus, you should at least be able to do minimax search to a depth of three.

For each of these five game boards, run the following matchups:

A. Minimax vs. minimax;
B. Alpha-beta vs. alpha-beta;
C. Minimax vs. alpha-beta (minimax goes first);
D. Alpha-beta vs. minimax (alpha-beta goes first).

For each matchup, report the following:

1. The final state of the board (who owns each square) and the total scores for each player;
2. The total number of game tree nodes expanded by each player in the course of the game;
3. The average number of nodes expanded per move and the average amount of time to make a move.

Tips

- Pseudocode for alpha-beta pruning is given in Figure 5.7, p. 170, in the 3rd edition.
- For alpha-beta pruning, try to come up with a move ordering to increase the amount of pruning. Discuss any interesting choices in your report.

For bonus points

- Design an interface for the game that would allow you to play against the computer. How well do you do compared to the AI? Does it depend on the depth of search, evaluation function, etc.?
- Design your own game boards and show results on them. Try to play the game on a larger board. How large can you go?
- Implement any advanced techniques from class lectures or your own reading to try to improve efficiency or quality of gameplay.

- Implement an agent for a version of the game where you have to flip a coin in order to figure out whether adjacent enemy squares can be conquered by a M1 Death Blitz move.

2.2 Extended rules (for four-unit students)

Assume each number on the board also represents a number of resources (or a resource drain, in the case of negative numbers). Also assume that the strength of each unit (each piece) is directly proportional to the number of resources its owner controls (i.e., that player's current score) and inversely proportional to the number of units that player has on the board at the current time.

Change the M1 Death Blitz move so that it takes into account the opposing player strengths as follows:

- **Battle:** Enemies touching squares that were taken as part of the move are not automatically turned to your side. Instead, a battle takes place for the square occupied by the enemy unit: all your units adjacent to the square plus the (new) unit that is attempting to take the square participate in the battle on your side, each with its unit strength; all the enemy units adjacent to the square plus the unit currently on the square oppose you in the battle, each with its own unit strength. The player with the highest cumulative strength wins the square.

  For example, in the first illustration of the M1 Death Blitz above, right before the battle, green has 3 pieces and controls 105 resources, so its unit strength is 35. Blue controls 4 pieces and 157 resources, so its unit strength is 39.25. The battle is between 2 green pieces and 4 blue pieces, so the opposing forces are 70 and 157, respectively, so in this case, blue keeps the square.

- **Duel:** As before, but only the two opposing units fight on the square. In the first illustration, this means the battle is opposing forces 35 (green) and 39.25 (blue).

- **Attrition:** Consider that the value of the resources each player controls decreases by a fraction at every game round. This only affects the unit strength, not the player score.

Change your implementation to reflect each of the new rules and report the same items as in Part 1 above **for alpha-beta vs. alpha-beta matchups on three maps you choose**. Try to come up with a good evaluation function for either of the extended rule sets.

Compare the optimal strategies and player scores for all the rule sets. In which cases do the units tend to cluster together? In which cases do the units tend to tile, or scatter on the board?

Report Checklist

Your report should briefly describe your implemented solution and fully answer the questions for every part of the assignment. Your description should focus on the most "interesting" aspects of your solution, i.e., any non-obvious implementation choices and parameter settings, and what you have found to be especially important for getting good performance. Feel free to include pseudocode or figures if they are needed to clarify your approach. Your report should be self-contained and it should (ideally) make it possible for us to understand your solution without having to run your source code. For full credit, your report should include the following.

Part 1:

1. For everybody: (1) Briefly discuss your implementation for search formulations (a) and (b). (2) List all possible solutions for each of five examples. (3) For each formulation and each example, give your search trace.
2. For four-unit students: For a range of map sizes (N), report average number of edges vs. N; average number of assignments and running time for search variants (a) and (b) (w/o forward checking and with forward checking and other heuristics). For variant (b), briefly discuss what heuristics you used.

Part 2:

1. For everybody: For matchups A-D, report items 1-3.
2. For four-unit students: For three types of extended rules, report items 1-3 as in Part 1 for alpha-beta vs. alpha-beta only on three boards of your choice. Discuss your evaluation function and game strategies that emerge.

Extra credit:

- We reserve the right to give **bonus points** for any advanced exploration or especially challenging or creative solutions that you implement. Three-unit students always get extra credit for submitting solutions to four-unit problems. **If you submit any work for bonus points, be sure it is clearly indicated in your report.**

Statement of individual contribution:

- All group reports need to include a brief summary of which group member was responsible for which parts of the solution and submitted material. We reserve the right to contact group members individually to verify this information.

WARNING: You will not get credit for any solutions that you have obtained, but not included in your report! For example, if your code prints out path cost and number of nodes expanded on each input, but you do not put down the actual numbers in your report, or if you include pictures/files of your output solutions in the zip file but not in your PDF. The only exception is animated paths (videos or animated gifs).

Submission Instructions

As for Assignment 1, **one designated person from the group** will need to submit on **Compass 2g** by the deadline. Three-unit students must upload under **Assignment 2 (three units)** and four-unit students must upload under **Assignment 2 (four units)**. Each submission must consist of the following two attachments:

1. A **report** in **PDF format**. As for Assignment 1, the report should briefly describe your implemented solution and fully answer all the questions posed above. **Remember: you will not get credit for any solutions you have obtained, but not included in the report.**

   As before, all group reports need to include a brief **statement of individual contribution**, i.e., which group member was responsible for which parts of the solution and submitted material.

   The name of the report file should be **lastname_firstname_assignment2.pdf**. Don't forget to include the names of all group members and the number of credit units at the top of the report.

2. Your **source code** compressed to a **single ZIP file**. The code should be well commented, and it should be easy to see the correspondence between what's in the code and what's in the report. You don't need to include executables or various supporting files (e.g., utility libraries) whose content is irrelevant to the assignment. If we find it necessary to run your code in order to evaluate your solution, we will get in touch with you.

   The name of the code archive should be **lastname_firstname_assignment2.zip**.

Multiple attempts will be allowed but only your last submission before the deadline will be graded. **We reserve the right to take off points for not following directions.**

Late policy: For every day that your assignment is late, your score gets multiplied by 0.75. The penalty gets saturated after four days, that is, you can still get up to about 32% of the original points by turning in the assignment at all. If you have a compelling reason for not being able to submit the assignment on time and would like to make a special arrangement, you must send me email **at least a week before the due date** (any genuine emergency situations will be handled on an individual basis).

Be sure to also refer to course policies on academic integrity, etc.