Project 2.2 (CSP)
Additional
Helpful Notes & Hints

Overview

1. Compared to P2.1, where the constraints are arbitrary functions that we have no knowledge about. Here we have more concrete domain knowledge and hence it allows for a more efficient formulation of the various heuristics.

2. We will be discussing ways to improve from the following directions:
    a. Problem formulation
    b. Variable ordering heuristic
    c. Value ordering heuristic
    d. Forward checking

Disclaimer

Besides the necessary assumptions stated in the instructions, everything else suggested in the subsequent slides is just one way among many to optimise. It is not meant to be the only way, I believe you can do even better

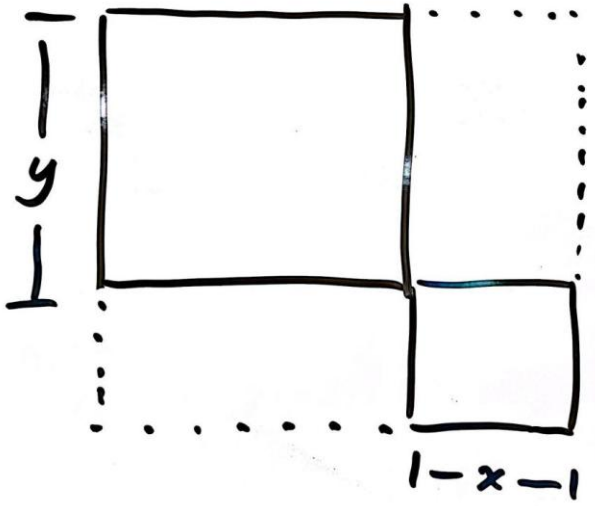
Some hints are only needed for bonus test cases. You need not worry about those otherwise.

Problem formulation

- How to initialise our board formulation? What is the relevant information?
- **Constraints**: Boundary and obstacles (squares with unit side length).
- **Domain**: All the remaining valid coordinates inside the board. (such that it **does not cause any overlapping** with the currently placed squares, obstacles and does not exceed the boundary).
- **Variables**: Squares of varying side length.
- But what else?
- To check whether a specific coordinate has been covered by some squares, given what we have, we must iterate through the domains of each variable to determine so in the worst case, can we do better by designing some global variables? (Maybe a binary representation of the board?)
- What else can we have? (To be discussed in forward checking part.)

A remark on conditions of overlapping

Only prune away those coordinates inside the obstacles is not enough. Consider the diagram below, where a square of side length x has already been placed, and a square of side length y is about to be placed. Under what condition will overlapping occur?

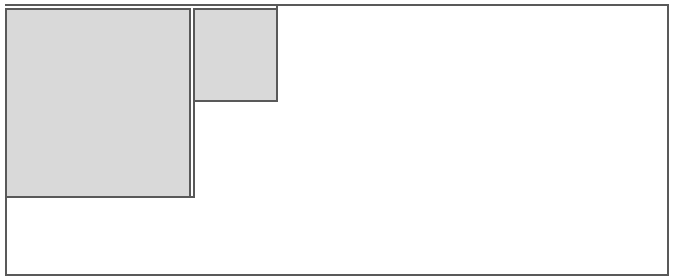


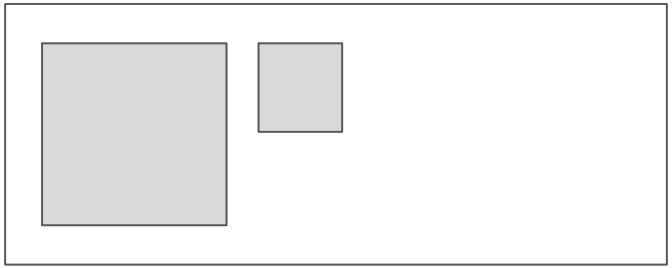
Variable ordering heuristic

- By default, it checks the cardinality of the domain of each variable, and it picks the one with the smallest domain (MRV).
- It takes O(n) time by default! But can we do better?
- Observation: Squares with larger size will occupy the largest area, and hence?
- Can we achieve O(1) time in selecting our variables to be assigned, at the compromise of our initialisation step? And how to adapt the unassign() function accordingly? (Consider making use of sort() in pre-processing.)

Value-ordering heuristic

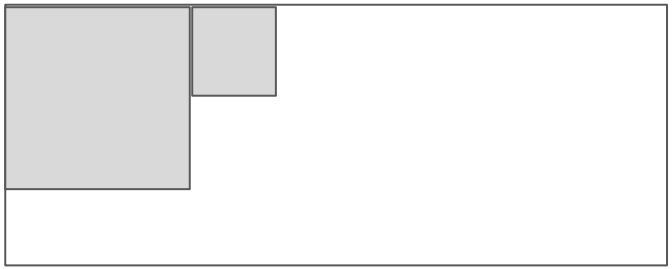
- Which coordinate should we pick first?
- Consider the following scenario, where the grey regions are squares that have been placed onto the board. For the remaining empty space on the board, which side yields a more desirable shape? Why so? (Note that it is given in the question prompt that the whole board must be filled eventually.)

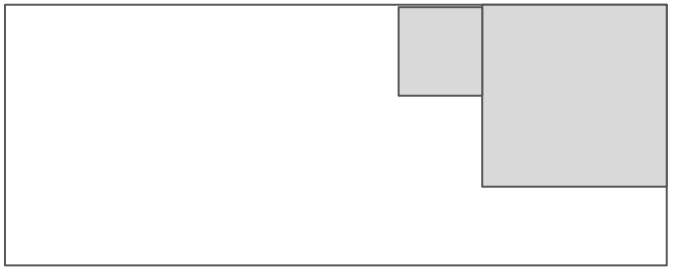




Value ordering heuristic continue

- Are LHS and RHS equivalent?
- So, combining the observation with the previous slide, how to **sort** our domain in a way to align with these observations to implement our value ordering heuristic?





Forward checking

- By default, forward checking checks whether the assignment of a specific variable will lead to some empty domain.
- But do we have to wait until the domain to become empty to conclude that certain direction is undesirable?
    1. If the cardinality of the domain of a specific variable (a square of side length k) is already less than the number of squares of side length k left, what does that imply?
    2. (Might only needed for one of the bonus cases, you need not worry about this otherwise.) So far, note that our variable ordering heuristic and value ordering heuristic have been deterministic. Hence, if we have previously determined that certain partially completed board will not work, (and since certain configuration can be reached by multiple ways of assignment), do we have to check it again? (This feature can be achieved by a reached set() in our initialisation, **however an inefficient formulation of the elements of this set** might slow down your code instead. Some data compression related functions from numpy library can be helpful.)

Forward Checking

3. (Might only needed for one of the bonus cases, you need not worry about this otherwise.) Consider a rectangle of length k and width 1, how many ways can we fill it up if we are only allowed to use squares? Hence, given a partially filled board, which subset of coordinates can be filled uniquely? How to detect those?

All the best!