Final Project Seminar

4/6/2021, CS 3510

Itinerary

Table of Contents

Project Overview

Project Introduction

- We will be designing algorithms for solving Minesweeper puzzles.
    - Board with squares
    - Some squares have mines
    - Other squares show the number of nearby mines
    - Don’t click on the mines

- Naively guessing solutions is an exponential-time search

Only 2000's kids will remember this!

Our Algorithm Specifications

- Input:
    - size: <m, n>
    - bombs: <b>
    - safe: <x,y> (Guaranteed safe start)
    - grid: <string of length m*n with domain [0-9]>
        - [0 - 8] is the number of nearby bombs
        - [9] is a bomb
        - Tile[x][y] = grid[y * n + x]

- Output:
    - Bitstring of length m * n
        - 0 represents no bomb
        - 1 represents bomb
    - Same access rules as input string
- Testing: diff(input_grid, output)
    - Same => 
    - Different => 

Example Input

size: 10,10

bombs: 20

safe: 5,5

grid:
9931192929399212322113920192100111012910000000112100000011291211001921929101221024432392001999299200

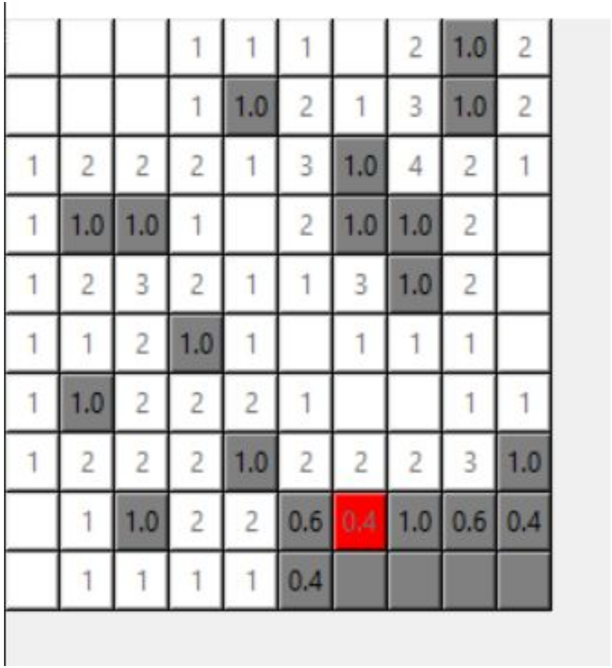
1.0	1.0	3	1	1	1.0	2	1.0	2	1.0
3	1.0	1.0	2	1	2	3	2	2	1
1	3	1.0	2		1	1.0	2	1	
	1	1	1		1	2	1.0	1	
						1	1	2	1
						1	1	2	1.0
1	2	1	1			1	1.0	2	1
1.0	2	1.0	1		1	2	2	1	
2	4	4	3	2	3	1.0	2		
1	1.0	1.0	1.0	2	1.0	1.0	2		

size: 10,10

bombs: 20

safe: 5,5

grid:
00011102920001921392122213942119910299201232
11392011291011101922210011122292223901922399
590111193399



Deliverables

- 2 implemented algorithms
- Accompanying project report
  - For each algorithm:
    - Algorithm explanation and analysis
    - Performance plots
- Details in project description

Team Formation

- Make a post on Piazza
- Pinned team forming feature

Constraint Satisfaction Problems

What is a CSP

A set of variable objects and constraints

- Triple ⟨X, D, C⟩
  - X: variables
  - D: domain of values for each X
  - C: constraints

- Example: Crossword Puzzle
  - X: Words
  - D: [A-Z]+, Words
  - C: $\text{Word}_1[i] = \text{Word}_2[j]$,
    $\text{len}(\text{Word}_1) = 7$
    ... and so on

Solving a CSP

CSPs are usually approached as search problems

Backtracking

Constraint Propagation

Local Search

Backtracking

- “Depth-first search” in the problem domain

```
prefix = ""
def backtrack(proposed_suffix):
    potential_solution = prefix + proposed_suffix

    if is_answer(potential_solution):
        submit_answer(potential_solution)
    if breaks_constraint(potential_solution):
        return

    while s := next_possible_suffix(potential_solution):
        backtrack(s)

backtrack("")
```

Constraint Propagation

Modify the constraints into an equivalent simpler set of constraints.

- Can prove solvability
  - {A + B = 1, A+ B = 2} => Impossible
- Can simplify constraints
  - {A + B = 1, A + B + C = 3} => {C = 2, A + B = 1}

Local Optimization

- Propose a (probably incorrect) solution, then try to change the proposed solution incrementally until it's as correct as possible

```
algorithm MIN-CONFLICTS is
    input: csp, A constraint satisfaction problem.
           max_steps, The number of steps allowed before giving up.
           current_state, An initial assignment of values for the variables in the csp.
    output: A solution set of values for the variable or failure.

    for i ← 1 to max_steps do
        if current_state is a solution of csp then
            return current_state
        set var ← a randomly chosen variable from the set of conflicted variables CONFLICTED[csp]
        set value ← the value v for var that minimizes CONFLICTS(var,v,current_state,csp)
        set var ← value in current_state

    return failure
```

Q&A