UNIVERSITÉ TOULOUSE 1 CAPITOLE

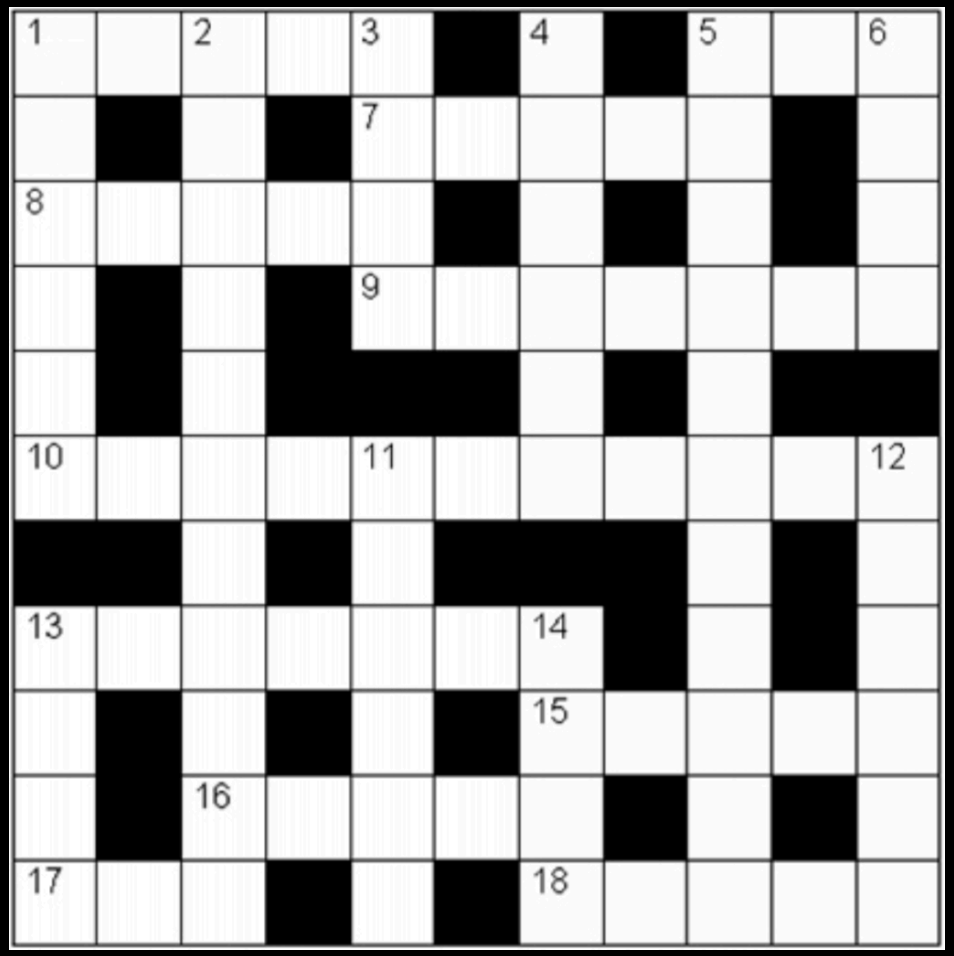
Computational Intelligence
Project 2: Reasoning

UT1C - MIAGE 2IS - Innovative Information Systems
Professors: Umberto GRANDI, Dennis WILSON
Student: Mahmoud AL NAJAR

Outline

- General Features of the grid
- Problem Modelling
- The Algorithms & Optimisation
  - CSP - Forward Checking
  - A*
- Algorithm Comparison

General Features



- Grid: 11 * 11
- Empty cells: 87
- Words: 21
- Intersections: 36

UNIVERSITÉ TOULOUSE 1 CAPITOLE

Problem Modelling

Modelling: Search

	Word-based	Letter-based	Intersection-based
STATE	Distribution of words	Distribution of letters in cells	Distribution of letters in intersections
ACTION	Adding a valid word to the grid	Adding a valid letter to the grid	Adding a valid letter to an empty intersection
GOAL	21 words filled	87 cells filled	36 intersections filled
STATE SPACE	267,755 ^ 21	26 ^ 87	26 ^ 36
BRANCHING FACTOR (b)	number of dictionary words * number of word variables = 267,755 * 21	len(alphabet) * number of cells = 26 * 87	len(alphabet) * number of intersections = 26 * 36
Depth (d)	21	87	36

Modelling: CSP

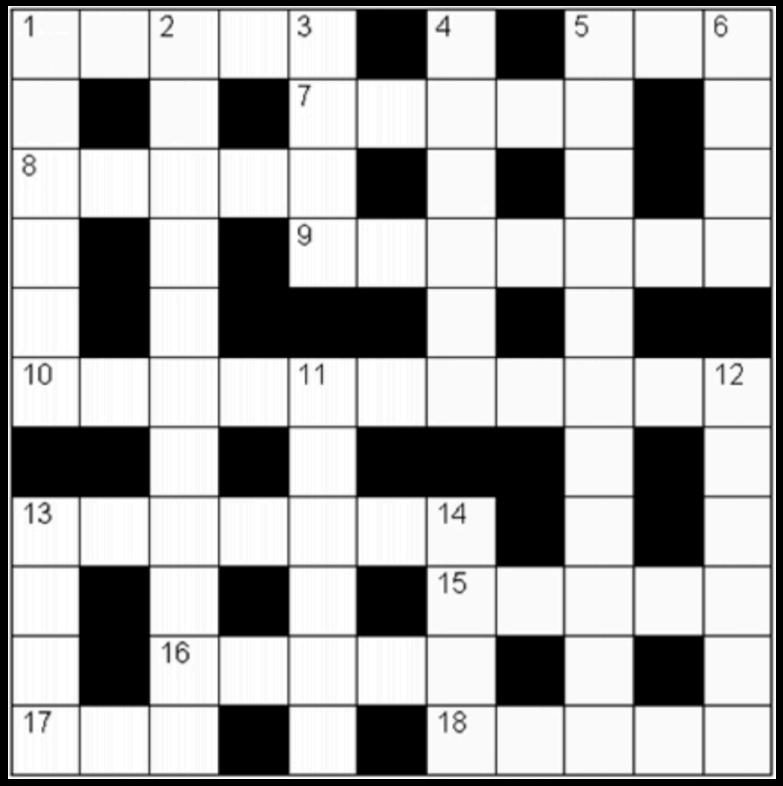
	Word-based	Letter-based
VARIABLES	21 variables	87 variables
DOMAINS	all dictionary words ~ 267,755 values	All English letters 26 values
CONSTRAINTS	• Lengths: ex. len(W1) = 5 • Intersections: X(w1,w2) ==> w1[i1] = w2[i2] • All words must be different	• Resulting words must exist in the dictionary: (L1 + L2 + Ln) in (dictionary) • All resulting words must be different

UNIVERSITÉ TOULOUSE 1 CAPITOLE

The Algorithms & Optimisation

CSP - Forward Checking

CSP - Forward Checking

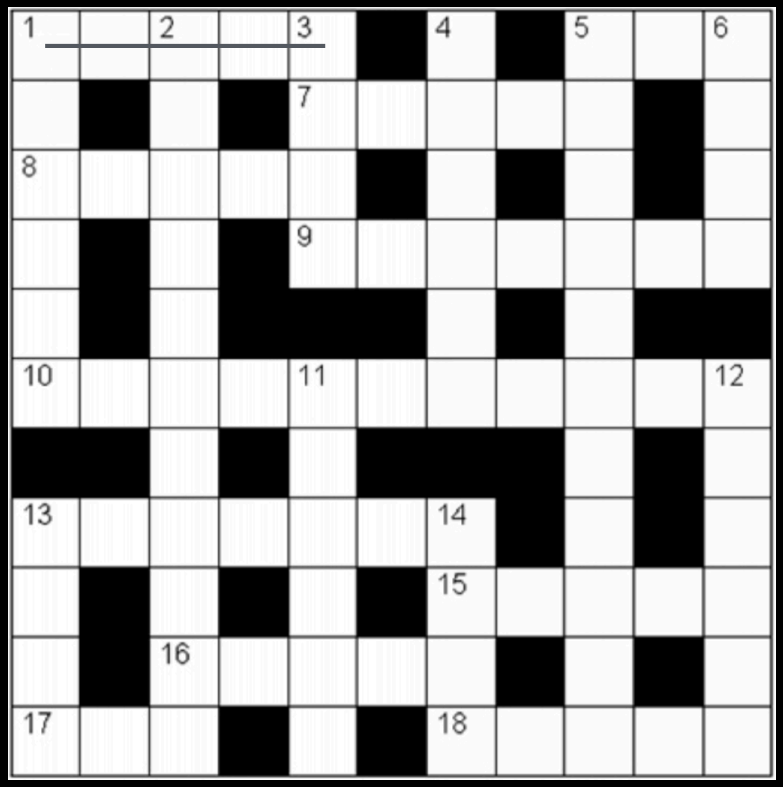


1. Starts by choosing a word variable to be filled, assigns a value based on the dictionary words and the word's length.

2. At each step, the algorithm moves on to a new word and tries to find a valid assignment based on the words that have been already solved, as well as the word's length

3. After a number of steps, if a certain word has no valid assignments, the algorithm backtracks and tries a different assignment

CSP - Optimisation



Minimising the sizes of initial domains:

- Word length:
  D(1H) would not include ‘DELL’

- Unique intersections:
  D(1H) would not include both of
  (‘AXAXA’ and ’AZAZA’)

UNIVERSITÉ TOULOUSE 1 CAPITOLE

The Algorithms & Optimisation
A*

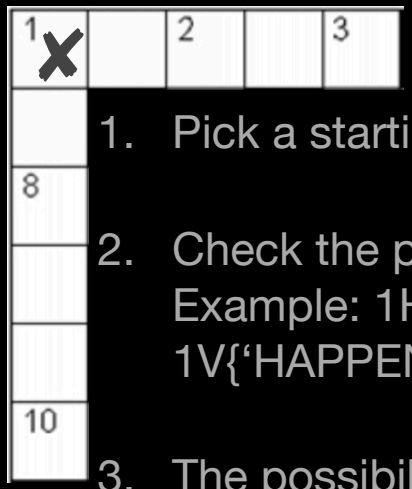
A* - Initialisation

Intersection-based

State: Distribution of letters in intersections

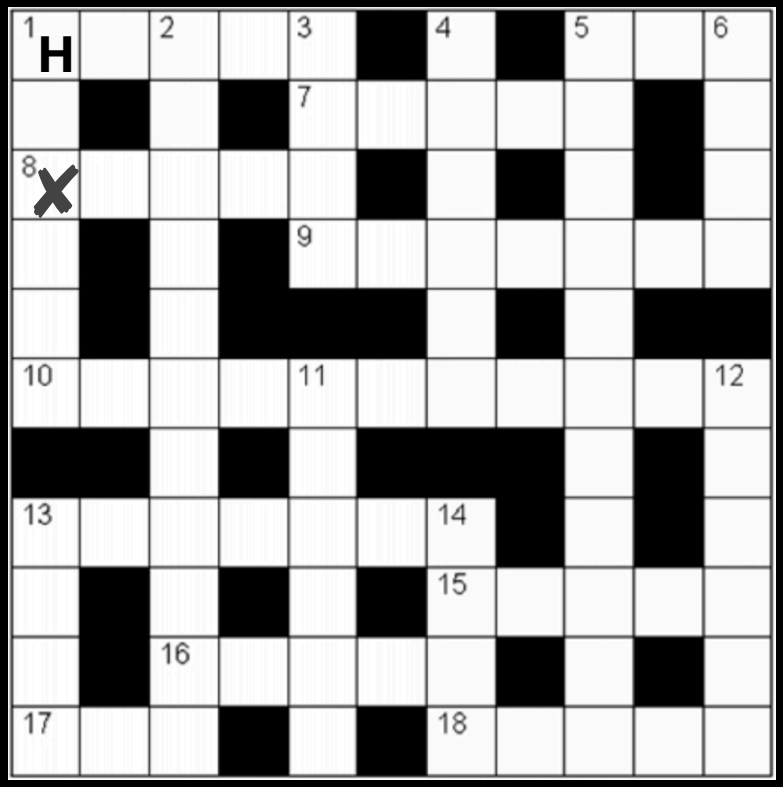
Action: Adding a valid letter to an empty intersection

Goal: 36 intersections filled



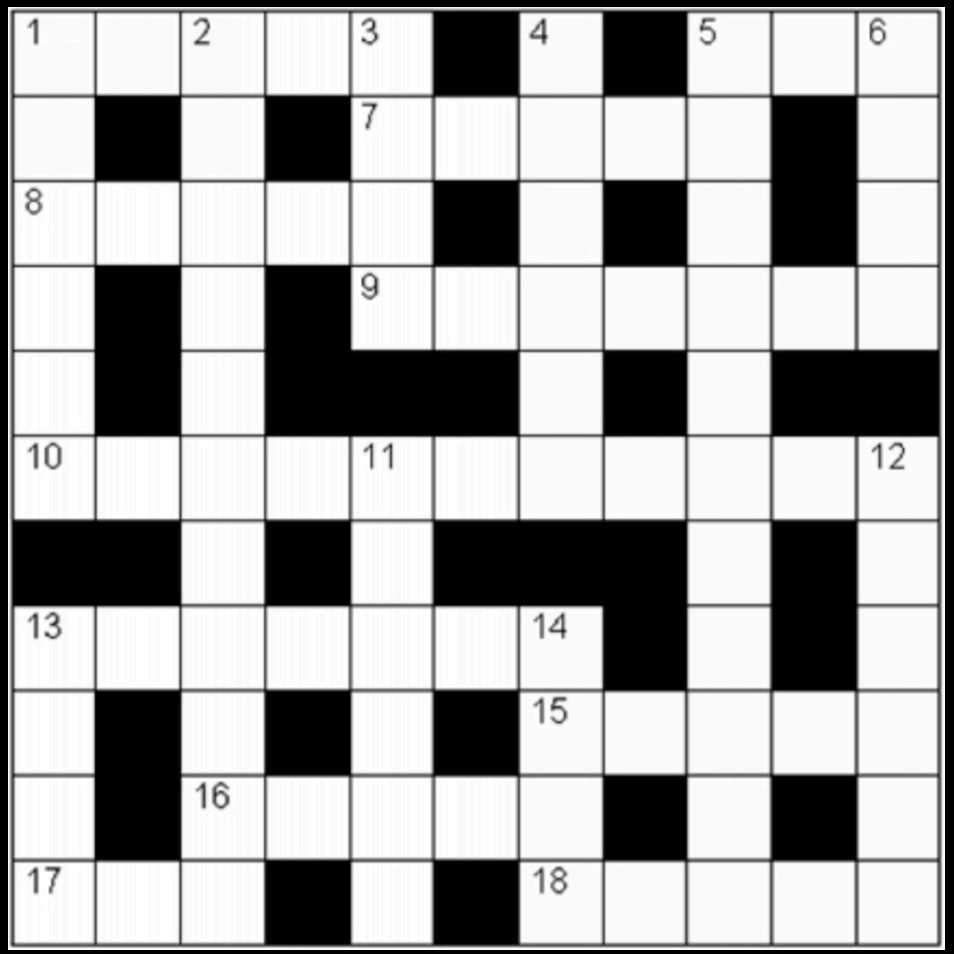
1. Pick a starting intersection X
2. Check the possibilities of neighbouring words 1H, 1V based on their lengths
   Example: 1H{‘HELLO’, ‘EVERY’, ‘SAFER’…},
   1V{‘HAPPEN’, ‘HOTELS’, ‘SCARED’, ‘LABELS’…}
3. The possibilities for X are the common letters between 1H and 1V where the intersect occurs; ==> X{‘H’, ’S’}
4. For every unique value of X, create an initial state and calculate its G and H
5. Add all initial states to the frontier

A* - The Loop



1. Current = get the state with the best F score (e.g. The state with initial intersection = ‘H’)
2. Check if the current state is a goal state
3. Based on the current state, select the next intersection to be solved (e.g. X)
4. Find the possibilities for (X) based on the current state’s solved intersections
5. Create new states with the new intersections and add them to the frontier
6. If any of the words have only non-intersection indices left, pick a valid word string and add it to the state (to avoid duplication)

A* - Heuristic



The h score of a state is the number of possible words that can be filled, which are connected to one or more intersections that are assigned in the state

h(state) = len(getPossibleWords(state))

A* - Optimisation

- Using an intersect-based approach:
  - At each step, instead of considering all possibilities of a word (including indices which are not important), you only consider the unique possibilities of intersections
- Selection criteria ==> pre-ordered list of intersections:
  - Least word possibilities
- Dictionary search:
  - An exhaustive dictionary which uses the combinations of (word length, letter, letter index) as keys

UNIVERSITÉ TOULOUSE 1 CAPITOLE

Algorithm Comparison

A* vs CSP

	CSP - Forward Checking	A*
Execution time	5 - 6 Seconds	37 - 38 seconds
Implementation time	5 minutes	3 weeks

Conclusion: Creating crossword puzzles is a Constraint Satisfaction Problem.

UNIVERSITÉ TOULOUSE 1 CAPITOLE

Thank you!