Searching

Alexandre David

B2-206
The Problem

- Your system is a given state and you want it to reach another state.
  - You have a set of rules that tell you how the system may evolve.
  - You don’t know how to get to the target state trivially.
- General problem, typically in the field of planning and AI.
- Classification: Graph algorithm.
Definitions

- **A state** is the snapshot configuration of a system, typically a tuple with the values of all the variables of the system.
- The system changes state by taking transitions. The rules are given by a **transition relation**.
- The set of all states is called the **state-space**.
- A state S is **reachable** if there exists a sequence of transitions from the initial state to S.
  - The sequence of transition is called **trace**, **path**, or **witness**, depending on the field.
Searching, a.k.a. State-space Exploration

Is the target state reachable? If yes, how?
Exploration Algorithm

**search**(init,target):
S={init,white}

while \{a,white\} \in S \neq \emptyset do
  pick \(a,white\) \in S
  if a \sim target then return true
  S = S[(a,black)/(a,white)]
  forall \(a \rightarrow a'\) do
    if \{b,color\} \mid b \sim a' = \emptyset then
      S = S \cup (a',white)
    fi
  done
done
return false
Correctness

- The algorithm explores all possible reachable states.
  - It will terminate if the state-space is finite. This is often the case, you can argue that.
  - When it terminates, it proves that a state is reachable or not.
  - You can add simple information to keep track of predecessors to generate a trace.
Technicalities

- How to represent $S$?
  - Hash table.
  - Compute a hash on a canonical representant of the equivalence class of your state.

- How to pick-up the next state to be explored?
  - FIFO: Breadth-first-search.
  - LIFO: Depth-first search.
  - Priority queue: Guided search with heuristics.
Search Orderings

Breadth-first-search (BFS)

Depth-first-search (DFS)

Gives shortest path but may be more expensive than heuristics or random search.
Application to Your Project

- Given a chess-board with pieces on it, move a piece from a position to another.
  - All the pieces may move by 1 in any direction if the target position is empty.
  - Not trivial to get a simple algorithm “guess” the solution, but this is an instance of a more general search problem.
- The initial state is given by the initial configuration of the board.
- The final state is given by the configuration of the board with the piece moved to the wanted position.
Formalizing the Problem

- The state is a board (array) $B[8][8]$ of pieces.

- 2 states $B$ and $B'$ are said equivalent (noted $B \sim B'$) iff $\forall i, j : B[i][j] = B'[i][j]$.

- You can try to code the fact that we don’t care about the nature of the pieces, except for the initial and final states, which is the problem.

- Transition relation:

$$a = B[i][j] \neq \bot, B[i'][j'] = \bot, i' = i \pm 1 \text{ xor } j' = j \pm 1$$

$$B \rightarrow B[a / B[i'][j'], \bot / B[i][j]]$$
Practice

- Hash table for $S$.
- Write a function to generate the successor states (transition relation).
- The successor states are looked-up in $S$.
- Have a queue (FIFO, LIFO, priority) to keep references to the “white” states.
- **BIG PROBLEM**: State-space explosion, so use a heuristic to guide the search.
  - $\perp$, P, Bishop, Knight, Rook, Knight, Queen, on any 8x8: $7^{64}$ states.
- Extension of the transition relation: Allow diagonal moves.