LECTURE 5

SUMMARY

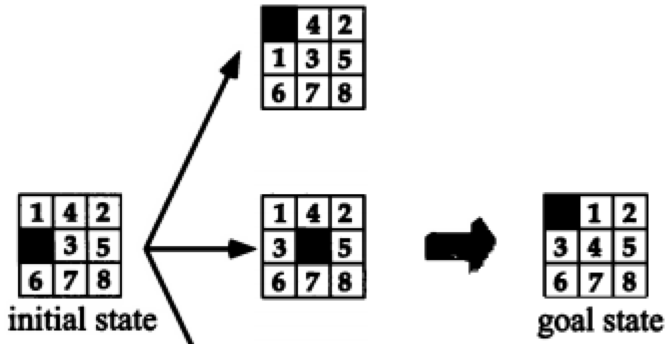
1. Search algorithms for agents

- **Searching** has a long history in AI. The main problems that have been addressed by search algorithm can be divided in three classes:
  1. **Constraint satisfaction problems** (CSP)
  2. **Path-Finding problems** (PFP)
  3. **2-player games**
- From the perspective of searching, **machine learning** (ML) may be viewed as *searching for the best hypothesis that is consistent with the (training) data*.

- **Examples**
  1. A **PFP** involves finding a path from an initial state of a problem to a final state

     - a classical toy PFP is the well-known **n-puzzle** problem.
     - there are $n=k^2$-1 tiles on a k x k board (one square is empty)
     - the **allowed moves**
       - To slide any tile that is horizontally or vertically adjacent to the empty square into the position of the empty square.
     - the **objective**
       - To find a path
         - transform a given initial configuration into a goal configuration, by making the allowed moves.

Initial configuration	**Goal configuration**
_ 2 3	9 6 1
8 5 1	3 _ 4
4 9 6	5 2 8



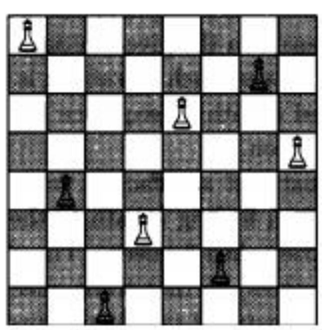
Example of a path-finding problem (8-puzzle).

[1]

- The classical AI **algorithm** used for solving PFP is A*
    - It may be proven to be **admissible** (i.e. complete and optimal) is the heuristic function does not overestimate its real value.
    - It is **exponential** both at time and space complexity O(b^d)
        - *b* is the branching factor (the average number of successors per state);
        - *d* is the depth of the search

2. A classical toy **CSP** is the well-known **n-queens** problem

- a CSP problem involves finding a goal configuration, rather than finding a path to a goal configuration.
- the **objective** is to place *n* queens on an *n* x *n* board so that theses queens will not threaten each other.
- the **goal** is to find a configuration that satisfies the given conditions (constraints).



Example of a constraint satisfaction problem (8-queens).

[1]

- the classical (AI) **algorithm** used for solving CSP is **backtracking.**
  - it performs a *depth-first* search of the space of potential solutions
    - it is better than the brute force **generate-and-test** method, but its runtime complexity is still **exponential**.
  - it is **exponential** as time complexity
    - a CSP is NP-complete

Distributed search

- Most search algorithms (for constraint satisfaction, path finding) were originally developed for single-agent problem solving.
- **Question** What kind of algorithms would be useful for cooperative problem solving by multiple agents?
  - In general, an agent is assumed to have *limited rationality* – the agent must do a limited amount of computations using only partial information on the problem and then take appropriate *actions* based on the available resources.

  - In most standard search algorithms (backtracking, A*), each step is performed sequentially, and for each step, the global knowledge of the problem is required (for ex., the A* algorithm extends the set of explored states from the initial state and chooses the most promising state within the whole set.

  - A search problem can be represented as a graph, and there exist search algorithms with which a problem is solved by performing *local computations* for each node in the graph
    - The execution order of these local computations can be *arbitrary* or *flexible*
    - The local computations can be executed *asynchronously* and *concurrently*

      ⇒ **ASYNCHRONOUS SEARCH ALGORITHMS** (the *local computations* are performed by *agents*)

  - When a problem is solved by multiple agents, *asynchronous search algorithms* are appropriate
    - If multiple agents are cooperatively solving a problem using the asynchronous search algorithm, the execution order of these agents can be arbitrary or highly flexible (otherwise, we need to synchronize the computations of the agents – this can be difficult).

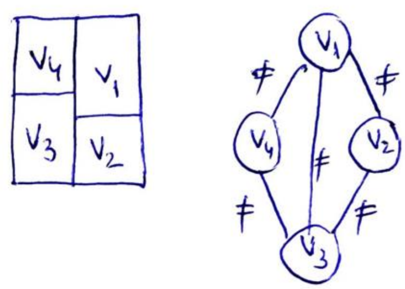
2. Constraint satisfaction problems ([1], Section 4.2)

2.1 Fundamentals of CSP in classical AI

- **Constraint satisfaction** (CS) consists of methods to solve problems stated in the form of a set of **constraints**.
  - examples of toy problems: graph colouring, 8-queens, etc
  - a specialization of CS is **constraint optimization** (CO)
    - an additional objective function that has to be optimized

    - deals with finding an optimal solution
- the general CSP is NP-complete
- **formally**, a CSP is a triple $< X, D, C >$, where:
  - $X = \{x_1, x_2, \ldots, x_n\}$ is a set of **variables**
  - $D = \{D_1, D_2, \ldots\ D_n\}$ is a set of **domains**
  - $C = \{c_1, c_2, \ldots\ c_k\}$ is the set of **constraints**
    - a constraint c_i is a relation defined on a subset of all variables (a predicate)
    - the constraint $p_i(x_{i1}, x_{i2}, \ldots\ x_{ij})$ is a predicate defined on $D_{i1} \times D_{i2} \times \ldots \times D_{ij}$
      - this predicate is true iff the value assignment of these variables satisfies the constraint
    - if *c* is a constraint and *var*(*c*) is the set of variables for *c*
      - if *var*(*c*) has one element $\Rightarrow$ *c* is a **unary** constraint
      - if *var*(*c*) has two elements $\Rightarrow$ *c* is a **binary** constraint

- a CSP is a **binary CSP** if all its constraints are **unary** or **binary**
  - a binary CSP can be visualized using a graph – **constraint graph**
    - vertices $\rightarrow$ variables
    - two vertices are connected iff there is at least one constraint inferring to the correspond variables
    - e.g.. a graph coloring problem



  - **any** CSP can be transformed into an equivalent **binary** CSP

- **Constraint programming (CP)** is a programming paradigm related to CS.
  - it is a form of declarative programming;
  - relations between variables are stated in the form of constraints
  - constraint logic implementations of **CP** are GNU Prolog, CHIP (*Constraint Handling in Prolog*).
- **solving** a CSP
  - finding an assignment of values to all variables such that all constraints are satisfied
    - all solutions
    - one solution
    - an optimal solution (given a criterion function)
- **over-constrained CSP**
  - there is no solution satisfying all the given constraints
  - $\Rightarrow$ the classical CSP problem is relaxed, s.t. to allow a solution
    - fuzzy CSP
    - probabilistic CSP

        - weighted CSP
        - hierarchical CSP
- other extensions of the classical CSP:
  - dynamic CSP (the set of constraints is dynamic)
  - machine/deep learning for CSP
  - …

- **examples** of CSP
    - Crossword puzzles
    - Sudoku game
    - Scheduling a set of tasks
    - Resource allocation
    - Transportation scheduling
    - Assignment problems (timetable, etc)

- CS and CO are also used in the research from various domains, such as:
  - **Search based software engineering** (SBSE)
    - Program understanding, design recovery, design patterns identification, software re-engineering, etc.
  - **Bioinformatics**
    - Sequence alignment, recognition and analysis of DNA, biological systems simulation, etc.
  - **Medicine**
    - Evaluation of treatments, hospital scheduling optimization, therapy planning etc,.
    - A computational model based on CSP for *autism*.

- approaches
  1. constraint propagation
     - for simplifying the original problem
  2. direct search for possible solutions
     - Backtracking (BT)-like algorithms
       - "intelligent" backtracking
         - BackJumping (BJ)
         - Conflict-directed BJ (CBJ)
         - Graph-based BJ
         - **Backmarking**
       - hybrid algorithms
         - BM+CBJ
       - optimizations
         - ordering heuristics
           - **fail-first-principle** (FFP)
             - variables with the least possible values are instantiated first
           - **minimum width ordering** (MWO)
             - if a variable does not depend on many earlier instantiated variables, then it will be easier to assign an appropriate value to it
           - **minimum conflict first (MCF)**

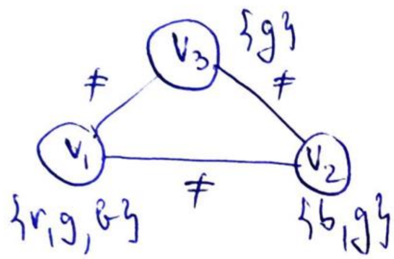
- the variable in conflict with a minimum number of variables is instantiated first
- Branch and Bound (A*)
  - constraint optimization
3. combinations of 1. and 2.
4. other methods
   - **lookahead algorithms**
     - forward checking (FC)
       - checks the satisfiability of the binary constraints and removes the values which are not compatible with the current variable's instantiation
     - minimal forward checking (MFC)
       - BM+FC
   - other
     - local search
     - genetic algorithms
     - particle swarm optimization (PSO)
     - neural networks
       - various NN-based approaches

2.2 Consistency techniques

- various consistency techniques
  - node consistency
  - arc consistency
  - path consistency
- the BT paradigm suffers from **trashing**
  - main causes
    - lack of node and arc consistency
    - **example**
      - the variables are instantiated in the order $V_1, V_2, \ldots V_n$
      - the binary constraint between V_i and V_j is such that for V_i=”a” it disallows any value of V_j

        $\Rightarrow$ in the backtrack search tree, whenever V_i is instantiated to “a”, the search will **fail** while trying to instantiate V_j
  - trashing due to arc inconsistency can be avoided if before the search each arc of the constraint graph is made consistent

    $\Rightarrow$ constraint propagation
  - **arc consistency**
    - an arc (V_i, V_j) is *consistent* if for every value x in the current domain of V_i there is some value există o valoare y in the domain D_j of V_j such that the instantiations $V_i = x$ and $V_j = y$ are allowed by the binary constraint between V_i and V_j.



   - (V_3, V_2) is consistent, but (V_2, V_3) is not consistent
- if an arc (V_i, V_j) is not consistent $\Rightarrow$ remove from the domain of V_i all the values x which do not verify the consistency conditions
- the **constraint graph consistency**
   - the consistency of the arc is repeatedly applied until all the arcs are consistent
   - algorithms; AC-1, AC-3,…
- !!! ensuring the arcs consistency is not enough for finding a solution of the CSP
   - 3 situations after ensuring the constraint graph consistency
      - the domain of one variable becomes empty $\Rightarrow$ the problem is **over-constrained**
      - if each domain has a unique value $\Rightarrow$ the solution of the CSP
      - if there exist multiple values in the domains of the variables $\Rightarrow$ a search is required
- **Theorem**. If a binary CSP has a **tree structure**, and it is **node** and **arc consistent**, then a solution can be constructed without backtracking

Bibliography

[1] Weiss, G. (Ed.): Multiagent Systems: *A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999 (available at www.cs.ubbcluj.ro/~gabis/weiss/Weiss.zip) [Ch. 3]

[2] Șerban Gabriela, *Sisteme Multiagent în Inteligenta Artificiala Distribuita. Arhitecturi si Aplicatii*, Editura RisoPrint, Cluj-Napoca, 2006

[3] Czibula Gabriela, *Sisteme inteligente. Instruire automată*, Editura RisoPrint, Cluj-Napoca, 2008