Lecture 3: Search

Intelligent Systems—Problem Solving
Pierre Talbot

Goals

⋆ Study a generic solving algorithm and its properties.

1 Search Strategy

A variable x is *unassigned* if $|d(x)| > 1$. A *variable selection strategy* is a function $select : \mathbf{D} \to X \cup \{\bot\}$, such that $select(d)$ is the next variable to choose, or $\bot$ if there is no unassigned variable in d. We suppose $X = x_1, \ldots, x_n$, that is, the variables are totally ordered. A simple selection strategy is to choose the first unassigned variable.

```
function $firstvar(d)$
    for $1 \leq i \leq |X|$ do
        if $|d(x_i)| > 1$ then
            return $x_i$
        end if
    end for
    return $\bot$
end function
```

Exercise 1

Define the variable selection strategy *first-fail*, which consists in selecting the unassigned variable with the smallest domain.

end of exercise.

Once a variable has been selected, we must choose which value to assign to it. This is the role of the *value selection strategy*. The *smallest* strategy selects the smallest value in the domain of the variable.

```
function $smallest(d, x)$
    return $\min d(x)$
```

end function

The combination of *first-fail* and *smallest* is a reasonnable search strategy to solve constraint problems. Next, we can assemble those strategies to obtain a simple solving algorithm based on enumeration.

2 Solve by Enumeration

The simplest search algorithm is to iterate over all combinations of all values of d, e.g., $d(x_1) \times \ldots \times d(x_n)$, and check which ones satisfy the constraints.

We write $d(x) = S$ the in-place update of the domain function d—similarly to what we would do in a programming language if d is represented as an array (e.g., $d[x] = S$) or dictionnary.

```
 1: function $enumerate(d, C)$
 2:     $x = firstvar(d)$
 3:     if $x = \perp$ then                          If all the variables are assigned to a value.
 4:         $s = \{x \mapsto v \mid d(x) = \{v\}\}$                We create an assignment from d.
 5:         for $c \in C$ do                   We check if the assignment satisfies each constraint c.
 6:             if $s \notin rel(c)$ then            If not, we return the empty set (no solution).
 7:                 return $\{\}$
 8:             end if
 9:         end for
10:         return $\{s\}$
11:     else
12:         $dx = d(x)$                                           We save the current domain of x.
13:         $v = smallest(d, x)$                                 We select a value from the domain.
14:         $d(x) = \{v\}$                   We explore the left branch of the search tree where $x = v$.
15:         $S_1 = enumerate(d, C)$
16:         $d(x) = dx \setminus \{v\}$         We explore the right branch of the search tree where $x \neq v$.
17:         $S_2 = enumerate(d, C)$
18:         $d(x) = dx$                                                        Restore the domain.
19:         return $S_1 \cup S_2$
20:     end if
21: end function
```

For clarity, we have used two specific variable and value selection strategies, but this algorithm can easily be made generic for any pair of strategies.

Exercise 2

Modify *enumerate* to stop after you found one solution. Use the line numbers to indicate where you modify the algorithm.

end of exercise.

Exercise 3

The algorithm *enumerate* explores a *search tree* where a leaf is reached when $x = \perp$ (line 3). Is it a binary tree? What is the tree search algorithm underlying *enumerate*: breadth-first search, depth-first search or best-first search?

end of exercise.

Exercise 4

The expression $s \notin \mathit{rel}(c)$ (line 7) is inefficient as the set $\mathit{rel}(c)$ might be large. Write a function $\mathit{check} : \mathbf{Asn} \times \mathbf{C} \rightarrow \{\texttt{true}, \texttt{false}\}$ such that $\mathit{check}(a, c)$ is true whenever the assignment a satisfies the constraint c. Write check for the constraints $x = y$ and $x \leq y + z$. Furthermore, formally state what it means to be "correct" for the function check.

end of exercise.

3 Propagate and Search

Exercise 5

Modify $\mathit{enumerate}$ to apply the procedure $\mathit{GAC_1}(d, C)$ in each node, which modifies d in-place.

end of exercise.

Exercise 6

What are the variables that can change between two calls to $\mathit{GAC_1}$, excluding the ones modified by $\mathit{GAC_1}$? Modify $\mathit{enumerate}$ to apply the procedure $\mathit{propagate}(d, C, E)$ instead of $\mathit{GAC_1}$ where E is the set of variables that change between two calls to $\mathit{propagate}$. We call this algorithm $\mathit{propsearch}$ for *propagate and search.*

end of exercise.

A solving algorithm $solve : \mathbf{D} \times \mathbf{C} \rightarrow \mathcal{P}(\mathbf{Asn})$ is sound and complete for any constraint network $\langle d, C \rangle$ if it satisfies:

$$\begin{array}{ll} sol(d,C) \subseteq solve(d,C) & \textit{(soundness)} \\ sol(d,C) \supseteq solve(d,C) & \textit{(completeness)} \end{array}$$

Those properties formalize what we usually mean by a "correct solving algorithm". Typically, if you want to define your own solving algorithm, you will have to prove those properties. For some classes of algorithms, such as genetic algorithms, solving is only complete, in the sense that it does not guarantee we can ever find all solutions (or even one), but if it finds one, it is a solution. On the other hand, a linear programming solver is typically sound but not complete due to numerical imprecision of floating-point numbers.

Exercise 7
Prove or disprove whether $soldom \circ propagate$ is a complete solving algorithm, where $soldom(d, C) \triangleq sol(d, \{\})$.

end of exercise.

We now abstract from a precise consistency and implementation of $propagate$. The soundness of $propsearch$ depends on whether $propagate$ is sound. What to say about completeness? By the answer to the previous question, we know that $propagate$ is not complete, but we still want $propsearch$ to be complete. To achieve that, we require $propagate$ to be *complete on singleton domain functions*, which is a weaker property than being complete on any domain function. Formally, for any constraint network $\langle d, C \rangle$, whenever $soldom(d) = \{a\}$, then $sol(d, C) \supseteq soldom(propagate(d, C, X))$, i.e., $propagate$ must decide whether the assignment a is a solution or not. If it is not a solution, $propagate$ must wipeout at least the domain of a variable, i.e., $\exists x \in X, d(x) = \{\}$.

Exercise 8 – Branch-and-bound
Modify $propsearch$ to return the solution such that an objective variable $obj \in X$ is maximized.

end of exercise.

4 More About Constraint Programming

- *Constraint networks: techniques and algorithms*, Christophe Lecoutre, 2008 (book).
- Coursera on *Basic Modeling for Discrete Optimization* to learn how to model constraint problems using MiniZinc: `https://www.coursera.org/learn/basic-modeling`.