Environmental Modeling and Decision Support Systems

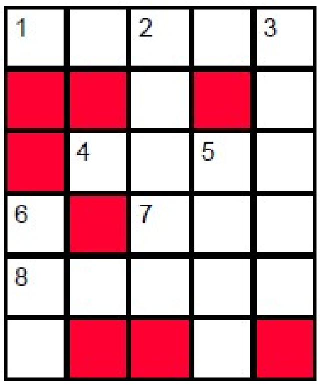
Prof. Dr. Struss, Dr. Dressler

Chapter 3.4 Constraints

We developed a very general notion of a behavior model, namely defined as a relation over the variable domains, and a fundamental step was to compute this relation for an aggregate model (i.e. the join of relations), which includes a consistency check (because a resulting empty relation is equivalent to an inconsistency).

This general problem statement is the subject of so-called constraint systems in AI, which, hence, offer a way to implement model-based prediction and consistency checking. We illustrate some foundations of the field of constraint satisfaction using a very different example (which shows the broad scope of applicability of the solutions).

Example: A Crossword Puzzle



Instructions

- fill in words from the list

List of Words

- Aft
- Ale
- Eel
- Hike
- Hoses
- Keel
- Knot
- Laser
- Lee
- Line
- Sails
- Sheet
- Steer
- Tie

Figure 3.4.1 Crossword Puzzle

A crossword is a word puzzle that normally takes the form of a square or rectangular grid of white and shaded squares. The goal is to fill the white squares with letters, forming words or phrases, by solving clues which lead to the answers.[1] In our example, it has been simplified by pre-determining the words to be filled in (let us assume, each one only once). As an exercise, try to solve it and, more importantly, try to observe **how** you solve it and try to describe this as a **general principle**.

Definitions

Variables:a symbol that stands for a value that may vary.
$V = \{v_1, v_2, ..., v_n\}$
Domains:the territory governed by a single ruler or government(one meaning in dictionary).
$D = \{DOM(v_1), DOM(v_2), ..., DOM(v_n)\}$

Constraints C:
relations $R_i \subseteq DOM(v_{k1}) \times DOM(v_{k2}) \times \ldots \times DOM(v_{ki})$
Constraint Satisfaction Problem (CSP):
Formally, a constraint satisfaction problem is given by a triple (X, D, C), where X is a set of variables, D is a domain of values, and C is a set of constraints. The problem is to determine one or all solutions, i.e. tuples that satisfy all constraints of C.

Example: CSP for the Crossword Puzzle

Variables: V = {1 across, 4 across, 2 down, ... }
Domains: For all i, DOM_i = {Aft, Ale, Eel, Hike, ...}
Constraints:
For each variable v_i:Equal(LengthWord($_{vi}$), NumberPositions(v_i))
For two variables v_i, v_j sharing positions p_{ik}, p_{jl}:Equal(Character(p_{ik}),Character(p_{jl}))

Graphical Representations of CSPs

In general, a **graph** is a tuple (V, U) where V is a set of nodes and $U \subseteq V \times V$ is a set of arcs. Nodes have different domains and an arc is a pair of nodes.The nodes in an arc are ordered whereas the nodes in an edge are not. An edge can be seen as a pair of arcs (x,y) and (y,x).

A binary CSP is often visualized as a constraint graph. There are two graphical representations: 1,Primal Constraint Graph:nodes represent variables and hyperarcs represent constraints.

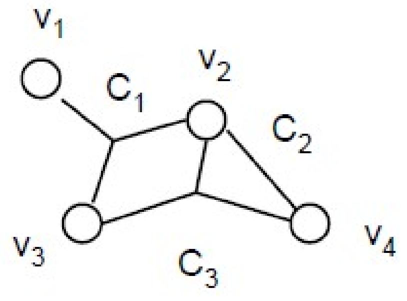


Figure 3.4.2 Primal constraint graph

2,Dual Constraint Graph:nodes represent constraints,multiarcs represent variables.

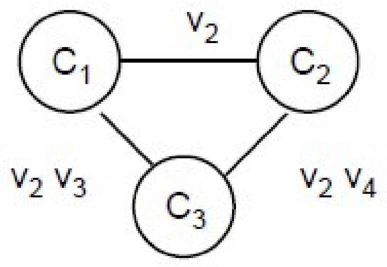


Figure 3.4.3 Dual constraint graph

In the following, we choose the first representation.

Example: Primal Constraint Graph for the Crossword Puzzle

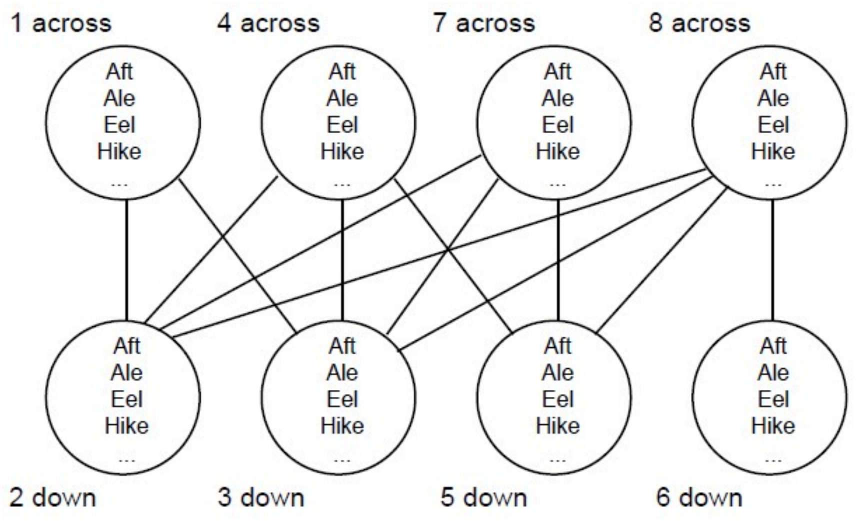


Figure 3.4.4 Primal constraint graph for the crossword Puzzle

Solving CSPs

There are different approaches to solving the problem:

Basic Search Methods:
General search strategies:
– generate and test
– chronological backtracking: backtracking to the last choice
"gather information while searching" strategies
– dependency-directed backtracking: (ddp) backtracking to a choice that caused the inconsistency
– truth maintenance systems: record dependencies and avoid choices leading to discovered inconsistencies.

The basic algorithm to search for solution tuples is **simple backtracking.**

The basic operation is to pick one variable at a time, and consider one value for it at a time, making sure that the newly picked label is compatible with all the labels picked so far. Assigning a value to a variable is called **labelling**. If labelling the current variable with the picked value violates certain constraints, then an alternative value, when available, is picked. If all the variables are labelled, then the problem of finding one solution is solved.

If at any stage no value can be assigned to a variable without violating any constraints, the label which was last picked is revised, and an alternative value, when available, is assigned to that variable. This carries on until either a solution is found or all the combinations of labels have been tried and have failed.

Since the BT algorithm will always backtrack to the last decision when it becomes unable to proceed, it is also called **chronological backtracking**.

Let n be the number of variables, e be the number of constraints, and a be the domain sizes of the variables in a CSP. Since there are altogether a^n possible combinations of n-tuples (candidate solutions), the **time complexity** of this backtracking algorithm is $O(a^n e)$.

Sometimes, there is enough information to know that revising the last decision will not lead to a solution. It is better to backtrack to the last choice that caused the inconsistency. This is called **dependency-directed backtracking**. Obviously, this avoids inspection of some tuples which would be visited by chronological backtracking. However, in a different branch, the algorithm would have to "re-discover" this inconsistency again.

If the (minimal) combinations of values that cause inconsistencies are stored, the algorithm can avoid ever visiting this combination (or supersets thereof) again. Recording such dependencies and inconsistencies is done by **truth-maintenance systems**, which often help to reduce the search space tremendously, of course, at the cost of storing and updating this information.

Problem Reduction

The basic Idea is reduce size of CSP without ruling out solutions.
How can this be done?
Constraint Filtering means removing domain values for nodes (1-consistency, "node consistency") that cannot be part of solutions, remove values for arcs (2-consistency, "arc consistency"), i.e. delete those values in the domain that do not occur in a tuple satisfying the constraint along this arc ("has no partner value from the domain of the other variable") , ..., remove values for paths of length k-1 (k-consistency), i.e. delete values from the domain.

Algorithm for Node Consistency

A node is **1-consistent** if and only if every value in every domain satisfies the unary constraints on the subject variable..

A CSP is **node-consistent (NC)** if and only if all nodes are 1-consistent..

```
Procedure NC (V, D, C)
For each $v_i \in V$
   For each $R_j \in C$ with $R_j \subseteq DOM(v_i)$
       For each $val \in DOM(v_i)$
         If $val \notin R_j$ Then $DOM(v_i) \leftarrow DOM(v_i) \setminus \{val\}$
Return (V, D, C)
```

Figure 3.4.5 Procedure NC

For each node v_i in the Graph nodes set V, each constraint R_i from the set of constraints C is applied to domain of v_i. Every value of a node is checked, and if it does not satisfy the constraint, it is removed from the domain of the node. As a result, nodes are left with only those values which satisfy the domain constraints.

Crossword Puzzle: After Applying Node Consistency

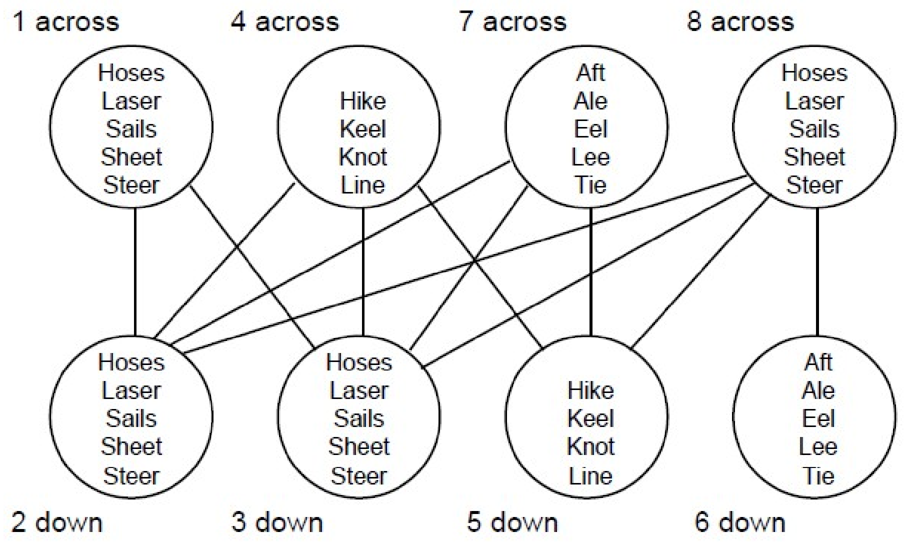


Figure 3.4.6 After applying node consistency

For each node, the word length constraint is applied, that a particular slot can have only have a word of particular length. For example on slot 7, word length is 3, so after constraint application, its domain is left only with 3 length words.

Definition: (Classical) Arc-Consistency (2-Consistency)

An arc (x, y) in the constraint graph of a CSP (Z, D, C) is **arc-consistent (AC)** if and only if for every value a in the domain of x which satisfies the constraint on x, there exists a value in the domain of y which is compatible with <x,a>

A CSP is **arc-consistent (AC)** if and only if every arc in its constraint graph is arc-consistent:

An important concept in CSPs is the concept of local consistency. Local consistency is a property that can be applied in a CSP, using (typically) polynomial algorithms, to remove inconsistent values either prior to or during search. Arc consistency is the most commonly used local consistency property in the existing constraint programming engines.

Algorithm for Arc Consistency

```
Procedure REVISE_DOMAIN ($v_i$,$v_k$)
deleted ← False;
For each $val_i \in DOM(v_i)$
    If Not exists $val_k \in DOM(v_k)$ such that $(val_i,val_k) \in R_{i,k}$
        Then
         $DOM(v_i) \leftarrow DOM(v_i) \setminus \{val_i\}$;
        deleted ← True
return(deleted)
```

Figure 3.4.7 Procedure REVISE_DOMAIN(Vi,Vk)

It is checked whether a particular arc(v_i,v_k) arc consistent. For an arc(v_i,v_k), each value of v_i from the domain of v_i is checked with existence of pair value of v_k, which satisfies constraint R_{ik}. If the pair value is not found then that value is removed from the domain v_i. If any value is removed at any point, it returns a 'true' value, otherwise false.

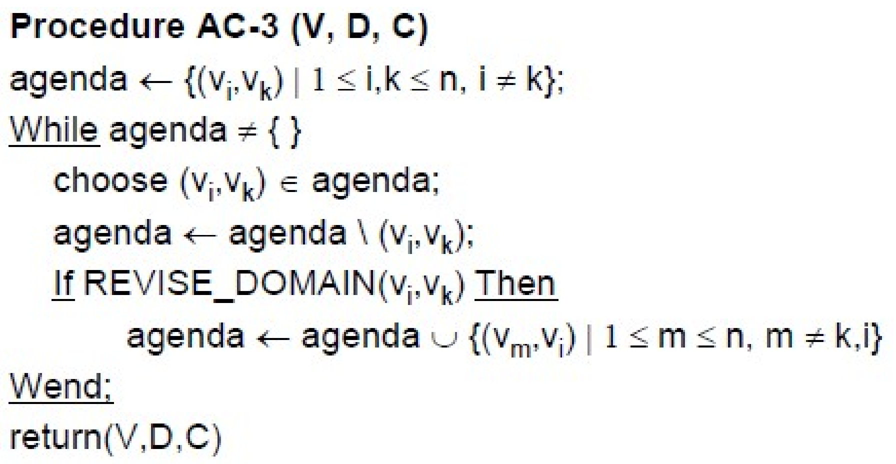
AC-1: A Simple Algorithm for Arc Consistency

```
Procedure AC-1 (V, D, C)
agenda ← $\{(v_i,v_k) \mid 1 \le i,k \le n,\ i \ne k\}$;
Repeat
    changed ← False;
    For each $(v_i,v_k) \in$ agenda
        changed ← REVISE_DOMAIN($v_i$,$v_k$) or changed
Until Not changed;
return(V,D,C)
```

Figure 3.4.8 Procedure AC-1(V,D,C)

All the node pairs are placed on the agenda. A pair (v_i,v_k) is selected from agenda and single arc consistency algorithm is applied. if the REVISE_DOMAIN returns true (i.e. in case of any change), this may have an impact on other constraints which are then checked again, until no change is detected.

AC-3: An Improved Algorithm for Arc Consistency

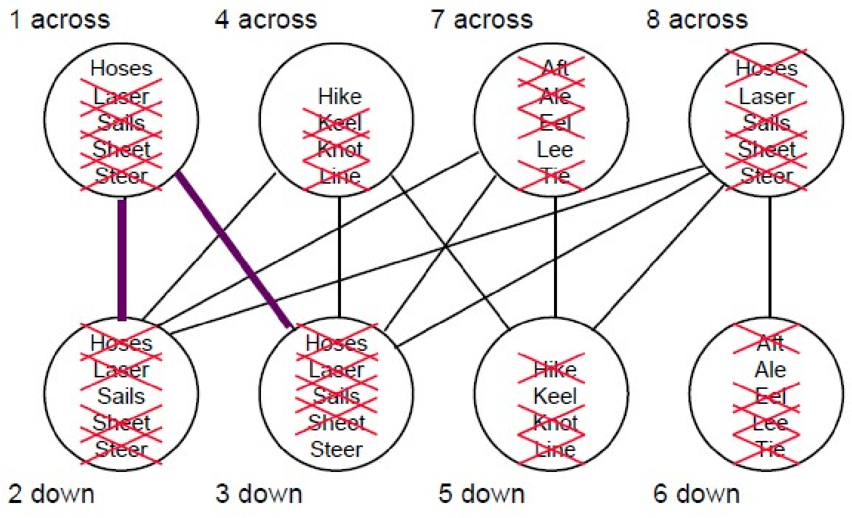


```
Procedure AC-3 (V, D, C)
agenda ← {($v_i$,$v_k$) | $1 \le i,k \le n$, $i \ne k$};
While agenda ≠ { }
    choose ($v_i$,$v_k$) ∈ agenda;
    agenda ← agenda \ ($v_i$,$v_k$);
    If REVISE_DOMAIN($v_i$,$v_k$) Then
        agenda ← agenda ∪ {($v_m$,$v_i$) | $1 \le m \le n$, $m \ne k,i$}
Wend;
return(V,D,C)
```

Figure 3.4.9 Procedure AC-3(V,D,C)

The previous algorithm AC-1 is not efficient, since after a single change on a particular arc, it checks each pair in the agenda again. This is corrected in algorithm AC-3; the chosen pair is removed from the agenda. If a change is detected for a node v_i, only those nodes are added in agenda again, which are effected by v_i.

Example: Crossword Puzzle: Applying Arc Consistency



1 across
4 across
7 across
8 across
Hoses ~~Laser~~ ~~Sails~~ ~~Sheet~~ ~~Steer~~
Hike ~~Keel~~ ~~Knot~~ ~~Line~~
~~Aft~~ ~~Ale~~ ~~Eel~~ Lee ~~Tie~~
~~Hoses~~ Laser ~~Sails~~ ~~Sheet~~ ~~Steer~~
~~Hoses~~ ~~Laser~~ Sails ~~Sheet~~ ~~Steer~~
~~Hoses~~ ~~Laser~~ ~~Sails~~ ~~Sheet~~ Steer
~~Hike~~ Keel ~~Knot~~ ~~Line~~
~~Aft~~ Ale ~~Eel~~ ~~Lee~~ ~~Tie~~
2 down
3 down
5 down
6 down

Figure 3.4.10 Applying Arc consistency

After applying arc consistency only single values as a solution are left (which is not the case in general).

Example: Crossword Puzzle: Solution

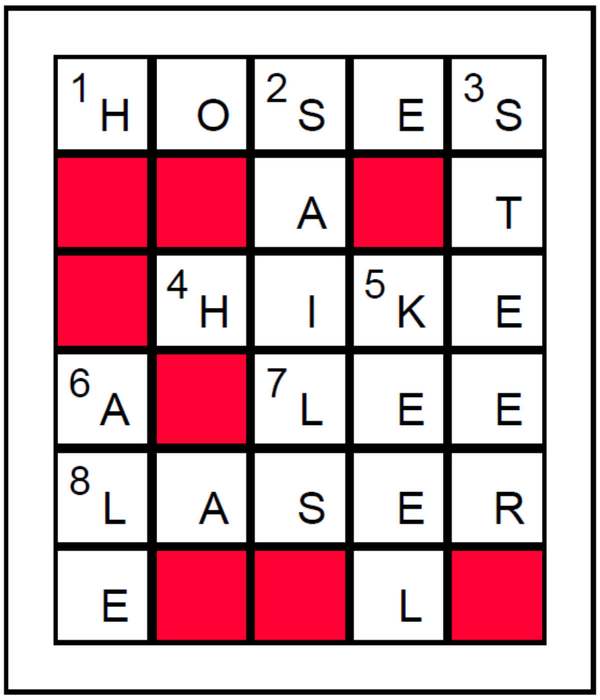


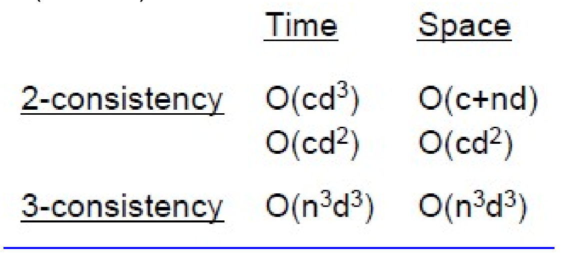
Figure 3.4.11 Solution of crossword puzzle

Procedure AC-1(V,D,C)

In a constraint network R = (V, D, C)
{ v_i, v_j } is path-consistent relative to vk if and only if for every consistent assignment ($v_i = a_i$, $v_j = a_j$), there is a value $a_k \in$ DOM(v_k) such that the assignments (v_i=a_i, v_k=a_k) and (v_k=a_k,v_j=a_j) are consistent.
A binary constraint R_{ij} is path-consistent relative to v_k if and only if for every tuple $(a_i,a_j) \in R_{ij}$ there is $a_k \in$ DOM(v_k) such that $(a_i,a_k) \in R_{ik}$ and $(a_k,a_j) \in R_{kj}$.
A constraint network is path-consistent if and only if every R_{ij} is path-consistent relative to v_k for every $k \neq i, j$.

Complexity of Constraint Filtering (Binary CSPs)

Polynomial Algorithms:
n: number of variables
c: number of constraints
d: (maximal) domain size



	Time	Space
2-consistency	$O(cd^3)$	$O(c+nd)$
	$O(cd^2)$	$O(cd^2)$
3-consistency	$O(n^3d^3)$	$O(n^3d^3)$

Figure 3.4.12

Example: Centigrade - Fahrenheit: 9C = 5(F-32)

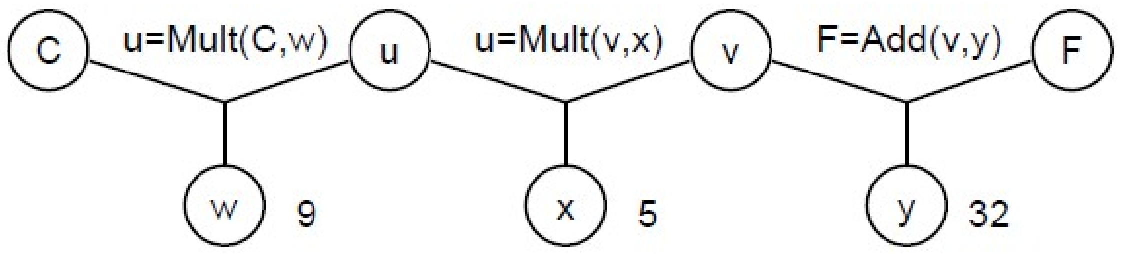


Figure 3.4.13

Constraints are relations and have no built-in “direction”. The constraint network in Fig. 3.4.13 can be used to calculate Fahrenheit from Celsius and vice versa. Compare this to statement in programming language like C++.

Completeness of Consistency Filtering

Local consistency is a necessary condition for global consistency, but not a sufficient one. In general, k-consistency does in general not guarantee global consistency, i.e. the existence of a solution ([Freuder 78]). For CSPs with a particular structure, this may be different: e.g. [Dechter-Pearl 88] show arc consistency is a sufficient condition for global consistency for tree-structured constraint graphs

Using local methods for calculations in constraint networks also has limitations, illustrated by the following example. Although y is uniquely determined by x=1, z=5, none of the constraints can calculate anything, because there is only one known variable.

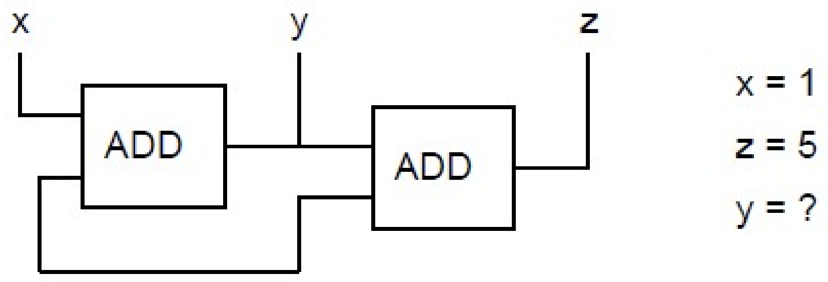


Figure 3.4.14 Incompleteness of consistency filtering

notes

[1] http://en.wikipedia.org/wiki/Crossword