

Optimierung und Entscheidungsunterstützung (OuE)

Anna Prenzel

31.05.2013

Hochschule Zittau/Görlitz

Topics for today:

- 1 Varieties of CSPs
- 2 K-Consistency
- 3 Backtracking
- 4 Optimization
- 5 Binary constraints

Varieties of Variables

Discrete Variables:

- $x \in \{1, 2, 3, 4, 5\}$
- $x \in \{true, false\}$ (Boolean CSP)
- can have **finite** or **infinite** domains (integers, strings)
- **infinite domain:** e.g. start date of a job

Continuous Variables

- real numbers, e.g. speed of a car

Varieties of Constraints

Unary Constraints:

- restricts the value of a single variable
- $x < 5$

Binary Constraints:

- relates two variables
- $x < y$

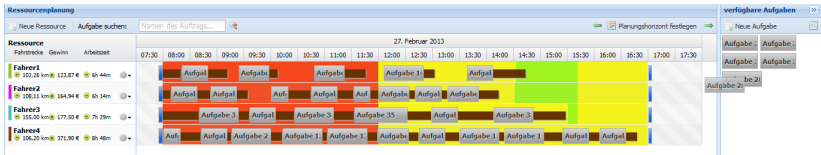
Higher-order constraints:

- involve three or more variables
- $x + y = z$
- $x * y + z^2 = 70$

Consistency for Decision Support

Forward checking (FC):

- after a job i has been positioned in the timeline:
- *for all unscheduled jobs j* : remove start times, that cause overlapping with job i
- remaining start times are highlighted green, when a job is dragged onto the timeline



Consistency for Decision Support

Minium Remaining Value Heuristic:

- variable selection:
- jobs with smaller time windows should be scheduled first → check time windows after each FC step
- otherwise their start times might be occupied by other jobs
- this would require backtracking

Consistency for Decision Support

Least Constraining Value Heuristic:

- value selection:
- *for each start time t in the time window of selected job $j \rightarrow$*
 - ① carry out FC with this start time
 - ② determine, how many values of the remaining variables have been ruled out
- choose start time t that rules out the least number of values after FC
- this is a **Look-Ahead-Strategy** (also called **Select-Value-Forward-Checking**)

Consistency for Decision Support

Arc consistency:

- after *successfully* applying FC, each of the remaining jobs can be positioned in the timeline
- however, it is not guaranteed, that planning can be continued *after* the next planning step
- applying arc consistency = looking ahead two planning steps

K-consistency

- look ahead K planning steps → not practicable
- no backtracking would be required anymore

Basic solver for CSP

Backtracking solver

```
back_solve(partial, variables, csp){

    if empty(variables) return true

    remove x from variables

    for each value v in D_{x}{

        extend partial to partial_neu with x <-- v

        if valid(partial_neu, constraints)
        if back_solve(partial_neu, variables, constraints)
            return true
        }
        else else
        insert x in variables

    return false

}
```

CSP solver with FC

```
back_solve(partial, variables, constraints){

  if empty(variables) return true

  remove x from variables

  for each value v in D_{x}{

    extend partial to partial_neu with x <-- v

    if(forward_checking(x, variables, constraints))
    if back_solve(partial_neu, variables, constraints)
      return true
    }
    else else
    insert x in variables
    restore_values(variables)
    return false
  }
```

CSP solver with FC

Comparison:

- $valid(partial_neu, constraints)$ checks partial solution
- $forward_checking(x, variables, constraints)$ modifies the remaining variables $\rightarrow$ whatever value is selected in the next iteration, it will be valid

CSP solver with FC

- not necessary to use partial solution in fc → domains of the variables have been already reduced with regard to the other variables in *partial_neu*
- fc returns **false**, if it results in an empty domain for one of the remaining *variables*
- fc can be exchanged by ac (arc consistency) → **Tip:** use AC as preprocessing step before calling *back_solve*

CSP solver with Variable/Value selection strategies

```
back_solve(partial, variables, constraints){

  if empty(variables) return true

  remove x from variables
  sort_values(D_{x})
  for each value v in D_{x}{

    extend partial to partial_neu with x <-- v

    if(forward_checking(x, variables, constraints))
      sort_variables(variables)
      if back_solve(partial_neu, variables, constraints)
        return true
  }
  else else
    insert x in variables
    restore_values(variables)
    return false
}
```

Find optimal solutions (Minimization problem)

- there can be several solutions satisfying all constraints
- **requirement:** quality of a solution can be measured by a goal function (e.g. total travel time of a tour)
- the optimal solution can be found with **Branch and Bound**

Optimization

- use a **heuristic function**, that computes a *lower bound on the quality* on the basis of the current *partial solution*
- if the heuristic value of the *partial solution* is greater than the current upper bound → The whole sub-tree under the node is pruned
- if the heuristic value of a *solution* is less than the current upper bound → update and store the newly found upper bound

Convert N-ary constraints to binary constraints

`http:
//ktiml.mff.cuni.cz/~bartak/constraints/binary.html`

Das war's für heute :-)