Constraint Programming

Lecture 1: an overview

Master 1 informatique – Université de Nantes

Laurent Granvilliers

CP problems

Combinatorial problems

CP is a computer programming paradigm for modelling and solving combinatorial problems.

- the variables take values from finite sets of possibilities
- the solutions must satisfy several constraints

CP problems

Combinatorial problems

CP is a computer programming paradigm for modelling and solving combinatorial problems.

- the variables take values from finite sets of possibilities
- the solutions must satisfy several constraints

The number of combinations of values that have to be tested can grow exponentially with the number of variables.

CP approach

Main ideas

- Heterogeneous constraints
  - boolean, integer, symbolic, real
  - logic formulas, constraints on sets
  - high-level constraints, domain-specific constraints

CP approach

Main ideas

- Heterogeneous constraints
  - boolean, integer, symbolic, real
  - logic formulas, constraints on sets
  - high-level constraints, domain-specific constraints
- Generic techniques
  - exhaustive exploration of the set of combinations
  - pruning techniques used to eliminate invalid combinations
  - heuristics

CP approach

Main ideas

- Heterogeneous constraints
  - boolean, integer, symbolic, real
  - logic formulas, constraints on sets
  - high-level constraints, domain-specific constraints
- Generic techniques
  - exhaustive exploration of the set of combinations
  - pruning techniques used to eliminate invalid combinations
  - heuristics
- Complex software systems
  - modelling and computer languages
  - combination of solvers
  - mapping tools, model analysis, symmetry breaking...

An assignment problem

Problem specification

Assign 4 tasks to 5 machines such that

- not every task can be assigned to every machine

Task	Machines
1	B, C, D, E
2	B, C
3	A, B, C, D
4	B, C

An assignment problem

Problem specification

Assign 4 tasks to 5 machines such that

- not every task can be assigned to every machine

Task	Machines
1	B, C, D, E
2	B, C
3	A, B, C, D
4	B, C

- each machine performs at most one task

A binary model

- Rename the machines $A \to 1, B \to 2, \ldots, E \to 5$

A binary model

- Rename the machines $A \to 1, B \to 2, \dots, E \to 5$
- Define binary variables $x_{ij} \in 0..1$ for all (i, j) such that task i can be assigned to machine j

  $x_{ij} \to 1$ means that task i is assigned to machine j

A binary model

- Rename the machines $A \to 1, B \to 2, \ldots, E \to 5$
- Define binary variables $x_{ij} \in 0..1$ for all (i,j) such that task i can be assigned to machine j

  $x_{ij} \to 1$ means that task i is assigned to machine j
- Every task is assigned to one machine

$$\forall i : \sum_j x_{ij} = 1$$

A binary model

- Rename the machines $A \to 1, B \to 2, \ldots, E \to 5$
- Define binary variables $x_{ij} \in 0..1$ for all (i, j) such that task i can be assigned to machine j

  $x_{ij} \to 1$ means that task i is assigned to machine j
- Every task is assigned to one machine

$$\forall i : \sum_j x_{ij} = 1$$

- Every machine performs at most one task

$$\forall j : \sum_i x_{ij} \leq 1$$

A discrete model

- Define discrete variables x_i and domains for all tasks

$$\begin{aligned} x_1 &\in \{B, C, D, E\} \\ x_2 &\in \{B, C\} \\ x_3 &\in \{A, B, C, D\} \\ x_4 &\in \{B, C\} \end{aligned}$$

$x_i \to M$ means that task i is assigned to machine M

A discrete model

- Define discrete variables x_i and domains for all tasks

$$\begin{aligned} x_1 &\in \{B, C, D, E\} \\ x_2 &\in \{B, C\} \\ x_3 &\in \{A, B, C, D\} \\ x_4 &\in \{B, C\} \end{aligned}$$

$x_i \rightarrow M$ means that task i is assigned to machine M

- Every machine performs at most one task

$$\forall\ (1 \leq i < j \leq n) :\ x_i \neq x_j$$

A discrete model

- Define discrete variables x_i and domains for all tasks

$$\begin{aligned} x_1 &\in \{B, C, D, E\} \\ x_2 &\in \{B, C\} \\ x_3 &\in \{A, B, C, D\} \\ x_4 &\in \{B, C\} \end{aligned}$$

$x_i \rightarrow M$ means that task i is assigned to machine M

- Every machine performs at most one task

$$\forall\ (1 \leq i < j \leq n) :\ x_i \neq x_j$$

- Every task must be assigned to one machine

Another discrete model

Modelling issue

The set of difference constraints $x_i \neq x_j$ for all $i \neq j$ means that all the variables must take different values.

Another discrete model

Modelling issue

The set of difference constraints $x_i \neq x_j$ for all $i \neq j$ means that all the variables must take different values.

A global constraint can be introduced.

- Every machine performs at most one task

$$\text{alldifferent}(x), \quad x = (x_1, x_2, x_3, x_4)$$

A graph model

- Define two sets of vertices

$$\mathcal{T} = \{1, 2, 3, 4\} \quad \mathcal{M} = \{A, B, C, D, E\}$$

A graph model

- Define two sets of vertices

$$\mathcal{T} = \{1, 2, 3, 4\} \quad \mathcal{M} = \{A, B, C, D, E\}$$

- Define the set of edges between $\mathcal{T}$ and $\mathcal{M}$

$$\mathcal{E} = \{(i, j) : \text{task } i \text{ can be assigned to machine } j\}$$

A graph model

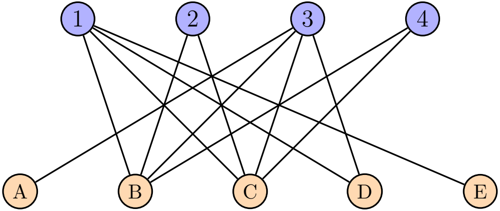
- Define two sets of vertices

$$\mathcal{T} = \{1, 2, 3, 4\} \quad \mathcal{M} = \{A, B, C, D, E\}$$

- Define the set of edges between $\mathcal{T}$ and $\mathcal{M}$

$$\mathcal{E} = \{(i, j) : \text{task } i \text{ can be assigned to machine } j\}$$

- Bipartite graph $(\mathcal{T} \cup \mathcal{M}, \mathcal{E})$



A solution

- Binary model: assignment of the variables

$$\left\{ \begin{array}{l} x_{12} \to 0, x_{13} \to 0, x_{14} \to 1, x_{15} \to 0, x_{22} \to 1, x_{23} \to 0, \\ x_{31} \to 1, x_{32} \to 0, x_{33} \to 0, x_{34} \to 0, x_{42} \to 0, x_{43} \to 1 \end{array} \right\}$$

A solution

- Binary model: assignment of the variables

$$\left\{ \begin{array}{l} x_{12} \to 0, x_{13} \to 0, x_{14} \to 1, x_{15} \to 0, x_{22} \to 1, x_{23} \to 0, \\ x_{31} \to 1, x_{32} \to 0, x_{33} \to 0, x_{34} \to 0, x_{42} \to 0, x_{43} \to 1 \end{array} \right\}$$

- Graph model: matching involving all tasks

$$\{(1, D), (2, B), (3, A), (4, C)\}$$

A solution

- Binary model: assignment of the variables

$$\left\{ \begin{array}{l} x_{12} \to 0, x_{13} \to 0, x_{14} \to 1, x_{15} \to 0, x_{22} \to 1, x_{23} \to 0, \\ x_{31} \to 1, x_{32} \to 0, x_{33} \to 0, x_{34} \to 0, x_{42} \to 0, x_{43} \to 1 \end{array} \right\}$$

- Graph model: matching involving all tasks

$$\{(1, D), (2, B), (3, A), (4, C)\}$$

- Discrete model: assignment of the variables

$$\{x_1 \to D, x_2 \to B, x_3 \to A, x_4 \to C\}$$

Variable

Variable

Given a set D called a domain, a variable $x \in D$ is a symbol representing an unknown element of D.

Variable

Variable

Given a set D called a domain, a variable $x \in D$ is a symbol representing an unknown element of D.

Assignment of variables

An assignment of variables $x_1 \in D_1, \ldots, x_n \in D_n$ is a mapping

$$\sigma = \{x_1 \to d_1, \ldots, x_n \to d_n\}$$

from variables to values $d_1 \in D_1, \ldots, d_n \in D_n$.

Search space

- Given variables $x_1 \in D_1, \ldots, x_n \in D_n$ the search space is the set of possible asssignments

$$S = D_1 \times \cdots \times D_n = \{(d_1, \ldots, d_n) : d_1 \in D_1, \ldots, d_n \in D_n\}$$

Search space

- Given variables $x_1 \in D_1, \ldots, x_n \in D_n$ the search space is the set of possible asssignments

$$S = D_1 \times \cdots \times D_n = \{(d_1, \ldots, d_n) : d_1 \in D_1, \ldots, d_n \in D_n\}$$

- Given discrete domains the size of the search space is

$$|S| = |D_1| \times \cdots \times |D_n|$$

Search space

- Given variables $x_1 \in D_1, \ldots, x_n \in D_n$ the search space is the set of possible asssignments

$$S = D_1 \times \cdots \times D_n = \{(d_1, \ldots, d_n) : d_1 \in D_1, \ldots, d_n \in D_n\}$$

- Given discrete domains the size of the search space is

$$|S| = |D_1| \times \cdots \times |D_n|$$

- Models of the assignment problem
  - binary: $|S| = 2^{12} = 4096$
  - discrete: $|S| = 4 \times 2 \times 4 \times 2 = 2^6 = 64$

Constraint

Constraint as formula

A constraint may be defined as a formula in a given theory.

$$\begin{aligned} c_1(x_1, x_2, x_4) : &\quad 2x_1 + x_2 - x_4 \leq 1 \\ c_2(x_3) : &\quad \cos^2 x_3 + \sin^2 x_3 = 1 \end{aligned}$$

Constraint

Constraint as formula

A constraint may be defined as a formula in a given theory.

$$
\begin{aligned}
c_1(x_1, x_2, x_4): &\quad 2x_1 + x_2 - x_4 \leq 1 \\
c_2(x_3): &\quad \cos^2 x_3 + \sin^2 x_3 = 1
\end{aligned}
$$

The scope of a constraint c is its set of variables X_c.

Constraint

Constraint as formula

A constraint may be defined as a formula in a given theory.

$$\begin{aligned} c_1(x_1, x_2, x_4): &\quad 2x_1 + x_2 - x_4 \leq 1 \\ c_2(x_3): &\quad \cos^2 x_3 + \sin^2 x_3 = 1 \end{aligned}$$

The scope of a constraint c is its set of variables X_c.

Constraint as relation

A constraint c defines a relation R_c included in the Cartesian product of domains of the variables from X_c.

Example

Given $x_1 \in \{B, C, D, E\}$ and $x_2 \in \{B, C\}$ the constraint $c : x_1 \neq x_2$ defines the relation

$$R_c = \{(B,C),(C,B),(D,B),(D,C),(E,B),(E,C)\} \subseteq D_1 \times D_2$$

x_1	x_2
B	C
C	B
D	B
D	C
E	B
E	C

Fundamental remarks

- A solution is an assignment of values to variables such that the constraint is satisfied when the variables are replaced by their values or the tuple of values belongs to the relation.

$$\{x_1 \to D, x_2 \to C\}$$

Fundamental remarks

- A solution is an assignment of values to variables such that the constraint is satisfied when the variables are replaced by their values or the tuple of values belongs to the relation.

$$\{x_1 \to D, x_2 \to C\}$$

- A constraint restricts the admissible variable values.

$$\{x_1 \to B, x_2 \to B\} \text{ is not a solution}$$

Fundamental remarks

- A solution is an assignment of values to variables such that the constraint is satisfied when the variables are replaced by their values or the tuple of values belongs to the relation.

$$\{x_1 \to D, x_2 \to C\}$$

- A constraint restricts the admissible variable values.

$$\{x_1 \to B, x_2 \to B\} \text{ is not a solution}$$

- A constraint can be used to derive new restrictions.

$$\frac{x_1 \neq x_2,\ x_2 = B}{x_1 \neq B}$$

Constraint system

Modelling a problem leads in general to several constraints involving the same variables that represent problem properties.

Constraint system

Modelling a problem leads in general to several constraints involving the same variables that represent problem properties.

Constraint system

A constraint system is a formula from first-order logic.

- a constraint is an atomic formula
- connectives $\wedge, \vee, \implies, \iff, \neg$
- quantifiers $\forall, \exists$

Constraint system

Modelling a problem leads in general to several constraints involving the same variables that represent problem properties.

Constraint system

A constraint system is a formula from first-order logic.

- a constraint is an atomic formula
- connectives $\wedge, \vee, \implies, \iff, \neg$
- quantifiers $\forall, \exists$

The atomic constraints may be defined over different theories such as **boolean** constraints, **integer** constraints, **real** constraints...

Constraint satisfaction problem

Constraint satisfaction problem

A CSP is a quantifier-free conjunction of constraints

$$c_1 \wedge \cdots \wedge c_m,\ x_1 \in D_1, \ldots, x_n \in D_n$$

denoted by the triple $P = \langle C, X, D \rangle$ where

- $X = \{x_1, \ldots, x_n\}$ is a set of variables,
- $D = \{D_1, \ldots, D_n\}$ is a set of domains,
- $C = \{c_1, \ldots, c_m\}$ is a set of constraints.

Solution to a CSP

Solution

A solution to $P = \langle C, X, D \rangle$ is a total assignment of the variables $\sigma = \{x_1 \to d_1, \ldots, x_n \to d_n\}$ such that each constraint $c \in C$ is satisfied.

Solution to a CSP

Solution

A solution to $P = \langle C, X, D \rangle$ is a total assignment of the variables $\sigma = \{x_1 \to d_1, \ldots, x_n \to d_n\}$ such that each constraint $c \in C$ is satisfied.

Given the scope $X_c = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$, we have

$$(d_{i_1}, \ldots, d_{i_k}) \in R_c.$$

Class of complexity

Decision problem

The decision problem associated to a CSP $\langle C, X, D\rangle$ is defined as

$$\exists x_1 \in D_1, \ldots, \exists x_n \in D_n : c_1 \wedge \cdots \wedge c_m$$

Class of complexity

Decision problem

The decision problem associated to a CSP $\langle C, X, D \rangle$ is defined as

$$\exists x_1 \in D_1, \ldots, \exists x_n \in D_n : c_1 \wedge \cdots \wedge c_m$$

On finite domains

The decision problem is NP-complete.

Class of complexity

Decision problem

The decision problem associated to a CSP $\langle C, X, D\rangle$ is defined as

$$\exists x_1 \in D_1, \ldots, \exists x_n \in D_n : c_1 \wedge \cdots \wedge c_m$$

On finite domains

The decision problem is NP-complete.

On continuous domains

The decision problem is in general undecidable.

So what can we do?

- No known algorithm solves these problems in polynomial time.

  Given an algorithm that solves a problem in this class, there are some instances for which the algorithm will take an exponential number of steps in the problem size.

So what can we do?

- No known algorithm solves these problems in polynomial time.

  Given an algorithm that solves a problem in this class, there are some instances for which the algorithm will take an exponential number of steps in the problem size.

- NP-complete problems have the property that a solution can be verified in polynomial time.

So what can we do?

- No known algorithm solves these problems in polynomial time.

  Given an algorithm that solves a problem in this class, there are some instances for which the algorithm will take an exponential number of steps in the problem size.

- NP-complete problems have the property that a solution can be verified in polynomial time.

- The goal is to develop algorithms that work well for a wide range of problems.

Exploring the search space

Complete solving

A solver is complete if it is able to find all the solutions or to prove that there is no solution.

- linear programming with rational numbers (polynomial)
- unification for equations on terms (polynomial)
- backtracking search for integer constraints (exponential)

Exploring the search space

Complete solving

A solver is complete if it is able to find all the solutions or to prove that there is no solution.

- linear programming with rational numbers (polynomial)
- unification for equations on terms (polynomial)
- backtracking search for integer constraints (exponential)

Incomplete solving

A solver is incomplete if it is not complete.

- constraint propagation
- local search
- numerical methods

Solving finite domain CSP

We will focus now on CSPs with finite domains.

Solving finite domain CSP

We will focus now on CSPs with finite domains.

Complete solving

- generate and test
- backtracking search
- pruning techniques

Solving finite domain CSP

We will focus now on CSPs with finite domains.

Complete solving

- generate and test
- backtracking search
- pruning techniques

Incomplete solving

- stochastic local search

Assignment and conflict

Assignment

Given a set of variables X, an assignment is total if all the variables from X are assigned. Otherwise, it is partial.

Assignment and conflict

Assignment

Given a set of variables X, an assignment is total if all the variables from X are assigned. Otherwise, it is partial.

Conflict

A conflict is an assignment violating a constraint.

Assignment and conflict

Assignment

Given a set of variables X, an assignment is total if all the variables from X are assigned. Otherwise, it is partial.

Conflict

A conflict is an assignment violating a constraint.

An early identification of conflicts leads to accelerate the solving process by pruning the search space.

Generate and test

```
GenTest(P:CSP)
begin
   GenTestRec(P,0,{})
end

GenTestRec(P:CSP, i:int, A:assignment)
begin
   if i=n and IsSolution(C,A) then
      label A as a solution
   else
      foreach d in D(i+1) do
         GenTestRec(P,i+1,A ∪ {x(i+1) → d})
      endfor
   endif
end
```

```
IsSolution(C:constraint set, A:assignment) return bool
var i:int, sat:bool
begin
    i := 1
    sat := true
    while ((i<=n) and (sat)) do
        if not (c(i) is satisfied by A)
        then
            sat := false
        else
            i := i+1
        endif
    endwhile
    return sat
end
```

Remarks

- The number of tests is equal to the size of the search space.

Remarks

- The number of tests is equal to the size of the search space.
- The search does not depend on the ordering of the variables.

Remarks

- The number of tests is equal to the size of the search space.
- The search does not depend on the ordering of the variables.
- Conflicts are not identified as soon as possible.

Backtracking

```
Backtraking(P:CSP)
begin
    BacktrakingRec(P,0,{})
end

BacktrakingRec(P:CSP, i:int, A:assignment)
begin
    if i=n then
        label A as a solution
    else
        foreach d in D(i+1) do
            if IsSat(P,A ∪ {x(i+1) → d}) then
                BacktrakingRec(P,i+1,A ∪ {x(i+1) → d})
            endif
        endfor
    endif
end
```

```
IsSat(C:constraint set, A:assignment) return bool
var i:int, sat:bool
begin
   i := 1
   sat := true
   while ((i<=n) and (sat)) do
      if A assigns all the variables of c(i) and
         not (c(i) is satisfied by A)
      then
         sat := false
      else
         i := i+1
      endif
   endwhile
   return sat
end
```

Example

Solve the assignment problem with domains

$$x_1 \in \{B, C, D, E\}, x_2 \in \{B, C\}, x_3 \in \{A, B, C, D\}, x_4 \in \{B, C\}$$

and constraints

$$(\forall\ 1 \leq i < j \leq 4) : x_i \neq x_j$$

by backtracking search considering the following variable orderings:

1. $x_1 < x_2 < x_3 < x_4$
2. $x_2 < x_4 < x_1 < x_3$

Remarks

- The worst-case complexity remains exponential.

Remarks

- The worst-case complexity remains exponential.
- Conflicts are identified as soon as possible.

Remarks

- The worst-case complexity remains exponential.
- Conflicts are identified as soon as possible.
- The search depends on the ordering of the variables.

Variable selection

Heuristics

The next variable to be assigned in a search node can be chosen according to the several heuristics.

Variable selection

Heuristics

The next variable to be assigned in a search node can be chosen according to the several heuristics.

- **MinDomain**: the variable whose domain has the least number of values (hence limiting the arity of the search tree)

Variable selection

Heuristics

The next variable to be assigned in a search node can be chosen according to the several heuristics.

- **MinDomain**: the variable whose domain has the least number of values (hence limiting the arity of the search tree)
- **MaxConstrained**: the variable occurring the most in the constraints (hence increasing the early detection of conflicts)

Variable selection

Heuristics

The next variable to be assigned in a search node can be chosen according to the several heuristics.

- **MinDomain**: the variable whose domain has the least number of values (hence limiting the arity of the search tree)
- **MaxConstrained**: the variable occurring the most in the constraints (hence increasing the early detection of conflicts)
- **Hybrid**: the variable occurring the most whose domain is the smallest one

Pruning the search tree

Remark

Assigning one variable can lead to prune domains of the non assigned variables.

Pruning the search tree

Remark

Assigning one variable can lead to prune domains of the non assigned variables.

Consider the assignment problem and let $\{x_1 \to B\}$. Then

1. B can be removed from the domain of x_2 since $x_1 \neq x_2$

Pruning the search tree

Remark

Assigning one variable can lead to prune domains of the non assigned variables.

Consider the assignment problem and let $\{x_1 \to B\}$. Then

1. B can be removed from the domain of x_2 since $x_1 \neq x_2$
2. B can be removed from the domain of x_3 since $x_1 \neq x_3$

Pruning the search tree

Remark

Assigning one variable can lead to prune domains of the non assigned variables.

Consider the assignment problem and let $\{x_1 \to B\}$. Then

1. B can be removed from the domain of x_2 since $x_1 \neq x_2$
2. B can be removed from the domain of x_3 since $x_1 \neq x_3$
3. B can be removed from the domain of x_4 since $x_1 \neq x_4$

Pruning the search tree

Remark

Assigning one variable can lead to prune domains of the non assigned variables.

Consider the assignment problem and let $\{x_1 \to B\}$. Then

1. B can be removed from the domain of x_2 since $x_1 \neq x_2$
2. B can be removed from the domain of x_3 since $x_1 \neq x_3$
3. B can be removed from the domain of x_4 since $x_1 \neq x_4$

   Now the domain of x_2 is reduced to $\{C\}$ (x_2 is assigned)

Pruning the search tree

Remark

Assigning one variable can lead to prune domains of the non assigned variables.

Consider the assignment problem and let $\{x_1 \to B\}$. Then

1. B can be removed from the domain of x_2 since $x_1 \neq x_2$
2. B can be removed from the domain of x_3 since $x_1 \neq x_3$
3. B can be removed from the domain of x_4 since $x_1 \neq x_4$

   Now the domain of x_2 is reduced to $\{C\}$ (x_2 is assigned)
4. C can be removed from the domain of x_3 since $x_2 \neq x_3$

Pruning the search tree

Remark

Assigning one variable can lead to prune domains of the non assigned variables.

Consider the assignment problem and let $\{x_1 \to B\}$. Then

1. B can be removed from the domain of x_2 since $x_1 \neq x_2$
2. B can be removed from the domain of x_3 since $x_1 \neq x_3$
3. B can be removed from the domain of x_4 since $x_1 \neq x_4$

   Now the domain of x_2 is reduced to $\{C\}$ (x_2 is assigned)
4. C can be removed from the domain of x_3 since $x_2 \neq x_3$
5. C can be removed from the domain of x_4 since $x_2 \neq x_4$

Pruning the search tree

Remark

Assigning one variable can lead to prune domains of the non assigned variables.

Consider the assignment problem and let $\{x_1 \to B\}$. Then

1. B can be removed from the domain of x_2 since $x_1 \neq x_2$
2. B can be removed from the domain of x_3 since $x_1 \neq x_3$
3. B can be removed from the domain of x_4 since $x_1 \neq x_4$

   Now the domain of x_2 is reduced to $\{C\}$ (x_2 is assigned)
4. C can be removed from the domain of x_3 since $x_2 \neq x_3$
5. C can be removed from the domain of x_4 since $x_2 \neq x_4$
6. failure!

Pruning operator

Pruning operator

A pruning operator θ for a constraint c is a correct and contracting operator that verifies

$$D \cap R_c \subseteq \theta(D) \subseteq D$$

for any variable domains D.

Branch and prune

```
BranchAndPrune(P:CSP)
begin
   BranchAndPruneRec(P,D)
end

BranchAndPruneRec(P:CSP, E:domains)
begin
   E' := Prune(P,E)
   if E' is empty then
      fail
   else if E' is a solution then
      label E' as a solution
   else
      foreach E'' in Branch(E') do
         BranchAndPruneRec(P,E'')
      endfor
   endif
end
```

- The Prune procedure applies pruning operators in order to contract the current domains.

  $\rightarrow$ there are several strategies!

- The Prune procedure applies pruning operators in order to contract the current domains.

  $\rightarrow$ there are several strategies!

- The Branch procedure creates sub-nodes in the search tree.

  $\rightarrow$ there are several strategies!

Forward checking

- Forward checking is a branch and prune algorithm.

Forward checking

- Forward checking is a branch and prune algorithm.
- The pruning operator performs a limited form of propagation. Let x_i be the last assigned variable at a given node and let C_i be the set of constraints depending on x_i.

Forward checking

- Forward checking is a branch and prune algorithm.
- The pruning operator performs a limited form of propagation. Let x_i be the last assigned variable at a given node and let C_i be the set of constraints depending on x_i.
  1. the satisfaction of each constraint $c \in C_i$ whose variables are all assigned is checked and a failure happens if at least one constraint is violated

Forward checking

- Forward checking is a branch and prune algorithm.
- The pruning operator performs a limited form of propagation. Let x_i be the last assigned variable at a given node and let C_i be the set of constraints depending on x_i.
  1. the satisfaction of each constraint $c \in C_i$ whose variables are all assigned is checked and a failure happens if at least one constraint is violated
  2. for each constraint $c \in C_i$ containing only one variable x_j that is not yet assigned, every value $a_j \in E_j$ from the current domain of x_j that is not consistent with respect to c is removed from D_j

Forward checking

- Forward checking is a branch and prune algorithm.
- The pruning operator performs a limited form of propagation. Let x_i be the last assigned variable at a given node and let C_i be the set of constraints depending on x_i.
  1. the satisfaction of each constraint $c \in C_i$ whose variables are all assigned is checked and a failure happens if at least one constraint is violated
  2. for each constraint $c \in C_i$ containing only one variable x_j that is not yet assigned, every value $a_j \in E_j$ from the current domain of x_j that is not consistent with respect to c is removed from D_j
  3. if D_j becomes empty then a failure happens and the algorithm backtracks

Example

Solve the assignment problem with domains

$$x_1 \in \{B, C, D, E\}, x_2 \in \{B, C\}, x_3 \in \{A, B, C, D\}, x_4 \in \{B, C\}$$

and constraints

$$(\forall\ 1 \leq i < j \leq 4) : x_i \neq x_j$$

by forward checking.

Partial look ahead

- Partial look ahead is stronger than forward checking in the sense that the pruning technique is iterated when some domain becomes reduced to one value (a new variable is assigned).

Partial look ahead

- Partial look ahead is stronger than forward checking in the sense that the pruning technique is iterated when some domain becomes reduced to one value (a new variable is assigned).
- Solve the assignment problem by partial look ahead.

What next?

Goal

Find a balance between branching and pruning in order to accelerate the overall solving process

What next?

Goal

Find a balance between branching and pruning in order to accelerate the overall solving process

Techniques

- Fixed-point pruning algorithms (constraint propagation)

What next?

Goal

Find a balance between branching and pruning in order to accelerate the overall solving process

Techniques

- Fixed-point pruning algorithms (constraint propagation)
- Consistency techniques

What next?

Goal

Find a balance between branching and pruning in order to accelerate the overall solving process

Techniques

- Fixed-point pruning algorithms (constraint propagation)
- Consistency techniques
- Domain-specific pruning operators

Stochastic local search

Main ideas

- A local search process follows a path from an initial total assignment of the variables to a solution or a dead-end.

Stochastic local search

Main ideas

- A local search process follows a path from an initial total assignment of the variables to a solution or a dead-end.
- At each step, a neighbourhood of the current assignment is explored.

Stochastic local search

Main ideas

- A local search process follows a path from an initial total assignment of the variables to a solution or a dead-end.
- At each step, a neighbourhood of the current assignment is explored.
- A move consists in selecting the next assignment in the neighbourhood in order to come closer to a solution.

Initial assignment

- The initial total assignment is generated randomly.
- Example for the assignment problem

$$\sigma_0 = \{x_1 \to C, x_2 \to B, x_3 \to D, x_4 \to C\}$$

Neighbourhood

Informal definition

A neighbourhood is a set of assignments not differing much from the current assignment, which must have a limited size.

Neighbourhood

Informal definition

A neighbourhood is a set of assignments not differing much from the current assignment, which must have a limited size.

- Example: generate the set of assignments obtained from the current assignment by changing the value of one variable considering all the other values from its domain

$$
\begin{aligned}
N(\sigma_0) \;=\; \{ \;\; & BBDC, DBDC, EBDC, \\
& CCDC, \\
& CBAC, CBBC, CBCC, \\
& CBDB \quad \}
\end{aligned}
$$

Cost function

Cost function

A cost function is a function that associates a natural number to any assignment such that a solution has cost 0.

Cost function

Cost function

A cost function is a function that associates a natural number to any assignment such that a solution has cost 0.

- Example: count the number of violated constraints

$$
\begin{aligned}
N(\sigma_0) \quad = \quad \{ \quad & BBDC:1, DBDC:1, EBDC:0, \\
& CCDC:3, \\
& CBAC:1, CBBC:2, CBCC:3, \\
& CBDB:1 \quad \}
\end{aligned}
$$

Move

Move

A move selects an assignment from the neighbourhood that decreases the cost function.

- If a candidate assignment has cost 0 then a solution is found and the algorithm stops

$$\sigma_1 = \{x_1 \to E, x_2 \to B, x_3 \to D, x_4 \to C\}$$

Move

Move

A move selects an assignment from the neighbourhood that decreases the cost function.

- If a candidate assignment has cost 0 then a solution is found and the algorithm stops

$$\sigma_1 = \{x_1 \to E, x_2 \to B, x_3 \to D, x_4 \to C\}$$

- Otherwise, a candidate with the smallest cost is selected
  - random choice is there are many candidates

Move

Move

A move selects an assignment from the neighbourhood that decreases the cost function.

- If a candidate assignment has cost 0 then a solution is found and the algorithm stops

$$\sigma_1 = \{x_1 \rightarrow E, x_2 \rightarrow B, x_3 \rightarrow D, x_4 \rightarrow C\}$$

- Otherwise, a candidate with the smallest cost is selected
  - random choice is there are many candidates
  - problem if no candidate decreases the cost (local optimum)

How to handle local optima?

Plateau search

Continue with assignments of equal cost

How to handle local optima?

Plateau search

Continue with assignments of equal cost

Tabu search

Mark the last p moves (tabu list) and avoid these moves

How to handle local optima?

Plateau search

Continue with assignments of equal cost

Tabu search

Mark the last p moves (tabu list) and avoid these moves

Constraint weighting

Given weights w_j associated to the constraints c_j let the cost function be defined as $\mathrm{cost}(\sigma) = \sum_j w_j \times b_j$ where $b_j = 1$ if σ violates c_j and $b_j = 0$ otherwise, such that w_j is initialized to 1 for all j, and it is incremented at a local optimum if c_j is violated

Skeleton of algorithm

```
LocalSearch(P:CSP, MaxRestart:int, MaxMove:int)
var found:bool
    restart:int
begin
   found := false
   restart := 0
   while ((not found) and (restart<MaxRestart)) do
      found := LocalSearchRun(P,MaxMove)
      restart := restart+1
   endwhile
end
```

```
LocalSearchRun(P:CSP, MaxMove:int) return bool
var iter:bool, found:bool
    move:int
    sol:assignment
begin
   iter := true
   found := false
   move := 0
   sol := GenerateRandom(D)
   while ((iter) and (move<MaxMove)) do
      if IsSolution(C,sol) then
         found := true
         iter := false
      else
         (sol,iter) := DoAmove(P,sol)
      endif
      move := move+1
   endwhile
   return found
end
```

Complete or not complete?

- Complete if the goal is to find all the solutions

Complete or not complete?

- Complete if the goal is to find all the solutions
- Complete if the goal is to prove that there is no solution

Complete or not complete?

- Complete if the goal is to find all the solutions
- Complete if the goal is to prove that there is no solution
- If the goal is to find only one solution
  - not complete should be tried (polynomial algorithm)

Complete or not complete?

- Complete if the goal is to find all the solutions
- Complete if the goal is to prove that there is no solution
- If the goal is to find only one solution
  - not complete should be tried (polynomial algorithm)
  - but complete may works well for some problems with few solutions and constraint systems that are hard to solve