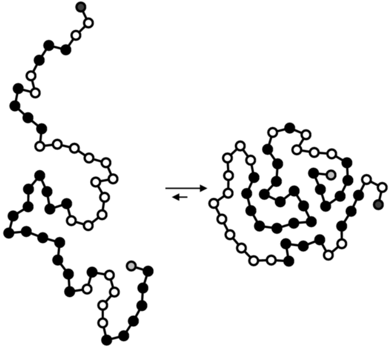
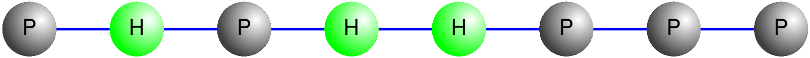
Constraint Programming and Protein Structure Prediction

Can we predict protein structure?

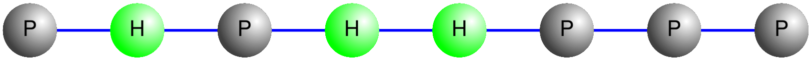


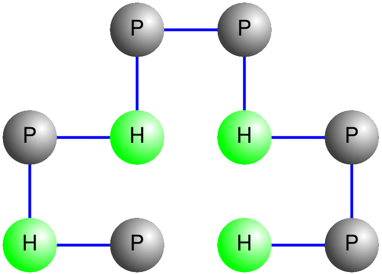
- Molecular Dynamics on Full Atom Models
- Simpler Protein Models:
  - Folding simulation
  - Stochastic optimization, e.g. Genetic Algorithms
  - Combinatorial optimization, e.g. Constraint Programming

Simple Proteins: HP-Model

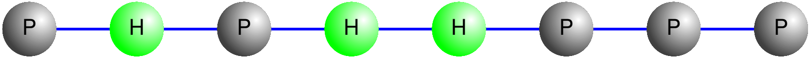


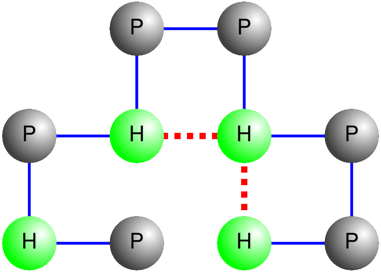
Simple Proteins: HP-Model





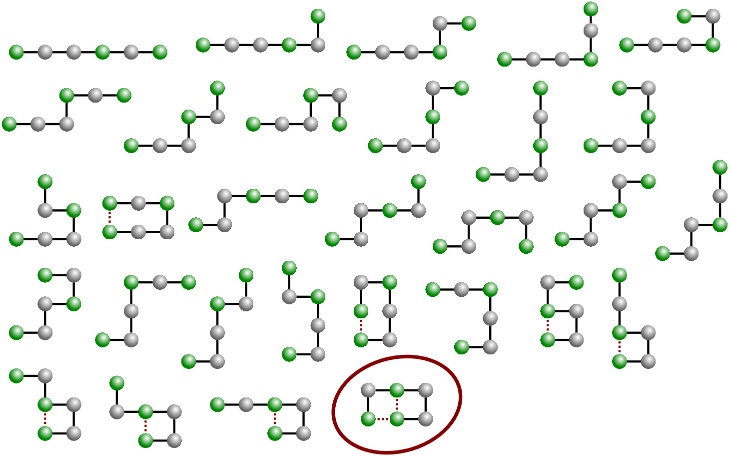
Simple Proteins: HP-Model





Structures in the HP-Model

Sequence HPPHPH



Constraint Programming

Constraint programming ...

- ... is a programming technique
- ... describes what rather than how
- ... i.e. it is declarative
- ... combines logic reasoning with search
- ... performs "intelligent" enumeration
- ... "slays NP-hard dragons"

Well, But What Are Constraints?

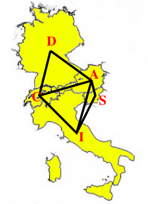
Example: Map Coloring

Constraints:
$A, C, D, I, S \in \{red, green, blue\}$,
$A \neq C$, $A \neq D$, $A \neq I$, $A \neq S$,
$C \neq D$, $C \neq I$, $I \neq S$

- We say only what a solution of the map coloring is
- We need not care how the problem is solved
- A solution is computed by guessing and reasoning
  E.g. guess $A =$ red implies $C, D, I, S \neq$ red;
  then guess $C =$ green …

Well, But What Are Constraints?

Example: Map Coloring

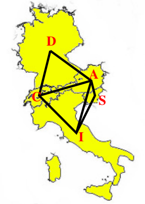


Constraints:
$A, C, D, I, S \in \{red, green, blue\}$,
$A \neq C$, $A \neq D$, $A \neq I$, $A \neq S$,
$C \neq D$, $C \neq I$, $I \neq S$

- We say only what a solution of the map coloring is
- We need not care how the problem is solved
- A solution is computed by guessing and reasoning
  E.g. guess $A =$ red implies $C, D, I, S \neq$ red;
  then guess $C =$ green ...

Well, But What Are Constraints?

Example: Map Coloring

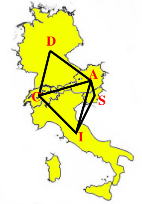


Constraints:
$A, C, D, I, S \in \{red, green, blue\}$,
$A \neq C$, $A \neq D$, $A \neq I$, $A \neq S$,
$C \neq D$, $C \neq I$, $I \neq S$

- We say only what a solution of the map coloring is
- We need not care how the problem is solved
- A solution is computed by guessing and reasoning
  E.g. guess $A =$ red implies $C, D, I, S \neq$ red;
  then guess $C =$ green . . .

Well, But What Are Constraints?

Example: Map Coloring

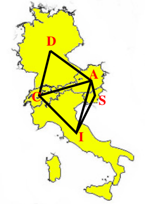


Constraints:
$A, C, D, I, S \in \{red, green, blue\}$,
$A \neq C$, $A \neq D$, $A \neq I$, $A \neq S$,
$C \neq D$, $C \neq I$, $I \neq S$

- We say only what a solution of the map coloring is
- We need not care how the problem is solved
- A solution is computed by guessing and reasoning
  E.g. guess $A =$ red implies $C, D, I, S \neq$ red;
  then guess $C =$ green . . .

Well, But What Are Constraints?

Example: Map Coloring

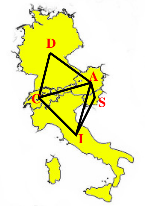


Constraints:
$A, C, D, I, S \in \{red, green, blue\}$,
$A \neq C$, $A \neq D$, $A \neq I$, $A \neq S$,
$C \neq D$, $C \neq I$, $I \neq S$

- We say only what a solution of the map coloring is
- We need not care how the problem is solved
- A solution is computed by guessing and reasoning
  E.g. guess $A =$ red implies $C, D, I, S \neq$ red;
  then guess $C =$ green ...

Well, But What Are Constraints?

Example: Map Coloring



Constraints:
$A, C, D, I, S \in \{red, green, blue\}$,
$A \neq C$, $A \neq D$, $A \neq I$, $A \neq S$,
$C \neq D$, $C \neq I$, $I \neq S$

- We say only what a solution of the map coloring is
- We need not care how the problem is solved
- A solution is computed by guessing and reasoning
  E.g. guess $A =$ red implies $C, D, I, S \neq$ red;
  then guess $C =$ green ...

Another Constraints Example

Example

A mathematician forgot the last position of a number code.
She only remembers

- it's odd
- of course, its a digit, i.e. in [0..9]
- it's no prime number and not 1.

She can derive the digit (by constraint reasoning)!

Another Constraints Example

Example

A mathematician forgot the last position of a number code.
She only remembers

- it's odd
- of course, its a digit, i.e. in [0..9]
- it's no prime number and not 1.

She can derive the digit (by constraint reasoning)!

Commercial Impact of Constraints

Some examples

Michelin and Dassault, Renault	Production planning
Lufthansa, Swiss Air, ...	Staff planning
Nokia	Software configuration
Siemens	Circuit verification
French National Railway Company	Train schedule

Constraint Satisfaction Problem (CSP)

Definition

A Constraint Satisfaction Problem (CSP) consists of

- *variables* $\mathcal{X} = \{X_1, \dots, X_n\}$,
- the domain D that associates finite domains $D_1 = D(X_1), \dots, D_n = D(X_n)$ to $\mathcal{X}$.
- a set of constraints C.

A solution is an assignment of variables to values of their domains that satisfies the constraints.

Constraint Satisfaction Problem (CSP)

Definition

A Constraint Satisfaction Problem (CSP) consists of

- *variables* $\mathcal{X} = \{X_1, \ldots, X_n\}$,
- the domain D that associates finite domains $D_1 = D(X_1), \ldots, D_n = D(X_n)$ to $\mathcal{X}$.
- a set of constraints C.

A solution is an assignment of variables to values of their domains that satisfies the constraints.

Constraint Satisfaction Problem (CSP)

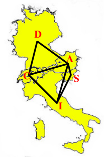
Definition

A Constraint Satisfaction Problem (CSP) consists of

- *variables* $\mathcal{X} = \{X_1, \dots, X_n\}$,
- the domain D that associates finite domains $D_1 = D(X_1), \dots, D_n = D(X_n)$ to $\mathcal{X}$.
- a set of constraints C.

A solution is an assignment of variables to values of their domains that satisfies the constraints.

We have already seen one example: map coloring.



A Simple Example CSP

- Variables $\mathcal{X} = \{X, Y, Z\}$
- Domains $D(X) = D(Y) = D(Z) = \{1, 2, 3, 4\}$
- Constraints $C = \{X < Y, Y < Z, Z \leq 2\}$

Remarks

- The constraint set is interpreted as the conjunction

$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2.$$

- The domains are interpreted as the constraint

$$X \in D(X) \text{ and } Y \in D(Y) \text{ and } Z \in D(Z).$$

A Simple Example CSP

- Variables $\mathcal{X} = \{X, Y, Z\}$
- Domains $D(X) = D(Y) = D(Z) = \{1, 2, 3, 4\}$
- Constraints $C = \{X < Y, Y < Z, Z \leq 2\}$

Remarks

- The constraint set is interpreted as the conjunction

$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2.$$

- The domains are interpreted as the constraint

$$X \in D(X) \text{ and } Y \in D(Y) \text{ and } Z \in D(Z).$$

A Simple Example CSP

- Variables $\mathcal{X} = \{X, Y, Z\}$
- Domains $D(X) = D(Y) = D(Z) = \{1, 2, 3, 4\}$
- Constraints $C = \{X < Y, Y < Z, Z \leq 2\}$

Remarks

- The constraint set is interpreted as the conjunction

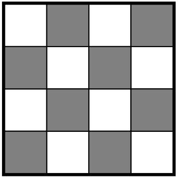
$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2.$$

- The domains are interpreted as the constraint

$$X \in D(X) \text{ and } Y \in D(Y) \text{ and } Z \in D(Z).$$

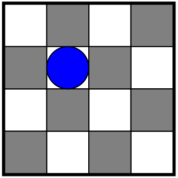
The N-Queens Problem

4-Queens: place 4 queens on 4×4 board without attacks



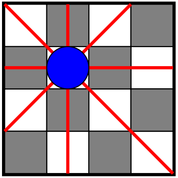
The N-Queens Problem

4-Queens: place 4 queens on 4×4 board without attacks



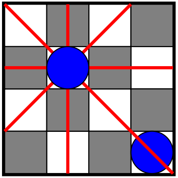
The N-Queens Problem

4-Queens: place 4 queens on 4×4 board without attacks



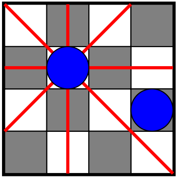
The N-Queens Problem

4-Queens: place 4 queens on 4×4 board without attacks



The N-Queens Problem

4-Queens: place 4 queens on 4×4 board without attacks



The N-Queens Problem

4-Queens: place 4 queens on 4×4 board without attacks

Model 4-Queens as CSP (Constraint Model)

- Variables
$$X_1, \dots, X_4$$
$X_i = j$ means "queen in column i, row j"

- Domains $D(X_i) = \{1, \dots, 4\}$ for $i = 1..4$

- Constraints (for different columns i and i')

no horizontal attack $(X_i \neq X_{i'})$

no attack in first diagonal $(i - X_i \neq i' - X_{i'})$

no attack in second diagonal $(i + X_i \neq i' + X_{i'})$

The N-Queens Problem

4-Queens: place 4 queens on 4×4 board without attacks

Model 4-Queens as CSP (Constraint Model)

- Variables
$$X_1, \dots, X_4$$
$X_i = j$ means "queen in column i, row j"

- Domains $D(X_i) = \{1, \dots, 4\}$ for $i = 1..4$

- Constraints (for different columns i and i')

  no horizontal attack $(X_i \neq X_{i'})$

  no attack in first diagonal $(i - X_i \neq i' - X_{i'})$

  no attack in second diagonal $(i + X_i \neq i' + X_{i'})$

The N-Queens Problem

4-Queens: place 4 queens on 4×4 board without attacks

Model 4-Queens as CSP (Constraint Model)

- Variables

$$X_1, \dots, X_4$$

$X_i = j$ means "queen in column i, row j"

- Domains $D(X_i) = \{1, \dots, 4\}$ for $i = 1..4$

- Constraints (for different columns i and i')

no horizontal attack $(X_i \neq X_{i'})$

no attack in first diagonal $(i - X_i \neq i' - X_{i'})$

no attack in second diagonal $(i + X_i \neq i' + X_{i'})$

The N-Queens Problem

4-Queens: place 4 queens on 4×4 board without attacks

Model 4-Queens as CSP (Constraint Model)

- Variables
$$X_1, \dots, X_4$$
$X_i = j$ means "queen in column i, row j"

- Domains $D(X_i) = \{1, \dots, 4\}$ for $i = 1..4$

- Constraints (for different columns i and i')

  no horizontal attack $(X_i \neq X_{i'})$

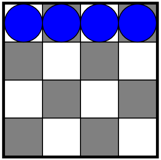
  no attack in first diagonal $(i - X_i \neq i' - X_{i'})$

  no attack in second diagonal $(i + X_i \neq i' + X_{i'})$

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 1$$

inconsistent!

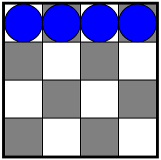
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 1$$

inconsistent!

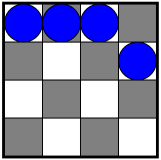
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 2$$

inconsistent!

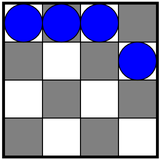
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 2$$

inconsistent!

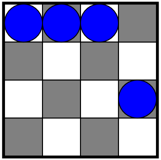
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 3$$

inconsistent!

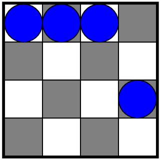
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 3$$

inconsistent!

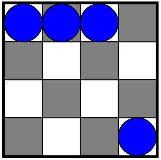
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 4$$

inconsistent!

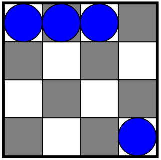
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 4$$

inconsistent!

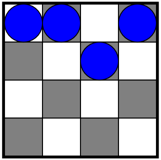
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 2, X_4 = 1$$

inconsistent!

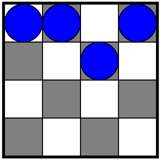
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 2, X_4 = 1$$

inconsistent! ...it's getting boring.

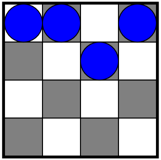
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 2, X_4 = 1$$

inconsistent!

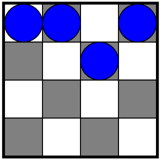
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 2, X_4 = 1$$

inconsistent!

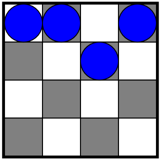
What's wrong with GT?

- Redundancy
- Inconsistency local!

Solving the CSP

Generate and Test

generate assignments and test each



$$X_1 = 1, X_2 = 1, X_3 = 2, X_4 = 1$$

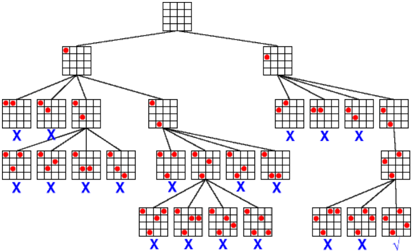
inconsistent!

What's wrong with GT?

- Redundancy
- Inconsistency local!

Overcoming GT's weakness

Backtracking

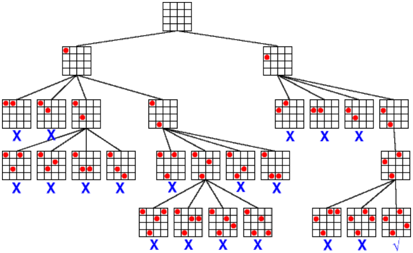


Problems

- Thrashing
- Redundancy
- Late Detection of Inconsistency

Overcoming GT's weakness

Backtracking

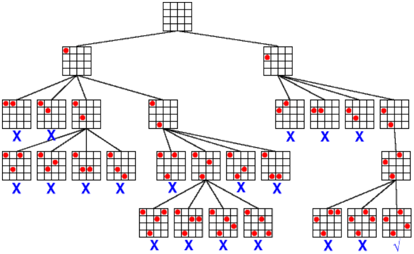


Problems

- Thrashing
- Redundancy
- Late Detection of Inconsistency

Overcoming GT's weakness

Backtracking

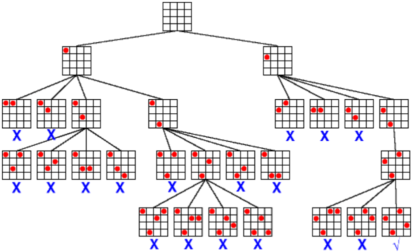


Problems

- Thrashing
- Redundancy
- Late Detection of Inconsistency

Overcoming GT's weakness

Backtracking

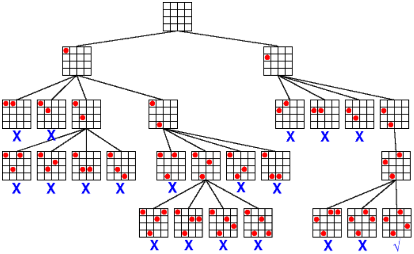


Problems

- Thrashing
- Redundancy
- Late Detection of Inconsistency

Overcoming GT's weakness

Backtracking



Problems

- Thrashing
- Redundancy
- Late Detection of Inconsistency

CP's Answer

Consistency Techniques

- detect inconsistency much earlier
- avoid redundancy and thrashing of BT

Definition

A consistency method transforms a CSP into an equivalent, consistent CSP.

How we will use it

Interleave consistency transformation and enumeration

CP's Answer

Consistency Techniques

- detect inconsistency much earlier
- avoid redundancy and thrashing of BT

Definition

A consistency method transforms a CSP into an equivalent, consistent CSP.

How we will use it

Interleave consistency transformation and enumeration

CP's Answer

Consistency Techniques

- detect inconsistency much earlier
- avoid redundancy and thrashing of BT

Definition

A consistency method transforms a CSP into an equivalent, consistent CSP.

How we will use it

Interleave consistency transformation and enumeration

Node and Arc Consistency

- Idea: Find equivalent, consistent CSP by removing values from the domains
- Examine one (elementary) constraint at a time
- Node consistency: unary constraints $c(X)$
  remove values from $D(X)$ that falsify c
- Arc consistency: binary constraints $c(X, Y)$
  remove from $D(X)$ values that have no support in $D(Y)$ such that c is satisfied and vice versa

Node and Arc Consistency

- Idea: Find equivalent, consistent CSP by removing values from the domains
- Examine one (elementary) constraint at a time
- Node consistency: unary constraints $c(X)$
  remove values from $D(X)$ that falsify c
- Arc consistency: binary constraints $c(X, Y)$
  remove from $D(X)$ values that have no support in $D(Y)$ such that c is satisfied and vice versa

Node and Arc Consistency

- Idea: Find equivalent, consistent CSP by removing values from the domains
- Examine one (elementary) constraint at a time
- Node consistency: unary constraints $c(X)$
  remove values from $D(X)$ that falsify c
- Arc consistency: binary constraints $c(X, Y)$
  remove from $D(X)$ values that have no support in $D(Y)$ such that c is satisfied and vice versa

Node and Arc Consistency

- Idea: Find equivalent, consistent CSP by removing values from the domains
- Examine one (elementary) constraint at a time
- Node consistency: unary constraints $c(X)$
  remove values from $D(X)$ that falsify c
- Arc consistency: binary constraints $c(X, Y)$
  remove from $D(X)$ values that have no support in $D(Y)$ such that c is satisfied and vice versa

Node Consistency

Definition

A unary constraint $c(X)$ is node consistent with domain D if $X = d$ satisfies $c(X)$ for each $d \in D(X)$.

Definition

A CSP $(\mathcal{X}, D, C)$ is node consistent, iff each of the unary constraints in C is node consistent with D.

Node Consistency Example

Our example CSP is not node consistent (see Z)

$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2$$
$$D(X) = D(Y) = D(Z) = \{1, 2, 3, 4\}$$

Node consistent, equivalent CSP

$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2$$
$$D(X) = D(Y) = \{1, 2, 3, 4\}, D(Z) = \{1, 2\}$$

Remark

- The 4-Queens CSP was node consistent, why?
- Computing node consistency is easy. Just look once at each unary constraint and remove inconsistent domain values.

Node Consistency Example

Our example CSP is not node consistent (see Z)

$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2$$
$$D(X) = D(Y) = D(Z) = \{1, 2, 3, 4\}$$

Node consistent, equivalent CSP

$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2$$
$$D(X) = D(Y) = \{1, 2, 3, 4\}, D(Z) = \{1, 2\}$$

Remark

- The 4-Queens CSP was node consistent, why?
- Computing node consistency is easy. Just look once at each unary constraint and remove inconsistent domain values.

Arc Consistency

Definition

A binary constraint $c(X, Y)$ is arc consistent with domain D if

- for each $d_X \in D(X)$ there is a $d_Y \in D(Y)$ s.t. $c(d_X, d_Y)$
- vice versa (for each $d_Y \in D(Y)$ there is a $d_X \in D(X)$ s.t. $c(d_X, d_Y)$)

Definition

A CSP $(\mathcal{X}, D, C)$ is arc consistent, iff each of the binary constraints in C is arc consistent with D.

Arc Consistency Example

The following CSP is node consistent but not arc consistent

$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2$$
$$D(X) = D(Y) = \{1, 2, 3, 4\}, D(Z) = \{1, 2\}$$

For example $4 \in D(Y)$ and $Y < Z$
Arc consistent, equivalent CSP

$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2$$
$$D(X) = D(Y) = D(Z) = \{\}$$

Remark
Our 4-Queens CSP is arc consistent.

Arc Consistency Example

The following CSP is node consistent but not arc consistent

$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2$$
$$D(X) = D(Y) = \{1, 2, 3, 4\}, D(Z) = \{1, 2\}$$

For example $4 \in D(Y)$ and $Y < Z$
Arc consistent, equivalent CSP

$$X < Y \text{ and } Y < Z \text{ and } Z \leq 2$$
$$D(X) = D(Y) = D(Z) = \{\}$$

Remark

Our 4-Queens CSP is arc consistent.

Computing Arc Consistency

```
procedure REVISE(c, X, Y, D)
  D(X) := { d_X ∈ D(X) such that there exists d_Y ∈ D(Y)
                                 where c(d_X, d_Y) is satisfied }
endproc

do
  D' := D
  foreach binary constraint c ∈ C do
    let X, Y denote the variables of c
    REVISE(c, X, Y, D)
    REVISE(c, Y, X, D)
  done
until D = D'
```

Remark

This algorithm is called AC-1, usually one uses improved variants of this algorithm (e.g. AC-3).

Computing Arc Consistency

```
procedure REVISE(c, X, Y, D)
  D(X) := { d_X ∈ D(X) such that there exists d_Y ∈ D(Y)
                               where c(d_X, d_Y) is satisfied }
endproc

do
  D' := D
  foreach binary constraint c ∈ C do
    let X, Y denote the variables of c
    REVISE(c, X, Y, D)
    REVISE(c, Y, X, D)
  done
until D = D'
```

Remark

This algorithm is called AC-1, usually one uses improved variants of this algorithm (e.g. AC-3).

Computing Arc Consistency

```
procedure REVISE(c, X, Y, D)
  D(X) := { d_X ∈ D(X) such that there exists d_Y ∈ D(Y)
                               where c(d_X, d_Y) is satisfied }
endproc

do
  D' := D
  foreach binary constraint c ∈ C do
    let X, Y denote the variables of c
    REVISE(c, X, Y, D)
    REVISE(c, Y, X, D)
  done
until D = D'
```

Remark

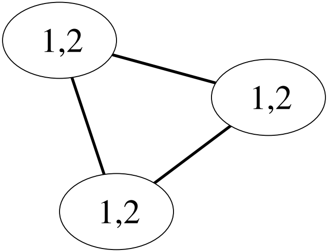
This algorithm is called AC-1, usually one uses improved variants of this algorithm (e.g. AC-3).

Avoiding Redundant Work: AC-3

```
Q :=empty queue
foreach binary constraint c ∈ C do
  push Q, (c, X, Y)
  push Q, (c, Y, X)
done

while Q ≠empty queue do
  (c,X,Y) := pop Q
  D':=D(X)
  REVISE(c, X, Y, D)
  if D(X) ≠ D' then
    for c' ∈ C and Z ∈ 𝒳 where c'(X, Z) or c'(Z, X) do
      push Q, (c', Z, X)
    done
  endif
done
```

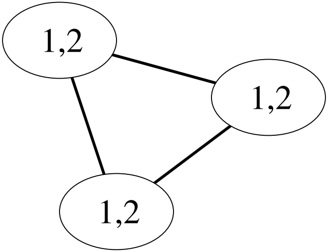
Node/Arc vs. Global Consistency



$$\mathcal{X} = \{X, Y, Z\}$$
$$D(X) = D(Y) = D(Z) = \{1, 2\}$$
$$C = \{X \neq Y, Y \neq Z, Z \neq X\}$$

- The CSP is node and arc consistent
- The CSP is globally inconsistent

Node/Arc vs. Global Consistency



$$\mathcal{X} = \{X, Y, Z\}$$
$$D(X) = D(Y) = D(Z) = \{1, 2\}$$
$$C = \{X \neq Y, Y \neq Z, Z \neq X\}$$

- The CSP is node and arc consistent
- The CSP is globally inconsistent

Consistency Methods: Summary

- Computing local consistency = constraint propagation
  - Node consistency
  - Arc consistency
  - (Hyper-arc consistency)
  - (Bounds consistency)
- Propagation is incomplete
- Solving a CSP requires search
  Combine backtracking and propagation

Complexity

- Local consistency: efficient
- CSP solving/global consistency: NP-hard

Consistency Methods: Summary

- Computing local consistency = constraint propagation
  - Node consistency
  - Arc consistency
  - (Hyper-arc consistency)
  - (Bounds consistency)
- Propagation is incomplete
- Solving a CSP requires search
  Combine backtracking and propagation

Complexity

- Local consistency: efficient
- CSP solving/global consistency: NP-hard

Consistency Methods: Summary

- Computing local consistency = constraint propagation
  - Node consistency
  - Arc consistency
  - (Hyper-arc consistency)
  - (Bounds consistency)
- Propagation is incomplete
- Solving a CSP requires search
  Combine backtracking and propagation

Complexity

- Local consistency: efficient
- CSP solving/global consistency: NP-hard

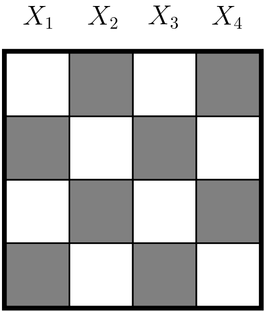
Consistency Methods: Summary

- Computing local consistency = constraint propagation
  - Node consistency
  - Arc consistency
  - (Hyper-arc consistency)
  - (Bounds consistency)
- Propagation is incomplete
- Solving a CSP requires search
  Combine backtracking and propagation

Complexity

- Local consistency: efficient
- CSP solving/global consistency: NP-hard

Solving 4-Queens (with Constraint Propagation)

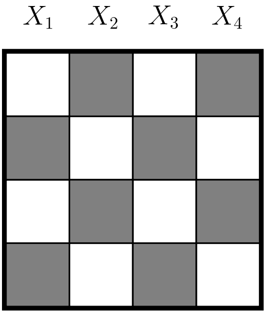


$$X_1, \ldots, X_4$$

$$D(X_i) = \{1, \ldots, 4\} \text{ for } i = 1..4$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Solving 4-Queens, $X_1 = 1$

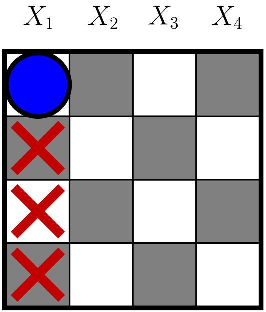


$$X_1, \ldots, X_4$$

$$D(X_i) = \{1, \ldots, 4\} \text{ for } i = 1..4$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Solving 4-Queens, $X_1 = 1$

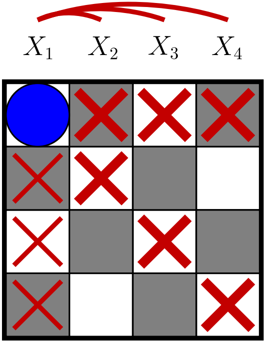


$$X_1, \ldots, X_4$$

$$D(X_1) = \{1\}, D(X_i) = \{1, \ldots, 4\} \text{ for } i = 2..4$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Solving 4-Queens, $X_1 = 1$

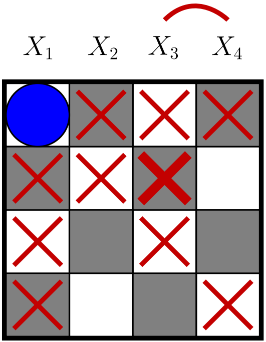


$$X_1, \ldots, X_4$$

$$D(X_1) = \{1\}, D(X_2) = \{3,4\}, D(X_3) = \{2,4\}, D(X_4) = \{2,3\}$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Solving 4-Queens, $X_1 = 1$

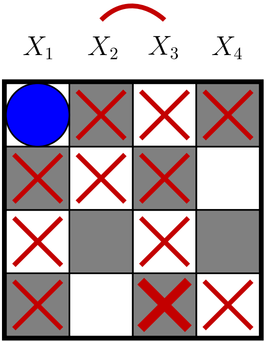


$$X_1, \ldots, X_4$$

$$D(X_1) = \{1\}, D(X_2) = \{3,4\}, D(X_3) = \{4\}, D(X_4) = \{2,3\}$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Solving 4-Queens, $X_1 = 1$

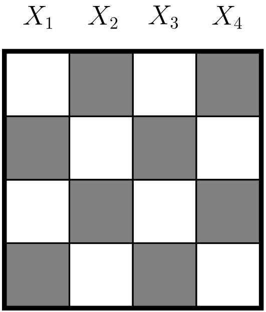


$$X_1, \dots, X_4$$

$$D(X_1) = \{1\}, D(X_2) = \{3, 4\}, D(X_3) = \{\}, D(X_4) = \{2, 3\}$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Solving 4-Queens, $X_1 = 2$

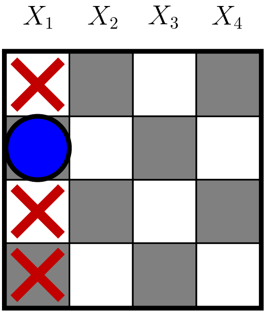


$$X_1, \ldots, X_4$$

$$D(X_i) = \{1, \ldots, 4\} \text{ for } i = 1..4$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Solving 4-Queens, $X_1 = 2$

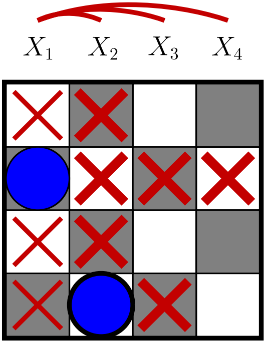


$$X_1, \ldots, X_4$$

$$\color{red}{D(X_1) = \{2\}}, D(X_i) = \{1, \ldots, 4\} \text{ for } i = 2..4$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Solving 4-Queens, $X_1 = 2$

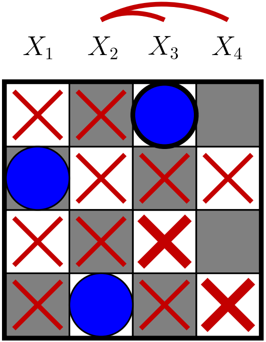


$$X_1, \ldots, X_4$$

$$D(X_1) = \{2\}, D(X_2) = \{4\}, D(X_3) = \{1,3\}, D(X_4) = \{1,3,4\}$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Solving 4-Queens, $X_1 = 2$

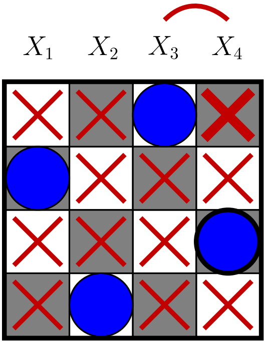


$$X_1, \ldots, X_4$$

$$D(X_1) = \{2\}, D(X_2) = \{4\}, D(X_3) = \{1\}, D(X_4) = \{3, 4\}$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Solving 4-Queens, $X_1 = 2$



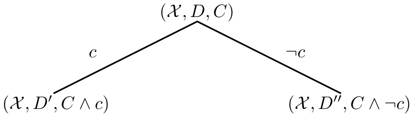
$$X_1, \ldots, X_4$$

$$D(X_1) = \{2\}, D(X_2) = \{4\}, D(X_3) = \{1\}, D(X_4) = \{3\}$$

$$X_i \neq X_{i'}, i - X_i \neq i' - X_{i'}, i + X_i \neq i' + X_{i'}$$

Constraint Search

- Combine Enumeration (backtracking) with propagation
- In general: enumeration by binary splits



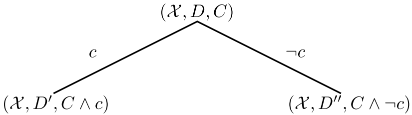
- Usually, we insert constraints of the form

$$X \diamond V, \qquad \diamond \in \{=, \leq, \geq, \dots\}$$

- Variable and value selection important!
  - for size of search tree
  - not for completeness/correctness

Constraint Search

- Combine Enumeration (backtracking) with propagation
- In general: enumeration by binary splits



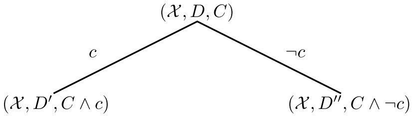
- Usually, we insert constraints of the form

$$X \diamond V, \qquad \diamond \in \{=, \leq, \geq, \dots\}$$

- Variable and value selection important!
  - for size of search tree
  - not for completeness/correctness

Constraint Search

- Combine Enumeration (backtracking) with propagation
- In general: enumeration by binary splits



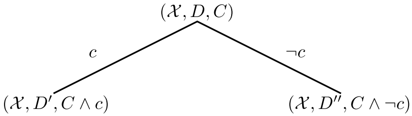
- Usually, we insert constraints of the form

$$X \diamond V, \qquad \diamond \in \{=, \leq, \geq, \dots\}$$

- Variable and value selection important!
  - for size of search tree
  - not for completeness/correctness

Constraint Search

- Combine Enumeration (backtracking) with propagation
- In general: enumeration by binary splits

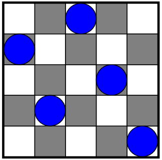


- Usually, we insert constraints of the form

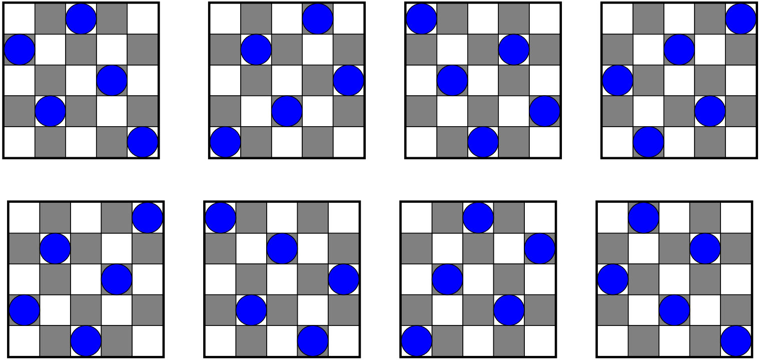
$$X \diamond V, \qquad \diamond \in \{=, \leq, \geq, \dots\}$$

- Variable and value selection important!
  - for size of search tree
  - not for completeness/correctness

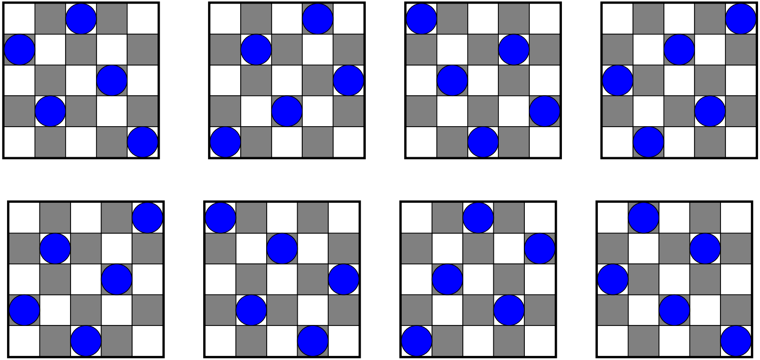
Symmetry



Symmetry

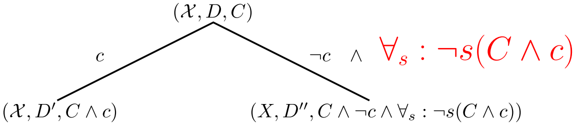


Symmetry



A symmetry is a (bijective) function on solutions.
This implies a symmetry function on constraints.

Symmetry Breaking Search



- Each right branch: forbid symmetries of the left branch
- By inserting a symmetric constraints for each symmetry

Constraint Optimization

Definition

A Constraint Optimization Problem (COP) is a CSP together with an objective function f on solutions.
A solution of the COP is a solution of the CSP that maximizes/minimizes f.

Solving by Branch & Bound Search
Idea of B&B:

- Backtrack & Propagate as for solving the CSP
- Whenever a solution s is found, add constraint "next solutions must be better than $f(s)$".

Constraint Optimization Example: Photo Problem

Alice,Bob,Carol, and Dave want to align for a photo
For example: $\boxed{\text{Alice, Carol, Dave, Bob}}$

However, they have preferences:

- Alice wants to stand next to Dave
- Bob wants to stand next to Dave and Carol
- Carol wants to stand next to Alice

Satisfy as many preferences as possible by constraint optimization.

Constraint Optimization Example: Photo Problem

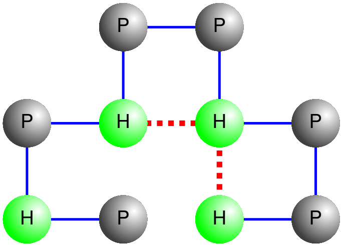
Alice,Bob,Carol, and Dave want to align for a photo
For example: $\boxed{\text{Alice, Carol, Dave, Bob}}$

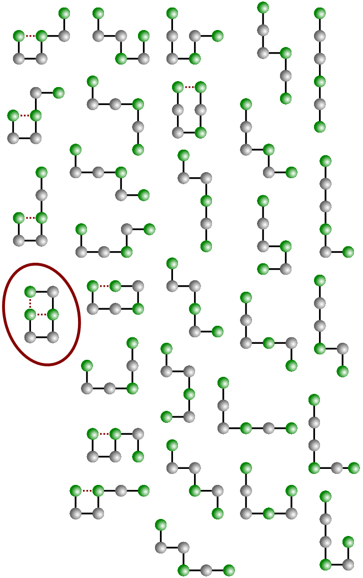
However, they have preferences:

- Alice wants to stand next to Dave
- Bob wants to stand next to Dave and Carol
- Carol wants to stand next to Alice

Satisfy as many preferences as possible by constraint optimization.

Application: Protein Structure Prediction





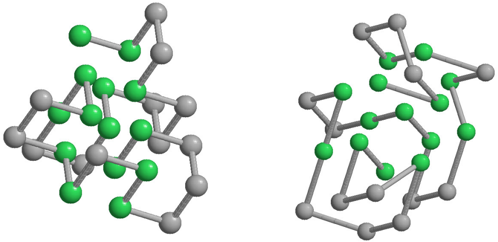
Exact Prediction in 3D cubic & FCC

The problem

IN: sequence s in $\{H, P\}^n$

HHPPPHHPHHPPHHHPPHHPPPHPPHH

OUT: self avoiding walk ω on cubic/fcc lattice with minimal HP-energy $E_{\mathrm{HP}}(s, \omega)$



A First Constraint Model

- Variables $X_1, \ldots, X_n, Y_1, \ldots, Y_n, Z_1, \ldots, Z_n$ and *HHContacts*

$$\begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} \text{ is the position of the } i\text{th monomer } \omega(i)$$

- Domains

$$D(X_i) = D(Y_i) = D(Z_i) = \{-n, \ldots, n\}$$

- Constraints
  1. positions i and $i+1$ are neighbored (chain)
  2. all positions differ (self-avoidance)
  3. relate *HHContacts* to X_i, Y_i, Z_i
  4. $\begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$

A First Constraint Model

- Variables $X_1, \ldots, X_n, Y_1, \ldots, Y_n, Z_1, \ldots, Z_n$ and $HHContacts$

$$\begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} \text{ is the position of the } i\text{th monomer } \omega(i)$$

- Domains

$$D(X_i) = D(Y_i) = D(Z_i) = \{-n, \ldots, n\}$$

- Constraints
  1. positions i and $i+1$ are neighbored (chain)
  2. all positions differ (self-avoidance)
  3. relate $HHContacts$ to X_i, Y_i, Z_i
  4. $\begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$

A First Constraint Model

- Variables $X_1, \dots, X_n, Y_1, \dots, Y_n, Z_1, \dots, Z_n$ and $HHContacts$

$$\begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} \text{ is the position of the } i\text{th monomer } \omega(i)$$

- Domains

$$D(X_i) = D(Y_i) = D(Z_i) = \{-n, \dots, n\}$$

- Constraints
  1. positions i and $i+1$ are neighbored (chain)
  2. all positions differ (self-avoidance)
  3. relate $HHContacts$ to X_i, Y_i, Z_i
  4. $\begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$

The First Model in More Detail (Cubic Lattice)

The Constraints cannot be expressed directly, i.e. we need auxiliary variables

$$Xdiff_{ij} = |X_i - X_j| \quad Ydiff_{ij} = |Y_i - Y_j| \quad Zdiff_{ij} = |Z_i - Z_j|$$

1. Positions i and $i+1$ neighbored (chain)

$$Xdiff_{i(i+1)} + Ydiff_{i(i+1)} + Zdiff_{i(i+1)} = 1$$

2. All positions differ (self-avoidance)

$$Xdiff_{ij} + Ydiff_{ij} + Zdiff_{ij} \neq 0 \quad (\text{for } i \neq j).$$

3. Relate *HHContacts* to X_i, Y_i, Z_i
   Detect HH-contact, if $Xdiff_{ij} + Ydiff_{ij} + Zdiff_{ij} = 1$ for $s_i = s_j = H$. Then add 1 to *HHContacts*.
   (Technically, use reified constraints)

The First Model in More Detail (Cubic Lattice)

The Constraints cannot be expressed directly, i.e. we need auxiliary variables

$$Xdiff_{ij} = |X_i - X_j| \quad Ydiff_{ij} = |Y_i - Y_j| \quad Zdiff_{ij} = |Z_i - Z_j|$$

1. Positions i and $i + 1$ neighbored (chain)

$$Xdiff_{i(i+1)} + Ydiff_{i(i+1)} + Zdiff_{i(i+1)} = 1$$

2. All positions differ (self-avoidance)

$$Xdiff_{ij} + Ydiff_{ij} + Zdiff_{ij} \neq 0 \quad (\text{for } i \neq j).$$

3. Relate *HHContacts* to X_i, Y_i, Z_i
   Detect HH-contact, if $Xdiff_{ij} + Ydiff_{ij} + Zdiff_{ij} = 1$ for $s_i = s_j = H$. Then add 1 to *HHContacts*.
   (Technically, use reified constraints)

The First Model in More Detail (Cubic Lattice)

The Constraints cannot be expressed directly, i.e. we need auxiliary variables

$$Xdiff_{ij} = |X_i - X_j| \quad Ydiff_{ij} = |Y_i - Y_j| \quad Zdiff_{ij} = |Z_i - Z_j|$$

1. Positions i and $i+1$ neighbored (chain)

$$Xdiff_{i(i+1)} + Ydiff_{i(i+1)} + Zdiff_{i(i+1)} = 1$$

2. All positions differ (self-avoidance)

$$Xdiff_{ij} + Ydiff_{ij} + Zdiff_{ij} \neq 0 \quad (\text{for } i \neq j).$$

3. Relate *HHContacts* to X_i, Y_i, Z_i
   Detect HH-contact, if $Xdiff_{ij} + Ydiff_{ij} + Zdiff_{ij} = 1$ for $s_i = s_j = H$. Then add 1 to *HHContacts*.
   (Technically, use reified constraints)

Solving the First Model

- Model is a COP (Constraint Optimization Problem)
- Branch and Bound Search for Minimizing *Energy*
- Combined with Symmetry Breaking
- How good is the propagation?
- Main problem of propagation: bounds on contacts/energy
  From a partial solution, the solver cannot estimate the maximally possible number of HH-contacts well.

Solving the First Model

- Model is a COP (Constraint Optimization Problem)
- Branch and Bound Search for Minimizing *Energy*
- Combined with Symmetry Breaking
- How good is the propagation?
- Main problem of propagation: bounds on contacts/energy
  From a partial solution, the solver cannot estimate the maximally possible number of HH-contacts well.

Solving the First Model

- Model is a COP (Constraint Optimization Problem)
- Branch and Bound Search for Minimizing *Energy*
- Combined with Symmetry Breaking
- How good is the propagation?
- Main problem of propagation: bounds on contacts/energy
  From a partial solution, the solver cannot estimate the maximally possible number of HH-contacts well.

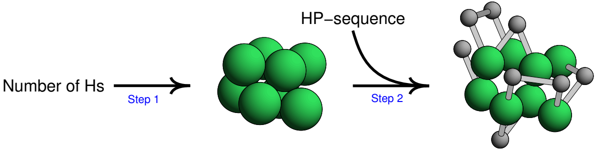
Solving the First Model

- Model is a COP (Constraint Optimization Problem)
- Branch and Bound Search for Minimizing *Energy*
- Combined with Symmetry Breaking
- How good is the propagation?
- Main problem of propagation: bounds on contacts/energy
  From a partial solution, the solver cannot estimate the maximally possible number of HH-contacts well.

Solving the First Model

- Model is a COP (Constraint Optimization Problem)
- Branch and Bound Search for Minimizing *Energy*
- Combined with Symmetry Breaking
- How good is the propagation?
- Main problem of propagation: bounds on contacts/energy
  From a partial solution, the solver cannot estimate the maximally possible number of HH-contacts well.

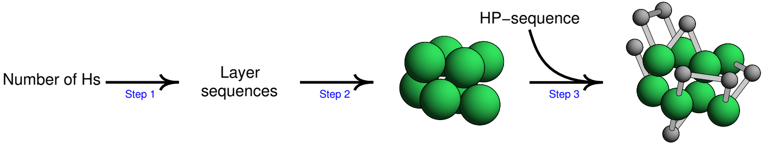
The Advanced Approach: Cubic & FCC



Steps

1. Core Construction
2. Mapping

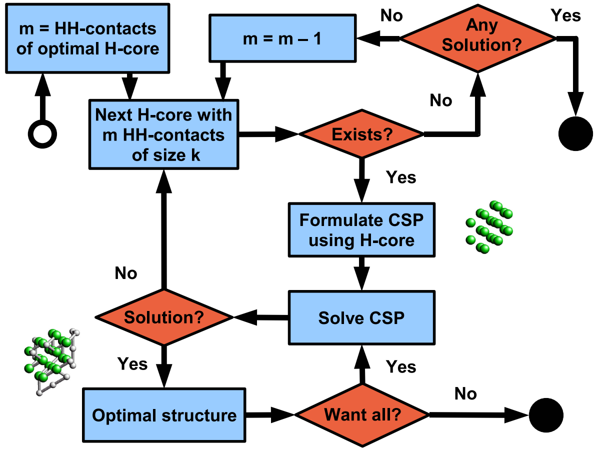
The Advanced Approach: Cubic & FCC



Steps

1. Bounds
2. Core Construction
3. Mapping

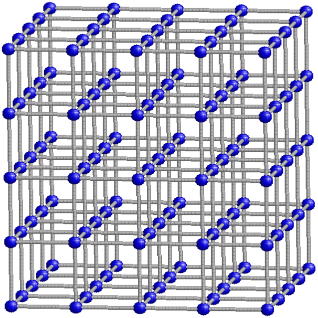
Workflow: Predict Best Structure(s) of HP-Sequence

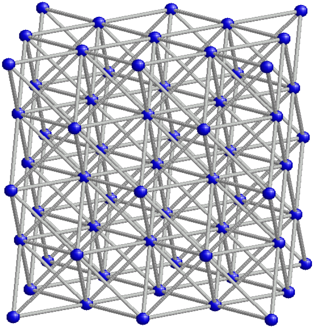


Computing Bounds

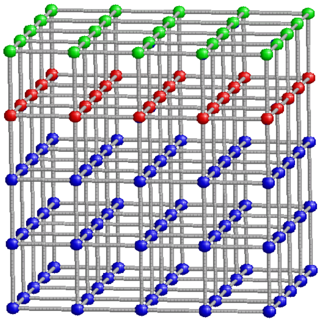
- Prepares the construction of cores
- How many contacts are possible for n monomers, if freely distributed to lattice points
- Answering the question will give information for core construction
- Main idea: split lattice into layers
  consider contacts
  - within layers
  - between layers

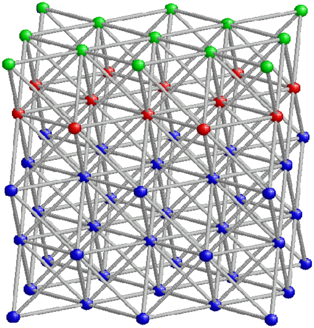
Layers: Cubic & FCC Lattice





Layers: Cubic & FCC Lattice



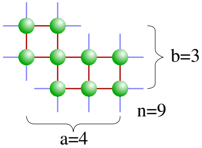


Contacts

Contacts =

Layer contacts + Contacts between layers

- Bound Layer contacts: Contacts $\leq 2 \cdot n - a - b$

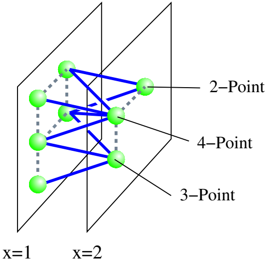


- Bound Contacts between layers
  - cubic: one neighbor in next layer

    $$Contacts \leq \min(n_1, n_2)$$

  - FCC: **four** neighbors in next layer

    $$i - \text{points}$$

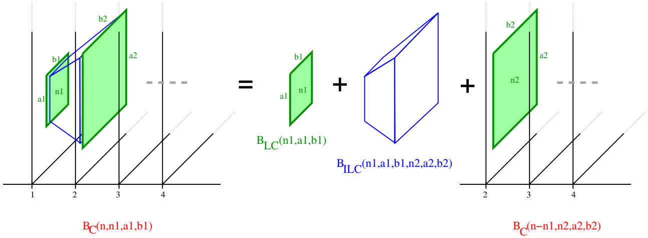


Bounding Interlayer Contacts in the FCC

- **Needed:**
  - upper bound for number of contacts between two successive layers in FCC
  - NOTE: Layers only described by parameters $(n_1, a_1, b_1); (n_2, a_2, b_2)$
- **Method:**
  - compute bounds for number of 1/2/3/4-points of first layer
  - distribute n_2 points greedily
  - technical difficulty: tight bounds of 1/2/3/4-points depend on further parameters
- **Result:** $\mathrm{B}^{\mathrm{FCC}}_{\mathrm{ILC}}(n_1, a_1, b_1, n_2, a_2, b_2)$

Recall: $\mathrm{B}^{\mathrm{cubic}}_{\mathrm{ILC}}(n_1, a_1, b_1, n_2, a_2, b_2) = \min(n_1, n_2)$

Recursion Equation for Bounds

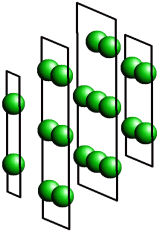


- $\mathrm{B_C}(n, n_1, a_1, b_1)$: Contacts of core with n elements and first layer L_1 : n_1, a_1, b_1
- $\mathrm{B_{LC}}(n_1, a_1, b_1)$: Contacts in L_1
- $\mathrm{B_{ILC}}(n_1, a_1, b_1, n_2, a_2, b_2)$: Contacts between E_1 and E_2 : n_2, a_2, b_2
- $\mathrm{B_C}(n - n_1, n_2, a_2, b_2)$: Contacts in core with $n - n_1$ elements and first layer E_2

Layer sequences

From Recursion:

- by Dynamic Programming: Upper bound on number of contacts
- by Traceback: Set of layer sequences



layer sequence $= (n_1, a_1, b_1), \ldots, (n_4, a_4, b_4)$

Set of layer sequences gives distribution of points to layers in all point sets that possibly have maximal number of contacts

Core Construction

Problem

IN: number n, contacts c

OUT: all point sets of size n with c contacts

- Optimization problem
- Core construction is a hard combinatorial problem

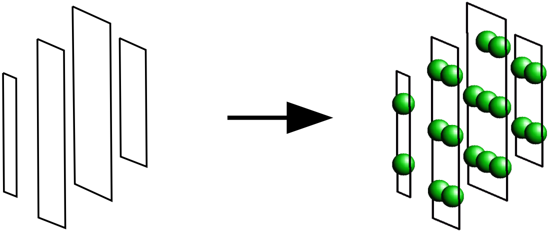
Core construction: Modified Problem

Poblem

IN: number n, contacts c, set of layer sequences S_{ls}

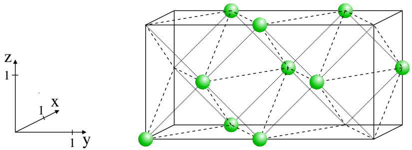
OUT: all point sets of size n with c contacts and layer sequences in S_{ls}

- Use constraints from layer sequences
- Model as constraint satisfaction problem (CSP)



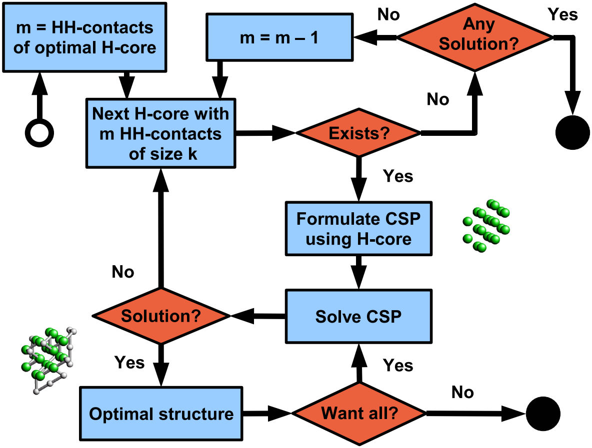
$(n_1, a_1, b_1), \ldots, (n_4, a_4, b_4)$ Core = Set of lattice points

Core Construction — Details



- Number of layers = length of layer sequence
- Number of layers in x, y, and z: Surrounding Cube
- enumerate numbers of layers ⇒ fix cube ⇒ enumerate points

Workflow

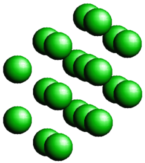


m = HH-contacts of optimal H-core
m = m – 1
No
Any Solution?
Yes
Next H-core with m HH-contacts of size k
Exists?
No
Yes
Formulate CSP using H-core
No
Solution?
Solve CSP
Yes
Yes
Optimal structure
Want all?
No

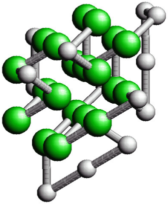
Mapping Sequences to Cores

find structure such that

- H-Monomers on core positions $\rightarrow$ hydrophobic core
- all positions differ $\rightarrow$ self-avoiding
- chain connected $\rightarrow$ walk



compact core



optimal structure

Mapping Sequence to Cores — CSP

Given: sequence s of size n and n_H Hs
core *Core* of size n_H

CSP Model

- Variables $X_1, \ldots, X_n$
  X_i is position of monomer i

  Encode positions as integers

$$\mathsf{I}\begin{pmatrix} x \\ y \\ z \end{pmatrix} \equiv x + M * y + M^2 * z$$

(unique encoding for 'large enough' M)

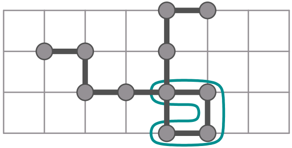
- Constraints
  1. $X_i \in Core$ for all $s_i = H$
  2. X_i and X_{i+1} are neighbors
  3. $X_1, \ldots, X_n$ are all different

Constraints for Self-avoiding Walks

- Single Constraints "self-avoiding" and "walk" weaker than their combination
- no efficient algorithm for consistency of combined constraint "self-avoiding walk"
- relaxed combination: stronger and more efficient propagation

k-avoiding walk constraint

Example: 4-avoiding, but not 5-avoiding



Putting it together

Predict optimal structures by combining the three steps

1. Bounds
2. Core Construction
3. Mapping

Some Remarks

- Pre-compute optimal cores for relevant core sizes
  Given a sequence, only perform Mapping step
- Mapping to cores may fail!
  We use suboptimal cores and iterate mapping.
- Approach extensible to HPNX
  HPNX-optimal structures at least nearly optimal for HP.
- Approach extensible to side chains
  H side chains form core.

Putting it together

Predict optimal structures by combining the three steps

1. Bounds
2. Core Construction
3. Mapping

Some Remarks

- Pre-compute optimal cores for relevant core sizes
  Given a sequence, only perform Mapping step
- Mapping to cores may fail!
  We use suboptimal cores and iterate mapping.
- Approach extensible to HPNX
  HPNX-optimal structures at least nearly optimal for HP.
- Approach extensible to side chains
  H side chains form core.

Putting it together

Predict optimal structures by combining the three steps

1. Bounds
2. Core Construction
3. Mapping

Some Remarks

- Pre-compute optimal cores for relevant core sizes
  Given a sequence, only perform Mapping step
- Mapping to cores may fail!
  We use suboptimal cores and iterate mapping.
- Approach extensible to HPNX
  HPNX-optimal structures at least nearly optimal for HP.
- Approach extensible to side chains
  H side chains form core.

Putting it together

Predict optimal structures by combining the three steps

1. Bounds
2. Core Construction
3. Mapping

Some Remarks

- Pre-compute optimal cores for relevant core sizes
  Given a sequence, only perform Mapping step
- Mapping to cores may fail!
  We use suboptimal cores and iterate mapping.
- Approach extensible to HPNX
  HPNX-optimal structures at least nearly optimal for HP.
- Approach extensible to side chains
  H side chains form core.

Time efficiency

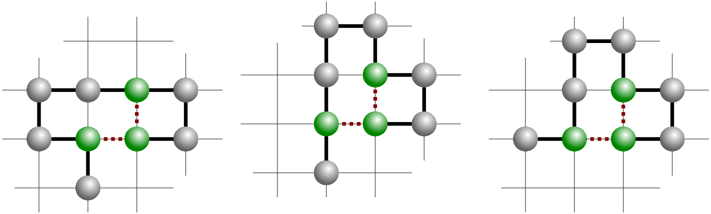
Prediction of one optimal structure
("Harvard Sequences", length 48 [Yue *et al.*, 1995])

CPSP	PERM
0,1 s	6,9 min
0,1 s	40,5 min
4,5 s	100,2 min
7,3 s	284,0 min
1,8 s	74,7 min
1,7 s	59,2 min
12,1 s	144,7 min
1,5 s	26,6 min
0,3 s	1420,0 min
0,1 s	18,3 min

- CPSP: "our approach", constraint-based
- PERM [Bastolla *et al.*, 1998]: stochastic optimization

Many Optimal Structures

Sequence HPPHPPPPHP



. . . ?

- There can be many ...
- HP-model is degenerated
- Number of optimal structures = degeneracy

Completeness

Predicted number of all optimal structures
("Harvard Sequences")

CPSP	CHCC
10.677.113	1500×10^3
28.180	14×10^3
5.090	5×10^3
1.954.172	54×10^3
1.868.150	52×10^3
106.582	59×10^3
15.926.554	306×10^3
2.614	1×10^3
580.751	188×10^3

- CPSP: "our approach"
- CHCC [Yue *et al.*, 1995]: complete search with hydrophobic cores

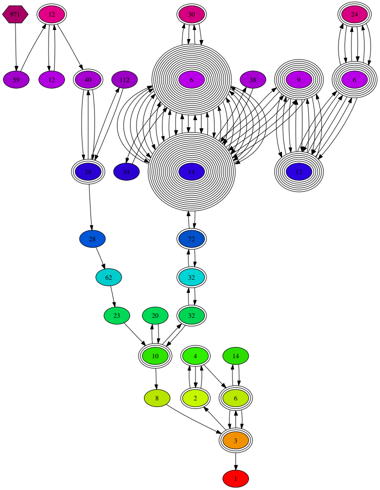
Unique Folder

- HP-model degenerated
- Low degeneracy $\approx$ stable $\approx$ protein-like
- Are there protein-like, unique folder in 3D HP models?
- How to find out?

Unique Folder

- HP-model degenerated
- Low degeneracy ≈ stable ≈ protein-like
- Are there protein-like, unique folder in 3D HP models?
- How to find out?

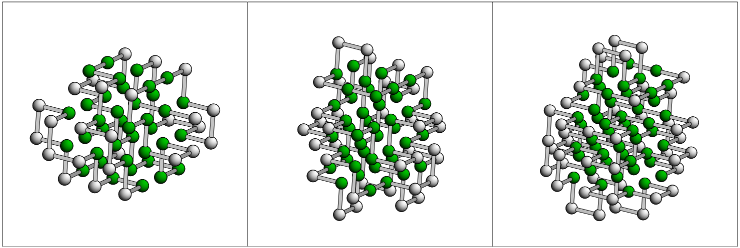
MC-search through sequence space



Unique Folder

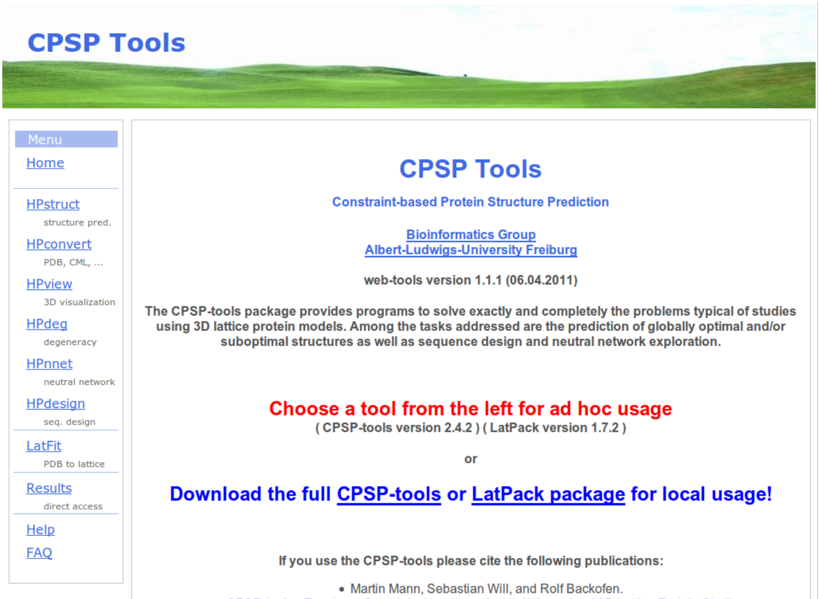
- HP-model degenerated
- Low degeneracy $\approx$ stable $\approx$ protein-like
- Are there protein-like, unique folder in 3D HP models?
- How to find out?

Yes: many, e.g. about 10,000 for $n=27$



Software: CPSP Tools

http://cpsp.informatik.uni-freiburg.de



CPSP Tools

Menu

Home

HPstruct
structure pred.

HPconvert
PDB, CML, ...

HPview
3D visualization

HPdeg
degeneracy

HPnnet
neutral network

HPdesign
seq. design

LatFit
PDB to lattice

Results
direct access

Help

FAQ

CPSP Tools

Constraint-based Protein Structure Prediction

Bioinformatics Group
Albert-Ludwigs-University Freiburg

web-tools version 1.1.1 (06.04.2011)

The CPSP-tools package provides programs to solve exactly and completely the problems typical of studies using 3D lattice protein models. Among the tasks addressed are the prediction of globally optimal and/or suboptimal structures as well as sequence design and neutral network exploration.

Choose a tool from the left for ad hoc usage
(CPSP-tools version 2.4.2) (LatPack version 1.7.2)

or

Download the full CPSP-tools or LatPack package for local usage!

If you use the CPSP-tools please cite the following publications:

- Martin Mann, Sebastian Will, and Rolf Backofen.