Counting Protein Structures by DFS with Dynamic Decomposition

Sebastian Will and Martin Mann

Albert-Ludwigs-University Freiburg
Bioinformatics at the Department of Computer Science

Workshop on
Constraint Based Methods for Bioinformatics 2006

Overview

Lattice Proteins

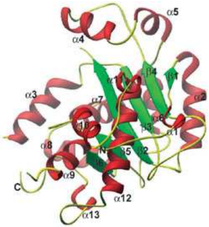
'Counting Protein Structures
by DFS with Dynamic Decomposition'

What we need to know:

- Protein Structures ? ✘
- Prediction as CSP ?
- Counting by Decomposition ?

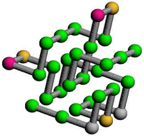
Protein structures

Real protein structure



⇒

Simple lattice proteins

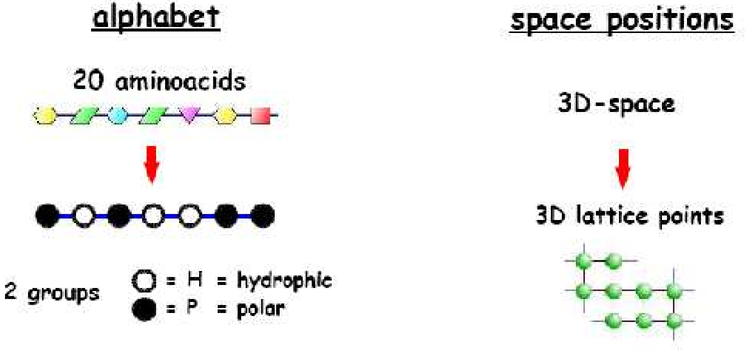


Lattice Proteins

- protein simplified to chain of monomers
- structure depends on underlying lattice
- energy depends on contact energy function

HP-Model

Abstractions



alphabet

20 aminoacids

2 groups

○ = H = hydrophic

● = P = polar

space positions

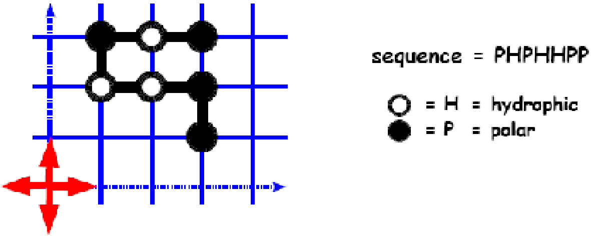
3D-space

3D lattice points

HP-Model
Structure Model

Structure

- sequence monomers are placed on lattice positions
- structure = selfavoiding walk on the lattice

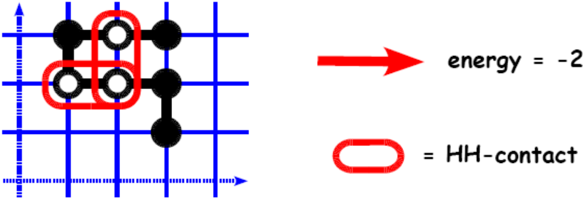


HP-Model
Energy function

Energy function

- focus on hydrophobic forces
- contact based → HH-contacts

$$\text{energy}(x,y) = \begin{cases} -1 & \text{if } (x,y == H) \text{ and } (x,y \text{ neighbors}) \\ 0 & \text{else} \end{cases}$$



Simple and nice... But what for?

Applications

Applications e.g.

- Neutral nets and protein evolution
- Exploring energy landscapes and protein kinetic
- Base for more complex protein models
- . . .

Therefore you need:

Prediction of optimal structures

- NP-complete in 3D-lattice (Berger & Leighton, 1998) (even in 2D)
- can be solved by Constraint Programming !
  (Backofen & Will, 2006)

Ok, what next?
CSP formulation

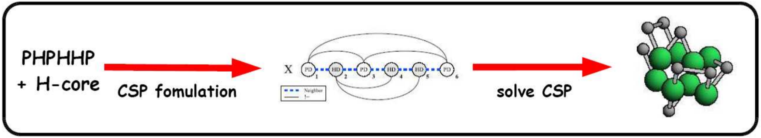
'Counting Protein Structures
by DFS with Dynamic Decomposition'

What we need to know:

- Protein Structures ? ✔
- Prediction as CSP ? ✘
- Counting by Decomposition ?

The Constraint Satisfaction Problem

A CSP for optimal structure prediction in the HP-lattice-model


PHPHHP
+ H-core

CSP fomulation

solve CSP

Rolf Backofen and Sebastian Will
'A constraint-based approach to fast and exact structure prediction

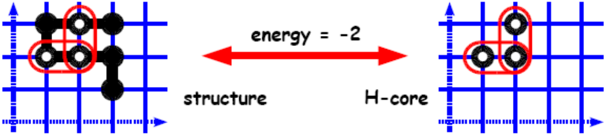
in three-dimensional protein models' 2006

Preliminary

H-Cores

H-Core of a given structure

- H-Core = set of H-monomer positions
- core energy ↔ structure energy (only HH-contacts important)



- optimality implies optimal structure energy
- candidates can be precomputed based on H-number
- hard problem too → (solved via CP)

⇒ for now used as black box and given … !

The Idea

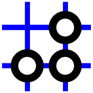
given

an HP-sequence

P-H-P-H-H-P-P

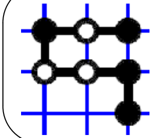
and

an optimal H-core



there is the question:

Exists a structure, so that all H-monomers are placed on H-core positions?



YES!

The CSP

A very very simple Formulation

For a given HP-sequence and an optimal H-core:

Variables

- one for each sequence monomer

Domains = sets of lattice positions

- H-Monomers: H-core positions (ensures optimality)
- P-Monomers: remaining lattice

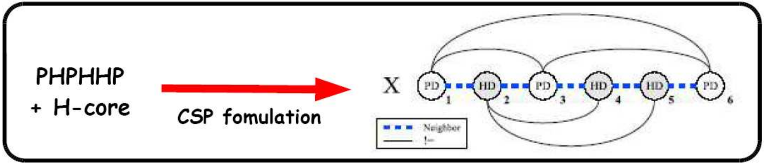
Constraints

- binary Neighboring constraints along the chain (backbone)
- one global Alldifferent constraint (selfavoiding structure)

$\Rightarrow$ encodes the selfavoiding walk

Structure prediction as CSP

The CSP as Constraint Graph:



From Solution to structure

- a CSP solution assigns a lattice position to each monomer
- solution = structure, and optimal due to H-core !
- normal CSP-solving approaches can be applied
  e.g. DFS-branching combined with constraint propagation

Ok, what next?

Decomposition strategy

'Counting Protein Structures
by DFS with Dynamic Decomposition'

What we need to know:

- Protein Structures ? ✓
- Prediction as CSP ? ✓
- Counting by Decomposition ? ✗

Counting CSP solutions

via Depth-first Search (DFS)

Counting all solutions

- is in complexity class #P-complete for counting problems
- = an important field of CP

- can be done by Constraint Propagation and DFS branching
- iterative process
- can be formulated as a recursion . . .

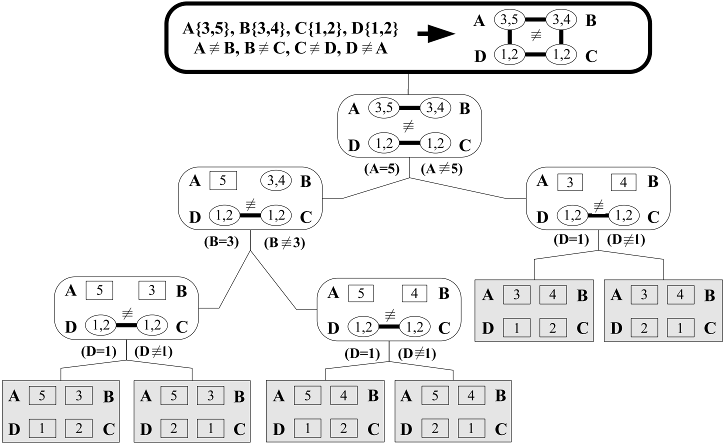
Counting DFS recursion

The 3 steps . . .

```
function CDFS($\mathcal{X}, \mathcal{D}, \mathcal{C}$)
                                                        ▷ reduce domains
    $(\mathcal{D}', \mathcal{C}') \leftarrow$ Propagate($\mathcal{X}, \mathcal{D}, \mathcal{C}$)
                                                        ▷ check for recursion stop
    if IsFailed($\mathcal{X}, \mathcal{D}', \mathcal{C}'$) then return 0
    else if IsSolved($\mathcal{X}, \mathcal{D}'$) then return 1
    else
                                                        ▷ branch search
        $c \leftarrow$ Select($\mathcal{X}, \mathcal{D}'$)
        return CDFS($\mathcal{X}, \mathcal{D}', \mathcal{C}' \cup \{c\}$) + CDFS($\mathcal{X}, \mathcal{D}', \mathcal{C}' \cup \{\neg c\}$)
    end if
end function
```

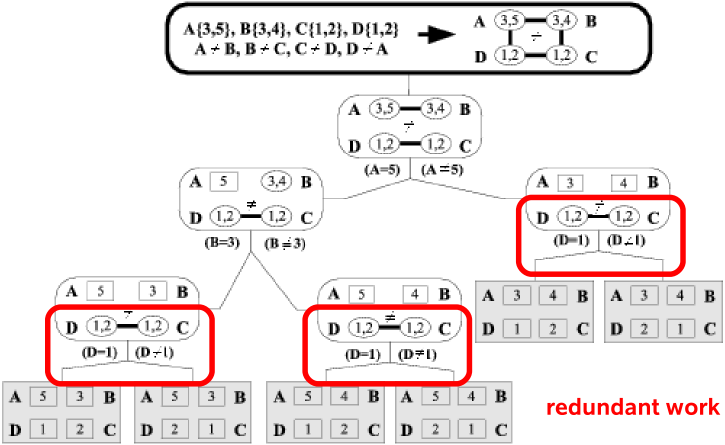
Counting DFS recursion

An example



Counting DFS recursion

The problem: redundancy!



redundant work

Counting DFS recursion

The problem: redundancy!

The problem

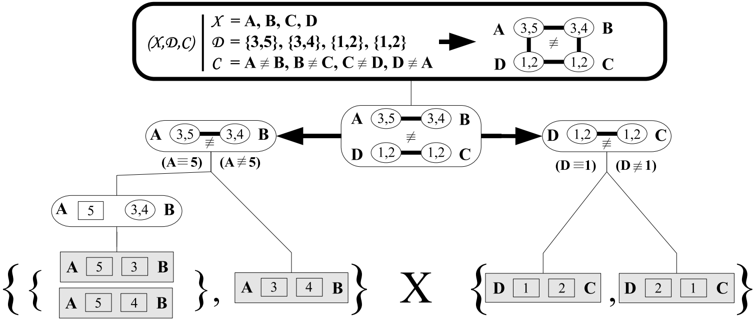
✘ Redundant work due to independent partial problems!

How to avoid?

- detect decomposition into independent partial problems
- do it dynamically during search (CSP changes)
- solve each partial problem independently
- generate the overall solution number (via product)

Counting using Decomposing DFS

First try with an example



$(\mathcal{X},\mathcal{D},\mathcal{C})$ | $\mathcal{X}$ = **A, B, C, D**

$\mathcal{D}$ = **{3,5}, {3,4}, {1,2}, {1,2}**

$\mathcal{C}$ = **A ≠ B, B ≠ C, C ≠ D, D ≠ A**

$$\{\,\{\ \boxed{A\ 5\ 3\ B},\ \boxed{A\ 5\ 4\ B}\ \},\ \boxed{A\ 3\ 4\ B}\,\}\ \times\ \{\,\boxed{D\ 1\ 2\ C},\ \boxed{D\ 2\ 1\ C}\,\}$$

Counting using Decomposing DFS

Counting using dynamic decomposing DFS

```
function DDFS($\mathcal{X}, \mathcal{D}, \mathcal{C}$)
                                                        ▷ reduce domains
    $(\mathcal{D}', \mathcal{C}') \leftarrow$ Propagate($\mathcal{X}, \mathcal{D}, \mathcal{C}$)
                                                        ▷ check for recursion stop
    if IsFailed($\mathcal{X}, \mathcal{D}', \mathcal{C}'$) then return 0
    else if IsSolved($\mathcal{X}, \mathcal{D}'$) then return 1
                                                        ▷ decomposing branching
    else  $s \leftarrow 1$                                        ▷ initialize counter
        $\mathfrak{D} \leftarrow$ Decompose($\mathcal{X}, \mathcal{D}', \mathcal{C}'$)
        for all $(\hat{\mathcal{X}}, \hat{\mathcal{D}}, \hat{\mathcal{C}}) \in \mathfrak{D}$ do
            $c \leftarrow$ Select($\hat{\mathcal{X}}, \hat{\mathcal{D}}$)
            $s = s \cdot ($ DDFS($\hat{\mathcal{X}}, \hat{\mathcal{D}}, \hat{\mathcal{C}} \cup \{c\}$) + DDFS($\hat{\mathcal{X}}, \hat{\mathcal{D}}, \hat{\mathcal{C}} \cup \{\neg c\}$))
        end for
        return $s$
    end if
end function
```

Decomposing DFS

What was it?

DDFS

- DFS → redundant solving of independent partial problems
- can be avoided by DDFS via dynamic decomposition
- overall solutions are generated during backtracking
- = general approach for exhaustive solution enumeration

But . . .

- early and strong decompositions neccessary
- ⇒ new requirements to variable and value selection
- ⇒ problem specific heuristics important
- applicable for global constraints too! (using binarisation)
- first recursion draft can be improved (see paper) ☺

Ok, what next?

Results

'Counting Protein Structures
by DFS with Dynamic Decomposition'

What we need to know:

- Protein Structures ? ✓
- Prediction as CSP ? ✓
- Counting by Decomposition ? ✓

OK, but does it help? Results . . .

Results

for structure prediction

Using DDFS for structure counting in HP-model

Decomposing versus 'normal' DFS

- in a first implementation:
  - 10x less branchings
  - 2x faster
- possible speedup higher !!!
  - further algorithmic improvements
  - optimize implementation (e.g. constraint graph handling)
  - better problem specific branching heuristics

Summary

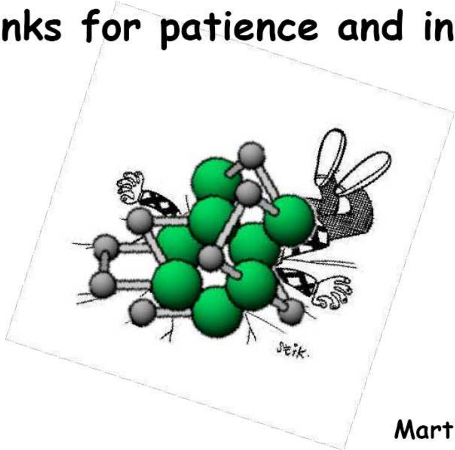
'Counting Protein Structures
by DFS with Dynamic Decomposition'

Lets summarise:

1. optimal structure prediction for lattice proteins can be formulated as CSP
2. counting all solutions via DFS yields redundancy
3. can be avoided by dynamic decompositions ⇒ DDFS
4. leads to big speedups and less branchings
5. DDFS is a general approach ⇒ other CSPs

That's all folks!

Thanks for patience and interest



Martin Mann