Valued Constraint Satisfaction

Tutorial – CP 2010

Martin Cooper
IRIT, Toulouse, France

Simon de Givry
INRA, Toulouse, France

Peter Jeavons
University of Oxford, UK

with contributed slides by Thomas Schiex (INRA, France), Javier Larrosa (UPC, Spain), D. Allouche & A. Favier (INRA, France), R. Dechter (UCI, USA), R. Marinescu (4C, Ireland)

Valued Constraint Satisfaction

- What is it and why do we need it?
- Can it be done efficiently?
- Search
- Problem transformations
- Open problems

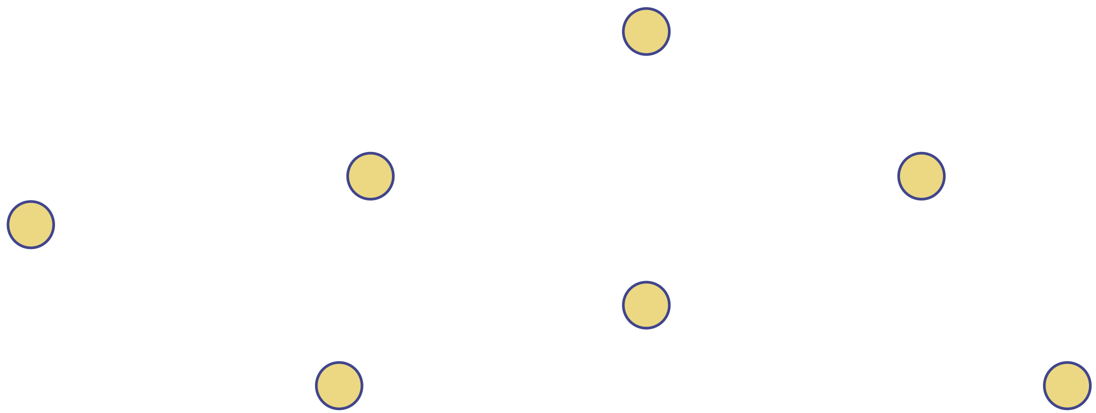
Chapter 1. What is it?

Motivation,

Definitions,

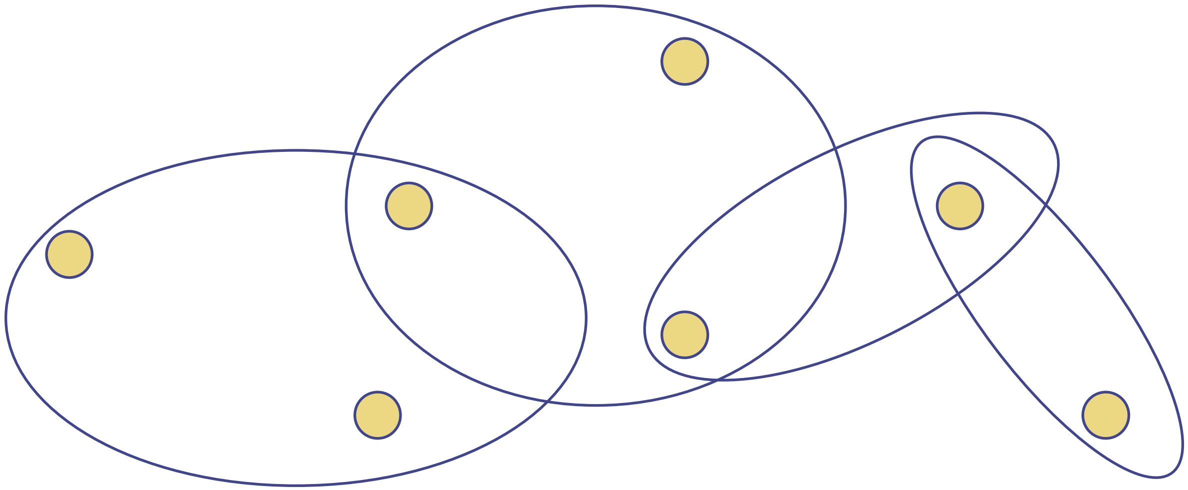
Some general theorems

A unifying abstraction



Variables = Talks to be scheduled at conference
Transmitters to be assigned frequencies
Amino acids to be located in space
Circuit components to be placed on a chip

A unifying abstraction



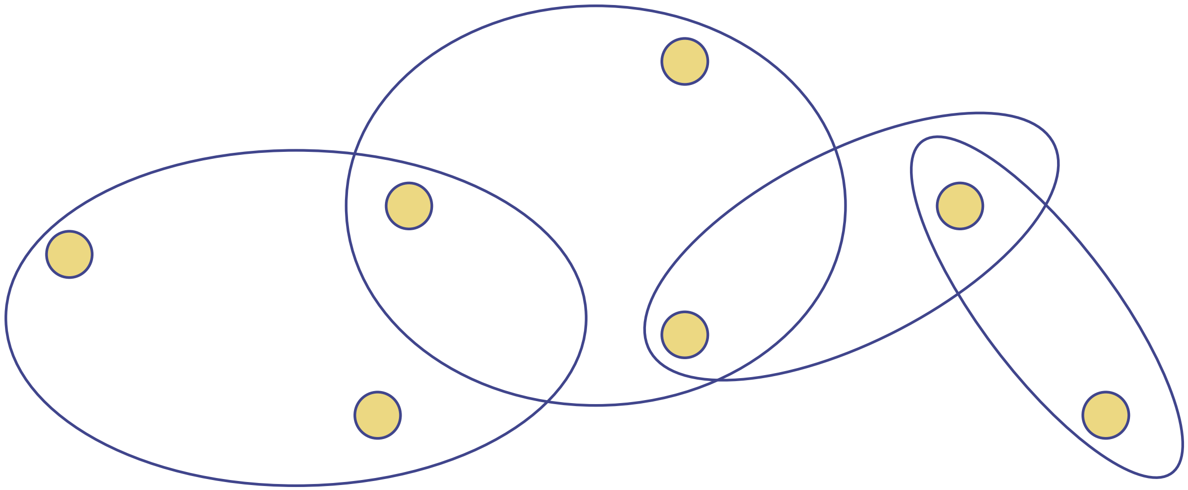
Constraints = All invited talks on different days

No interference between near transmitters

$x + y + z > 0$

Foundations dug before walls built

A unifying abstraction



A solution is an assignment of values to variables that satisfies all the constraints

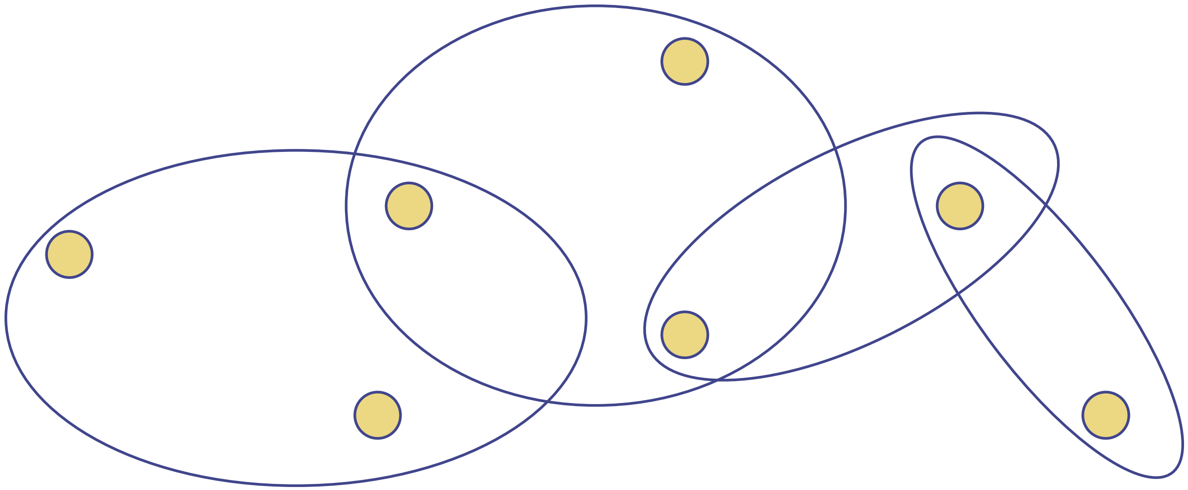
But what if…

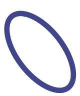
- There are lots of solutions, but some are better than others?
- There are no solutions, but some assignments satisfy more constraints than others?
- We don't know the exact constraints, only probabilities, or fuzzy membership functions?
- We're willing to violate some constraints if we can get a better overall solution that way?

Fragmentation

- COP
- Max-CSP
- Max-SAT
- WCSP
- FCSP
- HCLP
- Pseudo-Boolean Optimisation
- Bayesian Networks
- Random Markov Fields
- Integer Programming
- …

A unifying abstraction



Constraints  associate costs with each assignment

A solution is an assignment of values to variables that minimises the combined costs

Definition of a VCSP instance

- a set of n variables X_i with domains d_i
- a set of valued constraints, where each constraint has a
  - scope (list of variables)
  - cost function (function from assignments to costs)

It only remains to specify what the possible costs are, and how to combine them

Definition of a valuation structure

- a set S of costs
- a total order $<$
- minimum and maximum elements:

  we denote these by 0 and ∞
- an aggregation operator $\oplus$ which is commutative, associative, monotonic, and such that $\forall\alpha,\ \alpha\oplus 0=\alpha$

Examples of valuation structures

- If $S = \{0, \infty\}$, then VCSP $\equiv$ CSP
- If $S = \{0, 1, 2, \ldots, \infty\}$, and $\oplus$ is addition, then VCSP generalizes MAX-CSP
- If $S = [0,1]$, and $\oplus$ is max, then VCSP $\equiv$ Fuzzy CSP
- If $S = \{0, 1, \ldots., k\}$, and $\oplus$ is bounded addition $+_k$ where $\alpha +_k \beta = \min \{k, \alpha+\beta\}$, then VCSP $\equiv$ WCSP

Families of valuation structures

- A valuation structure is idempotent if
  $\forall\alpha,\ \alpha \oplus \alpha = \alpha$

All idempotent valuation structures are equivalent to Fuzzy CSP

Families of valuation structures

A valuation structure is strictly monotonic if

$\forall \alpha < \beta, \forall \gamma < \infty, \ \alpha \oplus \gamma < \beta \oplus \gamma$

A valuation structure is fair if

aggregation has a partial inverse, that is,

$\forall \alpha \geq \beta, \exists \gamma$ such that $\beta \oplus \gamma = \alpha$

All strictly monotonic valuation structures can be embedded in a fair valuation structure

Families of valuation structures

- A valuation structure is discrete if between any pair of finite costs there are finitely many other costs

All discrete and fair valuation structures
can be decomposed into
a contiguous sequence of valuation structures
with aggregation operator $+_k$

Bibliography

- For general background on VCSP and other formalisms for soft constraints, see the chapter on "Soft Constraints" by Meseguer, Rossi and Schiex, in the *Handbook of Constraint Programming*, Elsevier, 2006.

- For classification results on valuation structures see "Arc Consistency for Soft Constraints", *Cooper & Schiex,* AIJ, 2004.

Chapter 2. Efficiency

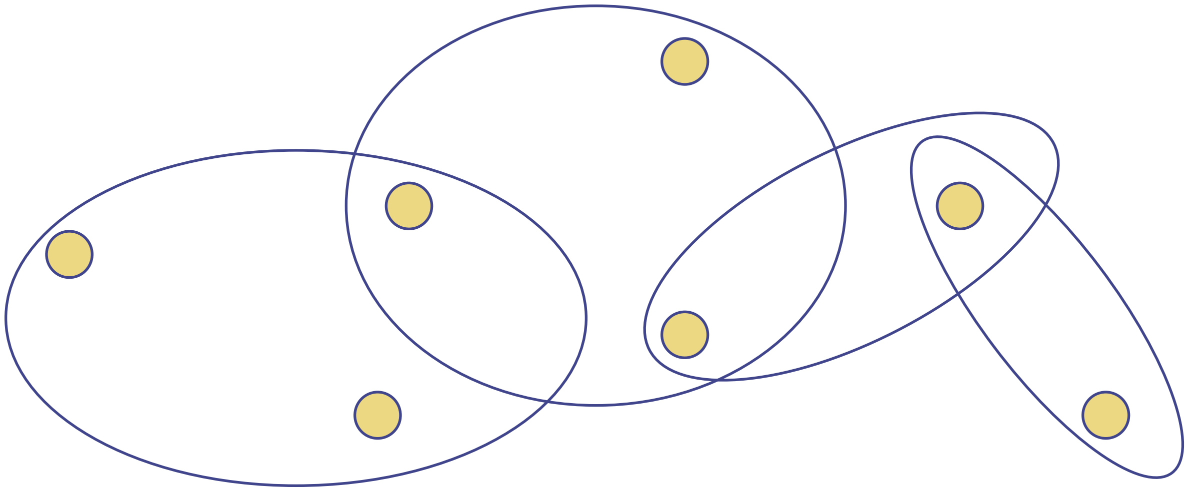
Structural restrictions,
Valued constraint languages,
Submodularity,
Multimorphisms

General question

Having a unified formulation allows us to ask *general* questions about efficiency:

When is the VCSP tractable?

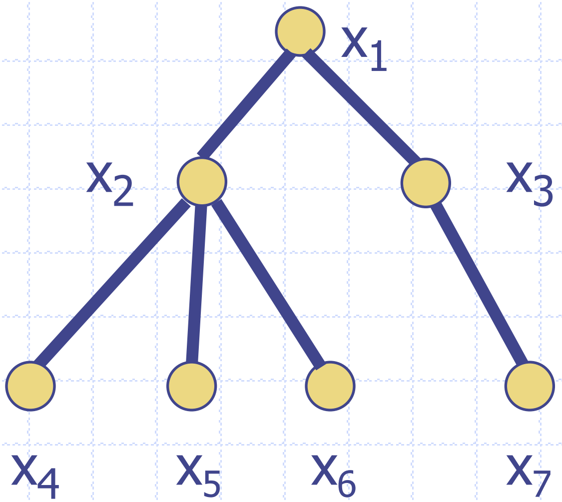
Problem features



- This picture illustrates the constraint *scopes*
- The set of scopes is sometimes called the *constraint hypergraph*, or the *scheme*
- Restricting the scheme can lead to tractability, as in the standard CSP

Structural tractability

Tree-structured binary VCSPs are tractable



Time complexity $O(e\ d^2)$
Space complexity $O(n\ d)$

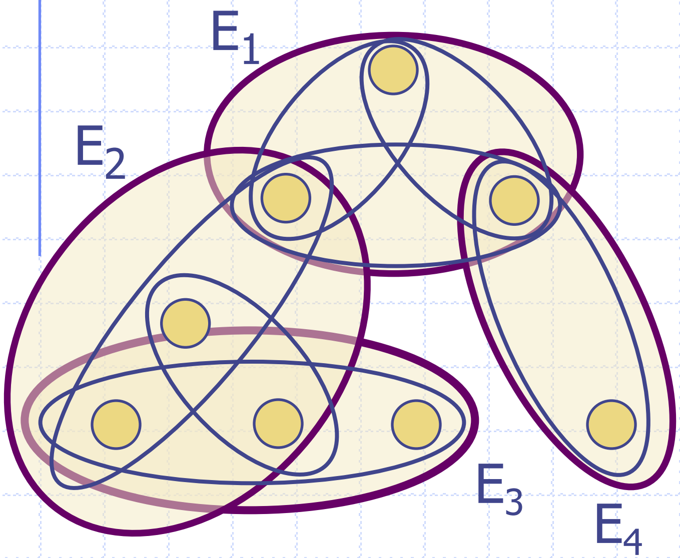
n: number of variables
d: maximum domain size
e: number of cost functions

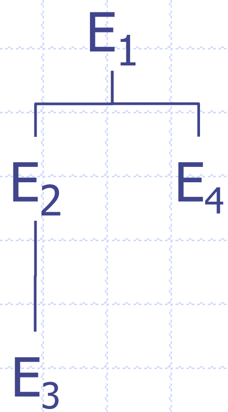
Proceed from the leaf nodes to a chosen root node

Project out leaf nodes by minimising over possible assignments

Tree decomposition

Bounded treewidth VCSPs are tractable





Time complexity $O(e\ d^{w+1})$
Space complexity $O(n\ d^{s})$

w: bounded treewidth
$= \max |E_i| - 1$

s: $\max \{|E_i \cap E_j| : i \neq j\}$

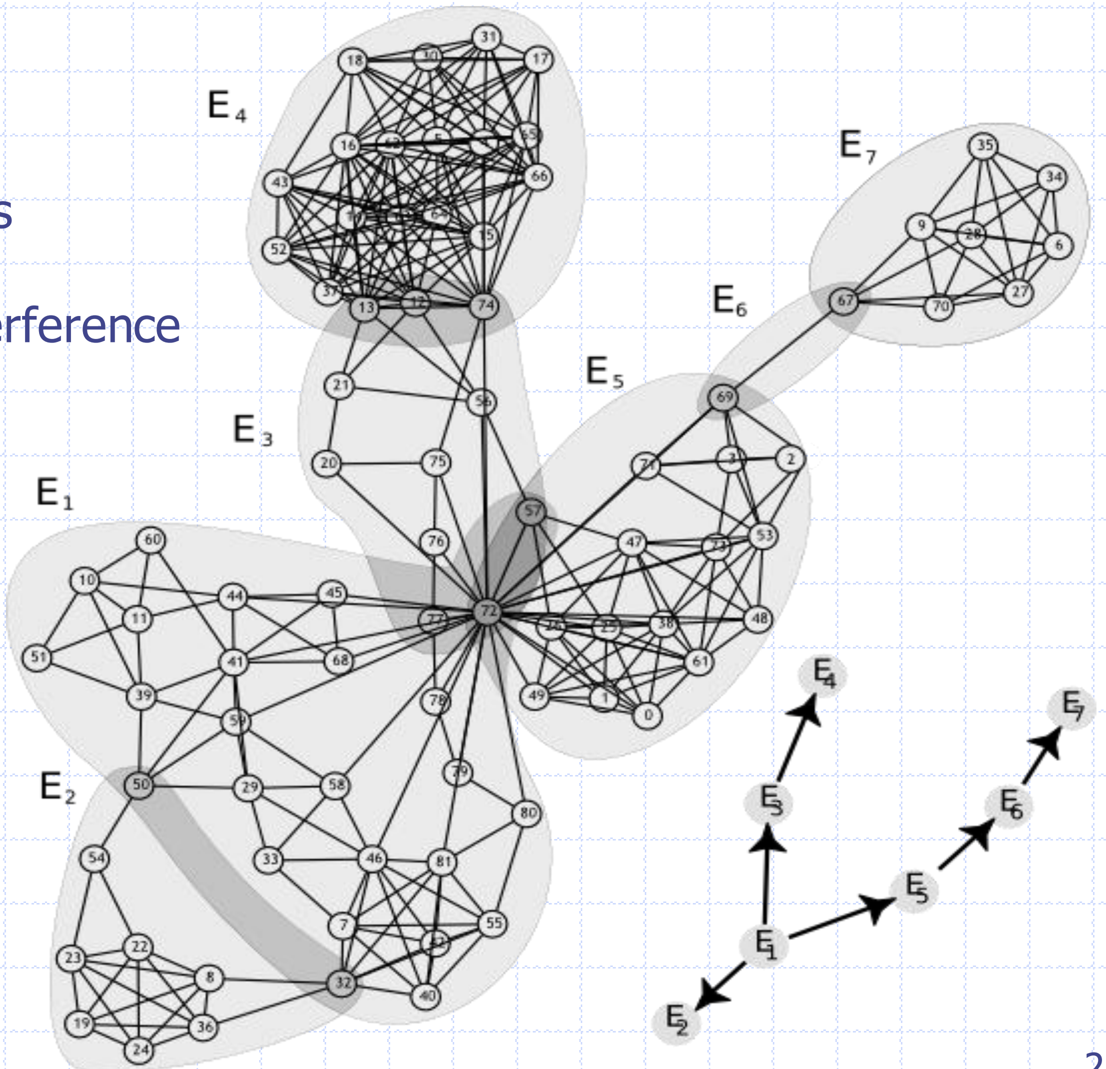
Finding a tree decomposition with minimum w* is NP-hard!

Tree decomposition example

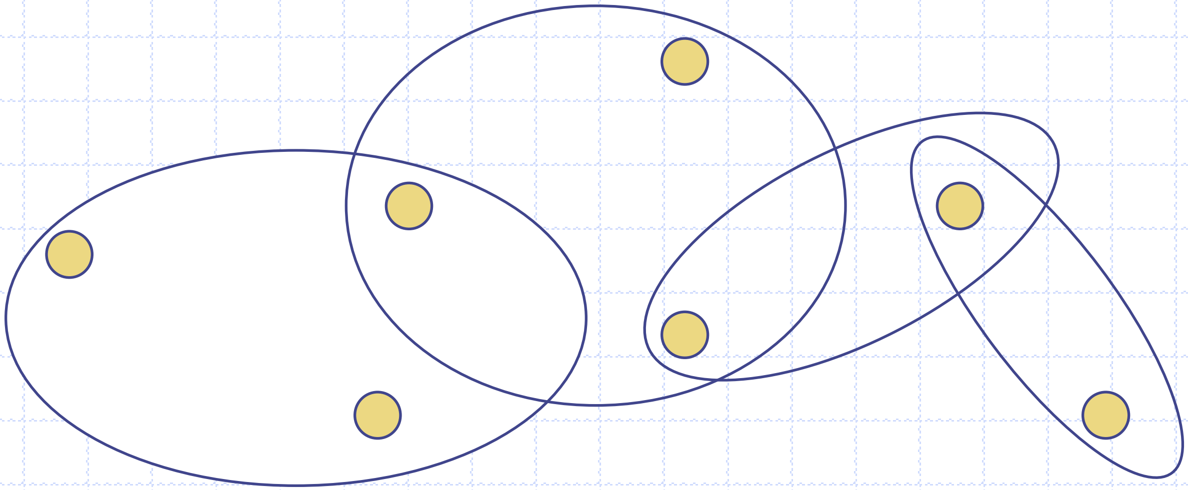
Benchmark problem
assigning frequencies
to transmitters
to minimise total interference

CELAR scen06r
$n = 82$
$d = 44$
$e = 327$

$w = 26$
$s = 3$

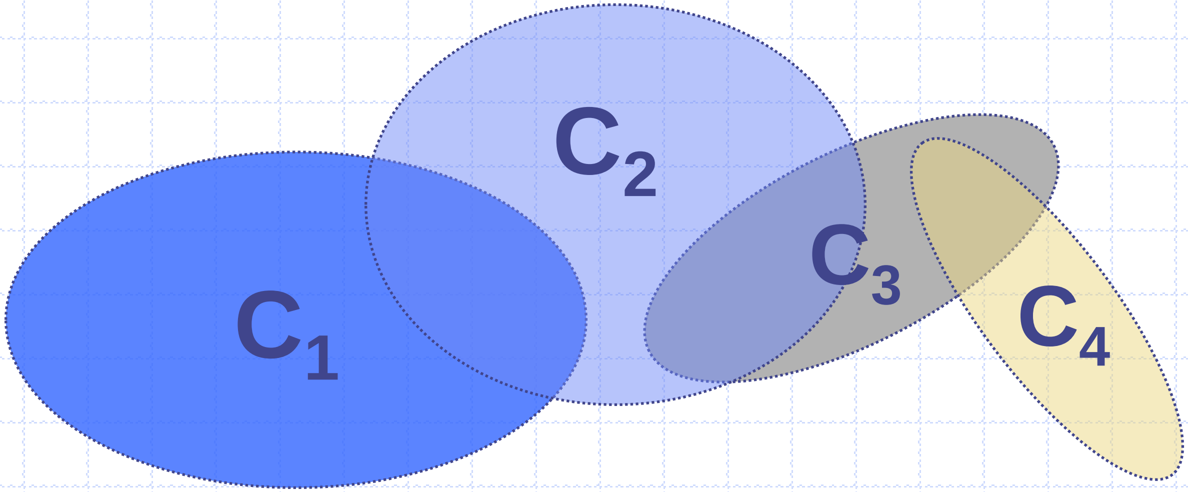


Problem features



- We have seen that structural features of a problem can lead to tractability
- This is very similar to the standard CSP
- What about other kinds of restrictions to the VCSP?

More problem features



- The picture now emphasises the cost functions
- Restricting the cost functions we allow can also lead to tractability

Valued constraint languages

- A set of cost functions is called a valued constraint language
- VCSP(Γ) represents the set of VCSP instances whose cost functions belong to the valued constraint language Γ
- For some choices of Γ, VCSP(Γ) is tractable
- We will consider some examples where the valuation structure contains non-negative real values and infinity, and aggregation is standard addition

Submodular functions

A class of functions that has been widely studied in OR is the submodular functions…

A cost function c is submodular if $\forall \mathbf{s},\mathbf{t}$

$$c(\min(\mathbf{s},\mathbf{t})) + c(\max(\mathbf{s},\mathbf{t})) \leq c(\mathbf{s}) + c(\mathbf{t})$$

where min and max are applied component-wise, i.e.

$$\min(\langle s_1,\ldots,s_k\rangle,\langle t_1,\ldots,t_k\rangle) = \langle \min(s_1,t_1),\ldots,\min(s_k,t_k)\rangle$$

VCSP($\Gamma_{\text{submodular}}$) is tractable

Examples of submodular functions

x	y	z	
0	0	0	0
0	0	1	1
0	1	0	7
0	1	1	1
1	0	0	∞
1	0	1	3
1	1	0	∞
1	1	1	0

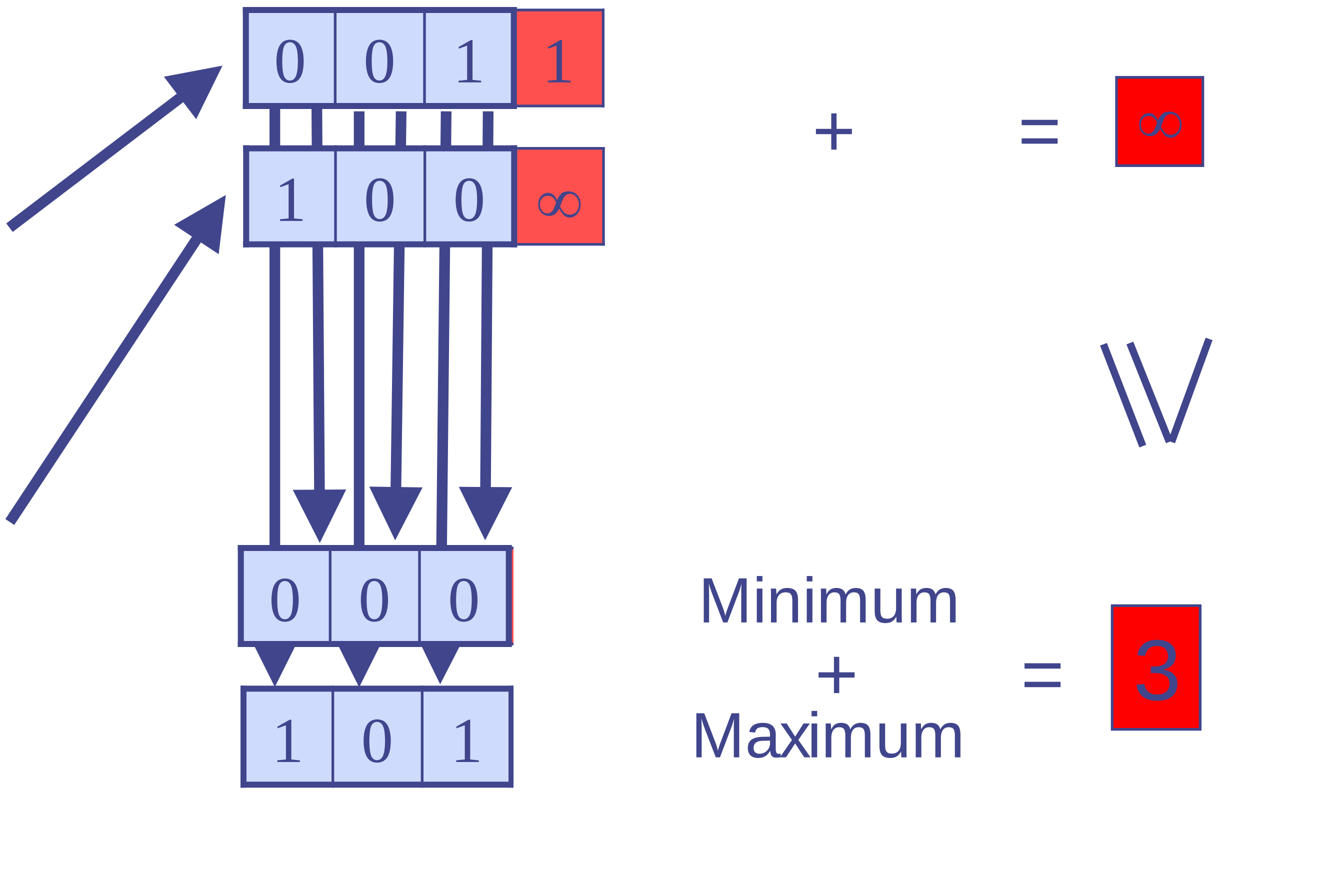
1	0	1

Maximum

Examples of submodular functions

$\forall s,t \;\; \text{Cost}(\text{Min}(s,t)) + \text{Cost}(\text{Max}(s,t)) \leq \text{Cost}(s) + \text{Cost}(t)$

	x	y	z	
	0	0	0	0
s	0	0	1	1
	0	1	0	7
	0	1	1	1
t	1	0	0	∞
	1	0	1	3
	1	1	0	∞
	1	1	1	0



Examples of submodular functions

- all unary functions
- all linear functions (of any arity)
- the binary function ϕ_{cut}

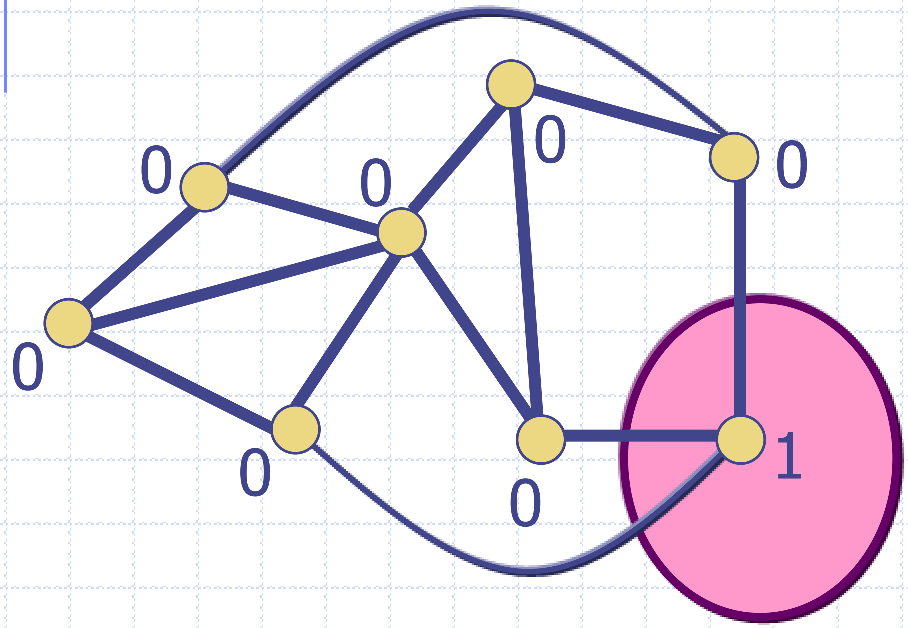
  where $\phi_{cut}(a,b)=1$ if $(a,b)=(0,1)$ (0 otherwise)
- the rank function of a matroid
- the Euclidean distance function between two points (x_1, x_2), (x_3, x_4) in the plane
- $\phi(x,y)=(x-y)^r$ if $x \geq y$ (∞ otherwise) for $r \geq 1$

(compare "*Simple Temporal CSPs with strictly monotone preferences*"

Khatib et al, IJCAI 2001)

Example: Min-Cut

The Min-Cut problem can be modelled by the single submodular binary cost function ϕ_{cut}



VCSP with domain {0,1}

Valued constraints on all edges (both ways) with cost function ϕ_{cut}

$\phi_{cut}(a,b)=1$ if $(a,b)=(0,1)$

Solution to VCSP is a Min-Cut

Algorithms

- The best known *general* algorithm for Boolean submodular function minimisation is $O(n^6)$

  (see Orlin "A faster strongly polynomial time algorithm for submodular function minimization", *Mathematical Programming*, 2009)

- However, many special cases can be solved much more efficiently…

Boolean submodular functions

Many Boolean submodular functions can be expressed using the binary function ϕ_{cut}

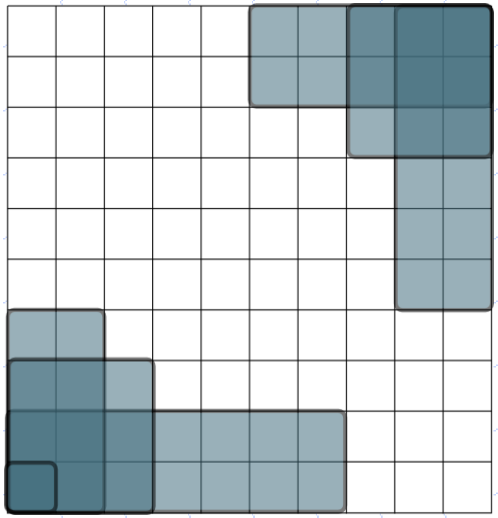
(these include all {0,1}-valued Boolean submodular functions, all binary and all ternary Boolean submodular functions, and many others)

$\boxed{\text{VCSP}(\{\phi_{cut}\}) \text{ is } O(n^3)}$

See Zivny & Jeavons "Classes of submodular constraints expressible by graph cuts", Constraints, 2010

Binary submodular functions

Binary submodular functions
over any finite domain
can be expressed as a sum of
"Generalized Interval" functions



(they correspond to Monge matrices)

$\boxed{\text{Binary VCSP}(\Gamma_{\text{submodular}}) \text{ is } O(n^3d^3)}$

See Cohen et al "A maximal tractable class of soft constraints", JAIR 2004

Beyond submodularity

$\forall s,t \;\; \mathrm{Cost}(\mathrm{Min}(s,t)) + \mathrm{Cost}(\mathrm{Max}(s,t)) \leq \mathrm{Cost}(s) + \mathrm{Cost}(t)$

x	y	z	
0	0	0	0
0	0	1	1
0	1	0	7
0	1	1	1
1	0	0	∞
1	0	1	3
1	1	0	∞
1	1	1	0

We say that the cost function has the ***multimorphism*** (Min,Max)

By choosing *other* functions, we can obtain other tractable valued constraint languages…

Known tractable cases

If the cost functions all have one of these eight multimorphisms, then the problem is tractable:

1) (Min,Max)
2) (Max,Max)
3) (Min,Min)
4) (Majority,Majority,Majority)
5) (Minority,Minority,Minority)
6) (Majority,Majority,Minority)
7) (Constant 0)
8) (Constant 1)

See Cohen et al "The complexity of soft constraint satisfaction", AIJ 2006

A dichotomy theorem

If the cost functions all have one of these eight multimorphisms, then the problem is tractable:

1) (Min,Max)
2) (Max,Max)
3) (Min,Min)
4) (Majority,Majority,Majority)
5) (Minority,Minority,Minority)
6) (Majority,Majority,Minority)
7) (Constant 0)
8) (Constant 1)

For Boolean cost functions…

In all other cases the cost functions have **no** significant common multimorphisms and the VCSP problem is **NP-hard**.

See Cohen et al “The complexity of soft constraint satisfaction”, AIJ 2006

Benefits of a general approach

- The dichotomy theorem immediately implies earlier results for SAT, MAX-SAT, Weighted Min-Ones and Weighted Max-Ones
- Multimorphisms have also been used to show that not all submodular functions can be expressed using binary functions (see Zivny et al "The expressive power of binary submodular functions", Discrete Applied Maths, 2009)
- Multimorphisms allow submodularity to be generalised to a bigger class of tractable languages (see Cohen et al "Generalizing submodularity and Horn clauses: Tractable optimisation problems defined by tournament pair multimorphisms", Theoretical Computer Science, 2008)

Bibliography

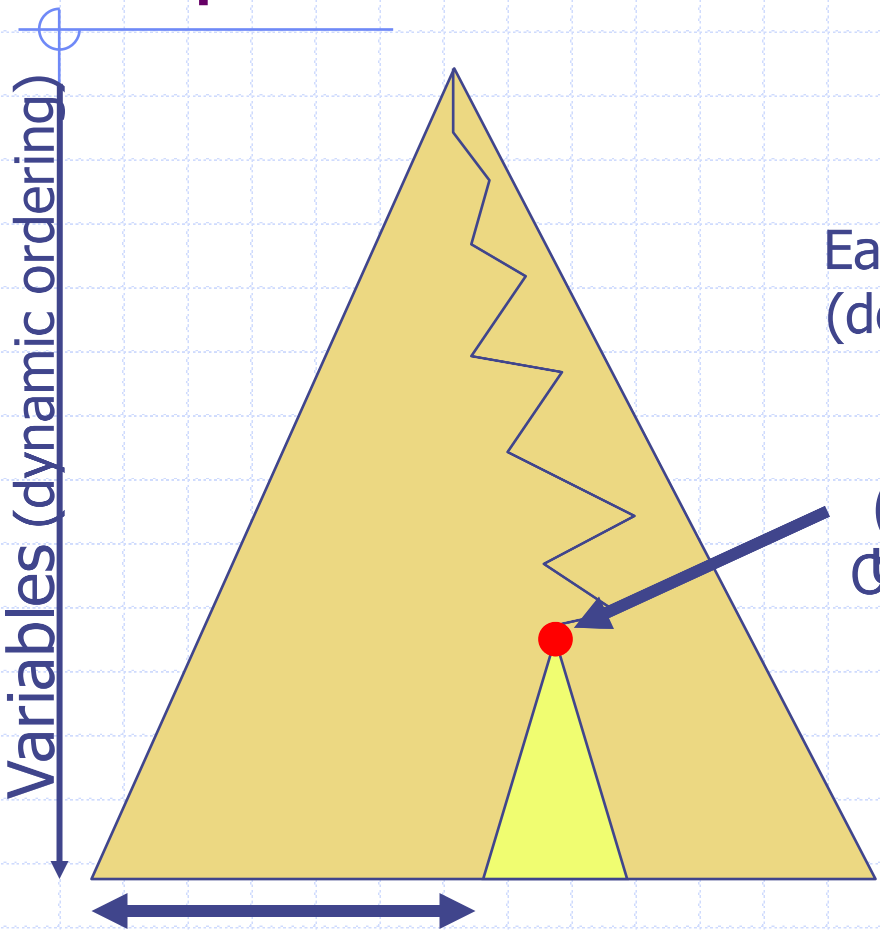
- For general background on tractable structures, see the chapter on "Tractable Structures" by Dechter, in the *Handbook of Constraint Programming*, Elsevier, 2006.

- For tractable valued constraint languages see "The complexity of soft constraint satisfaction", Cohen, Cooper, Jeavons & Krokhin, AIJ 2006.

Chapter 3. Search using problem transformations

Branch and Bound,
Equivalence-preserving operations,
Soft local consistency (node, arc, directional, virtual, optimal),
Soft global constraints.

Depth-First Branch and Bound (DFBB)



Each node is a VCSP subproblem
(defined by current conditioning)

(LB) Lower Bound = $c_{\varnothing}$

under estimation of the
best solution
in the sub-tree

Obtained by enforcing local consistency

If LB $c_{\varnothing}$ ≥ *k* then prune

(UB) Upper Bound
= best solution found so far = ***k***

Equivalence-preserving transformations (EPT)

- An EPT transforms VCSP instance P1 into another VCSP instance P2 with the same objective function.
- Examples of EPTs:

- Propagation of inconsistencies (∞ costs)

- UnaryProject

- Project/Extend

INCREMENTALITY!

UnaryProject(i,α)

Precondition: $0 \le \alpha \le \min\{c_i(a) : a \in d_i\}$

$c_0 := c_0 + \alpha$;
for all $a \in d_i$ **do**
  $c_i(a) := c_i(a) - \alpha$;

Increases the lower bound c_0 if all unary costs $c_i(a)$ are non-zero.

Project(M,i,a,α)

Precondition: $i \in M$, $a \in d_i$, $-c_i(a) \leq \alpha \leq \min\{c_M(x) : x[i]=a\}$

$c_i(a) := c_i(a) + \alpha$;
for all $x \in$ labelings(M) s.t. $x[i]=a$ **do**
    $c_M(x) := c_M(x) - \alpha$;

If $\alpha>0$, this projects costs from c_M to c_i

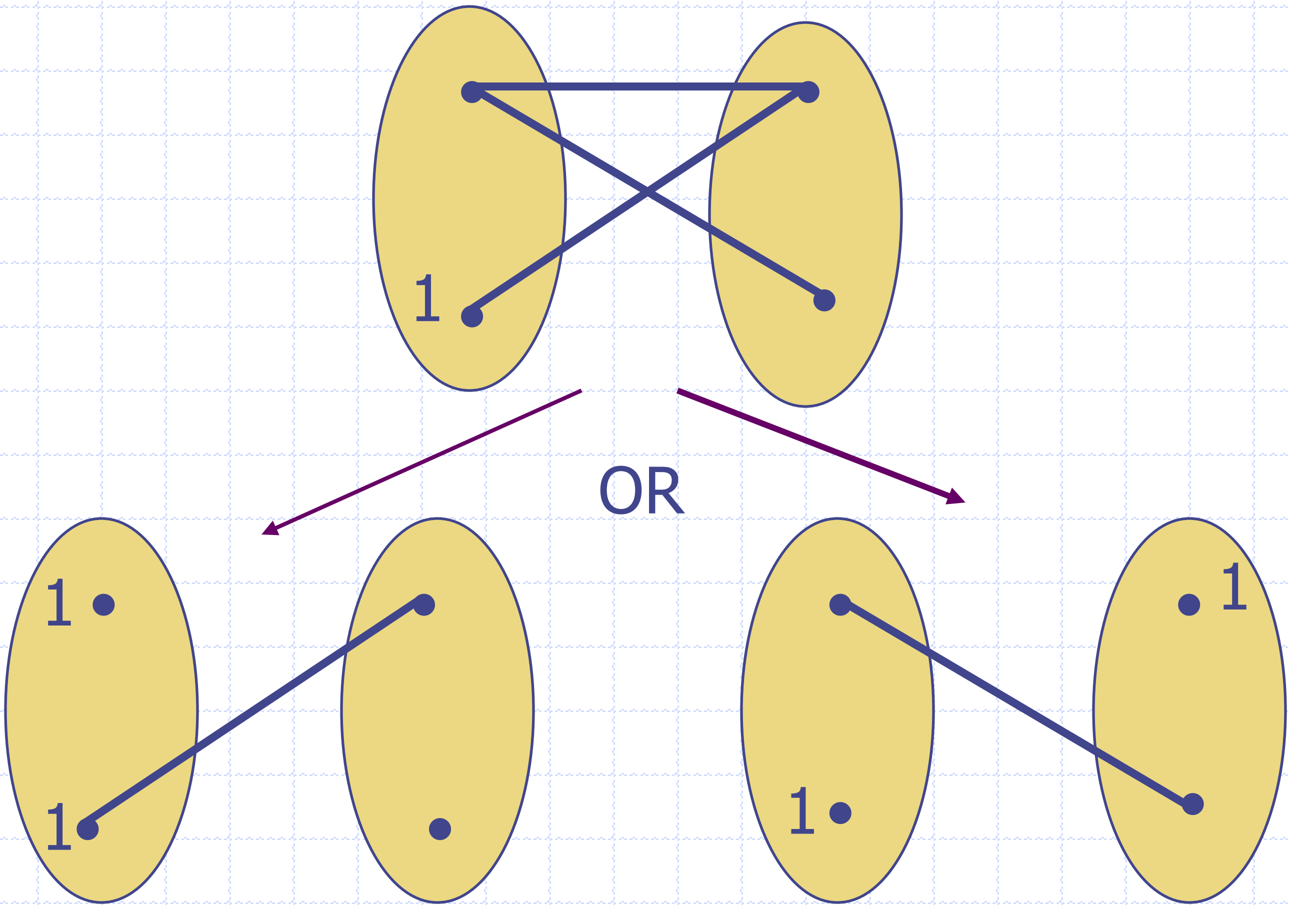
If $\alpha<0$, this extends costs from c_i to c_M

Node and soft arc consistency

- Node consistent (NC) if $\forall i$
  no UnaryProject(i,α) is possible for $\alpha>0$ and no propagation of ∞ costs possible between c_i and c_0 (forbidden values removed if $c_i+c_0 \geq k$)

- Soft arc consistent (SAC) if $\forall M,i,a$
  no Project(M,i,a,α) is possible for $\alpha>0$

The SAC closure is not unique



Different soft AC notions:

- **Directional:** send costs from X_j to X_i if $i<j$ (in the hope that this will increase c_0)
- **Existential:** $\forall i$, send costs to X_i simultaneously from its neighbor variables if this increases c_0
- **Virtual:** no *sequence* of Projects/Extends increases c_0
- **Optimal:** no *simultaneous set* of Projects/Extends increases c_0

Directional Arc Consistency

- for all $i<j$, $\forall a \in d_i\ \exists b \in d_j$ such that $c_{ij}(a,b) = c_j(b) = 0$.
- Solves tree-structured VCSPs
- FDAC (Full Directional AC) =
  Directional AC + Soft AC
- FDAC can be established in $O(end^3)$ time (or in $O(ed^2)$ time if $+_k$ is $+$)

Existential Arc Consistency

- node consistent and $\forall i$, $\exists a \in d_i$ such that $c_i(a) = 0$ and for all cost functions c_{ij} , $\exists b \in d_j$ such that $c_{ij}(a, b) = c_j(b) = 0$
- EDAC = Existential AC + FDAC
- EDAC can be established in

  $O(ed^2 \max\{nd,k\})$ time

Virtual Arc Consistency (VAC)

- If P is a VCSP instance then Bool(P) is the CSP instance whose allowed tuples are the zero-cost tuples in P-c_0
- If Bool(P) has a solution, then P has a solution of cost c_0 (but usually Bool(P) has no solution)
- Definition: P is VAC if Bool(P) is AC.

Approximating VAC

- If a sequence of AC operations in Bool(P) leads to a domain wipe-out, then a similar sequence of SAC operations in P increases c_0
- But, in this sequence, costs may need to be sent in more than one direction from the same c_M $\Rightarrow$ Introduction of *fractional* weights
- VACε algorithm may converge to a local minimum (and more, an instance P' which is *not* VAC)
- VACε can be established in $O(ed^2\ k/\varepsilon)$ time

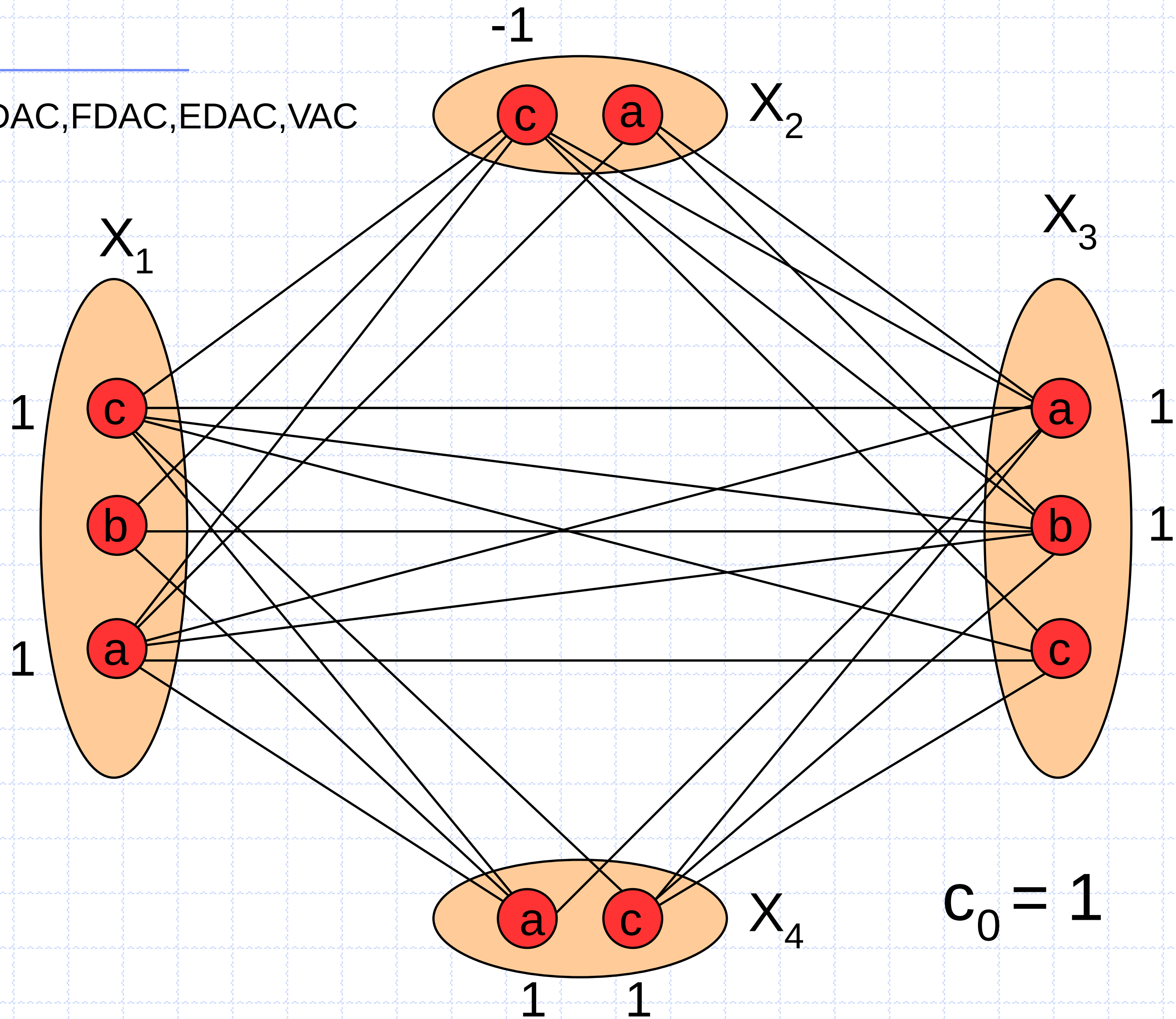
Optimal Soft Arc Consistency

- We can overcome this problem of convergence by solving a LP to find the *set* of simultaneous UnaryProject and Project operations which maximises c_0.
- The resulting VCSP instance is OSAC (Optimal Soft Arc Consistent).
- OSAC is strictly stronger than VAC.
- Unfortunately, the LP has O(edr+n) variables and O(ed^r+nd) constraints, and hence only useful for pre-processing.

$$p^{23}_{2c} = p^{34}_{3a} = p^{31}_{3b} = p^{12}_{1a} = p^{14}_{1c} = -1$$

$$p^{23}_{3a} = p^{23}_{3b} = p^{34}_{4c} = p^{31}_{1a} = p^{31}_{1c} = p^{12}_{2c} = p^{14}_{4a} = u_4 = 1$$

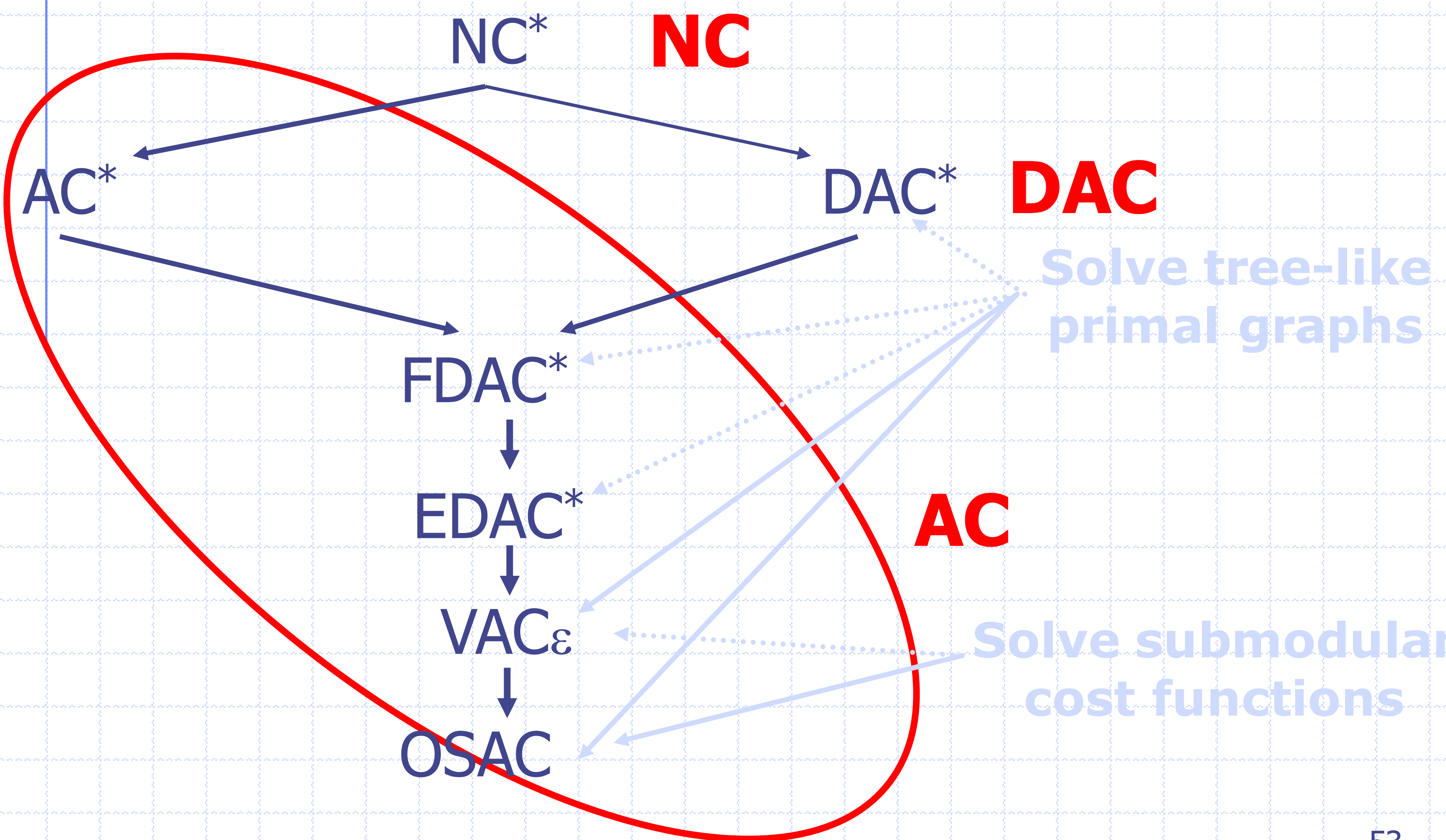
AC,DAC,FDAC,EDAC,VAC



$c_0 = 1$

Hierarchy

Special case: CSP (**k**=1)



Some practical observations

- For very hard-to-solve instances, maintaining VAC provides a significant speed-up, however for many problems, maintaining a simpler form of soft arc consistency (e.g. EDAC) is faster.
- Unary costs $c_i(a)$ and EAC value inform value and variable ordering heuristics

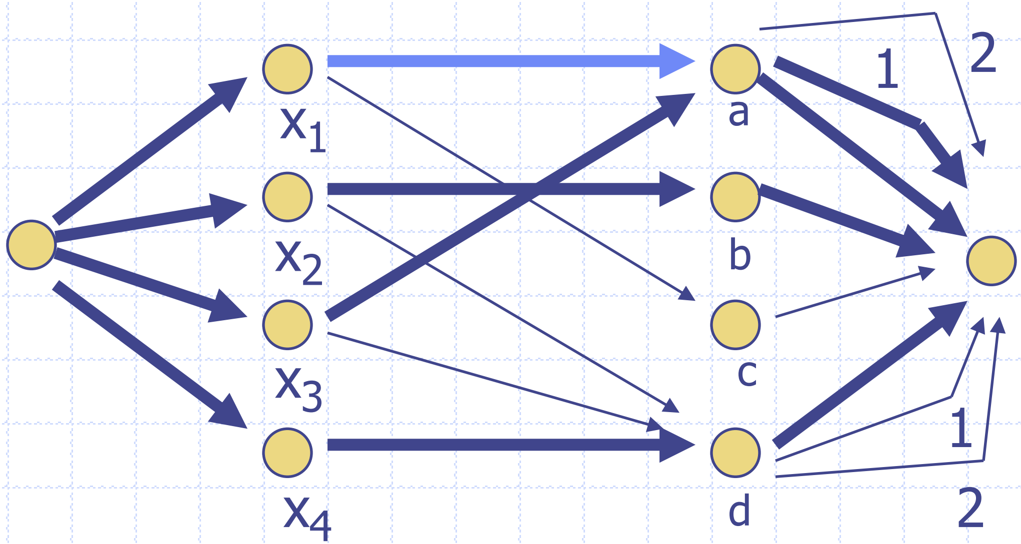
AC for soft global constraints

(van Hoeve et al, J. Heur. 2006) (Lee & Leung, IJCAI'09)

- Suppose that a global cost function c_M can be coded as the minimum cost of a maximum flow in a network in which (a) there is a one-to-one correspondence between max-flows and global labellings and (b) each assignment (x_i,a) has a corresponding edge e_{ia} such that the max-flow is 1 in e_{ia} if $x_i=a$ (0 if $x_i \neq a$).
- Then it is possible to project α from c_M to $c_i(a)$ by reducing cost(e_{ia}) by α.

Network representing soft Alldiff

- Min number of variables with same value
  variable-based costs (Beldiceanu & Petit, CPAIOR'04)



All edge capacities are equal to 1

All edge costs are 0 if not indicated

The flow shown is a min-cost max-flow with x_1=a.
We can project 1 from c_M to $c_1(a)$ by reducing the cost of the blue edge from 0 to –1.

Latin Square N x N with costs

Example of solution for N = 5:

2	1	3	5	4
4	2	1	3	5
1	5	4	2	3
5	3	2	4	1
3	4	5	1	2

All variables take a different value in each row and each column

A unary cost function for each cell $f^{i,j}(x_{i,j}) : D \rightarrow [0, MaxCost[$

Objective: 49

Objective = $\sum_i \sum_j f^{i,j}(x_{i,j})$

Latin Square with costs in CHOCO

```
//1- Create the model
int n = 4;
int maxcost = 10;
CPModel m = new CPModel();
//2- Create the variables
IntegerVariable[] nvars = makeIntVarArray("Q", n * n, 1, n);
IntegerVariable[] costvars = makeIntVarArray("C", n * n, 0, maxcost-1);
IntegerVariable obj = makeIntVar("O", 0, (maxcost-1)*n*n, Options.V_OBJECTIVE);
int[][][] costs = new int[n][n][n];
//3- Create the random unary costs
Random rand = new Random();
for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                        costs[i][j][k] = rand.nextInt(maxcost);}}}
```

Latin Square with costs in CHOCO

```
//4- Post constraints
for (int i = 0; i < n; i++) {
        IntegerVariable[] line = new IntegerVariable[n];
        IntegerVariable[] column = new IntegerVariable[n];
        for (int j = 0; j < n; j++) {
                row[j] = nvars[i*n +j];
                column[j] = nvars[i+j*n];
                m.addConstraint(Options.C_NTH_G, nth(nvars[i*n+j], costs[i][j], costvars[i*n+j], 0));
        }
        m.addConstraint(allDifferent(row));
        m.addConstraint(allDifferent(column));
}
m.addConstraint(eq(sum(costvars), obj));
//5- Create the solver
Solver s = new CPSolver();
s.read(m);
s.minimize(false);
```

Latin Square with costs in toulbar2

pbname, n, d, e, k

```
latin4 16 4 24 145
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 0 1 2 3 -1 salldiff var 1000000
4 4 5 6 7 -1 salldiff var 1000000
4 8 9 10 11 -1 salldiff var 1000000
4 12 13 14 15 -1 salldiff var 1000000
4 0 4 8 12 -1 salldiff var 1000000
4 1 5 9 13 -1 salldiff var 1000000
4 2 6 10 14 -1 salldiff var 1000000
4 3 7 11 15 -1 salldiff var 1000000
1 0 0 4
0 0
1 3
2 9
3 1
```

domain sizes

hard AllDifferent on rows

hard AllDifferent on columns

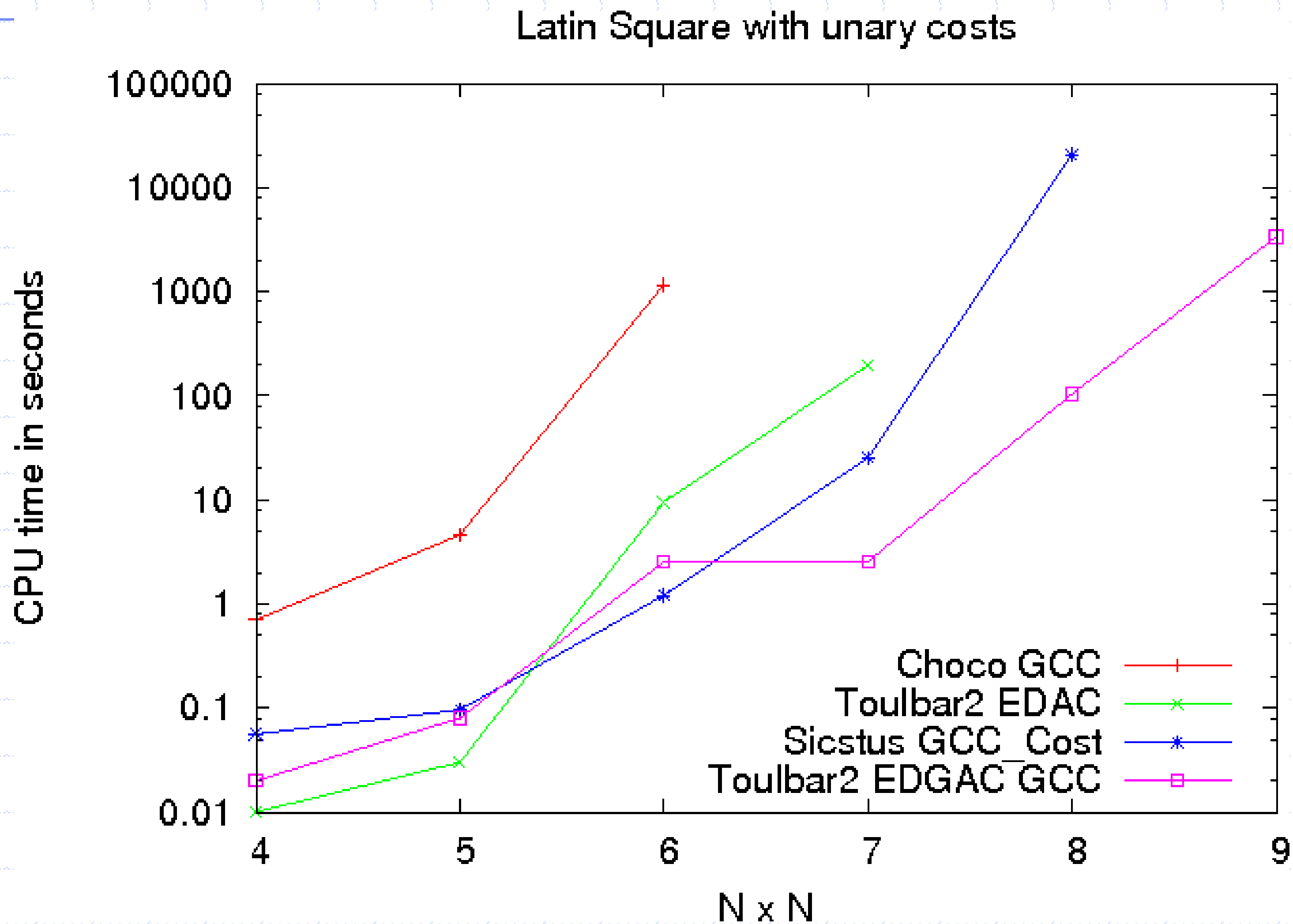
#vars, scope, defcost, #tuples

unary cost functions

value,cost

GCC_Cost (Régin, *Constraints* 2002)
EDGAC (Lee & Leung, *AAAI* 2010)

Latin Square with costs



choco v2.1.1, toulbar2 v0.9.3, sicstus v4.1.2 on linux PC 2.66 Ghz 64GB

Bibliography

- For an overview of soft local consistencies, see "*Soft arc consistency revisited*", Cooper, de Givry, Sanchez, Schiex, Zytnicki & Werner, AIJ 2010.

- For soft global constraints (FDGAC), see "*Towards Efficient Consistency Enforcement for Global Constraints in Weighted Constraint Satisfaction*", Lee & Leung, IJCAI 2009.

Chapter 4. Search exploiting the problem structure

BB-VE(2), BTD, AND/OR search

Solving methods

BTD, AND/OR graph search

Time: exp(treewidth)
Space: exp(treewidth)

Search: Conditioning

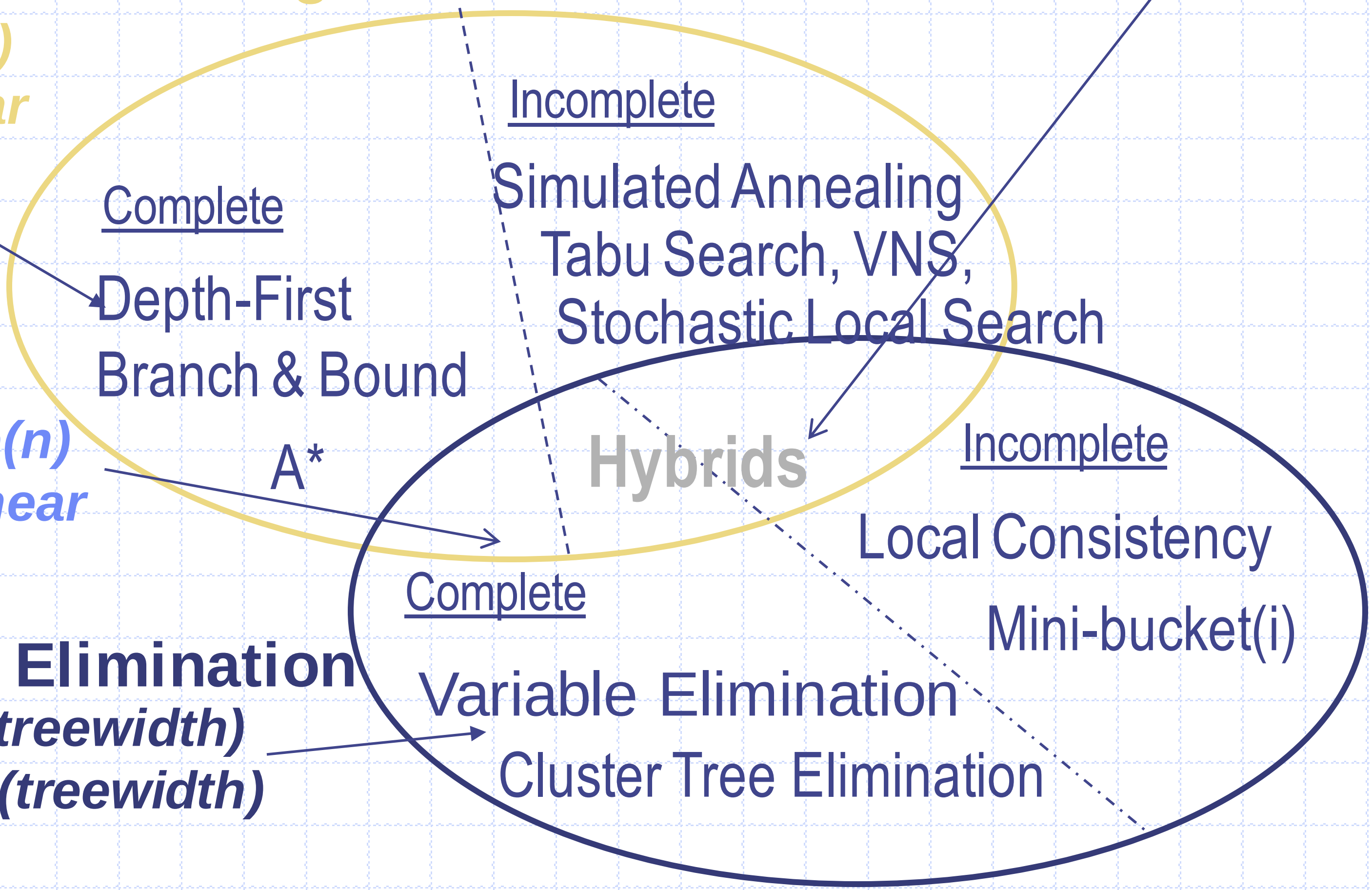
Time: exp(n)
Space: linear

BB-VE

Time: exp(n)
Space: linear

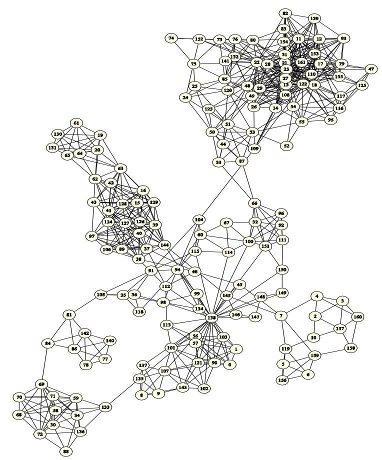
Inference: Elimination

Time: exp(treewidth)
Space:exp(treewidth)



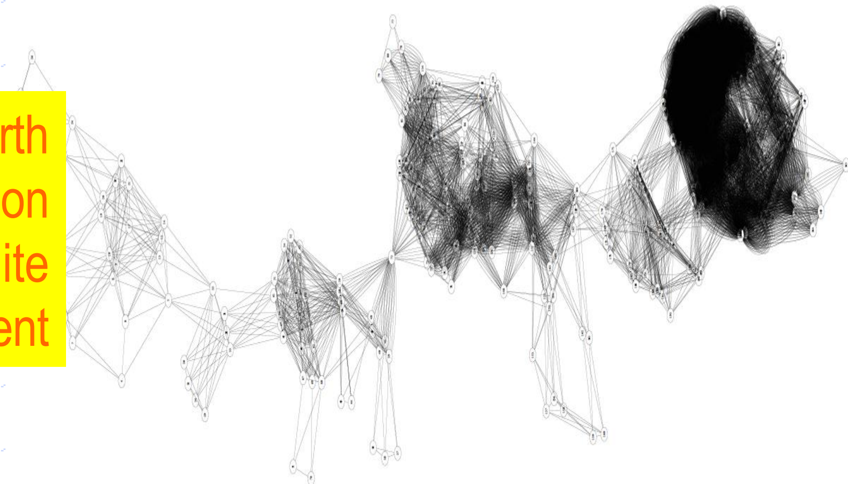
Many real applications have a structured network

Radio Link Frequency Assignment



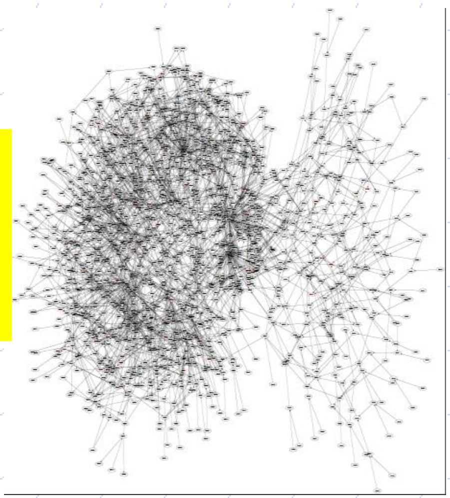
CELAR SCEN-07r
(*Constraints* 4(1), 1999)

Earth Observation Satellite Management



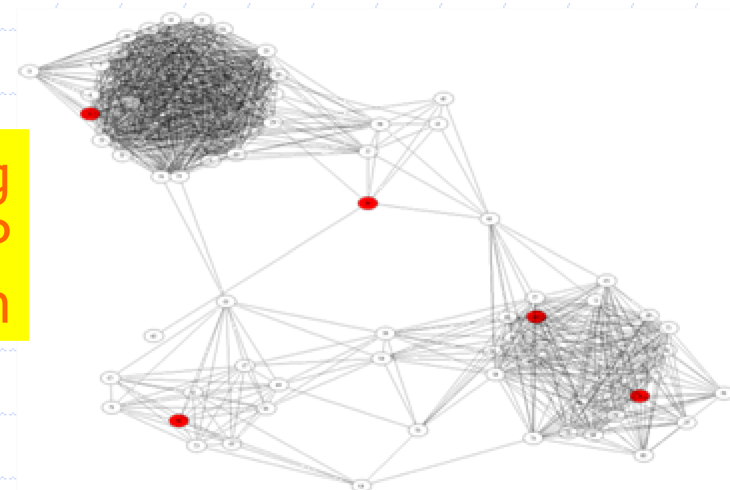
SPOT5 #509 (*Constraints* 4(3), 1999)

Mendelian Error Detection



langladeM7 sheep pedigree
(*Constraints* 13(1), 2008)

Tag SNP Selection



HapMap chr01 $r^2 \geq 0.8$ #14481
(*Bioinformatics* 22(2), 2006)

CELAR 6 results since 1993

n. of vars: n=100, domain size: d=44, n. of cost functions: e=1222

Time of optimality proof	Method(s) used	Publication
26 days (SUN UltraSparc 167 MHz)	Ad-hoc problem decomposition & Russian Doll Search *(22 vars only)*	(de Givry, Verfaillie, Schiex, CP 1997)
3 days (SUN Sparc 2)	Ad-hoc problem decomposition & PFC-MRDAC *(22 vars only)*	(Larrosa, Meseguer, Schiex, AIJ 1998)
8 hours (DEC Alpha 500MP)	Preprocessing rules & Cluster Tree Elimination	(Koster PhD thesis, 1999)
3 hours (PC 2.4 GHz)	B&B with EDAC & tree decomposition (BTD)	(de Givry, Schiex, Verfaillie, AAAI 2006)
3 minutes (PC 2.6 GHz)	BTD & variable ordering heuristisc &dicho branching	(Sanchez, Allouche, de Givry,Schiex, IJCAI 2009)

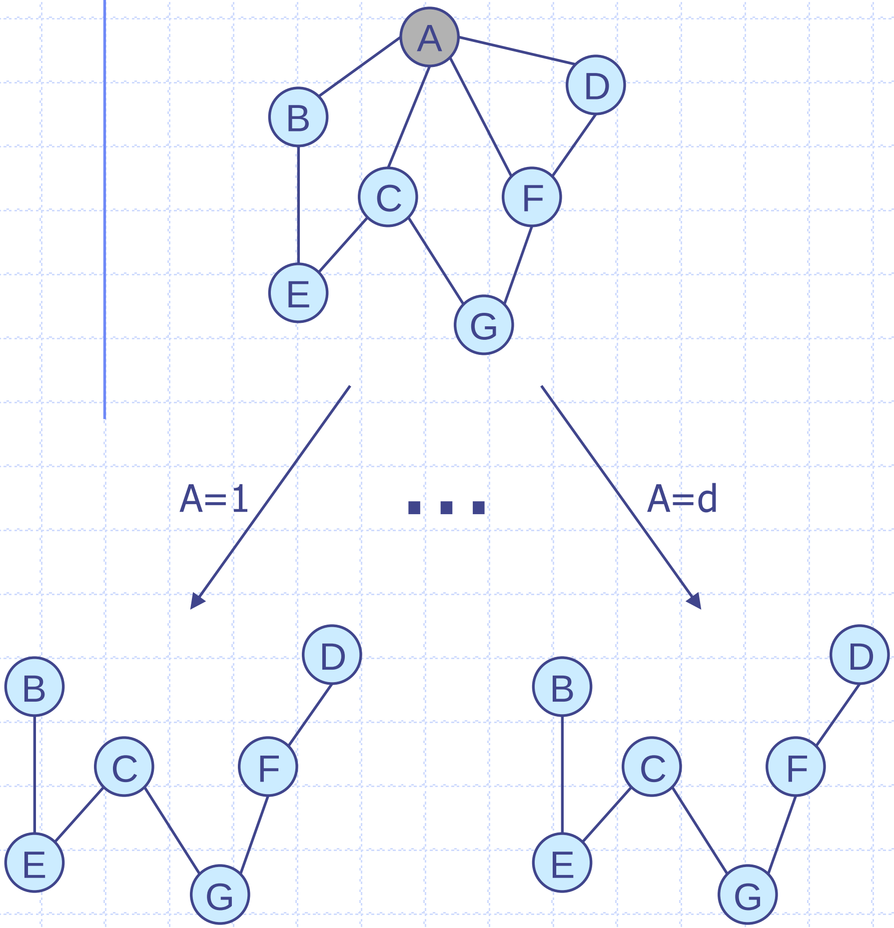
CELAR 7 (n=200) solved in 4.5 days (Sanchez et al, IJCAI 2009)

CELAR 8 (n=458) solved in 127 days (Allouche et al, CP 2010)

All CELAR and GRAPH instances are closed!

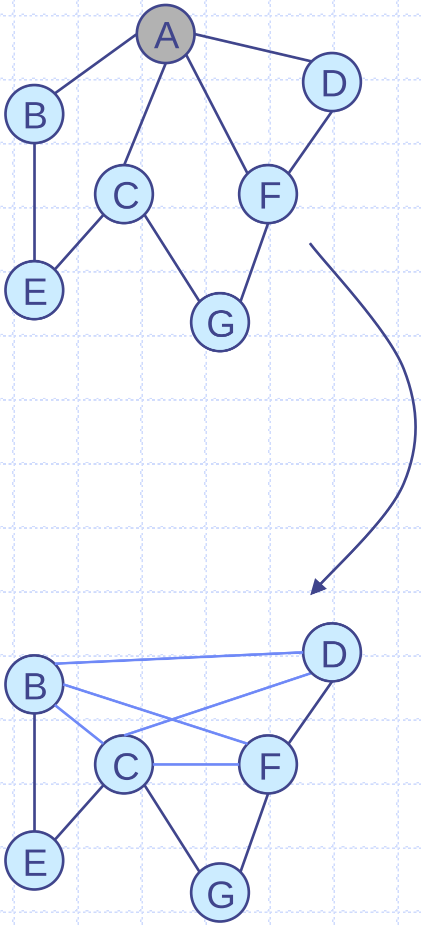
Conditioning vs. Elimination

Conditioning (search)



d "sparser" problems

Elimination (inference)

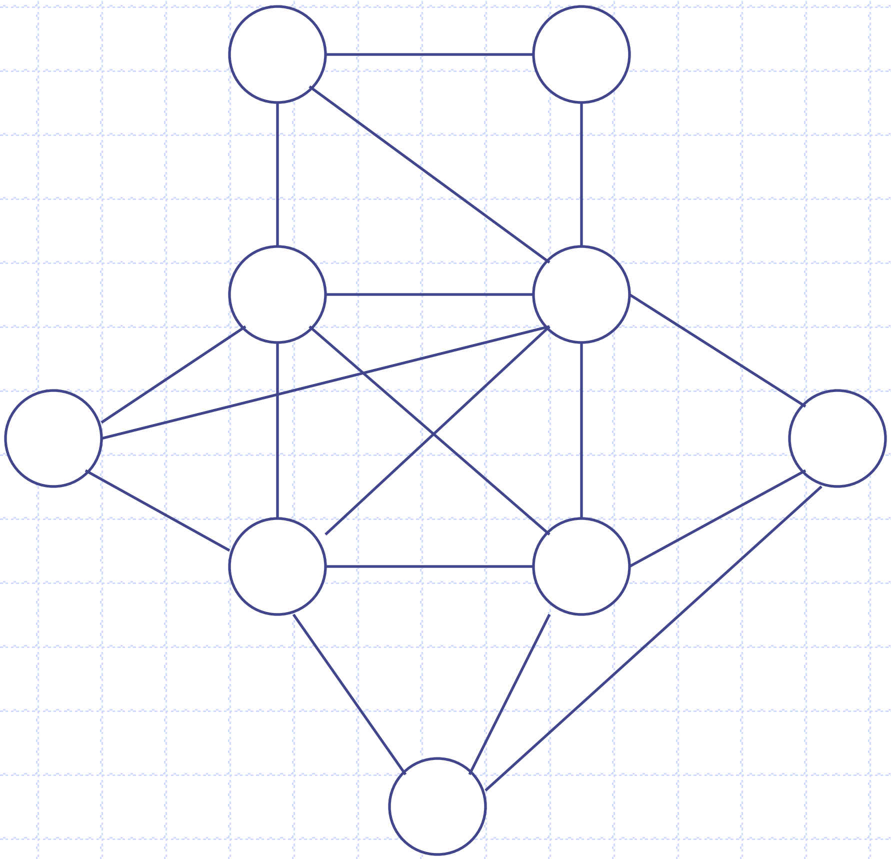


1 "denser" problem

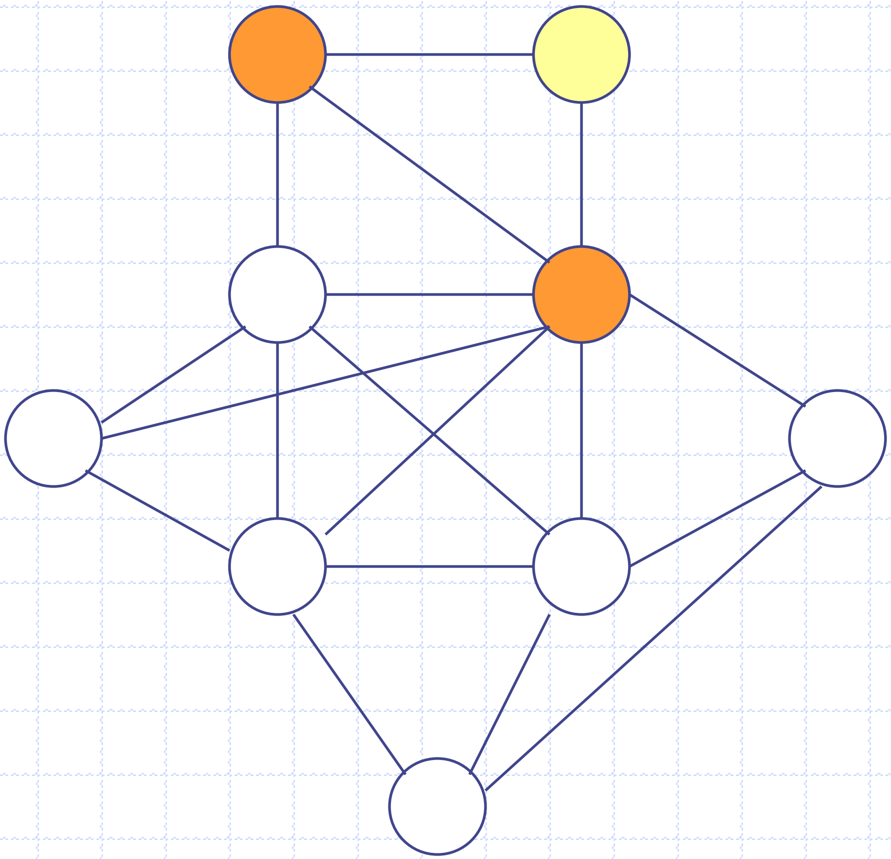
First hybrids: Search & Variable Elimination

- Condition, condition, condition … and then only eliminate (*Cycle-Cutset*)
- Eliminate, eliminate, eliminate … and then only search
- Interleave conditioning and elimination

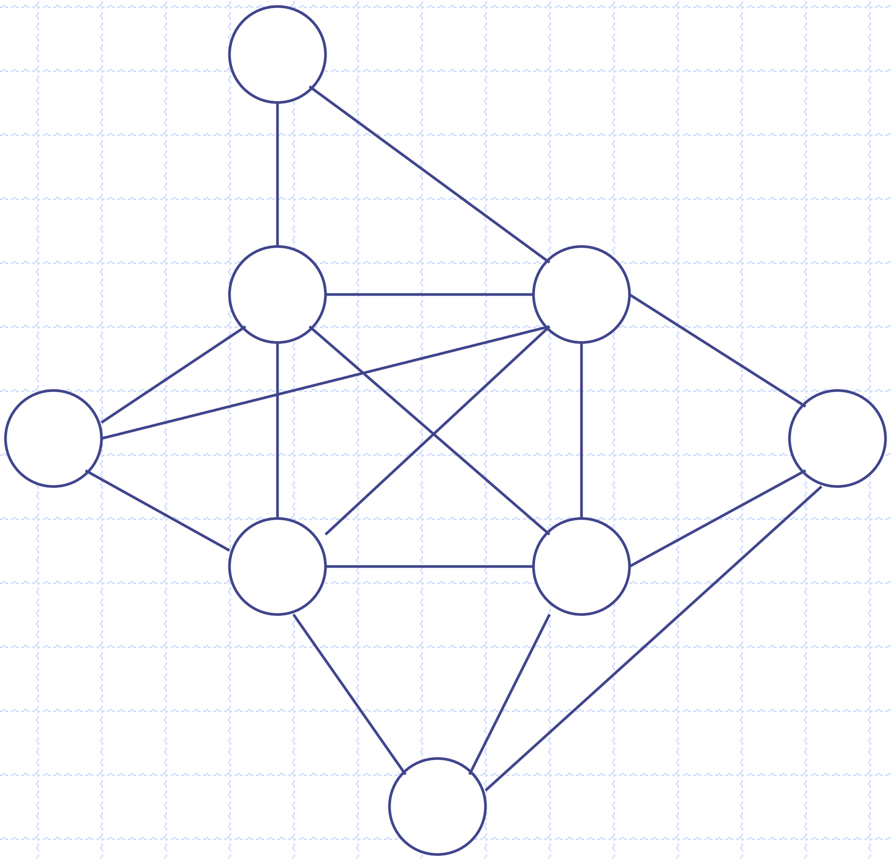
Interleaving Conditioning and Elimination BB-VE(2) (Larrosa & Dechter, CP 2002)



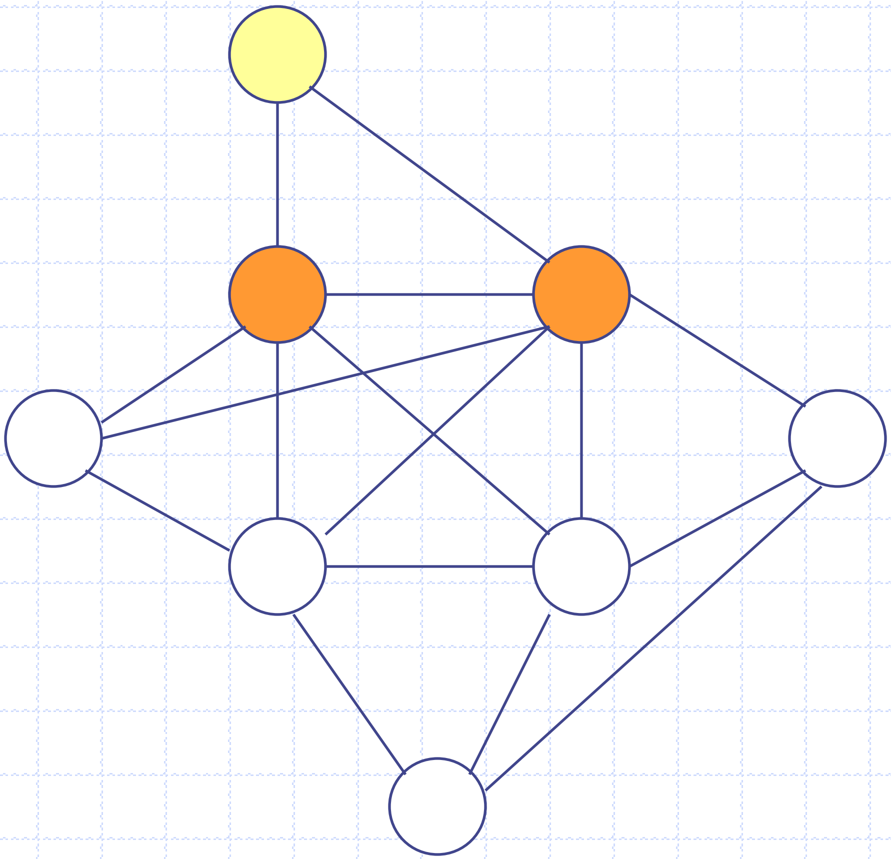
Interleaving Conditioning and Elimination BB-VE(2)



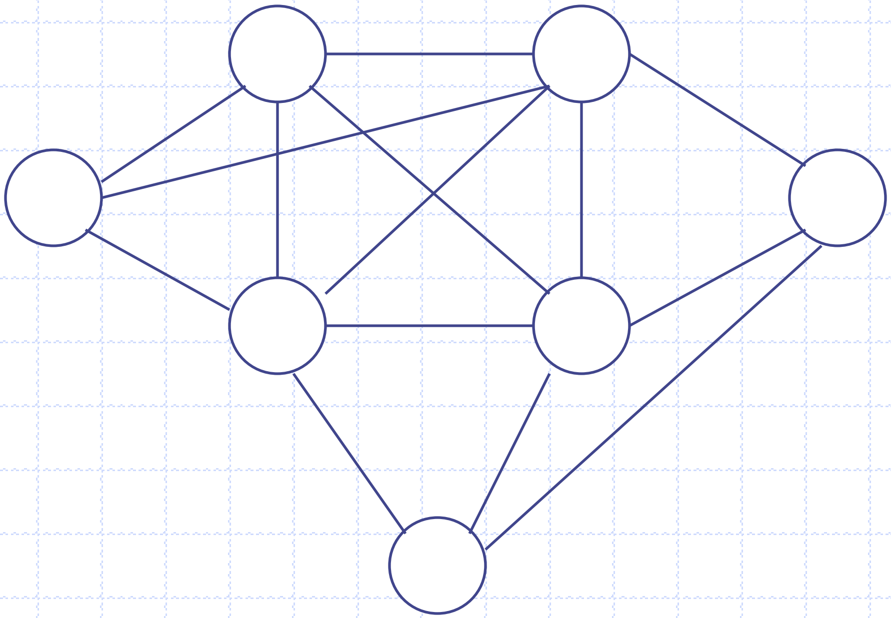
Interleaving Conditioning and Elimination BB-VE(2)



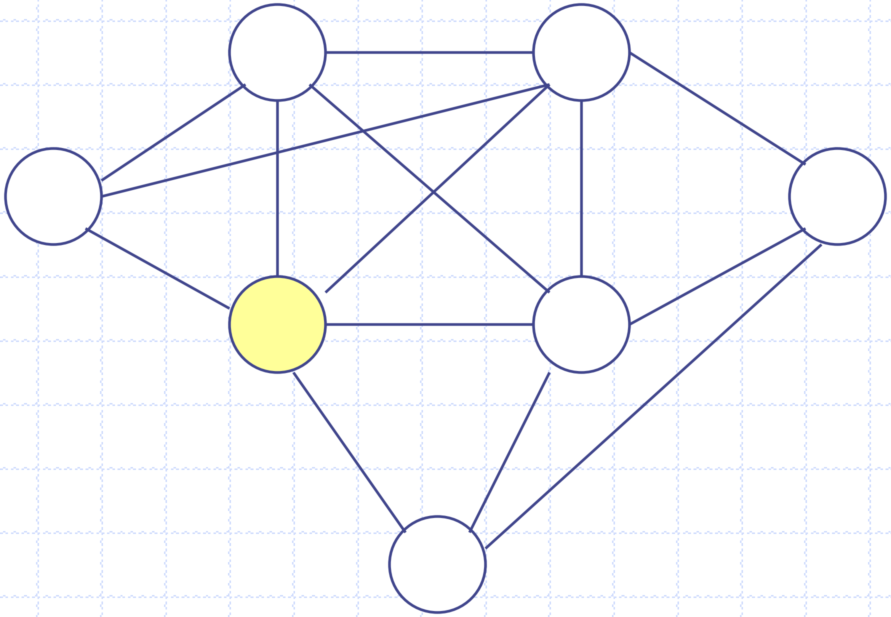
Interleaving Conditioning and Elimination BB-VE(2)



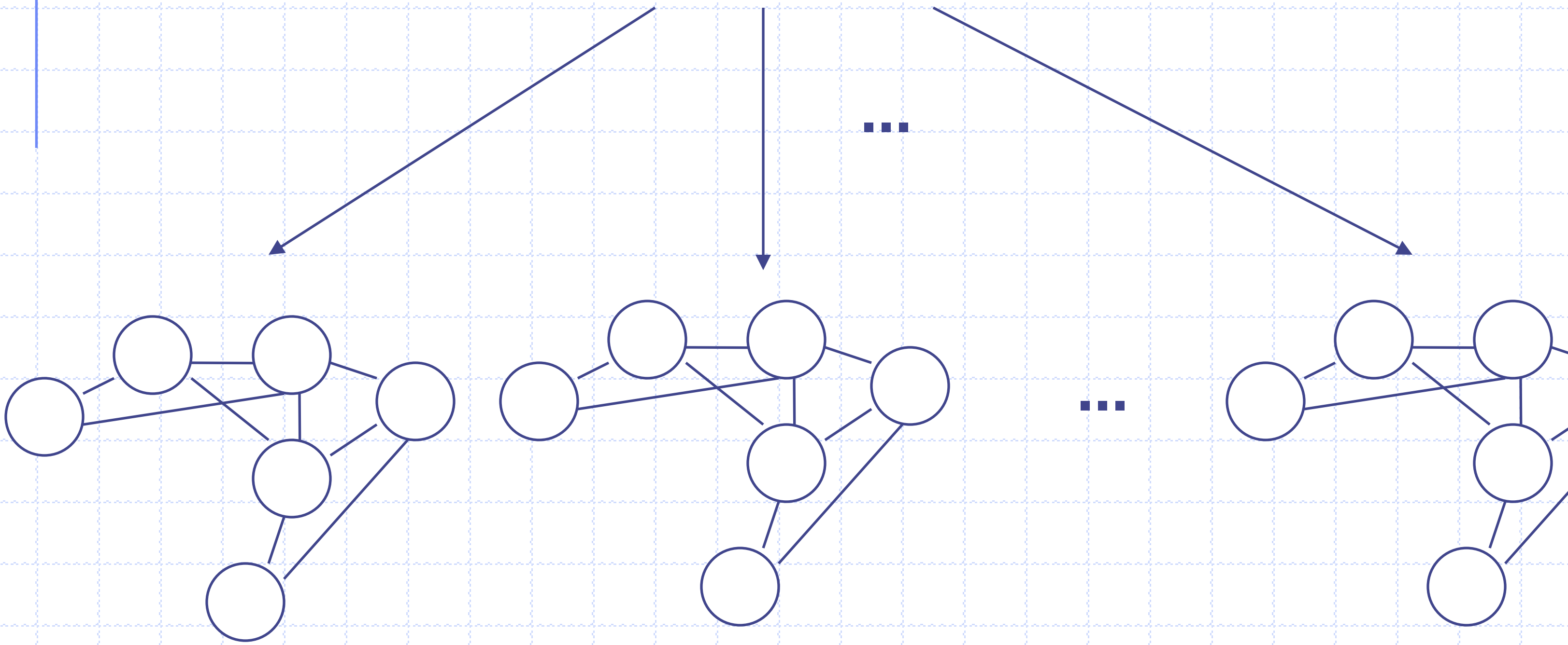
Interleaving Conditioning and Elimination BB-VE(2)



Interleaving Conditioning and Elimination BB-VE(2)



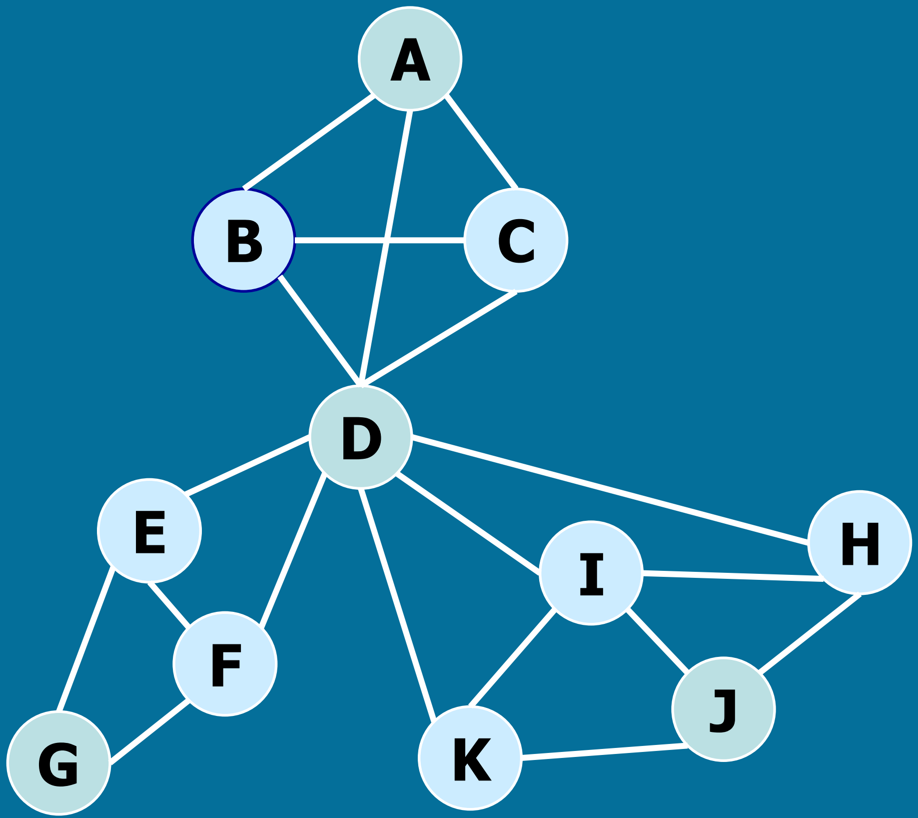
Interleaving Conditioning and Elimination BB-VE(2)



Second hybrids: Search & Cluster Tree Elimination

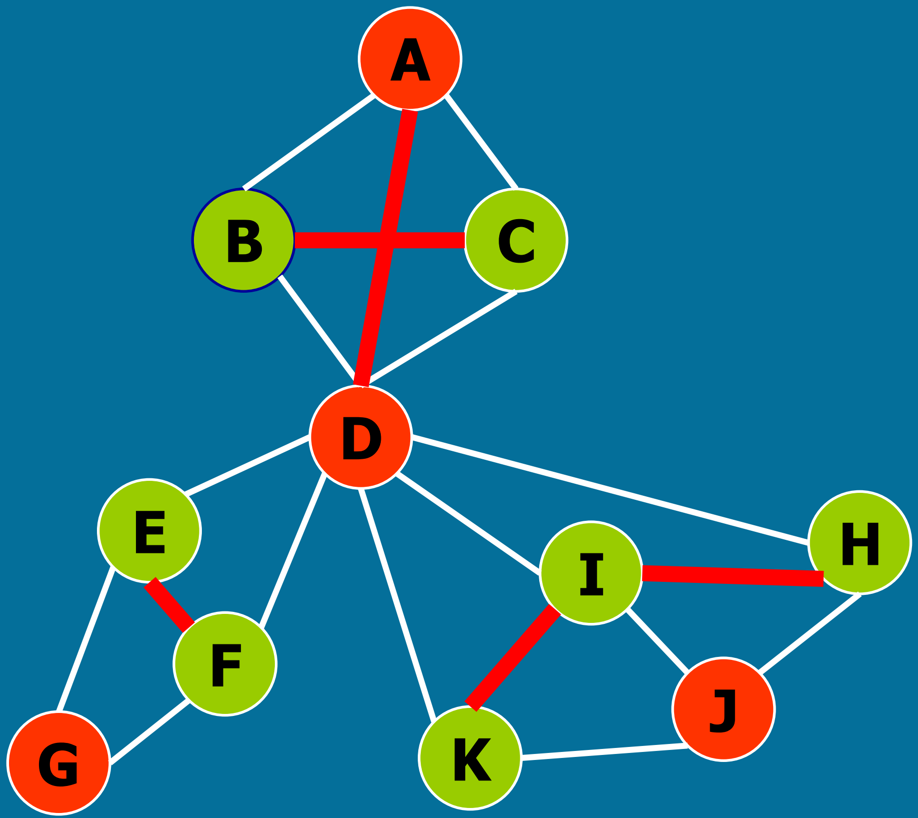
- Depth-First Branch and Bound exploiting a tree decomposition with:
  - A restricted variable ordering
  - Graph-based backjumping
  - Graph-based learning
  - ⇒ Lazy elimination of subproblems using search

Soft 2-Coloring example



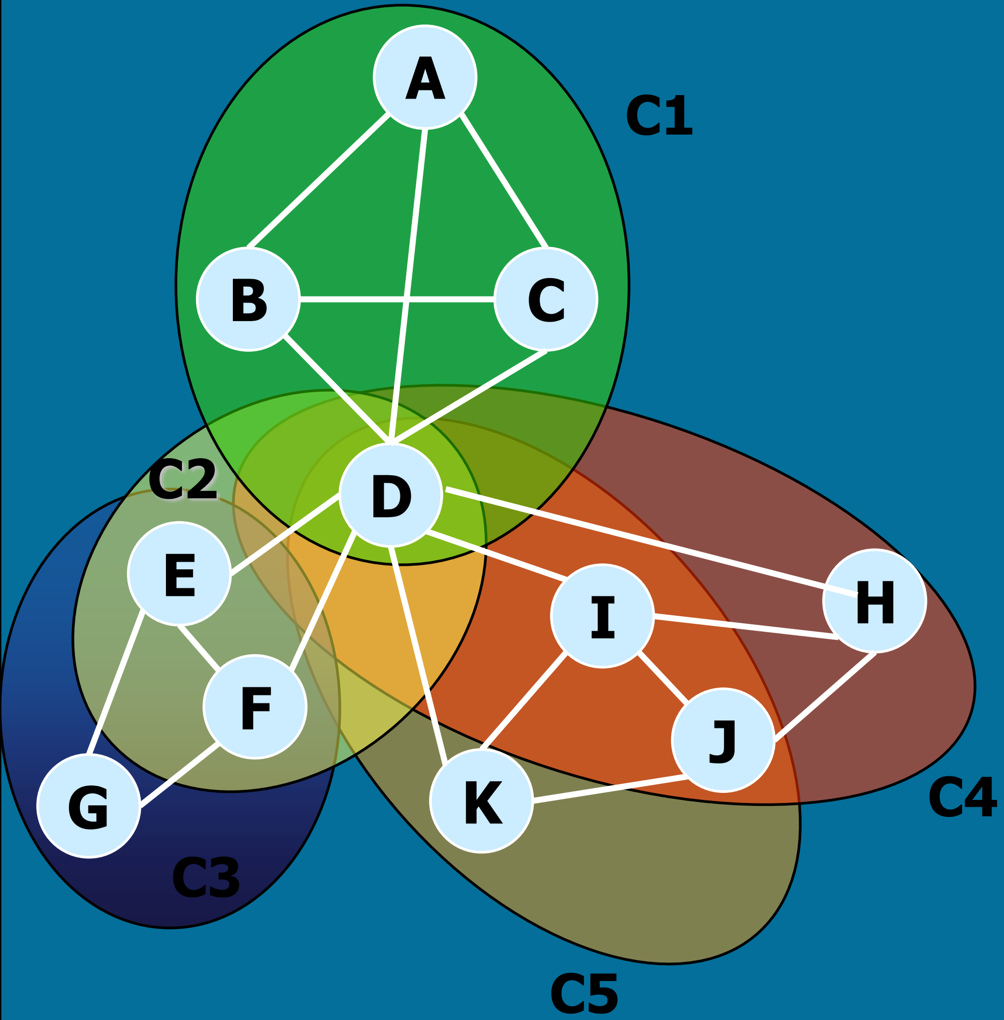
x_i	x_j	c_{ij}
red	red	1
red	green	0
green	red	0
green	green	1

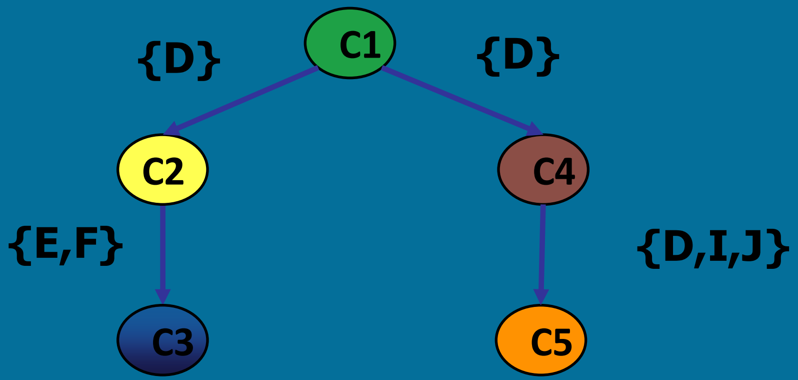
Soft 2-Coloring example



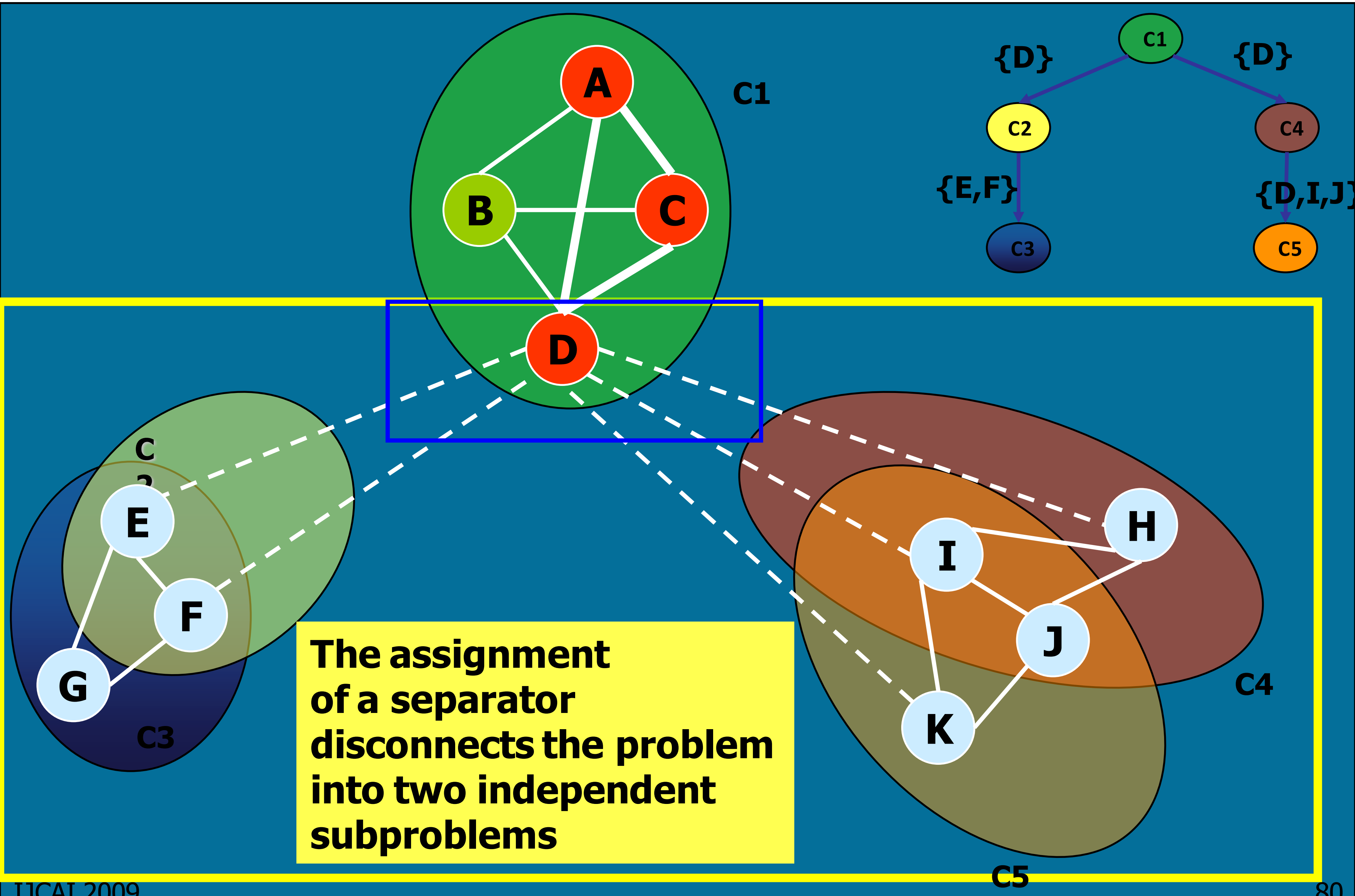
Optimal solution with a cost of 5

Tree Decomposition



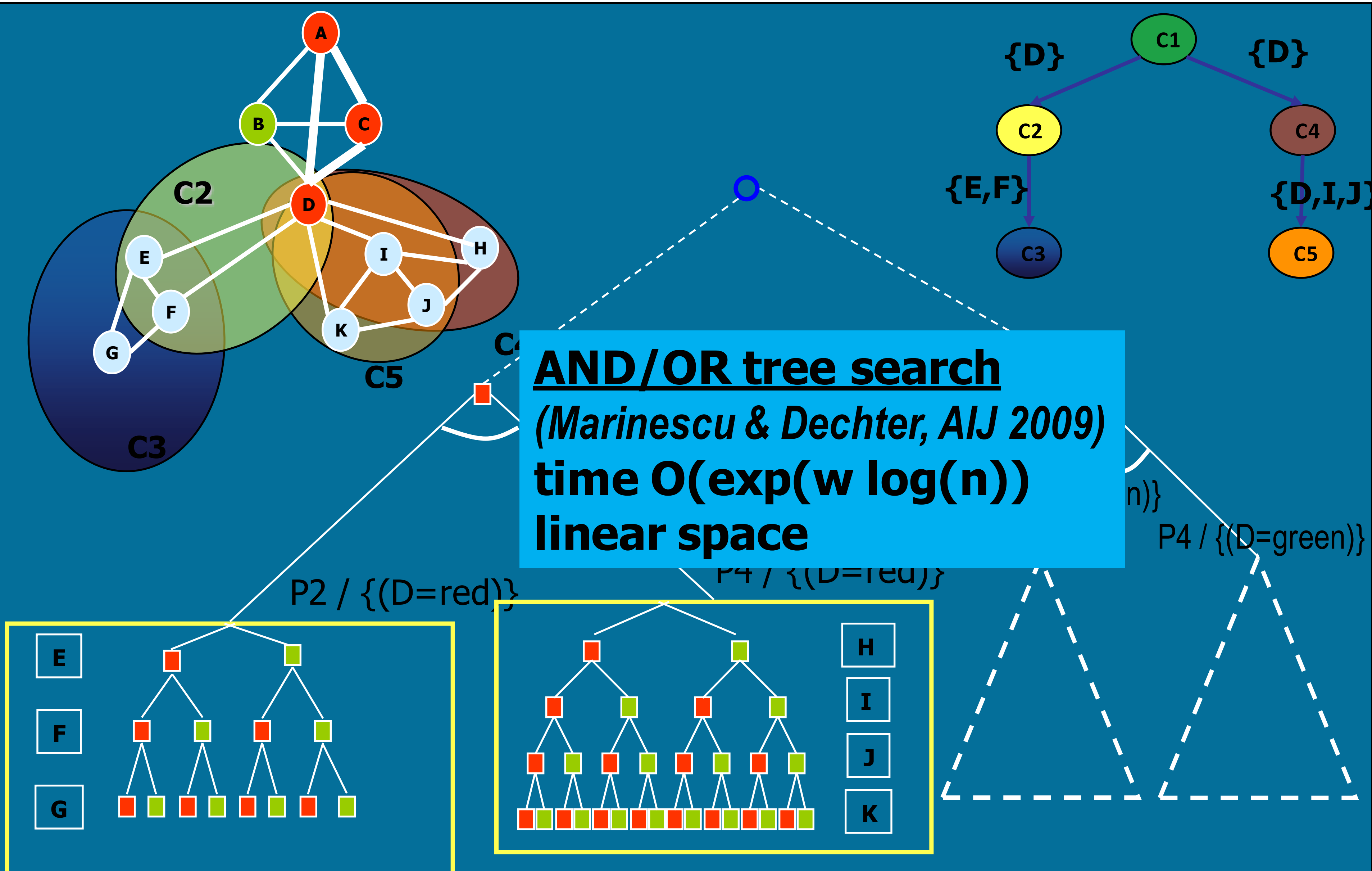


Search with Tree Decomposition



The assignment of a separator disconnects the problem into two independent subproblems

Search with Tree Decomposition



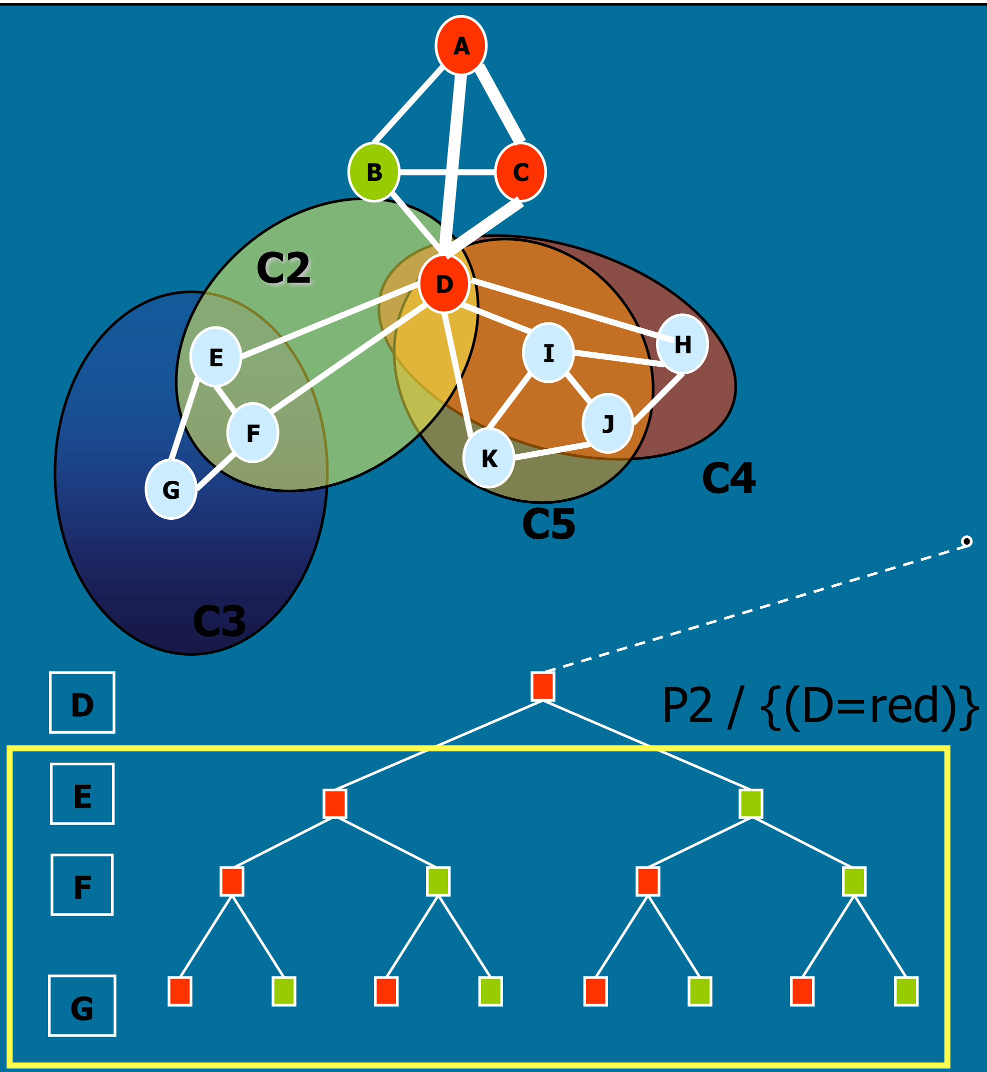
<u>AND/OR tree search</u>

(Marinescu & Dechter, AIJ 2009)

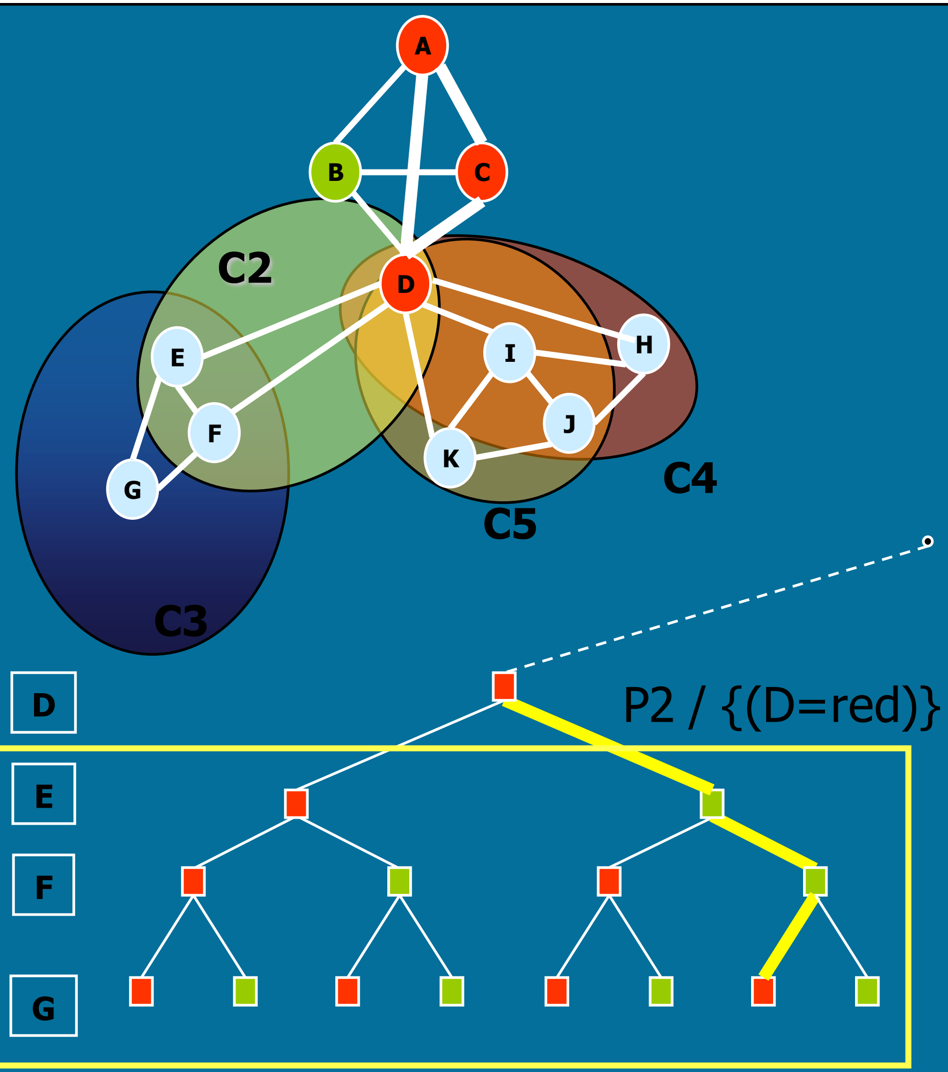
time O(exp(w log(n))

linear space

Search with Tree Decomposition



Search with Tree Decomposition



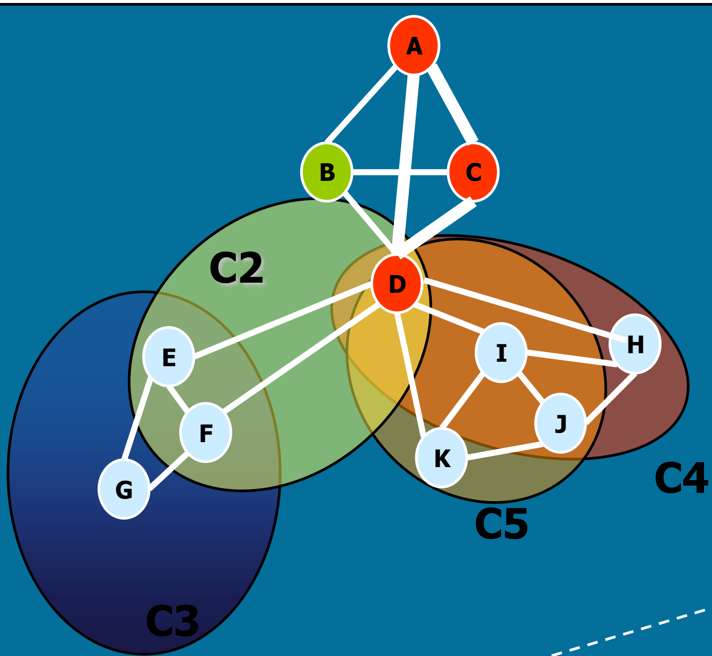
Record the optimum of P2 / {(D=red)}

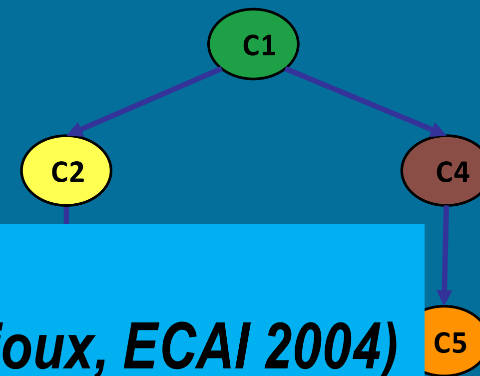
<u>AND/OR graph search</u>
(Marinescu & Dechter, AIJ 2009)
time O(exp(w))
space O(exp(w))

bound k = 5.

It may be useless to compute the optimum of P2 / {(D=red)}, only a lower bound is needed!

Backtrack bounded by Tree Decomposition





BTD

(Jégou & Terrioux, ECAI 2004)

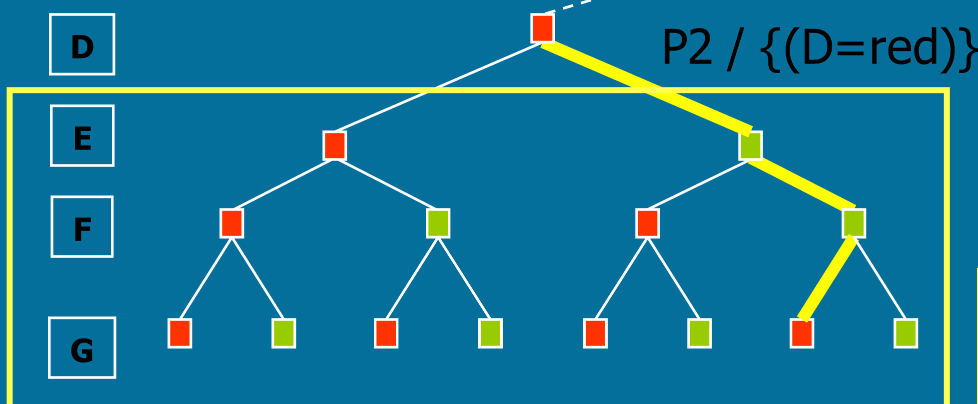
(de Givry et al., AAAI 2006)

time O(k*exp(w))

space O(exp(w))

per

It may be useless to compute the optimum of P2, only a lower bound is needed!



Add a local upper bound:

$UB_{P2\,/\,\{(D=red)\}} = k - 3 - LB_{P4\,/\,\{(D=red)\}}$

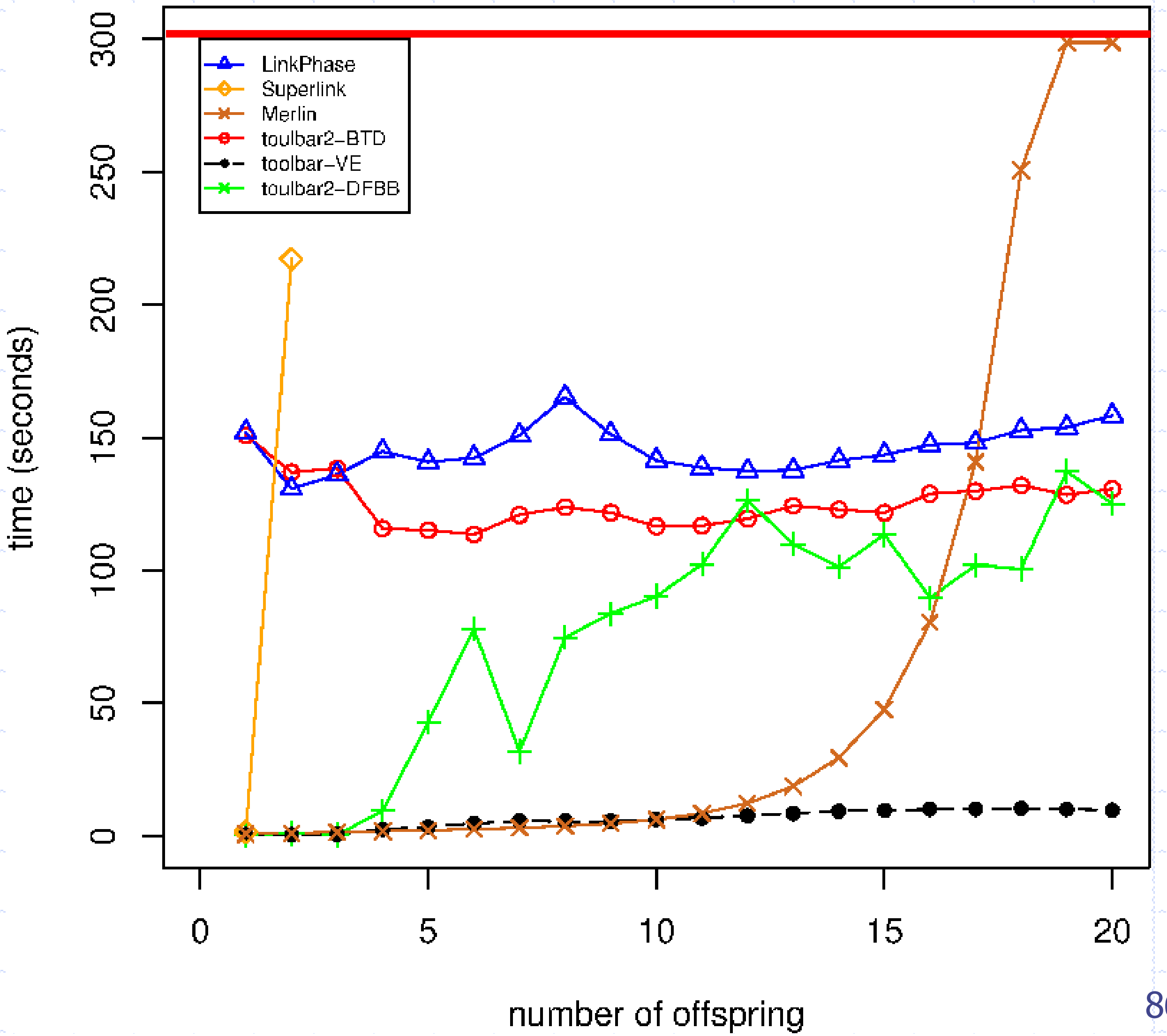
$$UB_{P2\,/\,\{(D=red)\}} = k - 3 - \max\left(\underbrace{c_{\varnothing}^{C4} + c_{\varnothing}^{C5}}_{\text{Maintaining local consistency}},\ \underbrace{LB_{P4\,/\,\{(D=red)\}}}_{\text{Recorded during search}} \right)$$

Some practical observations

- BTD may use much less memory than Variable Elimination thanks to pruning
- Impact of root cluster
  - ⇒ Choose the largest / most costly cluster as root
- Exploit small separators only (Jégou, Ndiaye & Terrioux, CP 2007)
  - ⇒ Give more freedom for the dynamic variable ordering heuristic
  - ⇒ Tuning based on treewidth versus separator size
- BB-VE(2) often faster than BTD

Haplotype reconstruction in half-sib pedigrees (Favier et al, WCB'10)

Read dataset HAPMAP Chr.X with 36,000 markers treewidth < 15



Bibliography

- For hybrids of search and inference, see the chapter 10 in *Constraint Processing*, Dechter, Morgan Kaufmann, 2003.
- For exploiting tree decomposition, see
  - "*Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP*", de Givry, Schiex & Verfaillie , AAAI 2006.
  - "*Memory intensive AND/OR search for combinatorial optimization in graphical models (Part I&II)*", Marinescu & Dechter, AIJ 2009.

http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP

Applications / benchmarks

toulbar2, aolib

- Resource Allocation
  - Frequency assignment (Allouche et al, CP 2010) CTE, BTD, VAC
    $n \leq 458$, $d=44$, $e(2) \leq 5{,}000$
  - Satellite management (Verfaillie et al, AAAI 1996) RDS, RDS-BTD
    $n \leq 364$, $d=4$, $e(2\text{-}3) \leq 10{,}108$
  - Uncapacitated warehouse location (Zytnicki et al, IJCAI 2005) EDAC,VAC, *ILP0/1*
    $n \leq 1{,}100$, $d \leq 300$, $e(2)$ 100,000
- Bioinformatics
  - Genetic linkage analysis (Marinescu & Dechter, AAAI 2006) AND/ORsearch
    $n \leq 1{,}200$, $d \leq 7$, $e(2\text{-}5) \leq 2{,}000$
  - Mendelian error detection (Sanchez et al, Constraints 2008) EDAC3,BB-VE
    $n \leq 20{,}000$, $d \leq 66$, $e(3) \leq 30{,}000$
  - RNA gene finding (Zytnicki et al, Constraints 2008) BAC
    $n \approx 20$, $d > 100$ million, $e(4) \approx 10$
  - Tag SNP selection (Sanchez et al, IJCAI 2009) RDS-BTD, *ILP0/1*
    $n \leq 1{,}500$, $d \leq 266$, $e(2) \leq 150{,}000$

Chapter 5. Open problems

Concerning problem definition, search, transformations, tractable classes

Possible extensions to VCSP

- Partial order instead of total order
- 2 arbitrary binary operators (e.g. calculating the sum of products instead of the min of the sum subsumes #CSP)
- Objective function not constructible using a binary aggregation operator (e.g. the median of the set of costs)

Tractability

- Can we characterize/unify all tractable classes of VCSP over non-Boolean domains?
- Are there interesting tractable classes apart from submodular functions?
- Are there more efficient algorithms for submodular function minimisation?

New search methods

- Identify and exploit *good* tree decompositions automatically
  - *Small treewidth versus small separators*
  - *Dynamic tree decomposition*
- Variable and value ordering heuristics
- Incomplete search strategies
  - *Large Neighbourhood Search,...*
- Parallel B & B / CTE methods

New problem transformations

- Applying rules involving ≥ 2 constraints
- Transformations which preserve

  at least one solution (if it exists)

  but do not necessarily preserve costs.
- Decomposition into several problems whose sum is equal to the original VCSP

Conclusion

- VCSP combines CSP and optimisation in a unified way
- Many CSP notions have been extended to the VCSP framework (consistency, global constraints, expressibility, tractability,…)
- Technology is usable and useful, but still maturing