Soft constraints: Algorithms (1)

T. Schiex

INRA - Toulouse, France

Solving soft CSP

Traditional queries:

- compute the cost of an optimal (non dominated) solution;
- find one/all optimal (non dominated) solutions;
- find a sufficiently good solution (cost less than k);
- prove that a given value/tuple is not used in any (optimal) solution;
- transform a soft CN into an equivalent but simpler soft CN…

In this part, we concentrate on the 3 first one, only for totally ordered structures (binary VCSP: minimization).

A simple case: idempotent VCSP

From the VCSP axioms:

Consider $\forall b \preccurlyeq_v a$,

- $a = (a \oplus \perp) \preccurlyeq_v (a \oplus b)$,
- $\Rightarrow (a \oplus b) \succcurlyeq_v a$.
- $b \preccurlyeq_v a \Rightarrow (a \oplus b) \preccurlyeq_v (a \oplus a) = a$ therefore $a \oplus b = a$.

$\oplus = \max_v$. Min-Max optimization problem: possibilistic/fuzzy CSP.

Introducing α-cuts

Consider a VCSP $\langle X, D, C, S \rangle$ and $\alpha \in E$.

The α-slice of $\langle X, D, C, S \rangle$ is the classical CSP $\langle X, D, C' \rangle$ where we authorize weakly forbidden tuples (less than α) and make all other hard (ex: fuzzy CSP).

All tuples with valuation lower than α are assigned valuation $\perp$ and all others are assigned $\top$.

$$C' = \{\varphi_\alpha \circ c, \forall c \in C\} \quad \varphi_\alpha(a) = (a \succcurlyeq_v \alpha ? \top : \perp)$$

The α-cut of a VCSP is a classical CSP ($\alpha = \top$: underlying CSP).

Solving a possibilistic VCSP

- Let t be an optimal solution of a possibilistic VCSP $\langle X, D, C, S \rangle$
- o its valuation.
- then, for any $\alpha \succ_v o$, t is a solution of the α-cut of $\langle X, D, C, S \rangle$
- all α-cuts with $\alpha \preccurlyeq_v o$ are inconsistent.

Ex: Prove.

Solving a possibilistic VCSP

Let A be the set of all valuations used in a possibilistic VCSP $\langle X, D, C, S \rangle$.$|A| \leq ed^2$.

- Solve all α-cuts for $\alpha \in A$: $O(ed^2)$ classical CSP to solve.
- Use binary (dichotomic) search: $O(\log(ed))$ CSP to solve.

Practical. All polynomial CSP classes conserved by α-cutting are also polynomial classes for possibilistic/fuzzy VCSP.

Solving a possibilistic VCSP

Open: is there a similar argument for partially ordered idempotent SCSP?
Ex: appply to the fuzzy dinner problem (reverse scale)
Ex: show the property does not hold for non idempotent (a solution of cost 100 may violate only constraints of cost 1).

- fish or meat: $f\ 0.8$, $m\ 0.3$
- water, Barolo or Greco di Tufo $w\ 0.7$, $b\ 1.0$, $g\ 0.9$

	w	b	g
f	0.6	0.7	1.0
m	0.6	1.0	0.5

Solving by Branch and Bound

Finding an optimal solution with a complete algorithm:

- finding an optimal solution (NP)
- proving that no better solution exists (optimality proof: co-NP)

The search space is in $O(d^n)$.

- Branch: partition the search space in (independant) subproblems.
- Bound: ignore subproblems that cannot contain an optimal solution

Simple: branching

The search space is described by $\langle X, D, C, S\rangle$ itself.

Consider a collection of hard constraints k_i. We can decompose the original problem into the collection $\langle X, D, C \cup \{k_i\}, S\rangle$.

- exhaustivity: $\vee_i k_i$ must eliminate no potential (optimal) solution.
- efficiency: do not search the same space twice $k_i \wedge k_j$ inconsistent.
- progress: the addition of k_i should simplify $\langle X, D, C, S\rangle$ and in fine make (X, D, C, S) trivial.

Branching methods

Variable based: select $x_j \in X$ s.t. $|D_j| > 1$.

- Use $(x_j = d_i)$ as k_i (by assignment). Branching factor $|D_j|$, depth n.
- Use $\{(x_j = d_1), (x_j \neq d_1)\}$ as $\{k_1, k_2\}$ (by assignment and refutation). BF 2, depth nd.
- Let $\{d_i\}$ be a partition of D_j. Use $(x_j \in d_i)$ as k_i (by domain spliting).

Constraint based: choose $c \in C$ s.t. $c = c_1 \vee c_2$. Use $k_1 = c_1$ and $k_2 = c_2 (\wedge \neg c_1)$ (eg. job shop scheduling: constraint splitting).

The branching tree

A rooted tree such that:

- the root is the original problem
- each son of a node is obtained by adding one of the selected k_i for the node.
- leaves are unbranchable problems (trivial to solve).

Branching by assignment: past (assigned) variables, future (unassigned) variables.

Ex: branching by assignment on the 3 queens problem.

Bounding

The branching tree is huge: pruning.
We suppose we have:

- a “procedure” that can compute a lower bound lb on the cost of an optimal solution of $\langle X, D, C, S\rangle$ at a given node.
- an upper bound ub on the cost of the problem (best known solution)
- (opt) a global lower bound glb on the cost of an optimal solution of the root problem.

At some node: if $lb \geq ub$ we can ignore the problem (cannot improve). If we find a solution of cost glb: we can stop.

Exploration strategy

- Depth first search: we branch on one of the most recently branched (deepest) subproblem.
- Breadth first search: we branch on one of the oldest (shallowest) subproblem.
- Best first search: we branch on the most promising subproblem (minimum lb in the open nodes).

BFS: explores less nodes. Offers a glb (min. of the open lb). Space exponential.

DFS: linear space.

Branch and Bound algorithm

Fonction $\texttt{DFBB}\,(t$: assig., ub : val.) : valuation

- $v \leftarrow lb(t)$;
- **if** $v \prec ub$ **then**
  - **if** $(|t| = n)$ **then return** v;
  - **Let** i be a future variable;
  - **foreach** $a \in d_i$ **do**
    - $ub \leftarrow \min(ub, \texttt{DFBB}\,(t \cup \{(i,a)\}, ub)\,)$;
  - **return** ub;
- **return** $\top$;

Ordering heuristics

- How to branch ? Select the variable x_j that will be assigned (variable ordering).
- Which problem to start with ? choose the first value (or k_i) that will be assigned to x_j (value ordering).

Variable: small domains (thin tree, hope that bounding will avoid later widening), degree: increase in lb.

Value: most promising... find a good ub rapidly. Problem dependent, smallest lb increase. We (almost) always have a solution.

Crucial component: the lb procedure

Must be:

- strong: the closest to the real value of the optimal solution the better.
- efficient: as costless to compute as possible.

Obviously antagonist aims. Matter of compromises and experimental evaluation (no theory of what a good lb is).

$\oplus = +$ used as an ideal practical example of non idempotent VCSP. All algorithms work for all practical instances of VCSP (can be optimized for $\oplus = \max$).

A first trivial lb (PBB, Freuder et al. 1992)

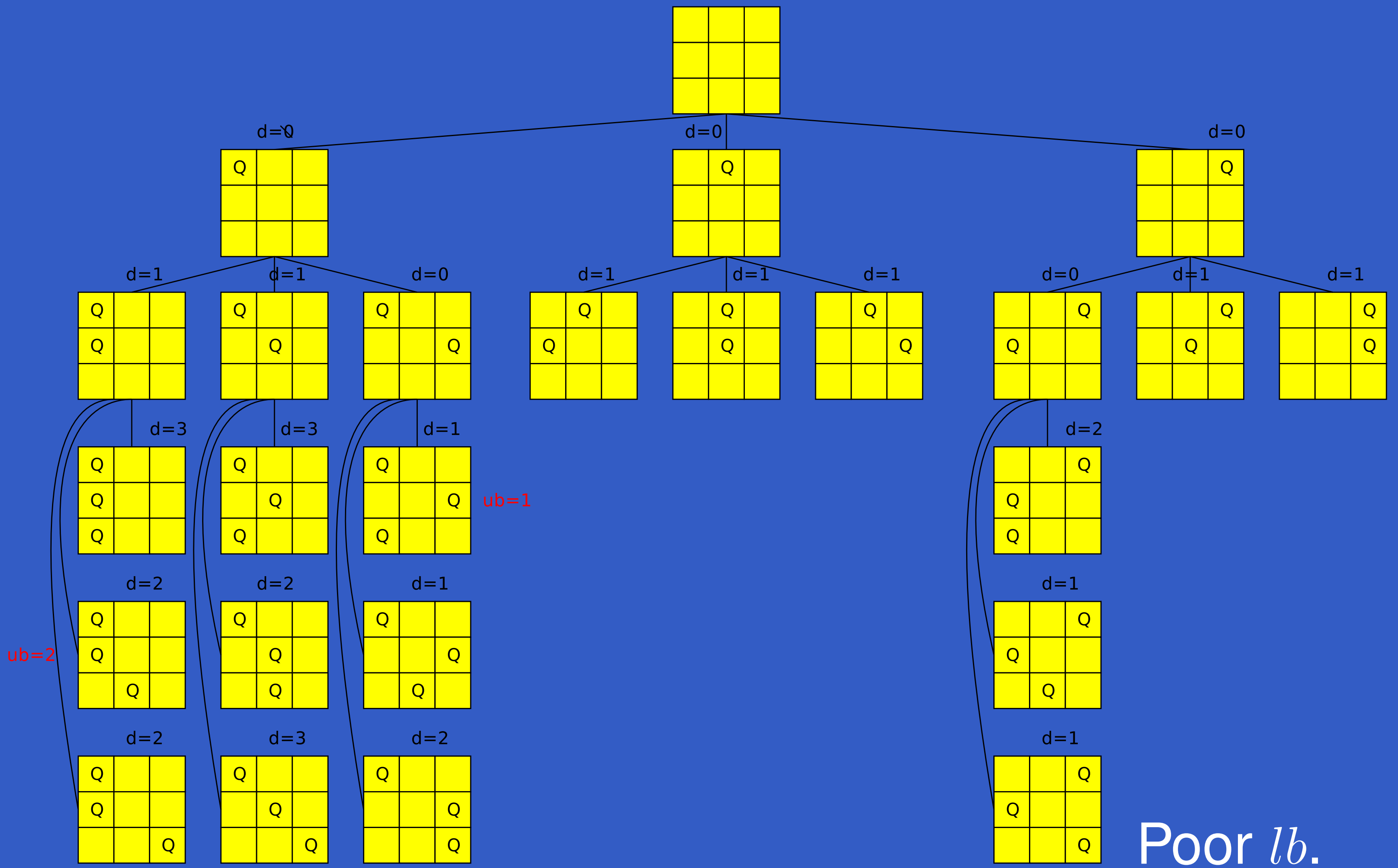
At a given node, let $AC \subset C$ be the set of assigned constraints (constraints connecting past/assigned variables).

Use

$$lb_d(t) = \bigoplus_{c_S \in AC} c(t[S])$$

Also called the “distance” (Partial CSP: number of constraints removed from the original problem needed to reach consistency. Reference to the metrics.).

The 3x3 queens



Poor lb.

PFC: Forward-checking based lb

The "distance" lower bound only takes into account constraints between past variables.

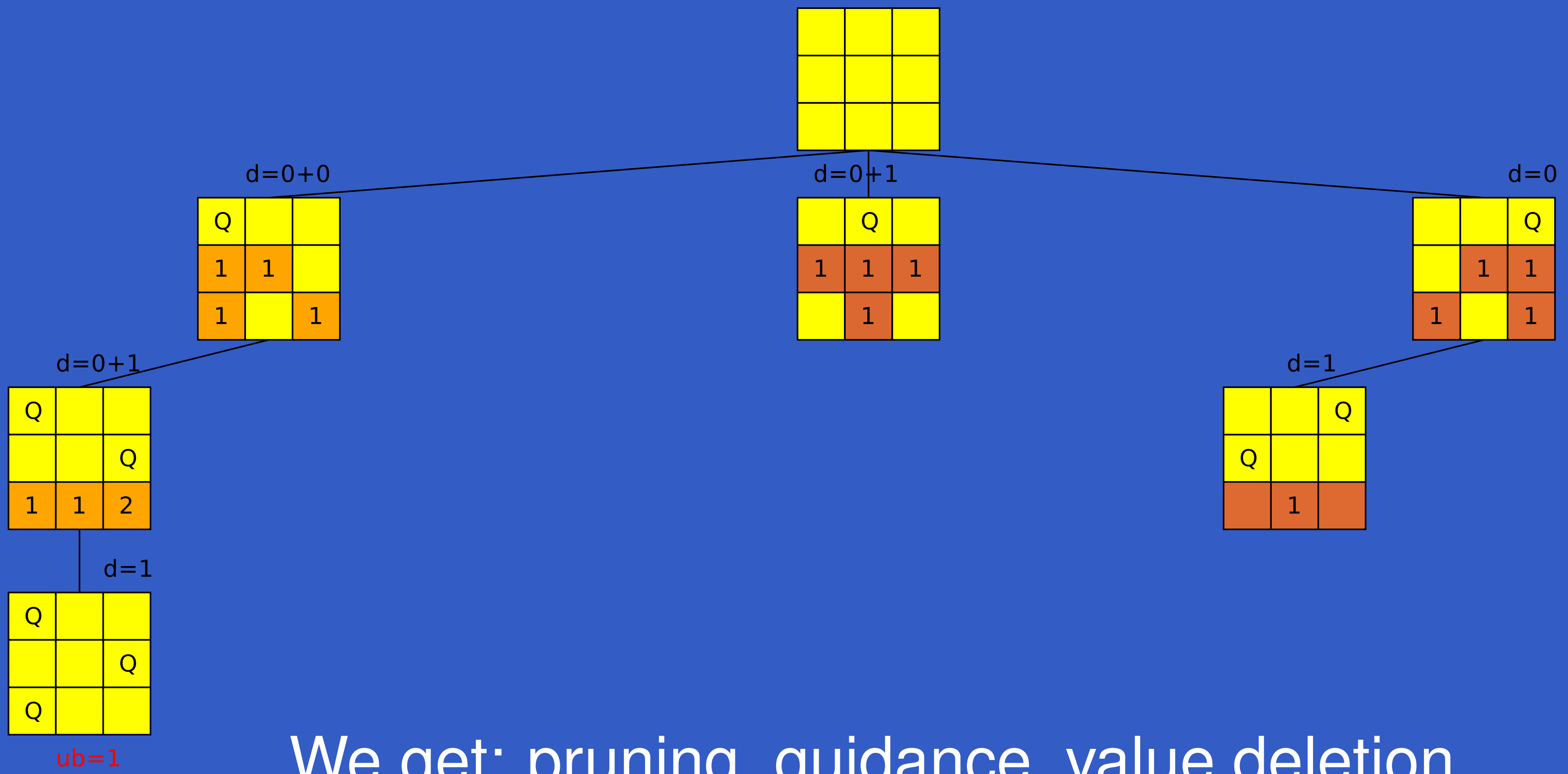
We should try to take into account more constraints.

FC: remove values that are inconsistent with past variables (constraints between past and future variables).

We cannot remove values. Assign counter fc_{jb} to value $b \in D_j$ = extra valuation if $x_j = b$: $c_{ij}(t[i], b)$.

$$lb_{fc}(t) = lb_d(t) \oplus \bigoplus_{x_i \in F} \min_{a \in D_i} fc_{ia}$$

The 3x3 queens



We get: pruning, guidance, value deletion.

$$lb_{fc}(j,b) = lb_d(t) \oplus fc(j,b) \oplus \bigoplus_{x_i \in F, i \neq j} \min_{a \in D_i} fc_{ia}$$

Still more ?

We haven't yet used the constraints between future variables (arc consistency ?).

- ac counter: ac_{ia} = extra guaranteed violations among future variables if $(x_i = a)$.
- Number of future variables with no consistent values with (i, a).

$$notlb = lb_d \oplus \bigoplus_{x_i \in F} \min_{a \in D_i} (fc_{ia} + ac_{ia})$$

$notlb$ is not a lower bound: we may pay the same cost twice. Ex: find a simple example that shows this.

Alternative: PFC-DAC

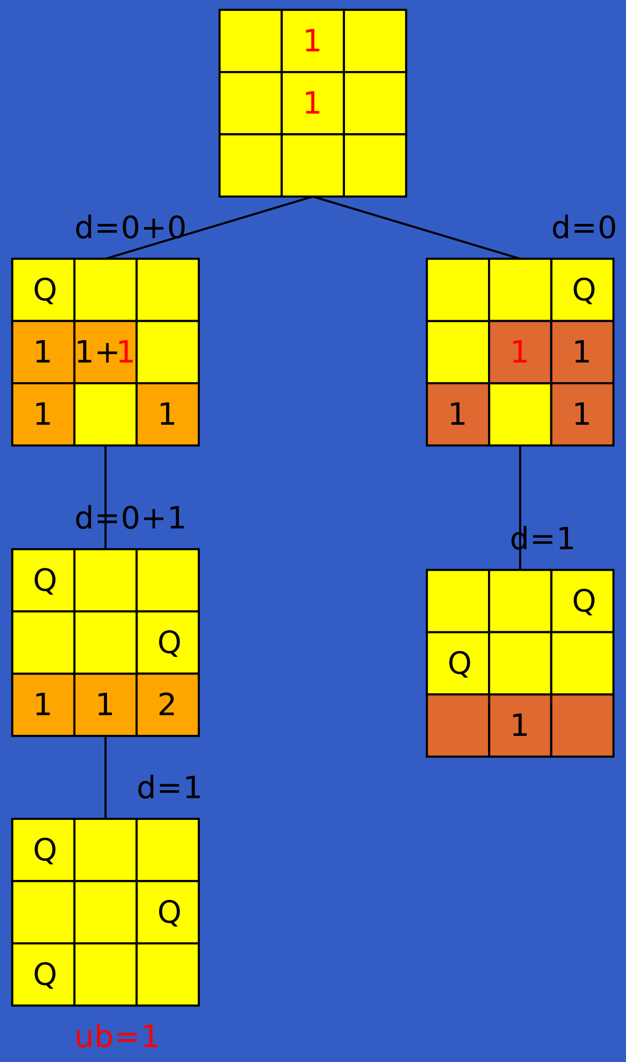
- use only ONE ac_{ia}, a “good” collection of ac_{ia} ?
- to avoid duplicated use: directed AC counts.

- variables are ordered $x_1 < \ldots < x_n$.
- for variable x_i, value a, dac_{ia} counts future variables which eg. *f*ollow x_i with no value compatible with (i, a).

Each constraint can participate in only one dac_{ia}. dac are computed before hand (statically).

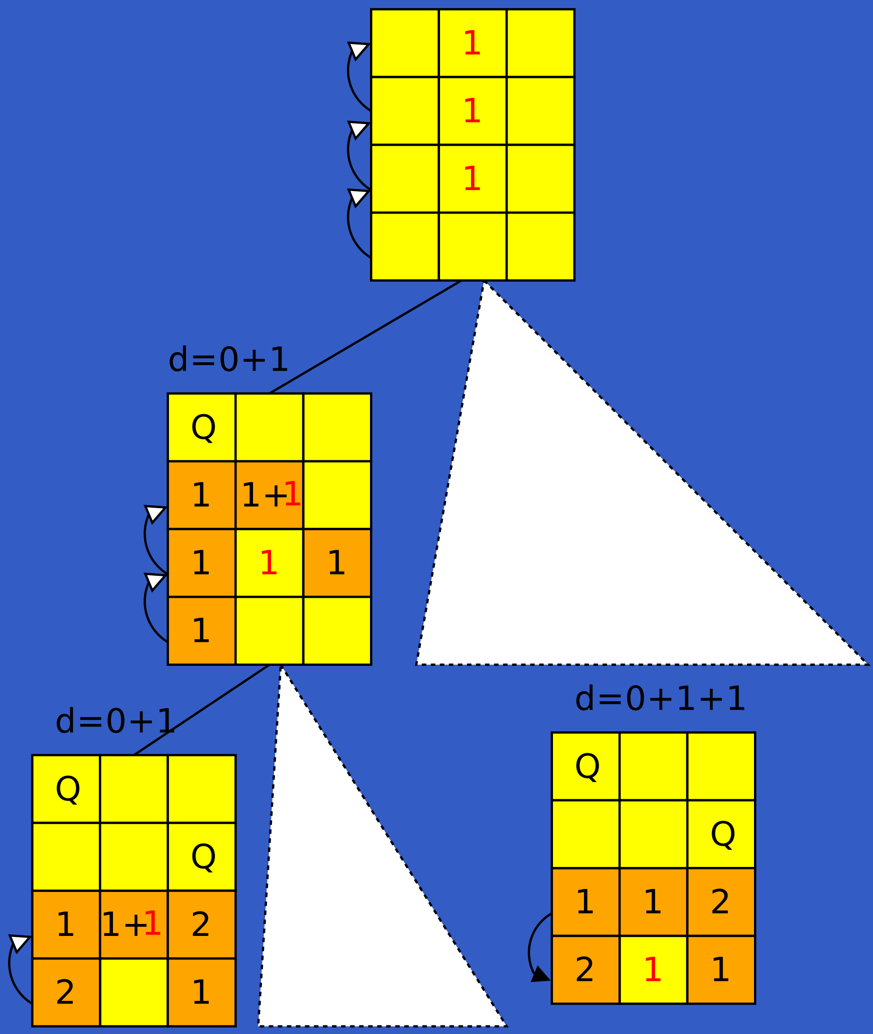
$$lb_{dac} = lb_d \oplus \bigoplus_{x_i \in F} \min_{a \in D_i} (fc_{ia} + dac_{ia})$$

3x3 queens



Some more pruning. Requires static ordering (dac and fc redundancy).

Can the DAC direction influence efficiency ?



Reversible DACs : PFC-RDAC

- at any node, a given constraint between unassigned variables is in a given direction.
- we choose the direction of constraints to maximize the lb.
- we can use dynamic variable ordering.

Maximizing the lb is NP-hard... heuristic greedy choice.

Value specific lb to prune (j, b) when $lb(j, b) \geq ub$.

$$lb_{rdac}(j,b) = lb_d(t) \oplus fc(j,b) \oplus dac(j,b) \oplus \bigoplus_{x_i \in F, i \neq j} \min_{a \in D_i} (fc_{ia} \oplus dac_{ia})$$

Still more: deletion propagation

- when a value is deleted because of $lb_{rdac}(j, b)$, it is possible that a dac_{ia} can be augmented.
- dynamically update dac counters after value deletion.

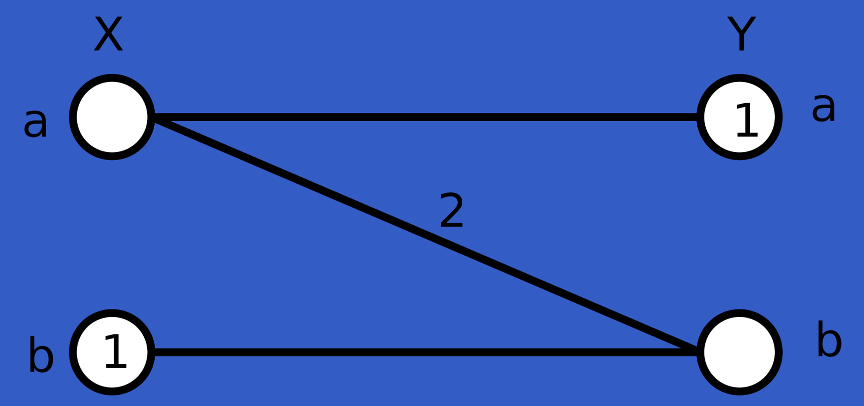
PFC-MRDAC (Larrosa et al. 1998). The flavor of arc consistency but without arc consistency.

May be counterproductive on random problems. . .

Weighted AC counts

DAC and RDAC counts have been generalized by so-called WAC counts (for additive VCSP).

For each constraint c_{ij}, we choose the fraction α of the constraint that will be used in i and the rest $(1-\alpha)$ will go to j.



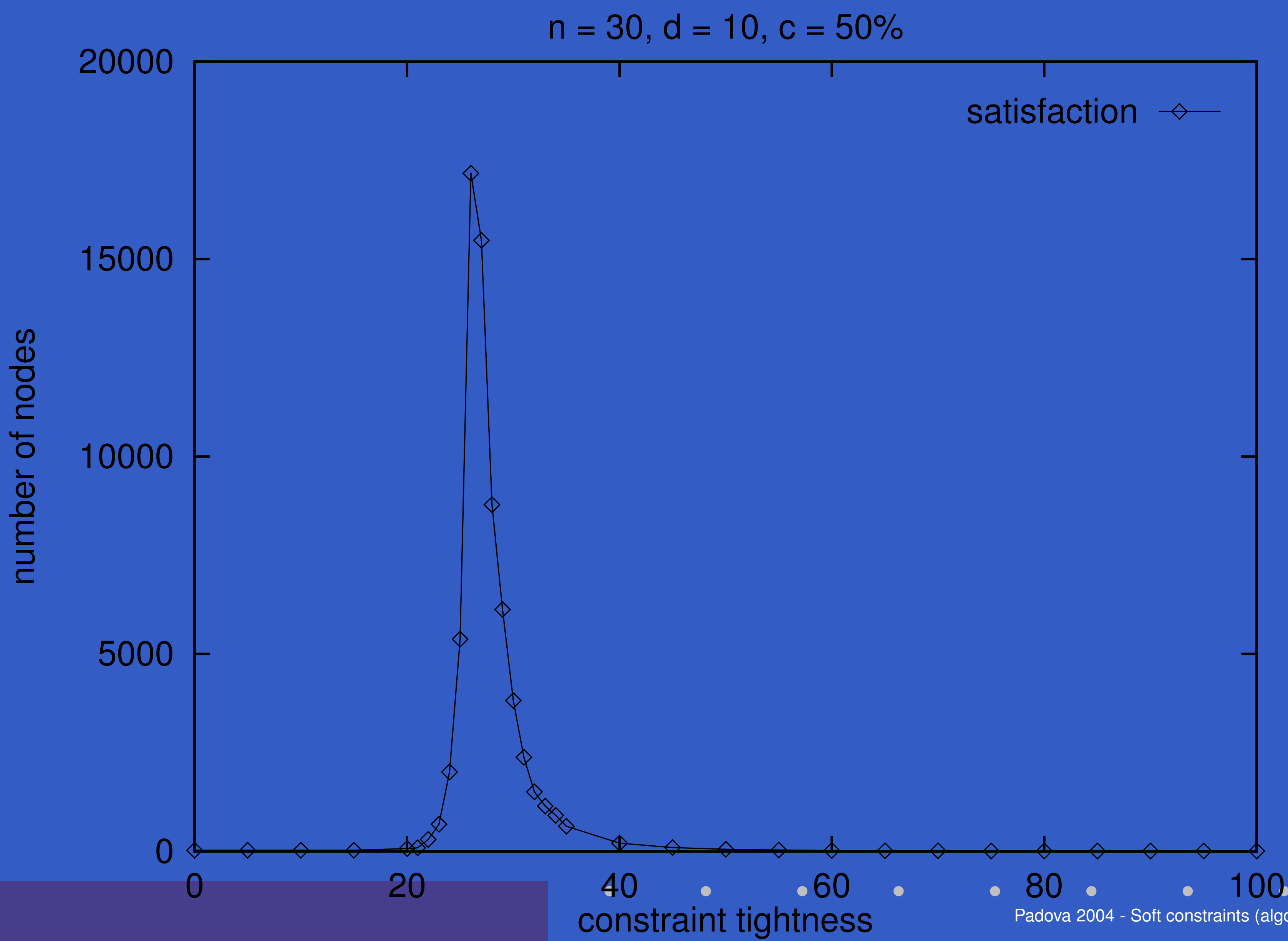
Experimentations

Although lb strengths can be compared, the efficiency/strength compromise is best assessed by experimental evaluation.

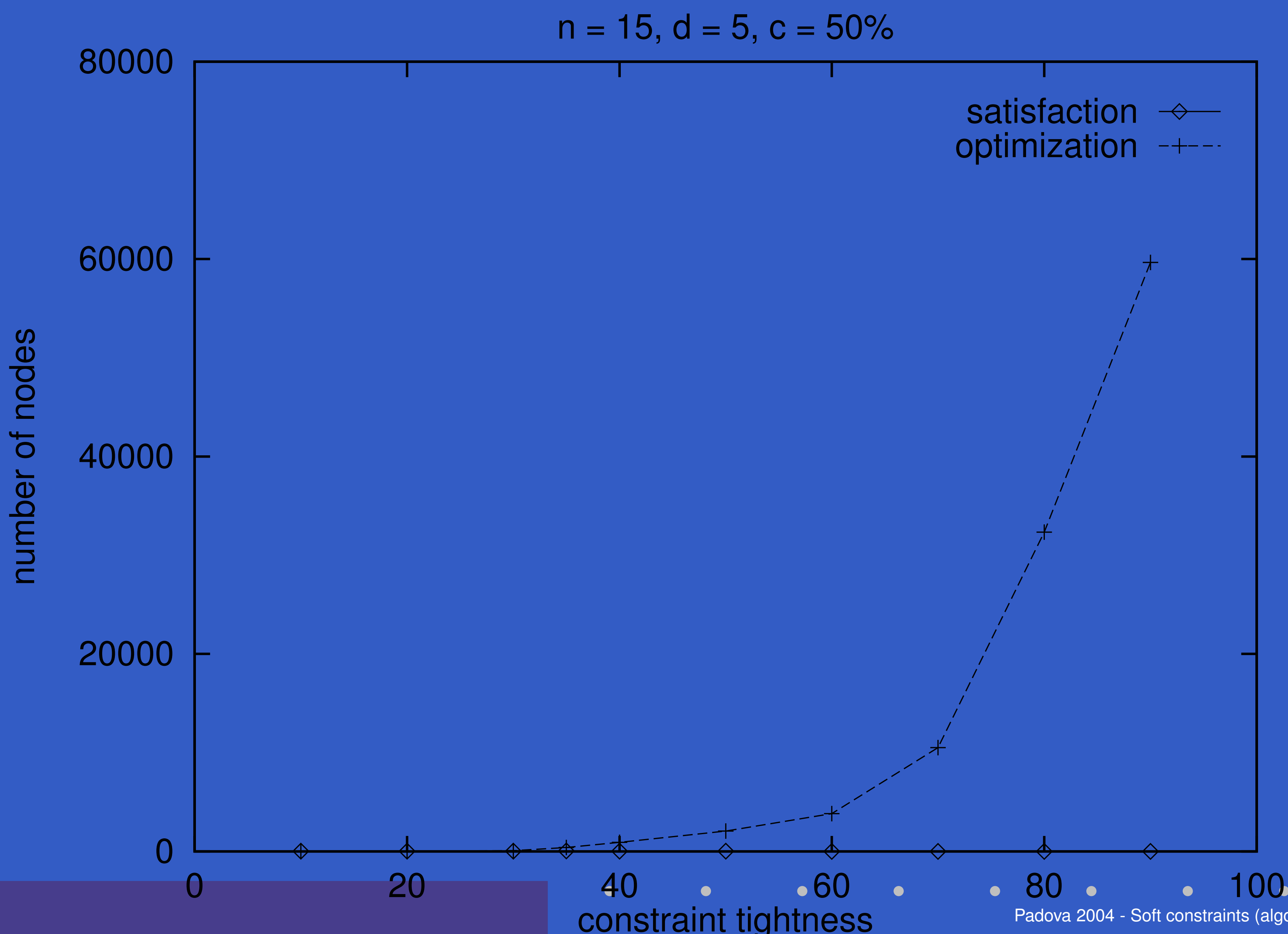
- academic problems: n-queens,. . .
- real problems: frequency allocation, satellite scheduling. . .
- random binary problems: same as random CSP. Use a cost of 1 when the constraint is violated.

A random CSP class is defined by $\langle n, d, p_1, p_2 \rangle$. p_1 is the number of constraints, p_2 the number of pairs in constraints that will receive cost 1.

Phase transition in classical CSP

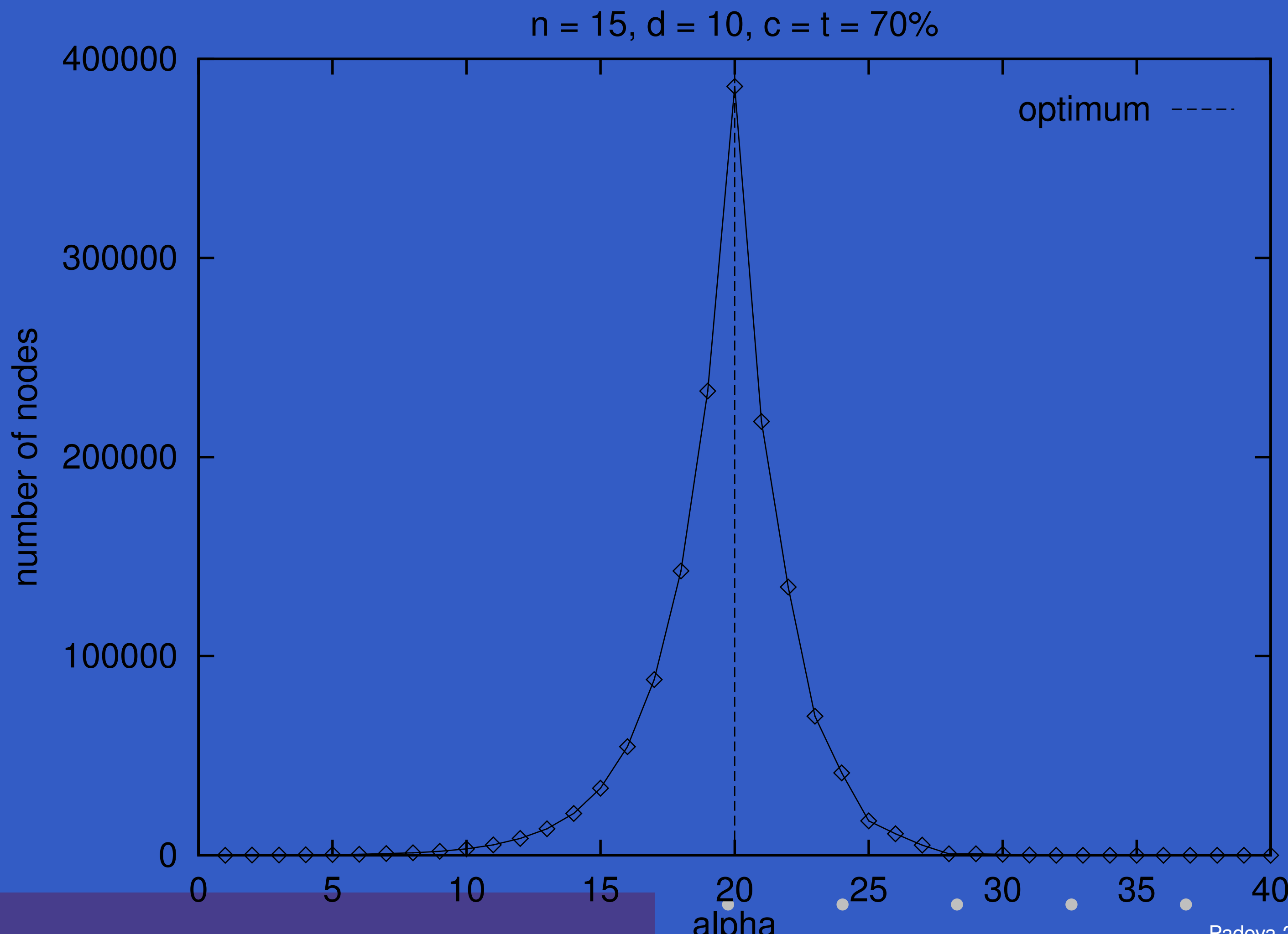


Additive VCSP (PFC)



Why is it so hard ?

Problem $P(\alpha)$: is there an assignment of valuation strictly lower than α ?



So...

- the proof of inconsistency ($P(1)$) is among the simplest problems;
- the proof of optimality ($P(opt)$) is the hardest problem;
- the proof of optimality ($P(opt)$) is harder than the production of an optimal solution ($P(opt+1)$) ;
- a depth first branch and bound algorithm solves a sequence of problems $P(\alpha)$; it hsa to solve $P(opt+1)$ and $P(opt)$ at least;
- starting from a good solution, possibly optimal, will not avoid the resolution of problem $P(opt)$.

Local search

Another general class of algorithms used to solve combinatorial optimization problem.

General idea: starting from a potential solution t, we try to locally modify t into t', close to t but potentially better. Repeat until satisfied.

Incomplete algorithms: does not try to solve $P(opt)$. Often quite efficient but may have pathological behavior. No guarantee (but asymptotic guarantee for some). Deals only with optimization.

Terminology

A solution is the object you want to optimize. Typically a complete assignment (may violate hard constraints).

A “move” is an elementary operation that allows to go from a solution t to another solution t' (a neighbor of t).

Moves must allow ultimately to reach any solution after a finite number of moves.

The set of all neighbors of t (reachable by one move): neighborhood of t.

A trial is a succession of moves. A local search is a succession of trials.

Moves, criteria

We assume we have additive VCSP (but works in general) with no hard constraints.

A solution = a complete assignment t.

A move: change the value of one (or more) variable(s) in t to another element of its domain.

Ex: in the 4 queens problem, give the neighborhood of $< 1, 2, 3, 4 >$.

We optimize $\varphi(t)$. Assume $\varphi(t)$ is valuation of the t (but this is not necessarily the case).

```
LocalSearch ();
x* ← NewSolution ();
for t = 1 to Max-Trials do
    x ← NewSolution ();
    for m = 1 to Max-Moves do
        x' ← ChooseNeighbor (x);
        δ ← (φ(x') − φ(x));
        if φ(x') < φ(x*) then
            x* ← x';
        if Accept? (δ) then
            x ← x';
return Nothing better than (x*, φ(x*))
```

Parameters

Max-trials: number of trials.

Max-Moves: number of moves per trial.

NewSolution: generates a new “solution” (random or heuristically).

ChooseNeighbor (t): chooses an element in the neighborhood of t.

Accept? (δ): accepts the move or not.

Important properties

Brute force methods. Should be able to explore a large number of solutions.

- a solution should be simple to represent
- the application of a move should be typically constant time
- the change in the criteria after a move should be incrementally computed from the previous one (constant time).

Ad-hoc langage for incremental maintenance of structures/criteria: LOCALIZER (P. van Hentenryck).

Descent search

ChooseNeighbor (x) : random choice of x' in the neighborhood of x.

Accept? (δ) : $(\delta \leq 0)$.

Accept only when it does not get worse.

Fast, stuc in local minima.

Greedy search

ChooseNeighbor (x): choose randomly a best neighbor (greedy).

Accept? (δ) : *true*
We always accept.

Greedyness does not mean we cannot go up (in a local minima).

Usual behavior

In a trial:

1. descent: a majority of moves improve the criteria.
2. this gradually becomes less and less frequent. . .
3. we get stuck in long “plateaus” and in local minima. Occasional improvements (greedy search).

Improvements

Random walk: with a probability p we decide to choose a random move instead of the usual move. One more parameter.

Taboo: we memorize the last k moves and forbid to use them again. Avoid to go back to already explored solutions. Again one parameter.

Simulated annealing

Inspired from physical statistics. Energy = φ, move = state change.

The probability of going from a state a to a state b with a higher (worse) energy is:

$$P(a, b, T) = e^{\frac{(a-b)}{k_B T}}$$

k_B is the Boltzmann constant. If we lower T (temperature) very slowly we get in minimal energy states.

Simulated annealing

- the probability of accepting a move m from x to x'
  - 1 if $\varphi(x') \leq \varphi(x)$
  - $e^{\frac{\varphi(x)-\varphi(x')}{T}}$ otherwise.
- we start with an initial T
- after a fixed number of moves, we decrease the temperature (cooling schedule. Geometric: $T^i = \alpha.T^{i-1}$)

Hard constraints

Hard constraints are difficult to cope with: infinite costs remove all “gradient information”.

Typical approach: relax the constraint by penalizing violation (larger than soft constraints).

When some hard constraint is repeatedly violated, increase its weight (for a period of time) (Breakout. . .)