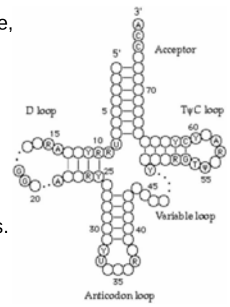
Soft Constraint Processing

16.412J/6.834J Cognitive Robotics

Martin Sachenbacher
(Using material from Thomas Schiex)

Example: Bioinformatics

- **RNA** is single-strand molecule, composed of A,U,G,C
- Function of RNA depends on **structure** (3-D folding)
- Structure induced by **base pairing**: Watson-Crick (A-U, G-C) and Wobble (G-U).
- Problem: Find RNA structure that **maximizes** base pairings.
- Cumbersome to frame as Optimal CSP!

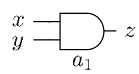


Overview

- Soft Constraints Framework
- Algorithms: Search (Branch-and-Bound)
- Algorithms: Inference (Dynamic Programming)
- Applications: Frequency Assignment Problems

From Optimal CSP to Soft CSP

- Optimal CSP: Minimize **function** f(y), s.t. **constraints** C(x) are **satisfiable**.



Utility Function

$f : a_1 \rightarrow [0,1]$

$f(\mathrm{G}) = .99$
$f(\mathrm{U}) = .01$

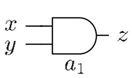
Constraint

$c :$

a_1	x	y	z
G	0	0	0
G	0	1	0
G	1	0	0
G	1	1	1
U	0	0	0
...			

From Optimal CSP to Soft CSP

- Soft CSP: **Extend** the notion of **constraints** to **include preferences**.



Soft Constraint

$c :$

a_1	x	y	z	
G	0	0	0	.99
G	0	1	0	.99
G	1	0	0	.99
G	1	1	1	.99
U	0	0	0	.01
...				...

Notation

- A k-**tuple** is a sequence of k objects $\langle v_1, \ldots, v_k \rangle$.
- The i-th component of a tuple t is denoted $t[i]$.
- The **projection** of a tuple t on a subset S of its components is denoted $t[S]$.
- The **cartesian product** of sets $A_1, \ldots, A_k$, denoted $\Pi_{i=1}^{k} A_i$, is the set of all k-tuples such that $t[i] \in A_i$.

Classical CSP

A constraint network $\langle X, D, C \rangle$

- set of **variables** $X = \{x_1, \ldots, x_n\}$
- set of **domains** $D = \{d_1, \ldots, d_n\}$
- set of **constraints** $C = \{c_1, \ldots, c_m\}$

A constraint $c \in C$ is a **relation** $c \subseteq \Pi_{x_j \in \mathrm{var}(c)} d_j$ on variables $\mathrm{var}(c)$ with arity $|\mathrm{var}(c)|$.

A complete assignment t is **allowed** if $\forall c \in C, t[\mathrm{var}(c)] \in c$.

Valued CSP

- For each **constraint/tuple**: a **valuation** that reflects preference (e.g. cost, weight, priority, probability, …).
- The **valuation of an assignment** is the combination of the valuations expressed by each constraint using a **binary operator** (with special axioms).
- Assignments can be compared using a **total order** on valuations.
- The problem is to produce an **assignment of minimum valuation**.

Formally: Valuation Structure

$S = \langle E, \oplus, \preceq, \perp, \top \rangle$

- E = **set of valuations**, used to assess assignments
- $\perp$ = **minimum element** of E, corresponds to totally consistent assignments
- $\top$ = **maximum element** of E, corresponds to totally inconsistent assignments
- $\preceq$ = **total order** on E, used to compare two valuations
- $\oplus$ = **operator** used to **combine** two valuations

Valued CSP

A constraint network $\langle X, D, C, S \rangle$

- set of **variables** $X = \{x_1, \ldots, x_n\}$
- set of **domains** $D = \{d_1, \ldots, d_n\}$
- set of **constraints** $C = \{c_1, \ldots, c_m\}$
- **valuation structure** $S = \langle E, \oplus, \preceq, \perp, \top \rangle$

A constraint $c \in C$ is a **function** $c : \Pi_{x_j \in \mathrm{var}(c)} d_j \to E$ mapping tuples over $\mathrm{var}(c)$ to valuations.

The **valuation** of a complete assignment t is $\bigoplus_{c \in C} c(t[\mathrm{var}(c)])$.

Required Properties

- $\forall \alpha, \beta \in E, (\alpha \oplus \beta) = (\beta \oplus \alpha)$. (Commutativity)
- $\forall \alpha, \beta, \gamma \in E, (\alpha \oplus (\beta \oplus \gamma)) = ((\alpha \oplus \beta) \oplus \gamma)$. (Associativity)
- $\forall \alpha, \beta, \gamma \in E, (\alpha \preceq \beta) \Rightarrow ((\alpha \oplus \gamma) \preceq (\beta \oplus \gamma))$. (Monotonicity)
- $\forall \alpha \in E, (\alpha \oplus \perp) = \alpha$. (Neutral element)
- $\forall \alpha \in E, (\alpha \oplus \top) = \top$. (Annihilator)

Exercise: Justify properties.

Instances of the Framework

	E	$\preceq$	$\perp$	$\top$	$\oplus$
Classical	{t,f}	$t \prec f$	t	f	$\wedge$
Weighted	$N_0^+ \cup \infty$	$\leq$	0	∞	$+$
Probabilistic	$[0,1]$	$\geq$	1	0	$*$
Fuzzy	$[0,1]$	$\geq$	1	0	$\min$

Many others in the literature.

From Valued CSP to Optimal CSP

- Introduce **decision variable** for each constraint
- Its **values** correspond to **different valuations**

$c:$

x	y	z	
a	a	a	0
a	b	a	0
b	a	a	1
b	b	b	1

$S = \langle N_0^+ \cup \infty, +, \leq, 0, \infty \rangle$

From Valued CSP to Optimal CSP

- Introduce **decision variable** for each constraint
- Its **values** correspond to **different valuations**

$c:$

d	x	y	z	
v_1	a	a	a	0
v_1	a	b	a	0
v_2	b	a	a	1
v_2	b	b	b	1

$S = \langle N_0^+ \cup \infty, +, \leq, 0, \infty \rangle$

From Valued CSP to Optimal CSP

- Introduce **decision variable** for each constraint
- Its **values** correspond to **different valuations**
- **Utility function** maps values to valuations
- Constraints become **relations**

$c:$

d	x	y	z
v_1	a	a	a
v_1	a	b	a
v_1	b	a	a
v_2	b	b	b

$f: d \rightarrow N_0^+ \cup \infty$

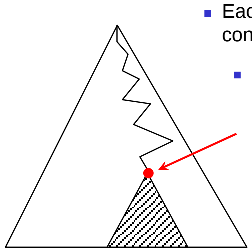
$f(v_1) = 0$
$f(v_2) = 1$

Multiattribute utility function $= +$

Overview

- Soft Constraints Framework
- **Algorithms: Search (Branch-and-Bound)**
- Algorithms: Inference (Dynamic Programming)
- Applications: Frequency Assignment Problems

Branch-and-Bound Search



- Each search node is a soft constraint subproblem
  - **Lower Bound (lb):** Optimistic estimate of best solution in subtree
    - **Upper Bound (ub):** Best solution found so far
      - Prune, if **lb ≥ ub**.

Branch-and-Bound Algorithm

- **Function** DFBB (t : assignment, ub : value): value
  - $v \leftarrow lb(t)$
  - **if** $v \prec ub$ **then**
    - **if** $|t| = n$ **then return** v
    - **let** x_i be an unassigned variable
    - **for each** $a \in d_i$ **do**
      - $ub \leftarrow \min(ub, \mathrm{DFBB}(t \cup \{(i,a)\}, ub))$
    - **return** ub
  - **return** $\top$

Time: O(exp(n))
Space: O(n)

Lower Bound Procedure

Must be:

- **Strong**: the closest to the real value, the better.
- **Efficient**: as easy to compute as possible.

Creates a **trade-off**. Choice is often a matter of compromises and experimental evaluation.

Distance Lower Bound

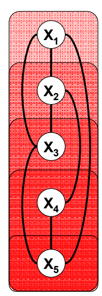
- At each node, let $AC \subseteq C$ be the set of constraints all of whose variables have been assigned.
- Use the bound

$$lb(t) = \bigoplus_{c \in AC} c(t[\mathrm{var}(c)])$$

- Problem: often **weak**, as it takes into account only **past variables**.

Improvement: Russian Doll Search

- Idea: we can add the value of the **optimal solution** to the **subproblem** over **future variables** to distance lower bound, and get a stronger lower bound.
- Must solve subproblem over future variables **beforehand**.
- Yields **recursive** procedure that solves increasingly large subproblems.



Russian Doll Search

- [Lemaitre Verfaillie Schiex 96]: Experiments with Earth Observation Satellite Scheduling Problems (maximization problem).
- Example: 105 variables, 403 constraints.
- Branch-and-Bound with distance lower bound: Aborted after **30 min**, best solution so far = **8095**.
- Russian Doll Search: Optimal solution = **9096** found in **2.5 sec**.

Overview

- Soft Constraints Framework
- Algorithms: Search (Branch-and-Bound)
- **Algorithms: Inference (Dynamic Programming)**
- Applications: Frequency Assignment Problems

Inference

- Inference produces new constraints that are **implied** by the problem.
- Makes problem more **explicit**, easier to solve.
- Operations on constraints: **combination** and **projection**.

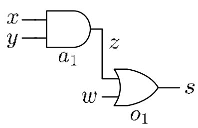
VCSP ⟶ VCSP'

Equivalent, simpler to solve

Combination

- $c_1 \bowtie c_2$ is constraint on $\mathrm{var}(c_1) \cup \mathrm{var}(c_2)$ s.t.

  $(c_1 \bowtie c_2)(t) = c_1(t[\mathrm{var}(c_1)]) \oplus c_2(t[\mathrm{var}(c_2)])$



c_1 :

a_1	x	y	z	
G	0	0	0	.99
G	0	1	0	.99
G	1	0	0	.99
G	1	1	1	.99
U	0	0	0	.01
...				...

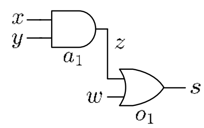
c_2 :

o_1	z	w	s	
G	0	0	0	.95
G	0	1	1	.95
G	1	0	1	.95
G	1	1	1	.95
U	0	0	0	.05
..				...

Combination

- $c_1 \bowtie c_2$ is constraint on $\mathrm{var}(c_1) \cup \mathrm{var}(c_2)$ s.t.

  $(c_1 \bowtie c_2)(t) = c_1(t[\mathrm{var}(c_1)]) \oplus c_2(t[\mathrm{var}(c_2)])$



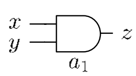
$c_1 \bowtie c_2$:

a_1	o_1	x	y	z	w	s	
G	G	0	0	0	0	0	.9405
G	G	0	0	0	1	1	.9405
G	U	0	0	0	0	1	.0495
U	G	1	1	0	0	0	.0092
U	U	0	0	0	0	0	.0005
...							...

Projection

- $c \Downarrow_Y, Y \subseteq X$ is a constraint on $\mathrm{var}(c_1) \cap Y$ s.t.

  $(c \Downarrow_Y)(t) = \min_{t'[Y]=t} c(t')$



c :

a_1	x	y	z	
G	0	0	0	.99
G	0	1	0	.99
G	1	0	0	.99
G	1	1	1	.99
U	0	0	0	.01
...				...

$c \Downarrow_{\{x,z\}}$:

x	z	
0	0	.99
1	0	.99
1	1	.99
0	1	.01

Inferring Solutions

- Constraint network $\langle X, D, C, S \rangle$
- Value of optimal solution obtained by **combining all constraints** C and **eliminating all variables** X :

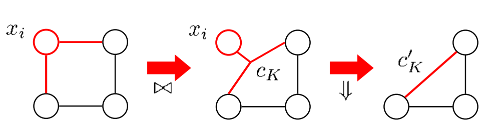
  $(c_1 \bowtie c_2 \bowtie \ldots \bowtie c_m) \Downarrow_\emptyset$

- Problem: **Very costly**: Time O(exp(n)), Space O(exp(n)).

Improvement: Bucket Elimination

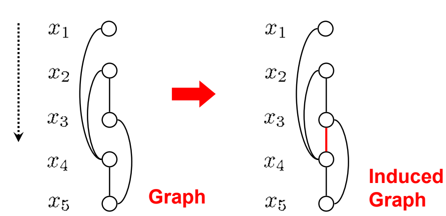
Idea: Eliminate variable as **soon as it no longer occurs** in **remaining** set of constraints.

- For variable $x_i \in X$, let $K_{x_i} = \{c \in C : x_i \in \mathrm{var}(c)\}$
- Compute **combination** c_K of all constraints in K_{x_i}
- Now **eliminate** x_i from $c_K : c'_K = c_K \Downarrow_{X \setminus \{x_i\}}$
- **Remove** K_{x_i} from C and **add** c'_K to C .



Induced Graph

- When processing a node (variable), **connect all neighboring nodes** not yet processed.



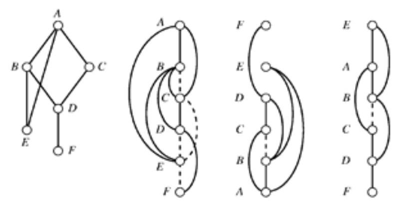
Bucket Elimination: Complexity

Let **width** be the maximum number of successors in the induced graph. Then:

- **Time** dominated by computation of largest c_K : **O(exp(width+1))**
- **Space** dominated by storage of largest c'_K : **O(exp(width))**

Impact of Variable Ordering

Width: 4 **Width: 2** **Width: 2**



Finding ordering with minimal width is NP-hard.

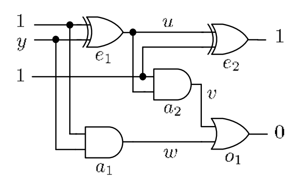
Min-Fill Ordering Heuristic

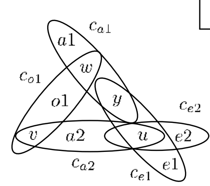
- Function MF (G: Graph with edges E and nodes $V = \{v_1, \ldots, v_n\}$): Order
  - **for** $j = 1$ **to** n
    - **let** v be a node in V with minimal number of edges required to connect its neighbors
    - put v in position j of order
    - $E \leftarrow E \cup \{(v_i, v_j) : (v_i, v) \in E, (v_j, v) \in E\}$
    - $V \leftarrow V \setminus \{v\}$

Often finds good orderings in practice.

Example

Full Adder Circuit





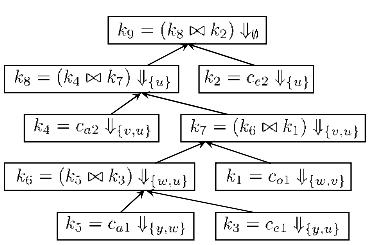
Full Adder Circuit Hyper-Graph

Example

MF Order

$1: o_1$
$2: e_2$
$3: e_1$
$4: a_2$
$5: a_1$
$6: y$
$7: w$
$8: v$
$9: u$

Computational Scheme ("Bucket Tree")



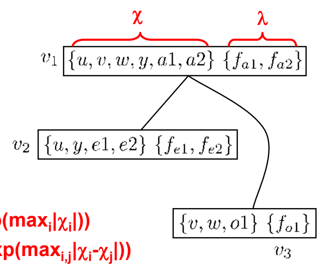
Tree Decomposition

A **tree decomposition** for a problem $\langle X, D, C, S \rangle$ is a triple (T, χ, λ), where $T = (V, E)$ is a rooted tree, and χ, λ are labeling functions associating with each node $v_i \in V$ two sets $\chi(v_i) \subseteq X$, $\lambda(v_i) \subseteq C$ such that:

- For each $c \in C$, there is exactly one v_i such that $c \in \lambda(v_i)$. For this v_i, $\text{var}(c) \subseteq \lambda(v_i)$ (**covering condition**);
- For each $x \in X$, the set $\{v_j \in V : x \in \chi(v_j)\}$ of vertices labeled with x induces a connected subtree of T (**connectedness condition**).

Example

Tree Decomposition



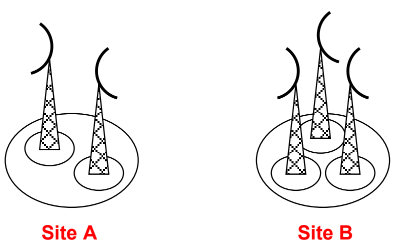
Time: $O(\exp(\max_i|\chi_i|))$

Space: $O(\exp(\max_{i,j}|\chi_i - \chi_j|))$

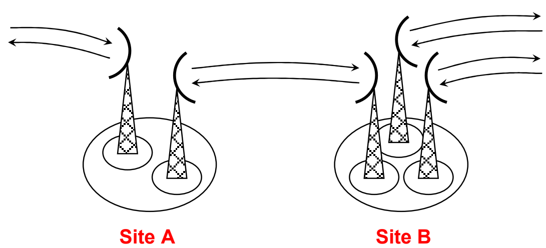
Overview

- Soft Constraints Framework
- Algorithms: Search (Branch-and-Bound)
- Algorithms: Inference (Dynamic Programming)
- **Applications: Frequency Assignment Problems**

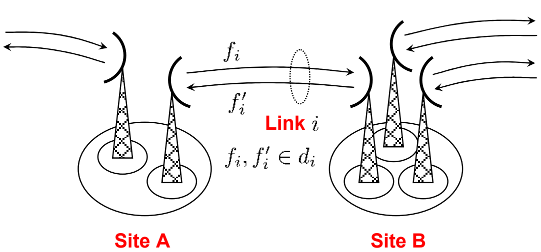
Frequency Assignment



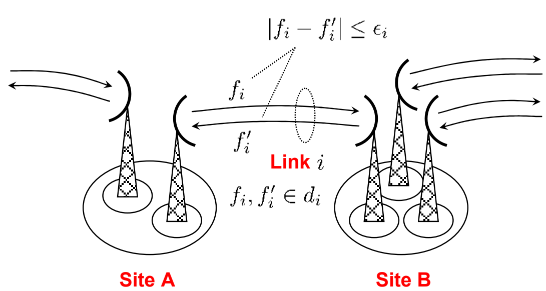
Frequency Assignment



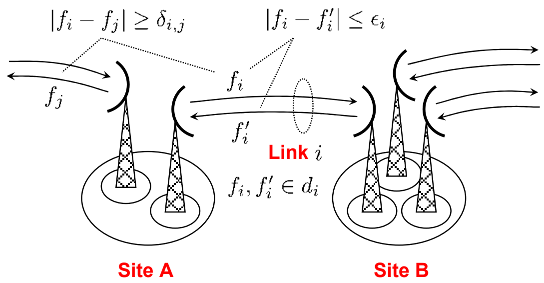
Frequency Assignment



Frequency Assignment



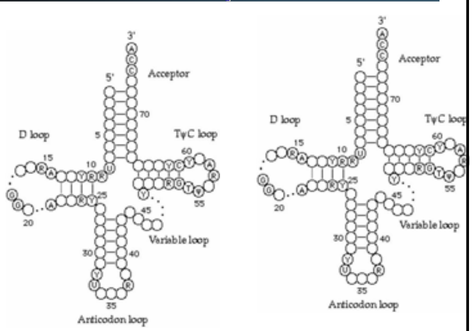
Frequency Assignment



Frequency Assignment

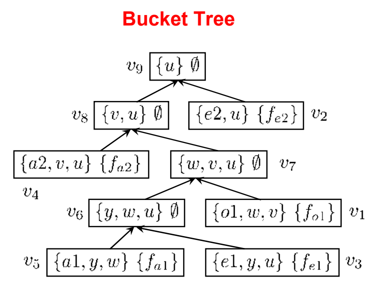
- Several instances available from CELAR (200 to 916 variables, 1200 to 5000 constraints, domain size >30)
- These are very hard instances of valued CSPs.
- Good results reported for dynamic programming.

RNA Structure Example

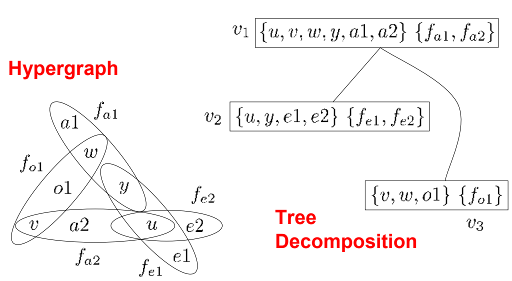


Example

$v_1 : o_1 \ f_{o1}$
$v_2 : e_2 \ f_{e2}$
$v_3 : e_1 \ f_{e1}$
$v_4 : a_2 \ f_{a2}$
$v_5 : a_1 \ f_{a1}$
$v_6 : y$
$v_7 : v$
$v_8 : w$
$v_9 : u$

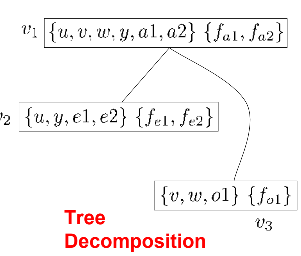


Tree Decomposition Example



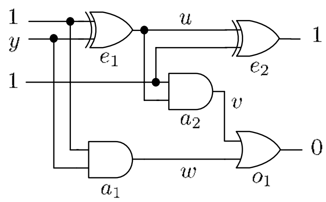
Dynamic Programming

- Inference on the tree: dynamic programming



Example

- AND-gates broken with 1% probability
- OR, XOR-gates broken with 5% probability
- Probabilistic valuation structure $([0,1], \geq, \cdot, 1, 0)$



Example

f_{a1}:

a_1	w	y	
G	0	0	.99
G	1	1	.99
B	0	0	.01
B	0	1	.01
B	1	0	.01
B	1	1	.01

f_{a2}:

a_2	u	v	
G	0	0	.99
G	1	1	.99
B	0	0	.01
B	0	1	.01
B	1	0	.01
B	1	1	.01

f_{e1}:

e_1	u	y	
G	1	0	.95
G	0	1	.95
B	0	0	.05
B	0	1	.05
B	1	0	.05
B	1	1	.05

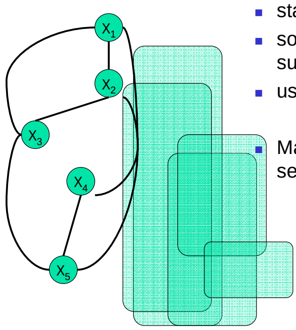
f_{o1}:

o_1	v	w	
G	0	0	.95
B	0	0	.05
B	0	1	.05
B	1	0	.05
B	1	1	.05

f_{e2}:

e_2	u	
G	0	.95
B	0	.05
B	1	.05

Russian Doll Search



- static variable ordering
- solves increasingly large subproblems
- uses previous LB recursively
- May speedup search by several orders of magnitude

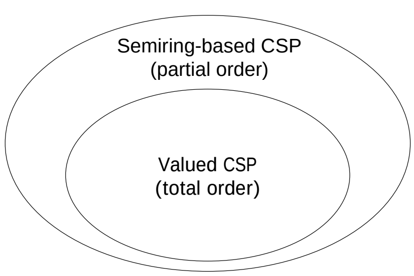
Soft Constraint Framework

- (X,D,C)
  - $X=\{x_1,\ldots, x_n\}$ variables
  - $D=\{D_1,\ldots, D_n\}$ finite domains
  - $C=\{c_1,\ldots, c_e\}$ cost functions
    - var(c_i) scope
    - $c_i(t)$: $\rightarrow$ E (ordered cost domain, T, $\perp$)

  annihilator (T), identity ($\perp$)

- **Obj. Function**: $F(X)= \oplus c_i\ (X)$ → { • commutative • associative • monotonic }
- **Solution**: $F(t) \neq$ T
- **Soft CN**: find minimal cost solution

Valued CSPs and Semiring CSPs



Specific Frameworks

- E = {t,f} ⊕ = and Classical CSP
- E = N ∪{∞} ⊕ = + Weighted CSP
- E = [0,1] ⊕ = * Probabilistic CSP

Lexicographic CSP, probabilistic CSP…

From VCSPs to OCSPs

- Introduce decision variable for each constraint
- Introduce domain value for each different value of a tuple's constraint

Basic Operations on Constraints

- Assignment (Conditioning)
- Combination (Join)
- Projection (Elimination)

Assignment (Conditioning)

x_i	x_j	$c(x_i,x_j)$
b	b	T=6
b	g	0
b	r	0
g	b	0
g	g	T=6
g	r	0
r	b	0
r	g	0
r	r	T=6

Assign(**c**,$\mathbf{x_i}$,b) →

$f(x_j)$
0

Assign(**f**,$\mathbf{x_j}$,g) →

$h_\emptyset$

Combination (Join)

x_i	x_j	$c(x_i,x_j)$
b	b	6
b	g	0
g	b	0
g	g	6

$\oplus$

x_j	x_k	$c(x_j,x_k)$
b	b	6
b	g	0
g	b	0
g	g	6

x_i	x_j	x_k	$f(x_i,x_j,x_k)$
b	b	b	12
b	b	g	6
b	g	b	0
b	g	g	6
g	b	b	6
g	b	g	0
g	g	b	6
g	g	g	12

$= 0 \oplus 6$

Projection (Elimination)

x_i	x_j	$c(x_i,x_j)$
b	b	4
b	g	6
b	r	0
g	b	2
g	g	6
g	r	3
r	b	1
r	g	0
r	r	6

Min

Elim(**c**,$\mathbf{x_j}$) →

x_i	$f(x_i)$
b	0
g	2
r	0

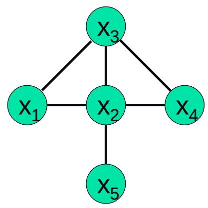
Elim(**f**,$\mathbf{x_i}$) →

$h_\emptyset$
0

Solutions

- $F(X) = \oplus c_i(X) \Downarrow_\emptyset$

Weighted CSP Example



For each vertex

x_i	$c(x_i)$
b	0
g	1
r	1

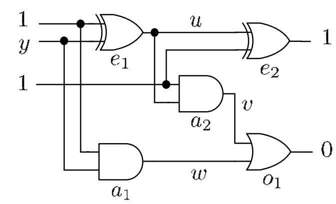
For each edge:

x_i	x_j	$c(x_i,x_j)$
b	b	**T**
b	g	0
b	r	0
g	b	0
g	g	**T**
g	r	0
r	b	0
r	g	0
r	r	**T**

F(*X*): number of non blue vertices

Probabilistic CSP Example

- AND-gates broken with 1% probability
- OR, XOR-gates broken with 5% probability
- Probabilistic valuation structure $([0,1], \geq, \cdot, 1, 0)$



Probabilistic CSP Example

f_{a1} :

a_1	w	y	
G	0	0	.99
G	1	1	.99
B	0	0	.01
B	0	1	.01
B	1	0	.01
B	1	1	.01

f_{a2} :

a_2	u	v	
G	0	0	.99
G	1	1	.99
B	0	0	.01
B	0	1	.01
B	1	0	.01
B	1	1	.01

f_{e1} :

e_1	u	y	
G	1	0	.95
G	0	1	.95
B	0	0	.05
B	0	1	.05
B	1	0	.05
B	1	1	.05

f_{o1} :

o_1	v	w	
G	0	0	.95
B	0	0	.05
B	0	1	.05
B	1	0	.05
B	1	1	.05

f_{e2} :

e_2	u	
G	0	.95
B	0	.05
B	1	.05

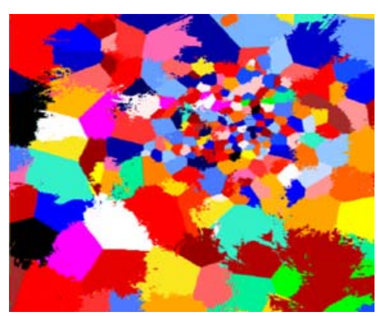
Application: Bioinformatics

- Multiple sequence alignment (DNA)
- Given *k* homologous sequences…
  - AATAATGTTATTGGTGGATCGATGA
  - ATGTTGTTCGCGAAGGATCGATAA
- … find the best alignment (sum)
  - AATAATGTTATTGGTG---GATCGATGATTA
  - ----ATGTTGTTCGCGAAGGATCGATAA---

Application: Resource Allocation

- Given a telecommunication network
- Find frequency for each communication link…
- … such that total interference is minimized

Application: Resource Allocation



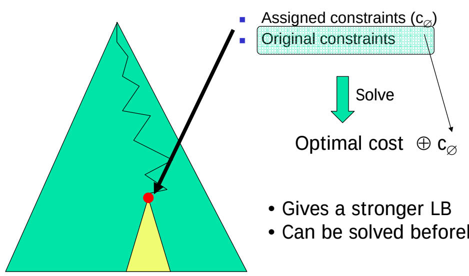
Overview

- Introduction and Definitions
- Solving soft constraints
  - By Search
  - By Inference

Depth-First Search

BT(X,D,C)
  if (X=∅) then Top := $c_\emptyset$
  else
    x_j := selectVar(X)
    forall $a \in D_j$ do
      $\forall_{c \in C \text{ s.t. } x_j \in var(c)}$ c:=Assign(c, x_j, a)
      $c_\emptyset := \Sigma_{c \in C \text{ s.t. } var(c)=\emptyset}\ c$
      if (LB<Top) then BT(X-{x_j},D-{D_j},C)

Improving the LB



- Gives a stronger LB
- Can be solved beforehand

Importance of Bounds

- Example: Frequency assignment problem
  - Instance: CELAR6-sub4
    - #var: 22 , #val: 44 , Optimum: 3230
  - Depth-first branch-and-bound search
  - UB initialized to 100000 → 3 hours
  - UB initialized to 3230 → 1 hour
    - Stochastic local search (SLS) can find UB=3230 in a few minutes

Overview

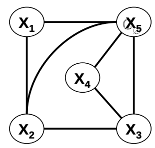
- Introduction and Definitions
- Solving soft constraints
  - By Search
  - By Inference

Synthesis

- Join all constraints
- Project
- Limitations: very costly (Time: exp(n), Space: exp(n))

Bucket Elimination

- Select a variable $\mathbf{X_i}$
- Compute the set K_i of constraints that involves this variable
- Add Elim($\oplus_{c \in K_i} c, X_1$)
- Remove variable and K_i
- Time: $\Theta(\exp(deg_i+1))$
- Space: $\Theta(\exp(deg_i))$

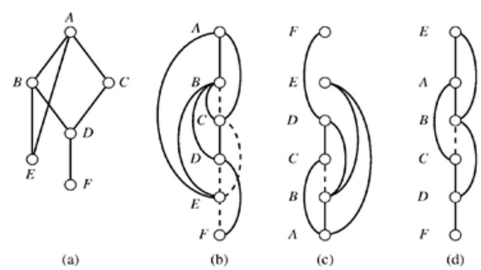


Tree Decomposition

Min-Fill Heuristics

- Input: A graph G = (V,E), V = {v_1, …, v_n}
- Output: An ordering of the nodes
- For j = 1 to n do
  - r ← a node in V with smallest number of fill edges
  - Put r in position j
  - Connect r's neighbors
  - Remove r from resulting graph

Induced Graph



Tree-structured Problems

- …

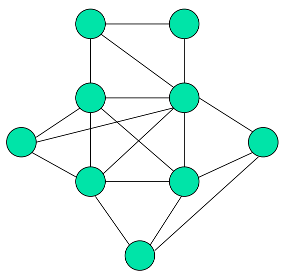
BnB with Variable Elimination

- Hybrid Method
- At each node
  - **Select an unassigned variable X_i**
  - If **$deg_i \le k$** then **eliminate X_i**
  - Else **branch on the values of X_i**

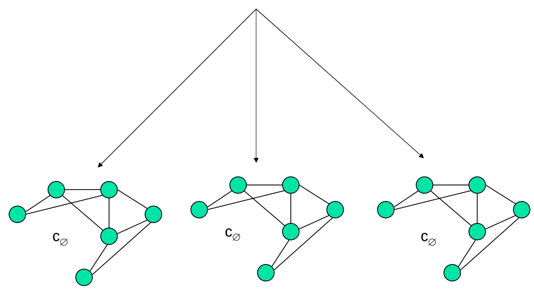
- Properties
  - BE-VE(-1) is BB
  - BE-VE(w*) is VE
  - BE-VE(1) is like cycle-cutset

BnB with Variable Elimination

- Example: BB-VE(2)



Example BnB-VE(2)



BnB with VE: Results

- Example: Still-life (academic problem)
  - Instance: n=14
    - #var:196 , #val:2
  - Branch-and-Bound ➔ 5 days
  - Variable Elimination ➔ 1 day
  - BB-VE(18) ➔ 2 seconds

Background

- Domain Splitting (e.g. Hentenryck's book)
- Bucket Elimination (and extension to super-bucket elimination/tree decomposition) (Dechter's book)
- Backtracking combined with tree decompositions (algorithm BTD, Jégou and Terrioux 03)
- Dynamic programming on tree decompositions (algorithm CTE, Dechter's book)
- Decision Diagrams (Bryant 86, Bahar 93)
- Soft constraints
- [A* search]

CSPs

- Domains $D = \{d_1, \ldots, d_n\}$
- Variables $X = \{x_1, \ldots, x_n\}$
- Constraints $C = \{c_1, \ldots, c_m\}$
- $c_j \in C$: Scope $\mathrm{var}(c_j)$, Function $\mathrm{var}(c_j) \to \{0, 1\}$

c_1:

x_1	x_2	x_3
a	a	c
a	b	c
b	b	c

c_2:

x_2	x_4
a	b
a	c
b	c

Example: 4-Queens

- Variables: Rows x_1, x_2, x_3, x_4
- Domains: Columns $1, 2, 3, 4$
- Constraints: $c_{12}(x_1, x_2)$, $c_{13}(x_1, x_3)$, $c_{14}(x_1, x_4)$, $c_{23}(x_2, x_3)$, $c_{24}(x_2, x_4)$, $c_{34}(x_3, x_4)$

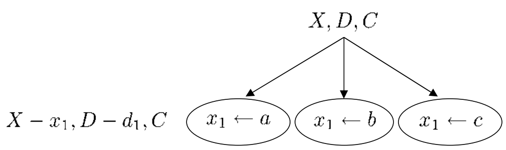
c_{12} :

x_1	x_2
1	3
1	4
2	4
3	1
4	1
4	2

	1	2	3	4
x_1				
x_2				
x_3				
x_4				

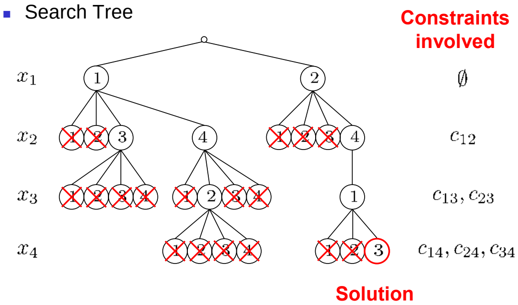
Backtracking Search

- Order on variables: $x_1 \prec \ldots \prec x_n$
- Choose value $\mathrm{val} \in d_i$ for unassigned variable x_i
- Check all completely assigned constraints
  - If inconsistent, prune and backtrack



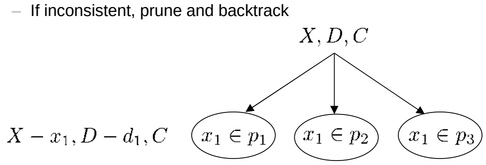
Example

- Search Tree



Generalization to Domain Splitting

- Partition domains into sets $P_i, \cup_{p \in P_i} = d_i$
- Choose subset $p \in P_i$ for unassigned variable x_i
- Check all completely assigned constraints
  - Combine (join) relevant parts of the constraints
  - If inconsistent, prune and backtrack



Example

- Partition $P_i = \{\{1,2\},\{3,4\}\},\ i = 1,2,3,4$
- E.g., check assignment $x_1 \in \{1,2\},\ x_2 \in \{3,4\}$:

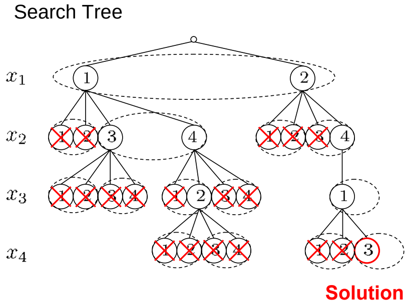
c_{12} :

x_1	x_2
1	3
1	4
2	4
3	1
4	1
4	2

Constraint satisfied ✓

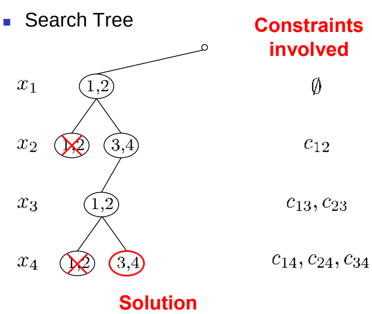
Example

- Search Tree



Example

- Search Tree

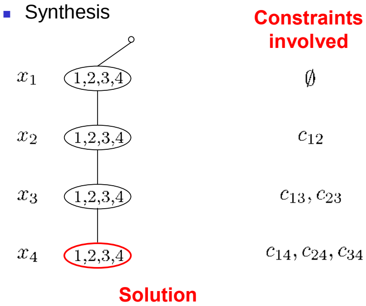


Cases

- Partition $|P_i| = |d_i|$: Limiting case of backtrack search (single assignments are tested, as before)
- Partition $|P_i| = 1$: Limiting case of constraint synthesis (single constraint is inferred): $c_1 \bowtie \ldots \bowtie c_m$
- Partition $1 < |P_i| < |d_i|$: Hybrid of search and inference (search on subsets of tuples)

Example

- Synthesis



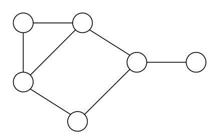
Example

- Synthesis

$$
\begin{array}{cc} \multicolumn{2}{c}{c_{12}:} \\ x_1 & x_2 \\ \hline 1 & 3 \\ 1 & 4 \\ 2 & 4 \\ 3 & 1 \\ 4 & 1 \\ 4 & 2 \end{array}
\bowtie
\begin{array}{cc} \multicolumn{2}{c}{c_{13}:} \\ x_1 & x_3 \\ \hline 1 & 2 \\ 1 & 4 \\ 2 & 1 \\ 2 & 3 \\ 3 & 2 \\ 3 & 4 \\ 4 & 1 \\ 4 & 3 \end{array}
\bowtie \ldots \bowtie
\begin{array}{cc} \multicolumn{2}{c}{c_{34}:} \\ x_3 & x_4 \\ \hline 1 & 3 \\ 1 & 4 \\ 2 & 4 \\ 3 & 1 \\ 4 & 1 \\ 4 & 2 \end{array}
=
\begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ \hline 2 & 4 & 1 & 3 \\ \ldots \end{array}
$$

Exploiting Structure

- Problem: Search is uninformed about CSP structure
  - |Pi| =|di|: leads to unnecessarily large search tree (thrashing)
  - |Pi| = 1: leads to unnecessarily large constraints
- We can do better by considering structure of graph
  - |Pi| = |di|: can be used to reduce size of search tree
  - |Pi| = 1: can be used to reduce size of constraints

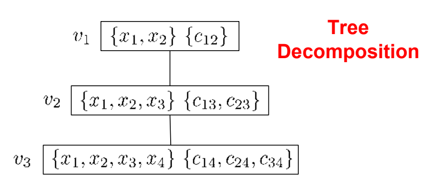


Bucket Elimination

- Define variable order $x_1 \prec \ldots \prec x_n$
- Eliminate the variables one-by-one
  - Combine constraints mentioning x_i in their scope ("bucket")
  - Project out x_i from result
- That is, variables disappear as soon as they no longer influence (cannot constrain) the result
- E.g., instead of $c_{12} \bowtie c_{13} \bowtie c_{23} \bowtie c_{14} \bowtie c_{24} \bowtie c_{34}$ bucket elimination needs to compute only $(((c_{14} \bowtie c_{24} \bowtie c_{34}) \Downarrow_{-x_4} \bowtie c_{13} \bowtie c_{23}) \Downarrow_{-x_3} \bowtie c_{12}) \Downarrow_{-x_2}$

Super-bucket Elimination

- Generalization of Bucket Elimination
- Eliminate variables in groups (i.e., in partial order)
- E.g. eliminate in order $\{x_4\} \prec \{x_3\} \prec \{x_1, x_2\}$:



Combination with Search

- To exploit decomposition in search, the order in which variables are assigned must be "compatible" with the order in which variables are eliminated
- More precisely, if variables are assigned in order $x_1 \prec \ldots \prec x_n$, variables have to be eliminated in reverse (partial) order:
  $\{x_n, \ldots, x_{n-k_1}\} \prec \{x_{n-k_1-1}, \ldots, x_{n-k_1-k_2}\} \prec \ldots$
  $\ldots \prec \{x_{n-k_1-\ldots-k_j-1}, \ldots, x_1\}$
- Construct (super-)buckets (tree decomposition scheme) from this reverse order

Combination with Search (Cont'd)

- A tree decomposition with compatible elimination order can be exploited during search as follows:
  - Let separator(vj) denote the separator of tree node vj (the set of variables that vj shares with its parent, vi)
  - Once a complete assignment has been found for a subtree, record it as a good at the separator (same for nogoods)
  - By checking the goods/nogoods during search, we can then avoid descending into the same subtree again and again
- This algorithm is called BTD (backtracking with tree decompositions) (Jégou Terrioux AIJ03)

BTD (Jégou Terrioux AIJ03)

- Input: (Partial) assignment A, tree node v_i, set of variables $X_{v_i} \subseteq \text{var}(v_i)$ to be assigned
- Output: "True" if assignment is consistent with all constraints C in subtree of v_i, "false" otherwise
- Initial call: BTD($\emptyset, v_1, \text{vars}(v_1)$)

BTD Pseudocode

- Function BTD(A,vi,Xvi)

```
If Xvi = ∅ Then
  Consistent ← True
  F ← children(vi)
  While F ≠ ∅ And Consistent Do
    Choose vj ∈ F
    F ← F \ {vj}
    If A ↓ separator(vi) is a good of vi/vj Then Consistent ← True
    Else
      If A ↓ separator(vj) is a nogood of vi/vj Then Consistent ← False
      Else
        Consistent ← BTD(A,vj,vars(vj) \ separator(vj))
        If Consistent Then
          Record the good A ↓ separator(vj) for vi/vj
        Else
          Record the nogood (A ↓ separator(vj)) for vi/vj
        End If
      End If
    End if
  End While
  Return Consistent
```

(continued on next slide)

BTD Pseudocode (Cont'd)

- *(Continued)*

```
Else
  Choose x ∈ Xvi
  dom ← Dx
  Consistent ← False
  While dom ≠ ∅ And Not Consistent Do
    Choose val ∈ dom
    dom ← dom \ {val}
    If (A ∧ {x ← val}) semijoin {c: c ∈ C ∧ var(c) ⊆ (var(A)∪{x})} ≠ ∅
    Then
      Consistent ← BTD(A ∧ {x ← val}, vi, Xvi \ {x})
    End If
  End While
  Return Consistent
End If
```

Note: Computes a full assignment, but returns only true/false.

Generalization to Domain Splitting

- Incorporate domain splitting into BTD, that is, search over sets of assignments A
- Yields new algorithm BTDS (backtracking with tree decompositions and domain splitting)
- Like BTD, BTDS records set of good tuples and nogood tuples for each separator
- Unlike BTD, BTDS maintains only assignments to v_i (instead of full assignment)
- Unlike BTD, BTDS returns assignments to separator of v_i (instead of only true/false)

BTDS

- Input: Set of (partial) assignments (constraint) A, tree node v_i, set of variables $X_{v_i} \subseteq \mathrm{var}(v_i)$
- Output: Assignments to separator of v_i that are consistent with all constraints C in subtree of v_i
- Initial call: BTDS($\emptyset, v_1, \mathrm{vars}(v_1)$)

BTDS Pseudocode

- Function BTDS(A,vi,Xvi)

```
If Xvi = ∅ Then
 F ← children(vi)
 While F ≠ ∅ And A ≠ ∅ Do
  Choose vj ∈ F
  F ← F \ {vj}
  Asep ← A ⇓ separator(vj)
  Aseprest ← Asep \ (goods(vj) ∪ nogoods(vj))
  If Aseprest ≠ ∅ Then
   Aseprestcons ← BTDS(Aseprest, vj, χ(vj) \ separator(vj))
   goods(vj) ← goods(vj) ∪ Aseprestcons
   nogoods(vj) ← nogoods(vj) ∪ (Aseprest \ Aseprestcons)
  End If
  A ← A semijoin goods(vj)
  End While
 Return A ⇓ separator(vi)
```

(continued on next slide)

BTDS Pseudocode (Cont'd)

- *(Continued)*

```
Else
 Choose x ∈ Xvi
 PartitionElements ← Px
 Aextended ← A
 While PartitionElements ≠ ∅ And Aextended = ∅ Do
  Choose p ∈ PartitionElements
  PartitionElements ← PartitionElements \ {p}
  Aextended ← (A ∧ {x ← p}) semijoin {c: c ∈ C ∧ var(c) ⊆ (var(A)∪{x})}
  If Aextended ≠ ∅ Then
   Aextended ← BTDS(Aextended, vi, Xvi \ {x})
  End If
 End While
 Return Aextended
End If
```

Cases

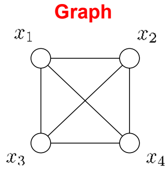
- Partition $|P_i| = |d_i|$: Yields backtracking algorithm BTD (Jégou Terrioux AIJ03))
- Partition $|P_i| = 1$: Yields dynamic programming algorithm CTE (Dechter 03)
- Partition $1 < |P_i| < |d_i|$: Hybrid of BTD and CTE

Note: for case |Pi|>1, algorithm has higher space complexity than CTE (exp(width) instead of exp(sep)). But, it should be possible to reduce the space complexity to exp(sep).

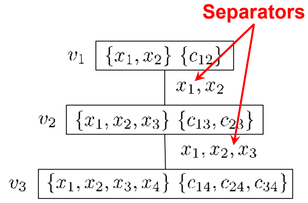
BTDS applied to 4-Queens

- Variable order $x_1 \prec x_2 \prec x_3 \prec x_4$
- Partition $P_i = \{\{1,2\},\{3,4\}\},\ i = 1,2,3,4$

Constraint Graph



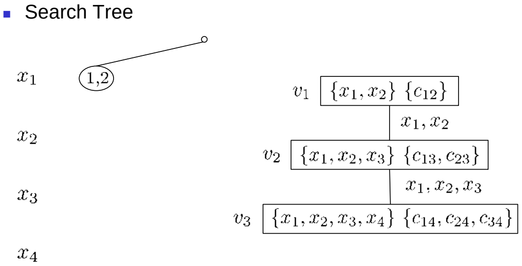
Separators



Tree Decomposition

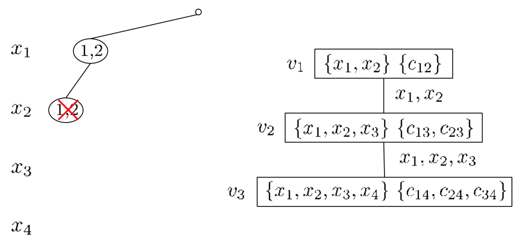
BTDS applied to 4-Queens

- Search Tree



BTDS applied to 4-Queens

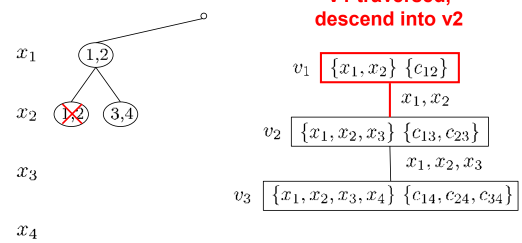
- Search Tree



BTDS applied to 4-Queens

- Search Tree

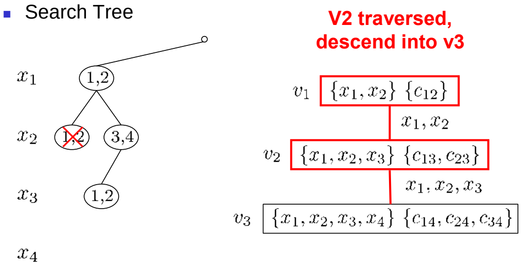
V1 traversed,
descend into v2



BTDS applied to 4-Queens

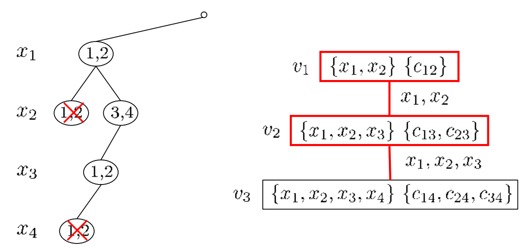
- Search Tree

V2 traversed,
descend into v3



BTDS applied to 4-Queens

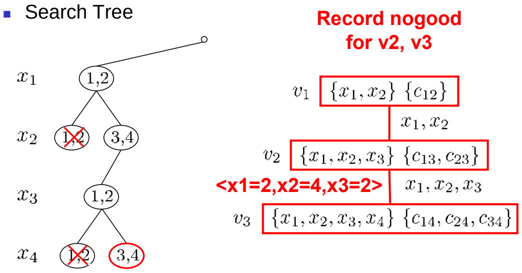
- Search Tree



BTDS applied to 4-Queens

- Search Tree

Record nogood
for v2, v3



Solution

Granularity of Domain Splitting

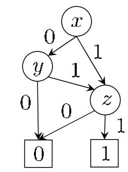
- Empirical observation and theoretical considerations (Jégou Terrioux AIJ03): BTD outperforms CTE (cluster tree elimination, i.e. dynamic programming on tree decomposition)
  - BTD is a “lazy” variant of CTE (dynamic programming)
- Therefore, $|P_i| = |d_i|$ (finest granularity) is optimal granularity of partitions in BTDS
  - Best to perform dynamic programming as lazily as possible
- But: This assumes that tuples are handled *explicitly*
  - More efficient, implicit datastructures are possible when manipulating whole *sets of tuples*

Symbolic Encoding

- Decision diagrams (Bryant 86): graph-based, canonical representation of (boolean) functions
- Time and space complexity depends on graph size rather than number of tuples of function represented

x	y	z
0	1	1
1	0	1
1	1	1

Function



Decision Diagram (ROBDD)

BTDS with Symbolic Encoding

- In many practical cases, decision diagrams much more compact than representing tuples explicitly
  - → Can make operations on sets of tuples (inference) more efficient
  - → But won't make operations on single tuples (search) more efficient
- Therefore, in BTDS, larger partition elements become more advantageous (shifts optimal granularity towards $|P_i| = 1$)
  - → In many practical cases, optimal granularity for partitions in BTDS becomes 1 < |Pi| < |di|
  - → Exploit both structure in graph and structure in tuples

Generalization to Optimization

- Domains $D = \{d_1, \ldots, d_n\}$
- Variables $X = \{x_1, \ldots, x_n\}$
- Constraints $C = \{c_1, \ldots, c_m\}$
- $c_j \in C$: Scope $\text{var}(c_j)$, Function $\text{var}(c_j) \to E$
- Valuation structure $(E, \preceq, \oplus, \bot, \top)$ **$\bot$ best, $\top$ worst**

c_1:

x_1	x_2	x_3	
a	a	c	.5
a	b	c	.5
b	b	c	.5

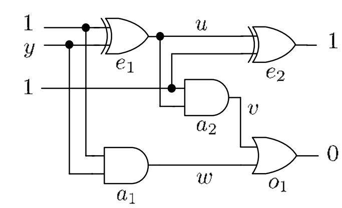
c_2:

x_2	x_4	
a	b	.4
a	c	.4
b	c	.4

Soft Constraints

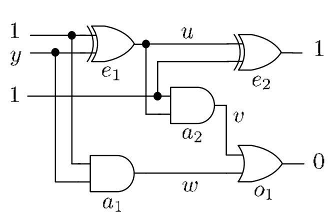
Example: Full Adder Diagnosis

- Variables $\{u, v, w, y, a_1, a_2, e_1, e_2, o_1\}$
- $\{a_1, a_2, e_1, e_2, o_1\}$ describe modes of gates
- Gates are either in good (G) or broken (B) mode



Example: Full Adder Diagnosis

- AND-gates broken with 1% probability
- OR, XOR-gates broken with 5% probability
- Probabilistic valuation structure $([0,1], \geq, \cdot, 1, 0)$



Example: Soft Constraints

f_{a1} :

a_1	w	y	
G	0	0	.99
G	1	1	.99
B	0	0	.01
B	0	1	.01
B	1	0	.01
B	1	1	.01

f_{a2} :

a_2	u	v	
G	0	0	.99
G	1	1	.99
B	0	0	.01
B	0	1	.01
B	1	0	.01
B	1	1	.01

f_{e1} :

e_1	u	y	
G	1	0	.95
G	0	1	.95
B	0	0	.05
B	0	1	.05
B	1	0	.05
B	1	1	.05

f_{o1} :

o_1	v	w	
G	0	0	.95
B	0	0	.05
B	0	1	.05
B	1	0	.05
B	1	1	.05

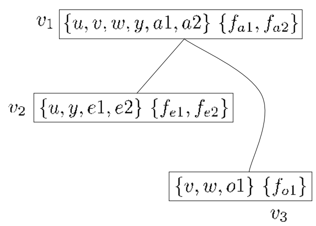
f_{e2} :

e_2	u	
G	0	.95
B	0	.05
B	1	.05

For details, see ECAI'04 paper.

Example: Tree Decomposition

- Eliminination order $\{o_1\} \prec \{e_1, e_2\} \prec \{u, v, w, y, a_1, a_2\}$



Depth-First Branch and Bound

- Recursive algorithm BTDval (Terrioux Jégou CP03) (back-tracking with tree decompositions for valued constraints) that extends BTD to soft constraints
- Records tuples and their values for each separator ("valued goods" instead of goods and nogoods)

BTDval (Terrioux Jégou CP03)

- Input: (Partial) assignment A, tree node v_i, set of variables $X_{v_i} \subseteq \mathrm{vars}(v_i)$, lower bound l_{v_i} (value of assignment so far), upper bound u_{v_i} (value of best solution found so far)
- Output: Value of best extension to subtree of v_i with value < u_{v_i}, or some value ≥ u_{v_i}, if that does not exist
- Initial call: BTDval($\emptyset, v_1, \mathrm{var}(v_1), \bot, \top$)

BTDval Pseudocode

- Function BTDval(A,vi,Xvi,lvi,uvi)

```
If Xvi = ∅ Then
  F ←  children(vi)
  While F ≠ ∅ And lci < uvi  Do
    Choose vj ∈ F
    F ← F \ {vj}
    If ⟨A ↓ separator(vj), v⟩ is a good of vi/vj Then lci ← lci ⊕ v
    Else
      v ← BTDval(A,vj,vars(vj) \ separator(vj), ⊥, uvi)
      lci ← lci ⊕ v
      Record the goods ⟨A ↓ separator(vj), v⟩ for vi/vj
    End If
  End While
  Return lci
```

(continued on next slide)

BTDval Pseudocode (Cont'd)

- *(Continued)*

```
Else
  Choose x ∈ Xvi
  dom ← Dx
  While dom ≠ ∅ And lvi < uvi Do
    Choose val ∈ dom
    dom ← dom \ {val}
    lval ← ((A ∧ {x ← val}) semijoin {c: c ∈ C ∧ var(c)⊆(var(A)∪{x})})↓∅
    If lvi ⊕ lval < uvi Then
      uvi ← min(uvi, BTDval(A ∧ {x ← val}, vi, Xvi \ {x}, lvi ⊕ lval, uvi)
    End If
  End While
  Return uvi
End If
```

Note: Computes a full assignment, but returns only a value.

Generalization to Domain Splitting

- Incorporate domain splitting into BTDval, that is, search over a whole set of valued assignments
- Yields BTDSval (backtracking with tree decompositions and domain splitting for valued constraints)
- Like BTDval, BTDSval records valued goods
- Unlike BTDval, BTDSval maintains only assignments to v_i, and returns assignments to separator of v_i

Sinking Operation

- Sinking operation (Bistarelli et al. SOFT03, Morris AAAI93): $\mathrm{sink}(c_j, \alpha)$ returns a new constraint where all values of tuples $\succeq \alpha$ have been replaced by $\top$

Constraint

f_{e2} :

e_2	u	
G	0	.95
B	0	.05
B	1	.05

Constraint sink(f_{e2},0.05)

e_2	u	
G	0	.95
B	0	0
B	1	0

Generalized Sinking Operation

- Generalized sinking operation $\mathrm{sink}(c_i, c_j)$ returns a new constraint where all values of tuples of c_i that are $\succeq$ values of tuples of c_j have been replaced by $\top$
- Generalizes the check $l_{v_i} < u_{v_i}$ to soft constraints

Constraints

f_{e2} :

e_2	u	
G	0	.95
B	0	.05
B	1	.05

f :

e_2	u	
G	0	.45
B	0	.05
B	1	.01

Constraint sink(f_{e2},f)

e_2	u	
G	0	.95
B	0	0
B	1	.5

BTDSval

- Input: Set of (partial) assignments (constraint) f_{A_i} tree node v_i, variables $X_{v_i} \subseteq \mathrm{vars}(v_i)$, upper bound f_u
  - No explicit lower bound (contained in valued assignments)
  - Note that the bounds (lower and upper) are now functions
- Output: Best assignments to separator of v_i with values $\prec f_u$, or values $\succeq f_u$, if not existent
- Notation: $f_\bot$: constraint with value $\bot$ for all tuples, $f_\top$: constraint with value $\top$ for all tuples
- Initial call: BTDSval($f_\bot, v_1, \mathrm{var}(v_1), f_\top$)

BTDSval Pseudocode

```
Function BTDSval(fa,vi,Xvi,fu)
  If Xvi = ∅ Then
    F ← children(vi)
    fa ← sink(fa,fu)
    While F ≠ ∅ And fa ≠ f⊤  Do
      Choose vj ∈ F
      F ← F \ {vj}
      fasep ← fa ⇓ separator(vj)
      faseprest  ← tuples of fasep that are not goods of vi/vj
      If faseprest ≠ f⊤  Then
        farestval ← BTDS(faseprest, vj, χ(vj) \ separator(vj), fu)
        Record tuples in farestval as goods of vi/vj
      End If
      fa ← fa semijoin goods(vj)
      fa ← sink(fa,fu)
    End While
  Return fa ⇓ separator(vi)
  (continued on next slide)
```

BTDSval Pseudocode (Cont'd)

```
(Continued)
  Else
    Choose x ∈ Xvi
    PartitionElements ← Px
    fa ← sink(fa,fu)
    Aextended ← fa
    While PartitionElements ≠ ∅ And Aextended ≠ f⊤ Do
      Choose p ∈ PartitionElements
      PartitionElements ← PartitionElements \ {p}
      Aextended ← (fa ∧ {x ← p}) semijoin {c: c ∈ C ∧ var(c) ⊆ (var(fa)∪{x})}
      Aextended ← sink(Aextended,fu)
      If Aextended ≠ f⊤ Then
        Aextended ← BTDS(Aextended, vi, Xvi \ {x}, fu)
        fu ← min(fu,Aextended)
      End If
    End While
    Return fu
  End If
```

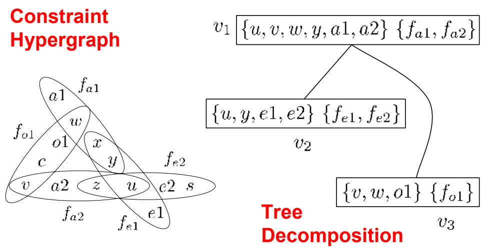
Cases

- Partition $|\ P_i\ | = |\ d_i\ |$: Yields backtracking algorithm BTDval (Jégou Terrioux CP03)
- Partition $|\ P_i\ | = 1$: Yields dynamic programming algorithm CTE with soft constraints (Dechter 03)
- Partition $1 < |\ P_i\ | < |\ d_i\ |$: Hybrid of BTDval and CTE

Note: for case |Pi|>1, algorithm has higher space complexity than CTE (exp(width) instead of exp(sep)). But, it should be possible to reduce the space complexity to exp(sep).

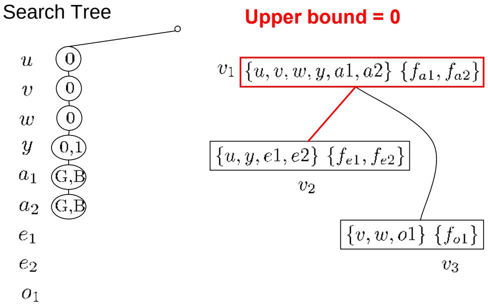
BTDSval applied to Full Adder

- Partition $P_u, P_v, P_w = \{\{0\},\{1\}\}$, all else $P_i = \{d_i\}$



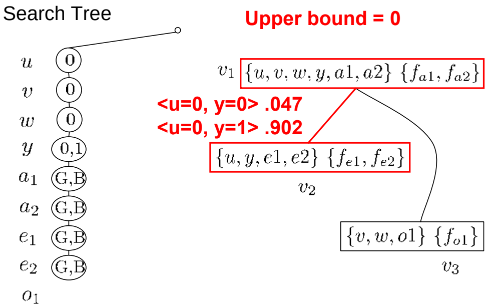
BTDSval applied to Full Adder

- Search Tree



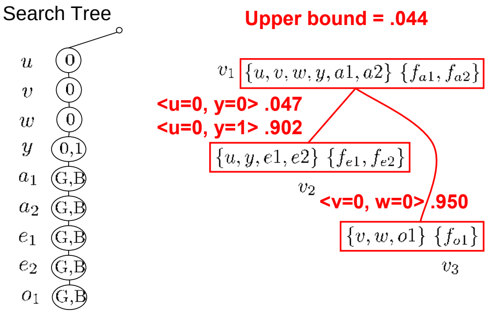
BTDSval applied to Full Adder

- Search Tree



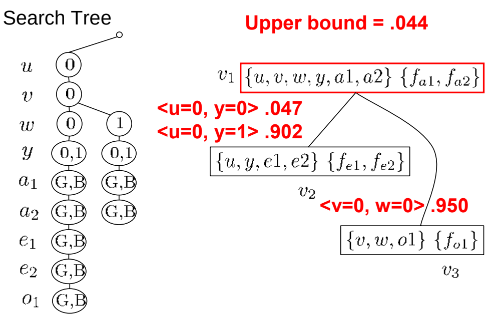
BTDSval applied to Full Adder

- Search Tree



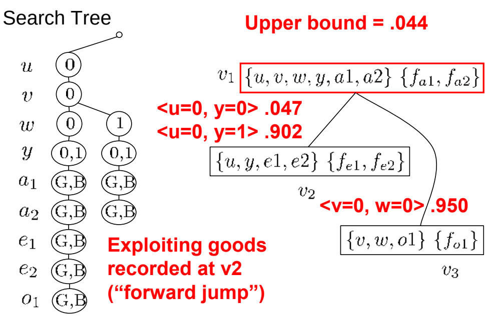
BTDSval applied to Full Adder

- Search Tree



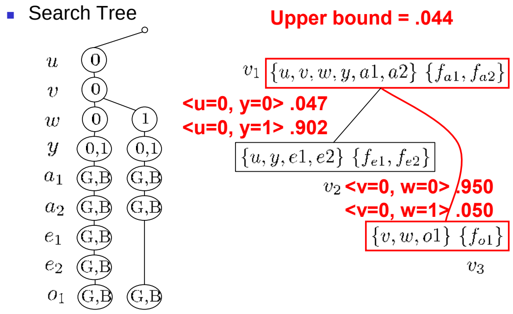
BTDSval applied to Full Adder

- Search Tree



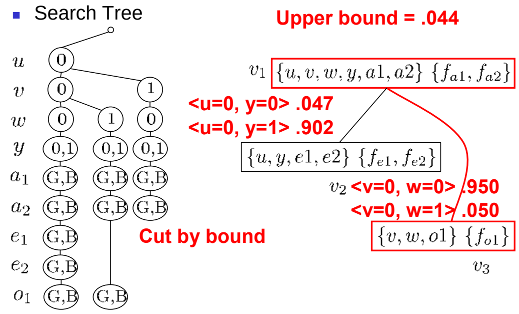
BTDSval applied to Full Adder

- Search Tree



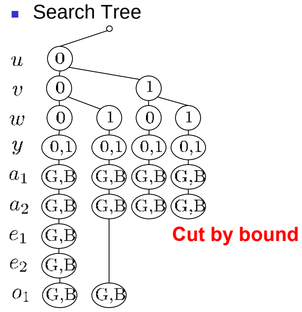
BTDSval applied to Full Adder

- Search Tree



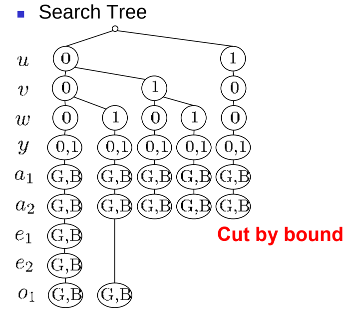
BTDSval applied to Full Adder

- Search Tree



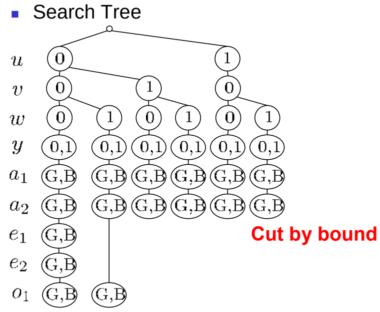
BTDSval applied to Full Adder

- Search Tree



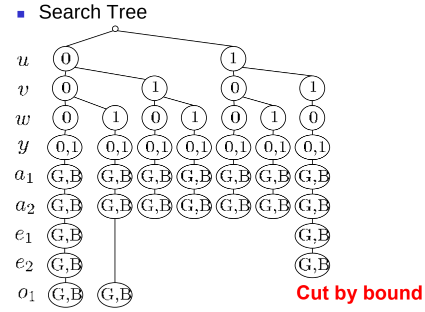
BTDSval applied to Full Adder

- Search Tree



BTDSval applied to Full Adder

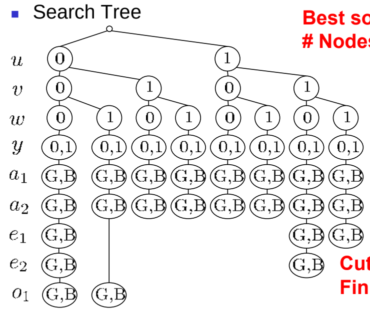
- Search Tree



BTDSval applied to Full Adder

- Search Tree

Best solution = .044
 # Nodes = 45



Cut by bound
Finished.

BTDSval with Symbolic Encoding

- Algebraic Decision Diagrams (ADDs, Bahar 93): graph-based, canonical representation of functions with non-binary values
- When encoding constraints as DDs in BTDSval, then like for BTDS, larger partition elements become more advantageous (shifts the optimal granularity towards $|P_i| = 1$)
  - → In many practical cases, optimal granularity for partitions in BTDSval becomes 1 < |Pi| < |di|

Experimental Results

- …

Best-First Search

- Replace depth-first branch-and-bound search in BTDval by best-first (A*) search
- Yields algorithm ATDval (A* search with tree decompositions for valued constraints)
- One search queue per each tree node v_i
- Search queues have entries $\langle A, v, v_i, X_{v_i}, F \rangle$
  - A: assignment
  - v: value
  - vi: tree node
  - Xvi: set of variables
  - F: set of children of vi

Best-First Search

- Problem: Search to be performed given a particular assignment A; values depend on this assignment
- Therefore, would have to maintain different search queues for each different assignment!
- Possible solution: Switch to dual problem (unary soft constraints, n-ary hard equality constraints)
  - See SOFT-04 paper

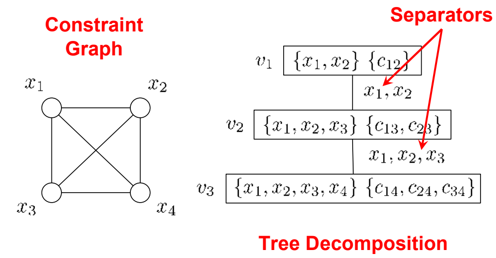
Related Work

- Set-based search (Jörg Denzinger, U Calgary)

Material

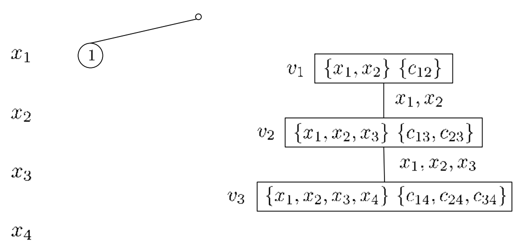
BTD applied to 4-Queens

- Variable order $x_1 \prec x_2 \prec x_3 \prec x_4$



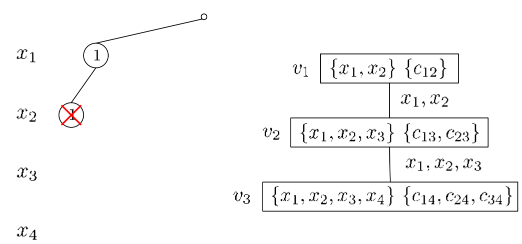
BTD applied to 4-Queens

- Search Tree



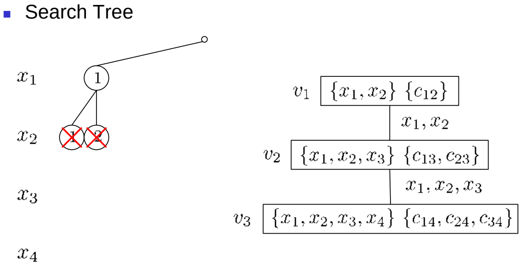
BTD applied to 4-Queens

- Search Tree



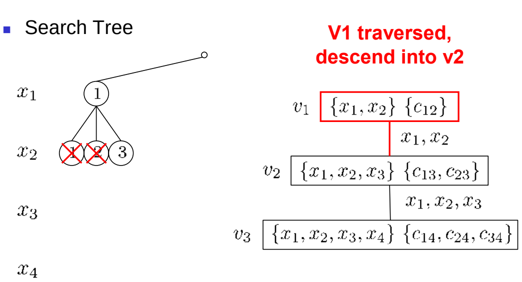
BTD applied to 4-Queens

- Search Tree



BTD applied to 4-Queens

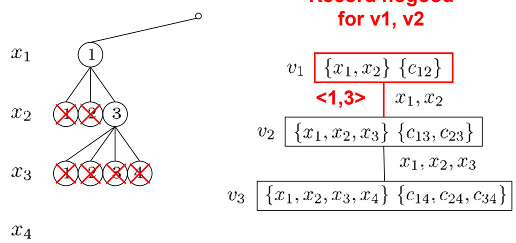
- Search Tree



BTD applied to 4-Queens

- Search Tree

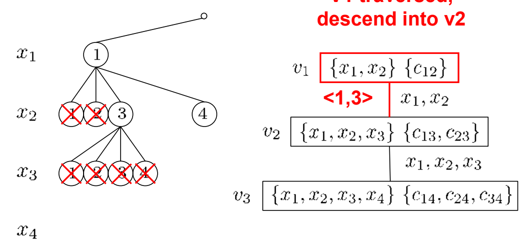
Record nogood for v1, v2



BTD applied to 4-Queens

- Search Tree

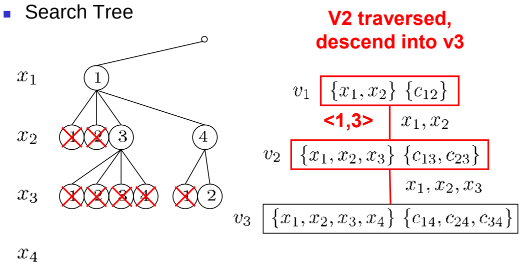
V1 traversed, descend into v2



BTD applied to 4-Queens

- Search Tree

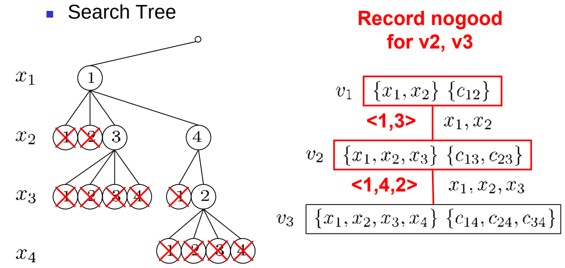
V2 traversed, descend into v3



BTD applied to 4-Queens

- Search Tree

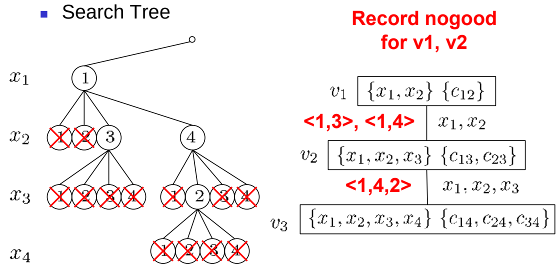
Record nogood for v2, v3



BTD applied to 4-Queens

- Search Tree

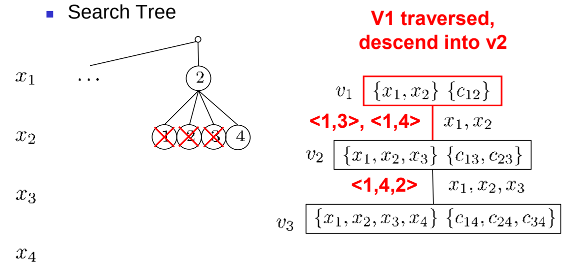
Record nogood for v1, v2



BTD applied to 4-Queens

- Search Tree

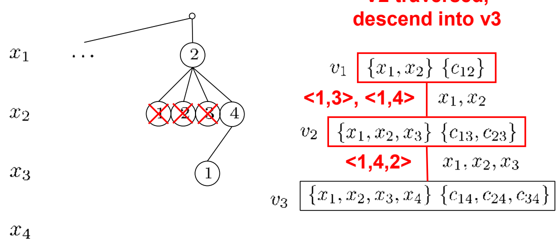
V1 traversed, descend into v2



BTD applied to 4-Queens

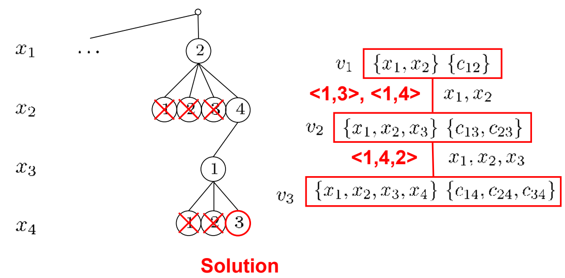
- Search Tree

V2 traversed, descend into v3



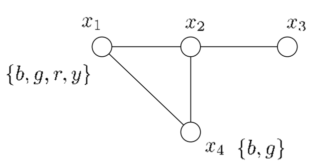
BTD applied to 4-Queens

- Search Tree



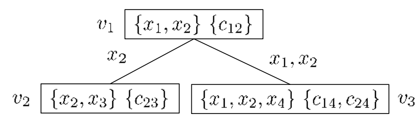
Example: "Soft" Graph Coloring

- Variables: x_1, x_2, x_3, x_4
- Domains: $\{b, g, r, y\}$ for x_1, x_2, x_3, $\{b, g\}$ for x_4
- Constraints:
  - Adjacent colors must be different
  - Combinations red and blue, red and green have penalty



Example: "Soft" Graph Coloring

- Tree Decomposition:



Example: "Soft" Graph Coloring

- Partitions: $\{\{b, g\}, \{r, y\}\}$ for x_1, x_2, x_3, $\{\{b, g\}\}$ for x_4