Diagnosis using Bounded Search and Symbolic Inference

Martin Sachenbacher and Brian C. Williams
MIT CSAIL

Overview

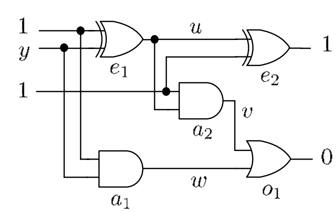
- Soft Constraints Framework
- Characterizing Structure in Soft Constraints
- Exploiting Structure in Soft Constraints
  - Set-based Search
  - Decomposition-based Search
  - Hybrids

Overview

- **Soft Constraints Framework**
- Characterizing Structure in Soft Constraints
- Exploiting Structure in Soft Constraints
  - Set-based Search
  - Decomposition-based Search
  - Hybrids

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



Valued CSP

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

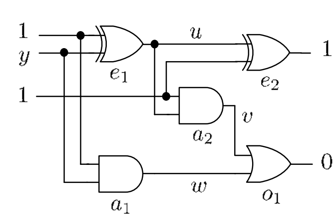
- set of **variables** $X = \{x_1, \ldots, x_n\}$
- set of **domains** $D = \{d_1, \ldots, d_n\}$
- set of **constraints** $F = \{f_1, \ldots, f_m\}$
- **valuation structure** $S = \langle E, \oplus, \preceq, \perp, \top \rangle$

A constraint $f \in F$ is a **function** $f : d_1 \times \ldots \times d_n \to E$ mapping assignments over X to valuations in E .

The complete **valuation** of an assignment t is $\bigoplus_{c \in C} c(t)$.

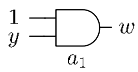
Example: Full Adder Diagnosis

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



Example: Full Adder Diagnosis

- Model gates $f_{a1}, f_{a2}, f_{e1}, f_{e2}, f_{o1}$ as soft constraints



f_{a1} :

	a_1	y	w	$\dots$	
Gate is good	G	0	0	$\dots$	.99
	G	1	1	$\dots$	.99
Gate is broken	B	0	0	$\dots$	.01
	B	0	1	$\dots$	.01
	B	1	0	$\dots$	.01
	B	1	1	$\dots$	.01

Inferring Solutions

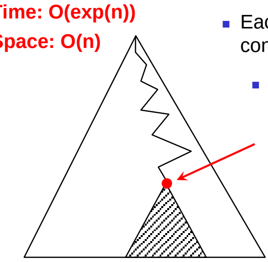
- Constraint network $\langle X, D, F, S \rangle$
- Value of optimal solution obtained by **combining** the constraints:

$$\min_t \bigoplus_{c \in C} c(t)$$

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

Branch-and-Bound Search

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



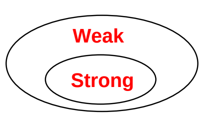
- Each search node is a soft constraint subproblem
  - **Lower Bound (lb):** Optimistic estimate of best solution in subtree
    - **Upper Bound (ub):** Best solution found so far
      - Prune, if **lb ≥ ub**.

Overview

- Soft Constraints Framework
- **Characterizing Structure in Soft Constraints**
- Exploiting Structure in Soft Constraints
  - Set-based Search
  - Decomposition-based Search
  - Hybrids

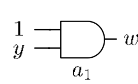
Structure in Soft Constraints

- Structure can be characterized as **independence properties** of functions f_i.
- **"Strong"** independence: Function f_i independent of all assignments to some variables $Z \subseteq X$.
- **"Weak"** independence: Function f_i independent of some assignments to some variables $Z \subseteq X$.



Strong Independence

- Support $\mathrm{sup}(f) := \{x_i \in X \mid \exists v_1, v_2 \in d_i \text{ s.t.}$
  $(f \oplus (x_i \leftarrow v_1)) \Downarrow_{X \setminus \{x_i\}} \neq f \oplus (x_i \leftarrow v_2)) \Downarrow_{X \setminus \{x_i\}}\}$



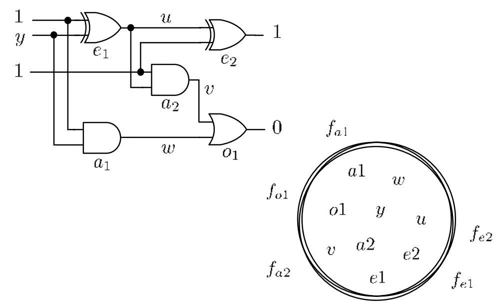
$\mathrm{sup}(f_{a1}) = \{a_1, y, w\}$

Subset of variables that function depends upon.

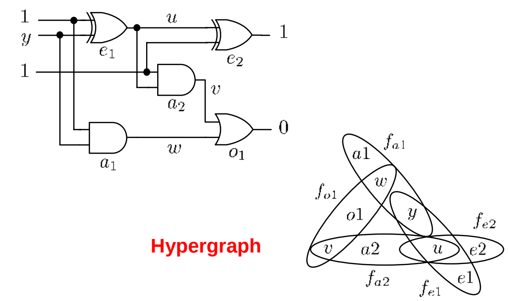
f_{a1} :

a_1	y	w	$\dots$	
G	0	0	$\dots$	.99
G	1	1	$\dots$	.99
B	0	0	$\dots$	.01
B	0	1	$\dots$	.01
B	1	0	$\dots$	.01
B	1	1	$\dots$	.01

Strong Independence



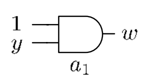
Strong Independence



Hypergraph

Weak Independence

- E.g. for constraint f_{a1}: if $a_1 = B$, then value is $.01$ regardless of y and w



f_{a1} :

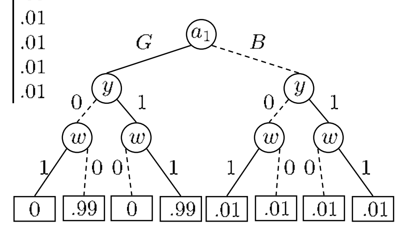
a_1	y	w	$\ldots$	
G	0	0	$\ldots$	.99
G	1	1	$\ldots$	.99
B	0	0	$\ldots$	.01
B	0	1	$\ldots$	.01
B	1	0	$\ldots$	.01
B	1	1	$\ldots$	.01

Sharing of common subassignments.

Weak Independence

f_{a1} :

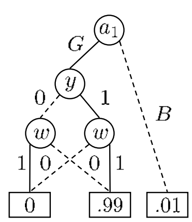
a_1	y	w	$\ldots$	
G	0	0	$\ldots$	.99
G	1	1	$\ldots$	.99
U	0	0	$\ldots$	.01
U	0	1	$\ldots$	.01
U	1	0	$\ldots$	.01
U	1	1	$\ldots$	.01



Weak Independence

f_{a1} :

a_1	y	w	$\ldots$	
G	0	0	$\ldots$	.99
G	1	1	$\ldots$	.99
U	0	0	$\ldots$	.01
U	0	1	$\ldots$	.01
U	1	0	$\ldots$	.01
U	1	1	$\ldots$	.01



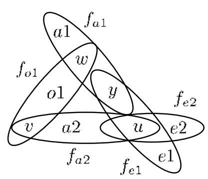
Algebraic Decision Diagram

Overview

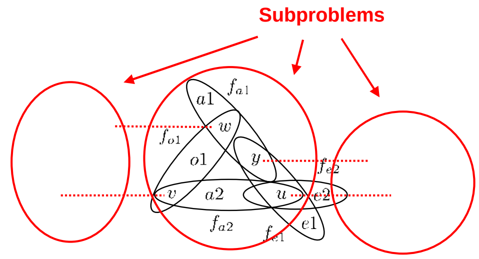
- Soft Constraints Framework
- Characterizing Structure in Soft Constraints
- **Exploiting Structure in Soft Constraints**
  - **Decomposition-based Search**
  - Set-based Search
  - Hybrids

Exploiting Strong Independence

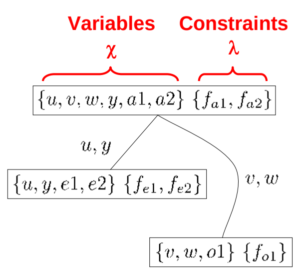
- Principle: Strong independence allows to **decompose** the problem into **subproblems** with smaller sets of variables.



Tree Decomposition

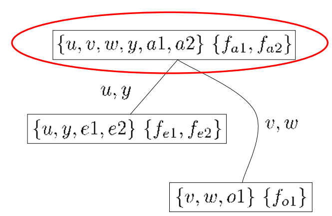


Tree Decomposition



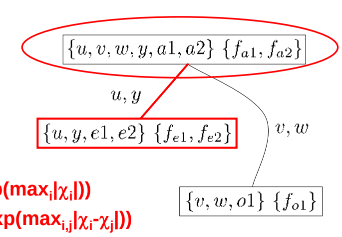
BnB with Tree Decomposition

- Algorithm BTD (Terrioux and Jégou CP-03)
- Record solutions for subproblems ("structural goods")



BnB with Tree Decomposition

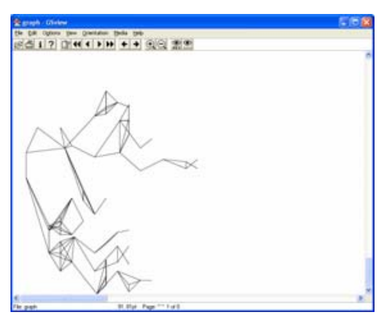
- Algorithm BTD (Terrioux and Jégou CP-03)
- Record solutions for subproblems ("structural goods")



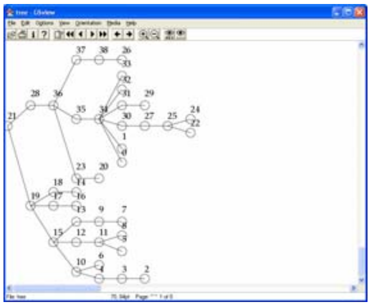
Time: O(exp($\max_i|\chi_i|$))

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

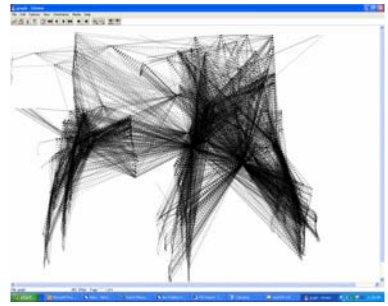
EO-1 Model: Constraint Graph



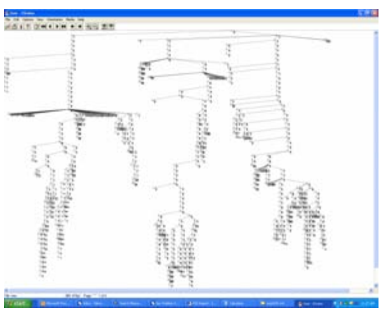
EO-1 Model: Tree Decomposition



Ssa2670-141 Circuit: Graph



Ssa2670-130 Circuit: Tree

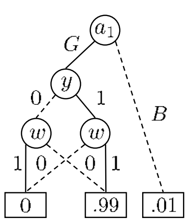


Overview

- Soft Constraints Framework
- Characterizing Structure in Soft Constraints
- **Exploiting Structure in Soft Constraints**
  - Decomposition-based Search
  - **Set-based Search**
  - Hybrids

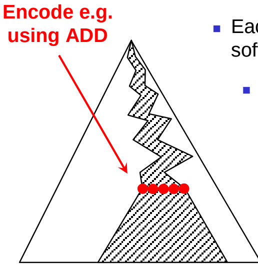
Exploiting Weak Independence

- Principle: Weak independence allows to consider **sets of assignments** at once instead of individual assignments.



Set-based Branch-and-Bound

Encode e.g. using ADD

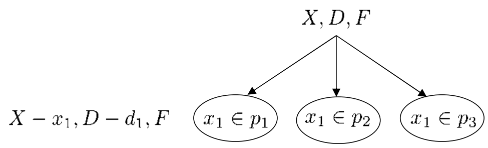


- Each search node is a **set of** soft constraint subproblems
  - **Lower Bound Function (f_{lb}):** Optimistic estimates of best solutions in subtree
    - **Upper Bound (ub):** Best solution found so far
      - Prune, if $f_{lb} \geq$ **ub**.

Domain Splitting

Generalize to search over **sets**:

- Partition domains into sets $P_i, \cup_{p \in P_i} = d_i$
- Choose **subset** $p \in P_i$ for unassigned variable x_i



Example: 4-Queens

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

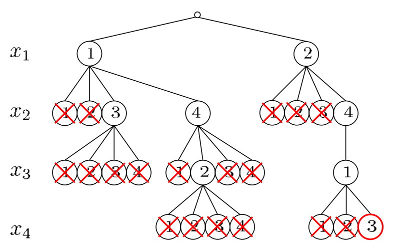
$f_{12}:$

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

	1	2	3	4
x_1		Q		
x_2				Q
x_3	Q			
x_4			Q	

Example

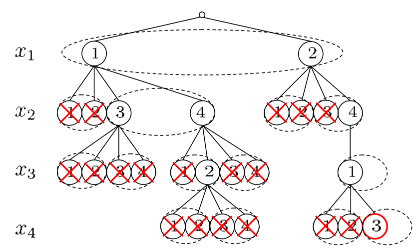
- Search Tree



Solution

Example

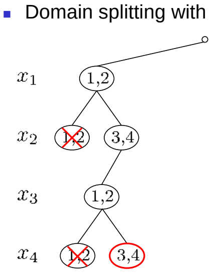
- Domain splitting with partitions $P_i = \{\{1, 2\}, \{3, 4\}\}$



Solution

Example

- Domain splitting with partitions $P_i = \{\{1, 2\}, \{3, 4\}\}$



Solution

Results

- C++ Implementation of SBBTD on Pentium 4 with 1 GB RAM, using ADD library from CUDD package
- Weighted version of 16-Queens-Problem (16 variables, 136 constraints, domain size 16)
- Using partition {{0,...,15}}: out of memory (> 1 GB)
- Using partition {{0},{1},…, {15}}: out of time (>10 min)
- Using partition {{0,...,7},{8,...,15}}: 104.8 sec

Overview

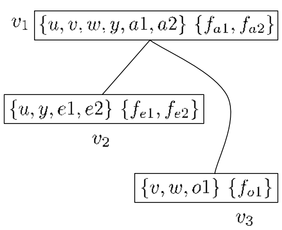
- Soft Constraints Framework
- Characterizing Structure in Soft Constraints
- **Exploiting Structure in Soft Constraints**
  - Set-based Search
  - Decomposition-based Search
  - **Hybrids**

Hybrid Algorithm SBBTD

- Exploits **both** strong independence using tree decomposition, and weak independence using set-based search.

SBBTD applied to Full Adder

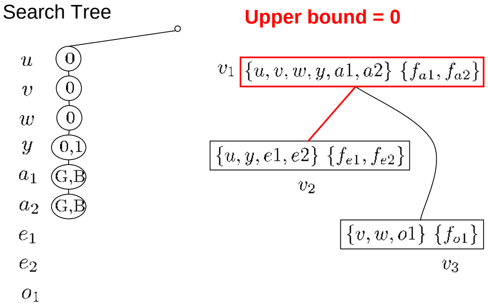
- Partition $P_u, P_v, P_w = \{\{0\},\{1\}\}$, all else $P_i = \{d_i\}$



SBBTD applied to Full Adder

- Search Tree

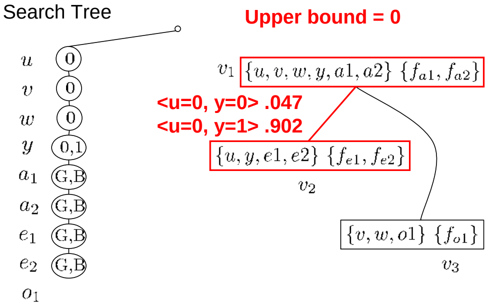
Upper bound = 0



SBBTD applied to Full Adder

- Search Tree

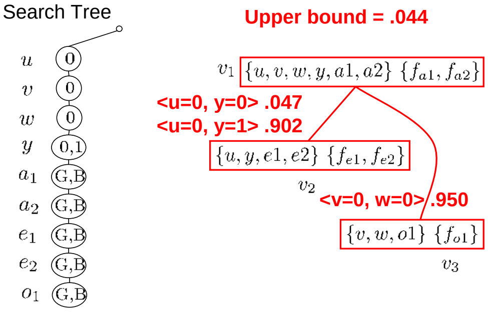
Upper bound = 0



SBBTD applied to Full Adder

- Search Tree

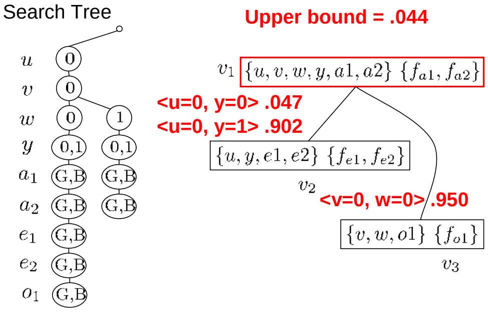
Upper bound = .044



SBBTD applied to Full Adder

- Search Tree

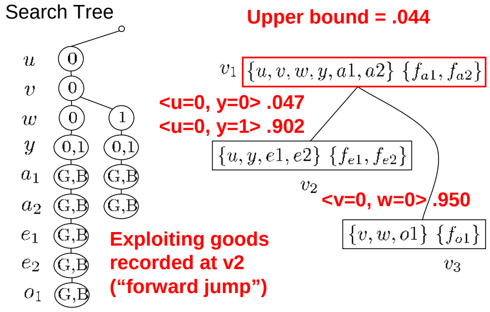
Upper bound = .044



SBBTD applied to Full Adder

- Search Tree

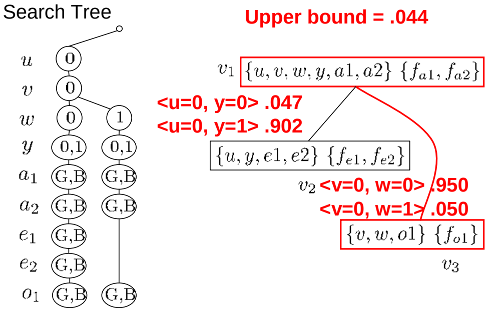
Upper bound = .044



SBBTD applied to Full Adder

- Search Tree

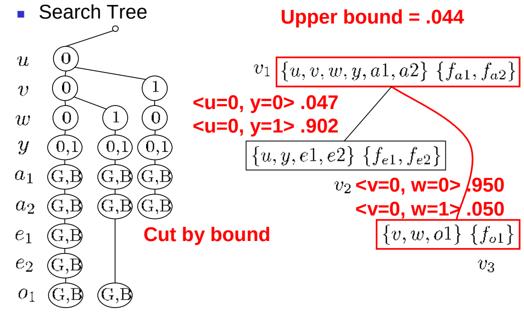
Upper bound = .044



SBBTD applied to Full Adder

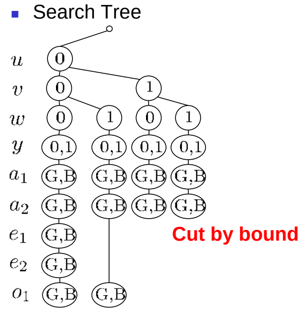
- Search Tree

Upper bound = .044



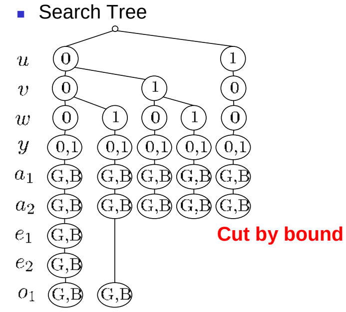
SBBTD applied to Full Adder

- Search Tree



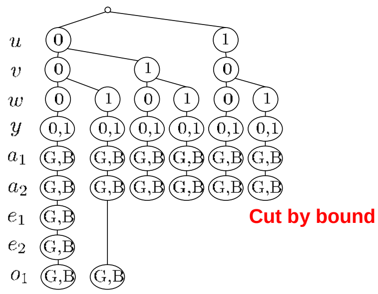
SBBTD applied to Full Adder

- Search Tree



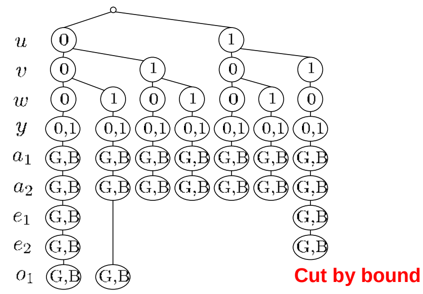
SBBTD applied to Full Adder

- Search Tree



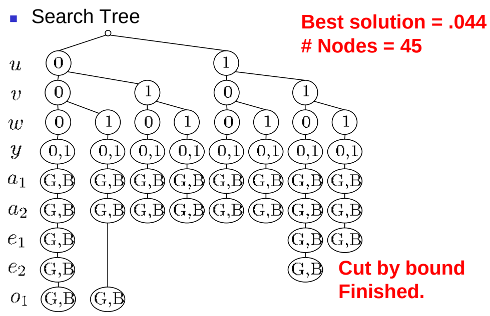
SBBTD applied to Full Adder

- Search Tree



SBBTD applied to Full Adder

- Search Tree



Results

- C++ Implementation of SBBTD on Pentium 4 with 1 GB RAM, using ADD library from CUDD package
- Weighted version of ssa0432-003 circuit (435 variables, 1027 constraints, domain size 2)
- Time to compute tree decomposition: 5 sec.
- Using partition {{0},{1}}: out of time (>10 min)
- Using partition {{0,1}}: 3 sec.

Future Work

- Determine optimal granularity of domain partitions?
- Combination with local filtering techniques?

Material

Material

Main Idea

- Diagnosis as soft constraint solving (DX-2004)
- Efficient techniques for solving soft constraints?
- Branch-and-Bound Search: memory efficient, but time exponential (backtracking)
- Inference: no backtracking, but memory exponential
- Techniques to exploit structure: Decision diagrams, Tree Decompositions
- Still memory exponential.
- Idea: hybrid of branch-and-bound search, symbolic encoding and tree decomposition.

Soft CSPs

- Unified framework for **constraints** and **preferences**.
- 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 for each constraint, using a **binary operator** (with special axioms).
- Assignments are compared using a total **order** on valuations.
- The problem is to produce an **assignment of minimum valuation**.

Formally: Valuation Structure

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

- E = **set of valuations**, used to assess assignments
- $\bot$ = **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 (commutative, associative, monotonic, neutral element $\bot$, annihilator $\top$)

Decision Diagrams

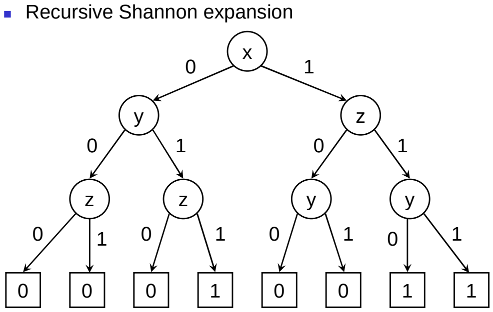
- Example

f :

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

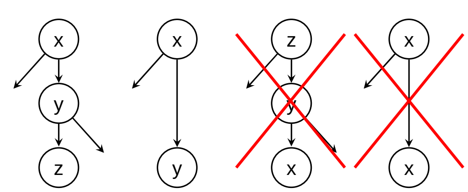
Binary Decision Trees

- Recursive Shannon expansion



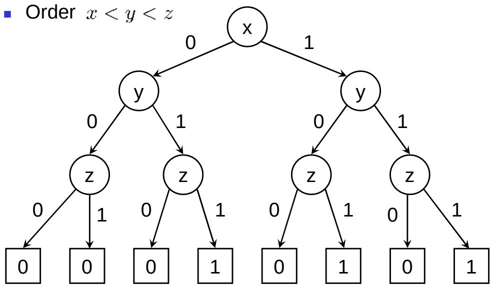
Variable Ordering

- Impose arbitrary total ordering on variables
- Variables must obey ordering along all paths
- Property: No conflicting assignments along path

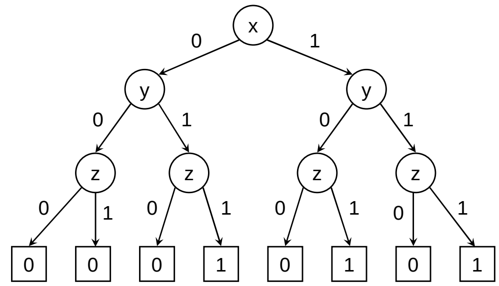


Ordered Binary Decision Tree

- Order $x < y < z$

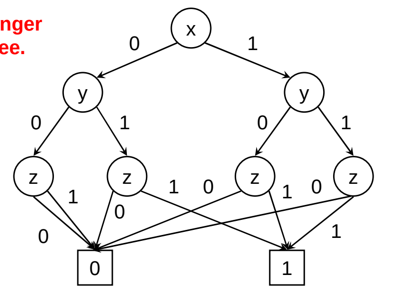


Ordered Binary Decision Tree

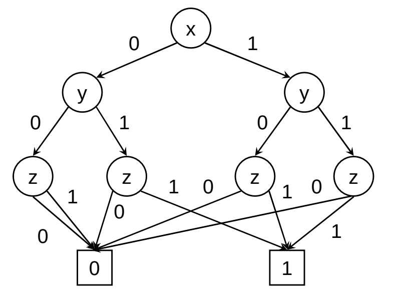


Rule 1: Collapse Leaf Nodes

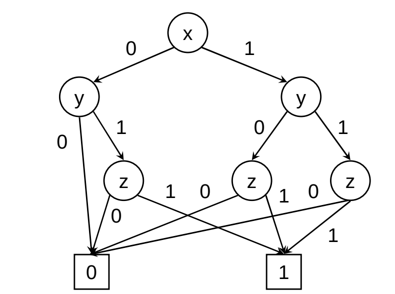
No longer a tree.



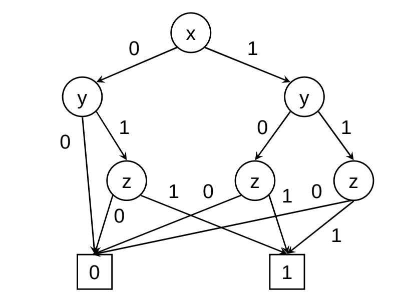
Rule 2: Remove Redundant Tests



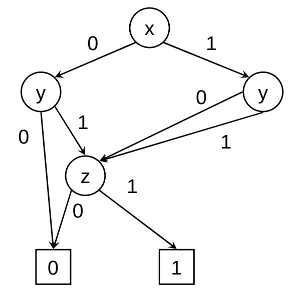
Rule 2: Remove Redundant Tests



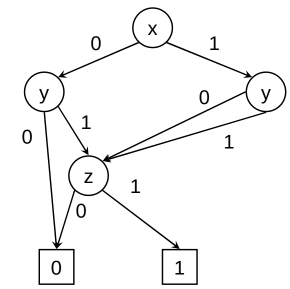
Rule 3: Isomorphic Subgraphs



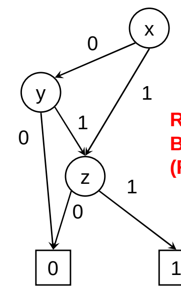
Rule 3: Isomorphic Subgraphs



Rule may become applicable again



Final Representation



Reduced, Ordered Binary Decision Diagram (ROBDD)

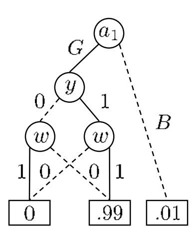
ROBDDs

- Contains only variables from **support**
- Time and space **complexity** of operations depends on **ROBDD size** rather than number of assignments
- Can be extended to functions with **non-binary values** and **non-binary variables**
  - Algebraic Decision Diagrams (ADDs) [Bahar et al. 93]
  - Multi-valued Decision Diagram (MDDs) [Kam 90]
- Significant compaction in many practical cases.

Example

- Full Adder

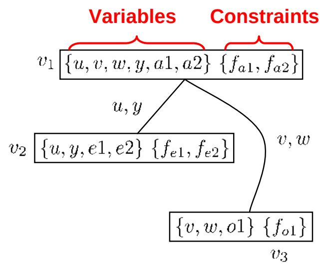
f_{a1} : a_1 y w …	
G 0 0 …	.99
G 1 1 …	.99
U 0 0 …	.01
U 0 1 …	.01
U 1 0 …	.01
U 1 1 …	.01



Tree Decomposition

- Every variable of the original problem must appear in at least one subproblem.
- Every constraint of the original problem must appear in a subproblem, along with all variables in its scope (**covering condition**).
- If a variable occurs in two subproblems, it must appear in every subproblem along the path that connects the two (**connectedness condition**).

Example: Full Adder



Tree Decomposition

- Principle: Solve each subproblem, then combine solutions using dynamic programming.
- Time **O(exp(tw))**, where **tree width tw** is maximum number of variables in a subproblem minus one.
- Space **O(exp(sw))**, where **separator width sw** is maximum number of variables shared between subproblems.
- Finding decompositions with minimal width is **NP-hard**, but good heuristics exist.
- The width is often **small** in practice.

BTD (Terrioux and Jégou CP-03)

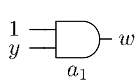
- Assign variables in subproblem, beginning with root of the tree decomposition.
- Inside a subproblem, use classical branch-and-bound, considering only the constraints of the subproblem.
- Once all variables in the subproblem have been assigned, consider its children (if any).
- Given a child, check if the current assignment, restricted to variables shared with the child, has been previously recorded as a good.

BTD (Terrioux and Jégou CP-03)

- If it is not previously recorded, compute solution for the subproblem given the current assignment and upper bound, and record it as new good.
- Add recorded value to value of current assignment.
- If resulting value is below the upper bound, proceed with the next child, else backtrack.

Projection

- $(f \Downarrow_Y)(t) := \min_{t'[Y]=t} f(t')$, $Y \subseteq X$



f_{a1} :

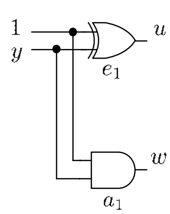
a_1	y	w	...	
G	0	0	...	.99
G	1	1	...	.99
B	0	0	...	.01
B	0	1	...	.01
B	1	0	...	.01
B	1	1	...	.01

$f_{a1} \Downarrow_{\{y,w\}}$:

y	w	...	
0	0	...	.99
1	1	...	.99
0	1	...	.01
1	0	...	.01

Combination

- $(f_1 \otimes f_2)(t) := f_1(t) \oplus f_2(t)$



f_{a1} :

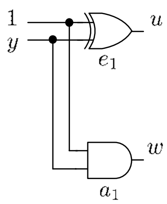
a_1	y	w	...	
G	0	0	...	.99
G	1	1	...	.99
B	0	0	...	.01
B	0	1	...	.01
B	1	0	...	.01
B	1	1	...	.01

f_{e1} :

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

Combination

- $(f_1 \otimes f_2)(t) := f_1(t) \oplus f_2(t)$



$f_{a1} \otimes f_{e1}$:

a_1 e_1 y u w ...	
G G 0 0 0 ...	.9405
G G 1 0 1 ...	.9405
G B 0 0 0 ...	.0495
G B 0 1 0 ...	.0495
G B 1 0 1 ...	.0495
G B 1 1 1 ...	.0495
...	...

Soft CSPs

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

Sinking Operation

- $\mathrm{sink}(c_j, \alpha)$ is a new constraint where all values of tuples $\succeq \alpha$ have been replaced by $\top$
- Generalizes the test $lb < ub$ to functions

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

Set-based Branch-and-Bound

- **Function** $\mathrm{SDFBB}(f_t$: assignments, ub : value): value
  - $f_{lb} \leftarrow lb(f_t)$
  - $f_t \leftarrow \mathrm{sink}(f_{lb}, ub)$
  - **if** $f_t \not\equiv \top$ **then**
    - **if** $\mathrm{var}(f_t) = n$ **then return** $f_t \Downarrow_\emptyset$
    - **let** x_i be an unassigned variable
    - **for each** $p \in P_i$ **do**
      - $ub \leftarrow \min(ub, \mathrm{SDFBB}(f_t \oplus (x_i \in p), ub))$
    - **return** ub
  - **return** $\top$

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

Branch-and-Bound Search

- **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)