Principles of AI Problem Solving
Tutorial IJCAI-05

Adnan Darwiche (UCLA, Los Angeles)
Rina Dechter (UCI, Irvine)
H. Geffner (UPF, Barcelona)

Problems

Problem solving in AI is about representation and automated solution of a wide variety of problems

diagnosis, planning, scheduling, logistics, control
games: sokoban, mastermind, 15-puzzle, n-queens, chess
robot navigation, traveling salesman, map coloring, . . .

Models

Common **structure** of certain classes of problems can be abstracted and expressed in terms of **mathematical model**; e.g.,

- **Constraint Satisfaction Problems (CSP)** are models of the form $\langle X, D, C \rangle$ where X, D, and C are sets of variables, domains, and constraints
- A **solution** to a CSP assigns to each variable a value from its domain such that all constraints satisfied

  Key point:

  – Many problems can be formulated as CSPs
  – If we know how to solve CSPs, we know to solve those problems
  – Same for other models . . .

Example: Linear Equations Model

- John's age is three times Peter's age
- In 10 years, John's age will be twice Peter's age
- How old are John and Peter now?

Formulate problem as **2-linear equations with 2 unknowns**:

$$\begin{aligned} J &= 3P \\ J + 10 &= 2(P + 10) \end{aligned}$$

Solve **model** using **general method**; e.g. variable elimination

$$3P + 10 = 2P + 20$$

Then

$$\begin{aligned} P &= 20 - 10 = 10 \\ J &= 3P = 30 \end{aligned}$$

Problem Solving in AI

- define models of interest
- develop effective methods for solving them

In this tutorial:

-- We'll cover a wide range of models, including State Models, SAT, CSPs, Bayesian Networks, Markov Decision Processes (MDPs), . . .

-- Focus on key principles underlying current solution methods:

- Search Space
- Pruning
- Learning
- Decomposition
- Compilation
- Variable Elimination

Plan for the tutorial

- Introduction (Hector)
  - *Models based on States*
  - *Models based on Variables*
  - *Overview of Techniques*
- Solving models with Search and Inference
  - *State-based Models* (Hector)
  - *Variable-based [Factored or Graphical] Models* (Rina)
- Solving models with Pure Inference and No Search (Adnan)
- Hybrid Methods (Rina)
- Wrap up

More about Tutorial

- assumes basic course in AI
- focuses on principles; not exhaustive (e.g., no approx. methods)
- conceptual but also technical

Please ask questions along the way . . .

Part 1: Introduction

Principles of AI Problem Solving: Introduction

- **Contents**
  - *Models based on States*
  - *Models based on Variables*
  - *Overview of Techniques*
- **Format; Style**
  - general, high-level view of field
  - emphasize intuitions and coherence
  - raise questions that will be addressed in detail later on

Models

- Models define **what** is to be solved
- Algorithms define **how** to solve models

E.g, we understand **what** $\sqrt{43}$ **is** without necessarily knowing **how to compute** value

Same with models: they define the solutions we are looking for, without commitment about their computation

State Models

- Basic State Model characterized by
  - finite and discrete state space S
  - an initial state $s_0 \in S$
  - a set $G \subseteq S$ of goal states
  - actions $A(s) \subseteq A$ applicable in each state $s \in S$
  - a state transition function $f(s, a)$ for $s \in S$ and $a \in A(s)$
  - action costs $c(a, s) > 0$

- A **solution** is a sequence of applicable actions a_i, $i = 0, \ldots, n$, that maps the initial state s_0 into a goal state $s \in S_G$; i.e.,

$$s_{i+1} = f(a_i, s_i) \text{ and } a_i \in A(s_i) \text{ for } i = 0, \ldots, n \text{ and } s_{n+1} \in S_G$$

- **Optimal** solutions minimize total cost $\sum_{i=0}^{i=n} c(a_i, s_i)$

Problems mapping naturally into State Models

- Grid Navigation
- 15-puzzle
- Rubik
- Route Finding in Map
- TSP (Traveling Salesman Problem)
- Jug Puzzles (e.g., 4 & 3 liter jars, have 2 liters in 4 lit. jar)
- $\vdots$

This is the model underlying **Classical Planning** . . .

Languages

State-Models often represented implicitly in terms of (planning) **languages**

E.g.: **Strips** is a simple language for representing the Basic State Models

- A **problem** in Strips is a tuple $\langle A, O, I, G \rangle$:
  - A stands for set of all **atoms** (boolean vars)
  - O stands for set of all **operators** (actions)
  - $I \subseteq A$ stands for **initial situation**
  - $G \subseteq A$ stands for **goal situation**

- Operators $o \in O$ **represented** by three lists

-- the **Add** list $Add(o) \subseteq A$
-- the **Delete** list $Del(o) \subseteq A$
-- the **Precondition** list $Pre(o) \subseteq A$

Strips: From Language to Model

Strips problem $P = \langle A, O, I, G \rangle$ determines **state model** $\mathcal{S}(P)$ where

- the states $s \in S$ are **collections of atoms**
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are s.t. $Prec(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s)$ are all 1

The (optimal) **solution** of problem P is the (optimal) **solution** of State Model $\mathcal{S}(P)$

Later on we'll see how **Strips descriptions** can play a **computational role** as well . . .

Model with Incomplete Information and Non-Determinsm

- finite and discrete state space S
- a **set** of possible initial states $S_0 \subseteq S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a **non-deterministic** transition function F s.t. $F(a,s)$ is a **set** of states, $a \in A(s)$
- action costs $c(a,s) > 0$

-- A **solution** is a sequence of actions that lead to S_G for **any** possible initial state and transition

-- An **optimal** solution miminizes the sum of action costs

-- Planning over this class of models called **Conformant Planning**

Model with Non-determinism and Full Feedback

- finite and discrete state space S
- a **set** of possible initial states $S_0 \subseteq S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a **non-deterministic** transition function F . . .
- action costs $c(a,s) > 0$
- states **fully observable**

-- **Solutions** become **functions** mapping states into actions (**closed-loop control policies**)

-- This is because dynamics is **Markovian** and past history of system is not relevant

-- **Optimal** solutions minimize cost in **worst case** (**min-max state policies**)

Stochastic Model with Full Feedback (Markov Decision Process - MDP)

- finite and discrete state space S
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- **transition probabilities** $P_a(s'|s)$ for s and $a \in A(s)$
- action costs $c(a, s) > 0$
- states **fully observable**

-- **Solutions** like before are **functions** mapping states into actions (**closed-loop control policies**)

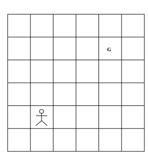
-- **Optimal** solutions minimize **expected** cost

-- This model underlies **Probabilistic Planning**; although variations possible (e..g, partial observability, achieving goal with certain probability, discounted formulations, . . .).

Example: Navigation Problems

Consider robot that has to reach target G when

- *initial state is known and actions are deterministic*
- *initial state is unknown and actions are deterministic*
- *states are fully observable and actions are stochastic*
- *states are partially observable and actions are stochastic . . .*



-- How do these problems map into the models considered?

-- What is the form of the solutions?

Models based on Variables: Factored or Graphical Models

- Constraint Satisfaction Problems (CSP)
- Satisfiability (SAT)
- Bayesian Networks
- Influence Diagrams, . . .

Several **tasks** associated with these models . . .

Constrain Satisfaction Problems (CSPs)

- CSPs are triplets $\langle \text{Variables,Domains,Constraints} \rangle$
- A **solution** assigns values to the variables from their corresponding domains satisfying all constraints
- A CSP is **consistent** if it has one or more solutions

E.g., first CSP below is consistent; second is not

$$\langle \{X, Y\}, \{D_X, D_Y = [1..10]\}, \{X + Y > 10, X - Y > 7\} \rangle$$

$$\langle \{A, B, C\}, \{D_A, D_B, D_C = [a, b]\}, \{A \neq B, B \neq C, A \neq C\} \rangle$$

Problems that Map into CSPs

- Scheduling
- Planning
- Resource allocation
- Map coloring
- N-queens
- $\vdots$

Satisfiability (SAT)

- SAT is special type of CSP where variables x, y, ... are **boolean** and constraints are **clauses**
$$x \lor \neg y \lor z \ldots$$
- A set of clauses denotes a formula in **Conjunctive Normal Form (CNF):** a conjunction of disjunctions of **literals**
- Current SAT solvers are very powerful, and used in Planning and Verification
- Any CSP can be mapped into SAT and vice versa, and solving techniques have a lot in common

The Graph Underlying Graphical Models

- SAT and CSP are both NP-Complete, yet complexity bounded by **treewidth of interaction graph**

- The **interaction graph** of a problem is an undirected graph where
  - the vertices are the variables, and
  - two variables are connected iff if they appear in same constraint/clause

- The **treewidth** measures how 'tree-like' is the interaction graph
  - treewidth $= 1$ implies **linear complexity**, while
  - **bounded treewidth** implies **polynomial complexity**

Bayesian Networks (BNs)

BNs are graphical models that express a **joint probability distribution** over a set of variables X_1, . . . , X_n by means of

- a **directed acyclic graph** over the variables

- **conditional probability tables** $P(X_i|pa(X_i))$ of each variable X_i given its parents $pa(X_i)$ in the graph

The joint distribution is the product of the tables:

$$P(X_1, \dots, X_n) = \prod_{i=1,n} P(X_i|pa(X_i))$$

BNs and CSPs are similar; they specify joint **probability** and joint **consistency** through local factors that define the interaction graph

Models and Tasks: SAT and CSPs

- **Consistency** is basic task in SAT and CSPs: *find a satisfying assignment o that no one exists*

Yet other tasks common:

- **Optimization:** find **best** satisfying assigment according to some **function** (COP)
- **Enumeration:** find **number** of satisfying assignments (**Model Counting**); find **all solutions**, . . .

All tasks are NP-hard; **Consistency** and **(Bounded) Optimization** are NP-Complete while **Enumeration** is #P.

Models and Tasks: Bayesian Networks

- **Enumeration:** find probability given evidence: $P(X = x|Y = y, Z = z, ...)$ (Bel)
- **Optimization:** find most probable instantiation given evidence (MPE)
- Other: find most probable instantiation of **subset** of variables given evidence (MAP)

All tasks are NP-hard; (Bounded) MPE is NP-Complete, and Bel is #P

Map

- Introduction
  - *Models based on States*
  - *Models based on Variables*
  - ▷ *Overview of Techniques*
    - ∗ Search Space
    - ∗ Pruning
    - ∗ Learning
    - ∗ Decomposition
    - ∗ Compilation
    - ∗ Variable Elimination
- Solving models with Search and Inference
  - *State-based Models*
  - *Variable-based [Factored or Graphical] Models*
- Solving models with Pure Inference and No Search
- Hybrid Methods

Search

- Basic tasks can be formulated as **search problem** in suitable **problem space**
- Problem or Search space is a Directed Graph given by
  - root node n_0 of the search
  - set of terminal nodes; either dead-ends or goals
  - branching rule generating children n' of non-terminal nodes n
  - costs $c(n, n') \geq 0$
- A solution is a directed path that connects the root node with a goal node. It is optimal if it minimizes the sum of the edge costs.

Direct Problem Space for Basic State Models

The **nodes** correspond to the **states** and

- *root node is initial state* s_0
- *goal nodes are the goal states*
- *dead ends are states* s *s.t no action applies in* s
- *branching rule:* $s \rightarrow s'$ *if* $s' = f(a, s)$ *for some* a *applicable in* s
- *cost is then* $c(s, s') = c(a, s)$

In spite of direct mapping from Basic State Model to Search Graph, it's good to keep in mind that first is a description of the **problem**, while second is the structure explored for finding a **solution**

When the State Model is described in **Strips**, this is the so-called **progression space**, as alternative spaces are possible . . .

Alternative Problem Spaces from Strips Encodings

- **regression space:** branch by applying actions backward from goal til finding conditions that hold in initial state
- **plan space:** branch by refining partial plan, removing its flaws

In certain cases, these alternative branching schemas/problem spaces more suitable (e.g., plan space seems best for optimal temporal planning)

Strips problems with fixed planning horizon can also be mapped into SAT, which works very well when **optimal parallel plans** are sought

Problem Space for Non-Deterministic State models

Conformant Planning can be formulated as Search Problem over **belief space**, where nodes are **belief states**, i.e., sets of states deemed possible

- *root node is set of possible initial states*
- *goal nodes are sets of goal states*
- *edge $n \rightarrow n'$ if for some action, n' is the set of states that may follow the states in n*
- . . .

-- This is most common formulation for Conformant Planning currently

-- Belief states represented often by **propositional formula** in suitable **'compiled'** form (e.g., OBDDs, d-DNNF, . . . ; more about this later on)

Problem Space for Graphical Models: OR Space

Consistency and **Optimization Problems** for SAT, CSP, and Bayesian Networks are formulated as **search problems** over suitable Search Graph.

In the standard formulations, the nodes are **partial assignments**:

- *root node is empty assignment*
- *dead-ends are partial assignments that violate a constraint*
- *'goal' nodes are complete assignments*
- *children n' of node n obtained* **by picking up unassigned variable in n, and assigning it a value**
- *costs $c(n, n')$ uniform in consistency problems, and dependent on local functions in COP and BNets.*

-- Choice of **branching variable** affects size of Search Tree and critical for performance

Problem Space for Graphical Models: AND/OR Space

- Solution to **enumeration problems**, like **model counting** over CNFs and **belief updating** over BNets, do not correspond to solution **paths** in graph but can be computed from it

- We will also see an **alternative formulation** of all these tasks in terms of **AND/OR Graphs** rather than (OR) Graphs that exploit **decomposition**

- This AND/OR space is (almost) explicit in some algorithms (e.g., Recursive Conditioning) and implicit in others (e.g., non-chronological backtracking).

Map

- Introduction
  - *Models based on States*
  - *Models based on Variables*
  - *Overview of Techniques*
    - ∗ Search Space
    - ▷ Pruning
    - ∗ Learning
    - ∗ Decomposition
    - ∗ Compilation
    - ∗ Variable Elimination

- Solving models with Search and Inference
  - *State-based Models*
  - *Variable-based [Factored or Graphical] Models*

- Solving models with Pure Inference and No Search

- Hybrid Methods

Pruning in Depth-First Search

- Search graph can be solved by Depth-First Search (DFS) and variations
- More effective DFS obtained by **pruning** nodes that cannot lead to acceptable solutions; e.g.,

  Consistency: *prune node n if it can only lead to dead-ends*

  State Models/Optimization: *prune node n if it can only lead to solutions with cost $>$ than given $Bound$*

- By playing with $Bound$ one can get Bounded DFS, IDA*, DFS Branch & Bound
- **Key issue:** how to **predict** when paths up to node n
  - can only lead to dead-ends? [consistency]
  - can only lead to solutions with cost $> Bound$? [optimization]

Pruning (2)

- Pruning criterion has to be **sound** and **cost-effective**
- Two ideas: **lower bounds** (LBs) and **constraint propagation** (CP)
  - **LBs:** *prune n if $f(n) > Bound$ where $f(n)$ is LB of cost of best solution that extends n*
  - **CP:** *prune value x from variable X domain, if $X = x$* **proved inconsistent** *with constraints and commitments in n; prune node n itself if some domain becomes empty*
- LBs and CP mechanisms can both be obtained as **inference in relaxed model**; e.g.
  - in Strips: forget 'deletes' and assume (relaxed) actions can be done in parallel ('simple reachability heuristic')
  - in CSPs: e.g., solve each constraint in isolation ('arc consistency')

We will say more about LBs and CP mechanisms . . .

Learning during Search (State Models)

- Results of (partial) search can be used to improve pruning in rest of the search
- E.g., if node shown not to lead to solution found again in the search, it can be pruned right away
- However one can do better; e.g., in IDA*, for example, right after all sons n' of node n return without a solution (for given $Bound$), **heuristic value** $h(n)$ can be **increased** to:

$$h(n) := min_{n':n \rightarrow n'} \; c(n,n') + h(n')$$

- Resulting algorithm known as IDA* + **Transposition Tables**
- Exactly same update rule used in **Learning Real Time A* (LRTA*)**
- Actually **updates** can be done without search at all and they eventually yield h^*! This is what **Value Iteration** does, which also applies to MDPs

Learning during Search (Factored Models)

- For consistency tasks (SAT, CSP), one can actually do even better
- Rather than **updating** the **value** of a node, update the **theory** itself!
- Use structure for identifying and ruling out **cause of the inconsistency**
- This is the idea of **no-good learning** in SAT and CSPs
- Learned information
  - applies to other nodes as well
  - results in non-chronological backtracking
  - enables further inferences and pruning
- It is a key idea in current SAT solvers

Decomposition

- Consider solving a SAT problem T made up of two independent sub-problems T' and T'' with n variables each, none in common
- By **decomposing** the problem in two as

$$SAT(T) = SAT(T') \ \& \ SAT(T'')$$

worst case complexity is reduced from $2^n * 2^n$ to $2^n + 2^n$

- Interestingly, if T' and T'' overlap over single variable X, T can still be decomposed by **conditioning** on X as

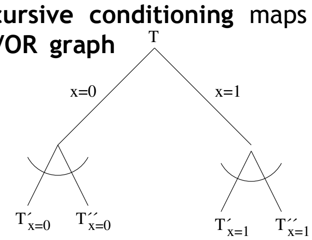
$$SAT(T) = \bigvee_x SAT(T'_{X=x}) \ \& \ SAT(T''_{X=x})$$

where $T_{X=x}$ means T with variable X replaced by value x.

- This idea can be applied recursively, even if T' and T'' overlap over **set** of variables

Decomposition and AND/OR Search Graph

- Decomposition by **recursive conditioning** maps search over **OR-graph** into search over **AND/OR graph**

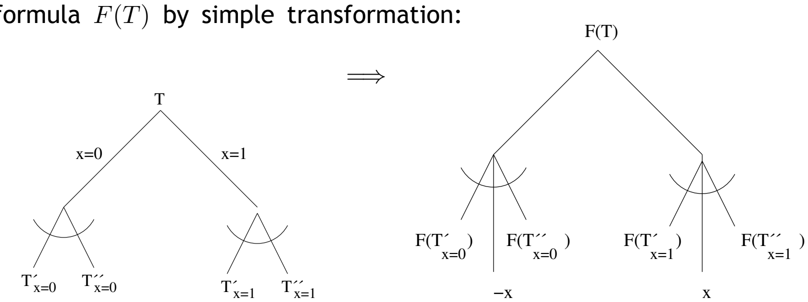


- By suitable choice of decompositions and caching, worst-case complexity can be reduced from $O(Exp(n))$ to $O(Exp(w^*))$, where $w^* \leq n$ is theory **treewidth** (e.g., linear for trees)
- Similar decomposition methods can be used (and are used!) for enumeration tasks like **Model Counting** (MC) and **Belief Update** but with different agregation operators; e.g.,

$$MC(T) = \sum_x MC(T'_{X=x}) \ * \ MC(T''_{X=x})$$

From Decomposition to Knowledge Compilation (1)

- Suitable 'trace' of AND/OR Search can be used to perform a large class of **intractable boolean operations** in time linear in size of 'trace'
- Indeed, just map AND/OR Graph for P into **logically equivalent** AND/OR formula $F(T)$ by simple transformation:



- AND/OR formula $F(T)$ is in Deterministic Decomposable Negation Normal Form (d-DNNF): **disjuncts** are exclusive, **conjuncts** share no variables, and **negations** affect vars only; closely related to **OBDDs**

Map

- Introduction
  - *Models based on States*
  - *Models based on Variables*
  - *Overview of Techniques*
    - ∗ Search Space
    - ∗ Pruning
    - ∗ Learning
    - ∗ Decomposition
    - ∗ Compilation
    - ▷ Variable Elimination
- Solving models with Search and Inference
  - *State-based Models*
  - *Variable-based [Factored or Graphical] Models*
- Solving models with Pure Inference and No Search
- Hybrid Methods

Variable Elimination

- Gaussian Elimination used to solve n linear equations T_n with n unknowns $X_1, \ldots, X_n$
  - *Eliminate X_n obtaining $n-1$ equations T_{n-1} with $n-1$ unknowns*
  - *Iterate til obtaining 1 equation T_1 with 1 unknown (X_1)*
  - *Solve T_1 for X_1 and plug result into T_2*
  - *Solve T_2 for X_2 and plug result x_2 for X_2 into T_3, etc*

- Method can be be generalized to **graphical models**; only change is way for **eliminating variables**; e.g.,
  - *in CNF, X_i eliminated by* **resolving upon** X_i
  - *in CSPs, X_i eliminated by* **join** *and* **project** *DB operations*
  - *in Belief Update (BNets), X_i eliminated by* **sum** *and* **products**, ...

Variable Elimination, Inference, and AND/OR Search

- Variable Elimination solves problems by **inference and no search**

- Yet same complexity bounds ($O(Exp(w^*))$) as Decomposition Methods with Caching that search over AND/OR graphs, and furthermore ...

- Variable Elimination can be understood as **bottom up search of same AND/OR graph!**

Few powerful ideas that span a large terrain and have a lot of connections and ramifications. This is what the tutorial is about ...

Map

- Introduction (Hector)
  - *Models based on States*
  - *Models based on Variables*
  - *Overview of Techniques*
    - ∗ Search Space
    - ∗ Pruning
    - ∗ Learning
    - ∗ Decomposition
    - ∗ Compilation
    - ∗ Variable Elimination
- Solving models with Search and Inference
  - ▷ *State-based Models* (Hector)
  - – *Variable-based [Factored or Graphical] Models* (Rina)
- Solving models with Pure Inference and No Search (Adnan)
- Hybrid Methods (Rina)

Part 2: Search and Inference

Techniques for Solving State Models: Focus

- **Models**
  - *Basic State Models: Complete Knowledge, Deterministic Actions*
  - *Markov Decision Processes: Stochastic Actions and Full Feedback*
- **Techniques**
  - *Problem Space: Branching Schemes*
  - *Pruning: Admissible heuristics or LBs* $h(s) \leq h^*(s)$
  - *Learning: Improving* $h(s)$ *while Searching*
- **Language**
  - *We assume* **Models** *specified in a* **Strips**-*like language*
  - *This takes us into what is called* **Planning** *in AI*

State Models Reminder

- Basic State Model characterized by
  - finite and discrete state space S
  - an initial state $s_0 \in S$
  - a set $G \subseteq S$ of goal states
  - actions $A(s) \subseteq A$ applicable in each state $s \in S$
  - a state transition function $f(s, a)$ for $s \in S$ and $a \in A(s)$
  - action costs $c(a, s) > 0$
- A **solution** is a sequence of applicable actions that map initial state s_0 into goal state
- **Optimal** solutions minimize total cost . . .

Strips Reminder

- A **problem** in Strips is a tuple $\langle A, O, I, G \rangle$:
  - A stands for set of all **atoms** (boolean vars)
  - O stands for set of all **operators** (actions)
  - $I \subseteq A$ stands for **initial situation**
  - $G \subseteq A$ stands for **goal situation**

- Operators $o \in O$ **represented** by three lists

  -- the **Add** list $Add(o) \subseteq A$
  -- the **Delete** list $Del(o) \subseteq A$
  -- the **Precondition** list $Pre(o) \subseteq A$

Strips: From Language to Model

Strips problem $P = \langle A, O, I, G \rangle$ determines **state model** $\mathcal{S}(P)$ where

- the states $s \in S$ are **collections of atoms**
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are s.t. $Prec(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s)$ are all 1

The (optimal) **solution** of problem P is the (optimal) **solution** of State Model $\mathcal{S}(P)$

Pruning: Getting Lower Bounds for Strips Problems

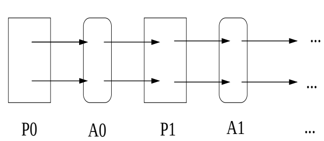
Admissible Heuristics (LBs) for Strips obtained by solving **relaxed models**

- ignore delete-lists
- ignore certain atoms
- decompose goals sets into smaller subsets
  - e.g., assume cost of achieving set given by cost of achieving most costly **pair** in the set . . .

Lower Bounds for Strips: Reachability Graph

Ignore deletes and apply all actions in parallel:

$$\begin{aligned} P_0 &= \{p \in s\} \\ A_i &= \{a \in O \mid Prec(a) \subseteq P_i\} \\ P_{i+1} &= \{p \in Add(a) \mid a \in A_i\} \end{aligned}$$



Define then **admissible heuristic**

$$h^1_G(s) \stackrel{\text{def}}{=} \text{min } i \text{ such that } G \subseteq P_i$$

Need No-op(p*) action for each* p*:* $Prec = Add = \{p\}$

Planning Graph = Reachability Graph + Mutexes

- **Better relaxation:** assume **no deletes** and that all actions can be done in parallel **except** certain **incompatible action pairs**

- This relaxation not tractable but good LB approximation exists, in particular for **parallel planning** (where only diff is that deletes are not ignored)
  - **action** pair **mutex** at i if incompatible or preconditions **mutex** at i
  - **atom** pair **mutex** at $i+1$ if all supporting action pairs **mutex** at i

- Mutex x, y at i implies that no valid plan can have both x and y at i but not the converse

- Define then more informed **admissible heuristic** $h^2_G(s)$ as:

$$h^2_G(s) \stackrel{\text{def}}{=} \text{min } i \text{ such that } G \subseteq P_i \text{ \& not mutex at } i$$

- Graph and resulting heuristic $h^2_G(s)$ computed for one state s only, but valid for **any** goal G . . .

How to use Strips Heuristics? Problem Spaces in Planning

- **Option 1: Progression Space:** search forward from s_0
  - *recompute graph and $h^2_G(s)$ for every s; this is costly*
- **Option 2: Regression Space:** search backward from Goal (Graphplan)
  - *no need to recompute graph which encodes $h^2_{G'}(s)$ for all goals G'! but high branching factor when lots of parallel actions*
- **Option 3: Action Space**: non-directional search
  - *Branch by picking an action a and time point i and trying the two possibilities: a in the plan at i ; a not in the plan at i*
  - *At each node n* **recompute planning graph** *from s_0* **respecting commitments in** *n, and* **prune** *n if Goals pushed beyond horizon*

Current State of the Art in Planning

No single best **Problem Space** or **Pruning Criterion** for all types of planning tasks:

- **Sequential (Classical) Planning: Heuristic Search** in Progression Space currently best with both admissible and non-admissible h's
- **Optimal Parallel Planning: SAT formulation** fed to state-of-the-art solvers (Siege) currenty best
- **Optimal Temporal Planning:** Search in **Plan Space (POCL)** best with pruning scheme based on **Constraint Propagation**

Learning while Searching in State Models

A number of algorithms combine **search** with **state value updates**:

- Learning Real time A* (LRTA*)
- IDA* + Memory (Tranposition Tables)
- Real Time Dynamic Programming (RTDP)
- MTD (algorithm for Game Trees better than Alpha-Beta)
- . . .

Other algorithms do **updates with no search**

- Value Iteration

Understanding Value Updates: Dynamic Programming

- Solutions to wide range of models can be expressed in terms of solution of so-called **Bellman equation:**

$$V(s) = min_{a \in A(s)}\ Q_V(a, s)$$

where cost-to-go term $Q_V(a, s)$ depends on model ($F(a, s)$: next states)

$c(a, s) + V(s')$, $s' \in F(a, s)$	for OR Graphs
$c(a, s) + \max_{s' \in F(a,s)} V(s')$	for Max AND/OR Graphs
$c(a, s) + \sum_{s' \in F(a,s)} V(s')$	for Additive AND/OR Graphs
$c(a, s) + \sum_{s' \in F(a,s)} P_a(s'\|s)V(s')$	for MDPs
$\max_{s' \in F(a,s)} V(s')$	for Game Trees

- The **greedy policy** π_V is **optimal** when $V = V^*$ solves Bellman

$$\pi_V(s) = \text{argmin}_{a \in A(s)}\ Q_V(a, s)$$

- **Question:** how to get V^*?

Updates with No Search: Value Iteration

- Value Iteration finds V^* by successive approximations
- Starting with an arbitrary V, uses Bellman equation to update V; e.g. for Basic State Models (OR Graphs)

$$V(s) := min_{a \in A(s)} \ [c(a,s) + V(s_a)]$$

- As long as **all states updated** sufficiently often (and certain general conditions hold), left and right hand sides converge, and $V = V^*$
- VI is simple and general but also **exhaustive**
- Can the updates be restricted to **subset of states** preserving optimality?
- Yes: like in Heuristic Search, use **Lower Bounds** and **Initial State**

Focusing Value Iteration using LBs and Initial State

- Say that a state s is
  - **inconsistent** if $V(s) < \min_{a \in A(s)} Q_V(a,s)$, and
  - **greedy** if reachable from s_0 using greedy policy π_V
- Then starting with an **admissible**, follow loop:
  - ***Find** an **inconsistent greedy** state s and **Update** it*
- Loops delivers greedy policy that is **optimal even if some states not updated or visited at all!**
- Unlike DP method, both LBs ($V \leq V^*$) and Initial State (s_0) used

Learning During Search in State Models

- Convergence of all Learning Algorithms in State Models (LRTA*, MTD, IDA*+TT, . . .) can be understood in these terms: *update an inconsistent greedy state in all iterations til no more such states*
- Speed up obtained by updating **multiple** such states in every iteration
- This can be done by implementing **Find** as a **DFS** over greedy states with inconsistent states as terminals (*Learning in Depth-First Search*)
- This is what **IDA*** + **Trans. Tables** (Basic State Models) and **MTD** (Game Trees) actually do
- Same idea underlies current **heuristic-search methods for solving MDPs:** LAO*, RTDP, HDP, . . .

Bibliography: Techniques for Solving State Models Part

- Heuristics for Strips Planning: [23, 6, 16, 14, 30]; admissible heuristics [13, 9, 12]; heuristics and the planning graph [6, 13, 24]; Graphplan [4].
- Planners searching in Regression Space [4, 5, 13].
- Planners searching in 'Plan Space': [22, 18, 32].
- Planners searching in (non-directional) 'Action Space': SAT-formulation and CSP formulations like [19, 28] (and in particular [10]), and [15].
- Temporal Planning: [21, 25, 17], Optimal Temporal Planning [31], Planning and Scheduling [29].
- Learning while Searching in State Models: IDA* with Transposition Tables [27], LRTA* [20], RTDP [1], MTD for Game Trees [26]; general LDFS framework [8].
- Value Iteration and Dynamic Programming [2, 3].
- Focusing Value Iteration: General Find-and-Revise procedure [7, 8]
- Learning in DFS [8], Heuristic Search Algorithms for MDPs [1, 11, 7]

References

[1] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81--138, 1995.

[2] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[3] D. Bertsekas. *Dynamic Programming and Optimal Control*, *Vols 1 and 2*. Athena Scientific, 1995.

[4] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, pages 1636--1642. Morgan Kaufmann, 1995.

[5] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of ECP-99*, pages 359--371. Springer, 1999.

[6] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1--2):5--33, 2001.

[7] B. Bonet and H. Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. IJCAI-03*, pages 1233--1238, 2003.

[8] B. Bonet and H. Geffner. Learning in DFS: A unified approach to heuristic search in deterministic, non-deterministic, probabilistic, and game tree settings. 2005.

[9] S. Edelkamp. Planning with pattern databases. In *Proc. ECP 2001*, 2001.

[10] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proc. AAAI-98*, pages 948--953, 1998.

[11] E. Hansen and S. Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35--62, 2001.

[12] P. Haslum, B. Bonet, and H. Geffner. New admissible heuristics for optimal planning. In *Proc. AAAI-05*, 2005. To appear.

[13] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proc. of the Fifth International Conference on AI Planning Systems (AIPS-2000)*, pages 70--82, 2000.

[14] M. Helmert. A planning heuristic based on causal graph analysis. In *Proc. ICAPS-04*, pages 161--170, 2004.

[15] J. Hoffmann and H. Geffner. Branching matters: Alternative branching in graphplan. In E. Giunchiglia, N. Muscettolla, and D. Nau, editors, *Proc. 13th Int. Conf. on Automated Planning and Scheduling (ICAPS-2003)*, pages 22--31. AAAI Press, 2003.

[16] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253--302, 2001.

[17] A. Jonsson, P. Morris, N. Muscettola, and K. Rajan. Planning in interplanetary space: Theory and practice. In *Proc. AIPS-2000*, pages 177--186, 2000.

[18] S. Kambhampati, C. Knoblock, and Q. Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76(1-2):167--238, 1995.

[19] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, pages 1194--1201. AAAI Press / MIT Press, 1996.

[20] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189--211, 1990.

[21] P. Laborie and M. Ghallab. Planning with sharable resources constraints. In C. Mellish, editor, *Proc. IJCAI-95*, pages 1643--1649. Morgan Kaufmann, 1995.

[22] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of AAAI-91*, pages 634--639, Anaheim, CA, 1991. AAAI Press.

[23] D. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2):111--159, 1999.

[24] X. Nguyen, S. Kambhampati, and R. Sanchez Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135(1-2):73--123, 2002.

[25] J. Penberthy and D. Weld. Temporal planning with continous change. In *Proc. AAAI-94*, pages 1010--1015, 1994.

[26] A. Plaat, J. Schaeffer, W. Pijls, and A.de Bruin. Best-first fixed-depth minimax algorithms. *Artificial Intelligence*, 87(1-2):255--293, 1996.

[27] A. Reinefeld and T. Marsland. Enhanced iterative-deepening search. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(7):701--710, 1994.

[28] J. Rintanen. A planning algorithm not based on directional search. In *Proceedings KR'98*, pages 617--624. Morgan Kaufmann, 1998.

[29] D. Smith, J. Frank, and A. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.

[30] V. Vidal. A lookahead strategy for heuristic search planning. In *Proc. ICAPS-04*, pages 150--159, 2004.

[31] V. Vidal and H. Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming. In D. McGuiness and G. Ferguson, editors, *Proceedings of 19th Nat. Conf. on Artificial Intelligence (AAAI-04)*, pages 570--577. AAAI Press/MIT Press, 2004.

[32] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27--61, 1994.

Solving problems with search and inference

Road Map:
Search in variable-based models

- Variable-based (Graphical) models
- Basic search
- Constraint propagation as bounded inference
- Improving search by bounded inference in branching ahead.
- Improving search by looking-back
- The alternative AND/OR search space

Road Map:
Search in variable-based models

- **Variable-based (Graphical) models**
  - **Constraints and cost networks**
  - **Probabilistic networks**
- Basic search and basic Inference
- Constraint propagation: bounded inference
- Improving search by branching ahead
- Improving search by looking-back
- The alternative AND/OR search space

Constraint Satisfaction

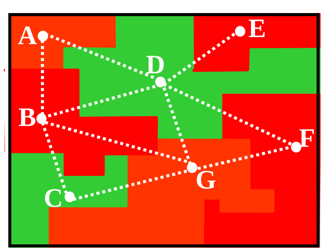
Example: map coloring

Variables - countries (A,B,C,etc.)

Values - colors (e.g., red, green, yellow)

Constraints: **$A \neq B$, $A \neq D$, $D \neq E$,** *etc.*

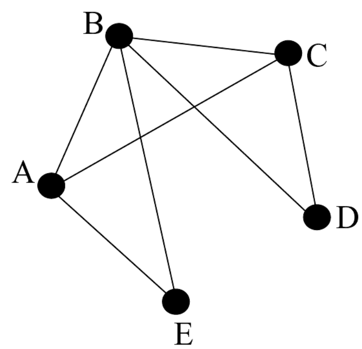
A	B
red	green
red	yellow
green	red
green	yellow
yellow	green
yellow	red



Task: consistency? Find a solution, all solutions, counting

Propositional Satisfiability

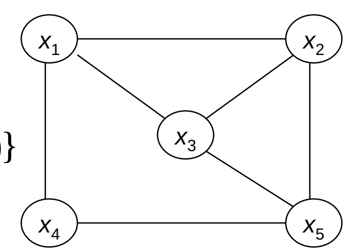
$\varphi = \{(\neg C), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$.



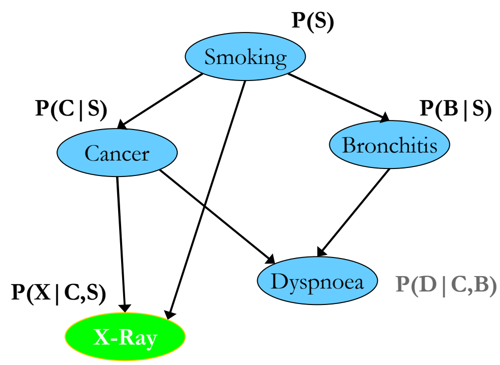
Constraint Optimization

- Variables ⇒ Nodes
- Constrained Variables ⇒ Edges
- e.g.:
  - $f_1(x_1,x_2,x_3)$
  - $f_2(x_2,x_3,x_5)$
  - $f_3(x_1,x_4)$
  - $f_4(x_4,x_5)$

$$\min_{t \in Sol}\{\sum_{i=1}^{m'} f_i(t)\}$$



Probabilistic Networks



P(D|C,B)

C	B	D=0	D=1
0	0	0.1	0.9
0	1	0.7	0.3
1	0	0.8	0.2
1	1	0.9	0.1

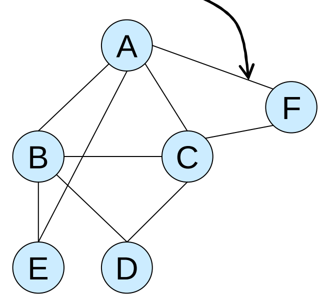
$P(S,C,B,X,D) = P(S) \cdot P(C|S) \cdot P(B|S) \cdot P(X|C,S) \cdot P(D|C,B)$

Graphical models

- A graphical model (X,D,C):
  - $X = \{X_1,\ldots X_n\}$ variables
  - $D = \{D_1, \ldots D_n\}$ domains
  - $C = \{F_1,\ldots,F_t\}$ functions (constraints, CPTS, cnfs)
  - Primal graph

$F_i := P(F \mid A, C)$

$F_i := F = A + C$

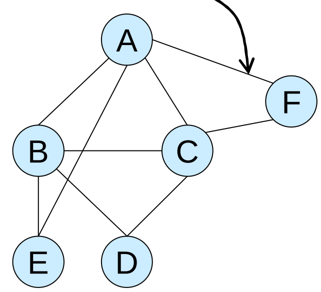


Graphical models

- A graphical model (X,D,C):
  - X = {X_1,…X_n} variables
  - D = {D_1, … D_n} domains
  - C = {F_1,…,F_t} functions (constraints, CPTS, cnfs)
  - Primal graph

$F_i := P(F \mid A, C)$

$F_i := F = A + C$



A reasoning problem defined by operators *combine* and *project*:

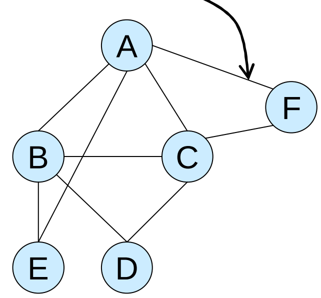
- ❑**MPE:** ***max$_X$ Π_j P_j***
- ❑**CSP:** $\Pi_X \times_j C_j$
- ❑**Max-CSP:** $\min_X \Sigma_j F_j$
- ❑**Belief updating:** $\Sigma_{X-y} \Pi_j P_i$
- ❑**Optimization:** $\min_X \Sigma j Fj$
- ❑**MEU:**

Graphical models

- A graphical model (X,D,C):
  - X = {X_1,…X_n} variables
  - D = {D_1, … D_n} domains
  - C = {F_1,…,F_t} functions (constraints, CPTS, cnfs)
  - Primal graph

$F_i := P(F \mid A, C)$

$F_i := F = A + C$



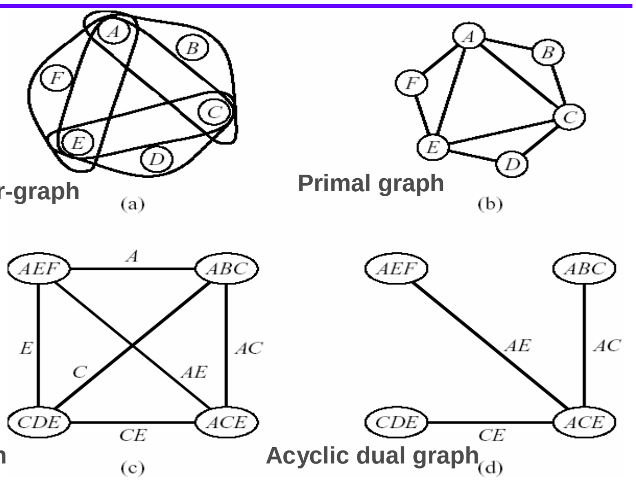
A reasoning problem defined by operators *combine* and *project*:

- ❑**MPE:** ***max$_X$ Π_j P_j***
- ❑**CSP:** $\Pi_X \times_j C_j$
- ❑**Max-CSP:** $\min_X \Sigma_j F_j$
- ❑**Belief updating:** $\Sigma_{X-y} \Pi_j P_i$
- ❑**Optimization:** $\min_X \Sigma j Fj$
- ❑**MEU:**

All these tasks are NP-hard
→ identify special cases
→ approximate

The Graphs of Graphical Models

R= {**f(AEF),f(ABC), f(ACE),f(CDE)}**

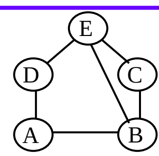


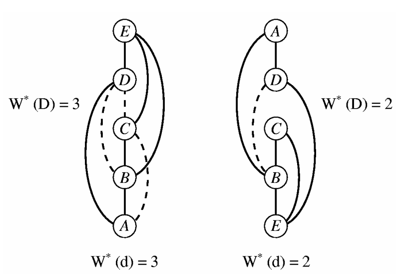
Hyper-graph **Primal graph**

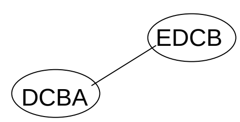
Dual graph **Acyclic dual graph**

Induced-width and Tree-width

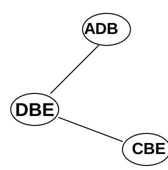
Induced-width
Of ordering





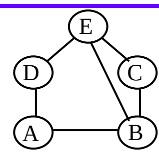


Tree-width
=3

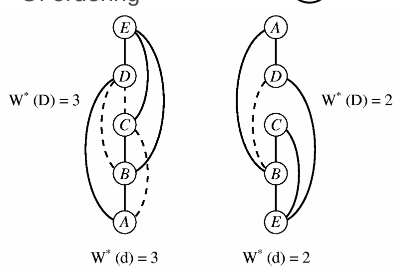


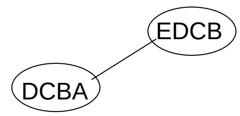
Tree-width
=2

Induced-width and Tree-width

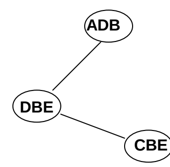


Induced-width
Of ordering





Tree-width
=3



Tree-width
=2

Tree-width of a graph = smallest cluster in a cluster-tree
Path-width of a graph = smallest cluster in a cluster-path

Two Primary Reasoning Methods

- Inference
  - Variable elimination
  - Tree-clustering
- Search
  - Backtracking (conditioning)
  - Branch and Bound
- Hybrids of search and inference
  - Cycle-cutset scheme

Road Map: Search in Variable-based Models

- Variable-based (Graphical) models
- **Basic search: DFS search, Backtracking**
- Constraint propagation: bounded inference
- Improving search by bounded-inference in branching ahead
- Improving search by looking-back
- The alternative AND/OR search space

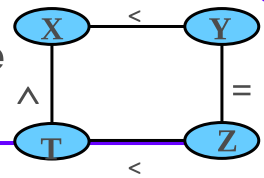
Focus: Constraint Networks

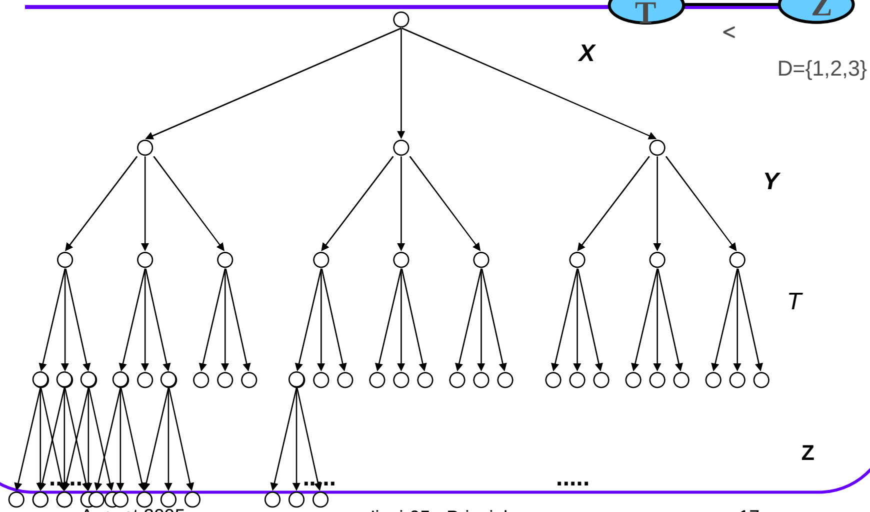
- Relation: allowed tuples (semantics)

X	Y	Z
1	3	2
2	1	3

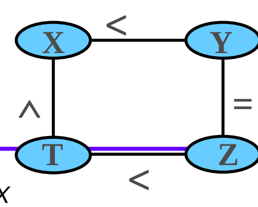
- Algebraic expression: $X + Y^2 \le 10, X \ne Y$
- Propositional formula: $(a \vee b) \to \neg c$

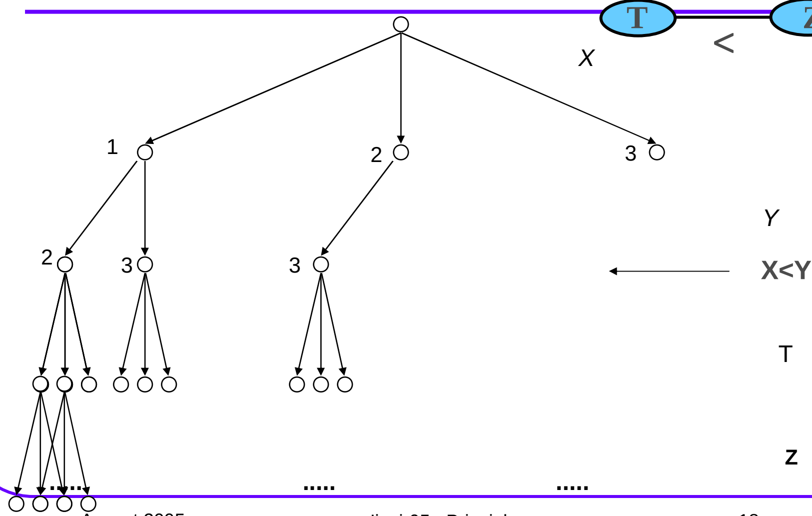
The Full Search Space



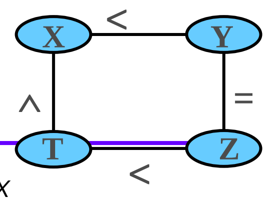


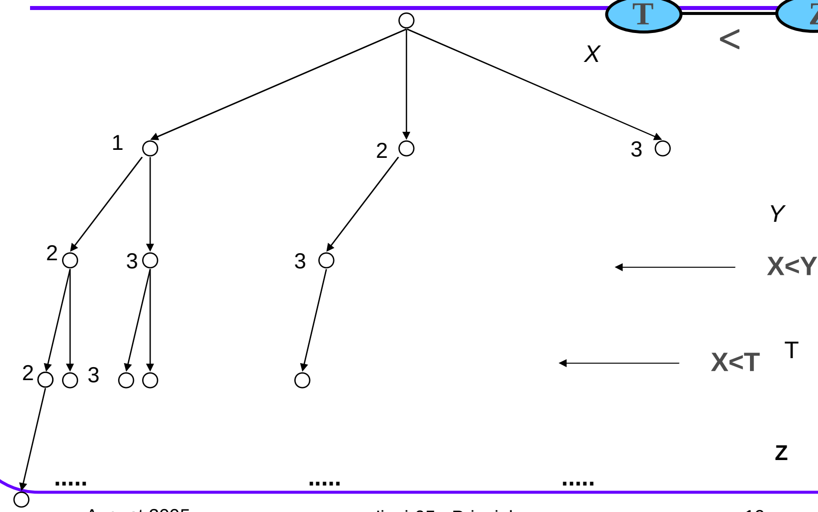
The Search Space



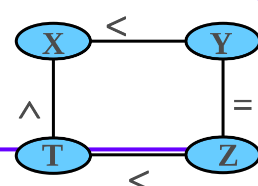


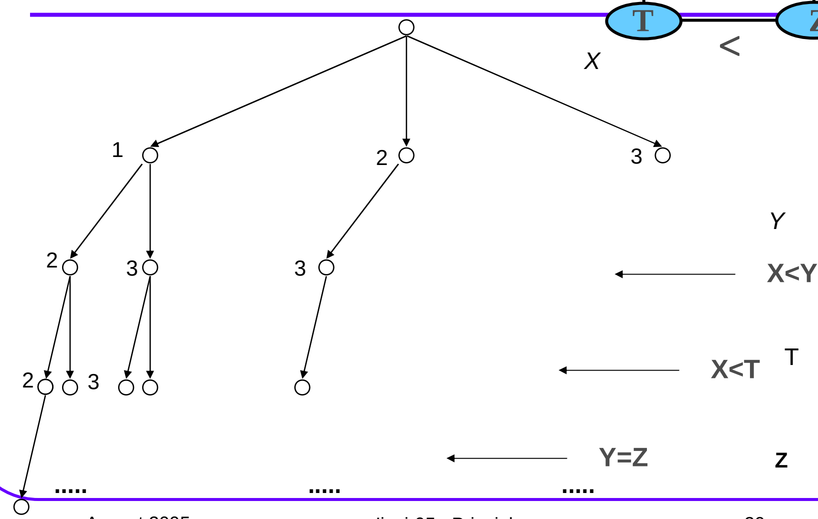
The Search Space



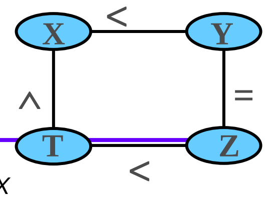


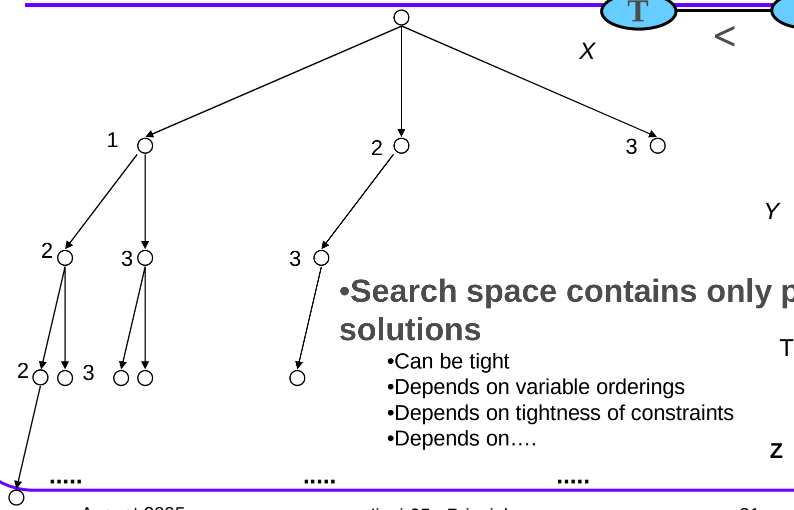
The Search Space





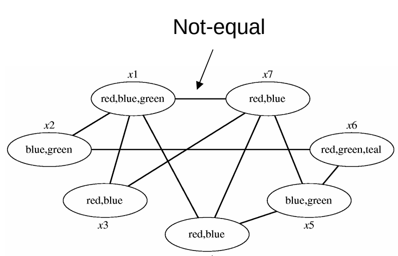
The Search Space

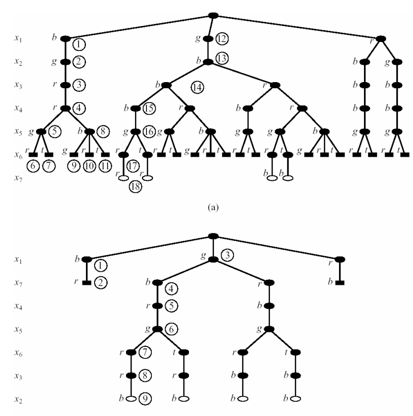




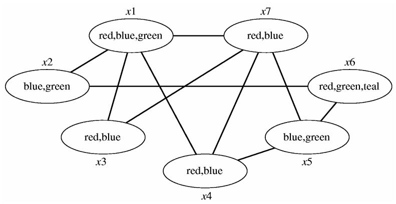
- Search space contains only partial solutions
  - Can be tight
  - Depends on variable orderings
  - Depends on tightness of constraints
  - Depends on….

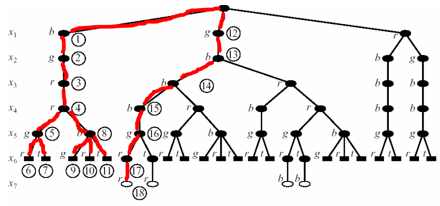
Traversing the search space:
Backtracking (DFS) Search for a Solution





Search for a Single Solution

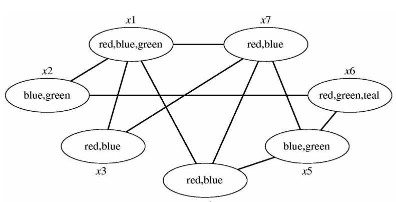


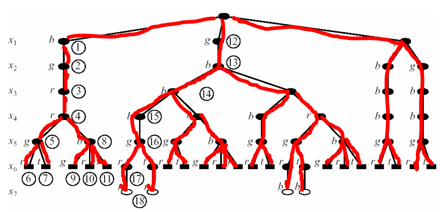


(a)

(b)

Search for All Solutions

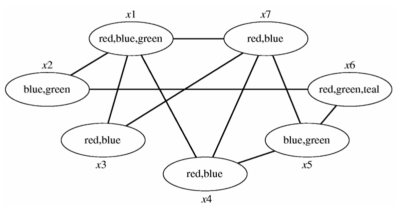


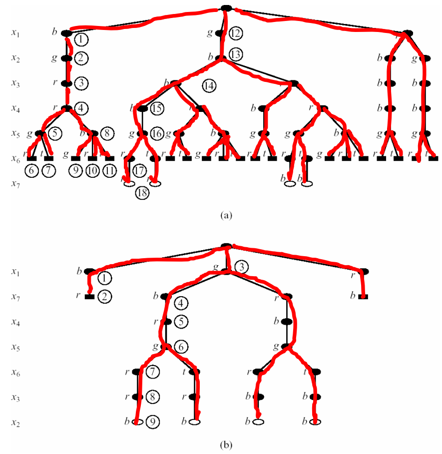


(a)

(b)

Search for All Solutions

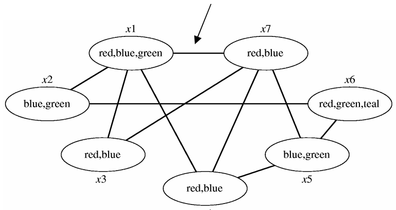


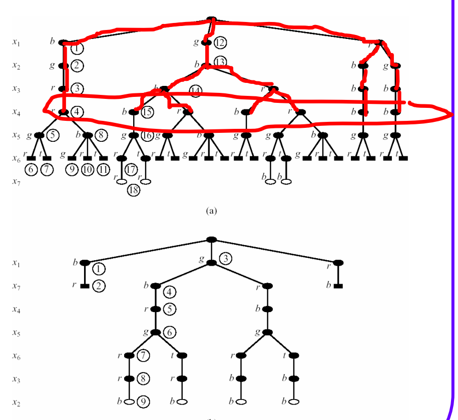


For all tasks
Time: O(exp(n))
Space: linear

Traversing Breadth-First (BFS)?

Not-equal





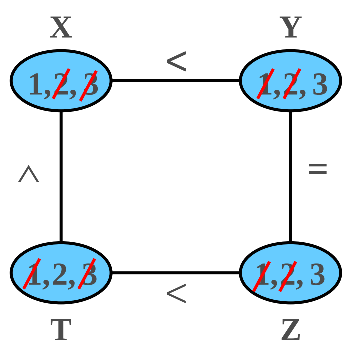
BFS space is exp(n) while no
Time gain → use DFS

Road Map:
Search in Variable-Based Models

- Variable-based (Graphical) models
- Basic search
- **Constraint propagation: bounded inference**
- Improving search by branching ahead
- Improving search by looking-back
- The alternative AND/OR search space
- Hybrid Schemes

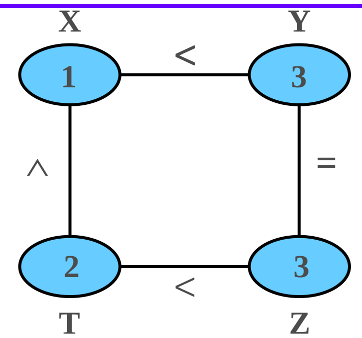
Arc-consistency

$1 \leq X, Y, Z, T \leq 3$
$X < Y$
$Y = Z$
$T < Z$
$X \leq T$



Arc-consistency

$1 \le X, Y, Z, T \le 3$
$X < Y$
$Y = Z$
$T < Z$
$X \le T$

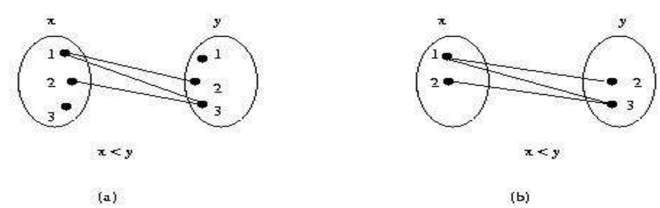


- Only domains are reduced: $R_X \leftarrow \prod_X R_{XY} \bowtie D_Y$
- Can be accomplished efficiently $O(ek^2)$

Arc-consistency

A binary constraint *R(X,Y)* is arc-consistent w.r.t. X is every value in x's domain has a match in y's domain.

$$R_X = \{1,2,3\},\ R_Y = \{1,2,3\},\ \text{constraint}\ \ X < Y$$



Revise(x, y) reduces domain of X to $R_X = \{1,2\}, O(k^2)$.

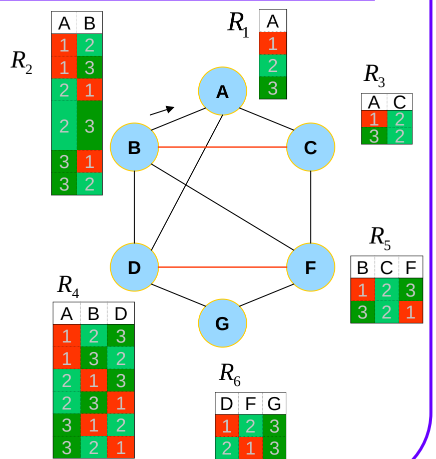
Distributed Relational Arc-consistency

The message that R2 sends to R1 is

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i))$$

R1 updates its relation and domains and sends messages to neighbors

$$D_i \leftarrow D_i \cap (\bowtie_{k \in ne(i)} D_k^i)$$



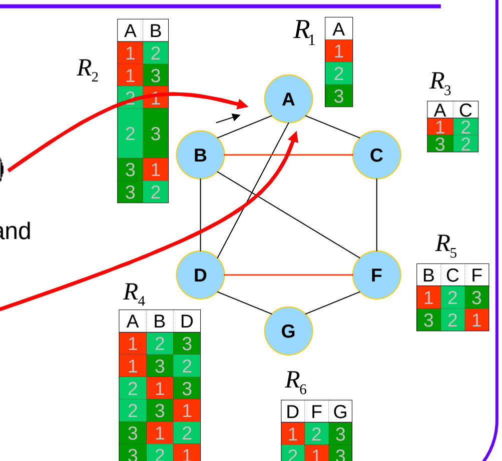
Relational Arc-consistency

The message that R2 sends to R1 is

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i))$$

R1 updates its relation and domains and sends messages to neighbors

$$D_i \leftarrow D_i \cap (\bowtie_{k \in ne(i)} D_k^i)$$



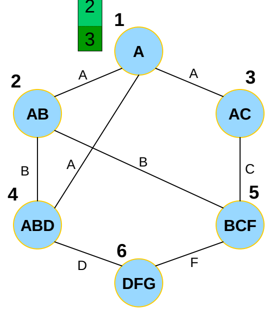
DR-AC on a Dual Join-Graph

R_1

A
1
2
3

R_2

A	B
1	2
1	3
2	1
2	3
3	1
3	2



R_3

A	C
1	2
3	2

R_4

A	B	D
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

R_6

D	F	G
1	2	3
2	1	3

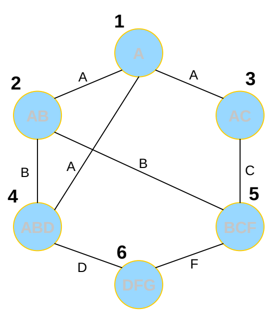
DR-AC on a dual join-graph

R_1

A
1
2
3

R_2

A	B
1	2
1	3
2	1
2	3
3	1
3	2



R_3

A	C
1	2
3	2

R_4

A	B	D
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

R_6

D	F	G
1	2	3
2	1	3

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i))$$

Iteration 1

R_1

A
1
2
3

h_2^1

A
1
2
3

h_3^1

A
1
3

h_4^1

A
1
2
3

h_5^2

B
1
3

h_4^2

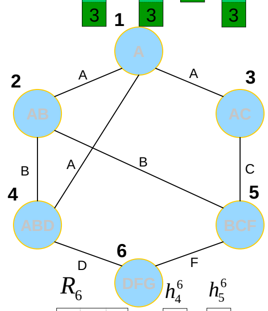
B
1
2
3

h_1^2

A
1
2
3

R_2

A	B
1	2
1	3
2	1
2	3
3	1
3	2



R_3

A	C
1	2
3	2

h_1^3

A
1
2
3

h_5^3

C
2

h_6^4

D
1
2

h_2^4

B
1
2
3

h_1^4

A
1
2
3

R_4

A	B	D
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

h_2^5

B
1
2
3

h_3^5

C
2

h_6^5

F
1
3

R_6

D	F	G
1	2	3
2	1	3

h_4^6

D
1
2
3

h_5^6

F
1
3

$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i)$$

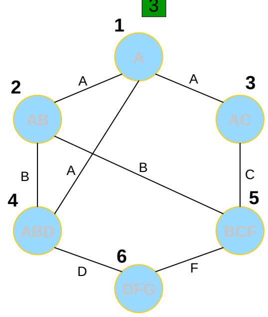
Iteration 1

R_2

A	B
1	3
2	1
2	3
3	1

R_1

A
1
3



R_3

A	C
1	2
3	2

R_4

A	B	D
1	3	2
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

R_6

D	F	G
2	1	3

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i))$$

Iteration 2

R_1

A
1
3

h_2^1

A
1
2
3

h_3^1

A
1
3

h_4^1

A
1
2
3

h_5^2

B
1
3

h_4^2

B
1
2
3

h_1^2

A
1
3

R_2

A	B
1	3
2	1
2	3
3	1

R_3

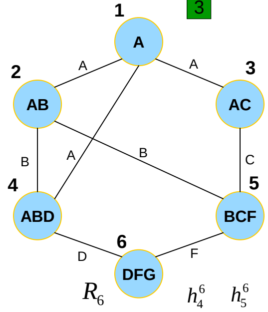
A	C
1	2
3	2

h_1^3

A
1
3

h_5^3

C
2



h_6^4

D
2

h_2^4

B
1
3

h_1^4

A
1
3

R_4

A	B	D
1	3	2
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

h_2^5

B
1
3

h_3^5

C
2

h_6^5

F
1

R_6

D	F	G
2	1	3

h_4^6

D
1
2

h_5^6

F
1
3

$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i)$$

Iteration 2

R_2

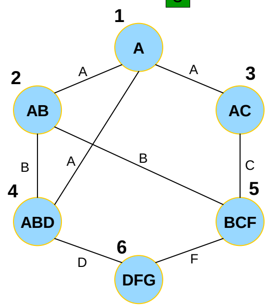
A	B
1	3
3	1

R_1

A
1
3

R_3

A	C
1	2
3	2



R_4

A	B	D
1	3	2
3	1	2

R_5

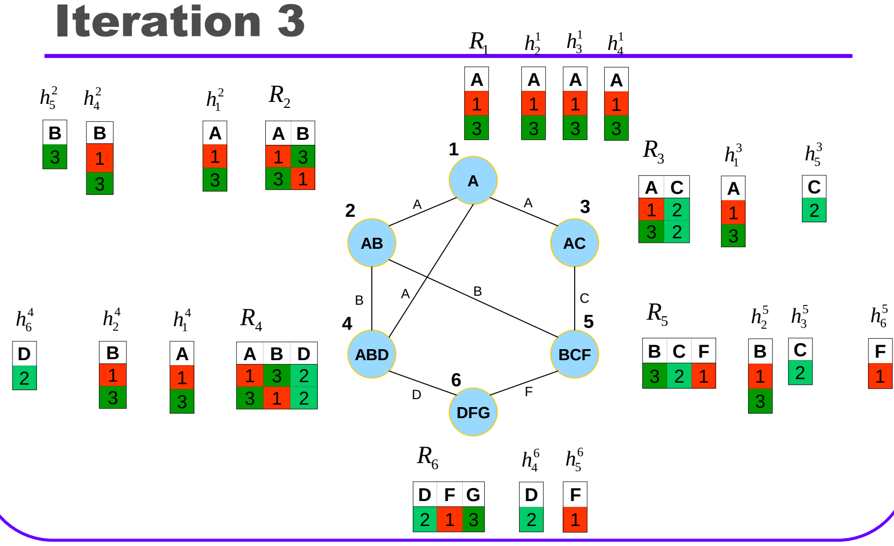
B	C	F
3	2	1

R_6

D	F	G
2	1	3

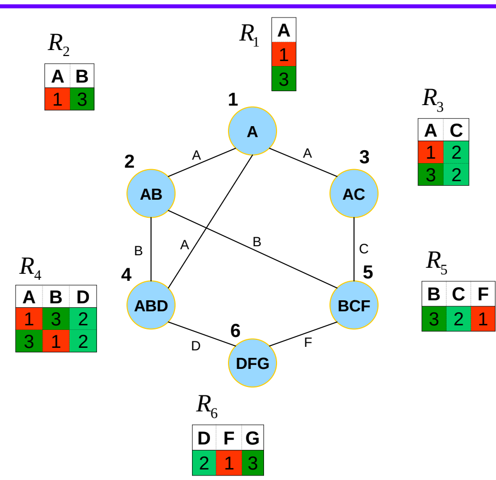
$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i))$$

Iteration 3



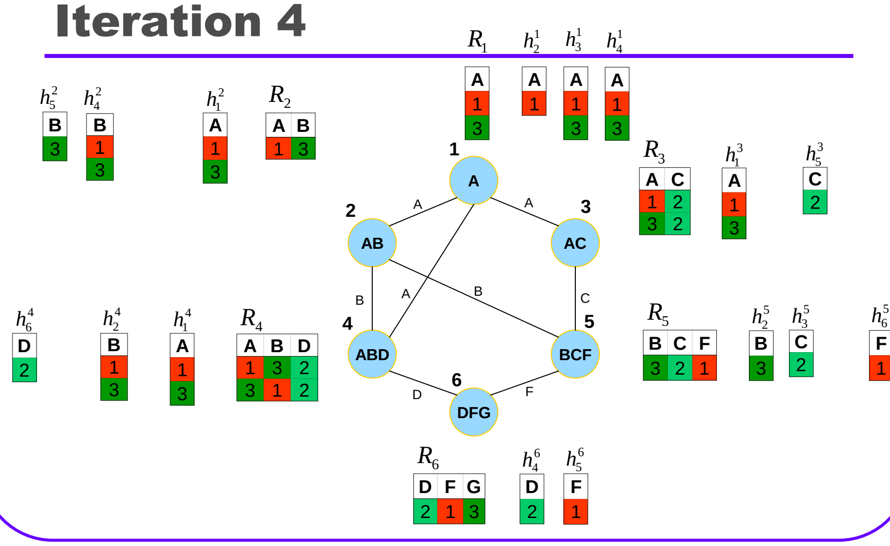
$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i)$$

Iteration 3



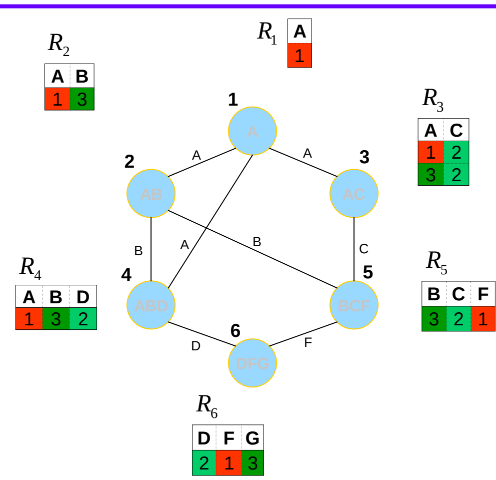
$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i))$$

Iteration 4



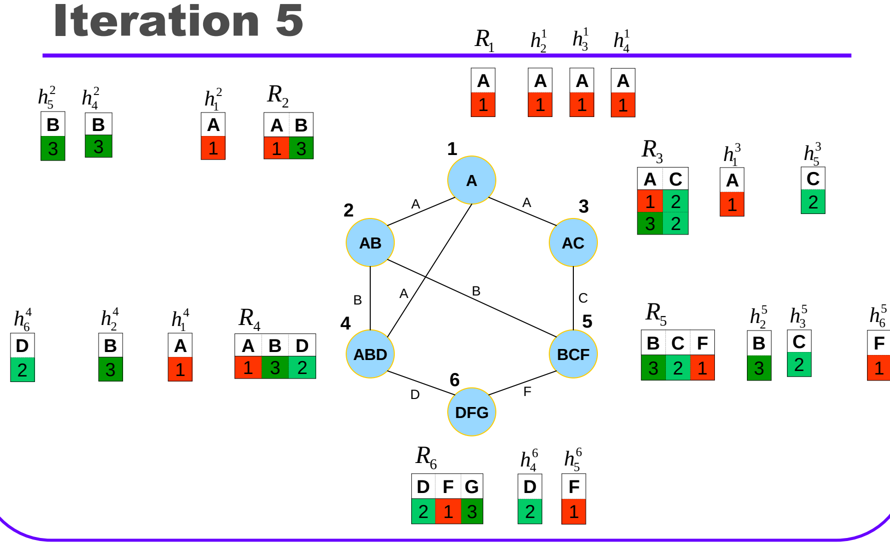
$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i)$$

Iteration 4

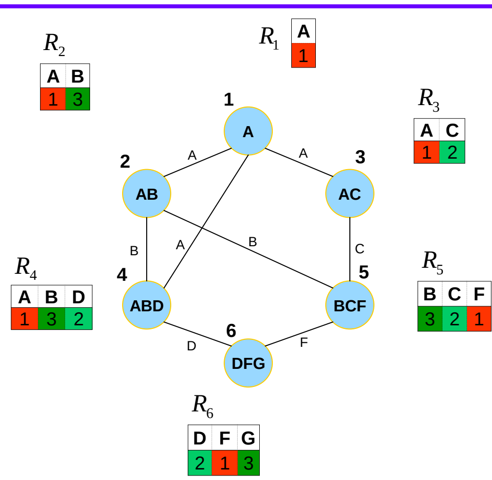


$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i))$$

Iteration 5



Arc-Consistent Network



Gausian and Boolean Propagation, Resolution

- Linear inequalities

$$x + y + z \leq 15, z \geq 13 \Rightarrow$$

- Boolean constraint propagation, unit resolution

$$(A \vee B \vee \neg C), (\neg B)$$

Gausian and Boolean Propagation, Resolution

- Linear inequalities

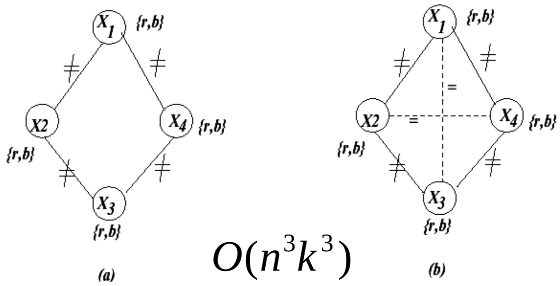
$$x + y + z \leq 15, z \geq 13 \Rightarrow$$
$$x \leq 2, y \leq 2$$

- Boolean constraint propagation, unit resolution

$$(A \vee B \vee \neg C), (\neg B) \Rightarrow$$
$$(A \vee \neg C)$$

Path-consistency

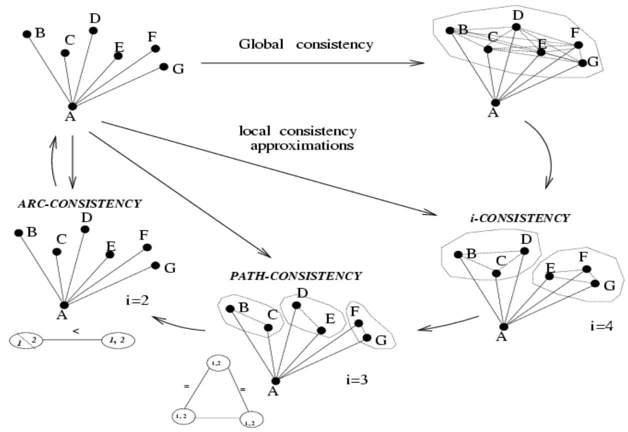
A pair (x, y) is path-consistent relative to Z, if every consistent assignment (x, y) has a consistent extension to z.



$$O(n^3k^3)$$

Figure 3.12: A graph-coloring graph (a) before path-consistency (b) after path-consistency

From Global to Local Consistency



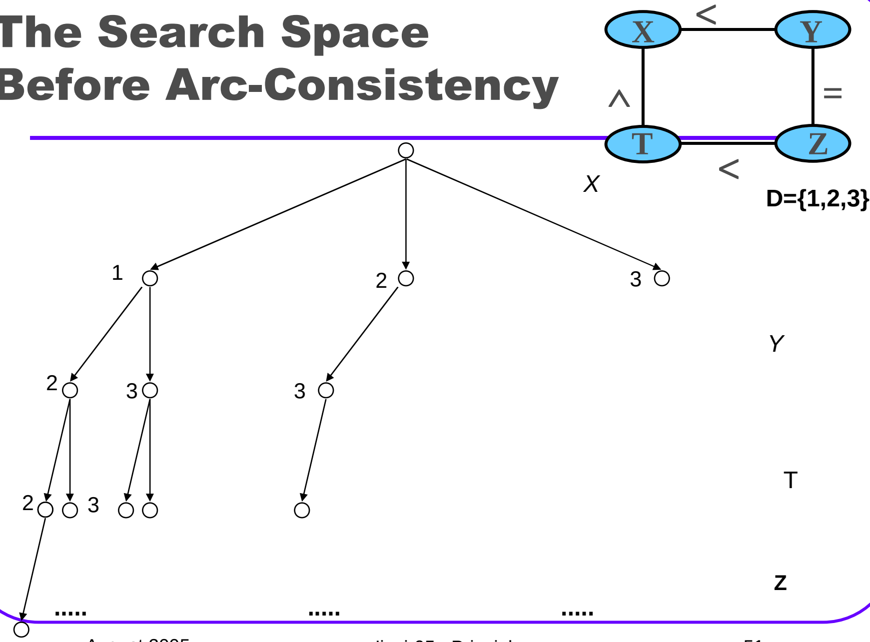
Road Map: Search in Variable-Based Models

- Variable-based (Graphical) models
- Basic search and basic Inference
- Constraint propagation: bounded inference
- **Improving search by constraint propagation in branching ahead**
- Improving search by looking-back
- The alternative AND/OR search space
- Hybrid Schemes

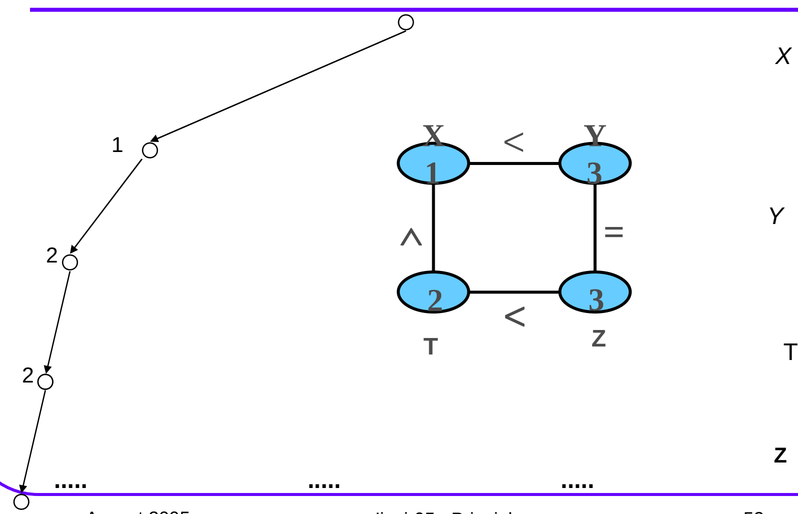
Improving Search by constraint propagation

- Before search: (reducing the search space)
  - By constraint propation (e.g., Arc-consistency)
- **During search:**
  - apply constraint propagation at each node,
  - pruning values, and
  - advising values and variable orderings.

The Search Space Before Arc-Consistency



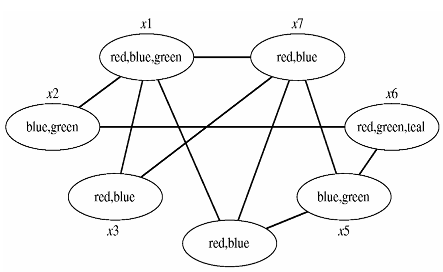
The Search Space After Arc-Consistency

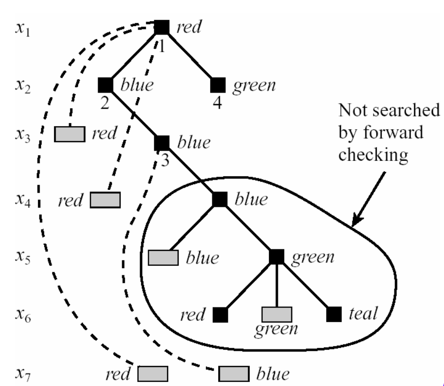


Branching-Ahead: Constraint Propagation in Search

- Apply some level of constraint propagation at each node,
  - Forward-checking (FC)
  - Arc consistency (MAC)
- Then:
  - Value pruning
  - Value ordering (choose a value that leaves **most** options open)
  - Variable ordering (choose a variable that leaves **least** options open

Forward-Checking for Value Ordering

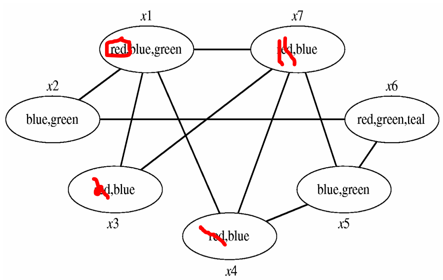


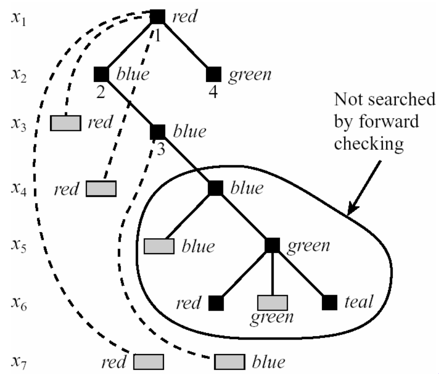


FW overhead: $O(ek^2)$

MAC overhead: $O(ek^3)$

Forward-Checking for Value Ordering

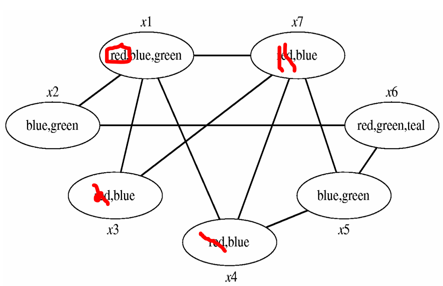


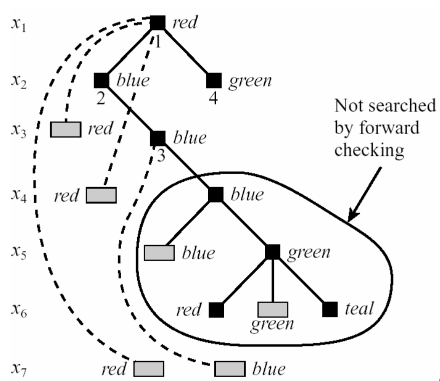


FW overhead: $O(ek^2)$

MAC overhead: $O(ek^3)$

MAC for Value Ordering



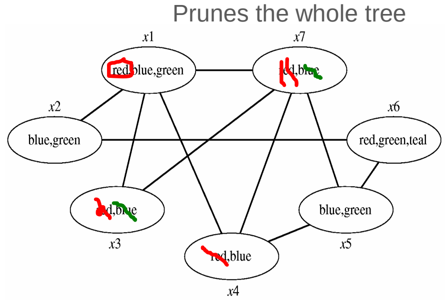


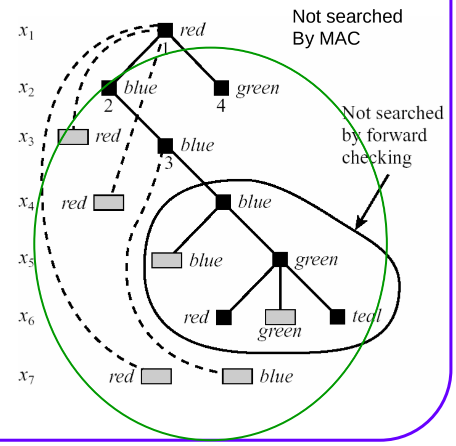
FW overhead: $O(ek^2)$

MAC overhead: $O(ek^3)$

MAC for Value Ordering

Arc-consistency prunes x1=red
Prunes the whole tree

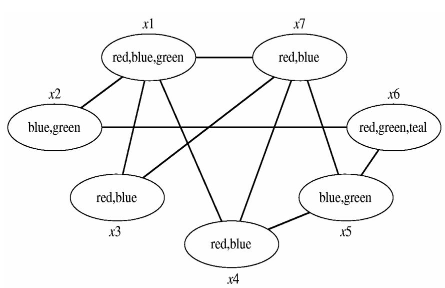


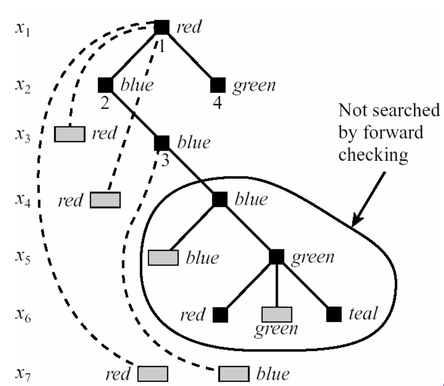


FW overhead: $O(ek^2)$

MAC overhead: $O(ek^3)$

Forward-Checking, Variable Ordering



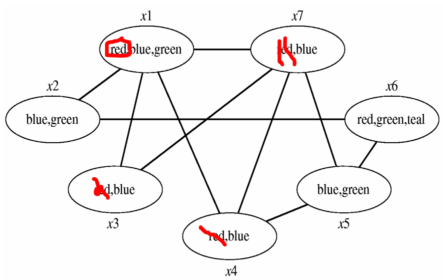


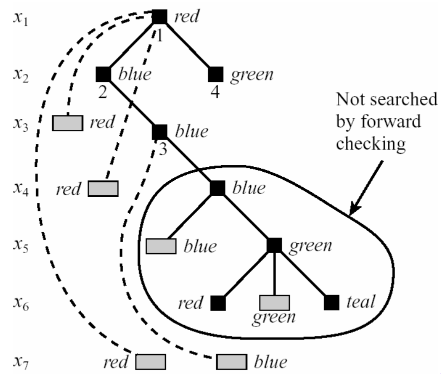
FW overhead: $O(ek^2)$

MAC overhead: $O(ek^3)$

Forward-Checking, Variable Ordering

After X1 = red choose X3 and not X2



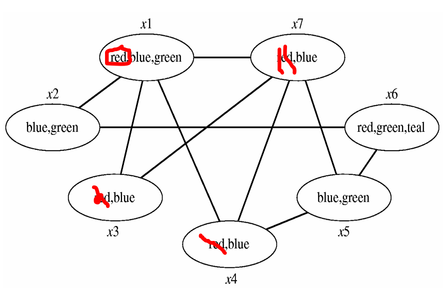


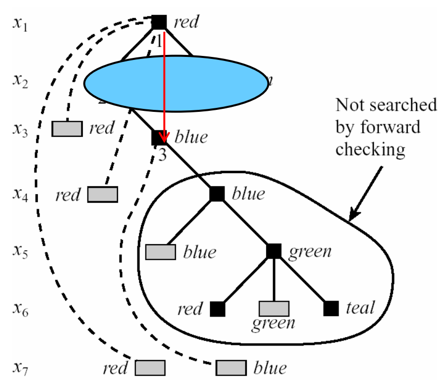
FW overhead: $O(ek^2)$

MAC overhead: $O(ek^3)$

Forward-Checking, Variable Ordering

After X1 = red choose X3 and not X2



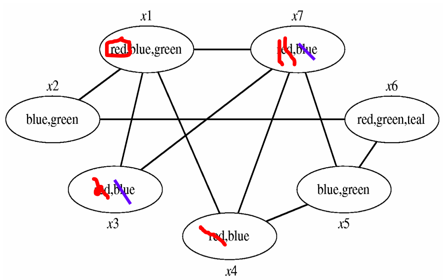


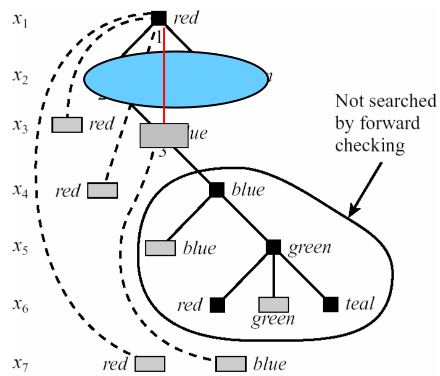
FW overhead: $O(ek^2)$

MAC overhead: $O(ek^3)$

Forward-Checking, Variable Ordering

After X1 = red choose X3 and not X2



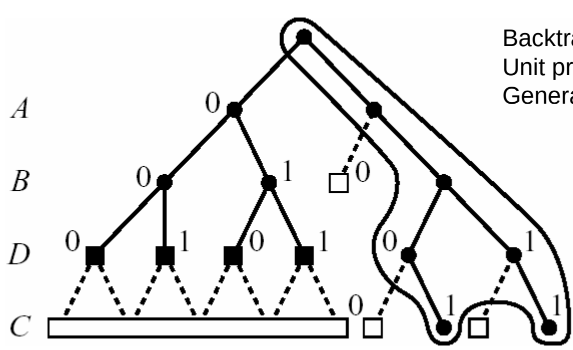


FW overhead: $O(ek^2)$

MAC overhead: $O(ek^3)$

Branching-Ahead for SAT: DLL
example: (~AVB)(~CVA)(AVBVD)(C)

(Davis, Logeman and Laveland, 1962)



Backtracking look-ahead with Unit propagation= Generalized arc-consistency

Only enclosed area will be explored with unit-propagation

Look-Back: Backjumping

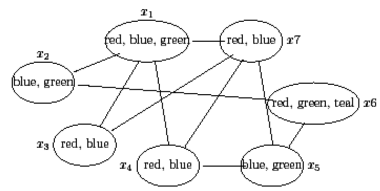
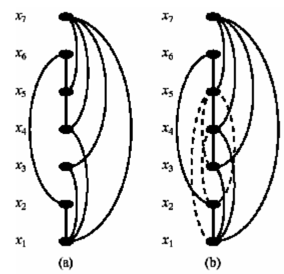
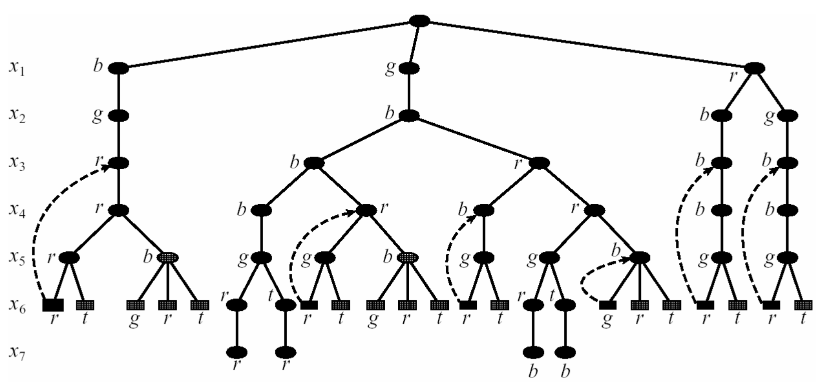


Figure 6.1: A modified coloring problem.



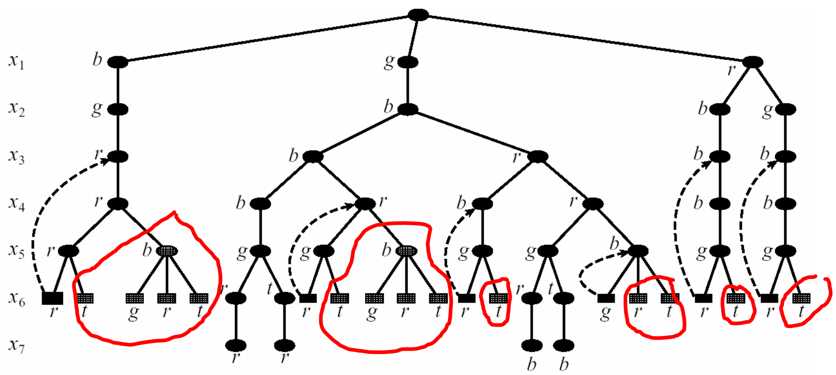
- (X1=r,x2=b,x3=b,x4=b,x5=g,x6=r,x7={r,b})
- (r,b,b,b,g,r) **conflict set** of x7
- (r,-,b,b,g,-) c.s. of x7
- (r,-,b,-,-,-,-) **minimal conflict-set**
- **Leaf deadend**: (r,b,b,b,g,r)
- Every conflict-set is a **no-good**

Jumps at dead-ends (Gascnnig 1977)



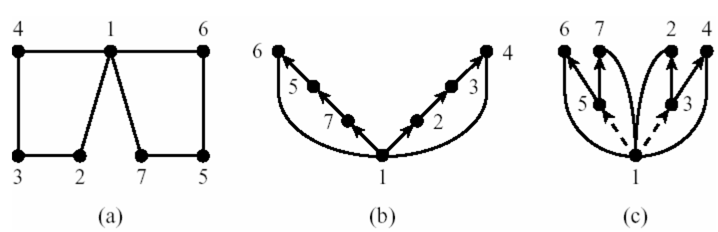
Example 6.3.1 In Figure 6.4, all of the backjumps illustrated lead to internal dead-ends, except for the jump back to $(\langle x_1, \mathit{green}\rangle, \langle x_2, \mathit{blue}\rangle, \langle x_3, \mathit{red}\rangle, \langle x_4, \mathit{blue}\rangle)$, because this is the only case where another value exists in the domain of the culprit variable. □

Jumps at Dead-Ends (Gascnnig 1977)



Example 6.3.1 In Figure 6.4, all of the backjumps illustrated lead to internal dead-ends, except for the jump back to $(\langle x_1, green\rangle, \langle x_2, blue\rangle, \langle x_3, red\rangle, \langle x_4, blue\rangle)$, because this is the only case where another value exists in the domain of the culprit variable. □

Complexity of Backjumping Uses Pseudo-Tree Analysis

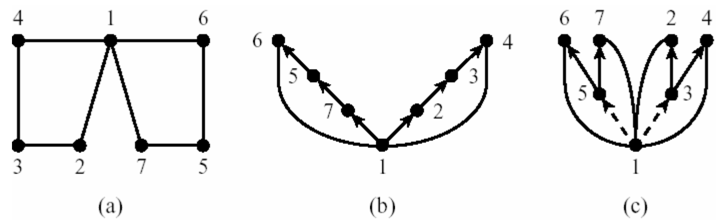


Simple: always jump back to parent in pseudo tree

Complexity for csp: exp(w*)

Complexity for csp: exp(w*log n)

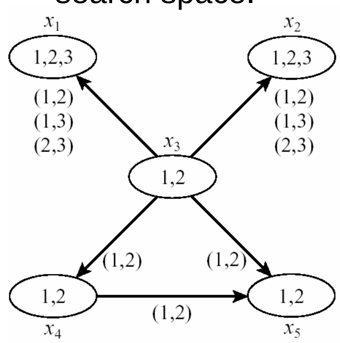
Complexity of Backjumping



Simple: always jump back to parent in pseudo tree
Complexity for csp: exp(w*)
Complexity for csp: exp(w*log n)
From exp(n) to exp(w*logn) while linear space

Look-back: No-good Learning

Learning means recording conflict sets used as constraints to prune future search space.



- (x1=2,x2=2,x3=1,x4=2) is a dead-end
- Conflicts to record:
  - (x1=2,x2=2,x3=1,x4=2) 4-ary
  - (x3=1,x4=2) binary
  - (x4=2) unary

No-good Learning Example

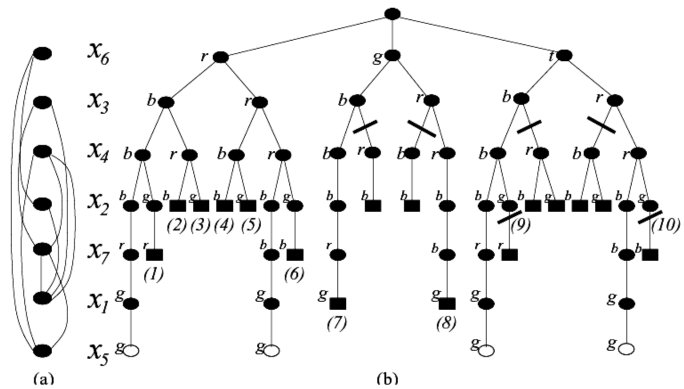


Figure 6.9: The search space explicated by backtracking on the CSP from Figure 6.1, using the variable ordering $(x_6, x_3, x_4, x_2, x_7, x_1, x_5)$ and the value ordering (*blue, red, green, teal*). Part (a) shows the ordered constraint graph, part (b) illustrates the search space. The cut lines in (b) indicate branches not explored when graph-based learning is used.

Complexity of Nogood-Learning for consistency

- The complexity of learning along d is time and space exponential in w*(d):
- The number of dead-ends is bounded by $O(nk^{w^*(d)})$
- Number of constraint tests per dead-end are $O(e)$

Space complexity is $O(nk^{w^*(d)})$
Time complexity is $O(n^2 e \cdot k^{w^*(d)})$

Complexity of Nogood-Learning for consistency

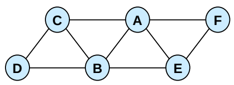
- The complexity of learning along d is time and space exponential in w*(d):
- The number of dead-ends is bounded by $O(nk^{w^*(d)})$
- Number of constraint tests per dead-end are $O(e)$

Space complexity is $O(nk^{w^*(d)})$
Time complexity is $O(n^2 e \cdot k^{w^*(d)})$
No-good Learning reduces time to O(exp(w*)) but O(exp(w*)) space.

All Solutions and Counting

- For all solutions and counting we will see
  - The additional impact of Good learning
  - BFS makes sense with good learning
  - BFS and DFS time and space exp(path-width)
  - Good-learning doesn't help consistency task

#CSP – OR Search Tree

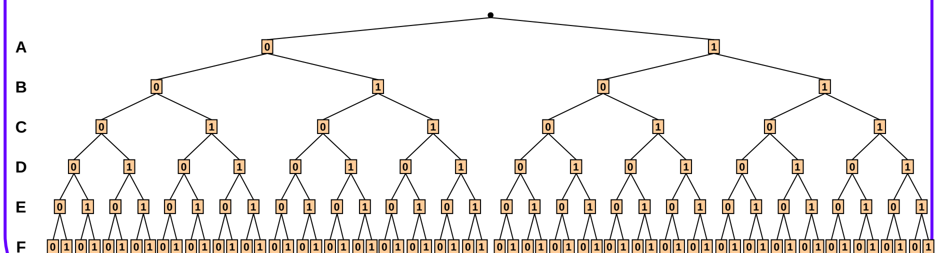


A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

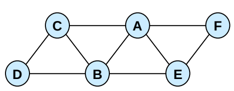
B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



#CSP - OR Search Tree

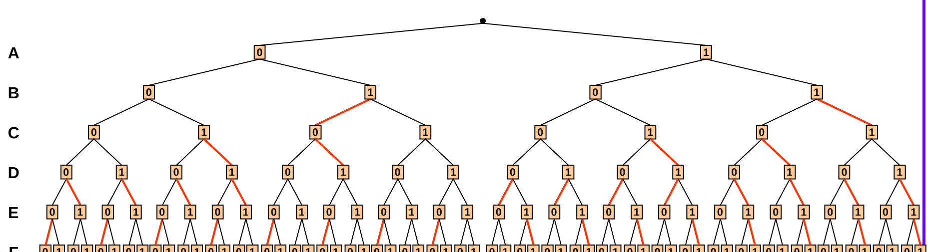


A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

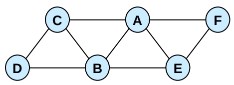
B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



#CSP - OR Search Tree



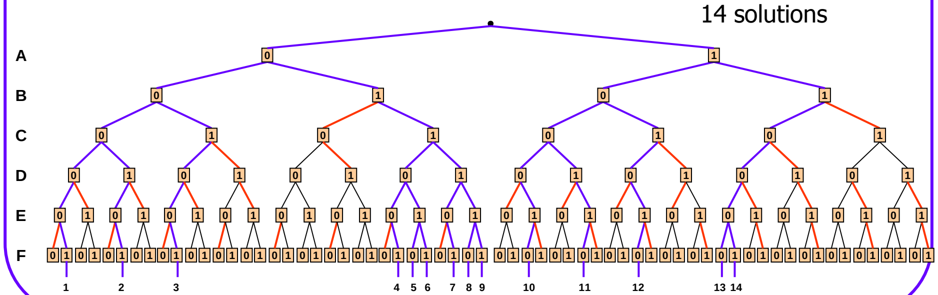
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

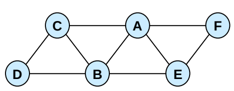
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

14 solutions



#CSP - Tree DFS Traversal

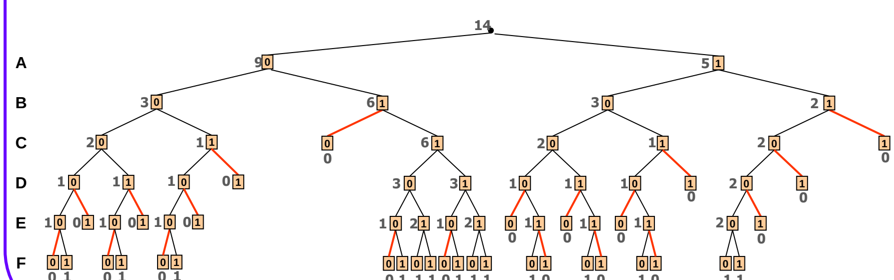


A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

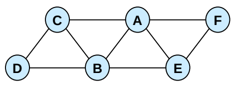
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



Value of node = number of solutions below it

#CSP - Searching the Graph by Good Caching



A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

A context(A) = [A]

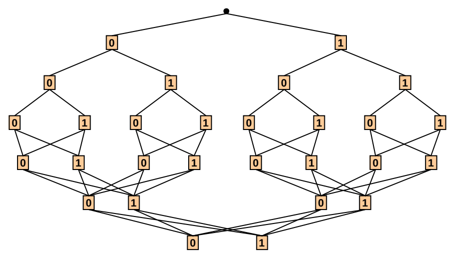
B context(B) = [AB]

C context(C) = [ABC]

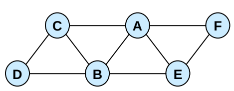
D context(D) = [ABD]

E context(E) = [AE]

F context(F) = [F]



#CSP - Searching the Graph by Good Caching



A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

A context(A) = [A]

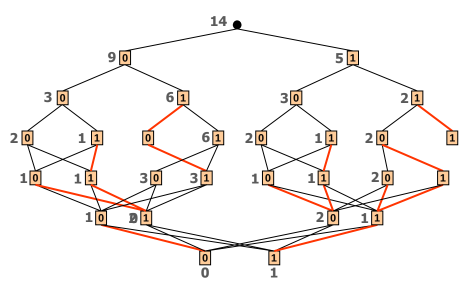
B context(B) = [AB]

C context(C) = [ABC]

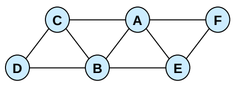
D context(D) = [ABD]

E context(E) = [AE]

F context(F) = [F]



#CSP - Searching the Graph by Good Caching



A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

A context(A) = [A]

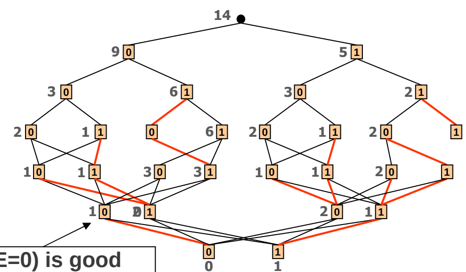
B context(B) = [AB]

C context(C) = [ABC]

D context(D) = [ABD]

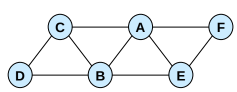
E context(E) = [AE]

F context(F) = [F]

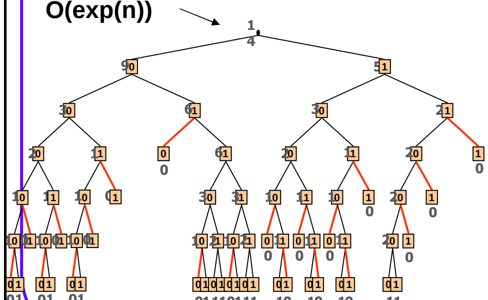


(A=0,E=0) is good
V(A=0,E=0)=1

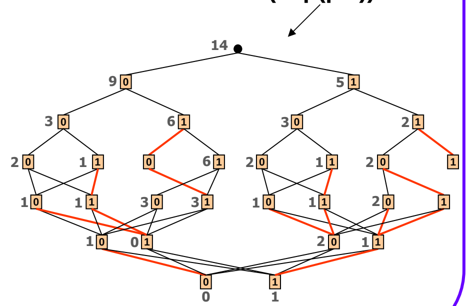
#CSP - Searching the Graph by Good Caching



No caching:
O(exp(n))



Good-caching:
O(exp(pw))



Summary: Time-Space for Constraint Processing

- Constraint-satisfaction one solution
  - Naive backtracking
    - Space: O(n),
    - Time: O(exp(n))
  - Backjumping
    - Space: O(n),
    - Time: O(exp(log n w*))
  - Learning no-goods
    - Space: O(exp(w*))
    - Time: O(exp(w*))
  - Variable-elimination
    - Space: O(exp(w*))
    - Time: O(exp(w*))
- Counting, enumeration
  - **Backtracking, backjumping**
    - **Space: O(n),**
    - **Time: O(exp(n))**
  - **Learning no-goods**
    - **space: O(exp(w*))**
    - **Time: O(exp(n))**
  - **Search with goods and no-goods learning**
    - **Space: O(exp(pw*))**
    - **Time: O(exp(pw*)), pw <=w*logn**
  - **Variable-elimination**
    - **Space: O(exp(w*))**
    - **Time: O(exp(w*))**
  - **BFS is time and space O(exp(pw*))**

Summary: Search Principles

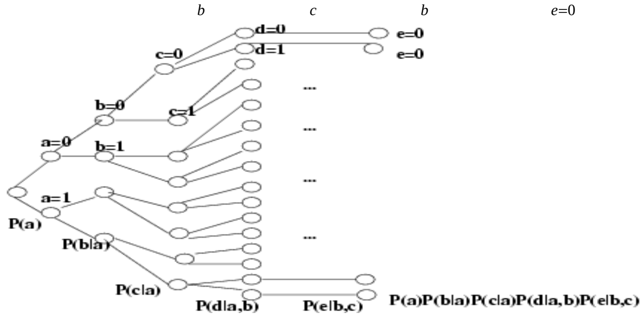
- DFS is better than BFS search
- Constraint propagation (bounded inference) prunes search space
- Constraint propagation yields good advise for how to branch and where to go
- Backjumping and no-good learning helps prune search space and revise problem.
- Good learning revise problem but helps only counting, enumeration

From Constraint Processing to

- **Belief networks tasks**
- **Optimization tasks**
- Task expressed as a **value** of a root node:
  - V(n) = probability of sub-tree below n
  - V(n) = optimal solution below n
  - V(n) can be derived recursively
  - V(n) have Bellman recursive equations

Belief Updating: Searching the Probability Tree

$$P(a, e=0) = P(a)\sum_b P(b \mid a)\sum_c P(c \mid a)\sum_b P(d \mid a,b)\sum_{e=0} P(e \mid b,c)$$

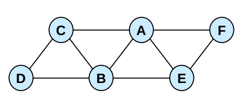


Brute-force Complexity: exponential time, linear space
Very similar to counting solution task

Caching Goods for Beliefs

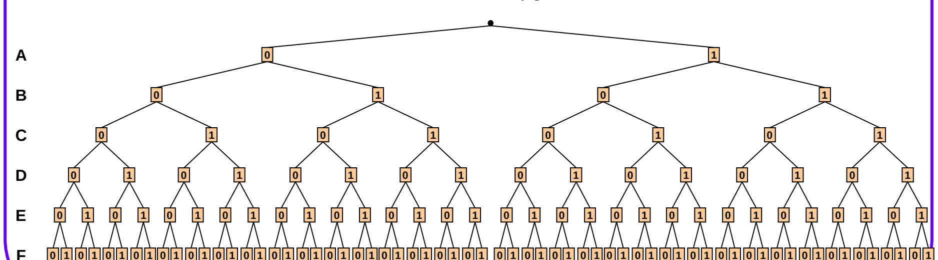
- For strictly positive distributions only caching goods is relevant.
- Both BFS and DFS are relevant
- Time and space O(exp(path-width))

Optimization Tasks

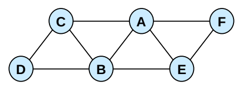


A	B	f_1	A	C	f_2	A	E	f_3	A	F	f_4	B	C	f_5	B	D	f_6	B	E	f_7	C	D	f_8	E	F	f_9
0	0	**2**	0	0	**3**	0	0	**0**	0	0	**2**	0	0	**0**	0	0	**4**	0	0	**3**	0	0	**1**	0	0	**1**
0	1	**0**	0	1	**0**	0	1	**3**	0	1	**0**	0	1	**1**	0	1	**2**	0	1	**2**	0	1	**4**	0	1	**0**
1	0	**1**	1	0	**0**	1	0	**2**	1	0	**0**	1	0	**2**	1	0	**1**	1	0	**1**	1	0	**0**	1	0	**0**
1	1	**4**	1	1	**1**	1	1	**0**	1	1	**2**	1	1	**4**	1	1	**0**	1	1	**0**	1	1	**0**	1	1	**2**

Objective function: $f(X) = \min_X \sum_{i=1}^{9} f_i(X)$



Optimization Tasks



A	B	f_1
0	0	**2**
0	1	**0**
1	0	**1**
1	1	**4**

A	C	f_2
0	0	**3**
0	1	**0**
1	0	**0**
1	1	**1**

A	E	f_3
0	0	**0**
0	1	**3**
1	0	**2**
1	1	**0**

A	F	f_4
0	0	**2**
0	1	**0**
1	0	**0**
1	1	**2**

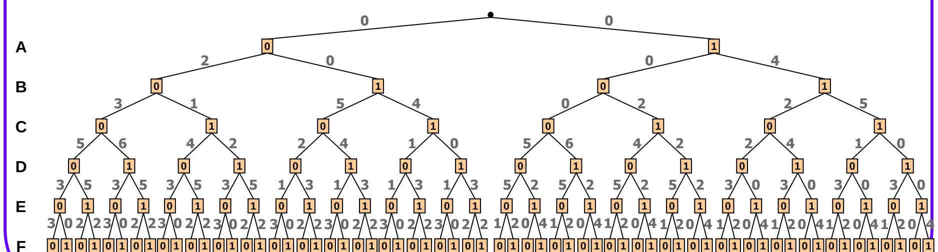
B	C	f_5
0	0	**0**
0	1	**1**
1	0	**2**
1	1	**4**

B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

B	E	f_7
0	0	**3**
0	1	**2**
1	0	**1**
1	1	**0**

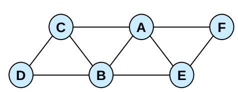
C	D	f_8
0	0	**1**
0	1	**4**
1	0	**0**
1	1	**0**

E	F	f_9
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**



Arc-cost is caculated based on cost components.

Tree DFS Traversal for Values



A	B	f_1
0	0	**2**
0	1	**0**
1	0	**1**
1	1	**4**

A	C	f_2
0	0	**3**
0	1	**0**
1	0	**0**
1	1	**1**

A	E	f_3
0	0	**0**
0	1	**3**
1	0	**2**
1	1	**0**

A	F	f_4
0	0	**2**
0	1	**0**
1	0	**0**
1	1	**2**

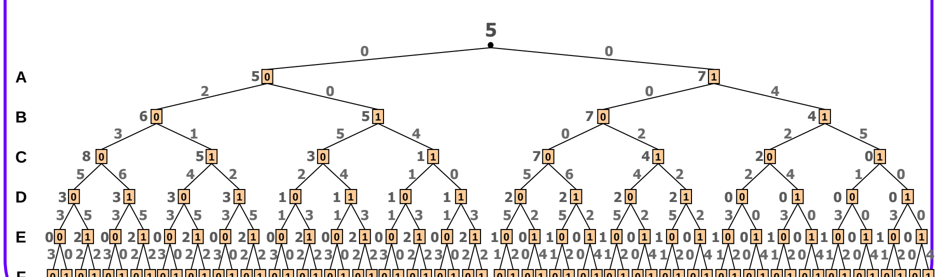
B	C	f_5
0	0	**0**
0	1	**1**
1	0	**2**
1	1	**4**

B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

B	E	f_7
0	0	**3**
0	1	**2**
1	0	**1**
1	1	**0**

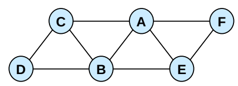
C	D	f_8
0	0	**1**
0	1	**4**
1	0	**0**
1	1	**0**

E	F	f_9
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**



Value of node = minimal cost solution below it

Tree DFS Traversal for Values



A	B	f_1
0	0	2
0	1	0
1	0	1
1	1	4

A	C	f_2
0	0	3
0	1	0
1	0	0
1	1	1

A	E	f_3
0	0	0
0	1	3
1	0	2
1	1	0

A	F	f_4
0	0	2
0	1	0
1	0	0
1	1	2

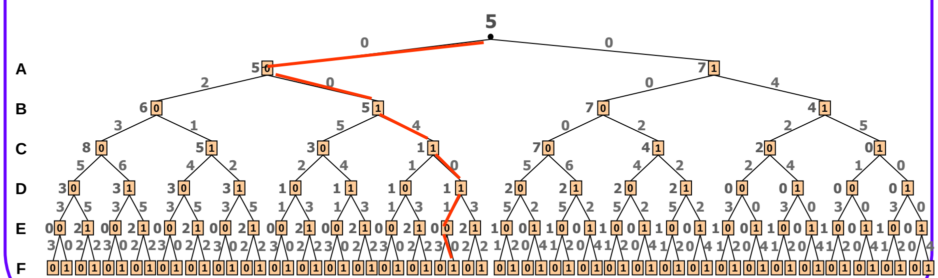
B	C	f_5
0	0	0
0	1	1
1	0	2
1	1	4

B	D	f_6
0	0	4
0	1	2
1	0	1
1	1	0

B	E	f_7
0	0	3
0	1	2
1	0	1
1	1	0

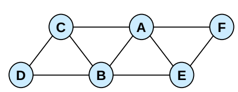
C	D	f_8
0	0	1
0	1	4
1	0	0
1	1	0

E	F	f_9
0	0	1
0	1	0
1	0	0
1	1	2



Value of node = minimal cost solution below it

Searching the Graph



A	B	f_1
0	0	2
0	1	0
1	0	1
1	1	4

A	C	f_2
0	0	3
0	1	0
1	0	0
1	1	1

A	E	f_3
0	0	0
0	1	3
1	0	2
1	1	0

A	F	f_4
0	0	2
0	1	0
1	0	0
1	1	2

B	C	f_5
0	0	0
0	1	1
1	0	2
1	1	4

B	D	f_6
0	0	4
0	1	2
1	0	1
1	1	0

B	E	f_7
0	0	3
0	1	2
1	0	1
1	1	0

C	D	f_8
0	0	1
0	1	4
1	0	0
1	1	0

E	F	f_9
0	0	1
0	1	0
1	0	0
1	1	2

A **context(A) = [A]**

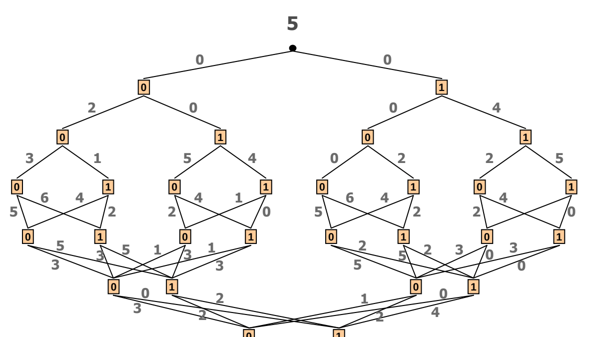
B **context(B) = [AB]**

C **context(C) = [ABC]**

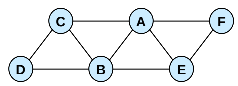
D **context(D) = [ABD]**

E **context(E) = [AE]**

F **context(F) = [F]**



Searching the Graph



A	B	f_1	A	C	f_2	A	E	f_3	A	F	f_4	B	C	f_5	B	D	f_6	B	E	f_7	C	D	f_8	E	F	f_9
0	0	**2**	0	0	**3**	0	0	**0**	0	0	**2**	0	0	**0**	0	0	**4**	0	0	**3**	0	0	**1**	0	0	**1**
0	1	**0**	0	1	**0**	0	1	**3**	0	1	**0**	0	1	**1**	0	1	**2**	0	1	**2**	0	1	**4**	0	1	**0**
1	0	**1**	1	0	**0**	1	0	**2**	1	0	**0**	1	0	**2**	1	0	**1**	1	0	**1**	1	0	**0**	1	0	**0**
1	1	**4**	1	1	**1**	1	1	**0**	1	1	**2**	1	1	**4**	1	1	**0**	1	1	**0**	1	1	**0**	1	1	**2**

A context(A) = [A]

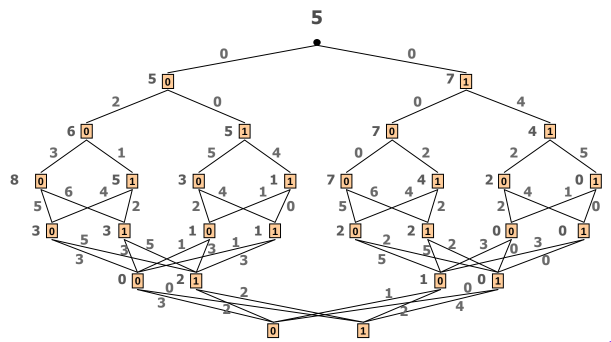
B context(B) = [AB]

C context(C) = [ABC]

D context(D) = [ABD]

E context(E) = [AE]

F context(F) = [F]



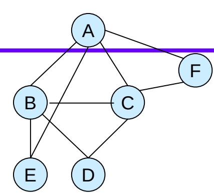
Summary:
Time-Space for Optimization/Belief

- **DFS Search (with/without backjumping)**
  - **Space: O(n), Time: O(exp(n))**
- **DFS Search with no-goods caching only**
  - **space: O(exp(w*)) Time: O(exp(n))**
- **Search with goods and no-goods learning**
  - **Space: O(exp(pw*)) Time: O(exp(pw*)), pw <=w*logn**
- **For optimization**
  - **BFS Space: O(exp(c*)) Time: O(exp(c*)), c*=best-solution length.**
- **Variable-elimination**
  - **Space: O(exp(w*)) Time: O(exp(w*))**

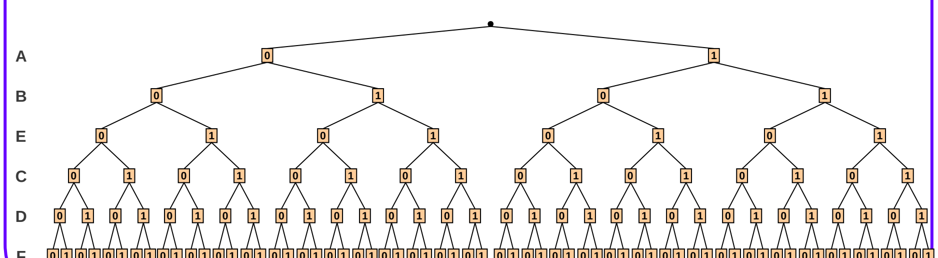
Road Map: Search in Graphical Models

- Variable-based (Graphical) models
- Basic search
- Constraint propagation: bounded inference
- Improving search by bounded-inference in branching ahead
- Improving search by looking-back
- **The alternative AND/OR search space**

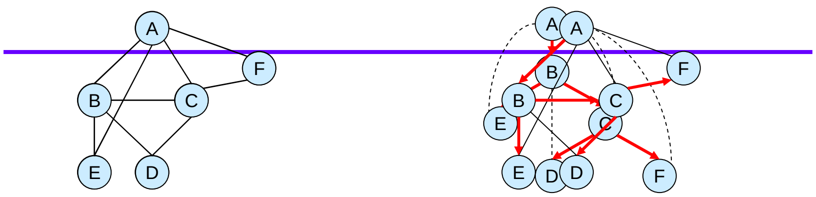
OR Search Space



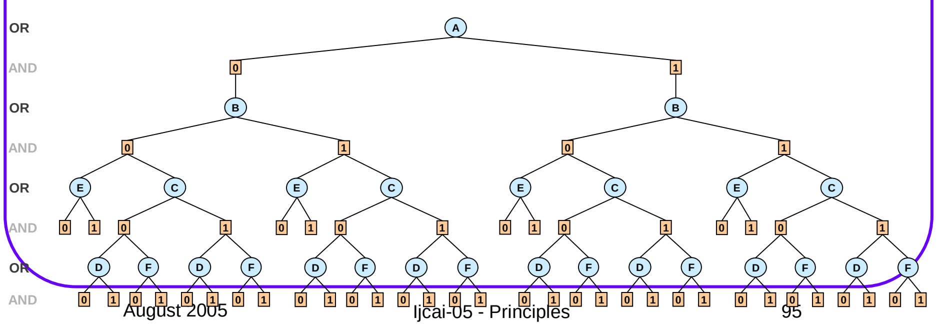
Ordering: A B E C D F



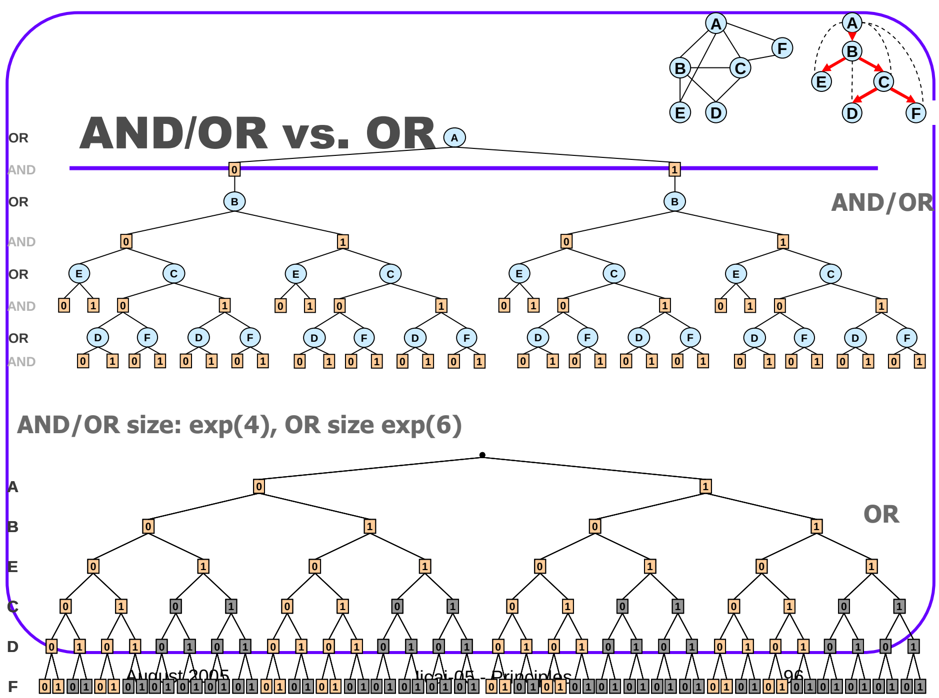
AND/OR Search Space



Primal graph DFS tree

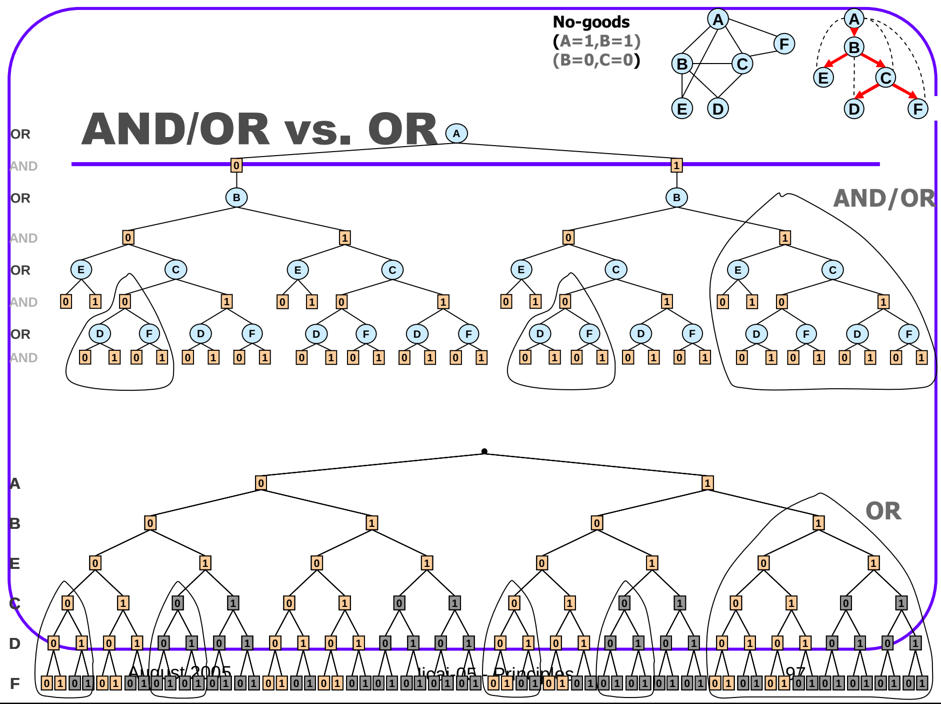


AND/OR vs. OR



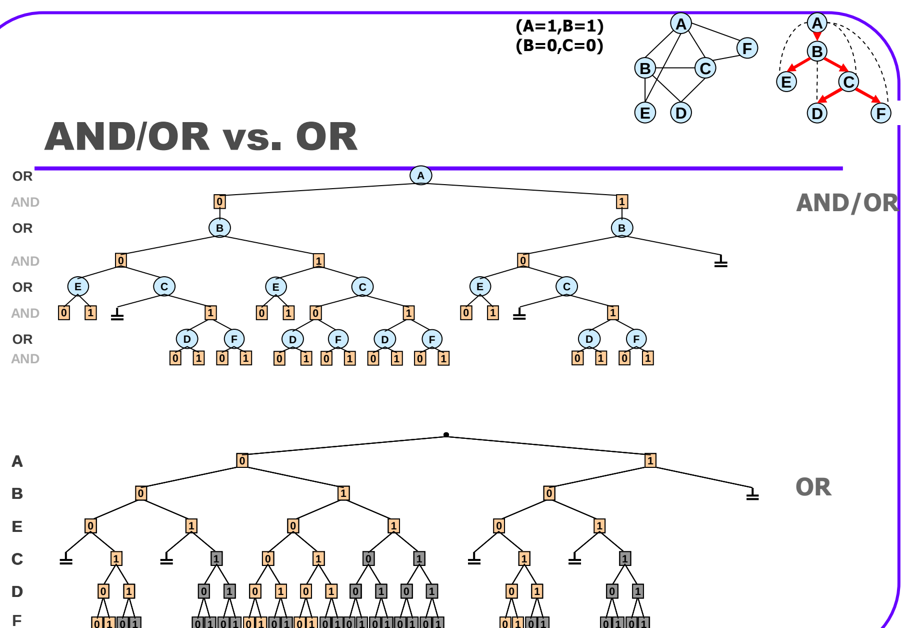
AND/OR vs. OR

No-goods
(A=1,B=1)
(B=0,C=0)

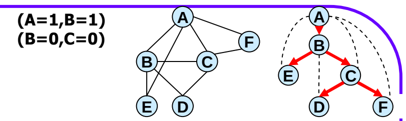


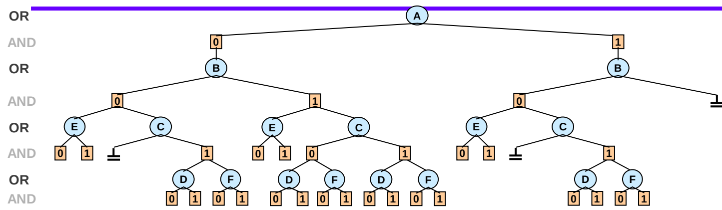
AND/OR vs. OR

(A=1,B=1)
(B=0,C=0)



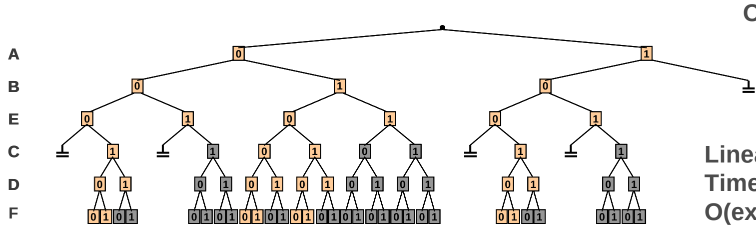
AND/OR vs. OR





AND/OR

Space: linear
Time:
O(exp(m))
O(w* log n)



OR

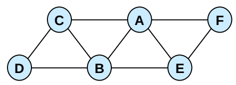
Linear space,
Time:
O(exp(n))

OR space vs. AND/OR space

width	height	OR space			AND/OR space		
		Time (sec.)	Nodes	Backtracks	Time (sec.)	AND nodes	OR nodes
5	10	3.154	2,097,150	1,048,575	0.03	10,494	5,247
4	9	3.135	2,097,150	1,048,575	0.01	5,102	2,551
5	10	3.124	2,097,150	1,048,575	0.03	8,926	4,463
4	10	3.125	2,097,150	1,048,575	0.02	7,806	3,903
5	13	3.104	2,097,150	1,048,575	0.1	36,510	18,255

Random graphs with 20 nodes, 20 edges and 2 values per node.

#CSP – AND/OR Search Tree

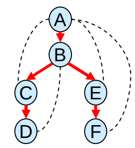


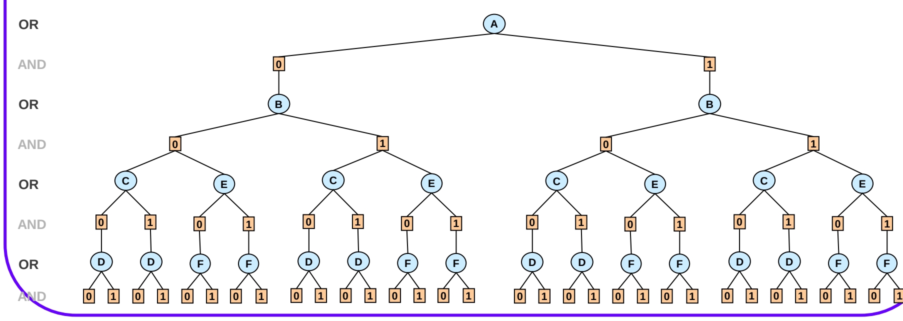
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

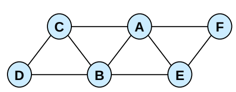
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





#CSP – AND/OR Search Tree

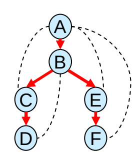


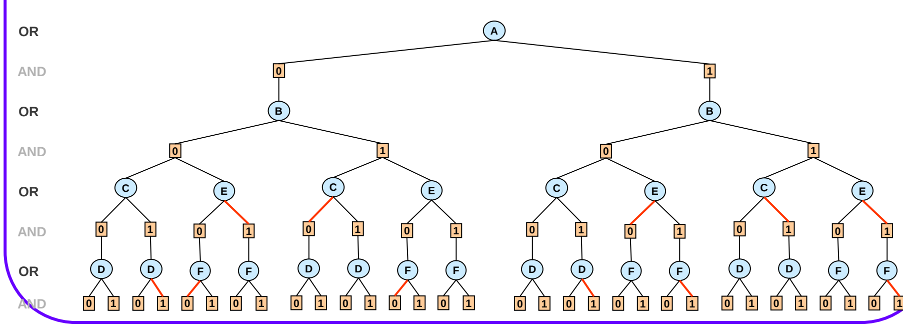
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

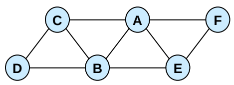
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





#CSP – AND/OR Tree DFS

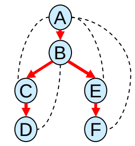


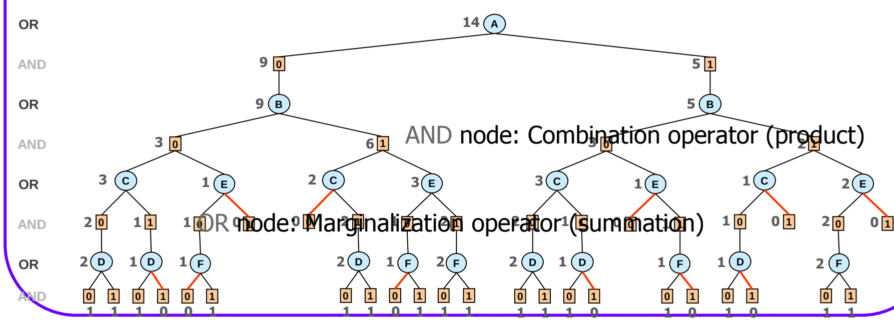
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



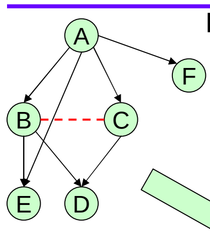


AND node: Combination operator (product)

OR node: Marginalization operator (summation)

Mixed Networks

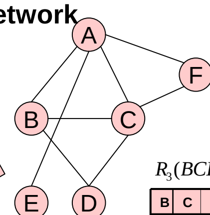
Belief Network



$P(D|B,C)$

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

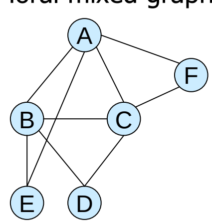
Constraint Network



$R_3(BCD)$

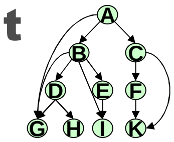
B	C	D
0	0	1
0	1	0
1	1	0

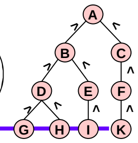
Moral mixed graph



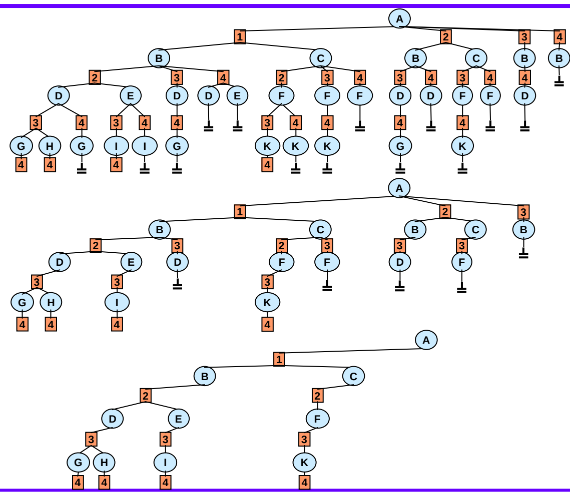
$$P_M(\bar{x}) = \begin{cases} P_B(\bar{x} \mid \bar{x} \in \rho) = \dfrac{P_B(\bar{x})}{P_B(\bar{x} \in \rho)}, \text{if } \bar{x} \in \rho \\ 0, \text{otherwise} \end{cases}$$

The Effect of Constraint Propagation





Domains are {1,2,3,4}

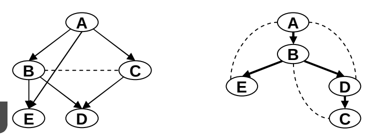


CONSTRAINTS ONLY

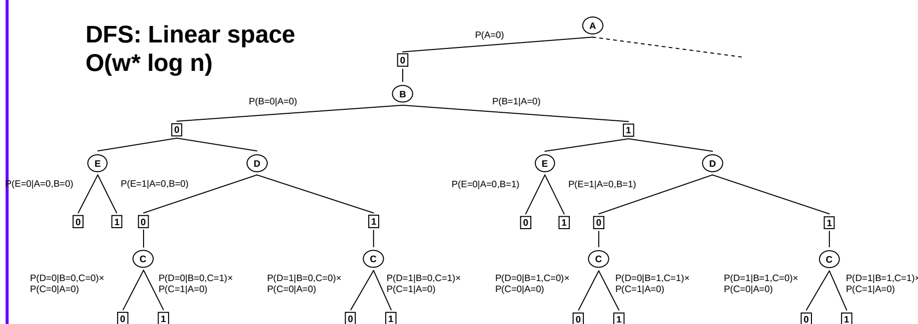
FORWARD CHECKING

MAINTAINING ARC CONSISTENCY

Belief-Updating



DFS: Linear space
O(w* log n)



Tasks and value of nodes

- V(n) is the value of the tree T(n) for the task:
  - Consistency: **v(n) is 0 if T(n) inconsistent, 1 othewise.**
  - Counting: **v(n) is number of solutions in T(n)**
  - Optimization: **v(n) is the optimal solution in T(n)**
  - Belief updating: **v(n), probability of evidence in T(n).**
  - Partition function: **v(n) is the total probability in T(n).**

- **Goal: compute the value of the root node recursively using DFS or BFS search of the AND/OR tree.**

- **Theorem: Complexity of AO DFS search is**
  - **Space: O(n)**
  - **Time: O(n k^m)**
  - **Time: O(exp(w* log n))**
- **Time and Space of BFS: O(exp(w* log n))**

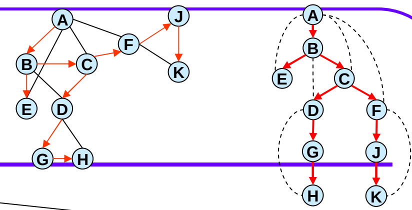
From Search Trees to Search Graphs

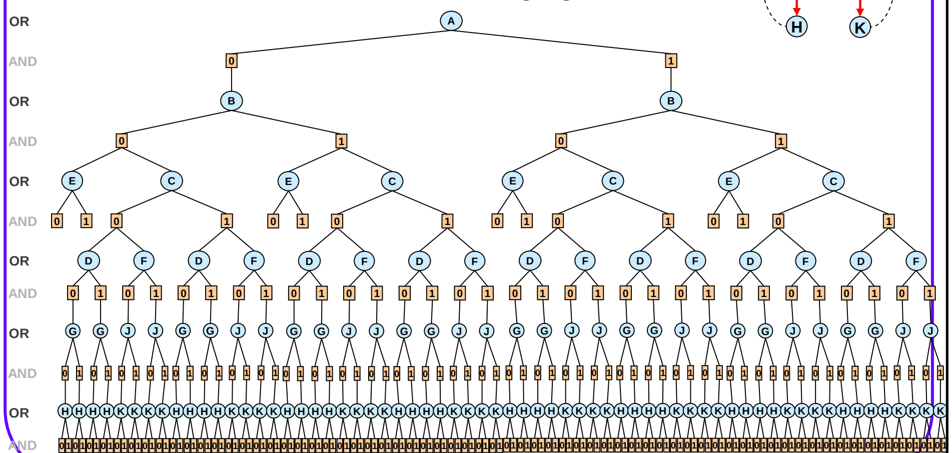
- Any two nodes that root identical subtrees/subgraphs can be merged

- Minimal AND/OR search graph: closure under merge of the AND/OR search tree

  - Inconsistent sub-trees can be pruned too.
  - Some portions can be collapsed or reduced.

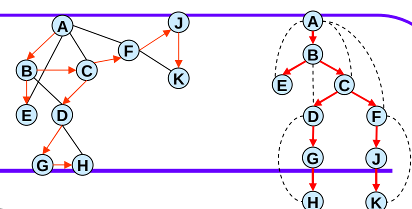
AND/OR Tree

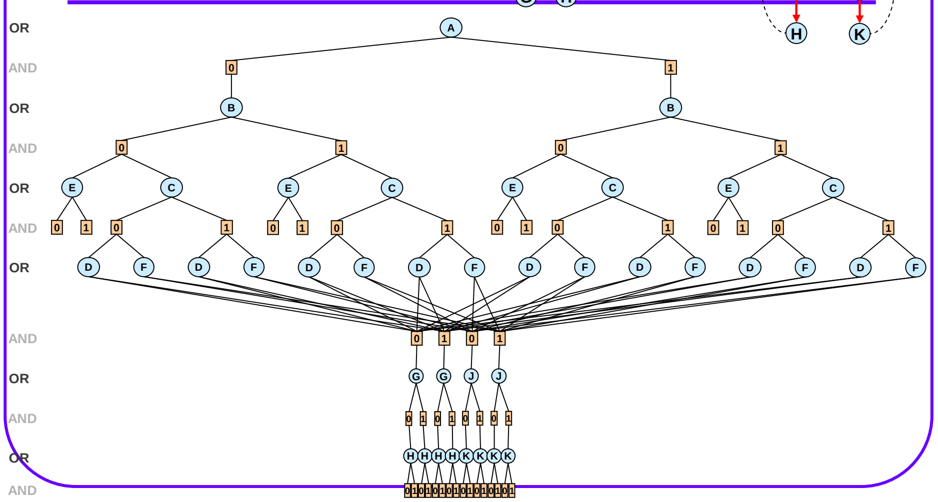




August 2005 Ijcai-05 - Principles 109

An AND/OR Graph: Caching Goods

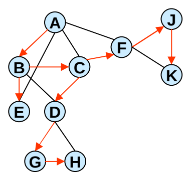


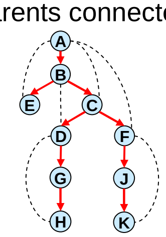


August 2005 Ijcai-05 - Principles 110

Context-based Caching

- Caching is possible when context is the same
- context = parent-separator set in induced pseudo-graph
  = current variable +
  parents connected to subtree below



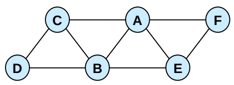


context(B) = {A, B}
context(c) = {A,B,C}
context(D) = {D}
context(F) = {F}

Complexity of AND/OR Graph

- **Theorem:** Traversing the AND/OR search graphis time and space exponential in the induced width/tree-width.
- If applied to the OR graph complexity is time and space exponential in the path-width.

#CSP – AND/OR Search Tree

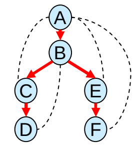


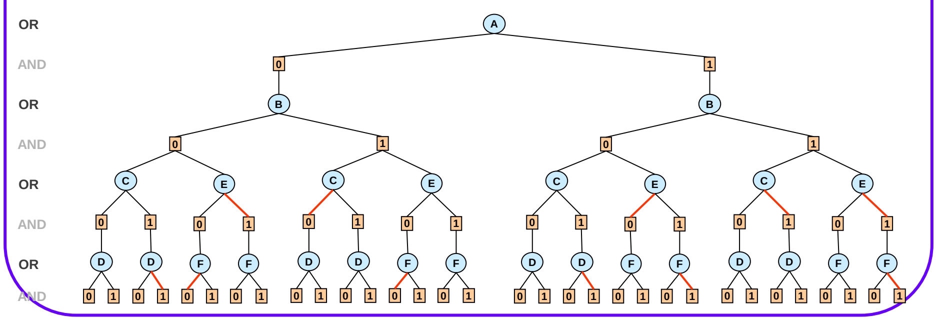
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

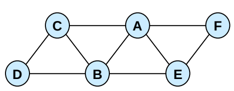
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





#CSP – AND/OR Tree DFS

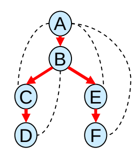


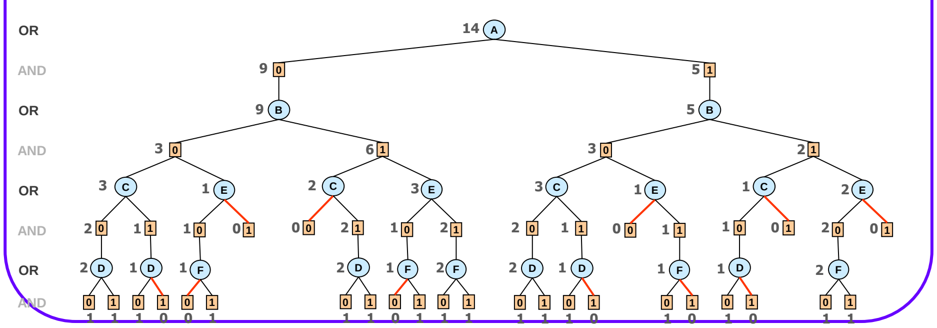
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

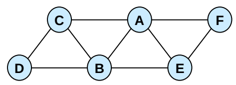
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





#CSP – AND/OR Search Graph (Caching Goods)

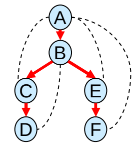


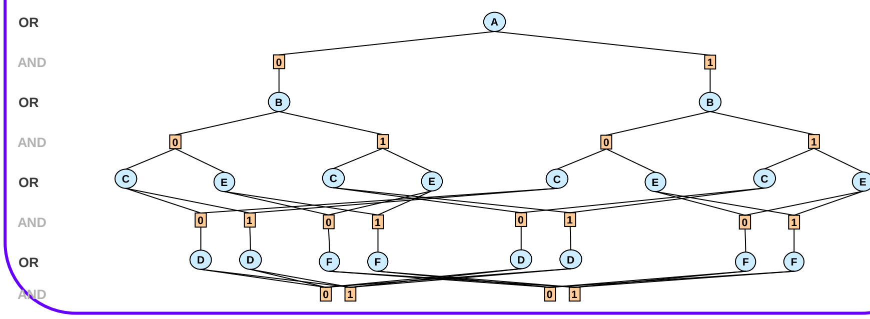
A	B	C	R_{ABC}
0	0	0	**1**
0	0	1	**1**
0	1	0	0
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	0

B	C	D	R_{BCD}
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	0
1	0	0	**1**
1	0	1	0
1	1	0	**1**
1	1	1	**1**

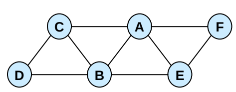
A	B	E	R_{ABE}
0	0	0	**1**
0	0	1	0
0	1	0	**1**
0	1	1	**1**
1	0	0	0
1	0	1	**1**
1	1	0	**1**
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	0





#CSP – AND/OR Search Graph (Caching Goods)

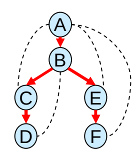


A	B	C	R_{ABC}
0	0	0	**1**
0	0	1	**1**
0	1	0	0
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	0

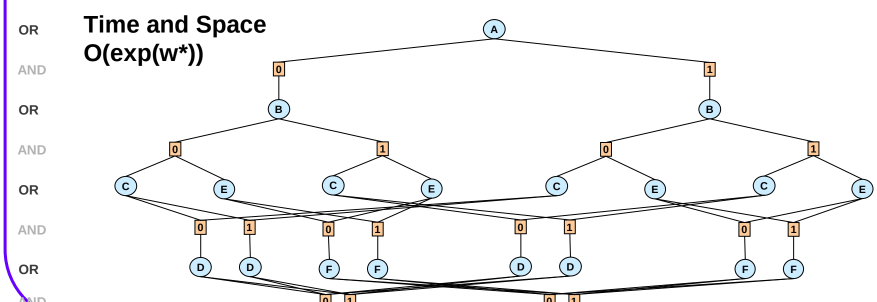
B	C	D	R_{BCD}
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	0
1	0	0	**1**
1	0	1	0
1	1	0	**1**
1	1	1	**1**

A	B	E	R_{ABE}
0	0	0	**1**
0	0	1	0
0	1	0	**1**
0	1	1	**1**
1	0	0	0
1	0	1	**1**
1	1	0	**1**
1	1	1	0

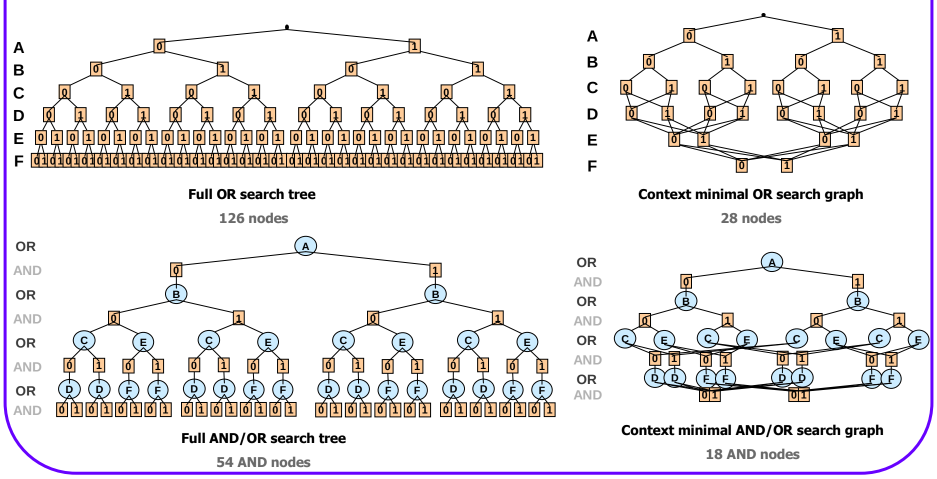
A	E	F	R_{AEF}
0	0	0	0
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	0



Time and Space $O(\exp(w^*))$



All Four Search Spaces



Searching AND/OR Graphs

- AO(i): searches depth-first, cache i-context
  - i = the max size of a cache table (i.e. number of variables in a context)

i=0 → i=w*

Space: O(n)
Time: O(exp(w* log n))

Space: O(exp w*)
Time: O(exp w*)

AO(i) time complexity?

Impact of AND/OR for Constraint Processing

- Minor impact for Constraint-satisfaction
  - Search with backjumping or without backjumping
    - Space: linear, Time: O(exp(logn w*))
  - Search with learning no-goods
    - time and space: O(exp(w*))
  - Variable-elimination
    - time and space: O(exp(w*))
- Counting, enumeration
  - Search with backjumping
    - Space: linear, Time: O(exp(n))
    - Space: linear, Time: O(exp(log n w*))
  - Search with no-goods caching only
    - space: O(exp(w*)) Time: O(exp(n))
    - space: O(exp(w*)) Time: O(exp(log n w*))
  - Search with goods and no-goods learning
    - Time and space: O(exp(path-width), O(exp(log n w*))
    - Time and space: O(exp(tree-width), O(exp(w*))
  - Variable-elimination
    - Time and space: O(exp(w*))

Impact of AND/OR for Any Graphical Model

- Optimization, Belief
  - Search with backjumping
    - **Space:** O(n), **Time:** O(exp(n))
    - Space: linear, Time: O(exp(log n w*))
  - Search with no-goods caching only
    - **space:** O(exp(w*)) **Time:** O(exp(n))
    - space: O(exp(w*)) Time: O(exp(log n w*))
  - Search with goods and no-goods learning
    - **Space:** O(exp(pw*)) **Time:** O(exp(pw*)), pw <=w*logn
    - Time and space: O(exp(w*), O(exp(w*))
  - For optimization can be
    - **Space:** O(exp(c*)) **Time:** O(exp(c*)), c*=best-solution length.
  - Variable-elimination
    - **Space:** O(exp(w*)) **Time:** O(exp(w*))

Algorithms for AND/OR Space

- **Backjumping** for CSPs (Gaschnig 1977, Dechter 1990, Prosser, Bayardo et. Al, 1994.)
- **Pseudo-search rearangement**, for any CSP task (Freuder 1987)
- **Recursive Conditioning** (Darwiche, 1999), explores the AND/OR tree or graph for any query
- **Searching tree-decomspositions** for optimization: (Jeagou, 2000)
- **Valued-elimination** (Bacchus, 2003)
- **Variable-elimination** (next session, pure inference)

Problem Solving by Inference

A. Darwiche

Basic Principles

- Inference by Variable Elimination
- Inference by Factor Elimination (Tree-clustering, Jointree)
- Inference by Recursive Conditioning (Decomposition)

A. Darwiche

The Inference Problem

- Answer queries about a function which is given in factored form.

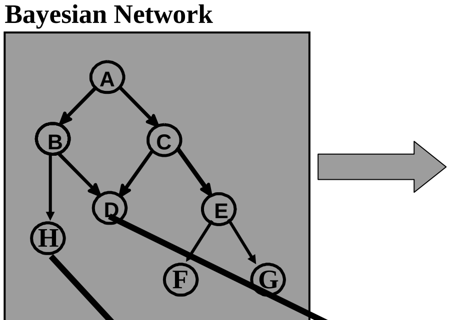
- Logic/Constraints: Boolean functions
- Probability: Probability functions

- Techniques apply to other types of inference: belief functions, penalty logics, etc.

A. Darwiche

Bayesian Networks

Bayesian Network



B	H	Pr(H|B)
t	t	.4
t	f	.6
.	.	.

B	C	D	Pr(D|BC)
t	t	t	.9
t	t	f	.1
.	.	.	
f	f	f	.3

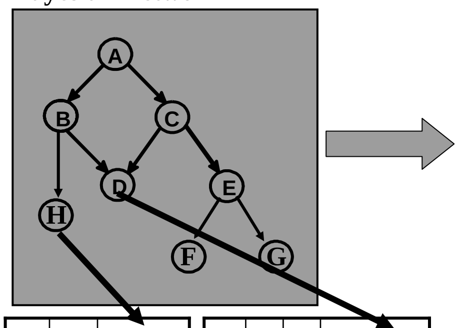
Probability Function

A	B	C	D	E	F	G	H	Pr(.)
t	t	t	t	t	t	t	t	.004
t	t	t	t	t	t	t	f	.001
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
f	f	f	f	f	f	f	f	.020

A. Darwiche

Bayesian Networks

Bayesian Network



B	H	Pr(H|B)
t	t	.4
t	f	.6
.	.	.

B	C	D	Pr(D|BC)
t	t	t	.9
t	t	f	.1
.	.	.	
f	f	f	.3

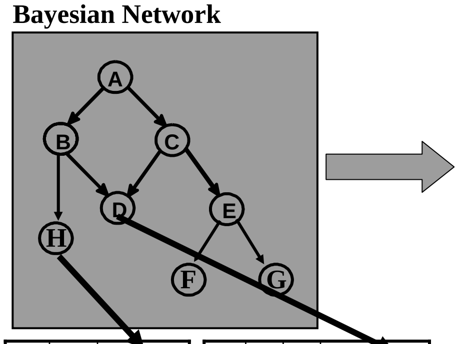
Probability Function

A	B	C	D	E	F	G	H	Pr(.)
t	t	t	t	t	t	t	t	.004
t	t	t	t	t	t	t	f	.001
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
f	f	f	f	f	f	f	f	.020

A. Darwiche

Bayesian Networks

Bayesian Network



B	H	Pr(H|B)
t	t	.4
t	f	.6
.	.	.

B	C	D	Pr(D|BC)
t	t	t	.9
t	t	f	.1
.	.	.	
f	f	f	.3

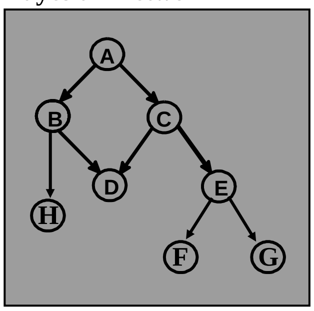
Probability Function

A	B	C	D	E	F	G	H	Pr(.)
t	t	t	t	t	t	t	t	.004
t	t	t	t	t	t	t	f	.001
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
f	f	f	f	f	f	f	f	.020

A. Darwiche

Probabilistic Inference

Bayesian Network



Probability Function

A	B	C	D	E	F	G	H	Pr(.)
t	t	t	t	t	t	t	t	.004
t	t	t	t	t	t	t	f	.001
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
f	f	f	f	f	f	f	f	.020

Pr(A=t or H=f)

MPE: row r with max Pr(r)

MAP: partial instantiation i with max Pr(i)

A. Darwiche

Propositional Knowledge Bases

Propositional KB

```
 A & ok_X => ¬B
¬A & ok_X =>  B

 B & ok_Y => ¬C
¬B & ok_Y =>  C
```

OK_Y	B	C	F(.)
T	T	T	1
T	F	F	0
.	.	.	.
F	F	F	1

Boolean Function

A	B	C	OK_X	OK_Y	F(…)
T	T	T	T	T	0
T	T	T	T	F	0
…	…	...	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…
F	F	F	F	F	1

A. Darwiche

Propositional Knowledge Bases

Propositional KB

```
 A & ok_X => ¬B
¬A & ok_X =>  B

 B & ok_Y => ¬C
¬B & ok_Y =>  C
```

Boolean Function

A	B	C	OK_X	OK_Y	F(…)
T	T	T	T	T	1
T	T	T	T	F	0
…	…	...	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…
F	F	F	F	F	0

Is there a satisfying assignment?

How many satisfying assignments

Are two KBs equivalent?

A. Darwiche

Constraint Satisfaction

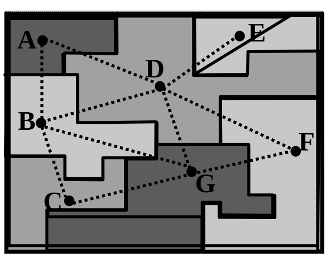
Example: map coloring

Variables - countries (A,B,C,etc.)

Values - colors (e.g., red, green, yellow)

Constraints:

A	B	F(AB)
red	red	0
red	green	1
…	…	…
…	…	…



A	B	C	…	F()
red	red	red	red	0
red	red	red	green	0
…	…	...	…	0
…	…	…	…	1
…	…	…	…	1
…	…	…	yellow	0

A. Darwiche

Constraint Satisfaction

Example: map coloring
Variables - countries (A,B,C,etc.)
Values - colors (e.g., red, green, yellow)
Constraints:

A	B	C	…	F()
red	red	red	red	0
red	red	red	green	1
…	…	...	…	0
…	…	…	…	1
…	…	…	…	1
…	…	…	yellow	0

Are the constraints consistent?

Find a solution, find all solutions

Count all solutions

A. Darwiche

Inference by Variable Elimination

A. Darwiche

Basic Principle

Reduce: A query about a function

$$f = f_1 f_2 \ldots f_m$$

over n variables into a query about a function f' over n-1 variables.

Eliminate a variable X from f, while keeping the result, f', as factored as possible

A. Darwiche

Basic Principle

If variable X appears in only one factor f_1 of f, all we have to do is replace f_1 with $elm(f_1,X)$:

$$f' = elm(f_1,X)\, f_2 \ldots f_m$$

If variable X appears in more than one factor, say, f_1 and f_2, we must combine (multiply) them first before eliminating X:

$$f' = elm(f_1 f_2,X)\, f_3 \ldots f_m$$

Notes:

The more factors we have to combine, the less factored the result is.

The order in which we eliminate variables matters only computationally.

A. Darwiche

Elimination in Probability

A	B	F(AB)
T	T	.1
T	F	.2
F	T	.2
F	F	.5

Eliminate B

Project on A

A	F(A)
T	.3
F	.7

A. Darwiche

Elimination in Probability

A	B	F(AB)
T	T	.1
T	F	.2
F	T	.2
F	F	.5

Eliminate B

Project on A

A	F(A)
T	.3
F	.7

A. Darwiche

Elimination in Probability

A	B	F(AB)
T	T	.1
T	F	.2
F	T	.2
F	F	.5

Eliminate B $\longrightarrow$ Project on A

A	F(A)
T	.3
F	.7

$$F(A) = \sum_B F(AB)$$

A. Darwiche

Elimination in Probability

A	B	F(AB)
T	T	.1
T	F	.2
F	T	.2
F	F	.5

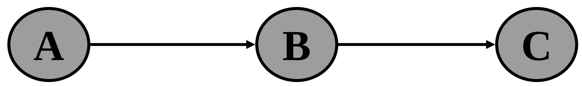
Eliminate B $\longrightarrow$ Project on A

A	F(A)
T	.3
F	.7

Different notions of elimination: <u>sum</u> out, <u>max</u> out
Each preserves ability to answer a different type of query
Sum out: compute probabilities   Max out: compute MPE
Sum then Max: compute MAP

A. Darwiche

Variable Elimination Example



A	$f_1(A)$
T	.6
F	.4

A	B	$f_2(AB)$
T	T	.9
T	F	.1
F	T	.2
F	F	.8

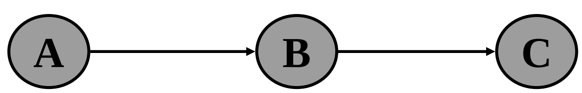
B	C	$f_3(BC)$
T	T	.3
T	F	.7
F	T	.5
F	F	.5

Multiply factors

To eliminate A

A. Darwiche

Variable Elimination Example



A	B	$f_4(AB)$
T	T	.54
T	F	.06
F	T	.08
F	F	.32

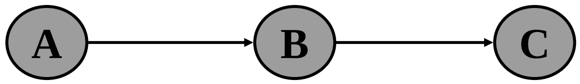
B	C	$f_3(BC)$
T	T	.3
T	F	.7
F	T	.5
F	F	.5

Eliminate A

To eliminate A

A. Darwiche

Variable Elimination Example



B	$f_5(B)$
T	.62
F	.38

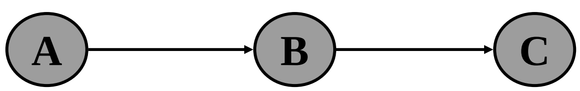
B	C	$f_3(BC)$
T	T	.3
T	F	.7
F	T	.5
F	F	.5

Multiply factors

To eliminate B

A. Darwiche

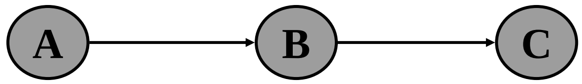
Variable Elimination Example



B	C	$f_6(BC)$
T	T	.186
T	F	.434
F	T	.190
F	F	.190

A. Darwiche

Variable Elimination Example

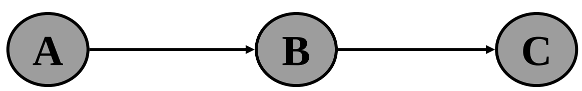


B	C	$f_6(BC)$
T	T	.186
T	F	.434
F	T	.190
F	F	.190

Eliminate B

A. Darwiche

Variable Elimination Example



C	$f_7(C)$
T	.376
F	.624

A. Darwiche

Elimination in Logic

A	B	C	F(ABC)
T	T	T	1
T	T	F	0
T	F	T	0
T	F	F	0
F	T	T	1
F	T	F	0
F	F	T	1
F	F	F	1

A => B, B => C

Eliminate B

Project on A C

A	C	F(AC)
T	T	1
T	F	0
F	T	1
F	F	1

A => C

A. Darwiche

Elimination in Logic

A	B	C	F(ABC)
T	T	T	1
T	T	F	0
T	F	T	0
T	F	F	0
F	T	T	1
F	T	F	0
F	F	T	1
F	F	F	1

A => B, B => C

Eliminate B

Project on A C

A	C	F(AC)
T	T	1
T	F	0
F	T	1
F	F	1

A => C

A. Darwiche

Elimination in Logic

A	B	C	F(ABC)
T	T	T	1
T	T	F	0
T	F	T	0
T	F	F	0
F	T	T	1
F	T	F	0
F	F	T	1
F	F	F	1

A => B, B => C

Eliminate B

Project on A C

A	C	F(AC)
T	T	1
T	F	0
F	T	1
F	F	1

A => C

A. Darwiche

Elimination in Logic

Δ

~A or B
~B or C

Resolve on B

~A or B
~B or C
~A or C

Throw out B clauses

A => B, B => C

Eliminate B

Project on A C

$\exists B.\Delta$

~A or C

A => C

A. Darwiche

Elimination in Logic

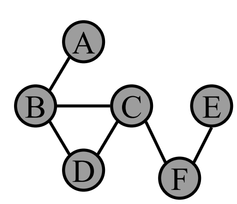
- Different notions of elimination:
  - Existential elimination (quantification)
  - Universal elimination (quantification)
- Preserve ability to answer different queries
  - E.g. Existential preserves SAT
- Different types of eliminations can be mixed to solve more sophisticated problems: diagnosis and planning

A. Darwiche

Interaction Graphs

- **Can simulate variable elimination on connectivity graph:**
  - **Node for every variable**
  - **Edge between two nodes iff appear in same factor**

f(ABCDEF) =
f_1(AB) f_2(BCD) f_3(CF) f_4(EF)

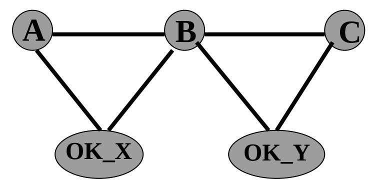


A. Darwiche

Interaction Graph: Example

```
 A & ok_X => ¬B
¬A & ok_X =>  B

 B & ok_Y => ¬C
¬B & ok_Y =>  C
```



Interaction Graph

Also known as <u>primal graph</u> in constraint satisfaction
Other types of graphs can be defined: <u>dual</u> & <u>hyper</u>

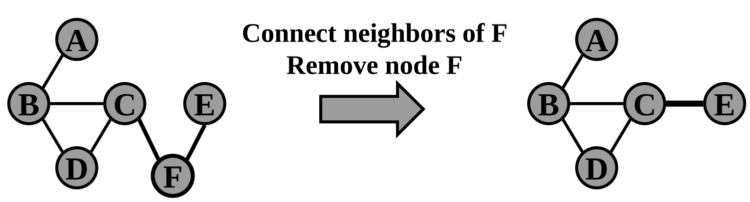
A. Darwiche

Interaction Graphs

- **Eliminate F:**

$$f(ABCDEF) = f_1(AB)\, f_2(BCD)\, \boxed{f_3(CF)\, f_4(EF)}$$

$$f'(ABCDE) = f_1(AB)\, f_2(BCD)\, \boxed{f_{34}(CE)}$$



Interaction graph for f **Interaction graph for f'**

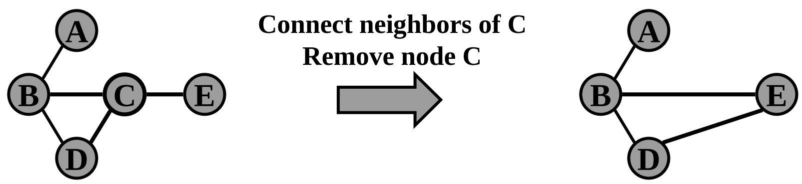
A. Darwiche

Interaction Graphs

- Eliminate C:

$$f'(ABCDEF) = f_1(AB)\boxed{f_2(BCD)\, f_{34}(CE)}$$

$$f''(ABCDEF) = f_1(AB)\boxed{f_{234}(BDE)}$$



Interaction graph for f'

Interaction graph for f''

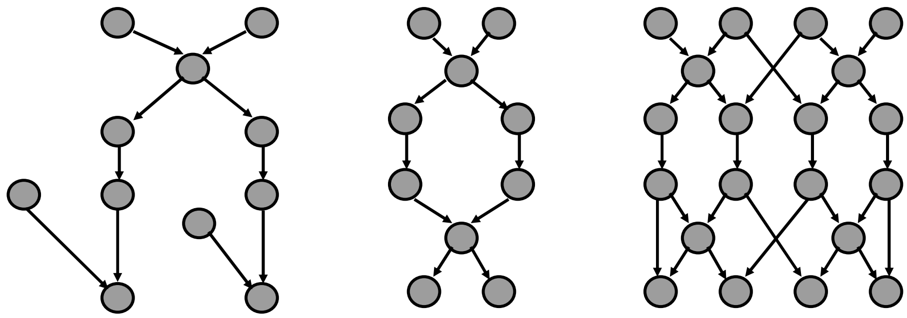
A. Darwiche

Induced Width / Treewidth

- **The factor constructed upon eliminating variable X will be over X's neighbors in interaction graph.**
- **If w is largest number of neighbors encountered when eliminating variables according to order O, then w is called the <u>induced width</u> of elimination order O.**
- **The complexity of variable elimination is O(n exp(w)), where n is number of variables**
- **The <u>induced width</u> of a graph is width of its best elimination order**
- **This is also known as <u>treewidth</u> in graph theoretic literature**

A. Darwiche

Treewidth



Higher treewidth

A. Darwiche

Bucket Elimination

A mechanism for managing variable elimination

$f(ABCDEF) = f_1(AB)\, f_2(BCD)\, f_3(CF)\, f_4(EF)$

Var	Factors
A	$f_1(AB)$
B	$f_2(BCD)$
C	$f_3(CF)$
D	
E	$f_4(EF)$
F	

Elm A

Var	Factors
A	
B	$f_2(BCD)$ $f_5(B)$
C	$f_3(CF)$
D	
E	$f_4(EF)$
F	

A. Darwiche

Bucket Elimination

A mechanism for managing variable elimination

$f(ABCDEF) = f_1(AB)\, f_2(BCD)\, f_3(CF)\, f_4(EF)$

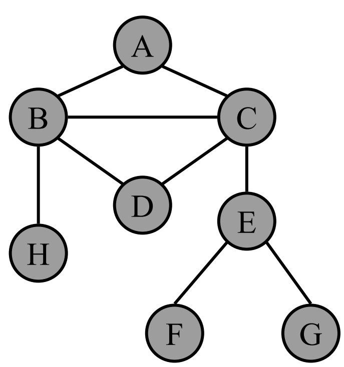
Var	Factors
A	
B	$f_2(BCD)$ $f_5(B)$
C	$f_3(CF)$
D	
E	$f_4(EF)$
F	

Elm B

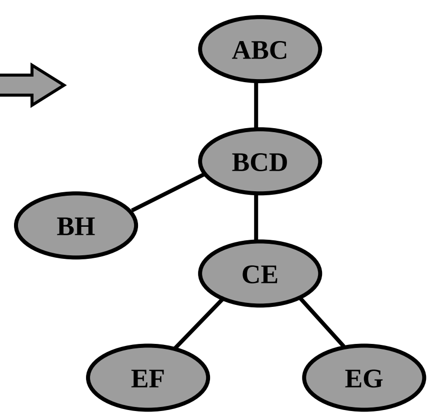
Var	Factors
A	
B	
C	$f_3(CF)$ $f_6(CD)$
D	
E	$f_4(EF)$
F	

Another View of Treewidth

Interaction graph



Tree decomposition



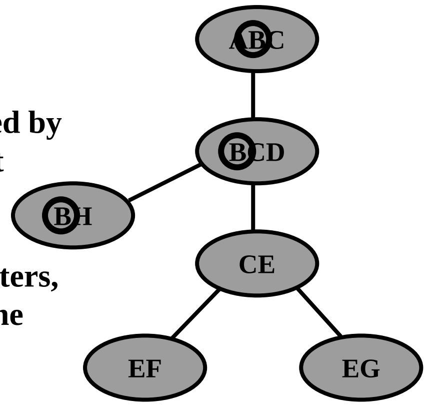
Tree Decomposition

• A tree of clusters

• Every pair of variables connected by an edge in interaction graph must appear together in some cluster

• If a variable appears in two clusters, it must appear in all clusters on the path between them

Tree decomposition



A. Darwiche

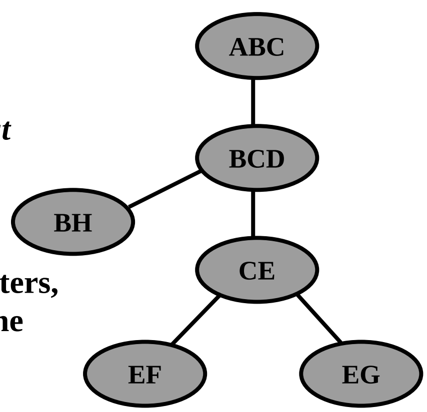
Tree Decomposition

• A tree of clusters

• *The variables of every factor must appear in some cluster*

• If a variable appears in two clusters, it must appear in all clusters on the path between them

Tree decomposition



A. Darwiche

Tree Decomposition

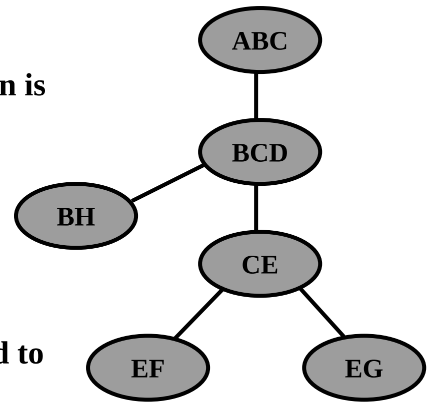
• Width of tree decomposition is size of its largest cluster - 1

• Width of best tree decomposition is treewidth of connectivity graph

• Treewidth quantifies the resemblance of a graph to a tree structure

• Tree decompositions correspond to jointrees (junction trees)

Tree decomposition



A. Darwiche

Inference by Factor Elimination

(Jointree algorithm)
(Tree clustering algorithm)

A. Darwiche

Basic Principle

Reduce: A query about a function
$f = f_1 f_2 \ldots f_m$
into a query about a function f' over m-1 factors.

Eliminate factor f_i while keeping result, f', as factored as possible

Allows n queries to be answered in O(n exp(w)) time instead of O(n^2 exp(w)) [Standard VE]

A. Darwiche

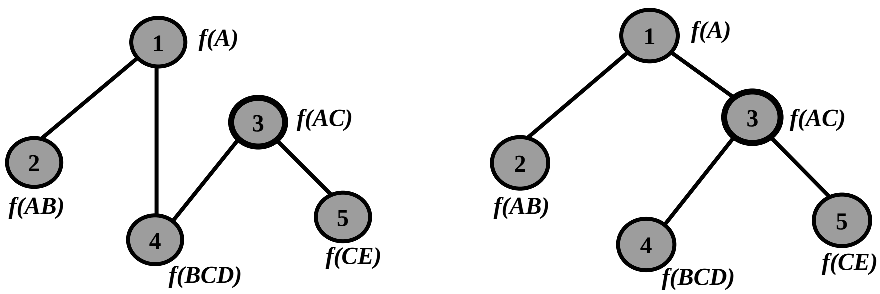
Basic Principles

- **To eliminate a factor f_1 :**
  - **Eliminate all variables appearing in f_1 but not in other factors $f_2 \ldots f_m$**
  - **Multiply resulting factor into one of $f_2 \ldots f_m$**
- **To control elimination:**
  - **Choose a factor to eliminate**
  - **Decide on which factor to multiply into**
- **Factor elimination is controlled by an elimination tree (instead of elimination order)**

A. Darwiche

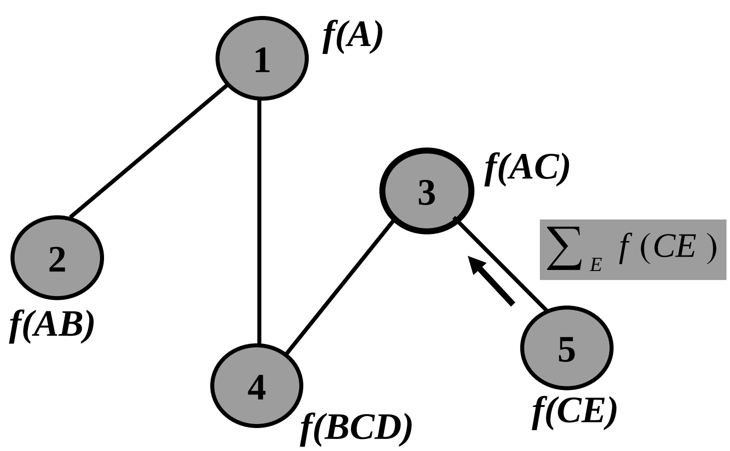
Elimination Trees

- **Spanning tree of factors**
- **Any spanning tree will do, some lead to more work than others**



A. Darwiche

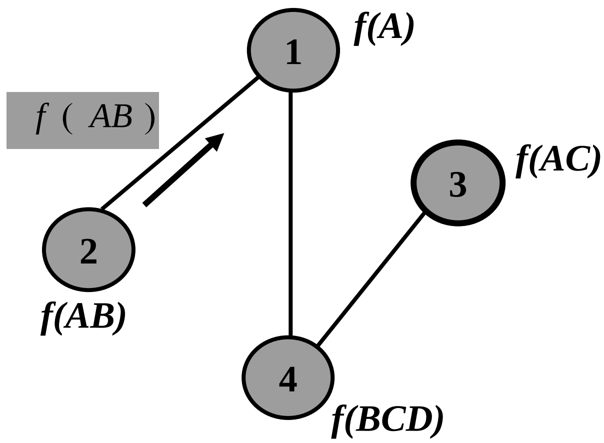
Factor Elimination



Eliminate factor f_5

A. Darwiche

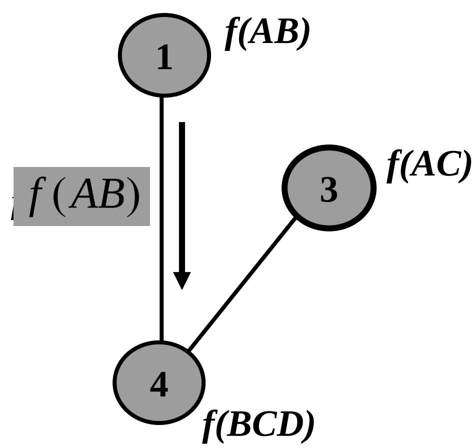
Factor Elimination



Eliminate factor f_2

A. Darwiche

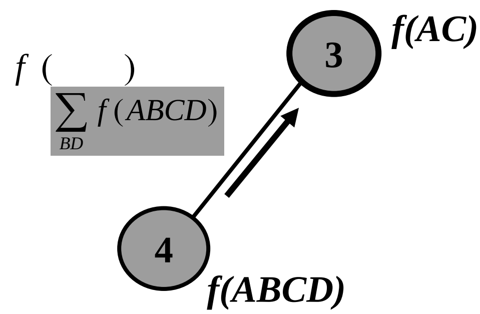
Factor Elimination



Eliminate factor f_1

A. Darwiche

Factor Elimination

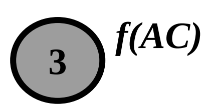


Eliminate factor f_4

A. Darwiche

Factor Elimination

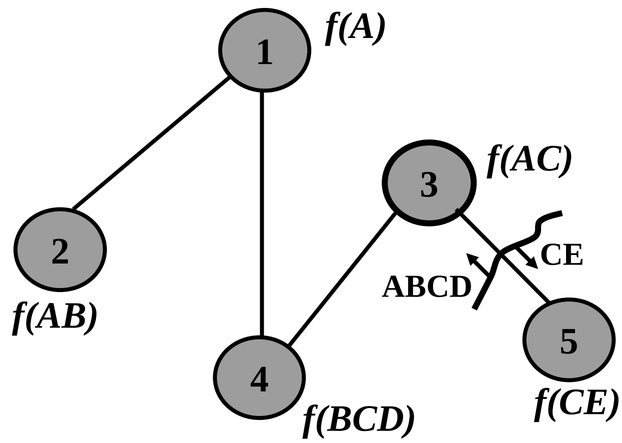
Resulting factor represents projection of original factors on variables AC



$$
\begin{aligned}
& f(AC) \\
&= \sum_{BDE} f(ABCDE) \\
&= \sum_{BDE} f(AB)\, f(A)\, f(AC)\, f(BCF)\, f(CE)
\end{aligned}
$$

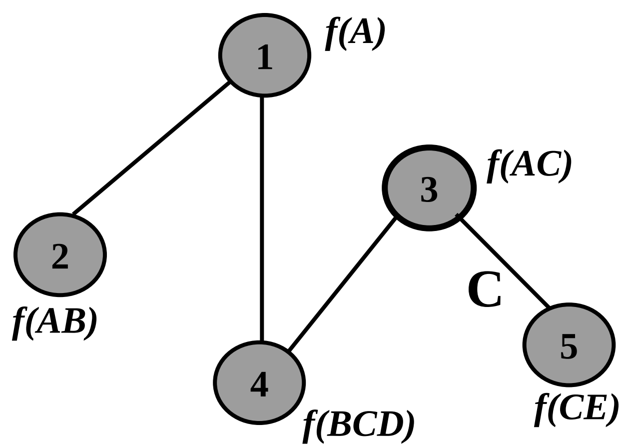
A. Darwiche

Separators



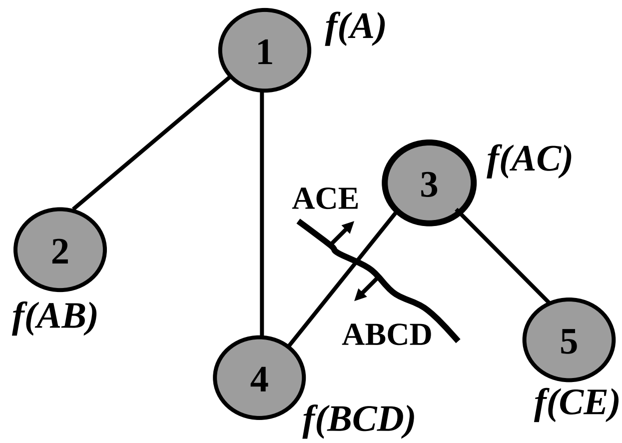
A. Darwiche

Separators



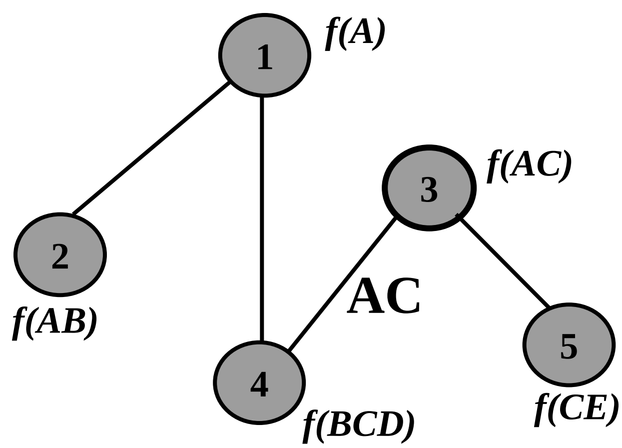
A. Darwiche

Separators



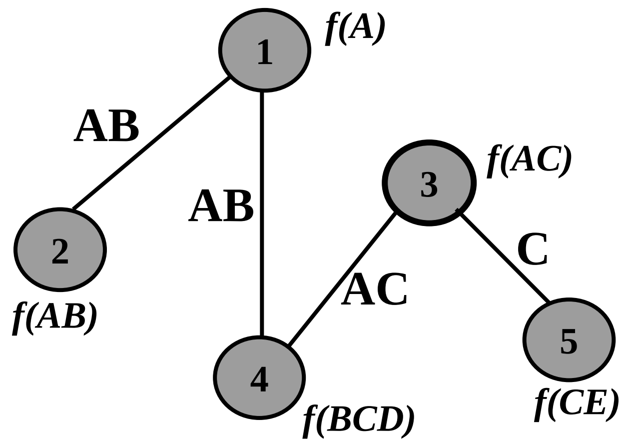
A. Darwiche

Separators



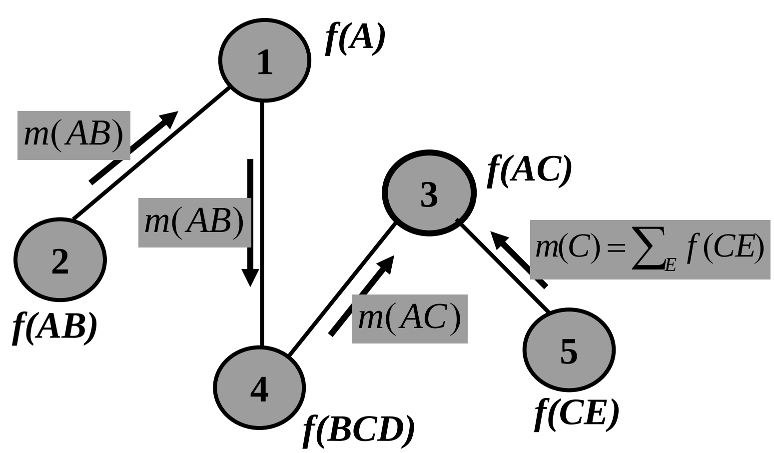
A. Darwiche

Separators



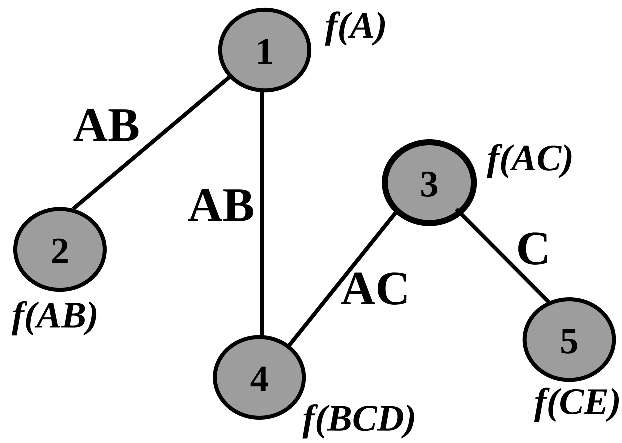
A. Darwiche

Messages over Separators



A. Darwiche

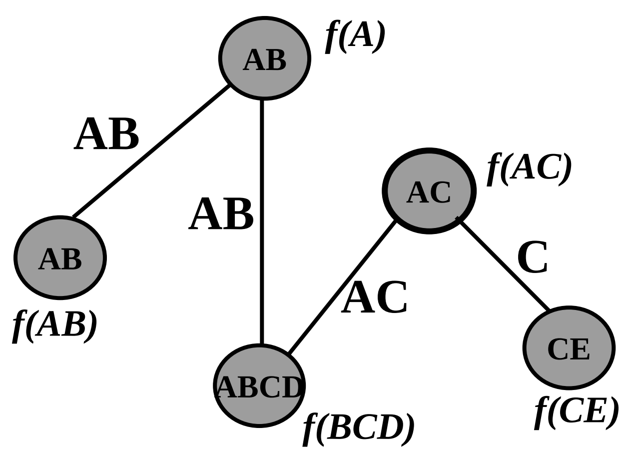
Clusters



Cluster of node: variables of its factor union neighboring separators

A. Darwiche

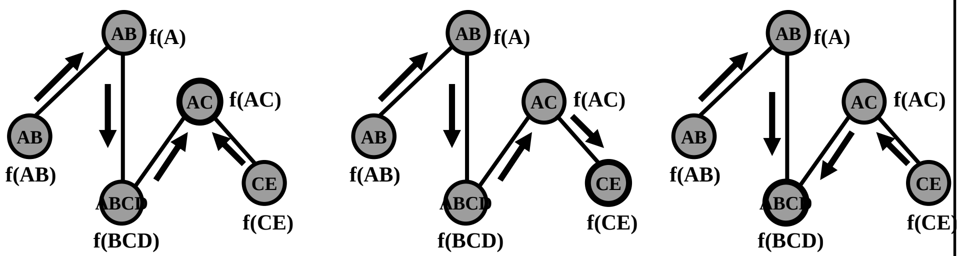
Clusters



Cluster of node: variables of its factor union neighboring separators
Width of elm tree: size of largest cluster -1

A. Darwiche

Complexity



Project on AC **Project on CE** **Project on ABCD**

-To project on variables of a cluster, choose cluster as root
-Number of messages passed is twice number of edges
-Sending a message from a cluster is exponential in size of cluster
-With appropriate elim tree, all messages passed in O(n exp(w))

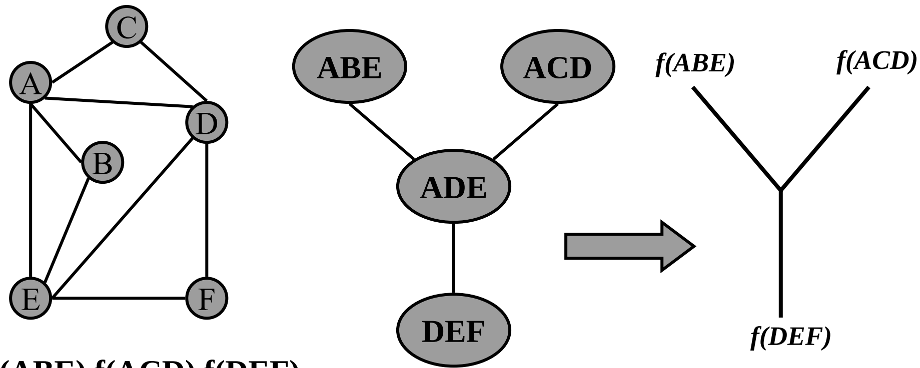
A. Darwiche

Tree Decomposition ⇨Elm Tree

Every tree decomposition of width w
embeds
an elimination tree of width w

A. Darwiche

Tree Decomposition ⇨Elm Tree



f(ABE) f(ACD) f(DEF)
Interaction graph

Tree decomposition

Elimination tree

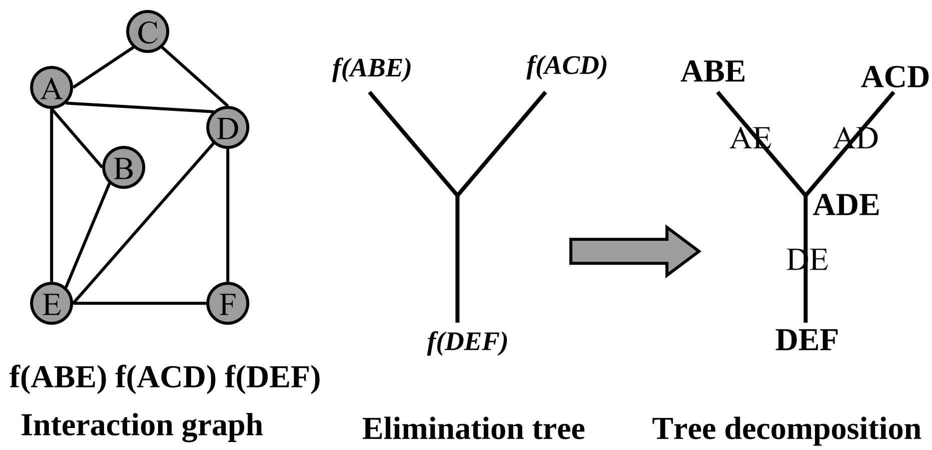
A. Darwiche

Elm Tree ⇨Tree Decomposition

Every elimination tree of width w
induces
a tree decomposition of width w

A. Darwiche

Elm Tree ⇨Tree Decomposition



A. Darwiche

Inference by Recursive Conditioning

A. Darwiche

Basic Principle

Reduce: A query about a function

$$f = f_1\, f_2 \ldots f_m$$

into queries about a decomposition

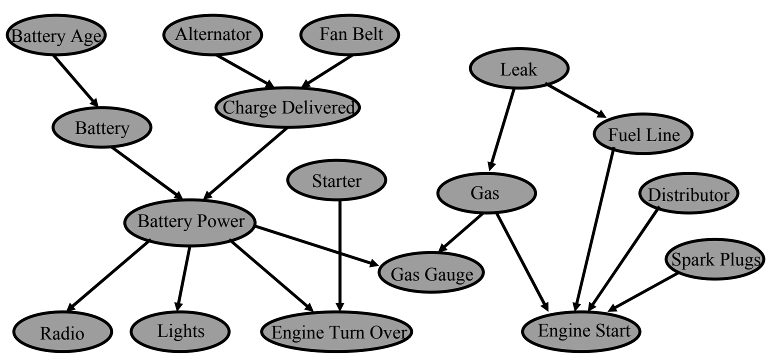
$$f_L = f_1 \ldots f_i \qquad f_R = f_{i+1} \ldots. f_m$$

Must condition on variables shared by f_L and f_R

Allows inference in O(n exp(w)) time
Facilitates time-space tradeoffs

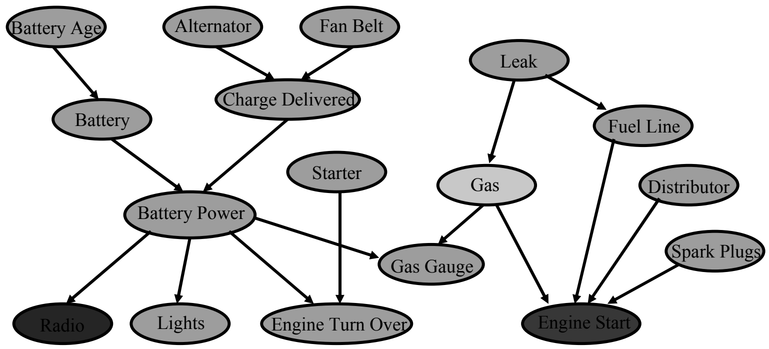
A. Darwiche

Decomposition



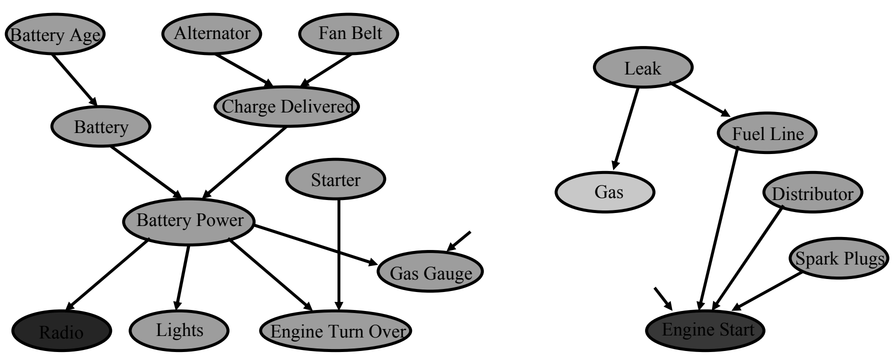
A. Darwiche

Decomposition



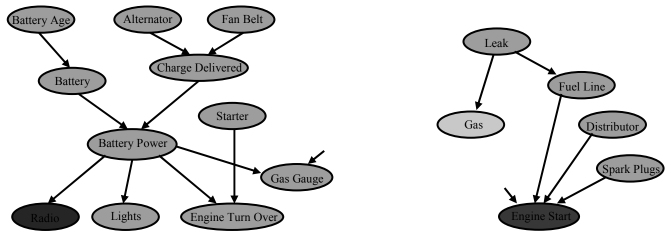
A. Darwiche

Decomposition



A. Darwiche

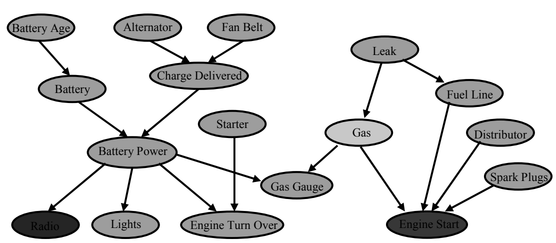
Decomposition



LP * RP

A. Darwiche

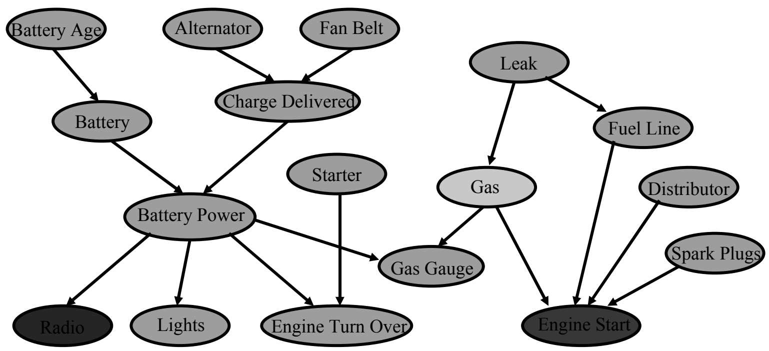
Decomposition



LP * RP

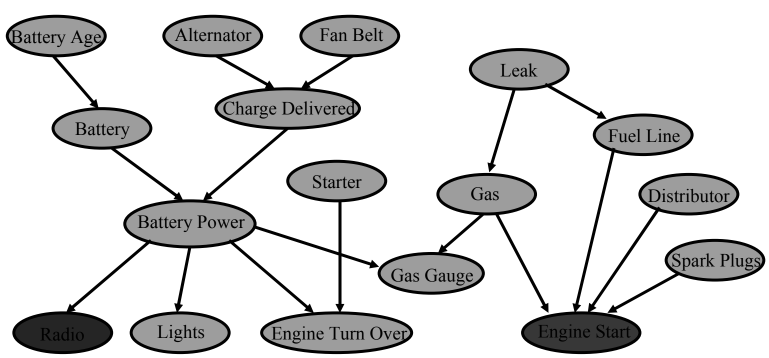
A. Darwiche

Causal Network



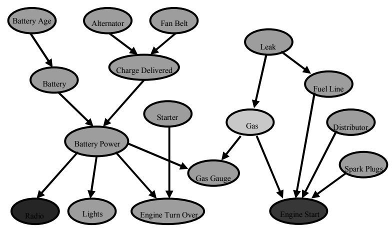
A. Darwiche

Causal Network

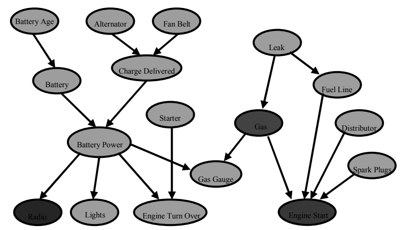


A. Darwiche

Case Analysis



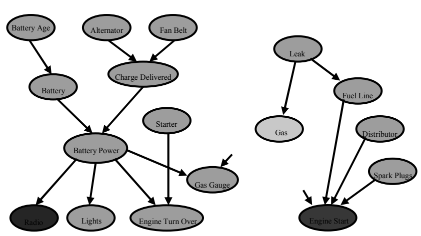
Case I



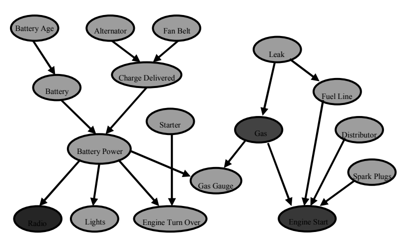
Case II

A. Darwiche

Case Analysis



Case I

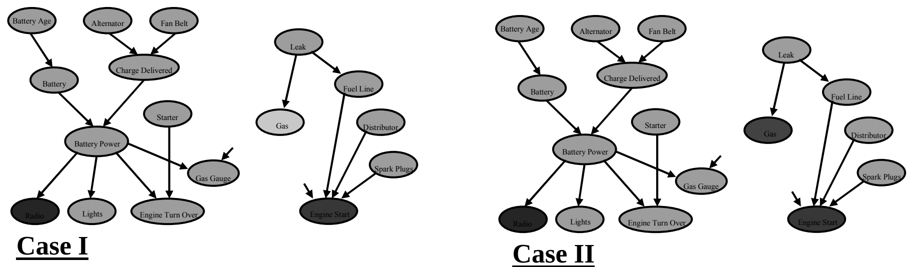


Case II

LP * RP

A. Darwiche

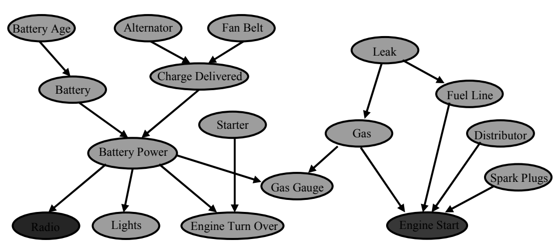
Case Analysis



LP * RP + LP * RP

A. Darwiche

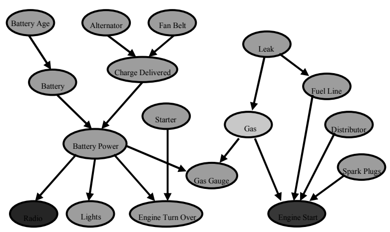
Case Analysis

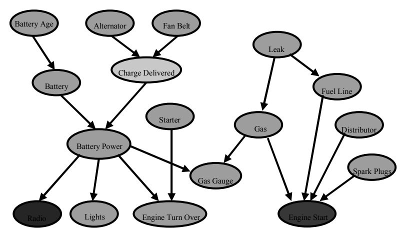


LP * RP + LP * RP

A. Darwiche

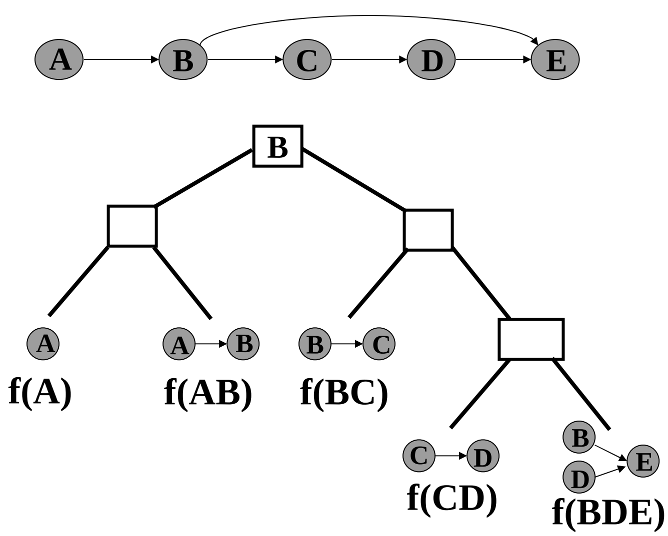
- **Decomposition and Case Analysis can answer any query**
- **Non-Deterministic!**





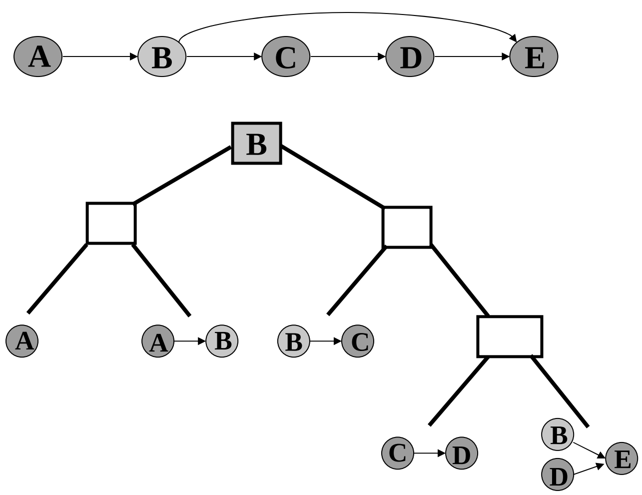
A. Darwiche

Decomposition Tree



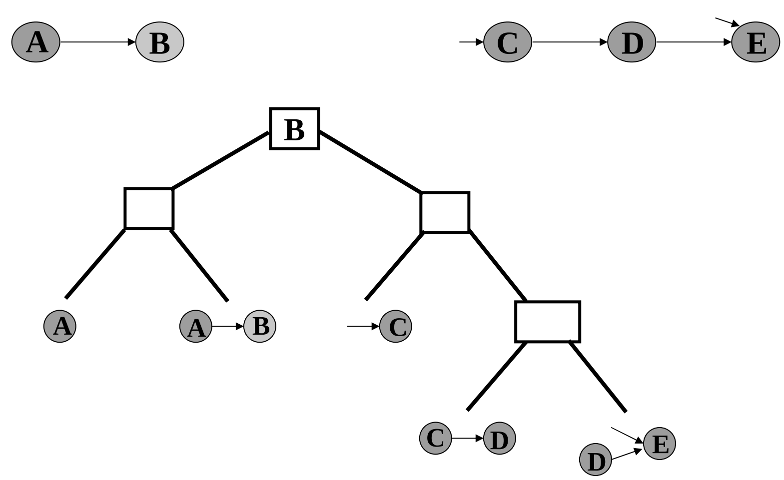
A. Darwiche

Decomposition Tree



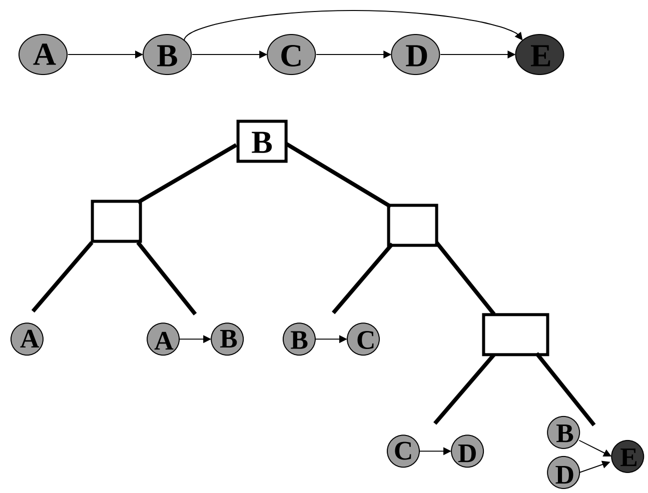
A. Darwiche

Decomposition Tree



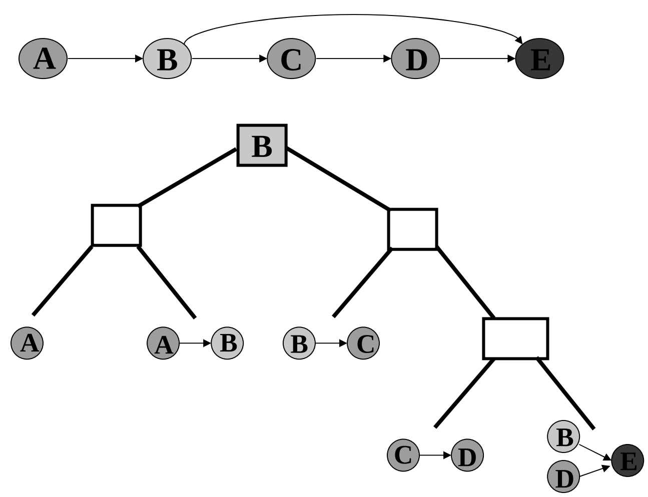
A. Darwiche

Decomposition Tree



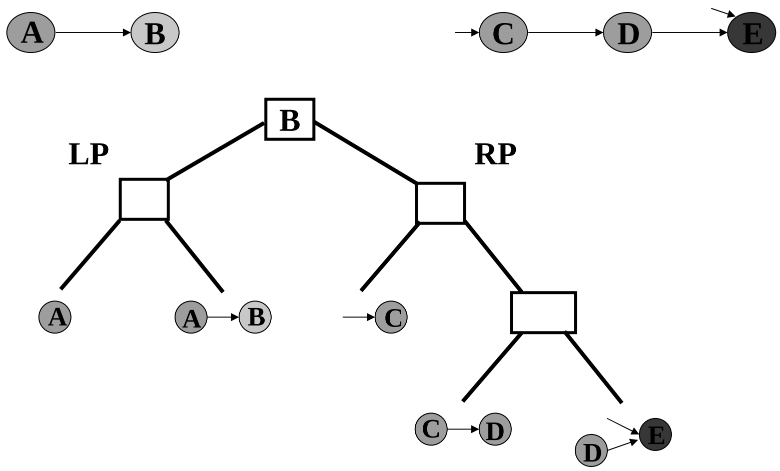
A. Darwiche

Decomposition Tree



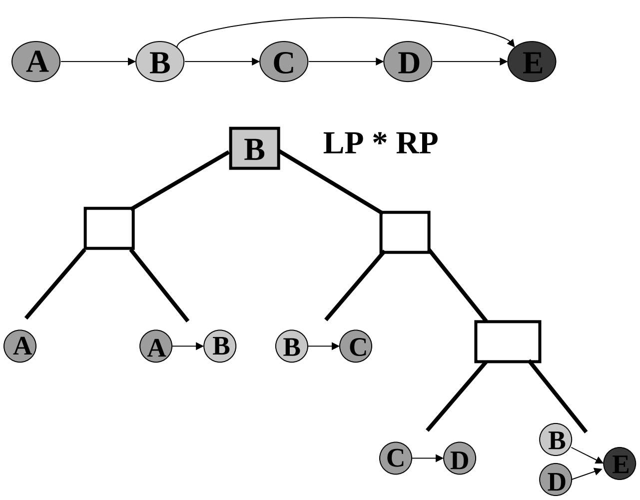
A. Darwiche

Decomposition Tree



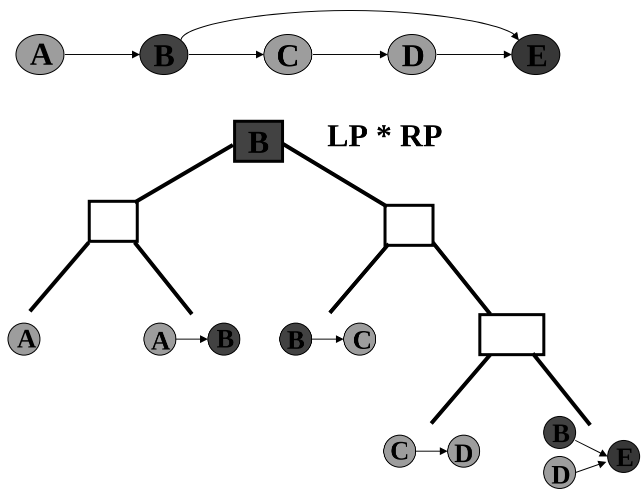
A. Darwiche

Decomposition Tree



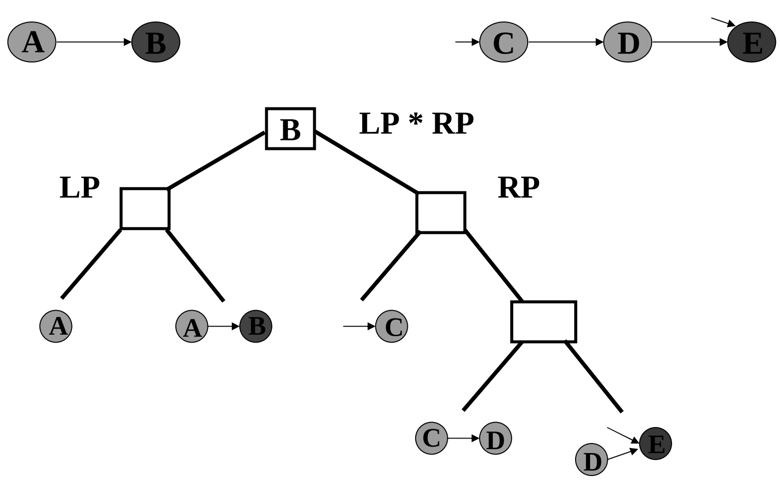
A. Darwiche

Decomposition Tree



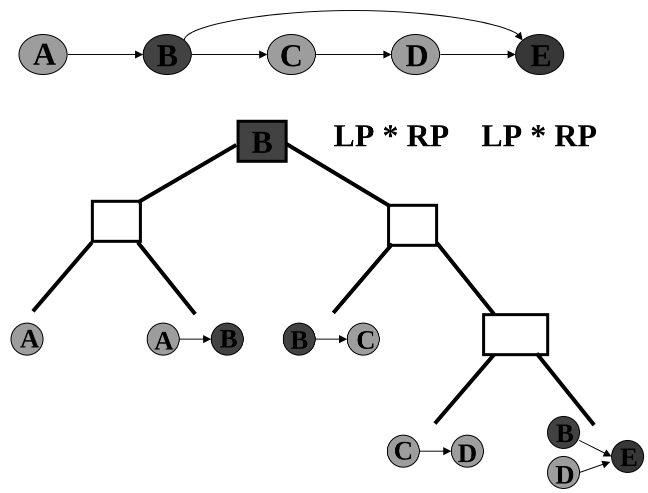
A. Darwiche

Decomposition Tree



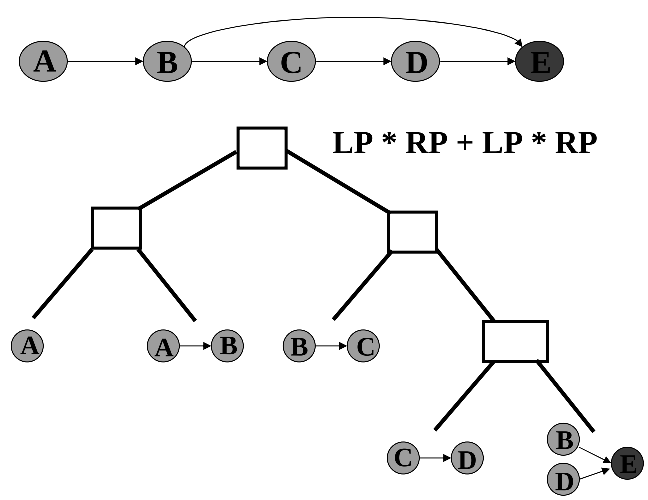
A. Darwiche

Decomposition Tree



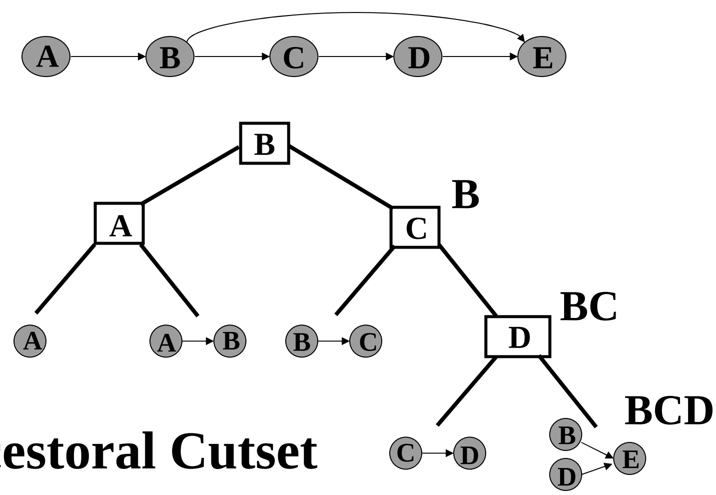
A. Darwiche

Decomposition Tree



A. Darwiche

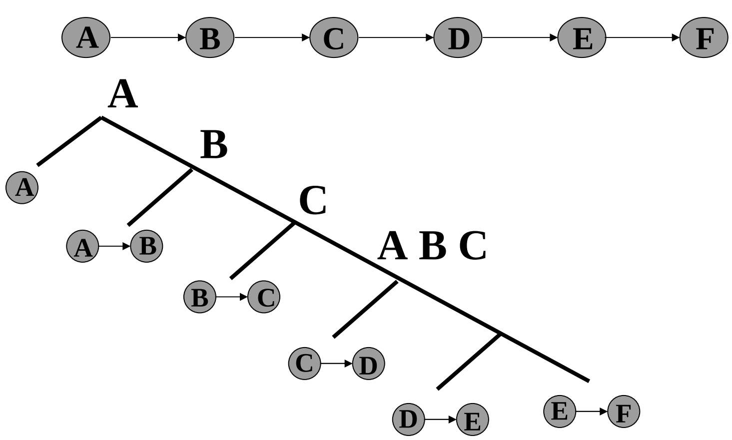
Computational Complexity



Ancestoral Cutset

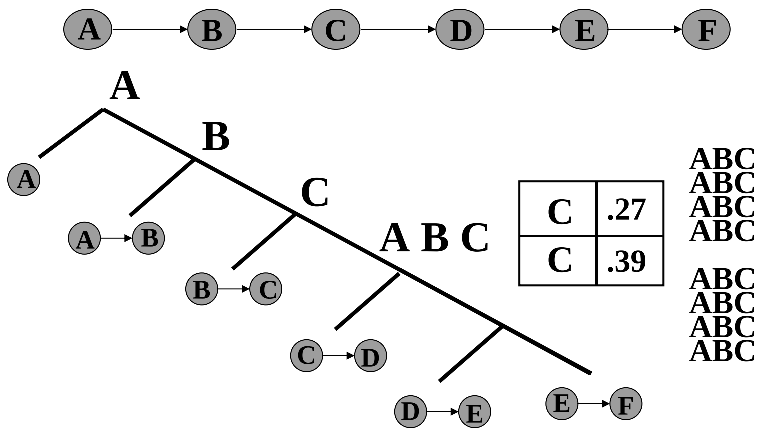
A. Darwiche

Decomposition Tree



A. Darwiche

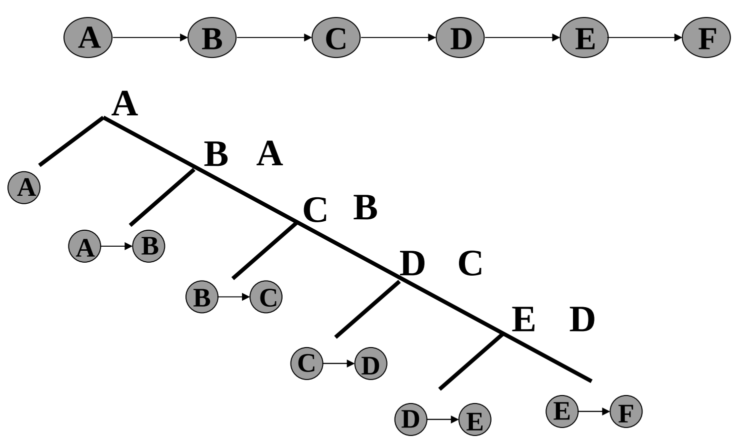
Decomposition Tree



Context(N)= A-Cutset(N)&Vars(N)

A. Darwiche

Decomposition Tree

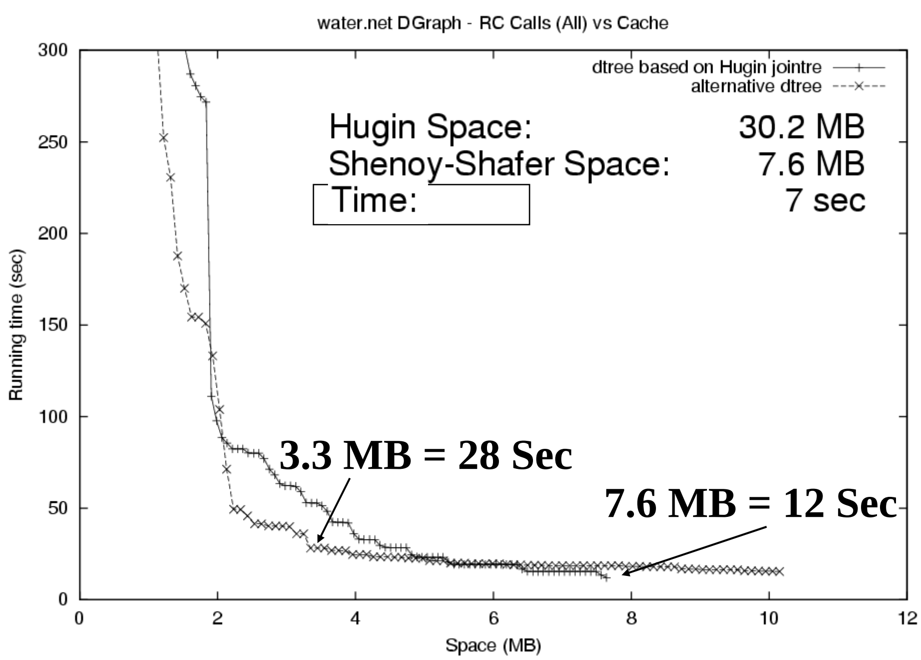


A. Darwiche

Computational Complexity

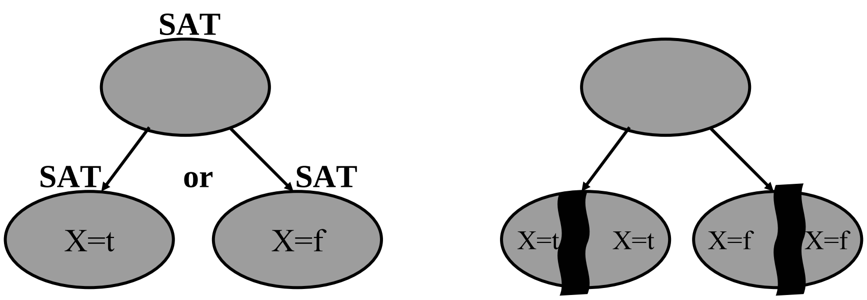
- Given
  - DAG with **n** nodes
  - elimination order of width **w**
- Can construct a dtree, such that
  - time complexity: **O(n exp(w logn))**
    space complexity **O(n)** [no caching]
- Can construct a dtree, such that
  - time complexity: **O(n exp(w))**
    space complexity: **O(n exp(w))** [full caching]

A. Darwiche



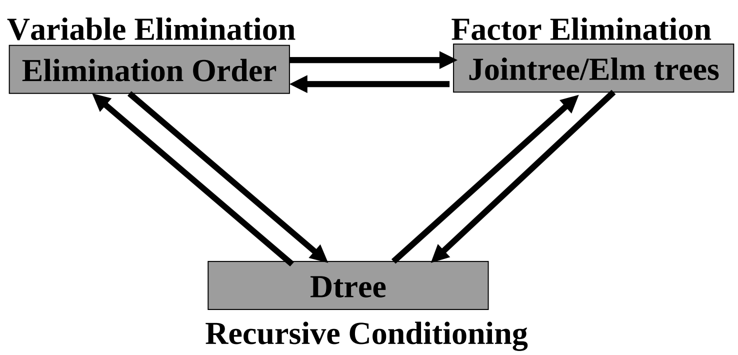
A. Darwiche

Recursive Conditioning in Logic/CSP



A. Darwiche

Graphical Models



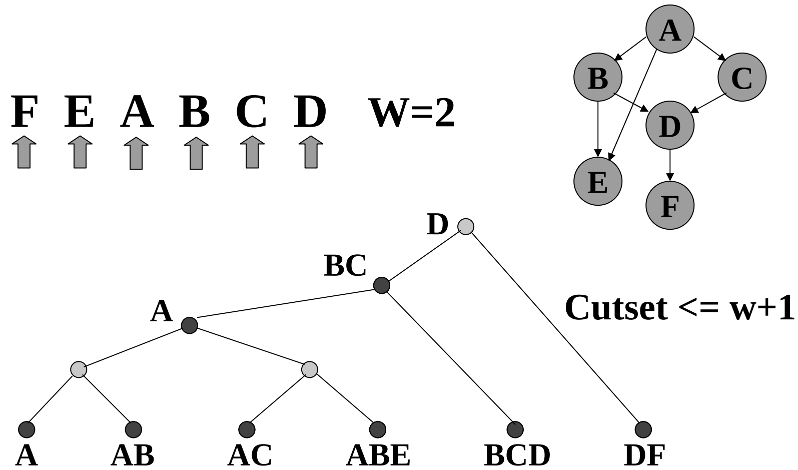
W

Width preserving transformations

A. Darwiche

Elimination Orders to dtrees

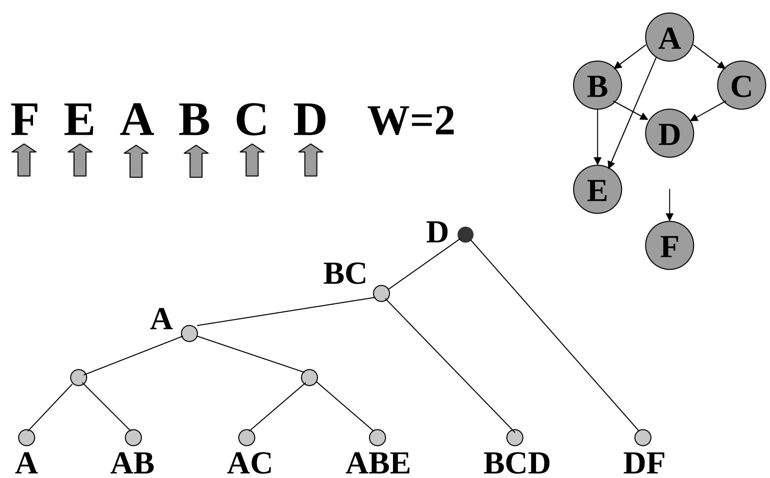
F E A B C D W=2



A. Darwiche

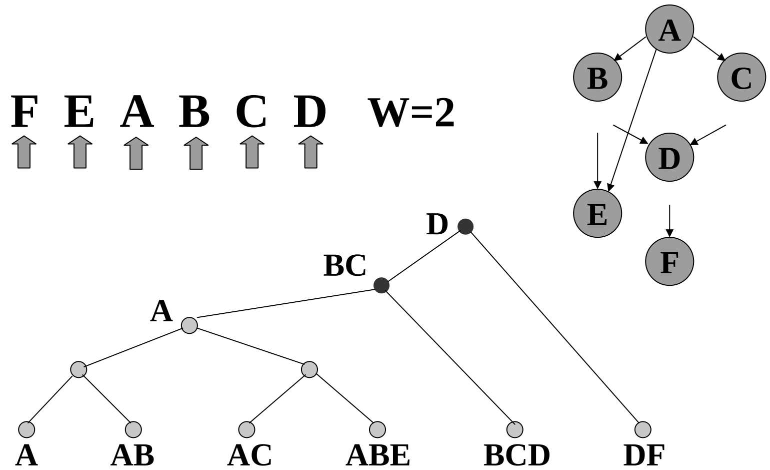
Elimination Orders to dtrees

F E A B C D W=2



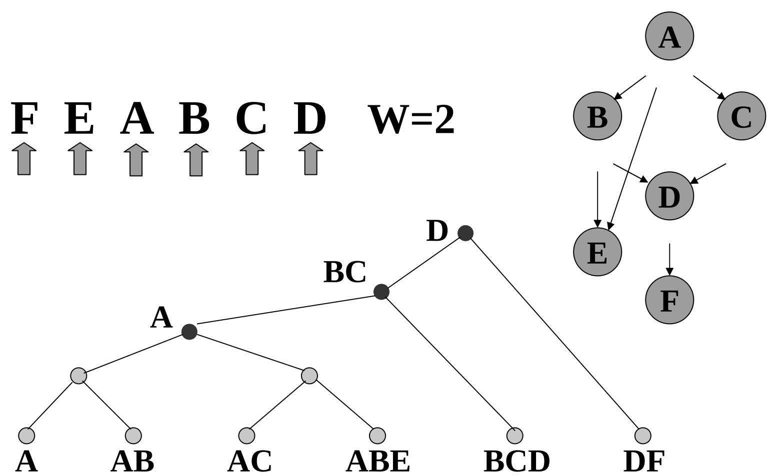
A. Darwiche

Elimination Orders to dtrees



A. Darwiche

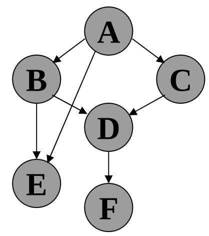
Elimination Orders to dtrees

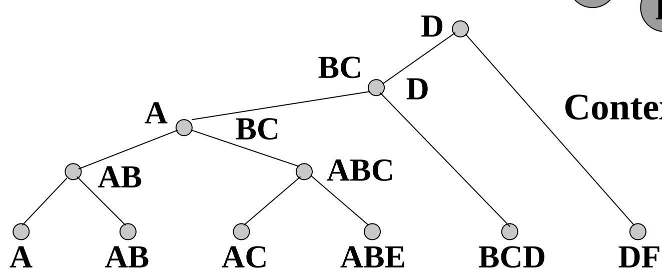


A. Darwiche

Elimination Orders to dtrees

F E A B C D W=2



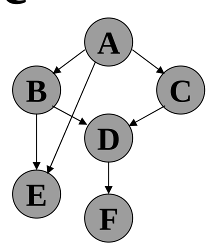


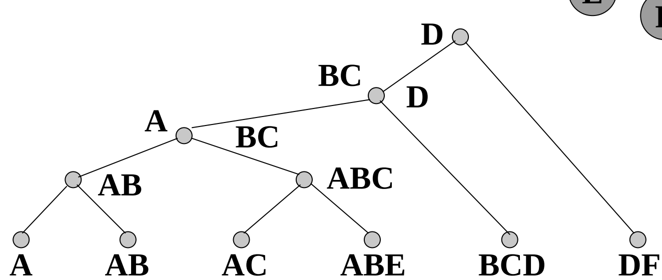
Context <= w+1

A. Darwiche

Dtrees to Jointree

F E A B C D W=2

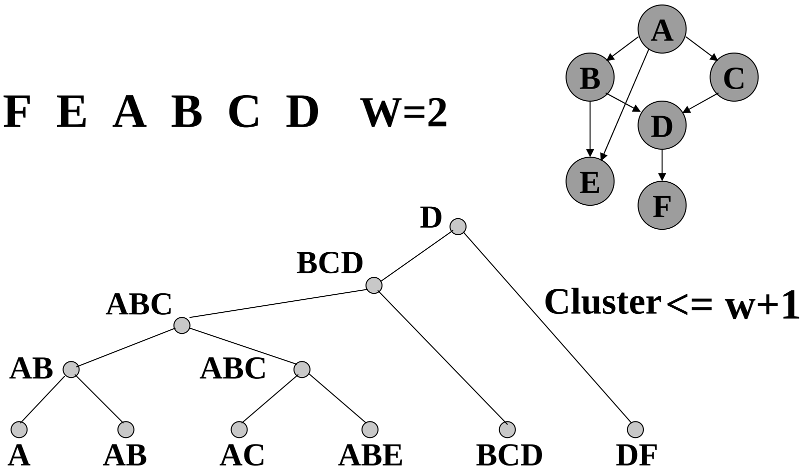




A. Darwiche

Dtrees to Jointrees

F E A B C D W=2

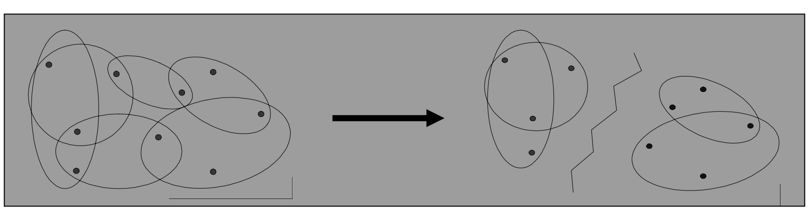


A. Darwiche

Hypergraph Partitioning

The problem of hypergraph partitioning is well-studied in VLSI design.
....and is alive!

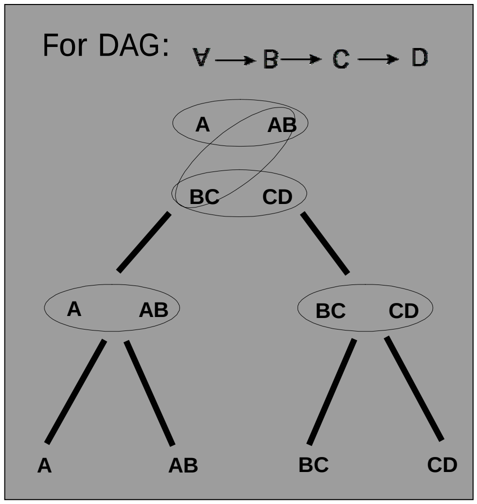
A hypergraph is a generalization of a graph, such that an edge is permitted to connect an arbitrary number of vertices, rather than exactly two.



The task of hypergraph partitioning is to find a way to split the vertices of a hypergraph into k approximately equal parts, such that the number of hyperedges connecting vertices in different parts is minimized.

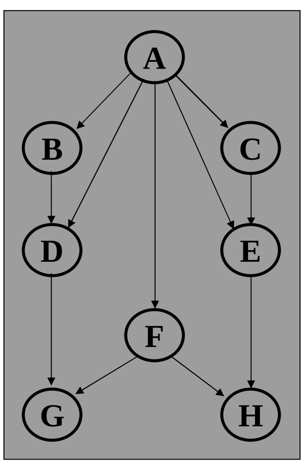
A. Darwiche

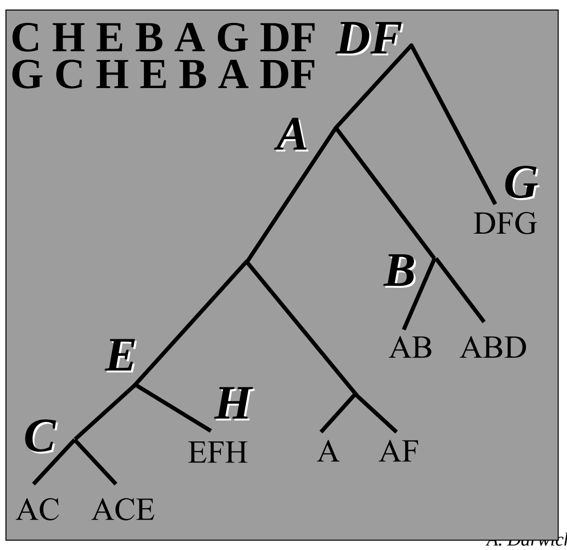
Hypergraph Partitioning => dtrees



A. Darwiche

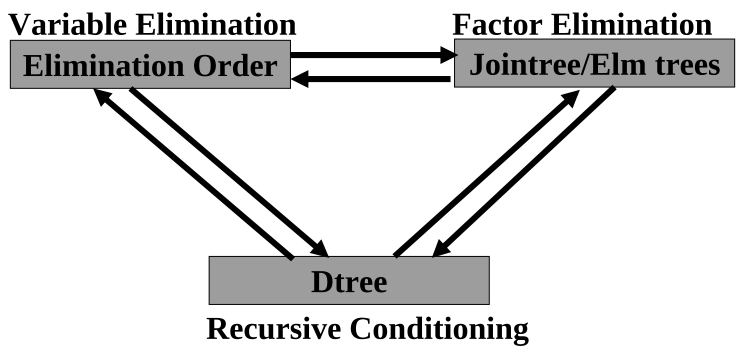
dtrees to Elimination Orders





A. Darwiche

Graphical Models



W

Width preserving transformations

A. Darwiche

Summary

- <u>Inference by variable elimination:</u> Elimination orders
- <u>Inference by factor elimination:</u> Elimination trees / tree decompositions / cluster trees / jointrees
- <u>Inference by recursive conditioning:</u> Decomposition trees (dtrees)
- <u>Treewidth</u> guarantees shared between all methods

A. Darwiche

Knowledge Compilation

A. Darwiche

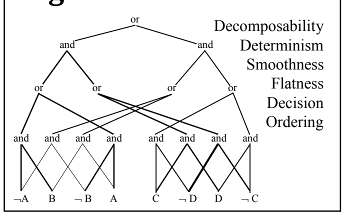
Knowledge Compilation

- <u>Representational factorization:</u> Compact representation/specification of a function (Bayesian network, CNF, CSP, …)
- <u>Computational factorization:</u> A representation of a function on which (some) inference can be performed in polytime
- <u>Knowledge Compilation:</u> Convert a representational factorization into a (smallest) computational factorization

A. Darwiche

KC in Logic: Boolean Circuits
(NNFs / AND-OR Graphs)

Negation Normal Form



Succinctness

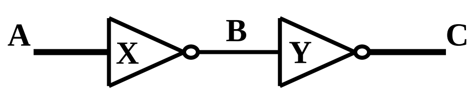
Polytime Operations

Consistency (CO)
Validity (VA)
Clausal entailment (CE)
Sentential entailment (SE)
Implicant testing (IP)
Equivalence testing (EQ)
Model Counting (CT)
Model enumeration (ME)

Projection (existential quantification)
Conditioning
Conjoin, Disjoin, Negate

A. Darwiche

Example from Diagnosis



KB =
```
 A & ok_X => ¬B
¬A & ok_X =>  B

 B & ok_Y => ¬C
¬B & ok_Y =>  C
```

Is A, ¬C a normal device behavior?

A. Darwiche

Example from Diagnosis

```
 A & ok_X => ¬B
¬A & ok_X =>  B

 B & ok_Y => ¬C
¬B & ok_Y =>  C
```

Is A, ¬C a normal device behavior?

```
 A & ok_X => ¬B
¬A & ok_X =>  B

 B & ok_Y => ¬C
¬B & ok_Y =>  C
```

A, ¬C, ok_X, ok_Y

Satisfiability Algorithm

A. Darwiche

Example from Diagnosis

```
 A & ok_X => ¬B
¬A & ok_X =>  B

 B & ok_Y => ¬C
¬B & ok_Y =>  C
```

A, ¬C, ok_X, ok_Y

Satisfiability Algorithm

Is there a satisfying assignment?

A	B	C	OK_X	OK_Y
T	T	T	T	T
T	T	T	T	F
…	…	...	…	…
…	…	…	…	…
…	…	…	…	…
F	F	F	F	F

A. Darwiche

Example from Diagnosis

```
 A & ok_X => ¬B
¬A & ok_X =>  B

 B & ok_Y => ¬C
¬B & ok_Y =>  C
```

A, ¬C, ok_X, ok_Y

Counting Algorithm

How many satisfying assignments?

A	B	C	OK_X	OK_Y
T	T	T	T	T
T	T	T	T	F
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…
F	F	F	F	F

A. Darwiche

Knowledge Compilation

```
 A & ok_X => ¬B
¬A & ok_X =>  B

 B & ok_Y => ¬C
¬B & ok_Y =>  C
```

Compiler

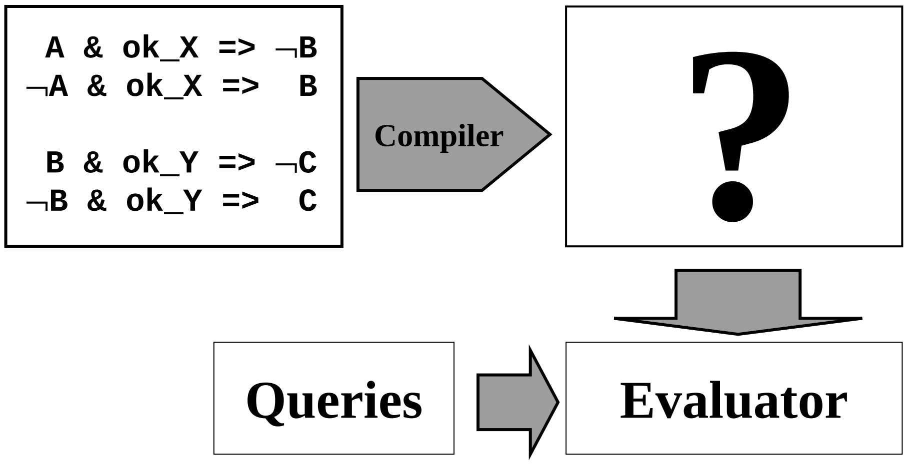
Compiled Structure

Queries

Evaluator

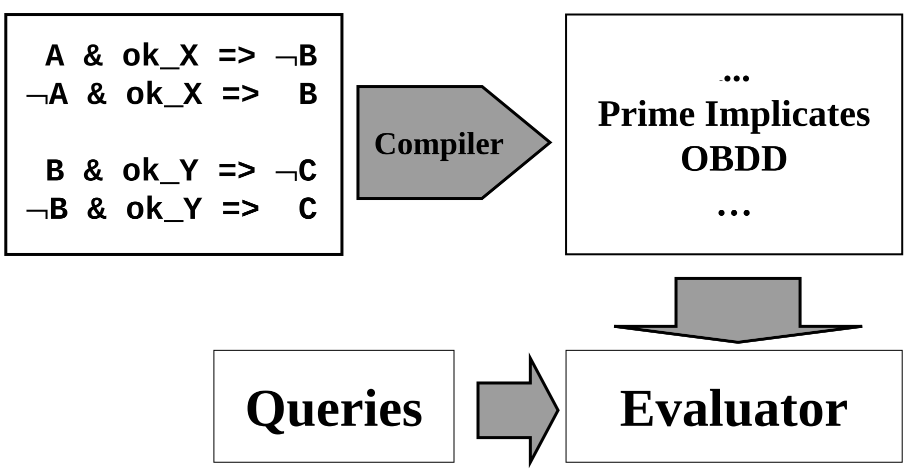
A. Darwiche

Knowledge Compilation



A. Darwiche

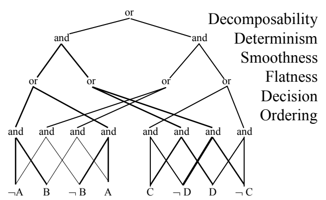
Knowledge Compilation



A. Darwiche

Knowledge Compilation

Negation Normal Form



Succinctness

Polytime Operations

Consistency (CO)
Validity (VA)
Clausal entailment (CE)
Sentential entailment (SE)
Implicant testing (IP)
Equivalence testing (EQ)
Model Counting (CT)
Model enumeration (ME)

Projection (existential quantification)
Conditioning
Conjoin, Disjoin, Negate

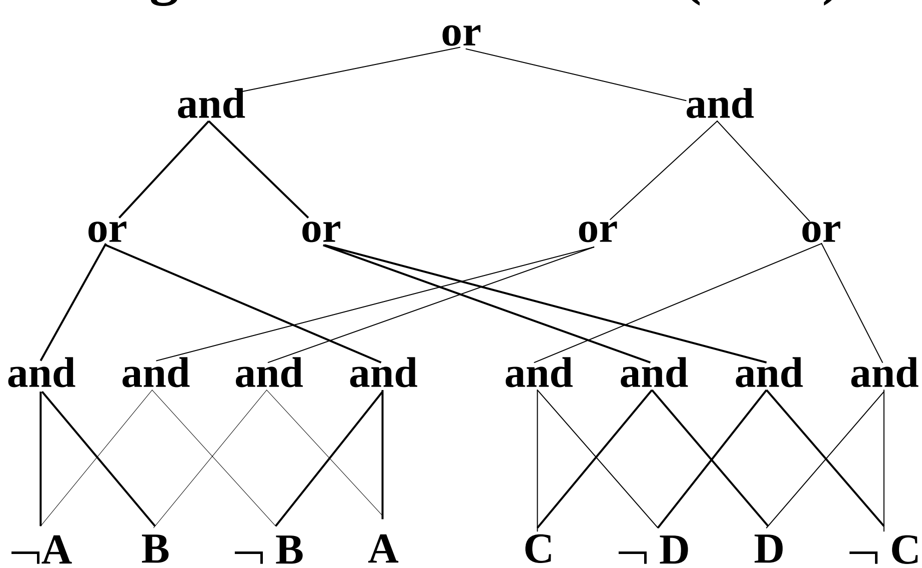
A. Darwiche

Applications

- Diagnosis
  - Is this a normal behavior?
  - What are the possible faults?
- Planning
  - Can this goal be achieved?
  - Generate a set of plans
- Probabilistic reasoning
  - What is the probability of X given Y
- Non-monotonic reasoning (penalty logics)
  - Does X follow preferentially from Y
- Formal verification / CAD:
  - Is it possible that the design will exhibit behavior X?
  - Are two designs equivalent?

A. Darwiche

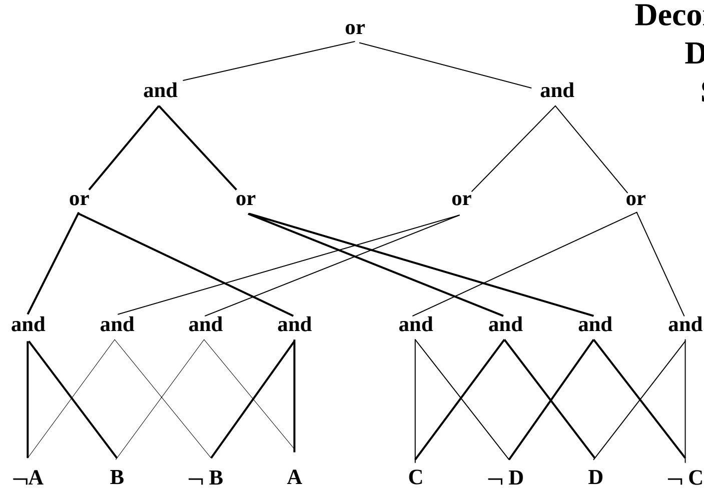
Negation Normal Form (NNF)



rooted DAG

A. Darwiche

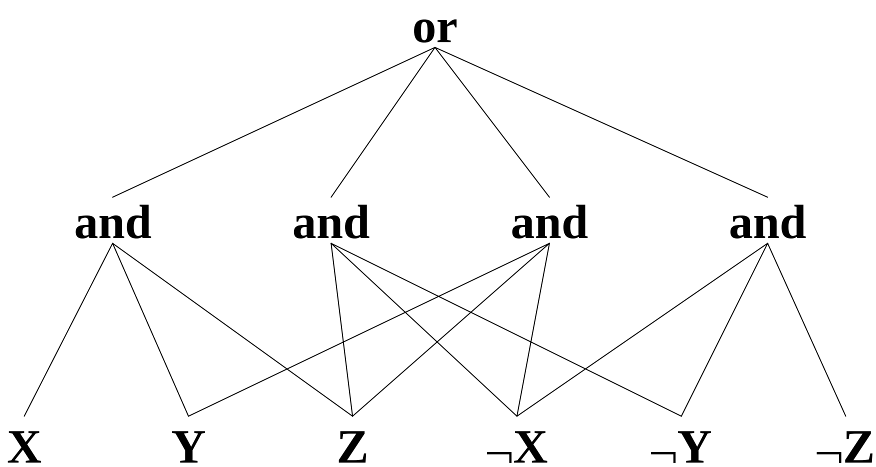
Negation Normal Form



Decomposability
Determinism
Smoothness
Flatness
Decision
Ordering

A. Darwiche

Flatness

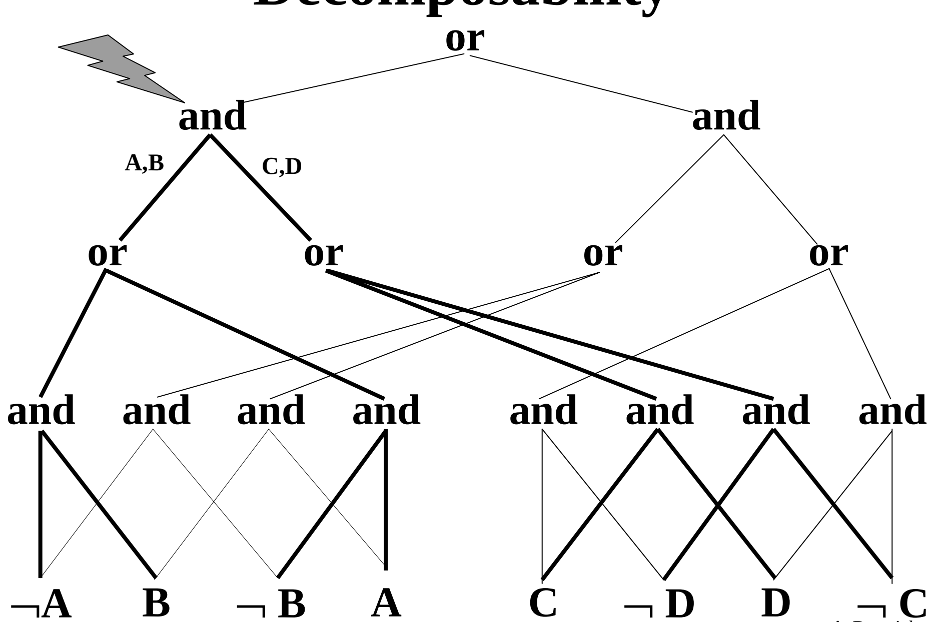


(X∧Y ∧Z)∨(Z∧¬X∧¬Y) ∨(Y∧Z∧¬X) ∨(¬X∧¬Y ∧¬Z)

Nested vs Flat languages

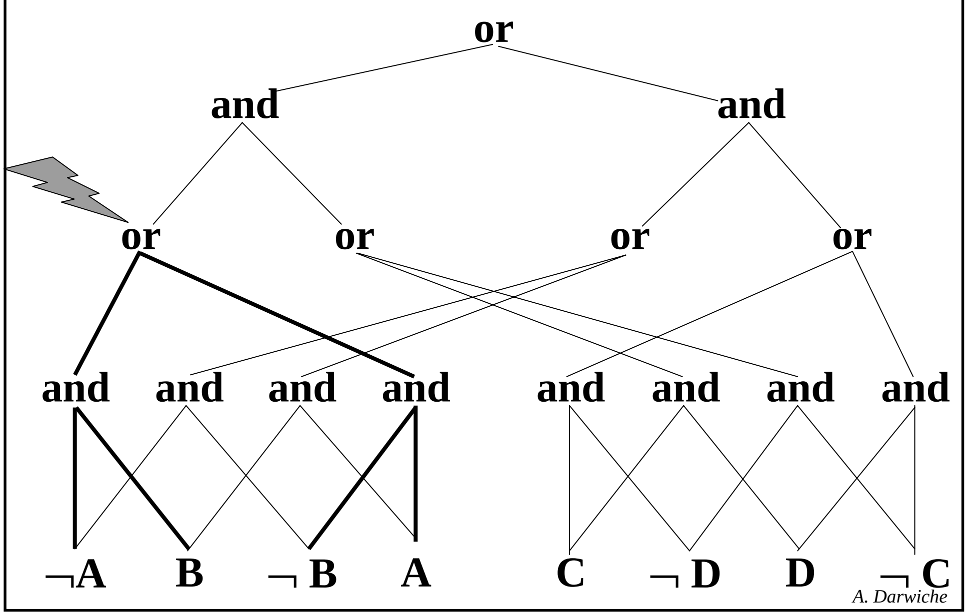
A. Darwiche

Decomposability



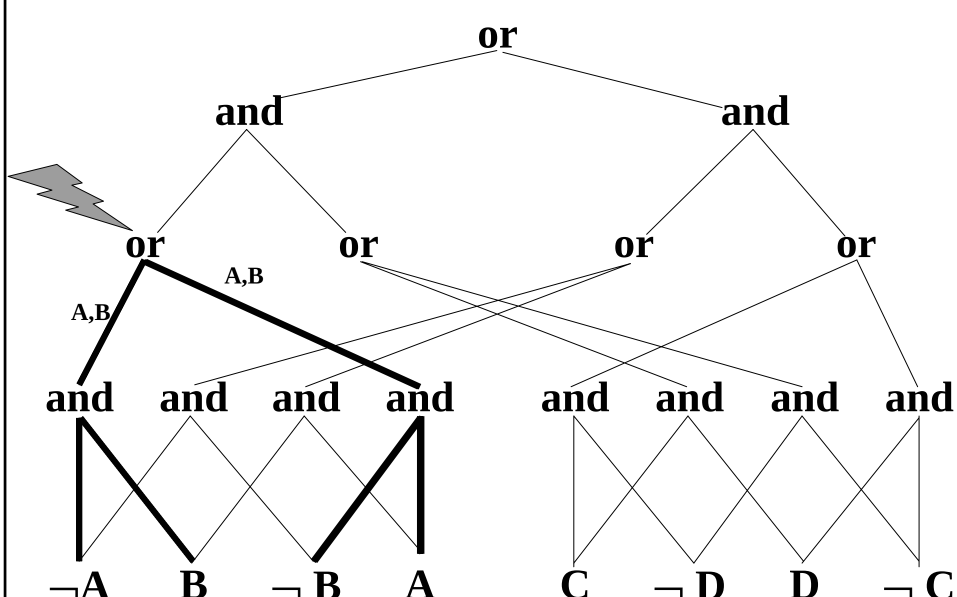
A. Darwiche

Determinism



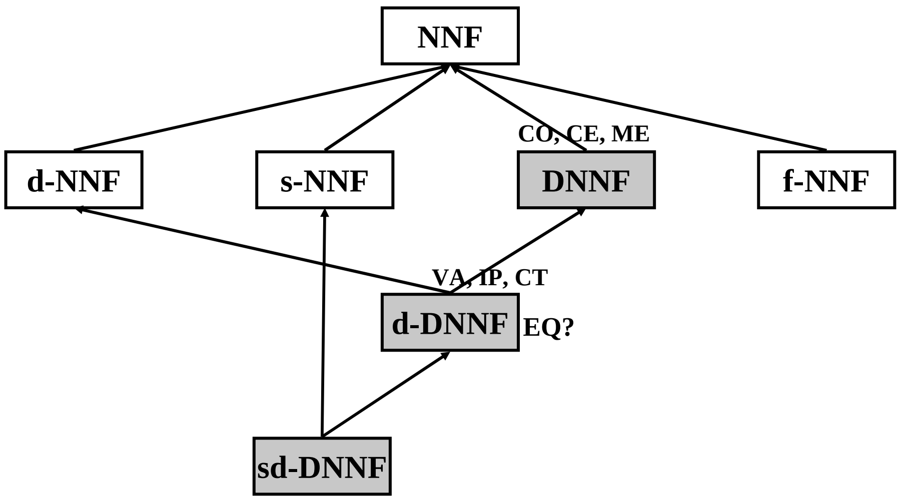
A. Darwiche

Smoothness



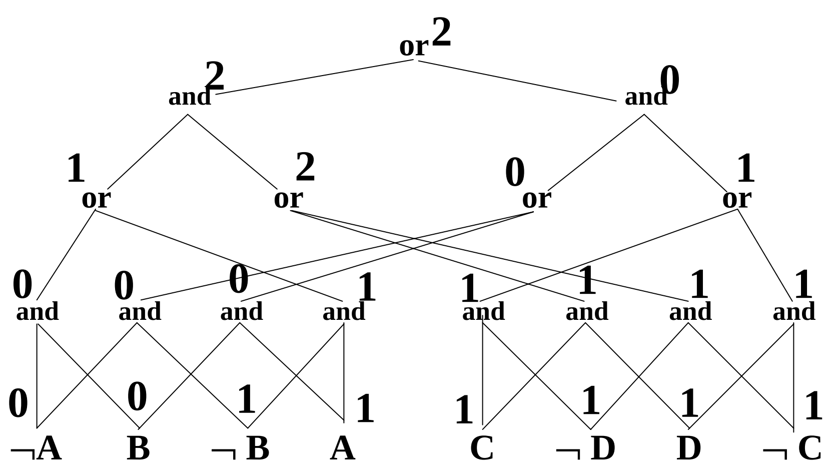
A. Darwiche

NNF Subsets



A. Darwiche

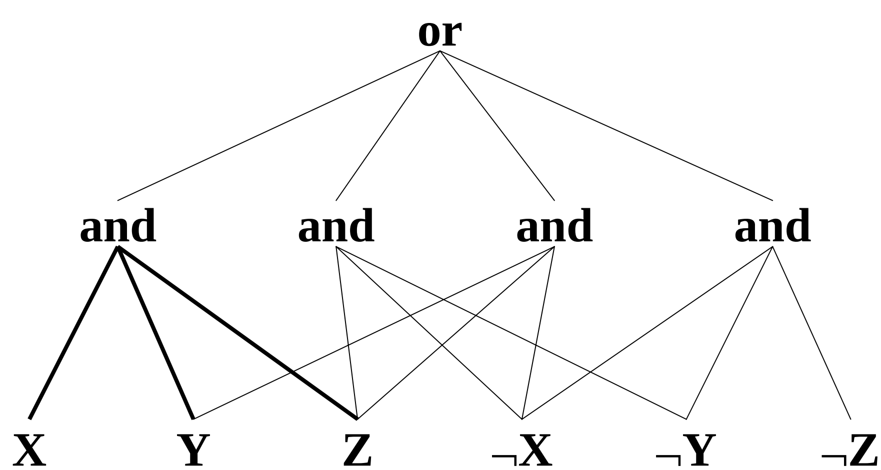
Counting



Given A, ¬ B

A. Darwiche

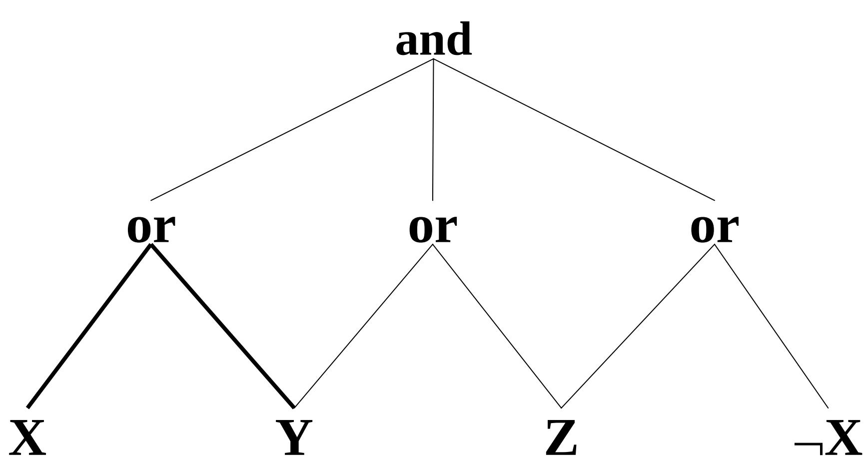
Simple Conjunction



Implies decomposability

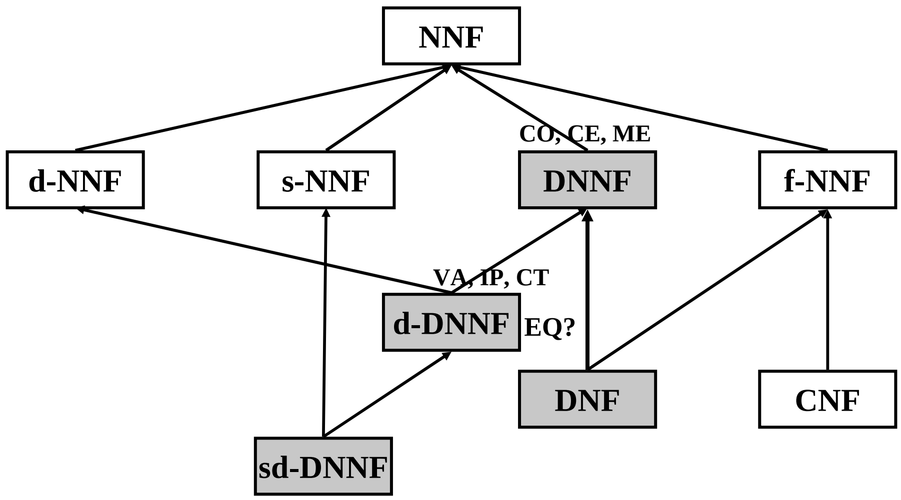
A. Darwiche

Simple Disjunction



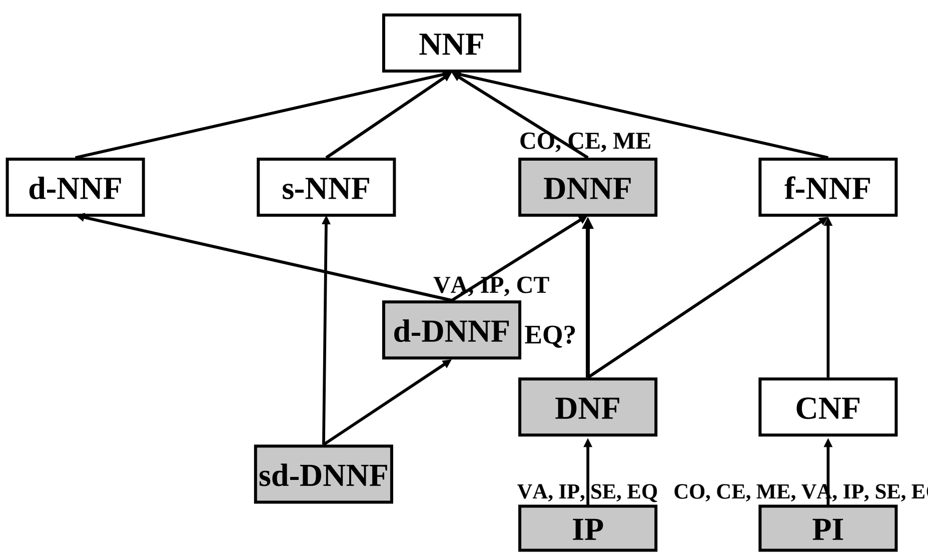
A. Darwiche

NNF Subsets



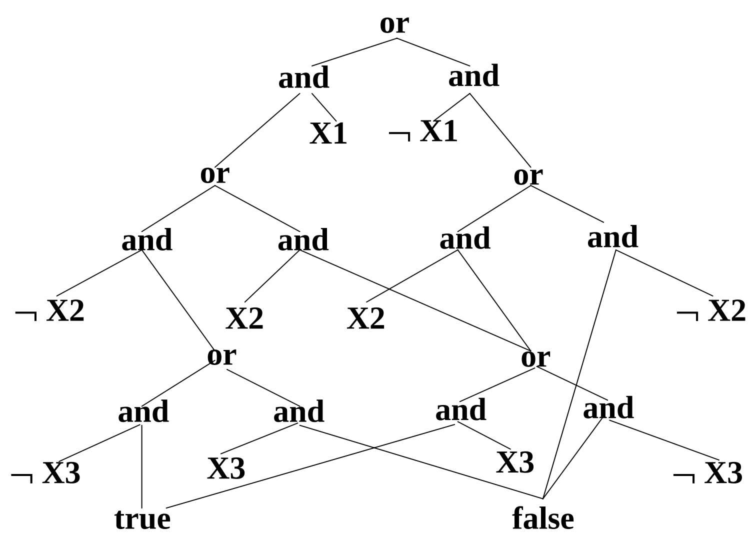
A. Darwiche

NNF Subsets



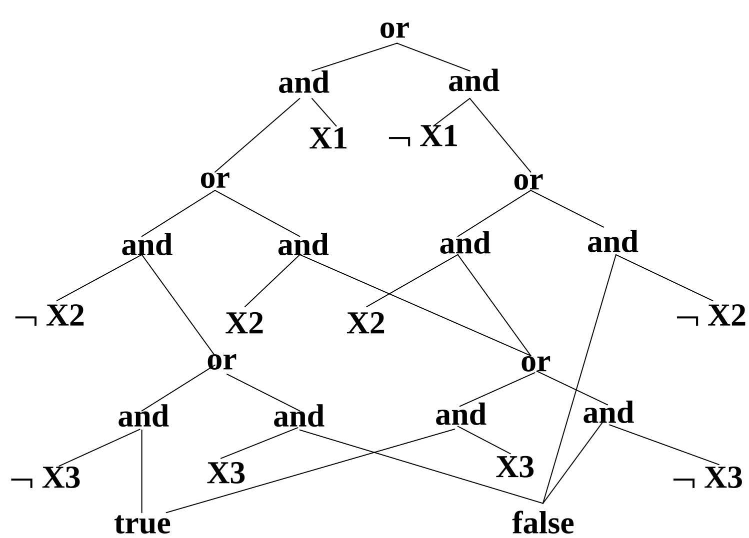
A. Darwiche

Decision



A. Darwiche

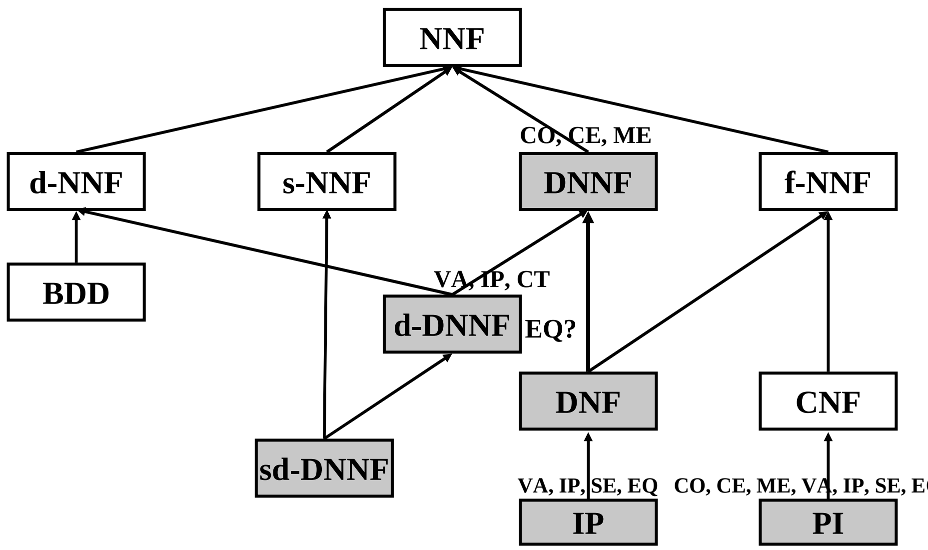
Decision



Decision implies determinism

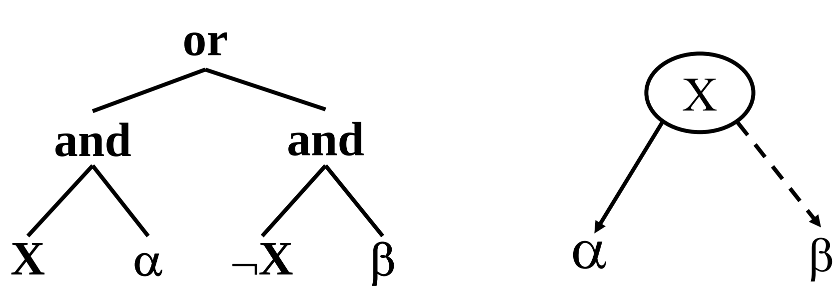
A. Darwiche

NNF Subsets



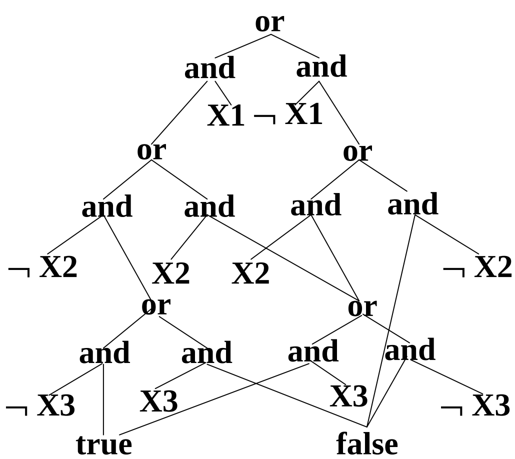
A. Darwiche

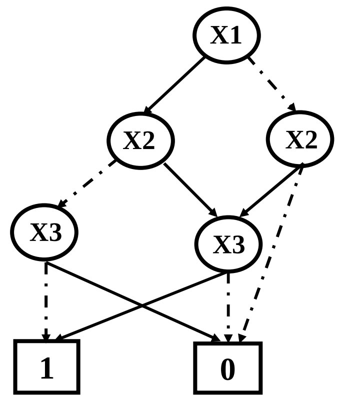
Decision Nodes



A. Darwiche

Binary Decision Diagram

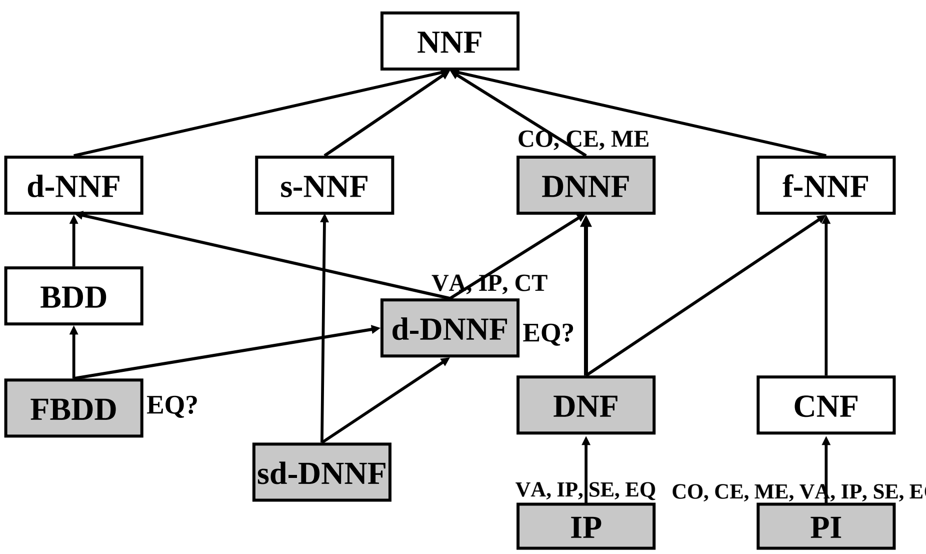




$(X2 \wedge X3) \vee (X1 \wedge \neg X2 \wedge \neg X3)$

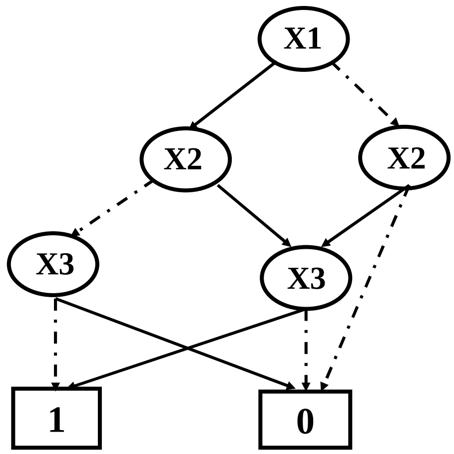
A. Darwiche

NNF Subsets



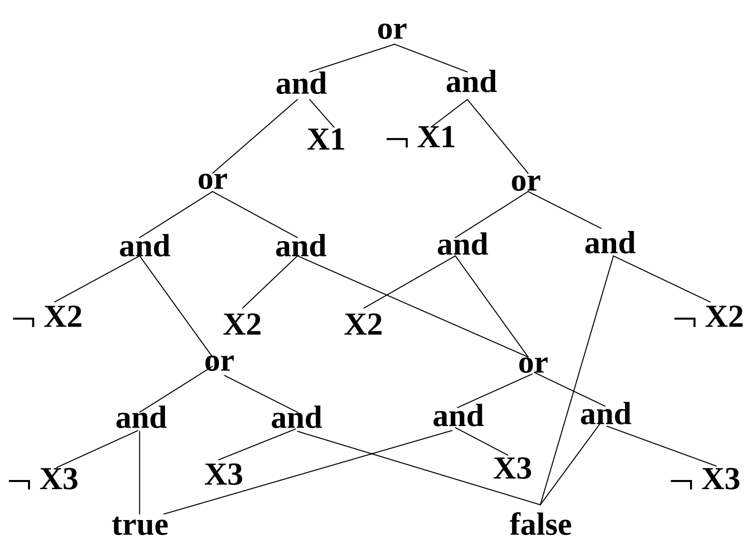
A. Darwiche

Free Binary Decision Diagram



Test-once property

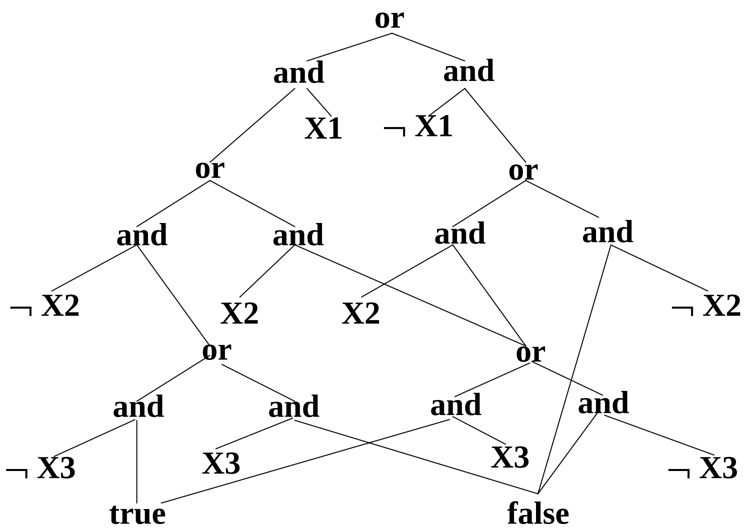
A. Darwiche



In the context of decision: Test-once = decomposability

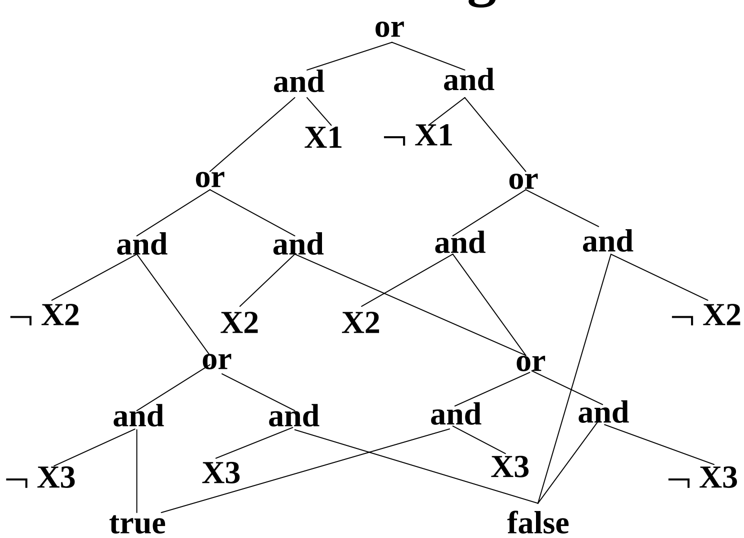
A. Darwiche

Ordering



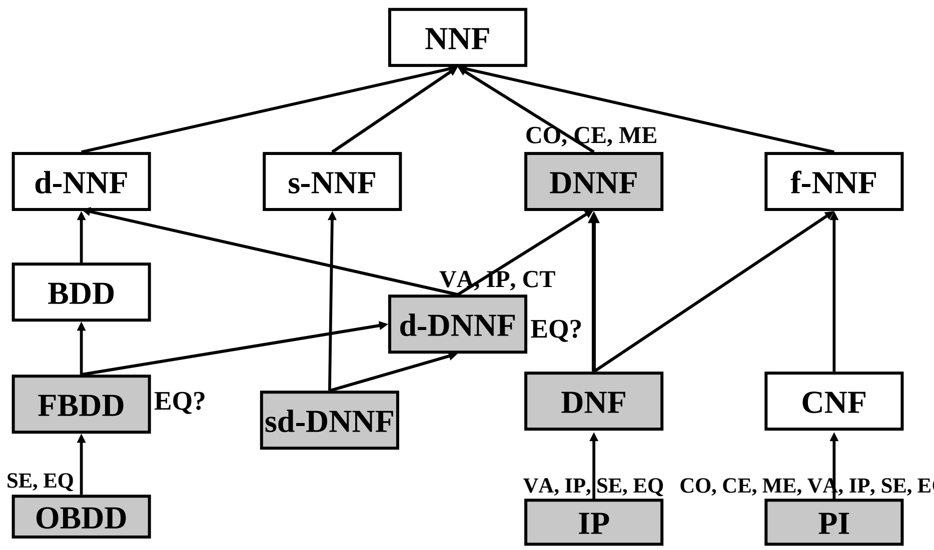
A. Darwiche

Ordering



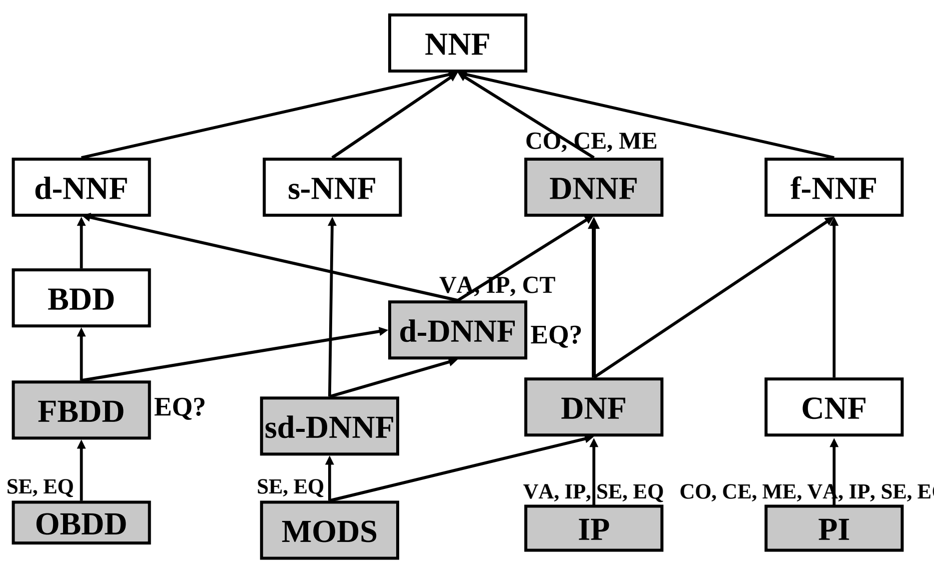
A. Darwiche

NNF Subsets



A. Darwiche

NNF Subsets



A. Darwiche

Language Succinctness

L1 at least as succinct as L2

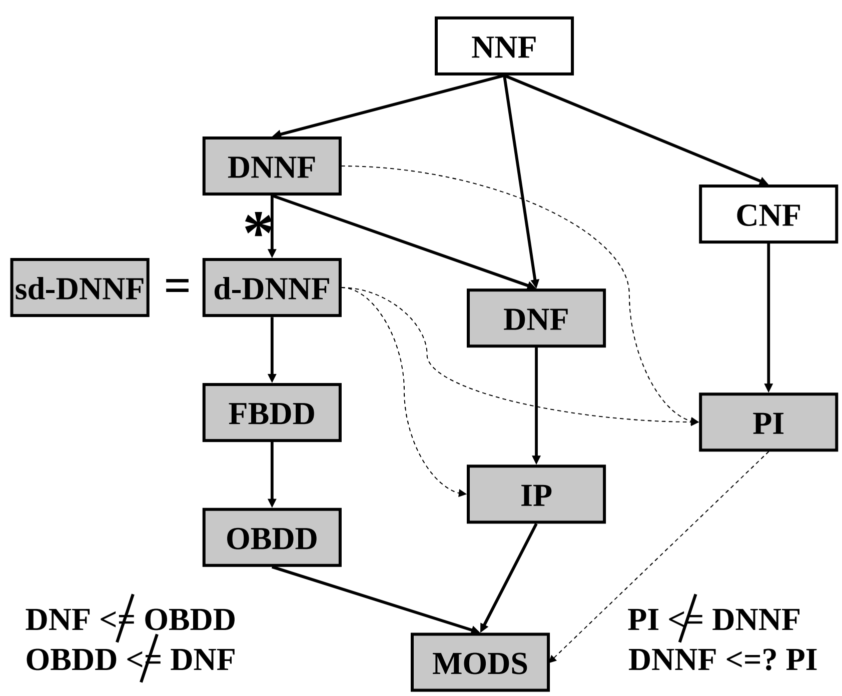
L1 <= L2

Size p(n) Size n

L1 is more succinct than L2

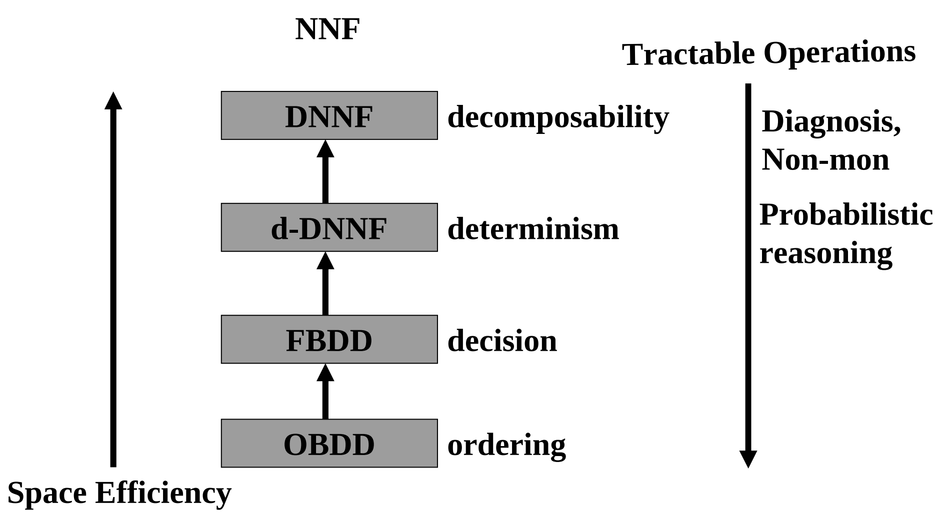
L1 < L2

A. Darwiche



A. Darwiche

Tractability vs Succinctness



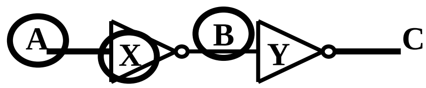
A. Darwiche

Propositional Transformations

- Project (Forget)
- Condition
- Conjoin
- Disjoin
- Negate

A. Darwiche

Projection



KB =

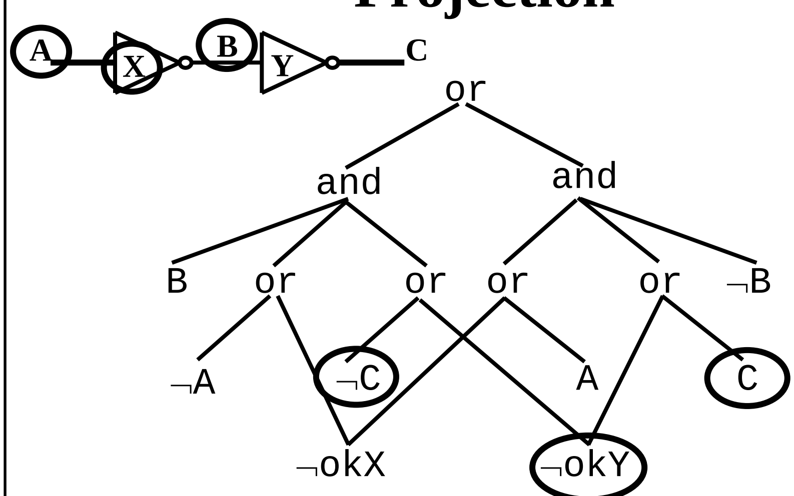
```
 A & ok(X) => ¬B
¬A & ok(X) =>  B

 B & ok(Y) => ¬C
¬B & ok(Y) =>  C
```

A & ok(X) => ¬B
¬A & ok(X) => B

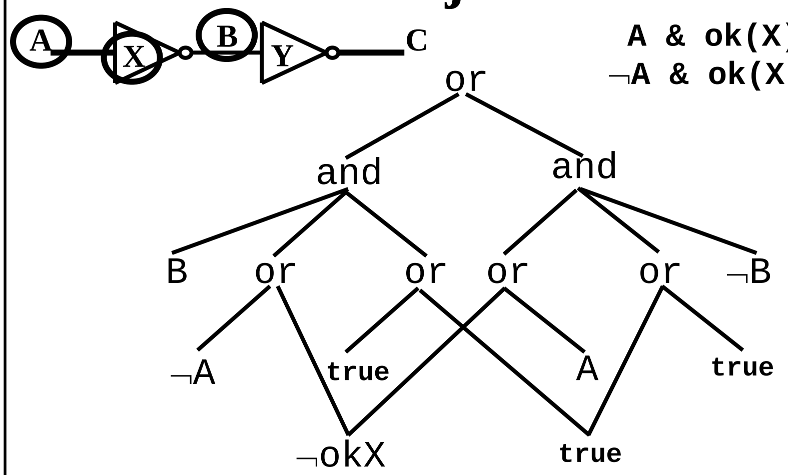
A. Darwiche

Projection



A. Darwiche

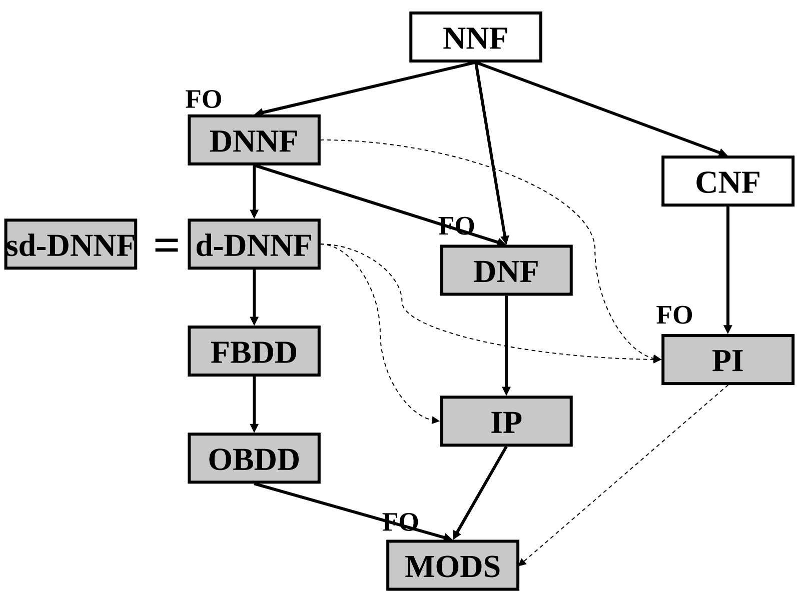
Projection




```
 A & ok(X) => ¬B
¬A & ok(X) =>  B
```

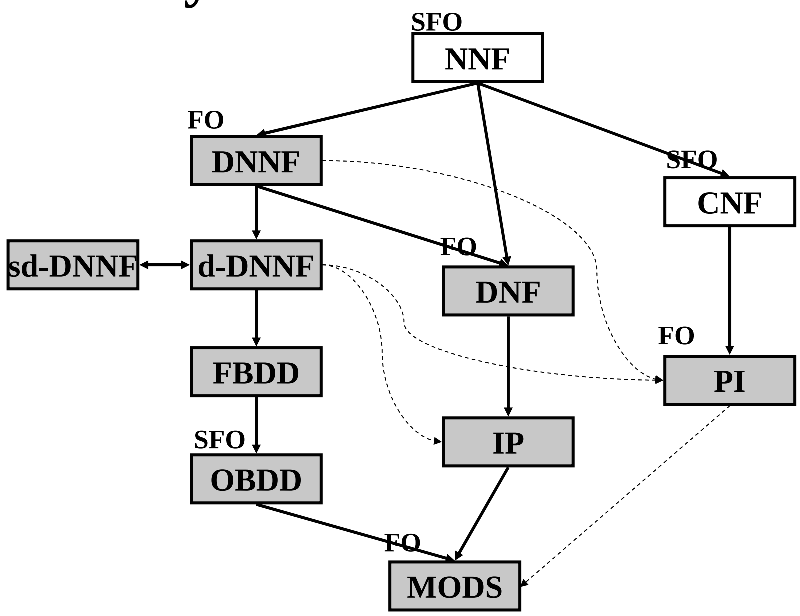
A. Darwiche

Polytime Transformations



A. Darwiche

Polytime Transformations



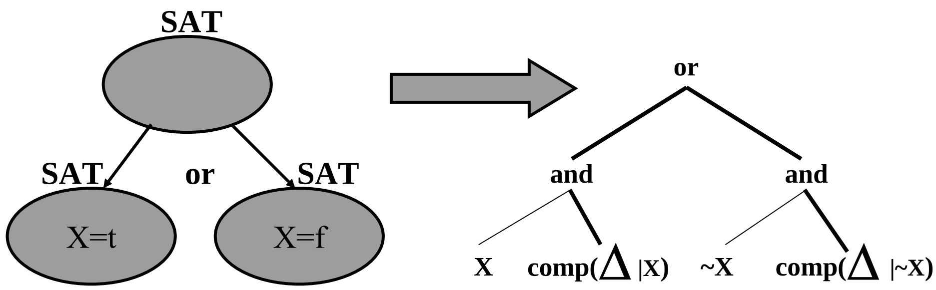
A. Darwiche

Polytime Transformations

- Closure under conjunction: none!
  Bounded conjunction: OBDD, DNF, IP, MODs
- Closure under disjunction: DNF, DNNF
  Bounded disjunction: OBBD, PI
- Closure under negation: FBDD, OBDD
- Conditioning: all!
- Forgetting multiple variables: DNNF, DNF, PI, MODS
- Forgetting single variable: OBDD

A. Darwiche

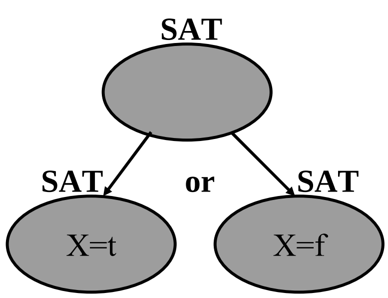
A Knowledge Compiler: DPLL with a Trace



DPLL with fixed order and caching → OBDD compiler
DPLL with caching → FBDD compiler

A. Darwiche

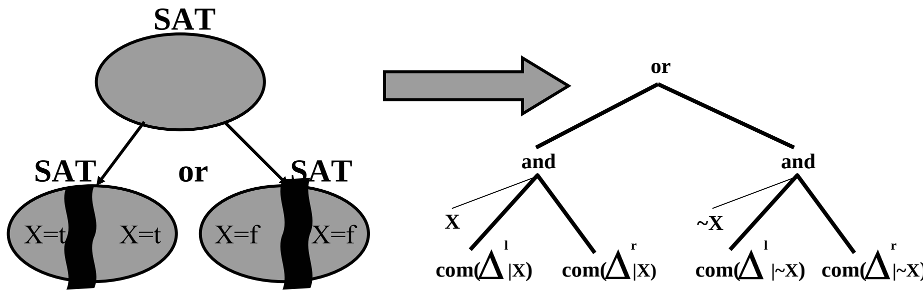
A Knowledge Compiler: DPLL with a Trace



DPLL with fixed order and caching → OBDD compiler
DPLL with caching → FBDD compiler

A. Darwiche

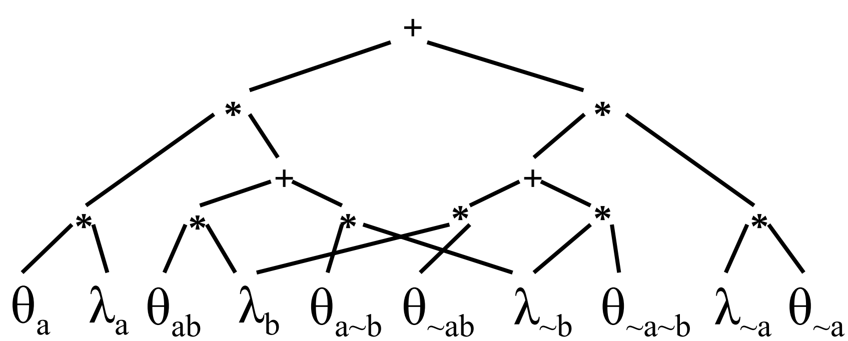
A Knowledge Compiler: DPLL with a Trace



DPLL with fixed order and caching → OBDD compiler
DPLL with caching → FBDD compiler
DPLL with caching and decomposition → d-DNNF compiler*
? → DNNF compiler

A. Darwiche

Knowledge Compilation in Probability: Arithmetic Circuits



ADD (Algebraic Decision Diagrams)
*BMD (Binary Moment Diagrams)
PDGs (Probabilistic Decision Graphs)…

A. Darwiche

Conclusion

- Knowledge compilation converts a representational factorization into a computational factorization:
  – Target language (succinctness, tractability)
- Knowledge compilers can be constructed by keeping the trace of various search algorithms
- Another view of knowledge compilation is based on deductive closure

A. Darwiche

Knowledge Compilation as Deductive Closure: Two Examples

1. Eliminate all variables $X_1,\ldots, X_n$, while keeping all intermediate CNFs
   – The intermediate CNFs represent projections of initial CNF on subsets $X_2,\ldots, X_n$, $X_3,\ldots, X_n$, $X_{n-1}X_n$, X_n
   – The result can be used to answer some hard queries in polytime
2. Add enough clauses to make CNF complete under unit resolution

A. Darwiche

References

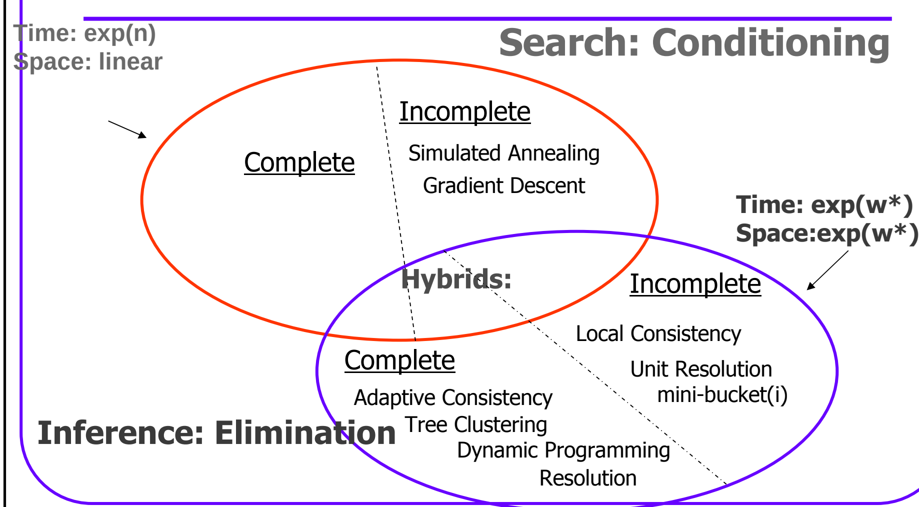
- Dechter. Bucket elimination: A unifying framework for probabilistic inference. In UAI, pages 211-219, 1996.
- Zhang and Poole. Exploiting causal independence in bayesian network inference. JAIR, 5:301-328, 1996.
- Lauritzen and Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems.
  Journal of Royal Statistics Society, Series B, 50(2):157-224, 1988.
- Jensen, Lauritzen, and Olesen. Bayesian updating in recursive graphical models by local computation. Computational Statistics Quarterly, 4:269-282, 1990.
- Huang and Darwiche. Inference in Belief Networks: A procedural guide.
  International Journal of Approximate Reasoning, 15(3): 225-263, 1996.
- Darwiche. Recursive Conditioning. Artificial Intelligence, 126(1-2):5-41, 2001.
- Darwiche and Marquis. A knowledge compilation map. JAIR, 17, 229-264, 2002.

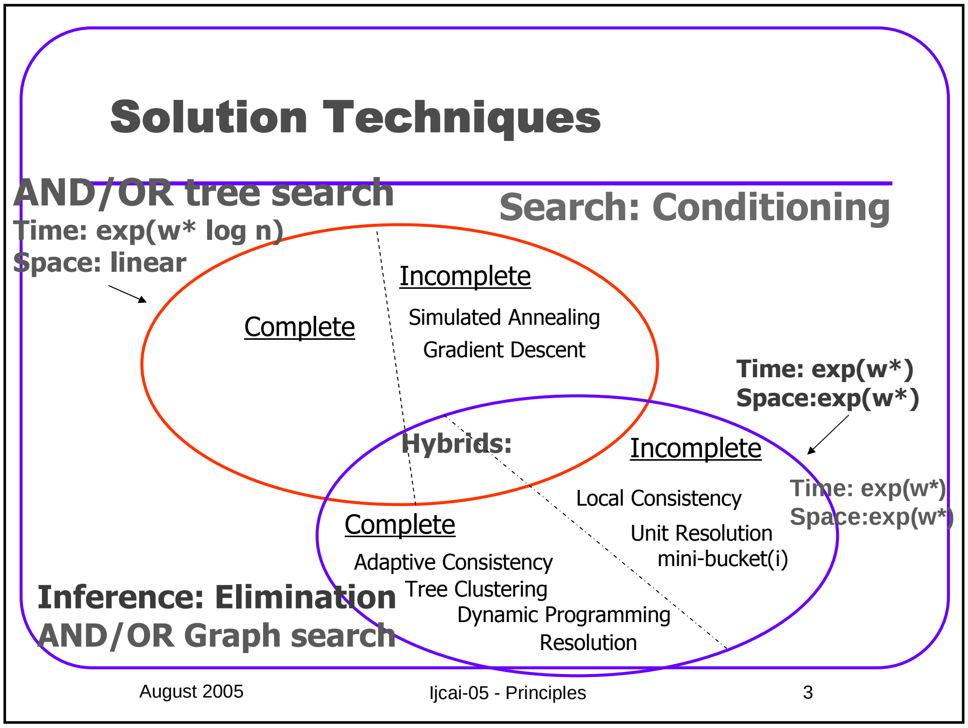
A. Darwiche

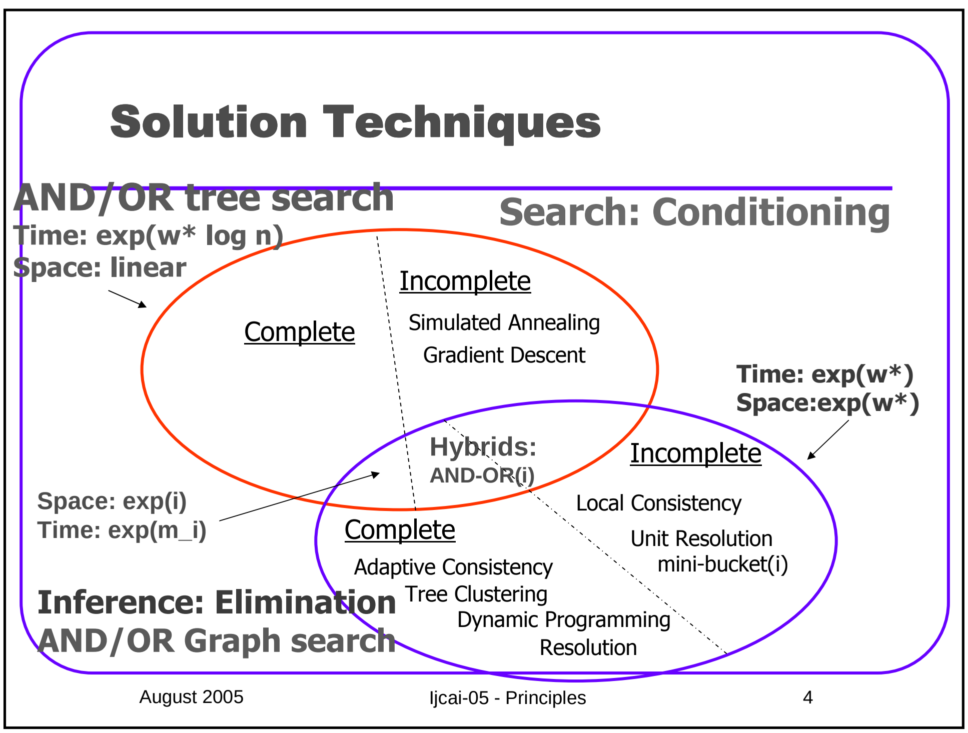
Road Map:
Search in Variable-Based Models

- Variable-based (Graphical) models
- Basic search and basic Inference
- Constraint propagation: bounded inference
- Improving search by branching ahead
- Improving search by looking-back
- The alternative AND/OR search space
- **Hybrid search and inference**

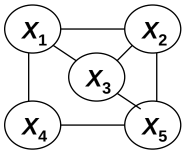
Solution Techniques




Solution Techniques
AND/OR tree search
Time: exp(w* log n)
Space: linear
Search: Conditioning
Complete
Incomplete
Simulated Annealing
Gradient Descent
Time: exp(w*)
Space:exp(w*)
Hybrids:
Incomplete
Time: exp(w*)
Space:exp(w*)
Local Consistency
Unit Resolution
mini-bucket(i)
Complete
Adaptive Consistency
Tree Clustering
Dynamic Programming
Resolution
Inference: Elimination
AND/OR Graph search
August 2005
Ijcai-05 - Principles
3

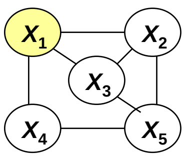

Solution Techniques
AND/OR tree search
Time: exp(w* log n)
Space: linear
Search: Conditioning
Complete
Incomplete
Simulated Annealing
Gradient Descent
Time: exp(w*)
Space:exp(w*)
Hybrids:
AND-OR(i)
Incomplete
Space: exp(i)
Time: exp(m_i)
Local Consistency
Unit Resolution
mini-bucket(i)
Complete
Adaptive Consistency
Tree Clustering
Dynamic Programming
Resolution
Inference: Elimination
AND/OR Graph search
August 2005
Ijcai-05 - Principles
4

Search Basic Step: Variable Branching by Conditioning

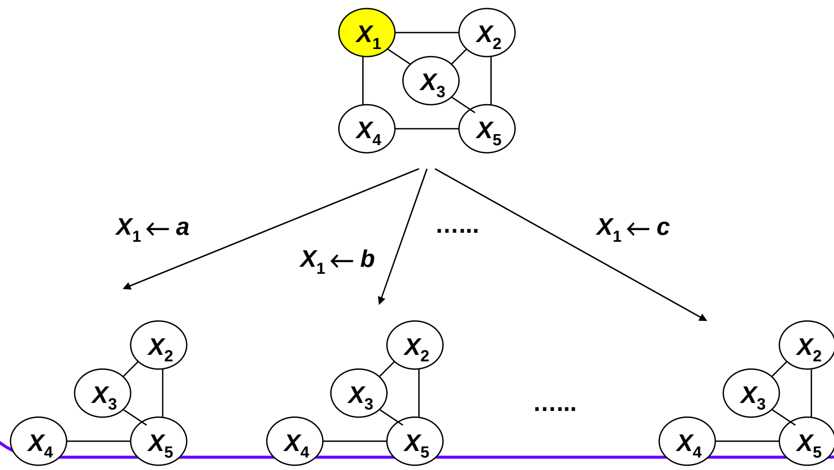


Search Basic Step: Variable Branching by conditioning

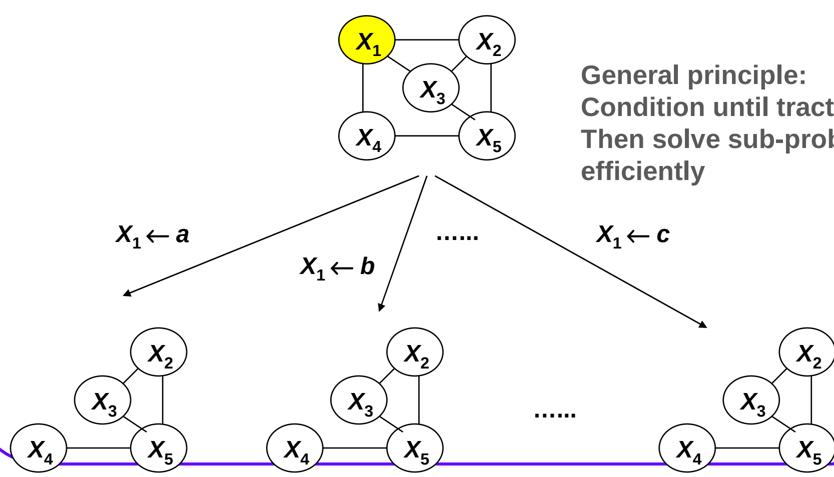
- Select a variable



Search Basic Step: Variable Branching by Conditioning

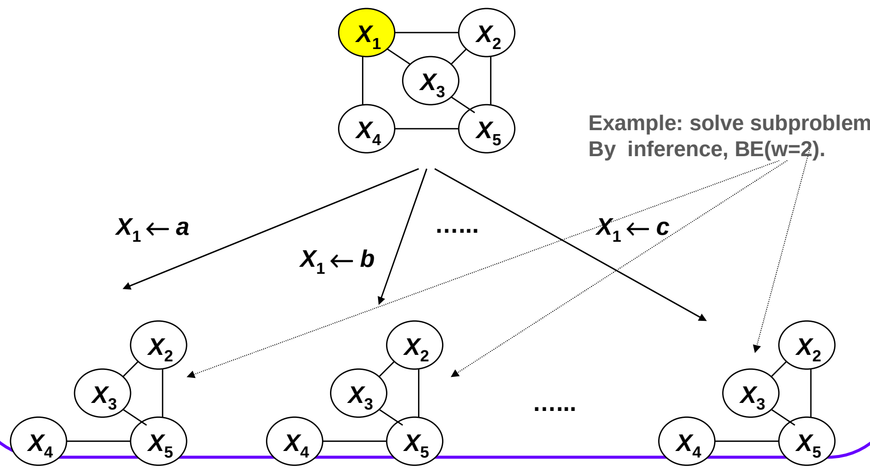


Search Basic Step: Variable Branching by Conditioning

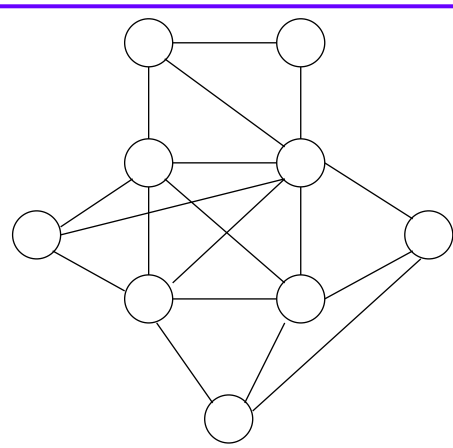


General principle:
Condition until tractable
Then solve sub-problems efficiently

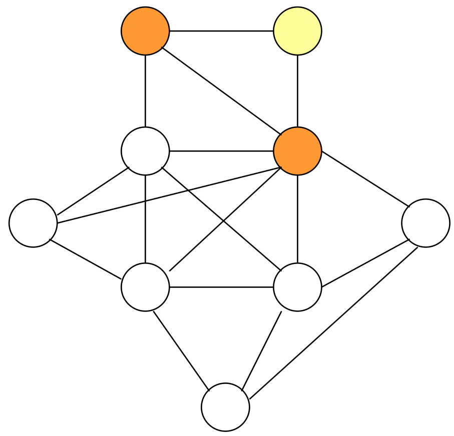
Search Basic Step: Variable Branching by Conditioning



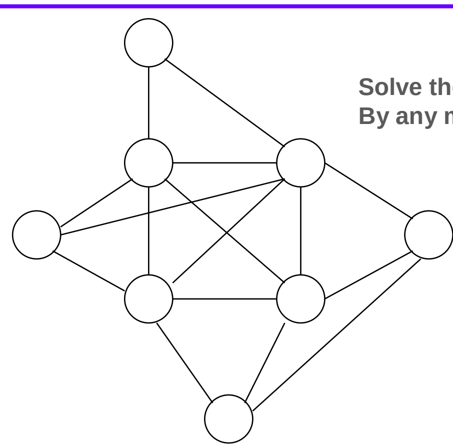
Eliminate First



Eliminate First



Eliminate First



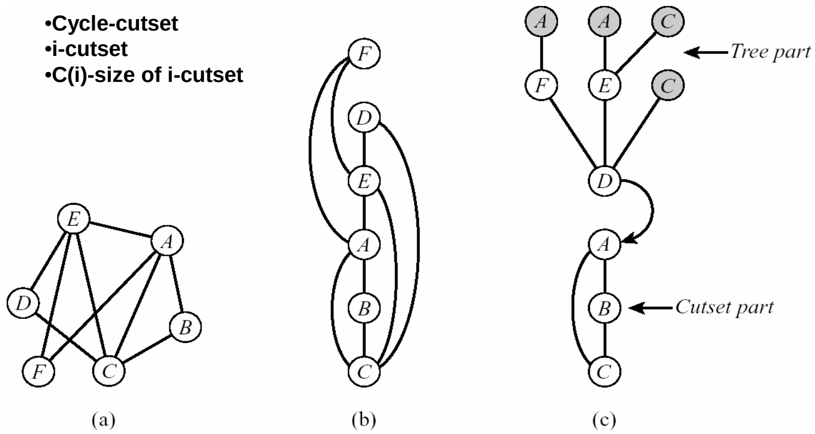
Solve the rest of the problem
By any means

Two Hybrids

- Condition, condition, condition… and then only eliminate (i-cutset, cycle-cutset)
- Interleave conditioning and elimination(elim-cond(i))

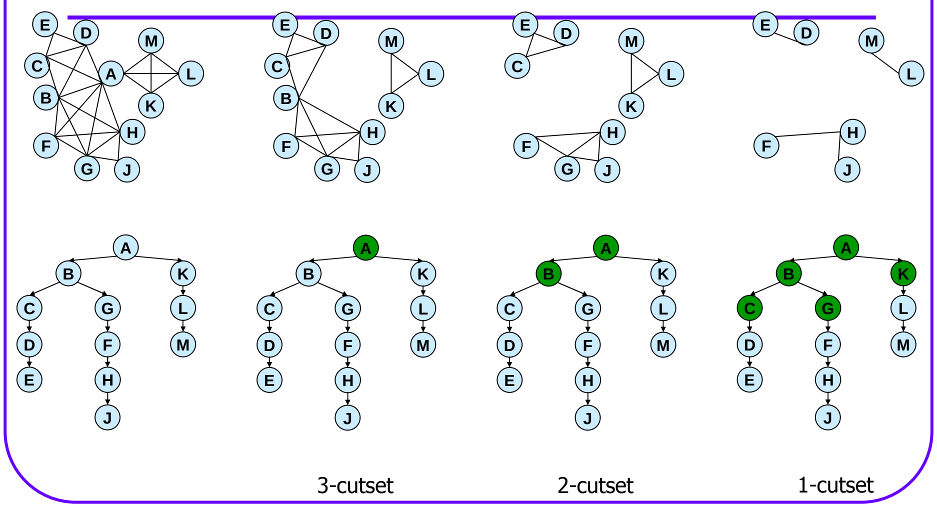
The Cycle-Cutset Scheme: Condition Until Treeness

- •Cycle-cutset
- •i-cutset
- •C(i)-size of i-cutset



Space: exp(i), Time: O(exp(i+c(i))

AND/OR i-cutset



Time-Space Complexity

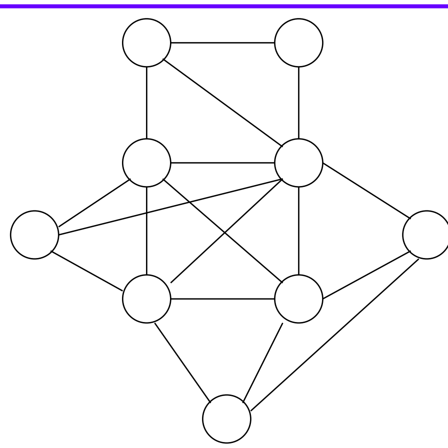
- <u>Space</u>: O(exp(*i*))
  - ***i*-cutset: a set that when removed the induced-width is i.**
  - ***c*(*i*): size of i-cutset.**
  - **m(i): depth of AO i-cutset**
- <u>Time</u>: O(exp(*i*+*c*(*i*))) on OR space
- <u>Time</u>: O(exp(*i*+*m*(*i*))) on AND/OR space and m(i) <= c(i)

Time-Space complexity

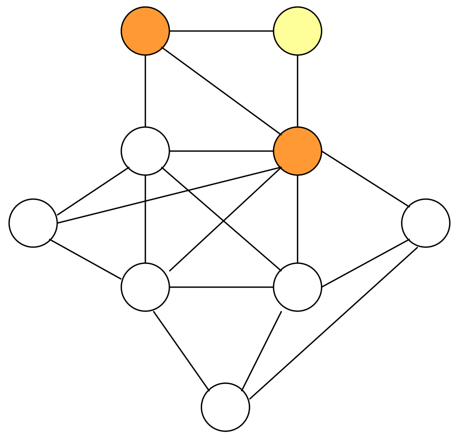
- Space: O(exp(*i*))
  - ***i*-cutset: a set that when removed the induced-width is i.**
  - ***c*(*i*): size of i-cutset.**
  - **m(i): depth of AO i-cutset**
- Time: O(exp(*i*+*c*(*i*))) on OR space
- Time: O(exp(*i*+*m*(*i*))) on AND/OR space and m(i) <= c(i)

$$c(1)+1 \geq 2+c(2) \geq \ldots \geq i+c(i) \geq \ldots w^*+c(w^*) = tw^*$$
$$tw^* \leq m(i)+i \leq c(i)+i,$$

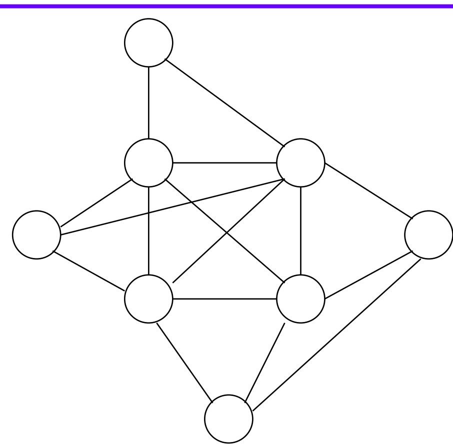
Interleaving Cond and Elim



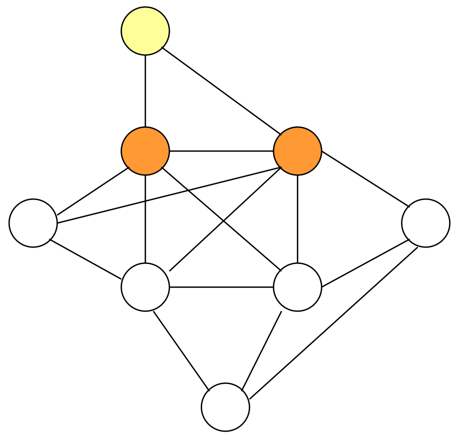
Interleaving Cond and Elim



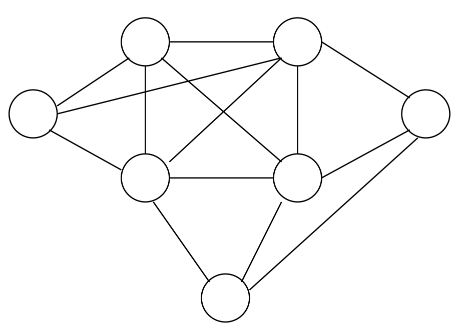
Interleaving Cond and Elim



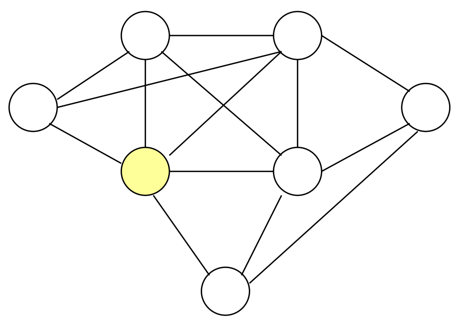
Interleaving Cond and Elim



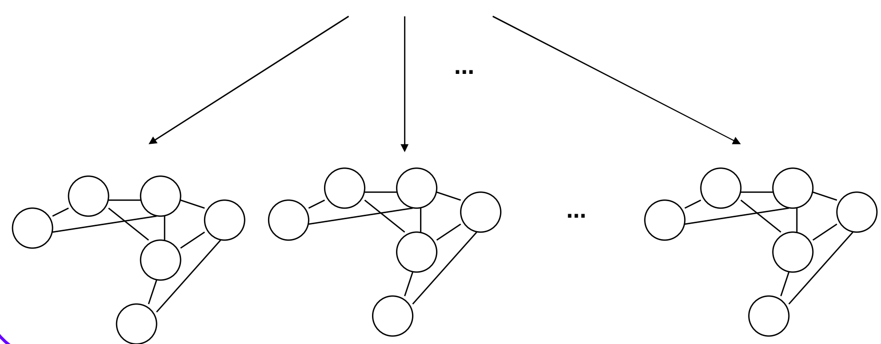
Interleaving Cond and Elim



Interleaving Cond and Elim



Interleaving Cond and Elim



Time-space Tradeoffs

- AO(i): dfs searche of AO, caches i-context
  - i = max number of variables in a context
- *AO i-cutset*
- Elimination-conditioning(i)

i=0 i i=w*

Space: O(n)

Time: O(exp(w* log n))

Space: O(exp(i))

Time: O(exp(m(i)+i)

m(i) - AO depth of w-cutset

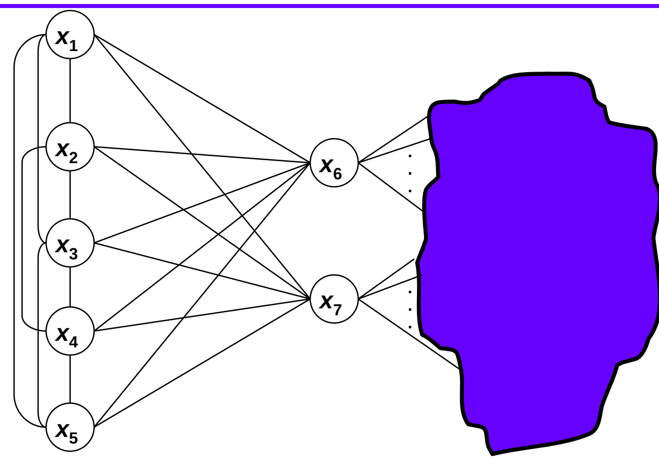
Space: O(exp w*)

Time: O(exp w*)

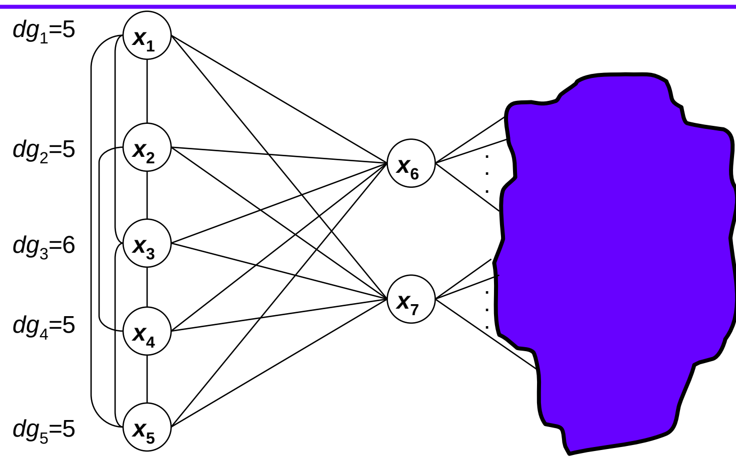
Super-Bucket Elimination, SBE(k)

- Eliminate *sets of variables* such that:
  - individual eliminations are too costly in space (namely, each variable in the set has degree larger than *k*)
  - the join degree is lower than *k*

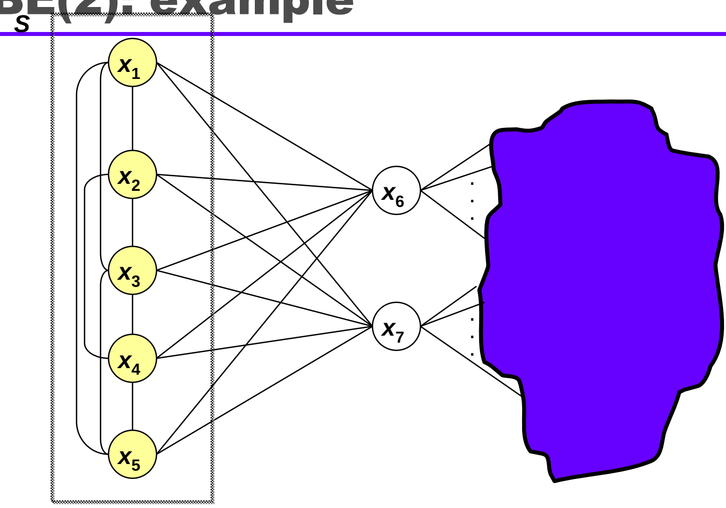
SBE(2): example



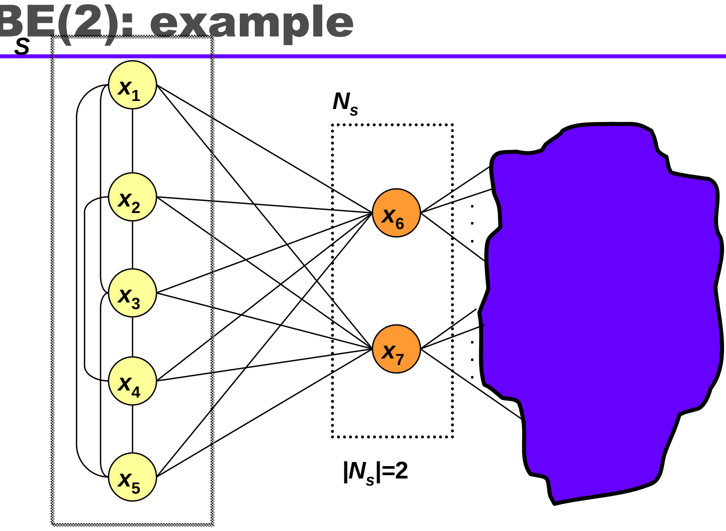
SBE(2): example



SBE(2): example

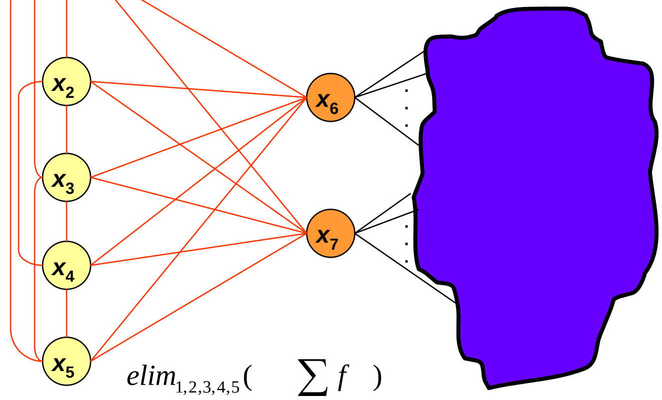


SBE(2): example



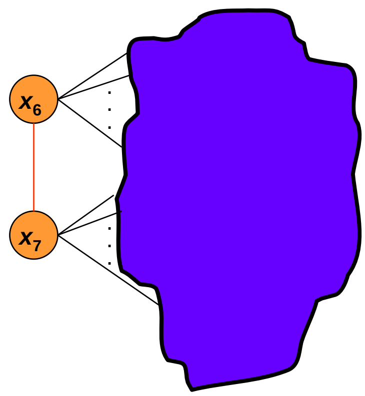
SBE(2): example

Super bucket: set of functions mentioning variables in the set ***S***



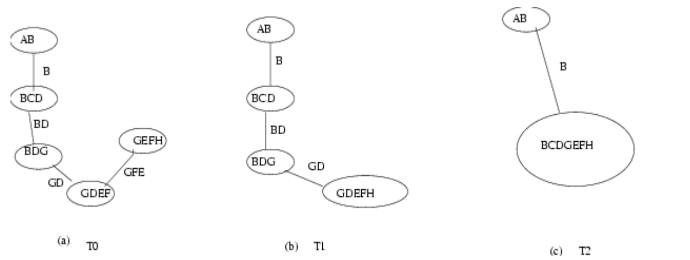
$$elim_{1,2,3,4,5}\left(\sum_{f \in super-bucket} f\right)$$

SBE(2): example



Super-buckets and Super-Clusters

Larger super-buckets (cliques) =>more time but less space



Complexity:

1. Time: exponential in clique (super-bucket) size
2. Space: exponential in separator size

SBE(i)

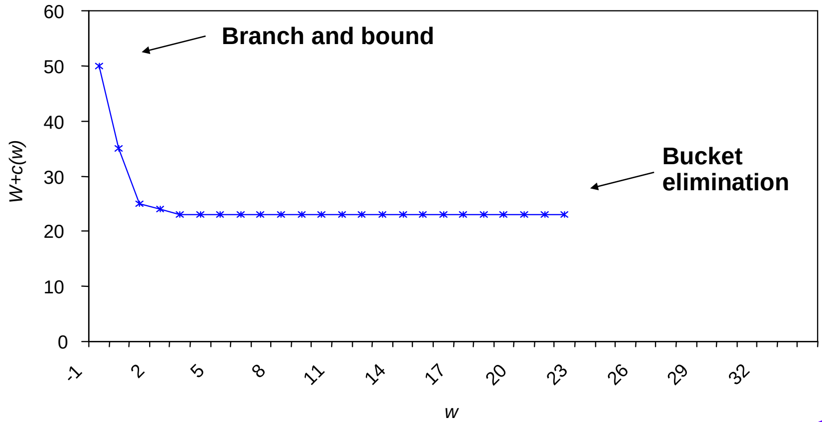
- Complexity:
  - **<u>space</u>: O(exp(i))**

    Separator-width
  - **<u>time</u>: O(exp(w_i*))**

    i-augmented induced width

Time vs Space for w-cutset

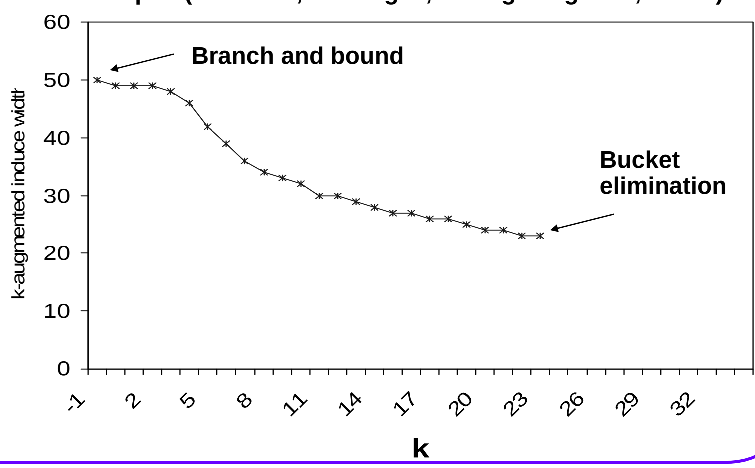
• **Random Graphs (50 nodes, 200 edges, average degree 8, *w**≈23)**



August 2005 Ijcai-05 - Principles 35

Time vs Space in SBE

• **Random Graphs (50 nodes, 200 edges, average degree 8, *w**≈23)**



August 2005 Ijcai-05 - Principles 36

Time-space Tradeoffs

- AO(i): dfs searche of AO, caches i-context
  - i = max number of variables in a context
- *AO i-cutset*
- Elimination-conditioning(i)

i=0   i   i=w*

Space: O(n)

Time: O(exp(w* log n))

Space: O(exp(i))

Time: O(exp(m(i)+i)

m(i) - AO depth of w-cutset

Space: O(exp w*)

Time: O(exp w*)

August 2005   Ijcai-05 - Principles   37

Time-space Tradeoffs

- AO(i): dfs searche of AO, caches i-context
  - i = max number of variables in a context
- *AO i-cutset*
- Elimination-conditioning(i)
- AO*(i) avoiding dead-end caches = SBE(i)
- i controls space

i=0   i   i=w*

Space: O(n)

Time: O(exp(w* log n))

Space: O(exp(i))

Time: O(exp(m(i)+i)

m(i) - AO depth of w-cutset

Space: O(exp w*)

Time: O(exp w*)

August 2005   Ijcai-05 - Principles   38

Hybrids in State Models

- Search = Best-first or Depth-first
- Inference = estimating v(n) for each node through h(n)
- Inference = search from goal to start in a relaxed model
- Reinforcement learning = improving heuristic during search
- Transposition tables = caching goods.

Some Connections
Inference, Search and Compilation

- Minimal OR space = OBDD
- Minimal AND/OR space = Tree-OBDD

Wrap-Up

- Models: State-models, graphical models
  - Graphical models are more informed
  - Tasks: satisfaction, optimization, counting/belief
- Algorithms: Inference, Search, Hybrids
- Inference requires space
- Search can trade space for time, naturally
- State-of-the art: anytime hybrid methods that trade space and time

Wrap-Up (continue)

- Algorithmic principles:
  - DFS vs BFS or inference
  - Constraint propagation, bounded inference, learning heuristics from relaxed models
  - Backjumping, AND/OR search space
  - no-good-caching, good-caching
  - Good-caching and transposition tables
  - Reinforcement learning = learning goods and no-goods during search while searching