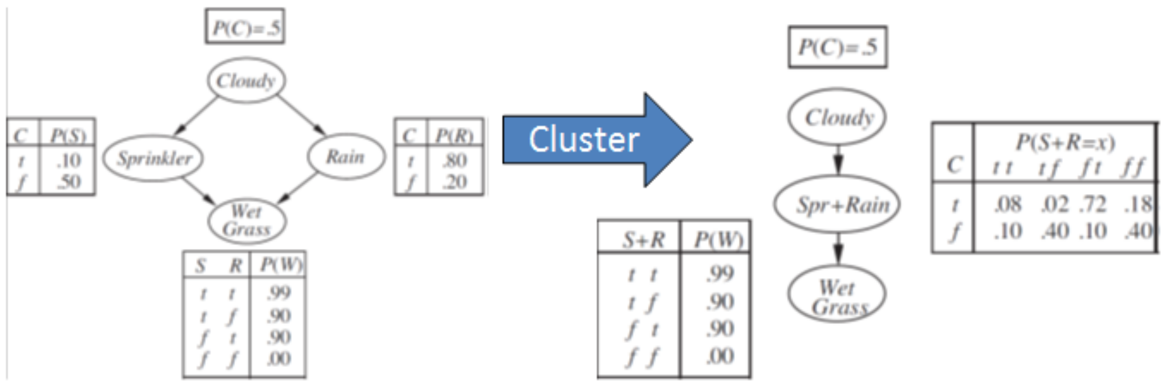
Title: **The Impact of AND/OR Search On Constraint Satisfaction and Counting**
Authors: Dechter and Mateesa, CP 2004, pages 731-736, 2004
Speaker: Robert Woodward
Scribe: Olufikayo Adetunji
Date: 03/04+06/2013

Background:

- *Goal of the paper*: Paper introduces AND/OR search spaces in graphical models, discusses compacting by merging subtrees, and explains how to use them for solution counting.
- *Applicability of the approach*: Graphical models (which include: constraint networks, Bayesian networks, Markov random fields, cost networks, and influence diagrams).

Question (Daniel G.): Are constraint networks a subset of reasoning graphical models or just an alternate name? In what ways do they differ? How do AND/OR graphs of constraint networks differ from AND/OR graphs of reasoning graphical models?

Answer: Graphical networks are general and constraint network is just one of the types of graphical networks, and so are Bayesian networks. Below is an example of a Bayesian network:



A multiply connected network with conditional probability tables

A clustered equivalent of the multiply connected network

Example question: If the sprinkler is on, what is the probability that it is raining? AND/OR graphs should be applicable to many graphical models if not all. Dechter has applied them to CSP and BN. This lecture focuses on CSPs.

Introduction (Slide 2):

The algorithms used to answer questions on graphical models can be classified into two categories: Inference-based and search-based algorithms

- *Inference-based methods*: include variable elimination, bucket elimination, tree-clustering, etc. Typically, they are exponential time and space in the tree-width. (Time: we need to solve the subproblems. Space: we need to store the unique constraint on the separators variables).

- *Search-based methods:* Like BT Search (which is an OR search-space). Space requirement is linear (because we store only the current path), and time requirement is exponential (CSPs are NP-complete) in terms of the variables.

AND/OR search spaces try to combine these two methods in order to exploit independencies during search to explore paths in parallel. In other words, in addition to the search methods of the OR space, the AND/OR search-space can also exploit independencies in the graph model to bound the space requirements.

Intuition on AND/OR Search (Slide 3):

Consider the constraint graph of a graph-coloring problem below. Fig.1 shows the constraint graph; Fig. 2 the OR search space; and Fig. 3 the AND/OR search space. Notice that variables Y and Z are independent of each other. If we considered them sequentially in backtrack search, we may have to consider big jumps back upon backtracking. The AND/OR search will try to avoid this situation by considering them independently in an OR node. In the AND/OR search space:

- *The OR node* represents a CSP variable.
- *The AND node* represents all the possible instantiations of the parent variable, yielding variable-value pairs (vvp's).

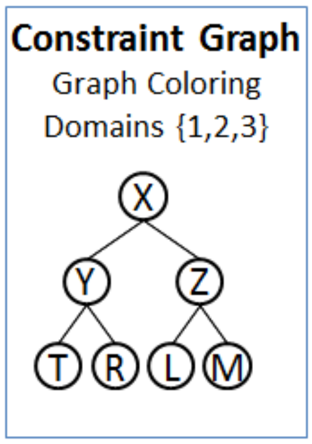


Fig. 1: CSP.

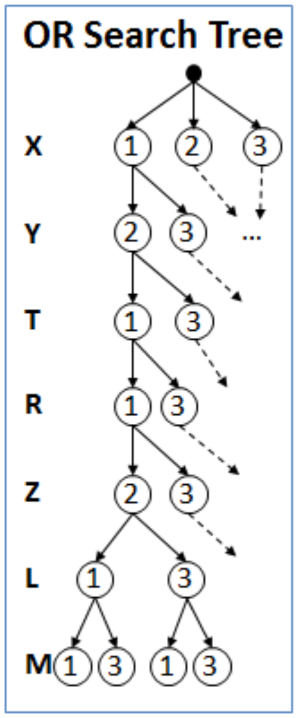


Fig. 2: OR search space.

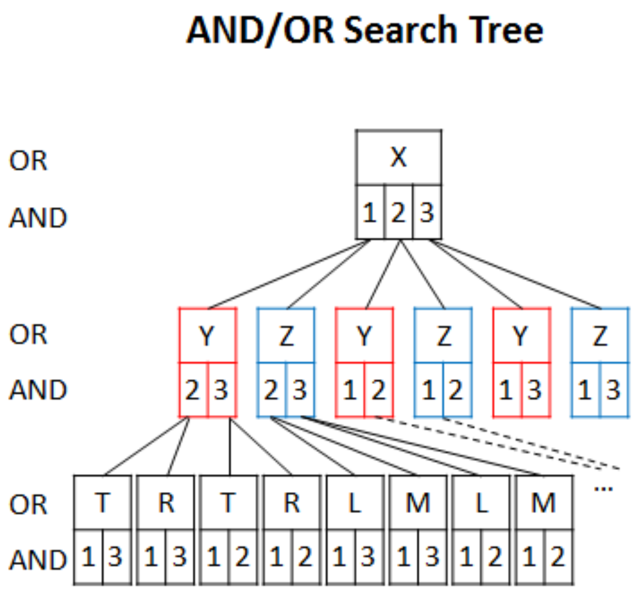


Fig. 3: AND/OR search.

Terminologies (*Slide 5*):

Below we recall some general terminology (see AI course):

State Space: is the set of all possible states that a system can be at.

State Graph: It is a directed graph showing all states and transitions (directed edges) between states.

Search tree: is a tree traversal of the state graph.

We define the AND/OR search space based on that terminology.

AND/OR Search Space (Slide 6):

- 4-tuple (S,O,Sg,So)
- S: Set of states (OR states and AND states)
- O: Set of operators.
  - *The OR operator* transforms an OR state into a disjunction of other states (in the example of Fig. 3, X can be assigned 1, 2, or 3).
  - *The AND operator* transforms an AND state to a conjunction of OR states (in the example of Fig. 3, ⟨X,1⟩ transitions to the conjunction of Y and Z.
- Sg: The goal node, where the search stops with a solution
- So: The start node, which is node X from the diagram below

Terminal nodes: are those states without children and are labeled `solved` or `unsolved`.

AND/OR Search Tree (Slide 7):

Given a CSP & a DFS spanning tree T (corresponds to a static variable ordering) of its graph

- Nodes of T are either:
  - OR nodes, which represent CSP variables, or
  - AND nodes, which represent variable-value pairs (assignments to variables).
- Successor nodes in T
  - of an OR node X, are all consistent value assignments ⟨X,v⟩
  - of an AND node ⟨X,v⟩, are all children nodes of X in T.
- It helps to think of the combination of an OR node and its AND descendants as a 'meta-node' and count as one node, i.e., OR nodes do not count in the depth of the tree. See fig. 4 below:

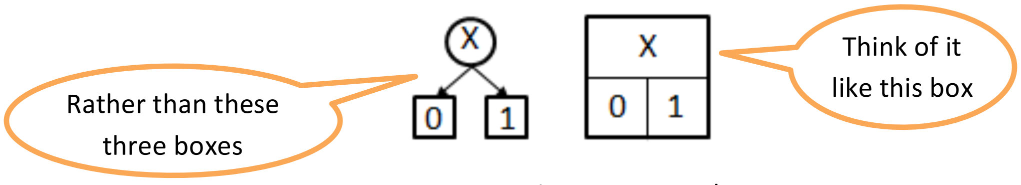


Fig. 4: Meta-node.

We all know how the OR tree in Fig. 2 is generated. Basically, we visit the variables sequentially X, Y, T, then R, etc.

For the AND/OR tree in Fig. 3, start at the same root X (as an OR node) and the values that it can be assigned (as the AND nodes) to make up a meta-node. Then, we can visit two variables Y *and* Z in parallel. For that reason, each AND node is directly linked to two OR nodes (one for Y and one for Z). Following in this manner, the AND nodes for the vvps of Y are linked to two OR nodes (for each of T and R), while the AND nodes of the vvps of Z are linked to the OR nodes to L and M.

Labeling AND/OR Search Tree Nodes (Slide 9):

- Label terminal OR nodes as `UNSOLVED/0`,
- Label terminal AND nodes as `SOLVED/1`,

- Label internal OR nodes as 1 iff *one* of its successor nodes is 1, and
- Label internal AND as nodes 1 iff *all* of its successor nodes are 1 .

Counting Solutions (Slide 10):

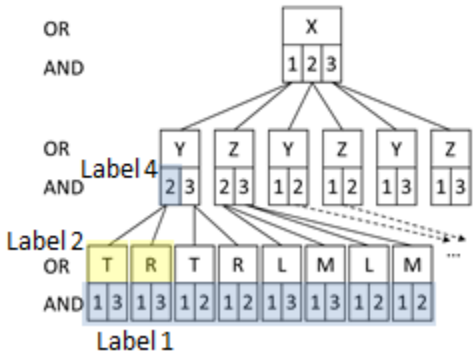


Fig. 5: Labeled tree nodes.

Solutions are counted as follows, proceedings from the leaves of the tree to its root:

- A terminal AND node is labeled 1.
- A terminal OR node is labeled 0.
- An internal AND node is labeled the product of the labels of its children.
- An internal OR node is labeled the sum of the labels of its children.
- The final value of the label at the OR root-node is the total number of solutions.

Consider Fig. 5:

- All shown terminal nodes are labeled 1.
- OR-node T is labeled 2 because it is connected to two AND nodes (i.e., ⟨T,1⟩ and ⟨T,3⟩).
- ⟨Y,2⟩ is labeled 4 because the label of the OR-node T is 2 and that of R is 2 (i.e., 2*2=4).

Legal Tree of a Constraint Graph (Slide 12):

- The legal tree is in fact the *pseudo-tree* (introduced by Freuder and Quinn in 1985).
- Given an undirected graph G = (V,E), a directed rooted tree T = (V,E') defined on all its nodes is *legal* if any edge in E that is not included in E' is a back-arc (i.e., connects a node to an ancestor).
    - o The edges in E' can be from E or not.
    - o The edges of E can be in E' or not.
    - o All the edges of E are either in E' or are back-arcs in T.
- Many legal trees of a CSP exist. A DFS of a CSP is a legal tree. Any chain is a legal tree. See examples in Fig. 6.
- Searching an OR tree corresponds to searching a chain legal-tree.

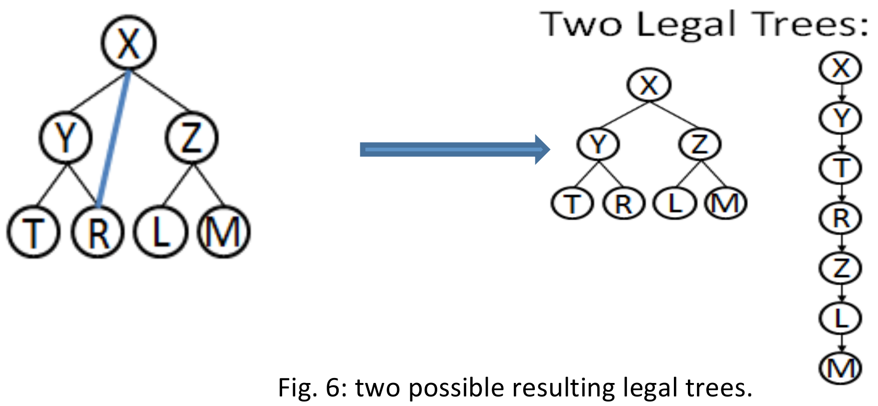


Fig. 6: two possible resulting legal trees.

Dynamic Variable Ordering (Slide 13):

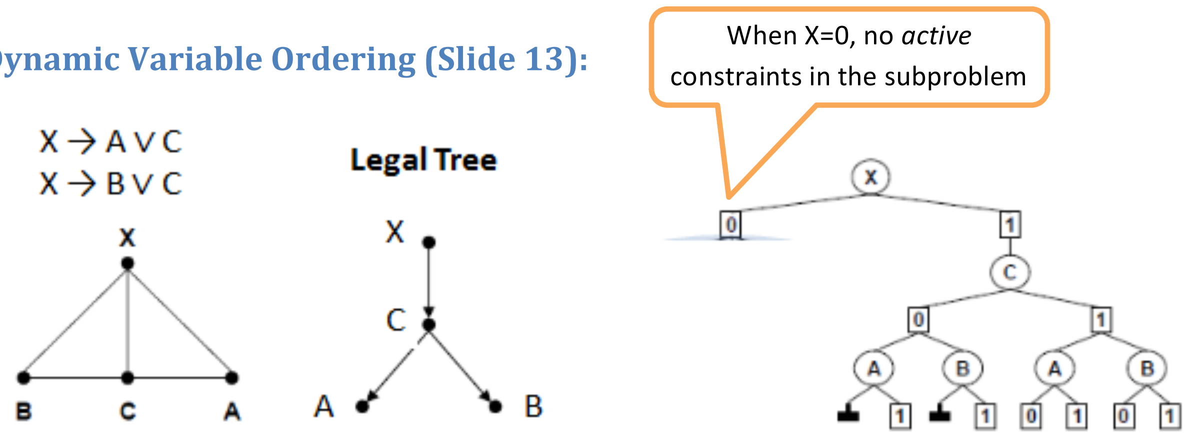


Fig. 7: Use of dynamic variable ordering.

- When X=0, the constraints are satisfied for any values of A,B, and C (0→x always holds no matter what x is). So we can work on A, B and C in parallel.
- When X=1, we have to order C first, then A and B can be put in parallel.
- The paper did not test/empirically evaluate the use of dynamic variable ordering in the AND/OR search space.

Minimal AND/OR Search Spaces (Slide 15):

- The minimal AND/OR search-space can be generated by merging OR nodes in the AND/OR tree (combining paths).
- Along two partial paths in the search-space over the same set of variables, if the search subtrees rooted at these paths are identical, the subtrees can be merged together.

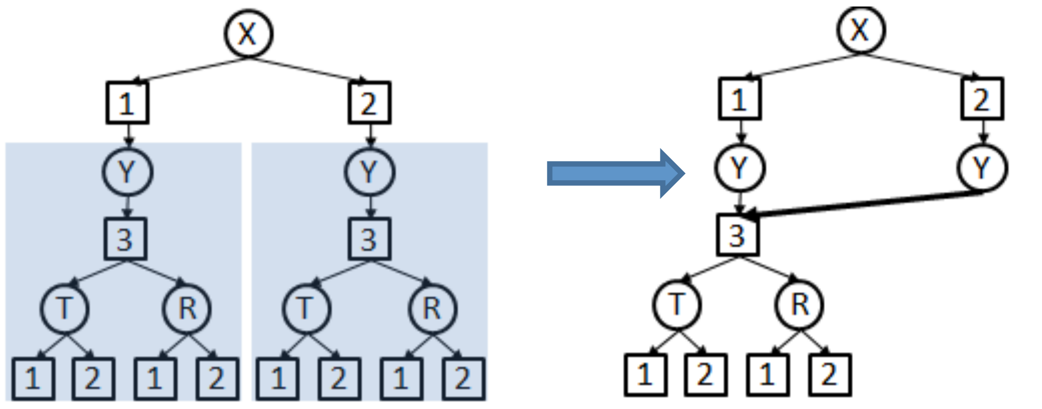


Fig. 8: Merging identical nodes.

- Fig. 8 shows an example of a merge of two identical subgraphs rooted at nodes Y.
- Merging paths always results in a unique minimal AND/OR search-graph regardless of the order of application of the merger operator.

How to Merge Nodes (Slide 16):

Merging subtrees can be done after generating all paths, which can be exponential in space. Below, we explain how to avoid generating the entire tree.

Given T, a legal tree, we denote:

- $G^{\mathcal{T}}$: the extended graph of G relative to $\mathcal{T}$, $G^{\mathcal{T}} = (V, E' \cup E)$
- $G^{*\mathcal{T}}$: the extended graph of G induced by marrying parents, relative to $\mathcal{T}$.

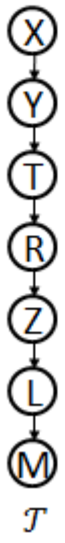


Fig. 9: Legal tree.

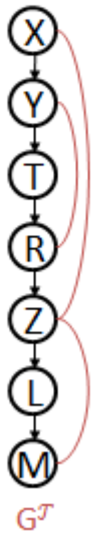


Fig. 10: $G^{\mathcal{T}}$ Extended graph of G relative to $\mathcal{T}$.

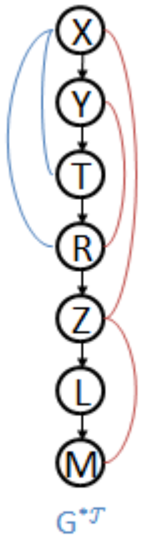


Fig. 11: $G^{*\mathcal{T}}$: Extended graph of G induced by marrying parents, relative to $\mathcal{T}$.

For example, in Fig. 11, R and X are parent of Z and married with the addition of the edge (R,X). Likewise, X and T are parent of R are married with the addition of the edge (T,X).

- Parents(ps_x): {Ancestors of X that have connections in $G^{*\mathcal{T}}$ to X or to descendants of X in $\mathcal{T}$}. In Fig. 11, the parent of T ps_T = {X, Y} because X and Y are parents of T and are also the parents of R (descendant of T).
- Parent-Separator(psa_x): {X}∪{ancestors of X that have connections in $G^{*\mathcal{T}}$ to descendants of X}. In Fig. 11, the parent-separator of T psa_T = {X, Y, T} because Y and T are parents of R (descendant of T).
- Nodes can be merged if two paths in the AND/OR graph have the same values as the parent-separator.

Example of Merging Nodes (Slide 17-19):

- In Fig. 12, there are two paths of assignment, S_1 (blue path) and S_2 (red path) ending at ⟨*X*,v⟩ (AND node with the value v assigned to some variable *X*). The matching of the ending assignment is important because it determines the possibility of a merge.
- For every state S_i, $S[psa_i]$ is called the *context* of S_i when psa_i is the parent-separators set of X_i relative to the legal tree T. In Fig. 13, S_1 and S_2 are contexts.
- It is possible to merge the similar nodes of two paths when: $S_1[psa_x]= S_2[psa_x]$

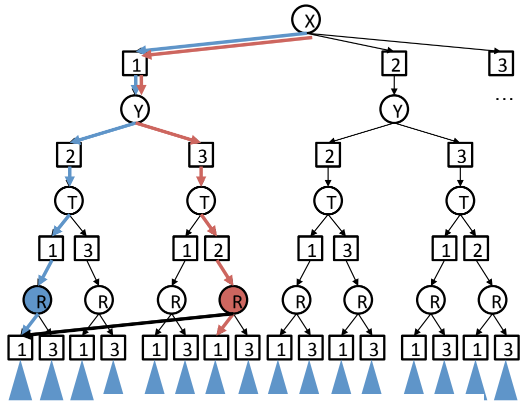


Fig. 12: two partial paths.

- In Fig. 13, S_1 and S_2 are two conditioned sub-problems (the sub-problem is conditioned on the assignment of S_1 and S_2) that need to be checked for the possibility of a merge.
- The required outcome is the knowledge of what conditions influence the appearance of the sub-trees. These conditions are parent-separators that affect the variables in the sub-problem.

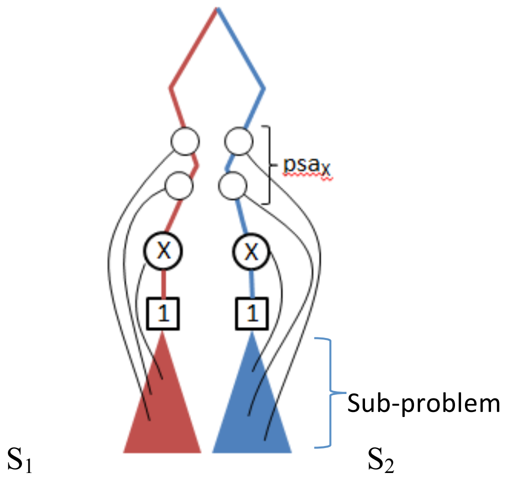


Fig. 13: Two conditioned sub-problems.

Size of AND/OR (Slide 20):

Notations:

- n: number of variables
- k: domain size
- w: induced width of G $^{\mathcal{T}}$
- m: depth of legal tree($m \leq w\log(n)$)

- The space requirement of an AND/OR tree: $O(nk^{w\log(n)})$.
- In the minimal AND/OR graph, the depth of the tree is bounded by the induced width: $O(nk^{w})$.

Solution Counting Algorithm (Slide 22):

The solution counting algorithm is in the original paper and in Robert's slides (slide 22). The algorithm builds the AND/OR tree iteratively and makes use of the structures below:

- OPEN: stores the unexplored nodes.
- CLOSE: stores the already explored nodes.
- G(n): structure used to build values (stores label of node).
- C(V): stores the context of a node.

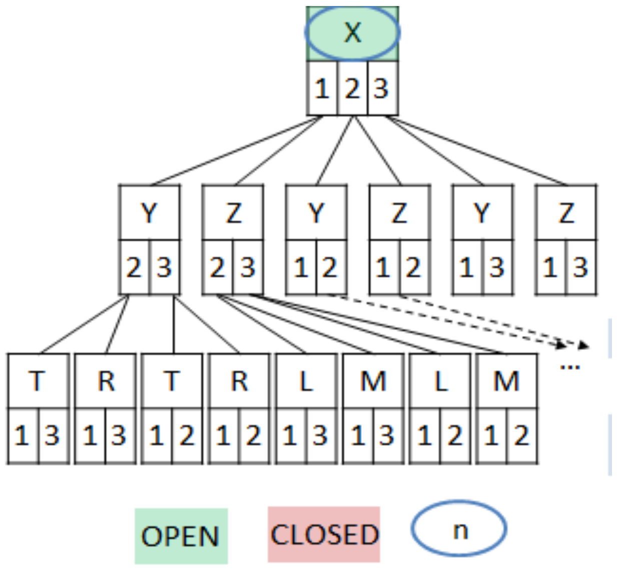


Fig. 14: AND/OR tree with labels open and closed.

Consider the diagram in Fig. 14:

1. Take the root-node of the tree and put it in OPEN.
2. Move the first node from OPEN to CLOSE.
3. Check whether that node is an AND node or an OR node. If the node has no successors/children, then end, otherwise expand it.
    - If the node has children, put them in OPEN.
4. Propagate: Run the propagation function (which counts the number of solutions rooted at a node) on a node only when all of its children have been evaluated. If it is an OR node, do the summation of the values of its children, else if it is an AND node, do the product of the values of its children. Then go back to the second step and repeat until OPEN is empty or all of its nodes are terminal nodes.

Empirical Evaluations (Slide 23):

The performances of the following algorithms were compared: AND/OR-spaces, OR-spaces, and bucket-elimination (with FC and with relational FC). The OR-search uses the same methods as the AND/OR-search, including merging. The only difference between them is that the AND/OR-search uses the DFS-tree, while the OR-search uses a chain for search.

Tests were performed on randomly generated CSPs with ternary constraints for different values of tightness and the same number of constraints. Forward checking filters values only if a variable has a constraint with just *one* future variable. Relational FC filters values even when a variable has constraints with *more than one* future variable.

- AND/OR-search outperformed OR-search, in terms of time.
- In BE, CPU time was the same for all values of constraint tightness because it takes the same amount of time to join the relations with the same scope but different tightness.

- For all algorithms, as the looseness (incorrectly called tightness in the paper) increases, the number of solutions also increases.
- In the presence of a clique-based tree, the AND/OR-search and the OR-search have the same performance, because they will both produce a chain-based clique.

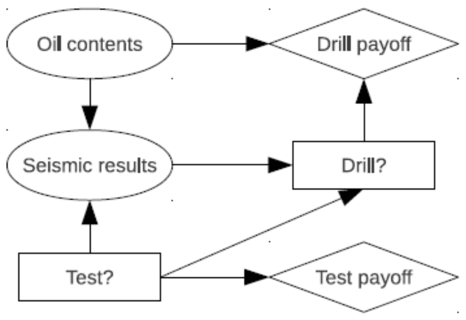
Use of Caching (Slide 25)

Caching refers to the record of no-goods. If the current node is an OR node that has no consistent successors, it is identified as a dead-end and a no-good is recorded, resulting in a new constraint. The set of constraints are modified to include this new constraint.

- Caching is not used in BE because it required too much space.
- Caching in AND/OR-search required very little time.
- Time requirement for caching in OR-search increased exponentially, with increase in tightness.
- The same results occurred for the number of nodes visited and the number of dead ends.

Extra Material (Slide 29):

- *Influence diagrams*: They are a generalization of the Bayesian network.
- *Chance* and *decision* are the two types of nodes the influence diagram can have. There is no decision-node in a Bayesian network.



Oil contents	P(O)		
	dry	wet	soak
	0.5	0.3	0.2

Test payoff	$U_1(T)$
Test?	
yes	-10
no	0

Seismic results		P(S | O,T)			
Oil cnt.	Test?	closed	open	diffuse	notest
dry	yes	0.1	0.3	0.6	0
dry	no	0	0	0	1
wet	yes	0.3	0.4	0.3	0
wet	no	0	0	0	1
soak	yes	0.5	0.4	0.1	0
soak	no	0	0	0	1

Drill payoff		$U_2(O,D)$
Oil cnt.	Drill?	
dry	yes	-70
dry	no	0
wet	yes	50
wet	no	0
soak	yes	200
soak	no	0

Fig. 15: An example of an influence diagram.

- Fig. 15 illustrates an influence-diagram example, where the goal is to make a smart decision about drilling for oil in a land area.