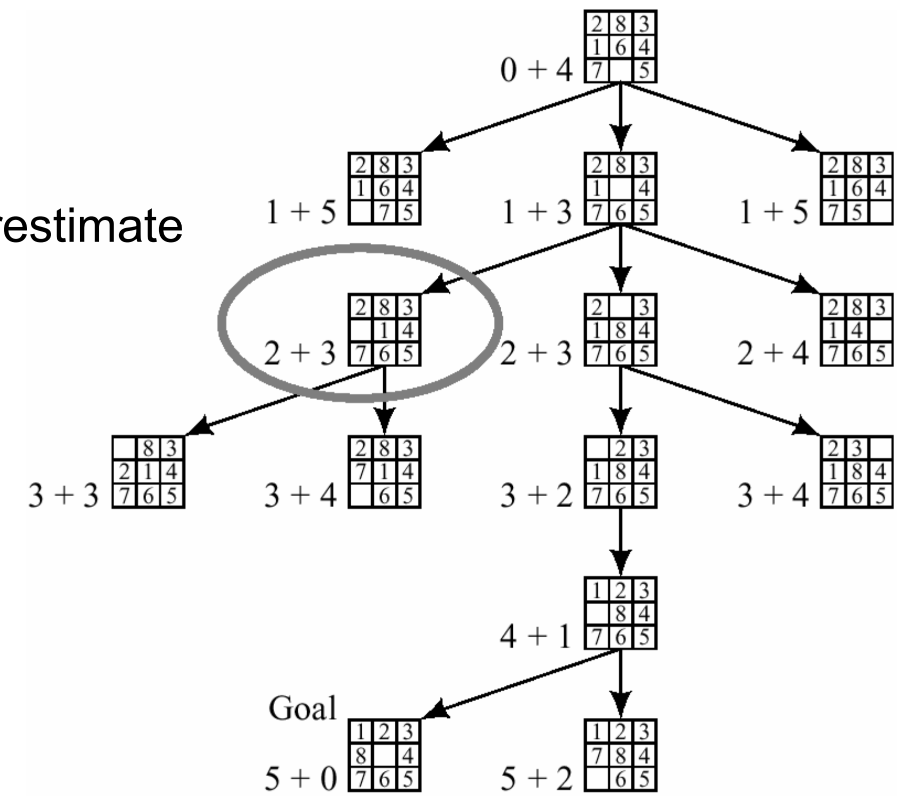
How I entered Constraints:
Some of the Early Milestones

Rina Dechter
UC-Irvine

Mechanical Heuristic generation

Observation: People generate heuristics by consulting simplified/relaxed models.
Context: Heuristic search (A*) of state-space graph (Nillson, 1980)
Context: Weak methods vs. strong methods
Domain knowledge: Heuristic function

h(n):Heuristic underestimate the best cost from n to the solution



13

THE OPTIMALITY OF A* REVISITED*

Rina Dechter and Judea Pearl

Cognitive Systems Laboratory
Computer Science Department
University of California, Los Angeles, CA 90024

ABSTRACT

This paper examines the optimality of A^*, in the sense of expanding the least number of distinct nodes, over three classes of algorithms which return solutions of comparable costs to that found by A^*. We first show that A^* is optimal over those algorithms guaranteed to find a solution at least as good as A^*'s for every heuristic assignment h. Second, we consider a wider class of algorithms which, like A^*, are guaranteed to find an optimal solution (i.e., admissible) if all cost estimates are optimistic (i.e., $h \le h^*$). On this class we show that A^* is not optimal and that no optimal algorithm exists unless h is also consistent, in which case A^* is optimal. Finally we show that A^* is optimal over the subclass of *best-first* algorithms which are admissible whenever $h \le h^*$.

1. INTRODUCTION AND PRELIMINARIES

1.1 A* and Informed Best-First Strategies

Of all search strategies used in problem solving, one of the most popular methods of exploiting heuristic information to cut down search time is the *informed best-first strategy*. The general philosophy of this strategy is to use the heuristic information to assess the "merit" latent in every candidate search avenue, then continue the exploration along the direction of highest merit. Formal descriptions of this strategy are usually given in the context of path searching problems, a formulation which represents many combinatorial problems such as routing, scheduling, speech recognition, scene analysis, and others. Given a weighted directional graph G with a distinguished start node s and a set of goal nodes Γ, the *optimal path problem* is to find a lowest cost path from s to Γ where the cost of the path may, in general, be an arbitrary function of the weights assigned to the nodes and branches along that path.

By far, the most studied version of informed best-first strategies is the algorithm A^* (Hart, Nilsson and Raphael, 1968) which was developed for *additive cost measures*, i.e. where the cost of a path is defined as the sum of the costs of its arcs. To match this cost measure, A^* employs a special additive form of the evaluation function f made up from the sum $f(n)=g(n)+h(n)$, where $g(n)$ is the cost of the currently evaluated path from s to n and h is a heuristic estimate of the cost of the path remaining between n and some goal node. A^* constructs a tree T of selected paths of G using the elementary operation of *node expansion*, i.e., generating all successors of a given node. Starting with s, A^* selects for expansion that leaf node of T which has the lowest f value, and only maintains the lowest-g path to any given node. The search halts as soon as a node selected for expansion is found to satisfy the goal conditions. It is known that if $h(n)$ is a lower bound to the cost of any continuation path from n to Γ, then A^* is *admissible*, that is, it is guaranteed to find the optimal path.

1.2 Previous Works

The *optimality* of A^*, in the sense of expanding the *least number of distinct nodes*, has been a subject of some confusion. The well-known property of A^* which predicts that decreasing errors h^*-h can only improve its performance [Nilsson, 1980, result 6] has often been interpreted to reflect some supremacy of A^* over other search algorithms of equal information. Consequently, several authors have assumed that A^*'s optimality is an established fact (e.g., Nilsson, 1971; Martelli, 1977; Méro, 1981; Barr and Feigenbaum, 1982). In fact, all this property says is that some A^* algorithms are better than other A^* algorithms depending on the heuristics which guide them. It does not indicate whether the additive rule $f=g+h$ is better than other ways of combining g and h (e.g., $f=g+h^2/(g+h)$); neither does it assure us that expansion policies based only on g and h can do as well as more sophisticated best-first policies using the entire information gathered along each path (e.g., $f(n) = \max\{f(n') \mid n'$ is on the path to $n\}$). These two conjectures will be examined in this paper, and will be given a qualified confirmation.

Gelperin (1977) has correctly pointed out that in any discussion of the optimality of A^* one should compare it to a wider class of equally informed algorithms, not merely those guided by $f=g+h$, and that the comparison class should include, for example, algorithms which adjust their h in accordance with the information gathered during the search. His analysis, unfortunately, falls short of considering the entirety of this extended class, having to follow an over-restrictive definition of *equally-informed*. Gelperin's interpretation of the statement "an algorithm B is *never more informed* than A" not only restricts B from using information unavailable to A, but also forbids B from processing common information in a better way than A does.

*Supported in part by NSF Grant No. MSC 81-142209

The Simplified models Paradigm

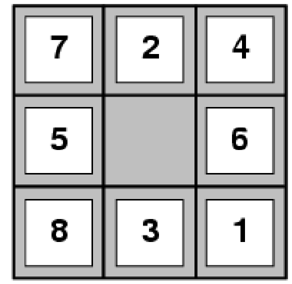
Pearl 1983 (*On the discovery and generation of certain Heuristics, 1983, AI Magazine, 22-23*) : "knowledge about easy problems could serve as a heuristic in the solution of difficult problems, i.e., that it should be possible to manipulate the representation of a difficult problem until it is approximated by an easy one, solve the easy problem, and then use the solution to guide the search process in the original problem."

The implementation of this scheme requires three major steps:
a) simplification,
b) solution, and
c) advice generation.

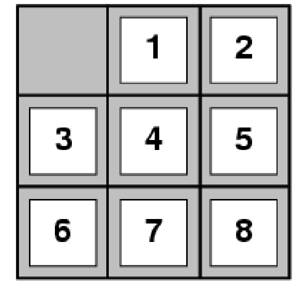
Simplified = relaxed is appealing because:
1. implies admissibility, monotonicity,
2. explains many human-generated heuristics (15-puzzle, traveling salesperson)

"We must have a simple **a-priori** criterion for deciding when a problem lends itself to easy solution."

Systematic relaxation of STRIPS



Start State



Goal State

STRIPS (**St**anford **R**esearch **I**nstitute **P**roblem **S**olver, Nillson and Fikes 1971) action representation:

Move(x,c1,c2)

Precond list: on(x1,c1), clear(c2), adj(c1,c2)

Add-list: on(x1,c2), clear(c1)

Delete-list: on(x1,c1), clear(c2)

Relaxation (Sacerdoti, 1974): Remove literals from the precondition-list:

1. clear(c2), adj(c2,c3) → #misplaced tiles
2. Remove clear(c2) → manhatten distance
3. Remove adj(c2,c3) → h3, a new procedure that transfer to the empty location a tile appearing there in the goal

But the main question remained:

“Can a program tell an easy problem from a hard one without actually solving?” (Pearl 1984, Heuristics)

Easy = Greedily solved?

Pearl, 84: Most easy problems we encounter are solved by **“greedy” hill-climbing methods without backtracking”** and that the features that make them amenable to such methods is their “decomposability”

The question now:
Can we recognize a greedily solved STRIPS problem?”

On the greedy solution of ordering/scheduling problems

Job-shop: minimizing weighted average flow-time on a single processor

$$C(1,2,...n) = \sum_{i=1} q_i \sum_{j=1}^{i} p_j$$

Spanish treasure problem: An unknown number of chests of Spanish treasure have been buried on a random basis in n sites. For each site there is a probability p_i that the chest is there and the cost of excavating a site is q_i. Find a sequence of excavations that will minimize the average cost of finding the first chest.

Greedy strategy: $\frac{q_i}{p_i}$

So, the question now: Can we characterize when does an ordering problem has a **ranking function**, or a **greedy rule**, (not necessarily using the cost function) that yields an optimal solution?

The result is in my thesis (1985) and later in a paper (1989) "On the greedy solution of ordering problems" ORSA Journal of Computing, Vol 1, No. 3, 1989. A paper which is largely un-cited and unknown

On the greedy solution… (continued)

Theorem: If P is any greedily optimized problem then an optimizing ranking function f has to agree with the ordering dictated by the cost function on pairs of elements.
Namely for every two elements a and b
C(a,b) > C(b,a) iff f(a) > f(b).

$$C(\sigma) - C(\sigma') = (u_{i+1}p_i - u_i p_{i+1}) = u_{i+1}u_i\left(\frac{p_i}{u_i} - \frac{p_{i+1}}{u_{i+1}}\right). \tag{8}$$

The theory explained all known greedy rules for ordering problems.

Conclusion: "The paper provides necessary and sufficient conditions for a problem to be greedily optimized by a uniform ranking function. The virtue of these conditions is that they are easy to test and thus may be useful in mechanizing the process of generating greedy strategies by computers."

ORSA Journal on Computing
Vol. 1, No. 3, Summer 1989

0899-1499/89/0103-0181 $01.25

On the Greedy Solution of Ordering Problems

AVI DECHTER *Department of Management Science, California State University, Northridge, CA 91330*

RINA DECHTER *Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, CA 90024*

(new paper in EJOR, 2001 "Greedy Solutions of selection and ordering Problems")

(Received: January 1988; final revision received: September 1989; accepted: February 1989)

The greedy method is a well-known approach for problem solving directed mainly at the solution of optimization problems. Leading theoretical frameworks dealing with the optimality of greedy solutions (e.g., the matroid and greedoid theories) tacitly assume that the greedy algorithm is always guided by the cost function to be optimized, namely, it builds a solution by adding, in each step, an element that contributes the most to the value of the cost function. This paper studies a class of problems for which this type of a greedy algorithm does not optimize the given cost function, but for which there exists a secondary objective function, called a greedy rule, such that applying the greedy algorithm to the secondary objective function yields a solution which is optimal with respect to the original cost function.

The greedy method is a well-known approach for problem solving directed mainly at the solution of optimization problems involving the selection and/or the ordering of elements from a given set so as to maximize or minimize a given objective function. Nilsson[9] views the greedy algorithm as an irrevocable (i.e., without backtracking) search strategy that uses local knowledge to construct a solution in a "hill climbing" process. The greedy control strategy selects the next state so as to achieve the largest possible improvement in the value of some measure that, as pointed out by Horowitz and Sahni,[4] may or may not be the objective function.

Our interest in greedy methods originated in earlier research in the area of heuristic problem solving.[2] The connection between these two subjects is twofold. First, greedy schemes are probably the closest to emulate human problem-solving strategies because they require only a minimum amount of memory space and because they often produce adequate (if not optimal) results. (Due to the small size of human short-term memory, it is very hard to conceive of a human conducting best-first or even backtracking search, both requiring retention of some properties of previously suspended alternatives.) Second, **greedily optimized** problems (i.e., for which a greedy algorithm produces optimal solutions) represent a class of relatively **easy** problems. Pearl[10] has demonstrated that many heuristics used in the solution of hard problems are based on simplified models of the problem domain, which admit easy solutions. Therefore, the ability to characterize classes of easy problems is important, particularly if the process of discovering heuristics is to be mechanized.

Subject classification: Analysis of algorithms: optimality of greedy algorithms.

This paper is concerned exclusively with **ordering problems**, involving a set of elements and a cost function defined on all permutations of the elements, where the task is to order the elements so as to maximize (or minimize) the value of the cost function. Job sequencing on a single machine and the traveling salesman problems are two examples of this class of problems.

A theoretical framework, called greedoid theory, which characterizes a class of ordering problems that can be solved optimally by greedy algorithms, is due to Korte and Lovasz.[6] The greedoid structure is a generalization of the matroid structure which provides a theoretical foundation for the optimality of the greedy algorithm **on selection problems**. (In contrast with ordering problems, selection problems involve a set of elements and a cost function defined on all **unordered** subsets of elements, where the task is to select a subset of elements which satisfies some property, so as to maximize (or minimize) the value of the cost function. The minimum weight spanning tree problem is a well known example of this class of problems. For further details on matroids refer, for example, to Lawler[8] or Welsh[12]).

The greedoid theory (as well as the matroid theory) considers only greedy algorithms that use the cost function to be optimized as their selection criterion, namely, which build the solution by adding, at each step, that element which results in maximum improvement in the value of that cost function. The appendix to this paper lists a number of known ordering problems for which this greedy algorithm does not optimize the cost function, but for which there exists a **secondary objective function**, which we call a **greedy rule**, such that

Freuder, JACM 1982 : “A sufficient condition for backtrack-free search”

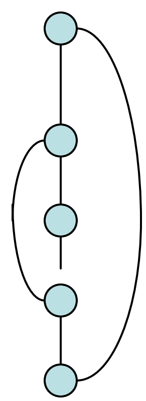
Whow! Backtrack-free is greedy!
I read Montanari (1974),I read mackworth, (1977)
Got absorbed…

Sufficient condition (Freuder 82):

1. Trees (width-1) and arc-consistency implies backtrack-free
2. Width=i and (i+1)-consistency implies backtrack-free search

If 3-consistent
no deadends

W=2



W=1

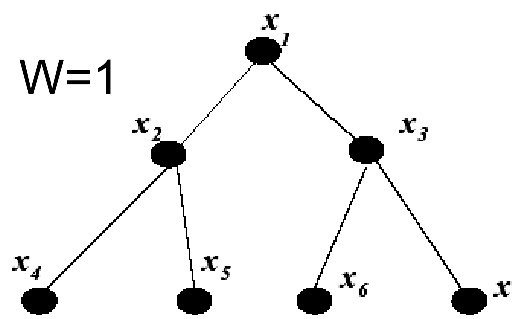


Figure 4.10: A tree network

Arc-consistent
No dead-ends

Else, impose consistency, but it add arcs except for trees. So trees are easy.

Freuder, JACM 1982 : “A sufficient condition for backtrack-free search”

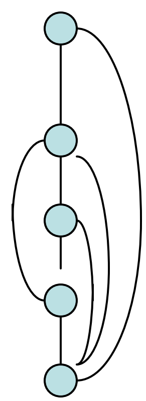
Whow! Backtrack-free is greedy!
I read Montanari (1974),I read mackworth, (1977)
Got absorbed

Sufficient condition (Freuder 82):

1. Trees (width-1) and arc-consistency implies backtrack-free
2. Width=i and (i+1)-sonsistency implies backtrack-free search

If 3-consistent
no deadends

W=2



W=3

W=1

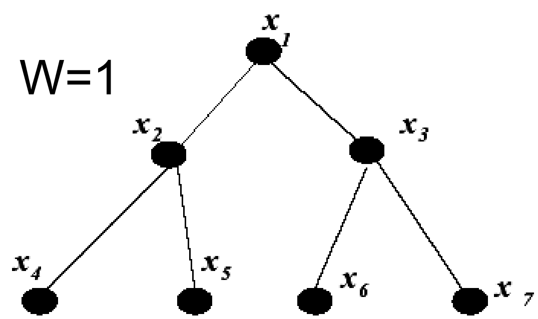


Figure 4.10: A tree network

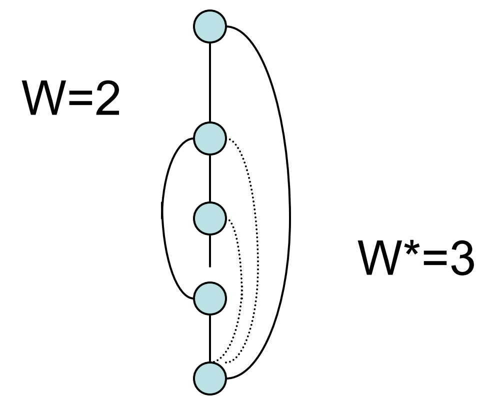
Arc-consistent
No dead-ends

Else, impose consistency, but it add arcs except for trees. So trees are easy.

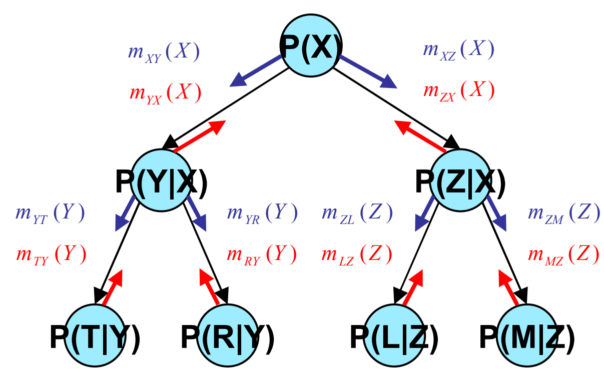
From width to induced-width

Dechter and Pearl, 1985:
if a problem has **induced-width** i it can be solved by **directional** i-c

Led to:
Directional-consistency,Adaptive-consistency,
Join-tree clustering and treewidth
Optimization (dynamic programming),
Counting, all solved by a single Algorithm,
with induced-width complexity.
Later, generalized to bucket-elimination for
probablistic reasoning, Later to cycle-cutset.



It is all based on generalizing **trees**:
(Macworth, Freuder, 85, Pearl 83)



Back to Automatic generation of heuristics… for CSPs

I did not abandon the general goal of heuristic generation. Just shifted to CSP where heuristic indicate existance of a solution. Or, alternatively, how many solutions are below a given node. Since trees are easy for counting, I relaxed the problem into a tree and… count (Dechter, 1985).

Results in 1985
On random 15 vars, 5 vals:

We revisited this idea with Kask, Gogate and Dechter (CP, 2004) estimated counts using GBP/IJGP.

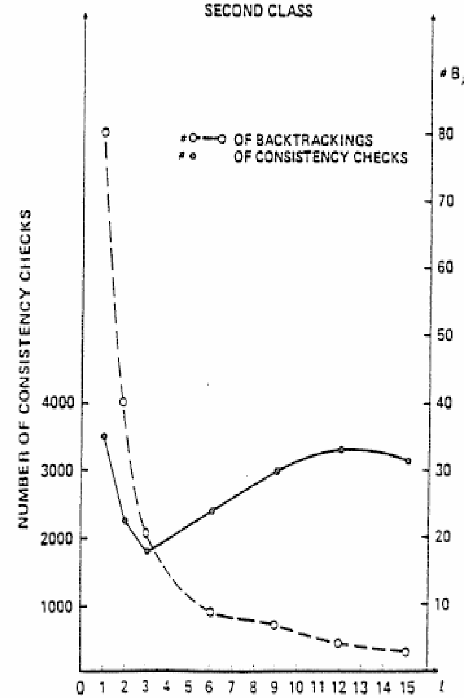


FIG. 16. The number of consistency checks and the number of backtrackings with parameterized advice (second class of problems).

From Then On…

(just tried to understand what was going on around me)

Backjumping and no-good learning (1987-88)
(Wanted to understand TMS and Logic programming)
Sat-based Nonmon-reasoning (with **Ben-Eliyahu, 1990)**
(answer-set programming)
(Wanted to understand default logic, logic programming)
Temporal constraint networks (with **Meiri** and Pearl**, 1988-90)**
(Understanding what Dean and Macdermoth and James Allen were doing)
Distributed constraints (with **Collin and Katz, 1990)**
Neural networks hyped up again.
On the expressiveness of networks with hidden variables(Dechter 1990),
from local to global consistency (Dechter 1992)
Neural networks (will explain)
Identifiability of structures (trees) from relations (with **Meiri and Pearl, 1990)**
Learnability / PAC learning.
Bucket-elimination (Dechter, 96) (bringing treewidth/induced-width
to Bayesian networks)
Understanding probabilistic reasoning through VE
Mini-buckets, (with **Rish 1997)** finally Generating heuristics for real
(with **Kask, Marinescu, 2001, 2004)**
AND/OR search (with **Mateescu, 2004)**

From Then On…

(just tried to understand what was going on around me)

Backjumping and no-good learning (1987-88)
(Wanted to understand TMS and Logic programming)
Sat-based Nonmon-reasoning (with **Ben-Eliyahu, 1990)**
(answer-set programming)
(Wanted to understand default logic, logic programming)
Temporal constraint networks (with **Meiri** and Pearl**, 1988-90)**
(Understanding what Dean and Macdermoth and James Allen were doing)
Distributed constraints (with **Collin and Katz, 1990)**
Neural networks hyped up again.
On the expressiveness of networks with hidden variables(Dechter 1990),
from local to global consistency (Dechter 1992)
Neural networks (will explain)
Identifiability of structures (trees) from relations (with **Meiri and Pearl, 1990)**
Learnability / PAC learning.
Bucket-elimination (Dechter, 96) (bringing treewidth/induced-width
to Bayesian networks)
Understanding probabilistic reasoning through VE
Mini-buckets, (with **Rish 1997)** finally Generating heuristics for real
(with **Kask, Marinescu, 2001, 2004)**
AND/OR search (with **Mateescu, 2004)**

ON THE EXPRESSIVENESS OF NETWORKS WITH HIDDEN VARIABLES

Rina Dechter [1]

Computer Science Department
Technion -- Israel Institute of Technology
Haifa, Israel, 32000
e-mail: dechter@techsel.bitnet

Abstract

This paper investigates design issues associated with representing relations in binary networks augmented with hidden variables. The trade-off between the number of variables required and the size of their domains is discussed. We show that if the number of values available to each variable is just two, then hidden variables cannot improve the expressional power of the network, regardless of their number. However, for $k \geq 3$, we can always find a layered network using k-valued hidden variables that represent an arbitrary relation. We then provide a scheme for decomposing an arbitrary relation, ρ, using $\frac{|\rho|-2}{k-2}$ hidden variables, each having k values ($k>2$).

1. Introduction

Hidden units play a central role in connectionist model, without which the model would not represent many useful functions and relations. In the early days of the Perceptrons [Minsky 1969] it was noted that even simple functions like the *XOR* were not expressible in a single layer perceptron; a realization that slowed research in the area until the notion of hidden units had emerged [Rumelhart 1988a, Hinton 1988]. Nevertheless, a formal treatment of the expressiveness gained by hidden units, and systematic schemes for designing systems with hidden units within the neural network paradigm are still not available.

Our intention is to investigate formally the role of hidden units and devise systematic schemes for designing systems incorporating hidden units. Specifically, we address the following task: given a relation on n variables, called visible, we wish to design a network having $n+h$ units whose stable patterns, (relative to the visible units) coincide with the original relation. This task is central to most applications of connectionist networks, in particular to its role as associative memory. The task will be investigated for a connectionist architecture which is different from classic connectionist networks in that it is based on **constraint networks.** The sequential constraint network model is defined next.

(1) This research was supported in part by NSF grant #IRI-8821444 and by an Air Force grant #AFOSR 88 0177 while the author was visiting the cognitive systems lab at UCLA.

A Network of binary constraints involves a set of n variables $X_1,...,X_n$, each represented by its domain values, $D_1, \ldots, D_n$, and a set of constraints. A **binary constraint** R_{ij} between two variables X_i and X_j is a subset of the cartesian product $D_i \times D_j$ that specifies which values of the variables are compatible with each other. A solution is an assignment of values to all the variables which satisfy all the constraints, and the **constraint satisfaction problems (CSP)** associated with these networks is to find one or all solutions. A binary CSP can be associated with a **constraint-graph** in which nodes represent variables and arcs connect pairs of variables which are constrained explicitly. Figure 1a presents a constraint network where each node represents a variable having values $\{a, b, c\}$ and each link is associated with a strict lexicographic order (where $X_i < X_j$ iff $i < j$). (The domains and the constraints explicitly indicated on some of the links.)

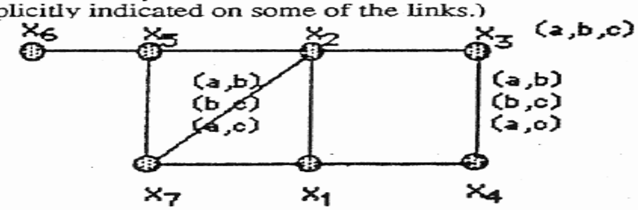


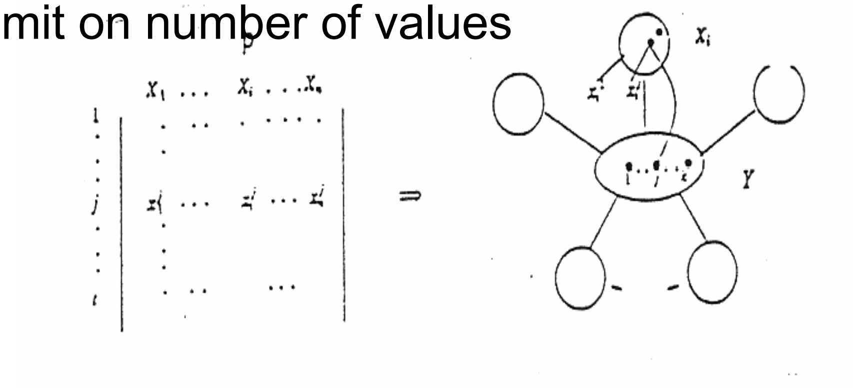
Figure 1: An example of a binary CN

Our constraint-based connectionist architecture assumes that each unit plays the role of a variable having k states, and that the links, representing the constraints, are quantified by compatibility relations between states of adjacent units. Each unit asynchronously updates its state

On the expressiveness of networks with hidden variables

Can a relation be expressed by a binary constraint networks with hidden variables?

Yes. If no limit on number of values



And, with limit?

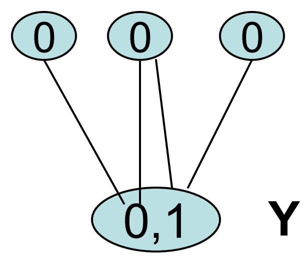
$$U_5 = \left\{ \begin{array}{ccccc} X_1 & X_2 & X_3 & X_4 & X_5 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right\}$$

With 2 values?
With 3 values?
How many hidden
Variables?

On the expressiveness of networks with hidden variables

Theorem: Relations which are not binary network decomposable cannot be binary network decomposable by adding **any number of bi-valued** Hidden variables.
Proof: We want to exclude (x1,x2,x3)=(0,0,0) using a variable Y={0,1}. …

$$U_5 = \left\{ \begin{array}{ccccc} X_1 & X_2 & X_3 & X_4 & X_5 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right\}$$



So, it is not possible that Y allows any pair but not triplets.

Reason: 3-consistent bi-valued binary networks are globally consistent

Theorem (Dechter, 1992): k-valued binary networks which are strong (k+1)-consistent are globally consistent.

However, No simple criterion for tractability emerged;
Semantic based tractability: row-convex constraints (van Beek 1995)
A whole major line of work by Jeavons and Cohen (1995-2007)
(Constraint book, chapter 10, 2003)

On the expressiveness of networks with hidden variables

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Z_1	Z_2	Z_3	T_1
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0	2	0	0	1
0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	2
0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	1	0	2
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	2	0	2
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2	0	0	0	2	0	2
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	2	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	0	0	2	0

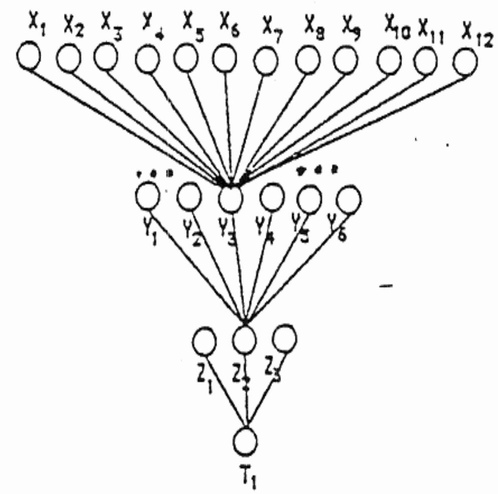


Figure 6: A layered decomposition of U^*_{12}

From Then On

(just tried to understand what was going on around me)

Backjumping and no-good learning (Dechter, 1987-88)
(Wanted to understand TMS and Logic programming)
Sat-based Nonmon-reasoning (with **Ben-Eliyahu, 1990)**
(answer-set programming)
(Wanted to understand default logic, logic programming)
Temporal constraint networks (with **Meiri** and Pearl**, 1988-90)**
(Understanding what Dean and Macdermoth and Allen James were doing)
Distributed constraints (with **Collin and Katz, 1990)**
Neural networks hyped up again.
On the expressiveness of networks with hidden variables(Dechter 1990),
from local to global consistency (Dechter 1992)
Neural networks (will explain)
Identifiability of structures (trees) from relations (Dechter, Meiri and Pearl, 1990)
Learnability / PAC learning.
Bucket-elimination (Dechter, 96) (bringing treewidth/induced-width to Bayesian networks)
Understanding probabilistic reasoning through VE
Mini-buckets, (Dechter and Rish 1997) finally Generating heuristics for real
(Kask, Marinescu, 2001, 2004)
AND/OR search (Dechter and Mateescu, 2004)

Structure Identification in Relational Data *

Rina Dechter, UC-Irvine

Judea Pearl, UCLA

*Supported in part by Air Force Office of Scientific Research, AFOSR 900136 and by NSF grant IRI-9157936

Another Example

$\rho =$
(relation)

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

$T = \{A \rightarrow B,\ B \wedge C \rightarrow D\}$
(Horn theory)

Identifying Tree Structures in Categorical Relations

(Dechter, 1990, Meiri, Dechter, Pearl, 1991)

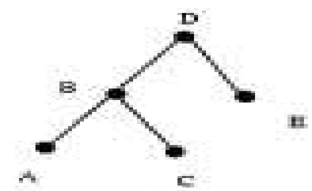
A	B	C	D	E
0	0	1	1	1
0	0	1	1	0
0	1	1	1	1
0	1	1	1	0
0	1	1	0	1
0	1	0	1	1
0	1	0	1	0
0	1	0	0	1

D	B
1	0
1	1
0	1

D	E
1	1
1	0
0	1

A	B
0	0
0	1

B	C
0	1
1	1
1	0



- A tree T **represents** a relation ρ iff

$$\rho = \bowtie_{(X_i, X_j) \in T} \rho_{X_i X_j}$$

- Convenience: storage, query processing, parallelism.

3.2 Identifying constraint trees

(unique Scheme Class)

Problem

- Given a relation ρ (e.g., as a table)
- If ρ has a tree decomposition, **find** one
- If it does not, **acknowledge** and find a **best** approximation.

Theorem 8 (Dechter, 1987): Let

$n(x_i)$ = the number of n-tuples in ρ for which $X_i = x_i$.

$n(x_i, x_j)$ = number of n-tuples in ρ for which $X_i = x_i$ and $X_j = x_j$.

The MWST algorithm using the arc-weights:

$$m(X_i, X_j) = \frac{1}{|\rho|} \sum_{(x_i, x_j) \in \rho_{X_i X_j}} n(x_i, x_j) \log \frac{n(x_i, x_j)}{n(x_i)\, n(x_j)}$$

is guaranteed to produce a tree-decomposition to ρ if such a decomposition exists.

The decomposition is exact iff $|\rho| = |\rho_T|$

Complexity: $O[(|\rho| + \log n)n^2]$

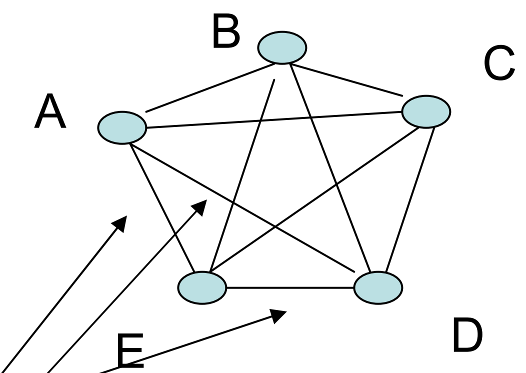
Best approximation? Open problem

Qualitative : Meiri, Dechter, Pearl (AAAI-90)

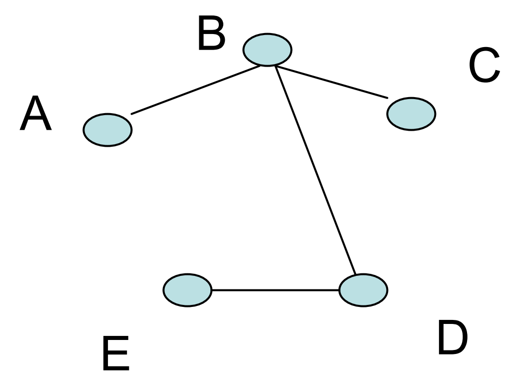
Example

A	B	C	D	E
0	0	1	1	1
0	0	1	1	0
0	1	1	1	1
0	1	1	1	0
0	1	1	0	1
0	1	0	1	1
0	1	0	1	0
0	1	0	0	1

n(A=0)=8, n(B=1)=6, n(B=0)=2
n(B=0,C=1)=2, n(B=1,C=1)=3…



m(B,C)=-13.97
mB,D)=-15.95
m(B,E)=-16.55
m(C,D)=-16.55
m(C,E)=-17.13
m(D,E)=-15.50
m(A,B)=m(A,C)=
m(A,E)=m(A,D)=-16.63



Qualitative tree decomposition
(Meiri, Dechter, Pearl, AAAI-90)

method: Given minimal network M

1. Consider all triangles $t=\{e_1, e_2, e_3\}$
   - generate labeling on arcs such that if e_1 is redundant in t
     $w(e_1) \le w(e_2)$, $w(e_1) = w(e_3)$
   - if e_1 is redundant and e_2 is not $w(e_1) < w(e_2)$

   (called triangle labeling)
2. Find a maximal weight spanning tree T w.r.t. w.

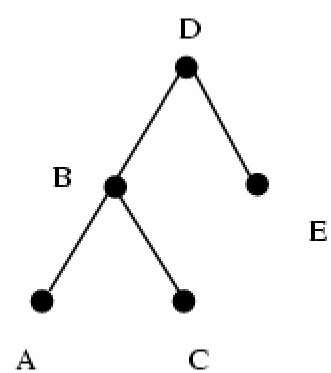
Definition: e_1 is redundant in $t=\{e_1\, e_2\, e_3\}$

iff $M_{e_1} = M_{e_2} \circ M_{e_3}$

3. Test if T decompose rel(M)
   if not there is no Tree decomposition

Identifying Structure from Relational Data (Dechter, Meiri and pearl 1988-1991)

Dechter, R., and Pearl, J., **"Structure identification in relational data."** In *Artificial Intelligence, Vol. 58,* 1992, pp. 237-270.
Meiri, I., Dechter, R., and Pearl, J., **"Uncovering trees in constraint networks."** (AAAI-1990), *Artificial Intelligence Journal*, Volume 86, 1996, pp. 245-267.
Dechter, R., "Decomposing a relation into a tree of binary relations." *Journal of Computer and System Sciences,* Vol. 41, 1990, pp. 2-24.
A preliminary version PODS pp. 185-189.



D	B
1	0
1	1
0	1

(a)

D	E
1	1
1	0
0	1

(b)

A	B
0	0
0	1

(c)

B	C
0	1
1	1
1	0

(d)

In Summary

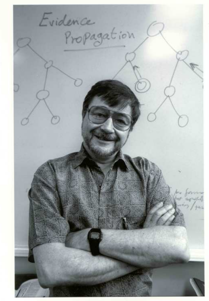
- Uncovering structure from data and how to exploit hidden variables are still central scientific questions….
- As to heuristic generation nowdays…
  - Simplification and solution steps combine (e.g., mini-bucket, heuritics for planning, using MDPs…)

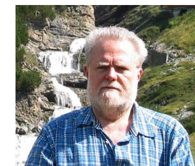
Thanks again and
Special thanks to all my students

Zeev Collin,
Itay Meiri,
Rachel Ben-Elyahoo,
Yousri El-Fattah
Dan Frost
Eddie Schwalb
Irina Rish
Kalev Kask
Bozhena Bidyuk
David Larkin
Robert Mateescu
Radu Marinescu
Vibhav Gogate
Lars Otten
Javier Larrosa

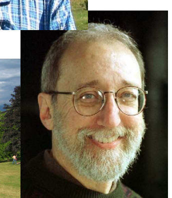
Hector Geffner

Judea Pearl





Ugo Montanari
Alan Mackworth
Eugene Freuder



My family