DM87
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 3

Mathematical Programming Formulations, Constraint Programming

Marco Chiarandini

Outline

Modeling: Mixed Integer Formulations

- Transportation Problem
- Weighted Bipartite Matching Problem (if $m = n \Rightarrow$ assignment)

Set Covering

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i \\ & x_j \in \{0,1\} \end{aligned}$$

Set Partitioning

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j = 1 \quad \forall i \\ & x_j \in \{0,1\} \end{aligned}$$

Set Packing

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \leq 1 \quad \forall i \\ & x_j \in \{0,1\} \end{aligned}$$

Traveling Salesman Problem

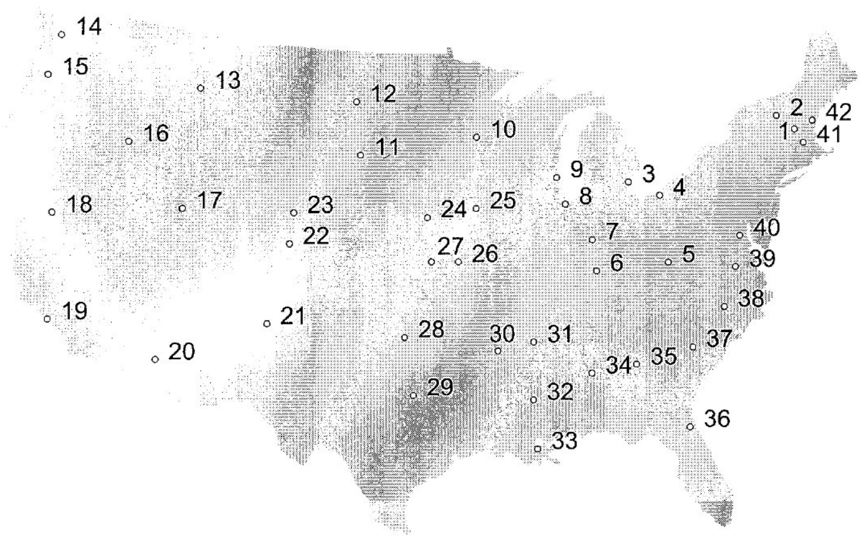


Figure 3.1 Locations of the 42 cities.

Traveling Salesman Problem

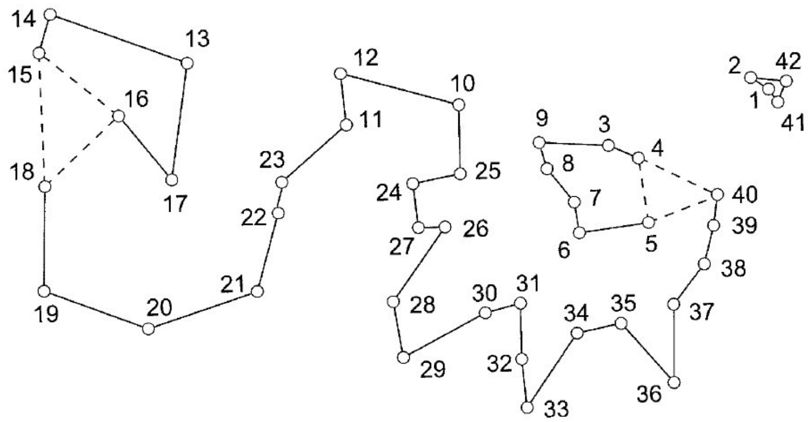


Figure 3.2 Solution of the initial LP relaxation.

Traveling Salesman Problem

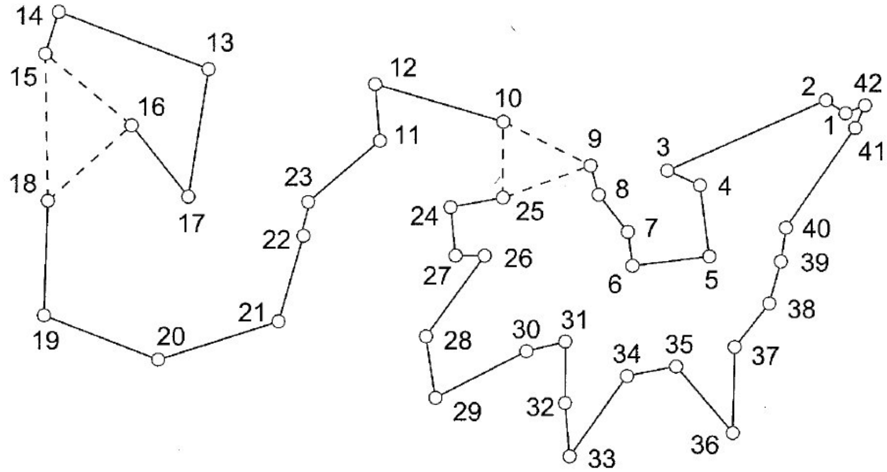


Figure 3.3 LP solution after three subtour constraints.

DM87 – Scheduling, Timetabling and Routing 4

Traveling Salesman Problem

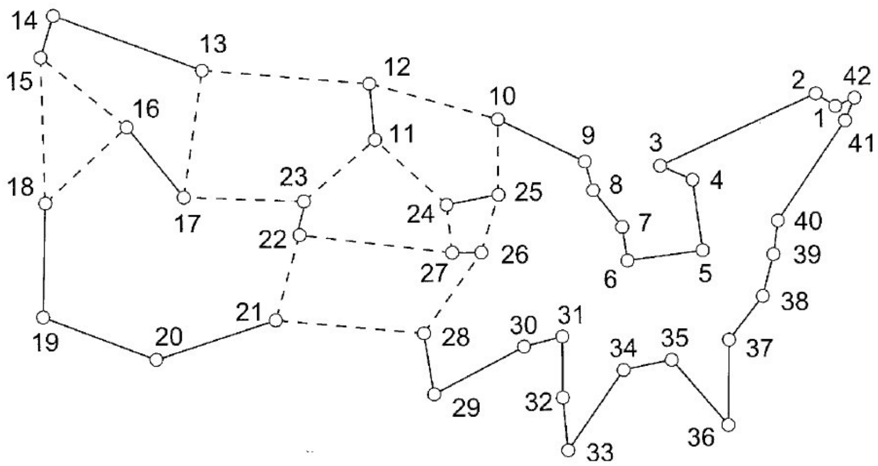


Figure 3.4 LP solution satisfying all subtour constraints.

DM87 – Scheduling, Timetabling and Routing 4

Traveling Salesman Problem

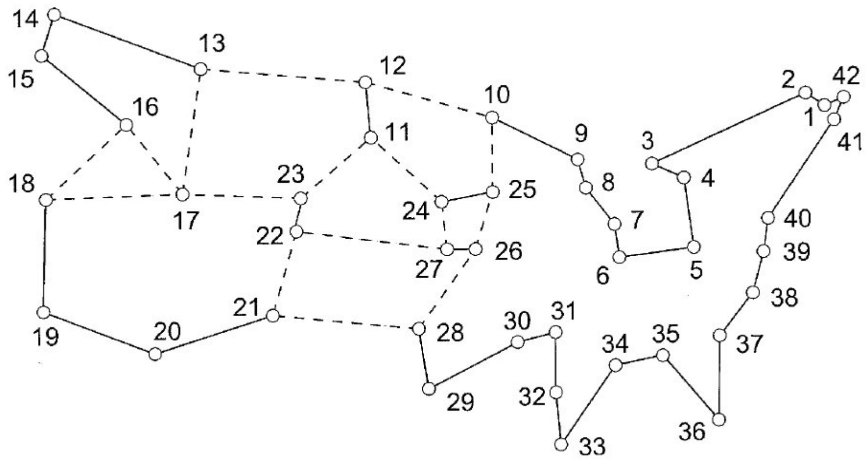


Figure 3.7 What is wrong with this vector?

DM87 – Scheduling, Timetabling and Routing 4

Traveling Salesman Problem

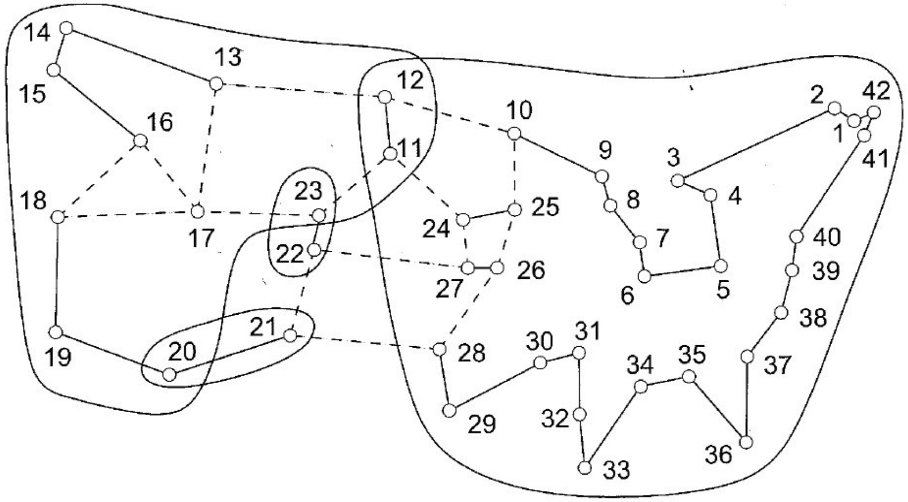


Figure 3.8 A violated comb.

Traveling Salesman Problem

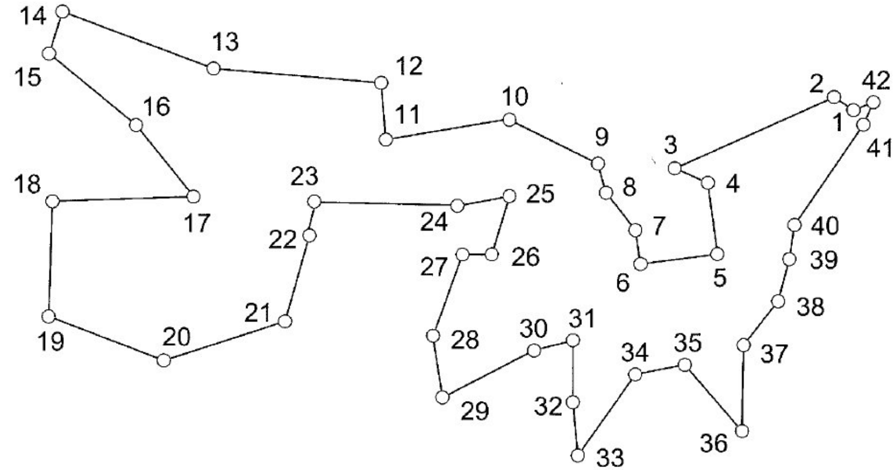


Figure 3.9 An optimal tour through 42 cities.

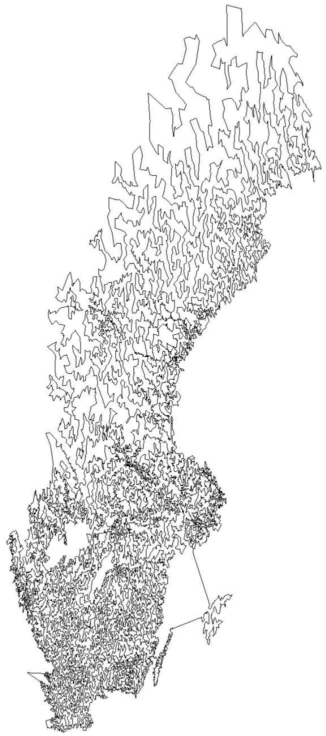
minimize $c^T x$ subject to

$0 \leq x_e \leq 1$ for all edges e,

$\sum(x_e\ :\ v$ is am emd of $e) = 2$ for all cities v,

$\sum(x_e\ :\ e$ has one end in S and one end not in $S) \geq 2$
for all nonempty proper subsets S of cities,

$\sum_{i=0}^{i=3}(\sum(x_e\ :\ e$ has one end in S_i and one end not in $S_i) \geq 10$,
for any comb

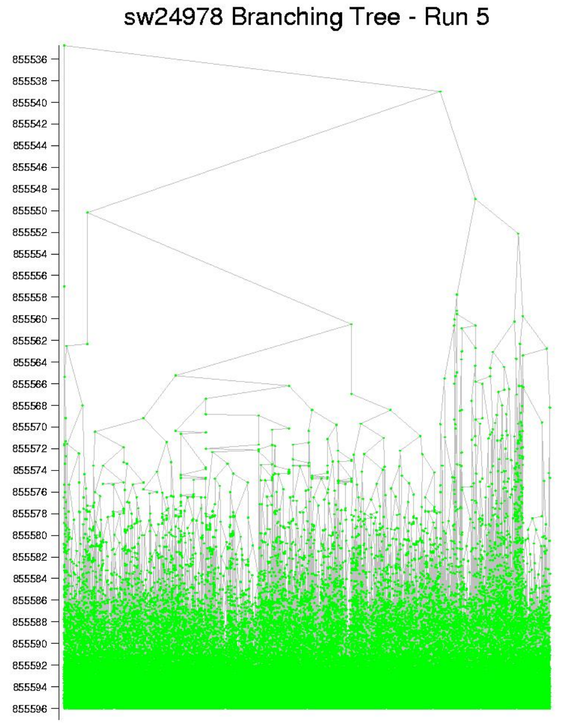


24,978 Cities

solved by LK-heuristic and prooved optimal by branch and cut

10 months of computation on a cluster of 96 dual processor Intel Xeon 2.8 GHz workstations

`http://www.tsp.gatech.edu`



24,978 Cities

solved by LK-heuristic and prooved optimal by branch and cut

10 months of computation on a cluster of 96 dual processor Intel Xeon 2.8 GHz workstations

http://www.tsp.gatech.edu

Modeling: Mixed Integer Formulations

- Formulation for $Qm|p_j = 1|\sum h_j(C_j)$ and relation with transportation problems
- Totally unimodular matrices and sufficient conditions for total unimodularity i) two ones per column and ii) consecutive 1's property
- Formulation of $1|prec|\sum w_j C_j$ and $Rm||\sum C_j$ as weighted bipartite matching and assignment problems.
- Formulation of set covering, set partitioning and set packing
- Formulation of Traveling Salesman Problem
- Formulation of $1|prec|\sum w_j C_j$ and how to deal with disjunctive constraints
- Graph coloring

Outline

Special Purpose Algorithms

Dynamic programming
procedure based on divide and conquer

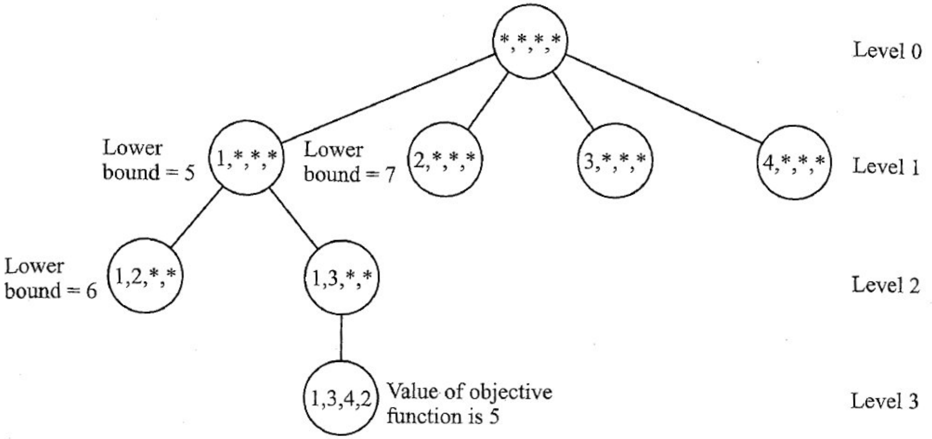
Based on principle of optimality the completion of an optimal sequence of decisions must be optimal

- Break down the problem in stages at which the decisions take place
- Find a recurrence relation that takes us backward (forward) from one stage to the previous (next)

In scheduling, this can be typically done only for objectives that are sequence independent (eg, the makespan).

Special Purpose Algorithms

Branch and Bound
divide and conquer + lower bounding technique



[Jens Clausen. (2003)]

Outline

Constraint Satisfaction Problem

- **Input:**
  - a set of variables $X_1, X_2, \ldots, X_n$
  - each variable has a non-empty domain D_i of possible values
  - a set of constraints. Each constraint C_i involves some subset of the variables and specify the allowed combination of values for that subset.
    [A constraint C on variables X_i and X_j, $C(X_i, X_j)$, defines the subset of the Cartesian product of variable domains $D_i \times D_j$ of the consistent assignments of values to variables. A constraint C on variables X_i, X_j is satisfied by a pair of values v_i, v_j if $(v_i, v_j) \in C(X_i, X_j)$.]
- **Task:**
  - find an assignment of values to all the variables $\{X_i = v_i, X_j = v_j, \ldots\}$
  - such that it is consistent, that is, it does not violate any constraint

If assignments are not all equally good, but some are preferable this is reflected in an objective function.

Search Problem

- initial state: the empty assignment $\{\}$ in which all variables are unassigned
- successor function: a value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables
- goal test: the current assignment is complete
- path cost: a constant cost

Two search paradigms:

- search tree of depth n
- complete state formulation: local search

Types of Variables and Values

- Discrete variables with finite domain:
  complete enumeration is $O(d^n)$
- Discrete variables with infinite domains:
  Impossible by complete enumeration.
  Instead a constraint language (constraint logic programming and constraint reasoning)
  Eg, project planning.

  $$S_j + p_j \leq S_k$$

  NB: if only linear constraints, then integer linear programming
- variables with continuous domains
  NB: if only linear constraints or convex functions then mathematical programming

Types of constraints

- Unary constraints
- Binary constraints (constraint graph)
- Higher order (constraint hypergraph)
  Eg, `Alldiff()`
  Every higher order constraint can be reconduced to binary
  (you may need auxiliary constraints)
- Preference constraints
  cost on individual variable assignments

General Purpose Solution Algorithms

Search algorithms

tree with branching factor at the top level nd
at the next level $(n-1)d$.
The tree has $n! \cdot d^n$ even if only d^n possible complete assignments.

- CSP is commutative in the order of application of any given set of action. (the order of the assignment does not influence)
- Hence we can consider search algs that generate successors by considering possible assignments for only a single variable at each node in the search tree.

Backtracking search

depth first search that chooses one variable at a time and backtracks when a variable has no legal values left to assign.

Backtrack Search

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment according to CONSTRAINTS[csp] then
        add {var = value} to assignment
        result ← RECURSIVE-BACKTRACKING(assignment, csp)
        if result ≠ failure then return result
        remove {var = value} from assignment
  return failure
```

Backtrack Search

- No need to copy solutions all the times but rather extensions and undo extensions
- Since CSP is standard then the alg is also standard and can use general purpose algorithms for initial state, successor function and goal test.
- Backtracking is uninformed and complete. Other search algorithms may use information in form of heuristics

General Purpose backtracking methods

1) Which variable should we assign next, and in what order should its values be tried?
2) What are the implications of the current variable assignments for the other unassigned variables?
3) When a path fails – that is, a state is reached in which a variable has no legal values can the search avoid repeating this failure in subsequent paths?

Which variable should we assign next, and in what order should its values be tried?

- Select-Initial-Unassigned-Variable
  degree heuristic (reduces the branching factor) also used as tied breaker
- Select-Unassigned-Variable
  Most constrained variable (DSATUR) = fail-first heuristic
  = Minimum remaining values (MRV) heuristic (speeds up pruning)
- Order-Domain-Values
  least-constraining-value heuristic (leaves maximum flexibility for subsequent variable assignments)

NB: If we search for all the solutions or a solution does not exists, then the ordering does not matter.

What are the implications of the current variable assignments for the other unassigned variables?

Propagating information through constraints

- Implicit in Select-Unassigned-Variable
- Forward checking (coupled with MRV)
- Constraint propagation
  - arc consistency: force all (directed) arcs uv to be consistent: $\exists$ a value in $D(v)$: $\forall$ values in $D(u)$, otherwise detects inconsistency

    can be applied as preprocessing or as propagation step after each assignment (MAC, Maintaining Arc Consistency)

    Applied repeatedly
  - k-consistency: if for any set of $k-1$ variables, and for any consistent assignment to those variables, a consistent value can always be assigned to any k-th variable.

    determining the appropriate level of consistency checking is mostly an empirical science.

Arc Consistency Algorithm: AC-3

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	R	B	G	R B	R G B	B	R G B
After *V=blue*	R	B	G	R	B		R G B

Figure 5.6 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.

Arc Consistency Algorithm: AC-3

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

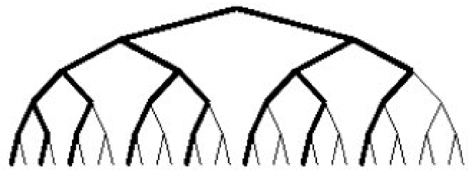
  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS[X_i] **do**
        add (X_k, X_i) to *queue*

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff we remove a value
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN[X_i] **do**
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint between X_i and X_j
      then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  return *removed*

Incomplete Search

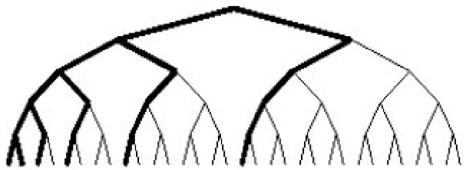
General purpose algorithms:

Credit-based search:



credit(16)

Limited Discrepancy Search:



lds(1)

Limited Discrepancy Search

- A discrepancy is a branch against the value of an heuristic
- Ex: count one discrepancy if second best is chosen
  count two discrepancies either if third best is chosen or twice the second best is chosen
- Explore the tree in order of an increasing number of discrepancies

Handling special constraints (higher order constraints)

Special purpose algorithms

- Alldiff
  - for m variables and n values cannot be satisfied if $m > n$,
  - consider first singleton variables
  - propagation based on bipartite matching considerations
- Resource Constraint atmost
  - check the sum of minimum values of single domains
    delete maximum values if not consistent with minimum values of others.
  - for large integer values not possible to represent the domain as a set of integers but rather as bounds.
    Then bounds propagation: Eg,
    $\mathtt{Flight271} \in [0, 165]$ and $\mathtt{Flight272} \in [0, 385]$
    $\mathtt{Flight271} + \mathtt{Flight272} \in [420, 420]$
    $\mathtt{Flight271} \in [35, 165]$ and $\mathtt{Flight272} \in [255, 385]$

When a path fails – that is, a state is reached in which a variable has no legal values can the search avoid repeating this failure in subsequent paths?

Backtracking-Search

- chronological backtracking, the most recent decision point is revisited
- backjumping, backtracks to the most recent variable in the conflict set (set of previously assigned variables connected to X by constraints).

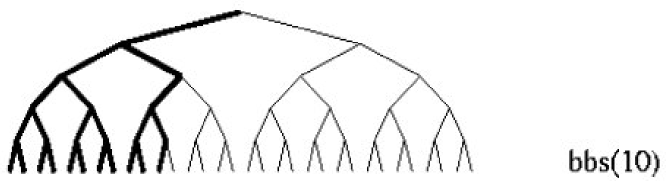
  every branch pruned by backjumping is also pruned by forward checking

  idea remains: backtrack to reasons of failure.

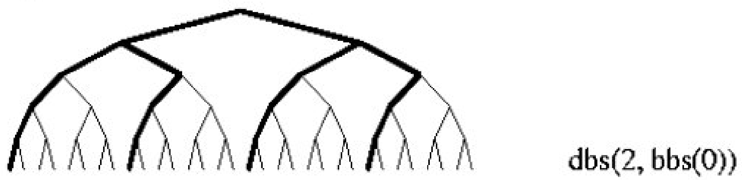
Incomplete Search

General purpose algorithms:

Bounded-backtrack search:



Depth-bounded, then bounded-backtrack search:



An Empirical Comparison

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV
USA	(> 1,000K)	(> 1,000K)	2K	60
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	817K
Zebra	3,859K	1K	35K	0.5K
Random 1	415K	3K	26K	2K
Random 2	942K	27K	77K	15K

Mendian number of consistency checks

The structure of problems

- Decomposition in subproblems:
  - connected components in the constraint graph
  - $O(d^c n/c)$ vs $O(d^n)$
- Constraint graphs that are tree are solvable in poly time by reverse arc-consistency checks.
- Reduce constraint graph to tree:
  - removing nodes (cutset conditioning: find the smallest cycle cutset. It is NP-hard but good approximations exist)
  - collapsing nodes (tree decomposition) divide-and-conquer works well with small subproblems

Optimization Problems

Objective function $F(X_1, X_2, \ldots, X_n)$

- Solve a modified Constraint Satisfaction Problem by setting a (lower) bound z^* in the objective function
- Dichotomic search: U upper bound, L lower bound

$$M = \frac{U + L}{2}$$

Constraint Logic Programming

Language is first-order logic.

- Syntax – Language
  - Alphabet
  - Well-formed Expressions
    E.g., 4X + 3Y = 10; 2X - Y = 0
- Semantics – Meaning
  - Interpretation
  - Logical Consequence
- Calculi – Derivation
  - Inference Rule
  - Transition System

Logic Programming

A logic program is a set of axioms, or rules, defining relationships between objects.

A computation of a logic program is a deduction of consequences of the program.

A program defines a set of consequences, which is its meaning.

The art of logic programming is constructing concise and elegant programs that have desired meaning.

Sterling and Shapiro: The Art of Prolog, Page 1.

Local Search for CSP

- Uses a complete-state formulation: a value assigned to each variable (randomly)
- Changes the value of one variable at a time
- Min-conflicts heuristic is effective particularly when given a good initial state.
- Run-time independent from problem size
- Possible use in online settings in personal assignment: repair the schedule with a minimum number of changes