Part 2: Algorithms

Overview

- Problem formulation:
  - Optimization task

- Exact algorithms:
  - Branch and Bound
  - Partial Forward Checking
  - Reversible DACs
  - Russian Doll Search
  - Bucket Elimination

tree search

- Approximate algorithms:
  - Local search approaches
  - Interval approximation

Terminology

- Variables: $i, j, k, ...$
- Values: $a, b, c, \ldots$
- Constraints: $f, g, h, \ldots$ $var(f)$: variables involved in f
- Domains: $D_0(i)$: initial domain of variable i
  $D(i)$: current domain of variable i

- P: set of assigned or past variables
- F: set of unassigned or future variables

- C_P: set of constraints involving past variables
- C_{PF}: set of constraints involving past and future variables
- C_F: set of constraints involving future variables

- τ: current assignment
- $\tau[i]$: current assignment projected over variable i

Problem Formulation

- Soft CSP: (X,D,C)
  - X is a set of n variables
  - $D = \{ D_0(1), D_0(2), \dots , D_0(n)\}$ is a collection of variable domains
  - C is a set of r soft constraints

$$f \in C \qquad f : \prod_{i \in var(f)} D_0(i) \longrightarrow [0, +\infty]$$

$$f(t) = \begin{cases} 0 & t \text{ satisfies completely } f \\ (0, +\infty) & t \text{ satisfies / violates partially } f \\ +\infty & t \text{ violates completely } f \end{cases}$$

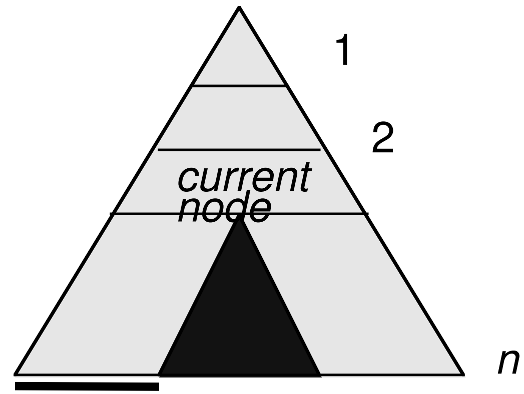
preferences

$f(t)$: cost associated with violation of f by t

- Goal: *minimize* $\sum_{f \in C} f(t)$ weighted CSP

(NP-hard)

Branch and Bound



- Depth-first tree search:
  - *internal node*: partial assignment
  - *leaf*: total assignment
- At each node:

  Distance: $dist(\tau) = \sum_{f \in C_P} f(\tau)$

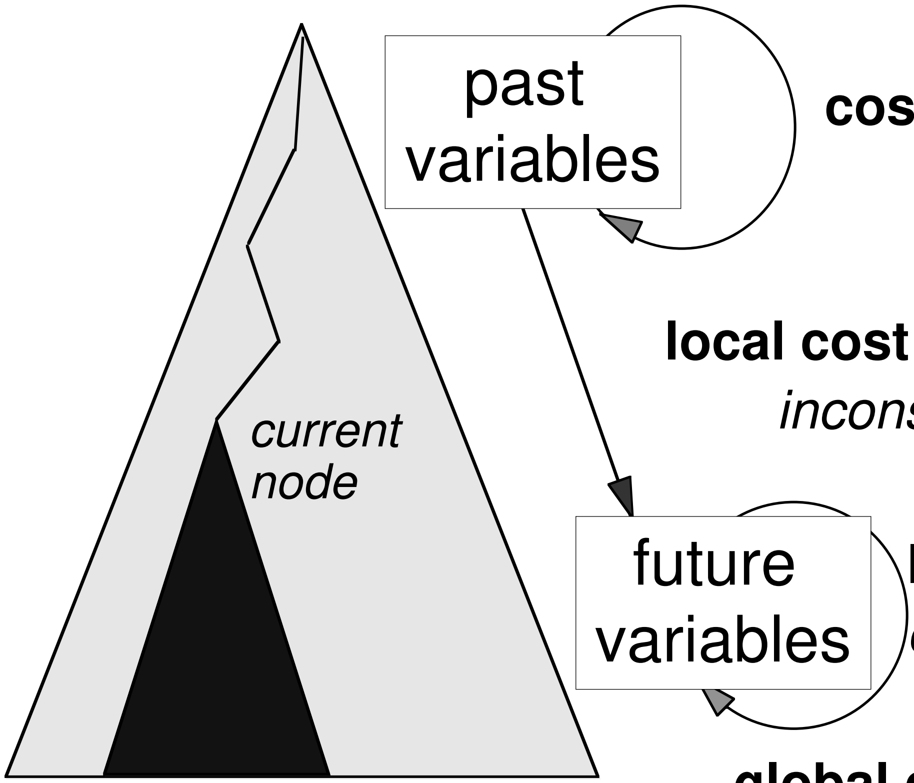
  Upper bound (UB): minimum distance of visited leaves: distance of the current best solution

  Lower bound (LB): underestimation of minimum distance among leaves below current node

  Pruning: $UB \leq LB$
- Simplest LB: $dist(\tau)$

Lower Bound

LB quality: very important for branch and bound efficiency



cost of past variables
distance

local cost of future var because past var
inconsistency counts

local cost of future variables
directed arc-inconsistency counts

global cost of future variables
russian doll search

Partial Forward Checking

[Freuder & Wallace 92]

- Branch and Bound + Lookahead

- Cost of value *a* of future variable *i* because constraint *f*:

$$cost\,(i,a,f) = min_{\,t[i]=a}\, f(t) \qquad t \in \textstyle\prod_{i \in var(f)} D\,(i) \quad \textit{valid t}$$

$$cost_0\,(i,a,f) = min_{\,t[i]=a}\, f(t) \qquad t \in \textstyle\prod_{i \in var(f)} D_0\,(i) \quad \textit{initial t}$$

- Lookahead: after assigning a value to a variable
  - inconsistency counts on future values are computed
  - inconsistency count of value *a* of future variable *i* :

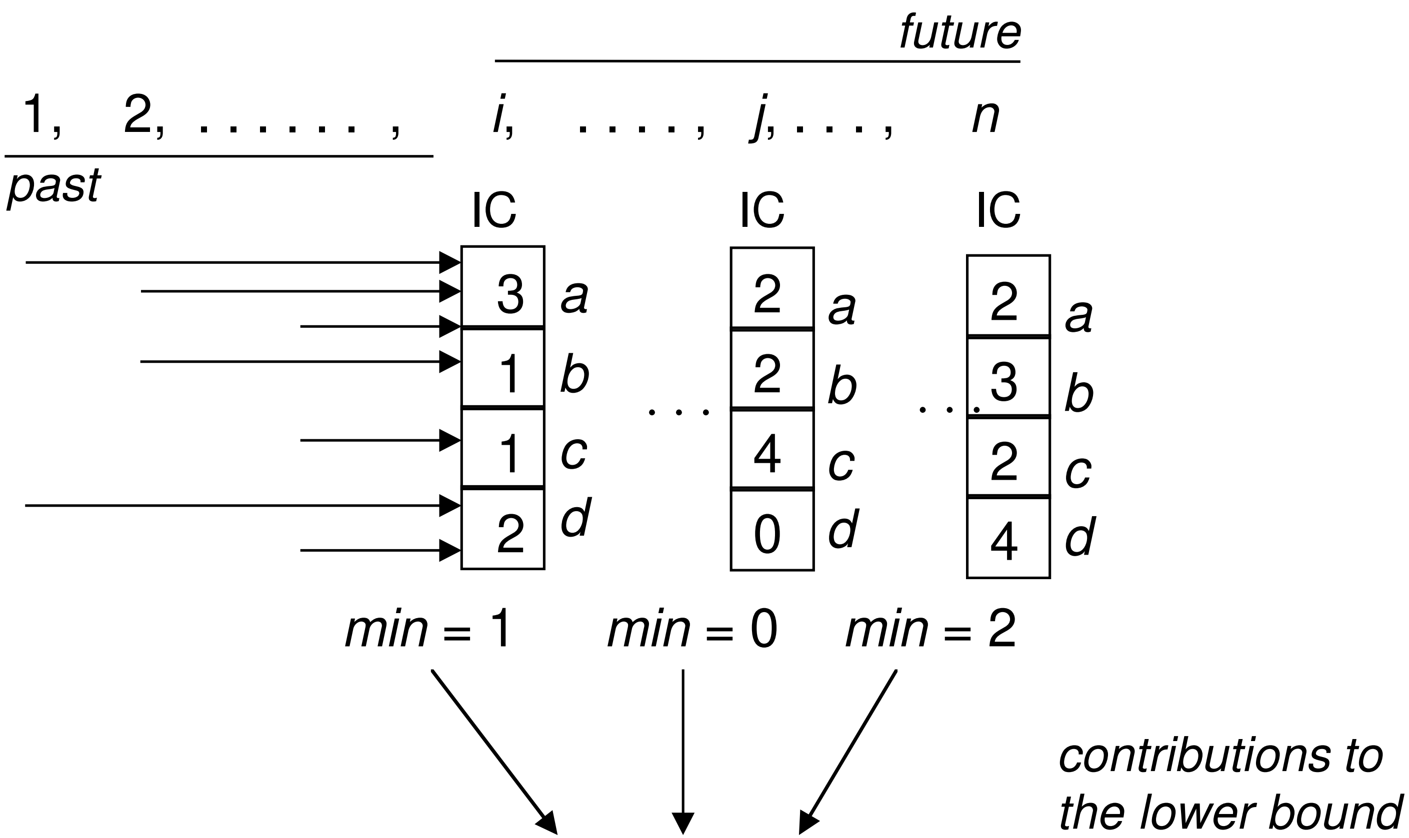
$$ic\,(i,a) = \textstyle\sum_{f \in C_{PF}} cost\,(i,a,f)$$

binary constraints

C_{PF}: constraints involving one past and one future variables

Inconsistency Counts

[Freuder & Wallace 92]



$$\sum_{i \in F} min_a\,(ic\,(i,a))$$

PFC Lower Bound

[Freuder & Wallace 92]

- New lower bound:

$$LB(\tau, F) = \underbrace{dist(\tau)}_{\textit{cost from } C_P} + \underbrace{\textstyle\sum_{i \in F} \min_a \{ic(i,a)\}}_{\textit{cost from } C_{PF}}$$

$$LB_{jb}(\tau, F) = dist(\tau) + ic(j,b) + \textstyle\sum_{i \in F - \{j\}} \min_a \{ic(i,a)\}$$

- Future value pruning: $LB_{jb}(\tau, F) \geq UB$
  - the cost of extending τ with (j,b) is greater than or equal to the cost of the current best solution
  - (j,b) can be pruned: it will never improve the current best solution
  - when backtracking to i, (j,b) must be restored

Static DAC

[Wallace 95]

- DAC: (directed arc-inconsistency counts) on future values
  - static variable ordering: 1, 2,. . ., *i*, . . ., *j*, . . ., *n*
  - C_F: constraints involving two future variables

binary constraints

$$dac\,(i,a) = \sum_{f \in C_F} cost_0\,(i,a,f) \qquad var(f) = (i, j),\ i < j$$

Static order: $\underbrace{1,\ 2,\ .\,.\,.\,.\,.\,.\,.\,.\,.}_{past}\,\overbrace{,\ i,\ .\,.\,.\,.\,.\,.\ ,\ j,\ .\,.\,.\,.\,.\ ,\ n}^{future}$

	DAC		DAC
a	1	*a*	2
b	2	*b*	0
c	1	*c*	1

$min = 1$ $min = 0$

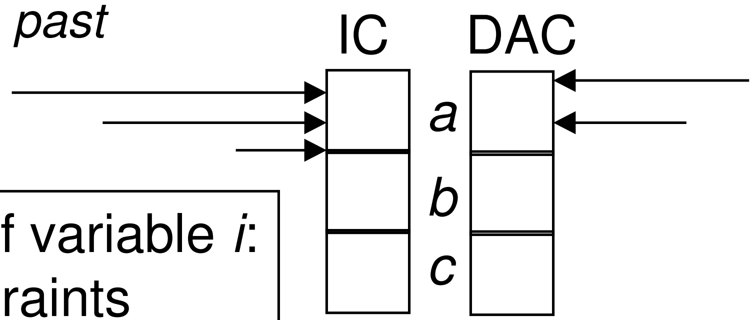
contributions to the lower bound

$$\sum_{i \in F} min_a\,(dac\,(i,a))$$

Combining IC + DAC

[Wallace 95] [Larrosa & Meseguer 96]

Static order: $\underbrace{1,\ 2,\ \ldots\ldots\ldots,}_{past}\ \overbrace{i,\ \ldots\ldots,\ j,\ \ldots\ldots,\ n}^{future}$



IC and DAC of value *a* of variable *i*:
– refer to different constraints
– can be added

- New lower bound:

$$LB(\tau,F) = \underbrace{dist(\tau)}_{\text{cost from } C_P} + \underbrace{\sum_{i\in F} \min_a \{ic(i,a) + dac(i,a)\}}_{\text{cost from } C_{PF} \cup C_F}$$

$$LB_{jb}(\tau,F) = dist(\tau) + ic(j,b) + dac(j,b) + \sum_{i\in F-\{j\}} \min_a \{ic(i,a) + dac(i,a)\}$$

DAC: Directed Constraints

$i \overset{}{\underset{f}{\text{———}}} j$

i j	*f*
a a	1
a b	2
b a	3
b b	4

dac	*i j*
a	1 1
b	3 2

min dac(*i*) + *min dac* (*j*) = 1+1=2 but *f*(*a*,*a*) = 1!

$\downarrow$

- For DAC usage, constraints must be directed
- DAC stored in the variable pointed by the constraint
- Implicit in static DAC by static variable order

$i \xleftarrow[f]{} j$

dac	*i j*
a	1 0
b	3 0

min dac(*i*) + *min dac* (*j*) = 1+0 =1 OK!

Graph-based DAC

[Larrosa, Meseguer, Schiex 99]

- Directed constraint graph G^F
  - $Nodes(G^F)=F$
  - $Edges(G^F)=\{(j,i) \mid C_F$, directed from j to $i\}$

- DAC based on G^F:

$$dac\ (i,a,G^F) = \sum_{f \in C_F} cost_0\ (i,a,f) \qquad var(f) = (i, j),\ (j, i) \in Edges(G^F)$$

- New lower bound:

$$LB\ (\tau,F,G^F) = dist\ (\tau) + \sum_{i \in F} \min_a \{ic\ (i,a) + dac\ (i,a,G^F)\}$$

$$LB_{jb}(\tau,F,G^F) = dist\ (\tau) + ic\ (j,b) + dac\ (j,b,G^F) + \sum_{i \in F-\{j\}} \min_a \{ic\ (i,a) + dac\ (i,a,G^F)\}$$

Reversible DAC

[Larrosa, Meseguer, Schiex 99]

- Reversible R DAC:
  - Any G^F is suitable for *LB* computation
  - Dynamically selects a *good* G^F (optimal NP-hard, greedy search)
  - Single operation: *reversing* edge direction

- Maintaining DAC: redefinition $cost_0 \longrightarrow cost$

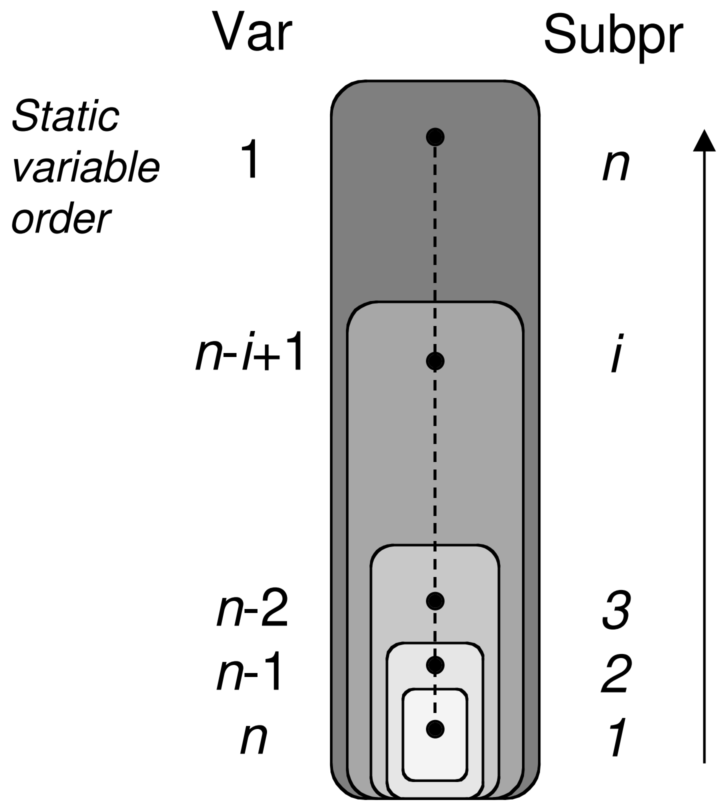
$$dac\,(i,a,G^F) = \sum_{f \in C_F} cost\,(i,a,f) \qquad var(f) = (i, j),\ (j, i) \in Edges(G^F)$$

  - Previous DAC: initially precomputed
  - DAC can be maintained at run time
    - removed values generate further DAC
    - AC adapted algorithm

Russian Doll Search

[Verfaillie, Lemaitre, Schiex 96]

- To replace one search by n successive searches on nested subproblems



•Sequence subpr: 1, 2, . . ., i, . . , n

•Each subproblem is optimally solved:

rds (*subpr*) = optimal cost of subpr

•When solving subproblem i+1 costs of solutions of subproblem i to 1 are used

Russian Doll Search

[Verfaillie, Lemaitre, Schiex 96]

- New lower bound:

$$LB(\tau,F) = \underbrace{dist\,(\tau)}_{\text{cost from } C_P} + \underbrace{\sum_{i \in F} \min_a \{ic\,(i,a)\}}_{\text{cost from } C_{PF}} + \underbrace{rds\,(F)}_{\text{cost from } C_F}$$

- Solving subproblem i+1:

$\underbrace{n\text{-}i}_{P},\ \underbrace{n\text{-}i\text{+}1,\ n\text{-}i\text{+}2,\ \ldots\ldots,\ n}_{F}$ $\qquad F$ = subpr i

$\underbrace{n\text{-}i,\ n\text{-}i\text{+}1}_{P},\ \underbrace{n\text{-}i\text{+}2,\ \ldots\ldots,\ n}_{F}$ $\qquad F$ = subpr i-1

. . . .

$\underbrace{n\text{-}i,\ n\text{-}i\text{+}1,\ n\text{-}i\text{+}2,\ \ldots\ldots}_{P},\ \underbrace{n}_{F}$ $\qquad F$ = subpr 1

Bucket Elimination

[Dechter 99]

- Dynamic programming/ Variable elimination:
  - no tree search
  - synthesize the best solution
  - static variable ordering: $1, 2, \ldots, n$
- Bucket k: set of constraints involving variable k and $i, j, \ldots$ such that $i, j, \ldots < k$.
- Operations on constraints:
  - Addition: $f(t) + g(t) = f(t[var(f)]) + g(t[var(g)])$
  - Projecting out variable k: $(f \Downarrow k)(t) = \min_{a \in D(k)} f(t) \quad t[k] = a$

Bucket Elimination

[Dechter 99]

static variable order

Var	Bucket
n	bucket n
n-1	bucket n-1
n-2	bucket n-2
1	bucket 1

Process from bucket n to 1:

Bucket $k = \{ f_1, f_2, \ldots, f_p\}$

1. Add all constraints, getting a new one

$$g = f_1 + f_2 + \ldots + f_p$$

2. Project out of g variable k

$$h = g \Downarrow k$$

3. Add h to the corresponding bucket

Comments:

— Constraint g summarizes all information of constraints $f_1, f_2, \ldots, f_p$

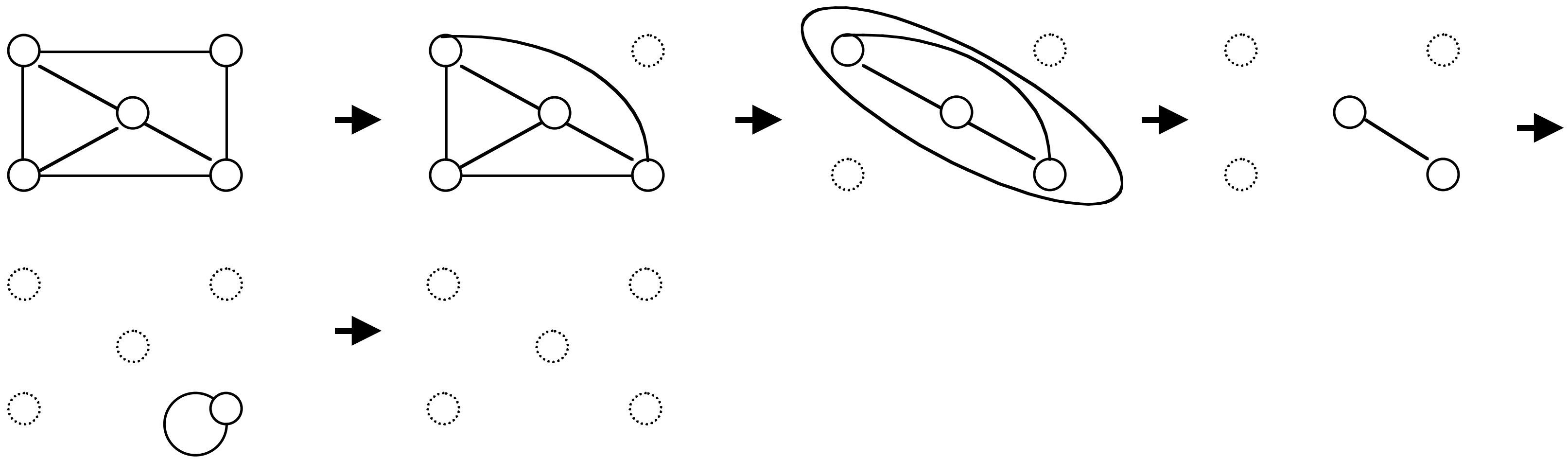
— Each iteration transforms a problem P into an equivalent P' with one less var

— The process iterates until no variables

Bucket Elimination

[Dechter 99]

Example:



Solution: from var 1 to *n*

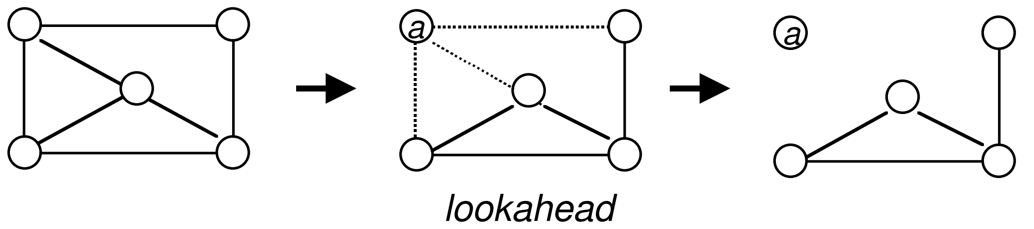
Assign to variable *i* the best value in *g* according with previous assignements

Complexity: Space and time exponential in the induced width of the graph: *max arity* of new constraints

Search + Variable Elimination

[Larrosa 00]

- Branching:

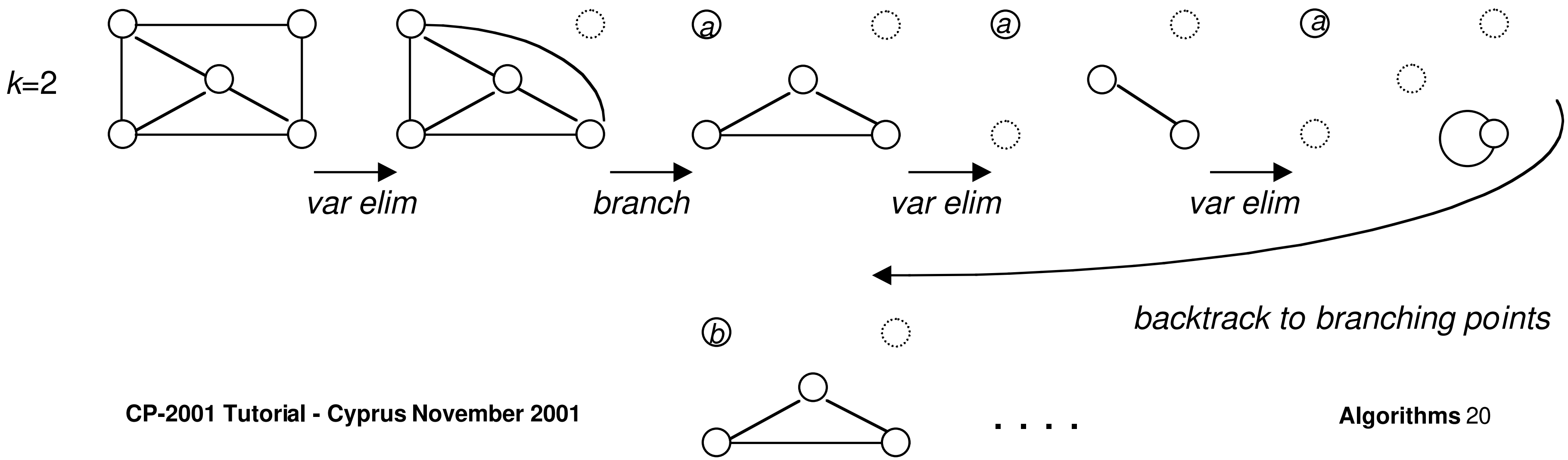


- Combining strategy: *k* parameter

  if eliminating variable *i* causes a new constraint with arity <= *k*

  then eliminate variable *i*

  else do branching on the most connected variable



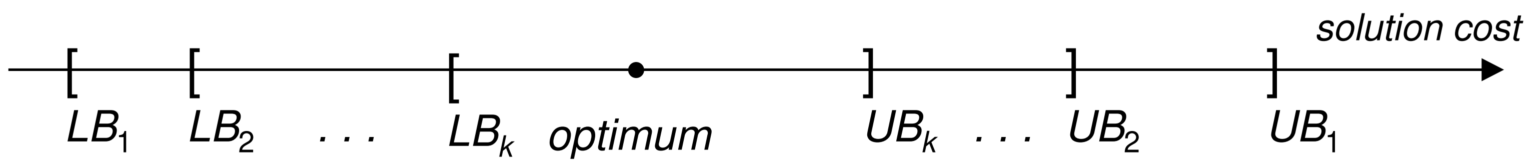
Local Search

- Optimization problem: *minimize* $\sum_{f \in C} f(t)$
- Local search approaches: produce an *upper bound*
  - hill climbing [Hao & Dorne 96]
  - simulated annealing [Wah & Chen 00]
  - tabu search [Galinier & Hao 97]
  - genetic algorithms: special operators
    [Lau & Tsang 01] [Wiese & Goodwin 01]
- Elements:
  - evaluation function: function to optimize
  - neighborhood: one move, two moves, . . .
  - selection criteria: new state, escaping local optima, randomization, *cooling schedule*, . . .

Interval Approximation

[Cabon, de Givry, Verfaillie 98]

- Idea:
  - in many cases, looking for the optimum is too complex
  - instead, look for an interval [*LB*, *UB*] containing the optimum
  - when [*LB*, *UB*] is narrow enough, stop search



- Anytime bounds:
  - *UB*: local search approaches
  - *LB*: three strategies
    - problem simplification: solve a simpler problem
    - objective simplification: sum of local costs
    - search simplification: russian doll + iterative deepening