Algorithms for Reasoning with graphical models

Class4
Rina Dechter
The Induced-Width

- Width along $d$, $w(d)$:
  - max # of previous parents

- Induced width $w^*(d)$:
  - The width in the ordered induced graph

- Induced-width $w^*$:
  - Smallest induced-width over all orderings

- Finding $w^*$
  - NP-complete (Arnborg, 1985) but greedy heuristics (min-fill).
Road Map

- Graphical models
- Constraint networks Model
- Inference
  - Variable elimination for Constraints
  - Variable elimination for CNFs
  - Greedy search for induced-width orderings
  - Variable elimination for Linear Inequalities
- Constraint propagation
- Search
- Probabilistic Networks
Finding a Small Induced-Width

- NP-complete
- A tree has induced-width of ?
- Greedy algorithms:
  - Min width
  - Min induced-width
  - Max-cardinality and chordal graphs
  - Fill-in (thought as the best)
- Anytime algorithms
  - Search-based [Gogate & Dechter 2003]
  - Stochastic (CVO) [Kask, Gelfand & Dechter 2010]
Min-width Ordering

MIN-WIDTH (MW)

**input:** a graph \( G = (V, E) \), \( V = \{v_1, \ldots, v_n\} \)

**output:** A min-width ordering of the nodes \( d = (v_1, \ldots, v_n) \).

1. \textbf{for} \( j = n \) to 1 by -1 \textbf{do}
2. \hspace{1em} \( r \leftarrow \) a node in \( G \) with smallest degree.
3. \hspace{1em} \textbf{put} \( r \) in position \( j \) and \( G \leftarrow G - r \).
   \hspace{1em} (Delete from \( V \) node \( r \) and from \( E \) all its adjacent edges)
4. \textbf{endfor}

**Proposition:** algorithm min-width finds a min-width ordering of a graph

**What is the Complexity of MW?**

\( O(e) \)
Greedy Orderings Heuristics

- Min-induced-width
  - From last to first, pick a node with smallest width, then connect parent and remove

- Min-Fill
  - From last to first, pick a node with smallest fill-edges

Complexity? $O(n^3)$
Min-Fill Heuristic

- Select the variable that creates the fewest “fill-in” edges

Eliminate B next?
Connect neighbors
“Fill-in” = 3: (A,D), (C,E), (D,E)

Eliminate E next?
Neighbors already connected
“Fill-in” = 0
Example
Different Induced-Graphs

(a) A Min-fill ordering

(b) A Miw ordering

(c) A Min-fill ordering

(d) A Miw ordering
A graph is chordal if every cycle of length at least 4 has a chord.

Deciding chordality by max-cardinality ordering:
- from 1 to n, always assigning a next node connected to a largest set of previously selected nodes.

A graph along max-cardinality order has no fill-in edges iff it is chordal.

The maximal cliques of chordal graphs form a tree.

[Tarjan & Yanakakis 1980]
Greedy Orderings Heuristics

- **Min-Induced-width**
  - From last to first, pick a node with smallest width

- **Min-Fill**
  - From last to first, pick a node with smallest fill-edges
  
  *Complexity?* \( O(n^3) \)

- **Max-Cardinality search**  \[Tarjan & Yanakakis 1980\]
  - From **first to last**, pick a node with largest neighbors already ordered.
  
  *Complexity?* \( O(n + m) \)
Max-cardinality ordering

MAX-CARDINALITY (MC)

**input:** a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$

**output:** An ordering of the nodes $d = (v_1, ..., v_n)$.

1. Place an arbitrary node in position 0.
2. for $j = 1$ to $n$ do
3. \hspace{1em} $r \leftarrow$ a node in $G$ that is connected to a largest subset of nodes in positions 1 to $j - 1$, breaking ties arbitrarily.
4. endfor

Proposition 5.3.3 [56] Given a graph $G = (V, E)$ the complexity of max-cardinality search is $O(n + m)$ when $|V| = n$ and $|E| = m$. 
Example

We see again that $G$ in the Figure (a) is not chordal since the parents of $A$ are not connected in the max-cardinality ordering in Figure (d). If we connect $B$ and $C$, the resulting induced graph is chordal.
Which Greedy Algorithm is Best?

- Min-Fill, prefers a node who add the least number of fill-in arcs.

- Empirically, fill-in is the best among the greedy algorithms (MW, MIW, MF, MC)

- Complexity of greedy orderings?
  - MW is $O(e)$, MIW: $O(n^3)$ MF $O(n^3)$ MC is $O(e+n)$
K-trees

Definition 5.3.4 (k-trees) A subclass of chordal graphs are k-trees. A k-tree is a chordal graph whose maximal cliques are of size $k + 1$, and it can be defined recursively as follows: (1) A complete graph with $k$ vertices is a k-tree. (2) A k-tree with $r$ vertices can be extended to $r + 1$ vertices by connecting the new vertex to all the vertices in any clique of size $k$. A partial k-tree is a k-tree having some of its arcs removed. Namely it will clique of size smaller than $k$. 
Finding a Small Induced-Width

- NP-complete
- A tree has induced-width of ?
- Greedy algorithms:
  - Min width (MW)
  - Min induced-width (MIW)
  - Max-cardinality and chordal graphs (MC)
  - Min-Fill (thought as the best) (MIN-FILL)
- Anytime algorithms
  - Search-based [Gogate & Dechter 2003]
  - Stochastic (CVO) [Kask, Gelfand & Dechter 2010]
Summary Of Inference Scheme

- Bucket elimination is time and memory exponential in the induced-width.

- Finding the $w^*$ is hard, but greedy schemes work quite well to approximate. Most popular is fill-edges.

- $W(d)$ is the induced-width along an ordering $d$. Smallest induced-width is also called tree-width.
Recent work in my group

- **Vibhav Gogate and Rina Dechter.** "A Complete Anytime Algorithm for Treewidth". *In UAI 2004.*
- **Andrew E. Gelfand, Kalev Kask, and Rina Dechter.** "Stopping Rules for Randomized Greedy Triangulation Schemes" in *Proceedings of AAAI 2011.*
- Kask, Gelfand and Dechter, BEEM: Bucket Elimination with External memory, AAAI 2011 or UAI 2011

- Potential project
Greedy Algorithms for Induced-Width

- Min-width ordering
- Min-induced-width ordering
- Max-cardinality ordering
- Min-fill ordering
- Chordal graphs
- Hypergraph partitionings

(Project: present papers on induced-width, run algorithms for induced-width on new benchmarks...)

class2 828X 2019
Min-width Ordering

MIN-WIDTH (MW)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: A min-width ordering of the nodes $d = (v_1, ..., v_n)$.

1. for $j = n$ to 1 by -1 do
2. $r \leftarrow$ a node in $G$ with smallest degree.
3. put $r$ in position $j$ and $G \leftarrow G - r$.
   (Delete from $V$ node $r$ and from $E$ all its adjacent edges)
4. endfor

Proposition: algorithm min-width finds a min-width ordering of a graph

Complexity: $O(e)$
Greedy Orderings Heuristics

**min-induced-width (miw)**
input: a graph $G = (V;E), V = \{v_1; \ldots; v_n\}$
output: An miw ordering of the nodes $d = (v_1; \ldots; v_n)$.
1. for $j = n$ to $1$ by $-1$ do
2. $r \leftarrow$ a node in $V$ with smallest degree.
3. put $r$ in position $j$.
4. connect $r$'s neighbors: $E \leftarrow E \cup \{(v_i; v_j) | (v_i; r) \in E; (v_j; r) \in E\}$,
5. remove $r$ from the resulting graph: $V \leftarrow V - \{r\}$.

**min-fill (min-fill)**
input: a graph $G = (V;E), V = \{v_1; \ldots; v_n\}$
output: An ordering of the nodes $d = (v_1; \ldots; v_n)$.
1. for $j = n$ to $1$ by $-1$ do
2. $r \leftarrow$ a node in $V$ with smallest fill edges for his parents.
3. put $r$ in position $j$.
4. connect $r$'s neighbors: $E \leftarrow E \cup \{(v_i; v_j) | (v_i; r) \in E; (v_j; r) \in E\}$,
5. remove $r$ from the resulting graph: $V \leftarrow V - \{r\}$.

**Theorem:** A graph is a tree iff it has both width and induced-width of 1.
A graph is chordal if every cycle of length at least 4 has a chord.

Finding $w^*$ over chordal graph is easy using the max-cardinality ordering.

The induced graph is chordal.

K-trees are special chordal graphs.

Finding the max-clique in chordal graphs is easy (just enumerate all cliques in a max-cardinality ordering).
Road Map

- Graphical models
- Constraint networks Model
- Inference
  - Variable elimination for Constraints
  - Variable elimination for CNFs
  - Greedy search for induced-width orderings
  - Variable elimination for Linear Inequalities
- Constraint propagation
- Search
- Probabilistic Networks
Variables domains are the real numbers

\[(3x_i + 2x_j \leq 3) \land (-4x_i + 5x_j \leq 1)\]

Definition 3.3.1 (Linear elimination) Let \(\alpha = \sum_{i=1}^{r-1} a_i x_i + a_r x_r \leq c\), and \(\beta = \sum_{i=1}^{r-1} b_i x_i + b_r x_r \leq d\). Then \(\text{elim}_r(\alpha, \beta)\) is applicable only if \(a_r\) and \(b_r\) have opposite signs, in which case \(\text{elim}_r(\alpha, \beta) = \sum_{i=1}^{r-1} (-a_i \frac{b_r}{a_r} + b_i) x_i \leq -\frac{b_r}{a_r} c + d\). If \(a_r\) and \(b_r\) have the same sign the elimination implicitly generates the universal constraint.
Linear Inequalities: Fourier Elimination

**Directional-Linear-Elimination** \((\varphi, d)\)

**Input:** A set of linear inequalities \(\varphi\), an ordering \(d = x_1, \ldots, x_n\).

**Output:** A decision of whether \(\varphi\) is satisfiable. If it is, a backtrack-free theory \(E_d(\varphi)\).

1. **Initialize:** Partition inequalities into ordered buckets.
2. for \(i \leftarrow n\) downto 1 do
3. if \(x_i\) has one value in its domain then
   . substitute the value into each inequality in the bucket
   and put the resulting inequality in the right bucket.
4. else, for each pair \(\{\alpha, \beta\} \subseteq \text{bucket}_i\), compute \(\gamma = \text{elim}_i(\alpha, \beta)\)
   if \(\gamma\) has no solutions, return \(E_d(\varphi) = \{\}\), “inconsistency”
   else add \(\gamma\) to the appropriate lower bucket.
5. return \(E_d(\varphi) \leftarrow \bigcup_i \text{bucket}_i\)
Theorem 4.8.3  Given a set of linear inequalities $\varphi$, algorithm DLE (Fourier elimination) decides the consistency of $\varphi$ over the Rationals and the Reals, and it generates an equivalent backtrack-free representation. $\square$
**Example**

\[
\text{bucket}_4 : \quad 5x_4 + 3x_2 - x_1 \leq 5, \quad x_4 + x_1 \leq 2, \quad -x_4 \leq 0,
\]
\[
\text{bucket}_3 : \quad x_3 \leq 5, \quad x_1 + x_2 - x_3 \leq -10
\]
\[
\text{bucket}_2 : \quad x_1 + 2x_2 \leq 0.
\]
\[
\text{bucket}_1 :
\]

Figure 4.23: initial buckets
Example

bucket$_4$ : $5x_4 + 3x_2 - x_1 \leq 5$, $x_4 + x_1 \leq 2$, $-x_4 \leq 0$
bucket$_3$ : $x_3 \leq 5$, $x_1 + x_2 - x_3 \leq -10$
bucket$_2$ : $x_1 + 2x_2 \leq 0$
bucket$_1$ :

Figure 4.23: initial buckets

bucket$_4$ : $5x_4 + 3x_2 - x_1 \leq 5$, $x_4 + x_1 \leq 2$, $-x_4 \leq 0$
bucket$_3$ : $x_3 \leq 5$, $x_1 + x_2 - x_3 \leq -10$
bucket$_2$ : $x_1 + 2x_2 \leq 0$ $\parallel$ $3x_2 - x_1 \leq 5$, $x_1 + x_2 \leq -5$
bucket$_1$ : $\parallel x_1 \leq 2$.

Figure 4.24: final buckets
Algorithms for Reasoning with graphical models

Class5
Rina Dechter
Road Map

- Graphical models
- Constraint networks Model

Inference
- Variable elimination for Constraints
- Variable elimination for CNFs
- Variable elimination for Linear Inequalities
- Constraint propagation (chapter 3 Dechter2)

Search

Probabilistic Networks
Outline

- Arc-consistency algorithms
- Path-consistency and i-consistency
- Arc-consistency, Generalized arc-consistency, relational arc-consistency
- Global and bound consistency
- Distributed (generalized) arc-consistency
- Consistency operators: join, resolution, Gaussian elimination
Sudoku – Approximation: Constraint Propagation

- **Variables**: empty slots
- **Domains** = \{1,2,3,4,5,6,7,8,9\}
- **Constraints**: 27 all-different

Each row, column and major block must be all-different

“Well posed” if it has unique solution: 27 constraints
Approximating Inference: Local Constraint Propagation

- **Problem:** Adaptive-consistency/Bucket-elimination algorithms are intractable when *induced-width* is large

- **Approximation:** bound the size of recorded dependencies, i.e. perform local constraint propagation (local inference)
From Global to Local Consistency
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.

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

Only domains are reduced:

$R_X \leftarrow \prod_X R_{XY} \Join D_Y$
Definition: Given a constraint graph $G$,

- A variable $X_i$ is arc-consistent relative to $X_j$ iff for every value $a \in D_{X_i}$ there exists a value $b \in D_{X_j}$ such that $(a, b) \in R_{X_i, X_j}$.

- The constraint $R_{X_i, X_j}$ is arc-consistent iff
  - $X_i$ is arc-consistent relative to $X_j$ and
  - $X_j$ is arc-consistent relative to $X_i$.

- A binary CSP is arc-consistent iff every constraint (or sub-graph of size 2) is arc-consistent.
Arc-consistency

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

Question: What will be the domain of $Y$ once the network is arc-consistent? Or, how many values will it have?
1 \leq X, Y, Z, T \leq 3
X < Y
Y = Z
T < Z
X \leq T

\[ R_X \Leftarrow \prod_X R_{XY} \otimes D_Y \]
**Revise**\((x_i, x_j)\)

**Input:** A subnetwork defined by two variables \(X = \{x_i, x_j\}\), a distinguished variable \(x_i\), domains: \(D_i\) and \(D_j\), and constraint \(R_{ij}\)

**Output:** \(D_i\), such that, \(x_i\) arc-consistent relative to \(x_j\)

1. For each \(a_i \in D_i\)
2. If there is no \(a_j \in D_j\) such that \((a_i, a_j) \in R_{ij}\)
3. Then delete \(a_i\) from \(D_i\)
4. Endif
5. Endfor

\[\boxtimes \Rightarrow \boxed{}\]

Figure 3.2: The Revise procedure

\[D_i \leftarrow D_i \cap \pi_i (R_{ij} \otimes D_j)\]
**Revise for Arc-Consistency**

\[ \text{REVISE}((x_i, x_j)) \]

**input:** a subnetwork defined by two variables \( X = \{x_i, x_j\} \), a distinguished variable \( x_i \), domains: \( D_i \) and \( D_j \), and constraint \( R_{ij} \)

**output:** \( D_i \), such that, \( x_i \) arc-consistent relative to \( x_j \)

1. for each \( a_i \in D_i \)
2. \( \text{if} \) there is no \( a_j \in D_j \) such that \((a_i, a_j) \in R_{ij}\)
3. \( \text{then} \) delete \( a_i \) from \( D_i \)
4. endif
5. endfor

**Complexity?**

\( O(k^2) \)

\[ D_i \leftarrow D_i \cap \pi_i(R_{ij} \bowtie D_j) \]

Figure 3.2: The Revise procedure
A matching diagram describing a network of constraints that is not arc-consistent (b) An arc-consistent equivalent network.
A matching diagram describing a network of constraints that is not arc-consistent (b) An arc-consistent equivalent network.
AC-1

AC-1(\mathcal{R})

input: a network of constraints \mathcal{R} = (X, D, C)

output: \mathcal{R}' which is the loosest arc-consistent network equivalent to \mathcal{R}

1. repeat
2. for every pair \{x_i, x_j\} that participates in a constraint
3. \hspace{1cm} Revise((x_i, x_j)) (or \hspace{1cm} D_i \leftarrow D_i \cap \pi_i(R_{ij} \bowtie D_j))
4. \hspace{1cm} Revise((x_j, x_i)) (or \hspace{1cm} D_j \leftarrow D_j \cap \pi_j(R_{ij} \bowtie D_i))
5. endfor
6. until no domain is changed

- **Proof:**
  - Convergence?
  - Completeness?

Figure 3.4: Arc-consistency-1 (AC-1)
AC-1

AC-1(\mathcal{R})

input: a network of constraints \( \mathcal{R} = (X, D, C) \)

output: \( \mathcal{R}' \) which is the loosest arc-consistent network equivalent to \( \mathcal{R} \)

1. repeat
2. for every pair \( \{x_i, x_j\} \) that participates in a constraint
3. Revise((x_i), x_j) (or \( D_i \leftarrow D_i \cap \pi_i(R_{ij} \Join D_j) \))
4. Revise((x_j), x_i) (or \( D_j \leftarrow D_j \cap \pi_j(R_{ij} \Join D_i) \))
5. endfor
6. until no domain is changed

Figure 3.4: Arc-consistency-1 (AC-1)

- **Complexity** (Mackworth and Freuder, 1986):
  - \( e = \) number of arcs, \( n \) variables, \( k \) values
  - \((ek^2\) each loop, \( nk \) number of loops), best-case = \( ek \)
  - Arc-consistency is: \( \Omega(ek^2) \)
  - Complexity of AC-1: \( O(enk^3) \)
AC-3

AC-3(\(\mathcal{R}\))

input: a network of constraints \(\mathcal{R} = (X, D, C)\)
output: \(\mathcal{R}'\) which is the largest arc-consistent network equivalent to \(\mathcal{R}\)

1. for every pair \(\{x_i, x_j\}\) that participates in a constraint \(R_{ij} \in \mathcal{R}\)
2. \(\text{queue} \leftarrow \text{queue} \cup \{(x_i, x_j), (x_j, x_i)\}\)
3. endfor
4. while \(\text{queue} \neq \{\}\)
5. select and delete \((x_i, x_j)\) from \(\text{queue}\)
6. \(\text{Revise}((x_i), x_j)\)
7. if \(\text{Revise}((x_i), x_j)\) causes a change in \(D_i\)
8. \(\text{then queue} \leftarrow \text{queue} \cup \{(x_k, x_i), i \neq k\}\)
9. endif
10. endwhile

Figure 3.5: Arc-consistency-3 (AC-3)

- Complexity: \(O(ek^3)\)
- Best case \(O(ek)\), since each arc may be processed in \(O(2k)\)
Exercise: Apply Arc-Consistency in Class

- Draw the network’s primal and dual constraint graph
- Network =
  - Domains \{1,2,3,4\}
  - Constraints: \( y < x, \ z < y, \ t < z, \ f < t, \ x \leq t+1, \ Y < f+2 \)
  - Apply AC-3?
AC-4 (just FYI)

AC-4(\mathcal{R})

input: a network of constraints \mathcal{R}
output: An arc-consistent network equivalent to \mathcal{R}

1. Initialization: \( M \leftarrow \emptyset \),
2. initialize \( S(x_i, a_i) \), \( \text{counter}(i, a_i) \) for all \( R_{ij} \)
3. for all counters
4. \hspace{1em} if \( \text{counter}(x_i, a_i, x_j) = 0 \) (if \( < x_i, a_i > \) is unsupported by \( x_j \))
5. \hspace{1em} then add \( < x_i, a_i > \) to \( \text{LIST} \)
6. endif
7. endfor
8. while \( \text{LIST} \) is not empty
9. \hspace{1em} choose \( < x_i, a_i > \) from \( \text{LIST} \), remove it, and add it to \( M \)
10. for each \( < x_j, a_j > \) in \( S(x_i, a_i) \)
11. \hspace{1em} decrement \( \text{counter}(x_j, a_j, x_i) \)
12. \hspace{1em} if \( \text{counter}(x_j, a_j, x_i) = 0 \)
13. \hspace{1em} then add \( < x_j, a_j > \) to \( \text{LIST} \)
14. endif
15. endfor
16. endwhile

- Complexity: \( O(ek^2) \)
- (Counter is the number of supports to \( a_i \) in \( x_i \) from \( x_j \). \( S(x_i, a_i) \) is the set of pairs that \( (x_i, a_i) \) supports)
Example applying AC-4

Example 3.2.9 Consider the problem in Figure 3.6. Initializing the $S_{x,a}$ arrays (indicating all the variable-value pairs that each $< x, a >$ supports), we have:

$S_{(z,2)} = \{ < x, 2 >, < y, 2 >, < y, 4 > \}$, $S_{(z,5)} = \{ < x, 5 > \}$, $S_{(x,2)} = \{ < z, 2 > \}$, $S_{(x,5)} = \{ < z, 5 > \}$, $S_{(y,2)} = \{ < z, 2 > \}$, $S_{(y,4)} = \{ < z, 2 > \}$.

For counters we have: $counter(x, 2, z) = 1$, $counter(x, 5, z) = 1$, $counter(z, 2, x) = 1$, $counter(z, 5, x) = 1$, $counter(z, 2, y) = 2$, $counter(z, 5, y) = 0$, $counter(y, 2, z) = 1$, $counter(y, 4, z) = 1$. (Note that we do not need to add counters between variables that are not directly constrained, such as $x$ and $y$.) Finally, $List = \{ < z, 5 > \}$, $M = \emptyset$. Once $< z, 5 >$ is removed from $List$ and placed in $M$, the counter of $< x, 5 >$ is updated to $counter(x, 5, z) = 0$, and $< x, 5 >$ is placed in $List$. Then, $< x, 5 >$ is removed from $List$ and placed in $M$. Since the only value it supports is $< z, 5 >$ and since $< z, 5 >$ is already in $M$, the $List$ remains empty and the process stops. 

□
Arc-Consistency Algorithms

- AC-1: brute-force, distributed \( O(nek^3) \)
- AC-3, queue-based \( O(ek^3) \)
- AC-4, context-based, optimal \( O(ek^2) \)
- AC-5,6,7,.... Good in special cases
- Important: applied at every node of search

\( n= \) number of variables, \( e= \) #constraints, \( k= \) domain size

From Arc-Consistency to Relational Arc-Consistency

- Sound
- Incomplete
- Always converges (polynomial)
Relational Distributed Arc-Consistency

**Primal**

- **A**
  - 1
  - 2
  - 3
- **B**
  - 1
  - 2
  - 3
- **C**
  - 1
  - 2
  - 3
- **D**
  - 1
  - 2
  - 3

**Constraints**
- A < B
- B = C
- A < D
- D < C

**Dual**

- **AB**
  - 1 2
  - 2 3
- **BC**
  - 1 2
  - 3 3
- **AD**
  - 1 2
  - 2 3
- **DC**
  - 1 2
  - 3 3

**Constraints**
- AB < BC
- AD < DC

class2 828X 2019
All Arc-consistent algorithms converge to an equivalent and loosest arc-consistent network!!!
Constraint Checking

\[ A \prec B \]

\[ 2 \quad [4 \ldots 10] \]

\[ 2 < C - A < 5 \]

\[ 6 \quad [4 \ldots 15] \]

\[ 3- B: \quad [5 \ldots 13] \]

\[ C: \quad [6 \ldots 14] \]

\[ 1- B: \quad [5 \ldots 14] \]

\[ C: \quad [6 \ldots 15] \]

\[ 2- A: \quad [2 \ldots 10] \]

\[ C: \quad [6 \ldots 14] \]
Is Arc-Consistency Enough?

- Example: a triangle graph-coloring with 2 values.
  - Is it arc-consistent?
  - Is it consistent?
- It is not path, or 3-consistent.
Outline

- Arc-consistency algorithms
- Path-consistency and i-consistency
- Arc-consistency, Generalized arc-consistency, relation arc-consistency
- Global and bound consistency
- Distributed (generalized) arc-consistency
- Consistency operators: join, resolution, Gaussian elimination
Path-Consistency

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

Figure 3.8: (a) The matching diagram of a 2-value graph coloring problem. (b) Graphical picture of path-consistency using the matching diagram.
Example: Path-Consistency

Figure 3.12: A graph-coloring graph (a) before path-consistency (b) after path-consistency
REVISE-3((x, y), z)

**input:** a three-variable subnetwork over (x, y, z), R_{xy}, R_{yz}, R_{xz}.

**output:** revised R_{xy} path-consistent with z.

1. for each pair \((a, b) \in R_{xy}\)
2. if no value \(c \in D_z\) exists such that \((a, c) \in R_{xz}\) and \((b, c) \in R_{yz}\)
3. then delete \((a, b)\) from \(R_{xy}\).
4. endif
5. endfor

![Figure 3.9: Revise-3](image)

\[ R_{ij} \leftarrow R_{ij} \cap \pi_{ij} (R_{ik} \otimes D_k \otimes R_{kj}) \]

- **Complexity:** \(O(k^3)\)
- **Best-case:** \(O(t)\)
- **Worst-case:** \(O(tk)\)
PC-1

PC-1(\mathcal{R})

**input:** a network \( \mathcal{R} = (X, D, C) \).
**output:** a path consistent network equivalent to \( \mathcal{R} \).

1. repeat
2. \hspace{1em} for \( k \leftarrow 1 \) to \( n \)
3. \hspace{2em} for \( i, j \leftarrow 1 \) to \( n \)
4. \hspace{3em} \( R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \bowtie D_k \bowtie R_{kj})/\ast (\text{Revise} - 3((i, j), k)) \)
5. \hspace{2em} endfor
6. \hspace{1em} endfor
7. until no constraint is changed.

\[
\text{Figure 3.10: Path-consistency-1 (PC-1)}
\]

- **Complexity:** \( O(n^5 k^5) \)
- \( O(n^3) \) triplets, each take \( O(k^3) \) steps \( \rightarrow O(n^3 k^3) \)
- Max number of loops: \( O(n^2 k^2) \)

Spring 2014
PC-3(\mathcal{R})

\textbf{input:} a network \( \mathcal{R} = (X, D, C) \).
\textbf{output:} \( \mathcal{R}' \) a path consistent network equivalent to \( \mathcal{R} \).

1. \( Q \leftarrow \{(i, k, j) \mid 1 \leq i < j \leq n, 1 \leq k \leq n, k \neq i, k \neq j \} \)
2. \textbf{while} \( Q \) is not empty
3. \hspace{1em} select and delete a 3-tuple \((i, k, j)\) from \( Q \)
4. \hspace{1em} \( R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \Join D_k \Join R_{kj}) \) /* (Revise-3((i, j), k))
5. \hspace{1em} \textbf{if} \( R_{ij} \) changed then
6. \hspace{1em} \hspace{1em} \( Q \leftarrow Q \cup \{((l, i, j)(l, j, i) \mid 1 \leq l \leq n, l \neq i, l \neq j\} \)
7. \hspace{1em} \textbf{endwhile}

Figure 3.11: Path-consistency-3 (PC-3)

- **Complexity:** \( O(n^3 k^5) \)
- **Optimal PC-4:** \( O(n^3 k^3) \)
- (each pair deleted may add: 2n-1 triplets, number of pairs: \( O(n^2 k^2) \) \( \rightarrow \) size of \( Q \) is \( O(n^3 k^2) \), processing is \( O(k^3) \))
Path-consistency Algorithms

- Apply Revise-3 \((O(k^3))\) until no change

\[
R_{ij} \leftarrow R_{ij} \cap \pi_{ij} (R_{ik} \bowtie D_k \bowtie R_{kj})
\]

- Path-consistency (3-consistency) adds binary constraints.
  - PC-1: \(O(n^5 k^5)\)
  - PC-2: \(O(n^3 k^5)\)
  - PC-4 optimal: \(O(n^3 k^3)\)
**Local i-Consistency**

**i-consistency:** Any consistent assignment to any i-1 variables is consistent with at least one value of any i-th variable

Figure 3.17: The scope of consistency enforcing: (a) arc-consistency, (b) path-consistency, (c) i-consistency
Directional i-Consistency

E: E \neq D, E \neq C, E \neq B
D: D \neq C, D \neq A
C: C \neq B
B: A \neq B
A:

Adaptive

\[ R_{DCB} \]

\[ R_{DC}, R_{DB} \]
\[ R_{CB} \]

\[ R_D \]
\[ R_C \]
\[ R_D \]
Boolean Constraint Propagation

- \((A \lor \neg B)\) and \((B)\)
  - \(B\) is arc-consistent relative to \(A\) but not vice-versa
- Arc-consistency by resolution:
  \[
  \text{res}((A \lor \neg B), B) = A
  \]

Given also \((B \lor C)\), path-consistency:

\[
\text{res}((A \lor \neg B), (B \lor C)) = (A \lor C)
\]

Relational arc-consistency rule = unit-resolution

\[
A \land B \rightarrow G, \neg G, \Rightarrow \neg A \lor \neg B
\]

class2 828X 2019
Gaussian 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
  \[ (A \lor B \lor \neg C), (\neg B) \ \Rightarrow \ (A \lor \neg C) \]
Unit Propagation

Procedure UNIT-PROPAGATION
Input: A cnf theory, $\varphi$, $d = Q_1, \ldots, Q_n$.
Output: An equivalent theory such that every unit clause
does not appear in any non-unit clause.
1. queue = all unit clauses.
2. while queue is not empty, do.
3. $T \leftarrow$ next unit clause from Queue.
4. for every clause $\beta$ containing $T$ or $\neg T$
5. if $\beta$ contains $T$ delete $\beta$ (subsumption elimination)
6. else, For each clause $\gamma = \text{resolve}(\beta, T)$.
   if $\gamma$, the resolvent, is empty, the theory is unsatisfiable.
7. else, add the resolvent $\gamma$ to the theory and delete $\beta$.
   if $\gamma$ is a unit clause, add to Queue.
8. endfor.
9. endwhile.

Theorem 3.6.1 Algorithm UNIT-PROPAGATION has a linear time complexity.
Variable Elimination

Eliminate variables one by one: "constraint propagation"

Solution generation after elimination is backtrack-free.
Road Map

- Graphical models
- Constraint networks Model
- Inference
  - Variable elimination:
  - Tree-clustering
  - Constraint propagation
- Search
- Probabilistic Networks