Tree Decomposition Methods

Summary

- Acyclic Networks
- Cluster Tree Elimination

Importance of Acyclic Networks

Solving Acyclic Network

- Topological structure define key features for a wide class of problems
- CSP: Inference in acyclic network is extremely efficient (polynomial)
- Idea: remove cycles from the network somehow
- We can always compile a cyclic graph into an acyclic tree-like structure
- We always pay a price in term of computational complexity
- The price we pay depends on the topology of he problem

Graph Concept: Brief Review

Hypergraphs

- Hypergraphs: $H = (V, S)$
  - Vertices: $V = \{v_1, \cdots, v_n\}$
  - Hyperegdes: $S = \{S_1, \cdots, S_k\}$ where $S_i \subseteq V$

Example (Hypergraph)

- $V = \{A, B, C, D, E, F\}$
- $S = \{\{A, E, F\}\{A, B, C\}\{C, D, E\}\{A, C, E\}\}$

Graph Concept: Bries Review

Primal Graph

- Primal Graph of a Hypergraph
  - Nodes $\rightarrow$ Vertices
  - Two nodes connected iff they appear in the same hyperedge
- For binary contraint networks, Hypergraph and Primal graph are identical

Example (Primal Graph)

- $V = \{A, B, C, D, E, F\}$
- $E = \{\{A, B\}\{A, C\}\{B, C\}\{A, E\}\{A, F\}$ $\{E, F\}\{C, D\}\{C, E\}\{D, E\}\}$

Graph Concept: Bries Review

Dual Graph

- Dual Graph of a Hypergraph
  - Nodes $\rightarrow$ Hyperedges
  - Two nodes connected iff they share at least one vertex
  - Edges are labeled by the shared vertices

Example (Dual Graph)

- $V = \{\{A, E, F\}\{A, B, C\}\{C, D, E\}\{A, C, E\}\}$
- $E = \{\{\{A, E, F\}\{A, B, C\}\}\{\{A, E, F\}, \{C, D, E\}\}$
  $\{\{A, E, F\}, \{A, C, E\}\}\{\{A, B, C\}, \{C, D, E\}\}$
  $\{\{C, D, E\}, \{A, C, E\}\}\{\{A, B, C\}, \{A, C, E\}\}\}$

Constraint Networks and Graph Representation

Graph for Consraint Networks

Any constraint network can be associated with a hypergraph

- Contraint network $\mathcal{R} = \{X, D, C\}$ with $C = \{R_{S_1}, \cdots, R_{S_r}\}$
- Hypergraph $\mathcal{H}_{\mathcal{R}} = (X, H)$ where $H = \{S_1, \cdots, S_r\}$
- Dual Graph $\mathcal{H}^d_{\mathcal{R}} = (H, E)$ where $< S_i, S_j > \in E$ iff $S_i \cap S_j \neq \{\}$
- Dual Problem $\mathcal{R}^d = \{H, D', C'\}$
  - $D' = \{D'_1, \cdots, D'_r\}$, D'_i set of tuples accepted by R_{S_i}
  - $C' = \{C'_1, \cdots, C'_k\}$, $C'_k = < S_i, S_j >$, enforces equality for the set variables $X_k = S_i \cap S_j$

Acyclicity of Constraint Network

Acyclic Network

- If the graph representation of a problem is acyclic then we can solve problem efficiently
- Even cyclic graphs can have a tree-like structure relative to solution techniques
- Some arc could be redundant
- In general it is hard to recognise redundant constraints

Acyclicity of Dual Problem

Redundant Constraints for Dual Problems

- For the dual graph representation checking whether a constraint is redundant is actually easy
- All constraints force equality over shared variables
- A constraint and its corresponding arc can be deleted if the variables labeling the arc are contained in an alternative path between the two endpoints
- Because the constraint will be enforced by the other paths.
- This property is called running intersection or connectedness

Example: Acyclicity of Dual Problem

Example (Acyclic Dual Problem)

Consider this dual graph:

- $V = \{\{A, E, F\}\{A, B, C\}\{C, D, E\}\{A, C, E\}\}$
- $E = \{\{\{A, E, F\}\{A, B, C\}\}\{\{A, E, F\}, \{C, D, E\}\}$ $\{\{A, E, F\}, \{A, C, E\}\}\{\{A, B, C\}, \{C, D, E\}\}$ $\{\{C, D, E\}, \{A, C, E\}\}\}$

We can remove redundant constraints:

- $\{\{A, E, F\}\{A, B, C\}\}$ because the alternative path $(AEF) - AE - (ACE) - AC - (ABC)$ enforce constraint on A
- $\{\{A, E, F\}\{C, D, E\}\}$ because the alternative path $(AEF) - AE - (ACE) - CE - (CDE)$ enforce constraint on E
- $\{\{C, D, E\}\{A, B, C\}\}$ because the alternative path $(CDE) - CE - (ACE) - AC - (ABC)$ enforce constraint on C

The remaining structure is a tree

Acyclic Network

Main Concepts

- Arc Subgraph of a graph $G = \{V, E\}$: any graph $G' = \{V, E'\}$ such that $E' \subseteq E$
- Running Intersection property: G dual graph of an hypergraph, G' an arc subgraph satisfies the running intersection properties if given any two nodes of G' that share a variable, there exists a path of labeled arcs, each containing the variable.
- Join Graph: an arc subgraph of the dual graph that satisfies the running intersection properties
- Join Tree: an acyclic join graph
- Hypertree: a Hypergraph whose dual graph has a join tree
- Acyclic Network: a network whose hypergraph is an hypertree

Solving Acyclic Network

Algorithm for Solving Acyclic Network

Algorithm 1 Tree Solver

Require: An Acyclic Constraint Network $\mathcal{R}$, A join-tree T of $\mathcal{R}$

Ensure: Determine consistency and generate a solution

$d = \{R_1, \cdots, R_r\}$ order induced by T (from root to leaves)

for all $j = r$ to 1 and for all edges $< j, k >$ in the T with $k < j$ **do**

  $R_k \leftarrow \pi_{S_k}(R_K \bowtie R_j)$

  if we find the empty relation **then**

    EXIT and state the problem has NO SOLUTION

  end if

end for

Select a tuple in R_1

for all $i = 2$ to r **do**

  Select a tuple that is consistent with all previous assigned tuples $R_1, \cdots, R_{i-1}$

end for

return The problem is CONSISTENT return the selected SOLUTION

Example: Solving Acyclic Problem

Example (Applying Tree Solver)

Consider this join-tree:

- $V = \{\{A, E, F\}\{A, B, C\}\{C, D, E\}\{A, C, E\}\}$
- $E = \{\{\{A, E, F\}, \{A, C, E\}\}\{\{C, D, E\}, \{A, C, E\}\}$ $\{\{A, B, C\}, \{A, C, E\}\}\}$

Assume constraints are given by

- $R_{ABC} = R_{AEF} = \{(0, 0, 1)(0, 1, 0)(1, 0, 0)\}$
- $R_{CDE} = R_{ACE} = \{(1, 1, 0)(0, 1, 1)(1, 0, 1)\}$
- $d = \{R_{ACE}, R_{CDE}, R_{AEF}, R_{ABC}\}$

Recognising Acyclic Networks

Main methods

- To apply the tree solver algorithm we need to know whether a network is acyclic
- This can not be decided simply by checking whether there are cycles in the primal or dual graph
- Two main methods
  - Primal based Recognition
  - Dual based Recognition

Primal Based Recognition

Primal Based Recognition: main concepts

- A hypergraph has a join tree iff its primal graph is chordal and conformal [Maier 1983]
- Conformal A primal graph is conformal to a hypergraph if there is a one to one mapping between maximal cliques and scopes of constraints
- Chordal A primal graph is chordal if every cycle of length at least 4 has a chord (an edge connecting two vertices that are non adjacent in the cycle)
- Checking whether a graph is chordal and conformal can be done efficiently using a max-cardinality order

Primal Based Recognition using max cardinality order

max cardinality order

- max-cardinality order is an ordering over vertices such that:
  - first node is chosen arbitrarily
  - then the node that is connected to a maximal number of already ordered nodes is selected (breaking ties arbitrarily)
- Chordal Graph if in a max-cardinality order each vertex and all its ancestors form a clique
- Find Maximal clique just list nodes in the order and consider each node ancestors

Primal Based Recognition: Procedure

Main idea

1. build a max-cardinality order
2. Test whether the graph is chordal
   - use the max-cardinality order
   - check if ancestors form a clique
3. Test whether the graph is conformal
   - use the max-cardinality order
   - extract maximal cliques, check conformality

Primal Based Recognition: algorithm

Primal Acyclicity

Algorithm 2 PrimalAcyclicicty

Require: A constraint network $\mathcal{R} = (X, D, C)$ and its primal graph G
Ensure: A join tree $T = (S, E)$ of $\mathcal{H}_{\mathcal{R}}$ if $\mathcal{R}$ is acyclic

```
Build d^m = {x_1, ..., x_n} max-cardinality order
Test Chordality using d^m:
for all i = n to 1 do
    if the ancestors of x_i are not all connected then
        EXIT (R is not acyclic)
    end if
end for
Test Conformality using d^m: Let {C_1, ..., C_r} be the maximal cliques (a node and all its ancestors)
for all i = r to 1 do
    if C_i corresponds to scope of one constraints C then
        (R is acyclic)
    else
        EXIT (R is not acyclic)
    end if
end for
Create a join tree of the cliques (e.g., create a maximum spanning tree were weights are number of shared variables)

return R is acyclic and T is a join tree
```

Example: Primal based recognition

Example (Primal based recognition)

Consider this hypergraph

- $V = \{A, B, C, D, E, F\}$
- $S = \{\{A, E, F\}\{A, B, C\}\{C, D, E\}\{A, C, E\}\}$

decide whether this constraint network is acyclic using the primal based recongition procedure.

Dual Based Recognition

Dual Based Recognition: Theoretical Result

- Maier 1983
- If a hypergraph has a join tree then any maximum spanning tree of its dual graph is a join tree
- Weight of the arc are the number of shared variables

Dual Based Recognition: Procedure

Main idea

- Build the dual graph of the hypergraph
- Compute a maximum spanning tree (weight = number of shared variables)
- Check whether the hypertree is a join tree
  - Efficient because there is only one path for each couple of nodes

Dual Based Recognition: algorithm

Dual Acyclicity

Algorithm 3 DualAcyclicicty

Require: A hypergraph $\mathcal{H}_{\mathcal{R}} = (X, S)$ of a constraint network $\mathcal{R} = (X, D, C)$

Ensure: A join tree $T = (S, E)$ of $\mathcal{H}_{\mathcal{R}}$ if $\mathcal{R}$ is acyclic

  $T = (S, E) \leftarrow$ generate a maximum spanning tree of the weighted dual constraint graph of $\mathcal{R}$

  for all couples u, v where $u, v \in S$ **do**

    if the unique path connecting them in T does not satisfy the running intersection property **then**

      EXIT ($\mathcal{R}$ is not acyclic)

    end if

  end for

  return $\mathcal{R}$ is acyclic and T is a join tree

Dual Based Recognition: Example

Example (Dual Based Recognition)

Consider this dual graph:

- $V = \{\{A, E, F\}\{A, B, C\}\{C, D, E\}\{A, C, E\}\}$
- $E = \{\{\{A, E, F\}\{A, B, C\}\}\{\{A, E, F\}, \{C, D, E\}\} \{\{A, E, F\}, \{A, C, E\}\}\{\{A, B, C\}, \{C, D, E\}\} \{\{C, D, E\}, \{A, C, E\}\}\}$

If we find a MST weighing edges with number of shared variables we obtain T:

- $V = \{\{A, E, F\}\{A, B, C\}\{C, D, E\}\{A, C, E\}\}$
- $E = \{\{\{A, E, F\}, \{A, C, E\}\}\{\{C, D, E\}, \{A, C, E\}\} \{\{A, B, C\}, \{A, C, E\}\}\}$

Which satisfies the running intersection property.

Compiling network to tree-like structures

Clustering

- Aim:
  - Compile network to acyclic structure
  - Solve the acyclic structure efficiently using a tree-solver alg.
- Clustering: grouping subsets of constraints to form a tree-like structure
- Solve each subproblem (replace the set of relations with the solution of the problem)
- Solve the acyclic network
- If all steps are tractable this process is very efficient

Clustering Approaches

Methods

- Join Tree Clustering
  - Given a constraint network
  - Computes an acyclic equivalent constraint problem
- Cluster Tree Elimination
  - More general scheme
  - Given a Tree Decomposition
  - Combine the acyclic problem solving with subproblem solution

Join Tree Clustering I

Basic Concept

- Input: Hypergraph $\mathcal{H} = \{X, H\}$, H set of scopes of constraints
- Output: Hypertree $\mathcal{S} = \{X, S\}$, and a partition of the original relations (Hyperedges) into the new hypertree edges
- $\mathcal{S}$ each edge defines a subproblem containing a constraint if its scope is contained in the hyperedge
- Each subproblem is solved independently
- Each subproblem is replaced with one constraint that has the scope of the hyperedges and accept the solution tuples of the subproblem
- The smaller the hyperdge size, the better.

Join Tree Clustering II

Basic Steps

1. Choose an order of variable
2. Create an induced graph given the ordering to ensure the running intersection property
3. Create a join tree
   - Identify all maximal cliques in the chordal graph $C_1, \cdots, C_t$
   - Create a tree structure T over the cliques (e.g., create a maximum spanning tree were weights are number of shared variables)
4. Allocate constraints to any clique that contains its scope (P_i subproblem associated with C_i).
5. Solve each P_i with R_i' its set of solutions
6. Return $C' = \{R_1', \cdots, R_t'\}$

Induced graph

Induced Graph and Induced Width

- Given graph $G :< V, E >$ and order d over V
- Ancestors: neighbours that precedes the vertices in the ordering
- G^* induced graph of G over d is obtained by:
  - process variables from last to first
  - when processing v, add edges to connect all ancestors of v
- The width of a node is the number of ancestors of the node
- The width of a graph is the maximal width of its nodes
- The induced width $w^*(d)$ of G given d is the width of G^*
- The induced width w^* of G is minimum induced width over all possible orderings

Induced Graph and chordality

Induced Graph and chordality

- A graph is chordal iff it has a perfect elimination ordering [Fulkerson and Gross 1965]
- Perfect elimination ordering: ordering of the vertices such that, for each vertex v, v and its ancestors form a clique
- An induced graph $< G^*, d >$ is chordal:
  - d is a perfect elimination ordering for G^*

Example

Example (Creating the join tree)

Consider the following graph and assume it is a primal graph of binary contraint newtork:

- Variables: A, B, C, D, E, F Edges: $(A, B)(A, C)(A, E)(B, E)(B, D)(C, D)(D, F)$

Consider the orderings

- $d_1 = F, E, D, C, B, A$
- $d_2 = A, B, C, D, E, F$

Example contd.

Example (Creating the join tree)

The resulting join trees are:

- d_1 Cliques:
  $Q_1 = (A, B, C, E), Q_2 = (B, C, D, E), Q_3 = (D, E, F)$
  Edges: $< Q_1, Q_2 >, < Q_2, Q_3 >$
- d_1 Cliques: $Q_1 = (D, F), Q_2 = (A, B, E), Q_3 = (B, C, D), Q_4 = (A, B, C)$
  Edges: $< Q_1, Q_3 >, < Q_2, Q_4 >, < Q_3, Q_4 >$

Creating the chordal graph

max-cardinality order

- Creating the chordal graph using a max-cardinality order is more efficient
- do not add useless edges if graph is already chordal

Ensuring the graph is conformal

conformality

- When finding the maximal cliques we might violate conformality
  - could create maximal cliques that have no mapping to constraints
- Conformality is enforced in later steps
  - by creating a unique constraint for each sub problem

Complexity of JTC

Complexity

- The running time of join tree clustering is dominated by computing the set of solutions of each sub problem
- This is exponential in the size of the clique
- Running time is dominated by running time to solve the subproblem of the maximal clique
- Size of maximal cliques is the induced width of the graph plus one
- The order used to compute the cliques is crucial
- Finding the best ordering is hard

Finding a Complete Solution

Constraint Propagation

- Once we have solved the subproblems we still need to
  - force arc-consisteny
  - expand local solution to a global solution (if problem is consistent)
- We can use Tree-Solver for this