Dedicated Propagation Algorithms

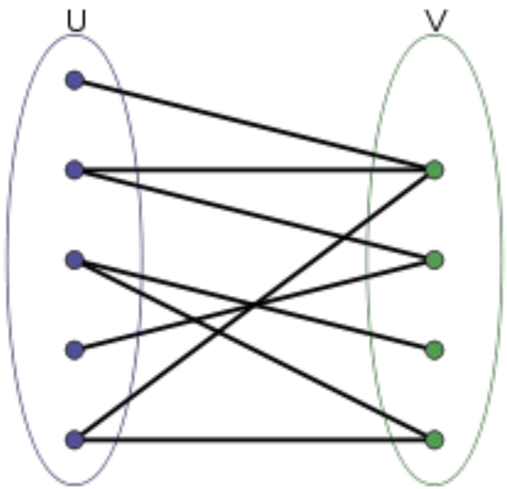
- Dedicated ad-hoc algorithms provide effective and efficient propagation.
- Often:
  - GAC is maintained in polynomial time;
  - many more inconsistent values are detected compared to the decompositions;
  - computation is done incrementally.

A GAC Propagation Algorithm

- Maintains GAC on alldifferent([X_1, X_2, …, X_k]) and runs in polynomial time.
  - Jean-Charles Régin, "A Filtering Algorithm for Constraints of Difference in CSPs", in the Proc. of AAAI'1994
- Establishes a relation between the solutions of the constraint and the properties of a graph.
  - Maximal matching in a bipartite graph.
- A similar algorithm can be obtained with the use of flow theory.

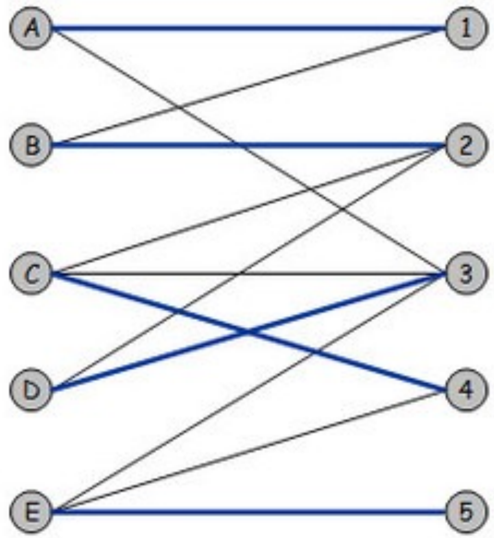
A GAC Algorithm for alldifferent

- A bipartite graph is a graph whose vertices are divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V.



A GAC Algorithm for alldifferent

- A matching in a graph is a subset of its edges such that no two edges have a node in common.
  - Maximal matching is the largest possible matching.
- In a bipartite graph, maximal matching covers one set of nodes.



A GAC Algorithm for alldifferent

- Observation
  - Given a bipartite graph G constructed between the variables [X_1, X_2, …, X_k] and their possible values (variable-value graph),
  - an assignment of values to the variables is a solution iff it corresponds to a maximal matching in G.
    - A maximal matching covers all the variables.
  - By computing all maximal matchings, we can find all the consistent partial assignments.

Example

Variable-value graph

$D_0 = \{0, 1\}$

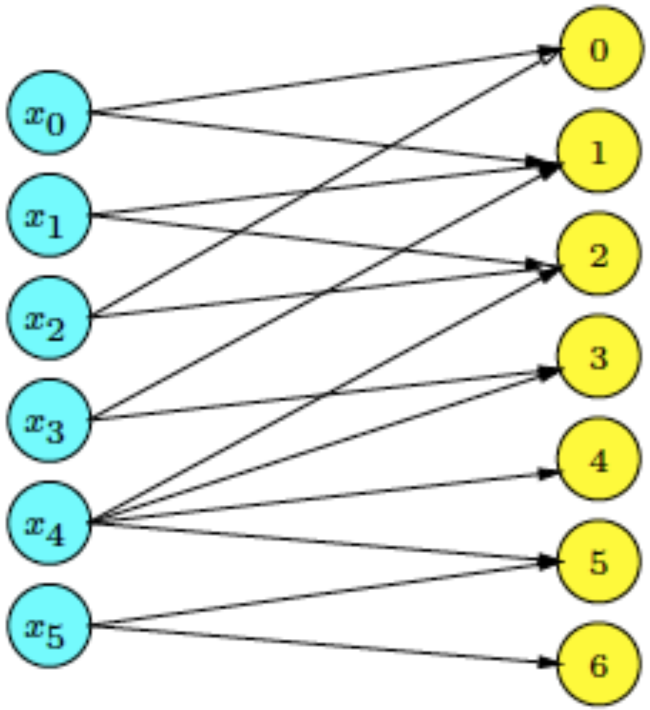
$D_1 = \{1, 2\}$

$D_2 = \{0, 2\}$

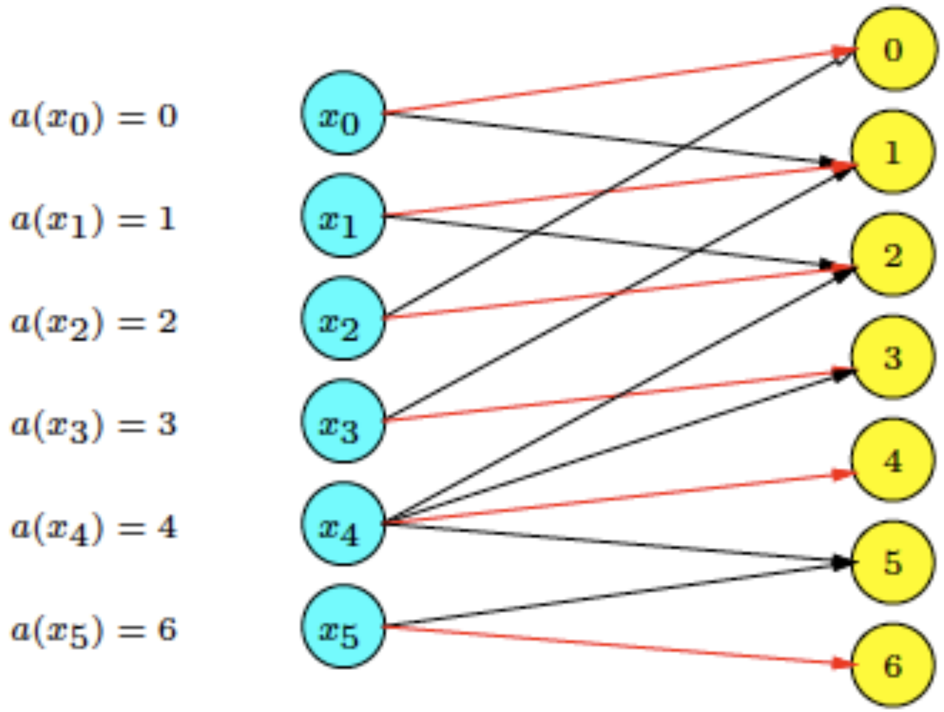
$D_3 = \{1, 3\}$

$D_4 = \{2, 3, 4, 5\}$

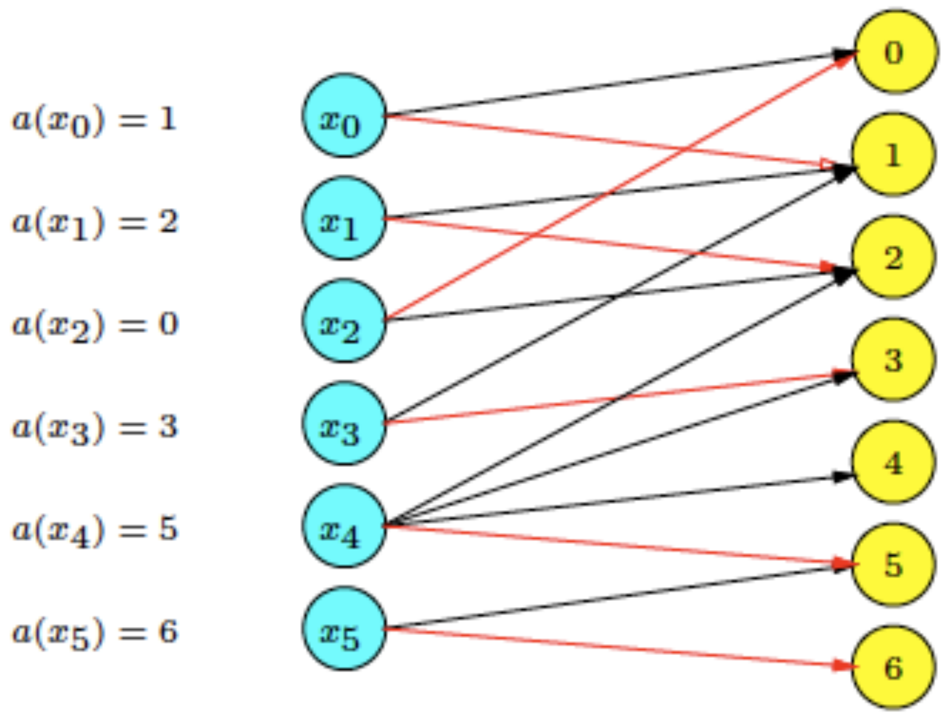
$D_5 = \{5, 6\}$



A Maximal Matching



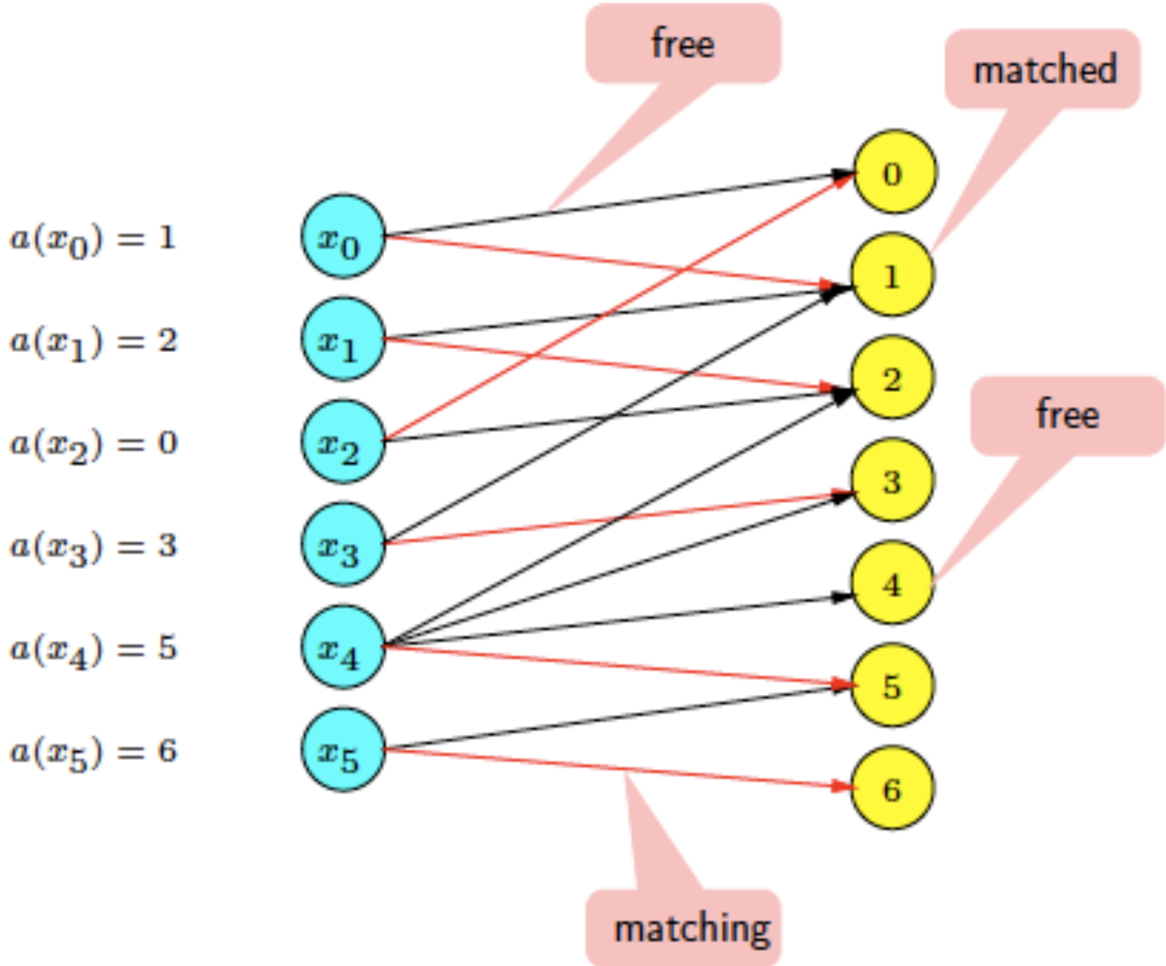
Another Maximal Matching



Matching Notations

- Edge
  - matching if takes part in a matching;
  - free otherwise.
- Node
  - matched if incident to a matching edge;
  - free otherwise.
- Vital edge
  - belongs to every maximal matching.

Free, Matched, Matching



Algorithm

- Compute all maximal matchings.
- No maximal matching exists → failure.
- An edge free in all maximal matchings →
  - Remove the edge.
  - Amounts to removing the corresponding value from the domain of the corresponding variable.
- A vital edge →
  - Keep the edge.
  - Amounts to assigning the corresponding value to the corresponding variable.
- Edges matching in some but not all maximal matchings →
  - Keep the edge.

All Maximal Matchings

- Inefficient to compute them naïvely.
- Use matching theory to compute them efficiently.
  - One maximal matching can describe all maximal matchings!

Alternating Path and Cycle

- Alternating path
  - Simple path with edges alternating free and matching.
- Alternating cycle
  - Cycle with edges alternating free and matching.
- Length of path/cycle
  - Number of edges in the path/cycle.
- Even path/cycle
  - Path/cycle of even length.

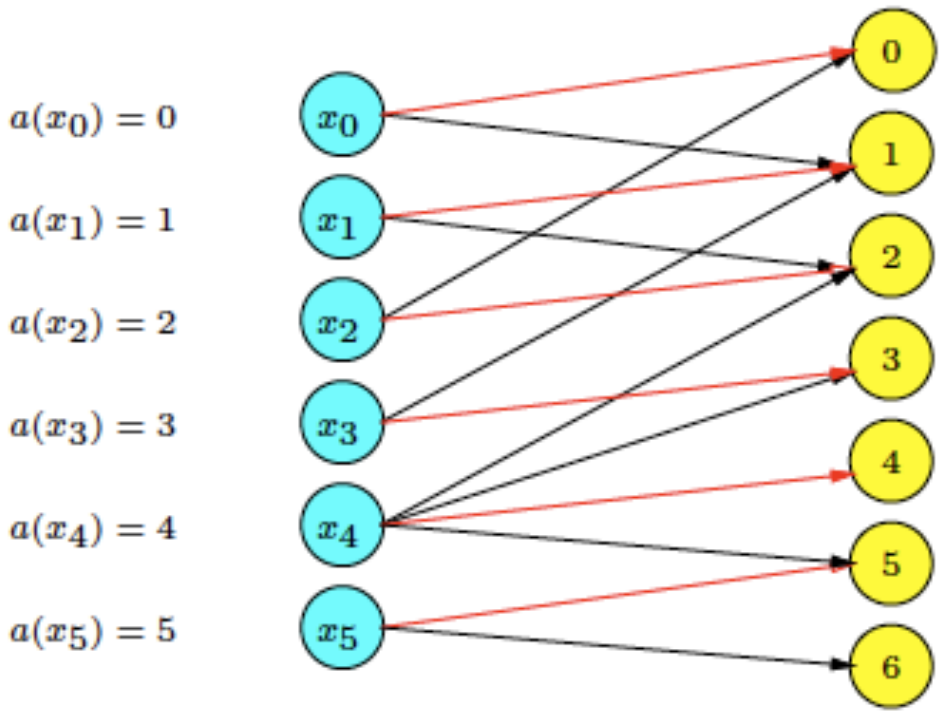
Matching Theory

- A result due to Claude Berge in 1970.
- An edge e belongs to a maximal matching iff for some arbitrary maximal matching M:
  - either e belongs to M;
  - or e belongs to even alternating path starting at a free node;
  - or e belongs to an even alternating cycle.

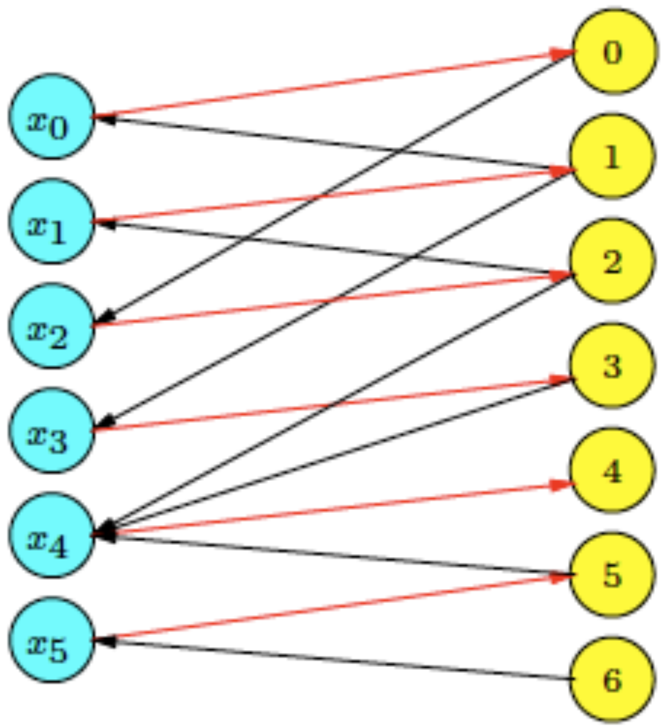
Oriented Graph

- To compute alternating path/cycles, we will orient edges of an arbitrary maximal matching:
  - matching edges → from variable to value;
  - free edges → from value to variable.

An Arbitrary Maximal Matching



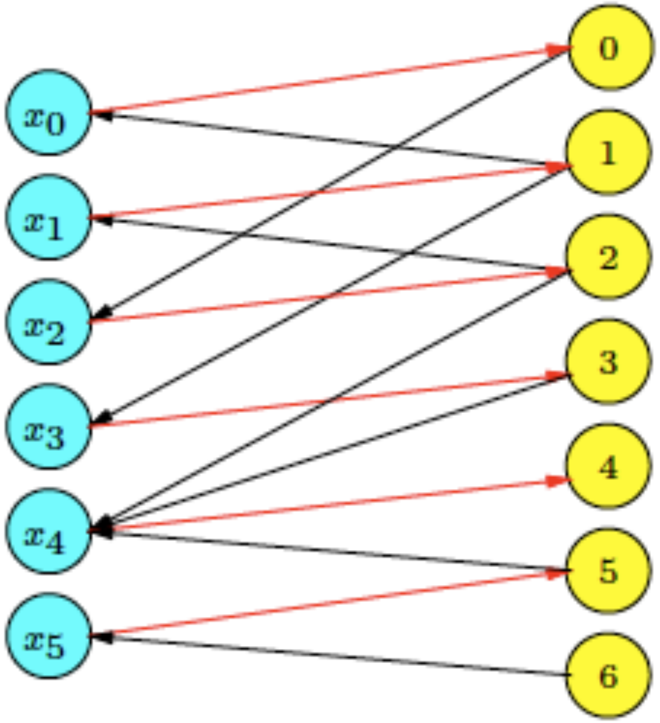
Oriented Graph

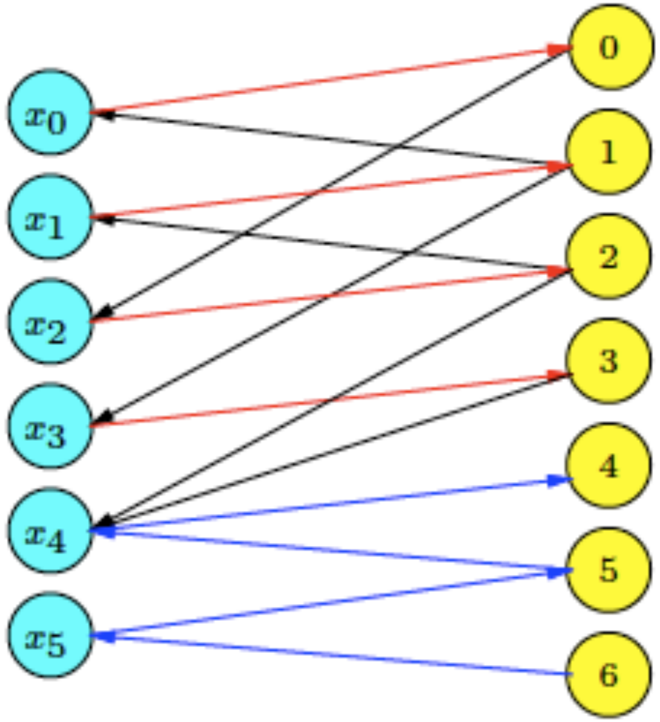


Even Alternating Paths

- Start from a free node and search for all nodes on directed simple path.
  - Mark all edges on path.
  - Alternation built-in.
- Start from a value node.
  - Variable nodes are all matched.
- Finish at a value node for even length.

Even Alternating Paths



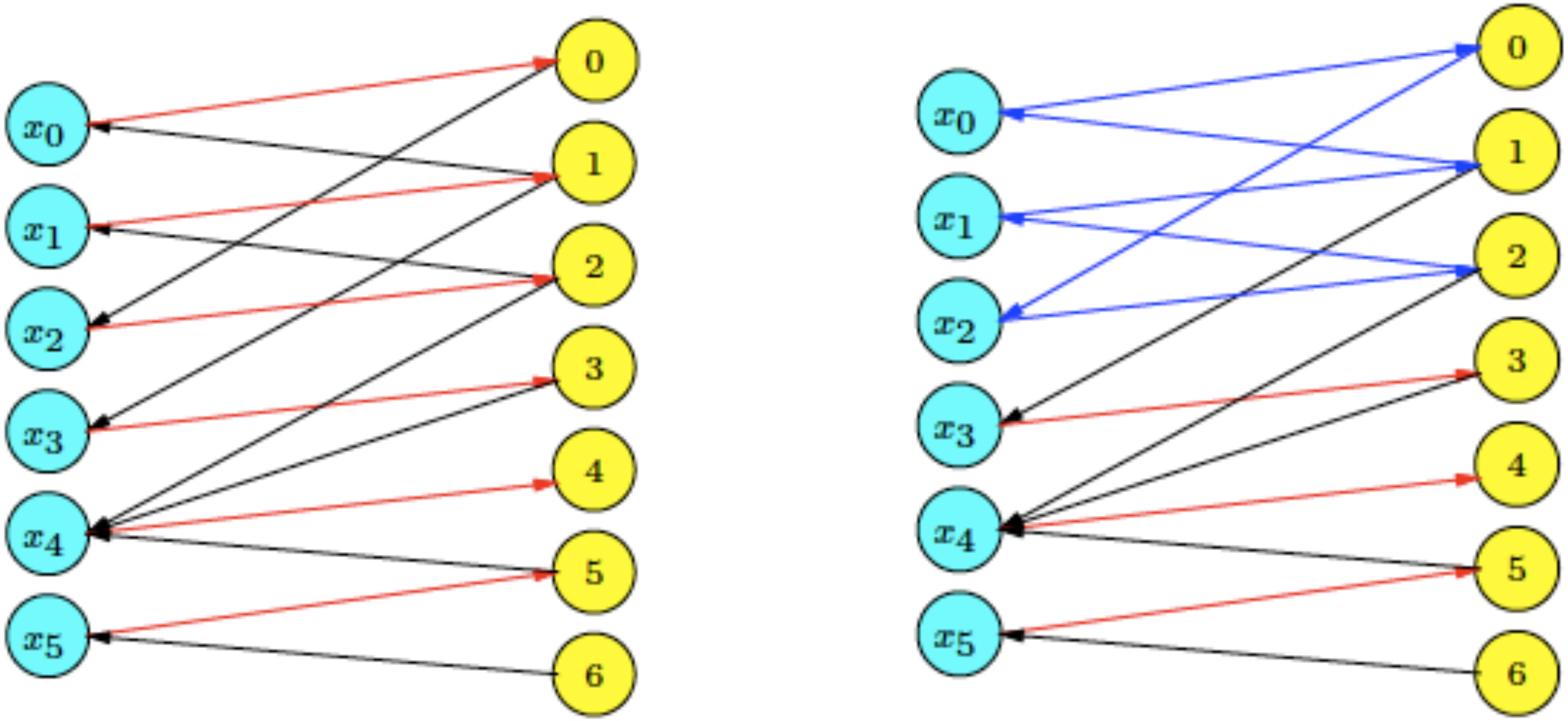


- Intuition: edges can be permuted.

Even Alternating Cycles

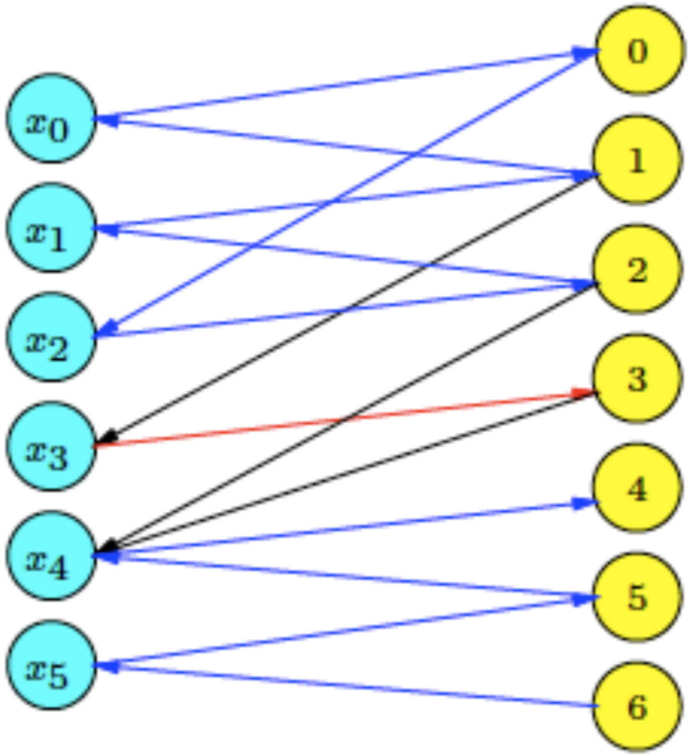
- Compute strongly connected components (SCCs).
  - Two nodes a and b are strongly connected iff there is a path from a to b and a path from b to a.
  - Strongly connected component: any two nodes are strongly connected.
  - Alternation and even length built-in.
- Mark all edges in all strongly connected components.

Even Alternating Cycles



- Intuition: variables consume all the values.

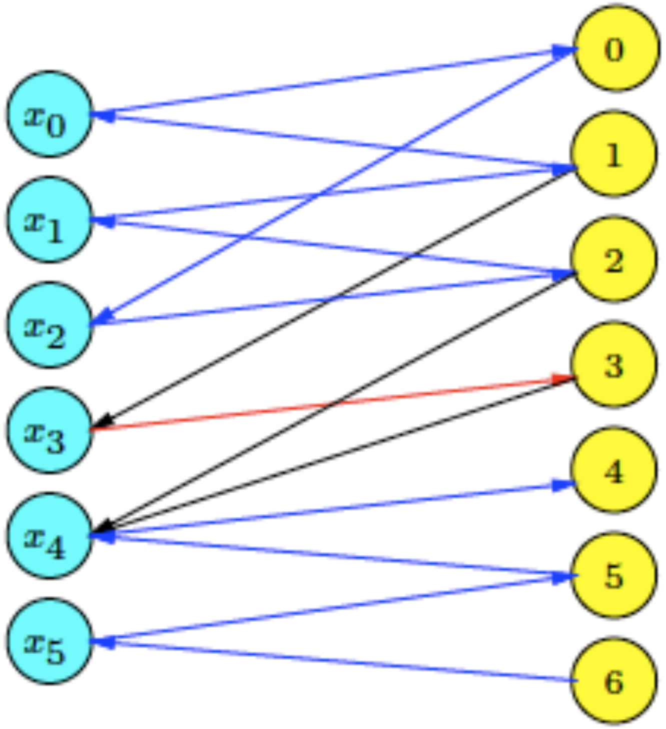
All Marked Edges



Removing Edges

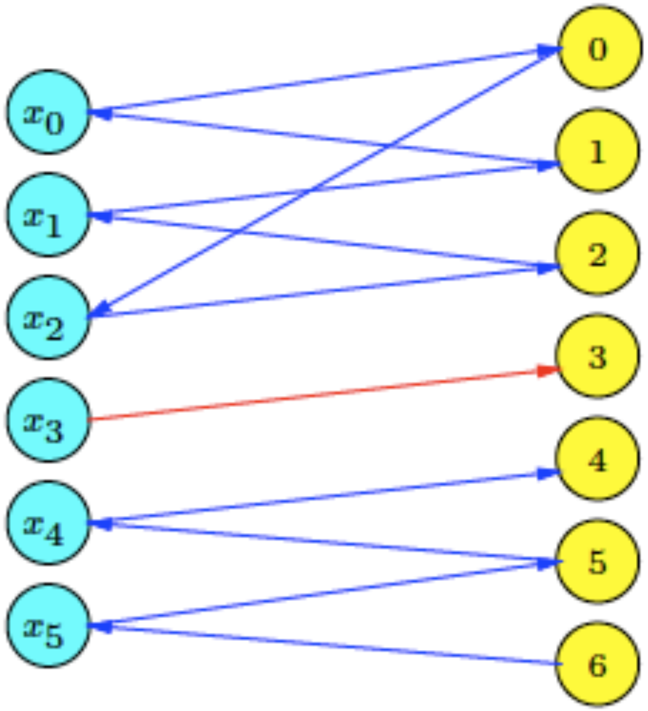
- Remove the edges which are:
  - free (does not occur in our arbitrary maximal matching) and not marked (does not occur in any maximal matching);
  - marked as black in our example.
- Keep the edge matched and not marked.
  - Marked as red in our example.
  - Vital edge!

Removing Edges



$D(X_0) = \{0,1\}$, $D(X_1) = \{1,2\}$, $D(X_2) = \{0,2\}$, $D(X_3) = \{1,3\}$
$D(X_4) = \{2,3,4,5\}$, $D(X_5) = \{5,6\}$

Edges Removed



$D(X_0) = \{0,1\}$, $D(X_1) = \{1,2\}$, $D(X_2) = \{0,2\}$, $D(X_3) = \{\cancel{1},3\}$

$D(X_4) = \{\cancel{2},\cancel{3},4,5\}$, $D(X_5) = \{5,6\}$

Summary of the Algorithm

- Construct the variable-value graph.
- Find a maximal matching M; otherwise fail.
- Orient graph (done while computing M).
- Mark edges starting from free value nodes using graph search.
- Compute SCCs and mark joining edges.
- Remove not marked and free edges.

Incremental Properties

- Keep the variable and value graph between different invocations.
- When re-executed:
  - remove marks on edges;
  - remove edges not in the domains of the respective variables;
  - if a matching edge is removed, compute a new maximal matching;
  - otherwise just repeat marking and removal.

Runtime Complexity

- alldifferent($[X_1, X_2, \ldots, X_k]$) with m = $\sum_{i \in \{1,..k\}} |D(X_i)|$
- First call
  - Consistency check in O($\sqrt{k}$m) time.
    - Matching → O($\sqrt{k}$ m)
    - Alternating path → O(m)
    - SCCs → O(k+m)
  - Establishing GAC in O(m) time.
- After q variable domains have been modified
  - Matching in O(min{qm, $\sqrt{k}$ m}) time.
  - Establishing GAC in O(m) time.

Dedicated Ad-hoc Algorithms

- Is it always easy to develop a dedicated algorithm for a given constraint?
- A nice semantics often gives us a clue!
  - Graph theory
  - Flow theory
  - Combinatorics
  - Automata theory
  - Dynamic programming
  - Complexity theory, …

Dedicated Ad-hoc Algorithms

- GAC may as well be NP-hard!
  - E.g., nvalue, sequence+gcc, gcc using variables for occurrences.
  - Algorithms which maintain weaker consistencies are of interest.
    - BC
    - Between GAC and BC
    - GAC on some variables, BC on others
    - …

Dedicated Ad-hoc Algorithms

- What if it is difficult to:
  - decompose a constraint;
  - build an efficient and effective dedicated algorithm?

Outline

- Local Consistency
  - Generalized Arc Consistency (GAC)
  - Bounds Consistency (BC)
- Constraint Propagation
  - Propagation Algorithms
- Specialized Propagation
  - Global Constraints
    - Decompositions
    - Ad-hoc Algorithms
- Global Constraints for Generic Purposes

Global Constraints for Generic Purposes

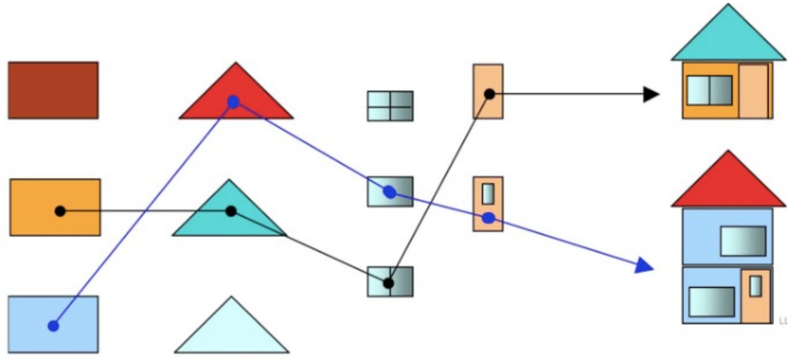
- Help propagate a wide range of constraints.
  - Table constraint.
  - Formal language-based constraints.

Table (Extensional) Constraint

- $C(X_1, X_2) = \{(0,0), (0,2), (1,3), (2,1)\}$
- Several algorithms exist to maintain GAC.
  - More efficient than $O(|D(X_1)|*|D(X_2)|*\ldots*|D(X_k)|)$.
  - More effective than the decomposition.
    - E.g., (X_1= 0 AND X_2 = 2 AND X_3 = 2) OR (X_1= 1 AND X_2 = 1 AND X_3 = 2) OR (X_1= 1 AND X_2 = 2 AND X_3 = 3)

Product Configuration Problems

- Compatibility constraints on product components.
  - Often only certain combination of components work together.
- Compatibility may not be a simple pairwise relationship.



A Configuration Problem

- Valid hw products are defined in a table of compatible components (Products):

Products	Motherboard	CPU	Freq	RAM	Hard drive
$Product_1$	TypeA	Intel	2GHz	5GB	100GB
$Product_2$	TypeB	Intel	3GHz	8GB	200GB
$Product_3$	TypeB	Amd	2GHz	5GB	200GB
…					

- Assume we have products P_i to configure each with 5 components for motherboard, CPU, Freq, RAM and h. drive $[X_{i1}, X_{i2}, X_{i3}, X_{i4}, X_{i5}]$.
- For each product P_i, we post table($[X_{i1}, X_{i2}, X_{i3}, X_{i4}, X_{i5}]$, Products).

Crossword Puzzles

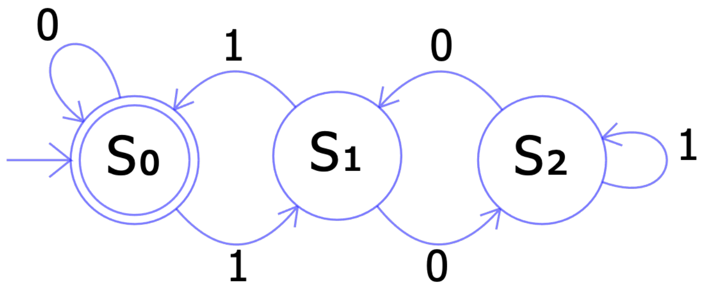
- Valid words are defined in a table of compatible letters (i.e. dictionary).
  - table([X_1,X_2,X_3], dictionary)
  - table([X_1,X_{13},X_{16}], dictionary)
  - table([X_4,X_5,X_6,X_7], dictionary)
  - …
- No simple way to decide acceptable words other than to put them in a table.

Formal Language-based Constraints

- The table constraint requires precomputing all the solutions of a constraint.
  - May not always be possible or practical.
- We can use a deterministic finite-state automaton to define the solutions.
  - Useful especially when valid assignments need to obey certain patterns.

Deterministic Finite State Automaton

- A dfsa is a finite-state machine that accepts or rejects a given string of symbols, by running through a state sequence uniquely determined by the string.
  - Recognizes a regular language.
- E.g., a dfa that accepts binary numbers that are multiples of 3.



  - Some accepted strings: 0, 11, 110, 1100, 1001, 10111101, …
  - Not accepted strings: 10, 100, 101, 10100, …

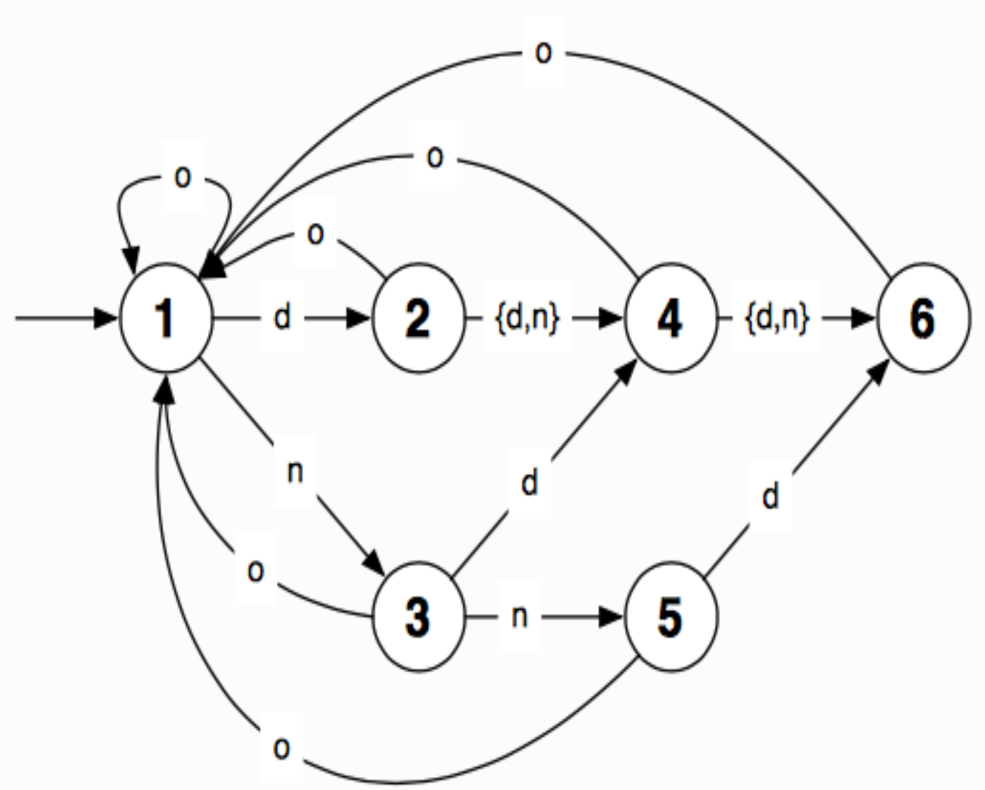
Regular Constraint

- A dfsa A is defined by a 5-tuple (q, sigma, t, q_0, f) where:
  - q : a finite set of states
  - sigma: a set of symbols (i.e. alphabet)
  - t: a partial transition function q x sigma → q
  - q_0 : initial state
  - f ⊆ q: accepting (final) states
- regular([X_1, X_2, …, X_k], A) holds iff <X_1, X_2, …, X_k> forms a string accepted by a dfsa A.

Rostering Problems

- Shifts are subject to regulations.
  - E.g., successive night shifts must be limited.
- In a nurse rostering problem, suppose:
  - each nurse is scheduled for each day either: (d) on day shift, (n) on night shift, or (o) off;
  - in each four day period, a nurse must have at least one day off;
  - no nurse can be scheduled for 3 night shifts in a row.

A Nurse Rostering Problem



- $q = \{q_1, \ldots, q_6\}$
- sigma = {d, n, o}
- t:

	d	n	o
1	2	3	1
2	4	4	1
3	4	5	1
4	6	6	1
5	6	0	1
6	0	0	1

- $q_0 : q_1$
- $f = q = \{q_1, \ldots, q_6\}$

- Assume nurses N_i to be scheduled for 30 days $[D_{i1}, \ldots, D_{i30}]$.
- For each nurse N_i, we post regular($[D_{i1}, \ldots, D_{i30}]$, A)

Regular Constraint

- Useful in sequencing and rostering problems.
- Many constraints are instances of regular:
  - among, lex, precedence, stretch, …
- Efficient GAC propagation with a dedicated algorithm and a decomposition into a sequence of ternary constraints.
  - Another example of the power of decompositions!