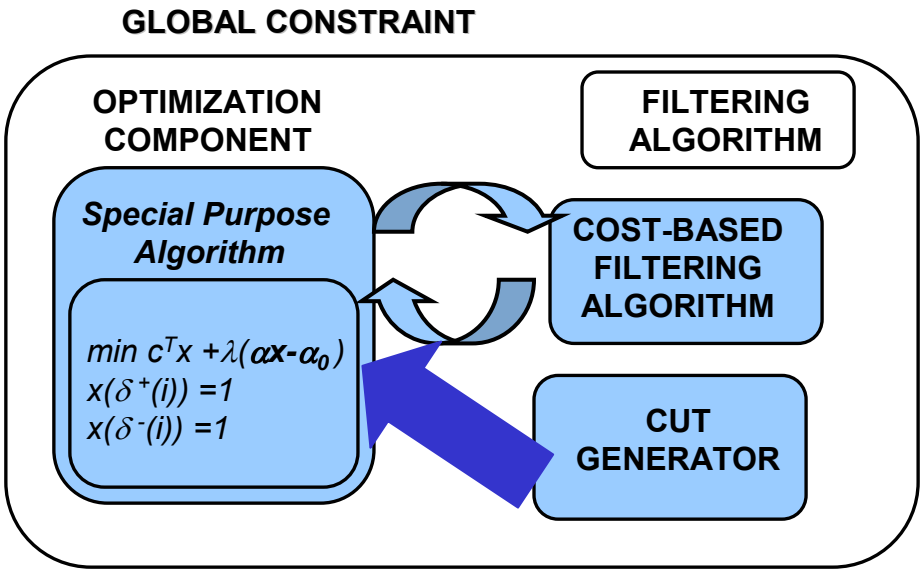
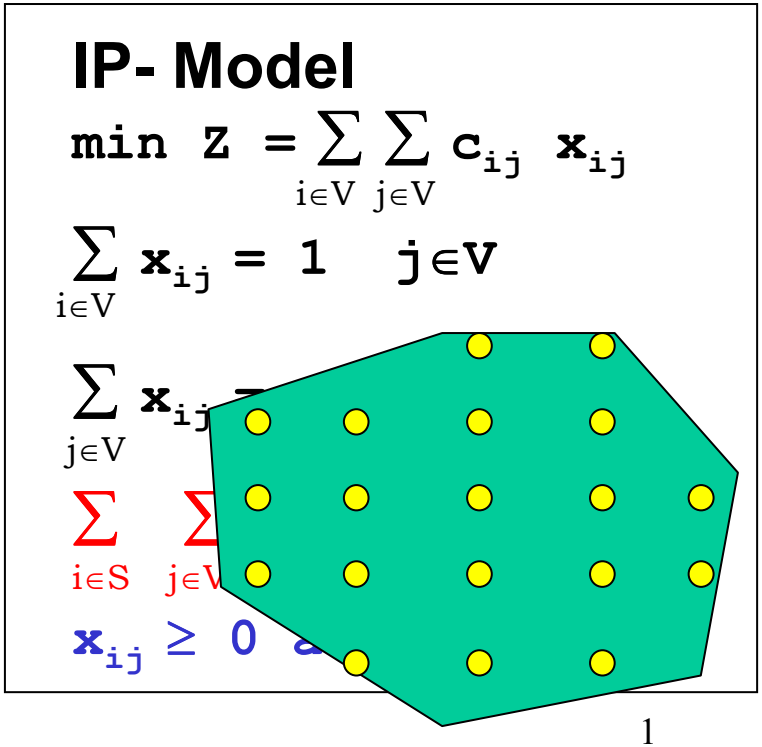
Combining Constraint Programming and Integer Programming





Constraint Programming (CP) Preliminaries

• **CP on Finite Domains** CP(FD): a programming paradigm exploiting *Constraint Satisfaction* techniques

• A **Constraint Satisfaction** Problem (CSP) consists of:

– *a set of **variables*** (V_1, V_2,…,V_n)

– *a discrete domain* (D_1,D_2,…,D_n) *for each variable*

– *a set of **constraints** on te variables:*

"relations among variables which represent a subset of the Cartesian product of the domains $D_1 \times D_2 \times \ldots \times D_n$"

Solution of a CSP:
an assignment of values to variables consistent with the constraints

E. Tsang: "Foundations of Constraint Satisfaction" Academic Press,1992.

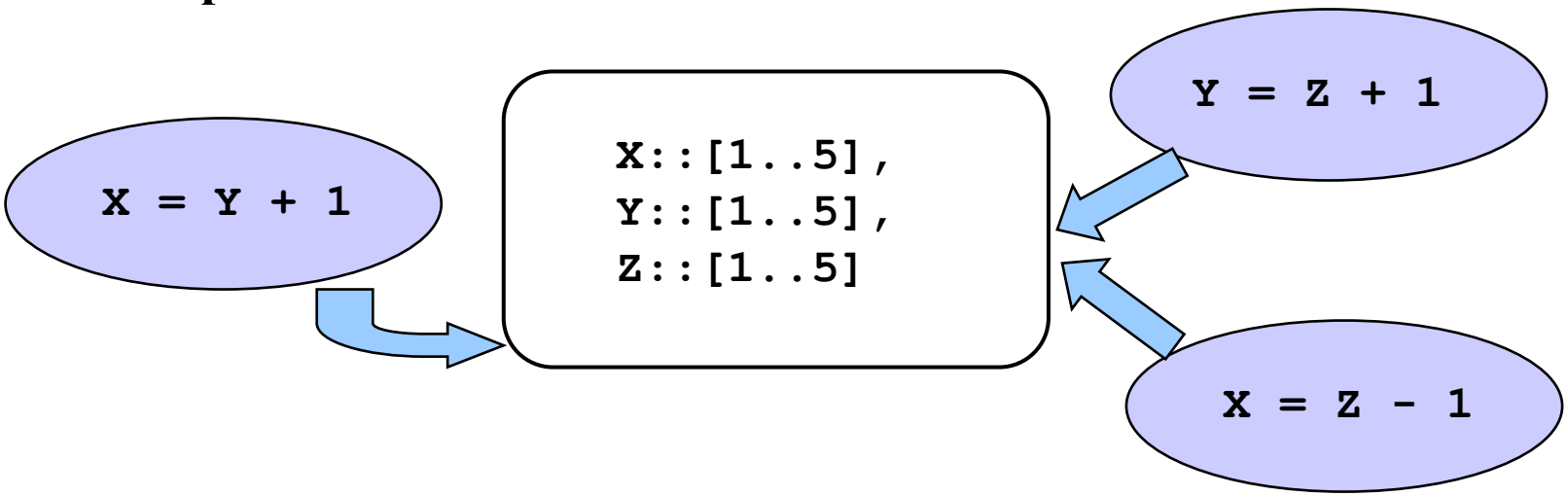
CP Preliminaries – interaction among constraints

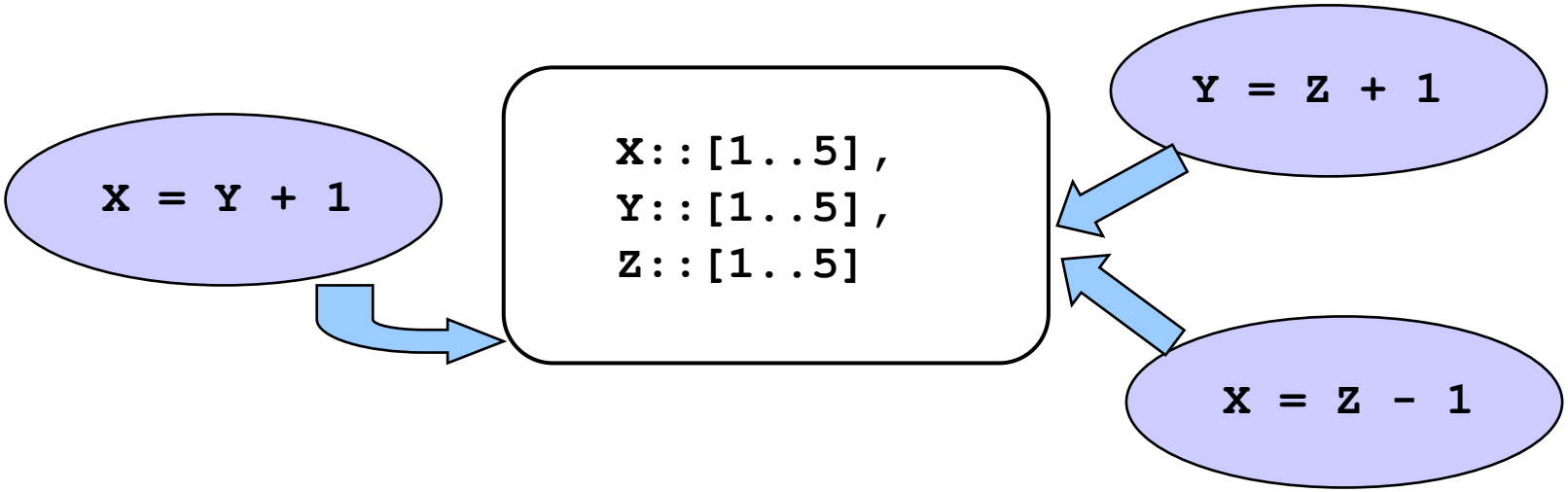
Each variable is involved in **many constraints**

➔ Any change in the domain of a ***single*** variable may propagate to other variables.

Constraints agents view: during their lifetime, the constraints alternate their status between **suspended** and **woken** states (triggered by events).

Example:





– First propagation of `X = Y + 1` yields

```
X::[2..5],
Y::[1..4],
Z::[1..5]
```

`X = Y + 1`
is then suspended

– *Second propagation of* `Y = Z + 1` *yields*

```
X::[2..5],
Y::[2..4],
Z::[1..3]
```

`Y = Z + 1` is then suspended

– *The domain of* `Y` *has changed,* `X = Y + 1` is then awakened

```
X::[3..5],
Y::[2..4],
Z::[1..3]
```

`X = Y + 1` is then suspended

– Third propagation of `X = Z - 1` yields

```
X::[]
Y::[2..4]
Z::[1..3]
```

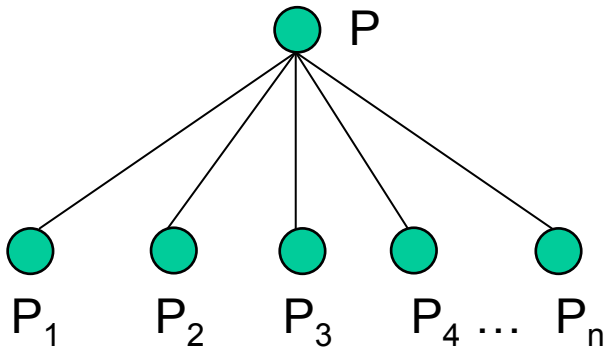
`FAILURE` detected

CP Preliminaries – search

• **Pruning** all the infeasible values from variable domains often has the same complexity as solving the original problem

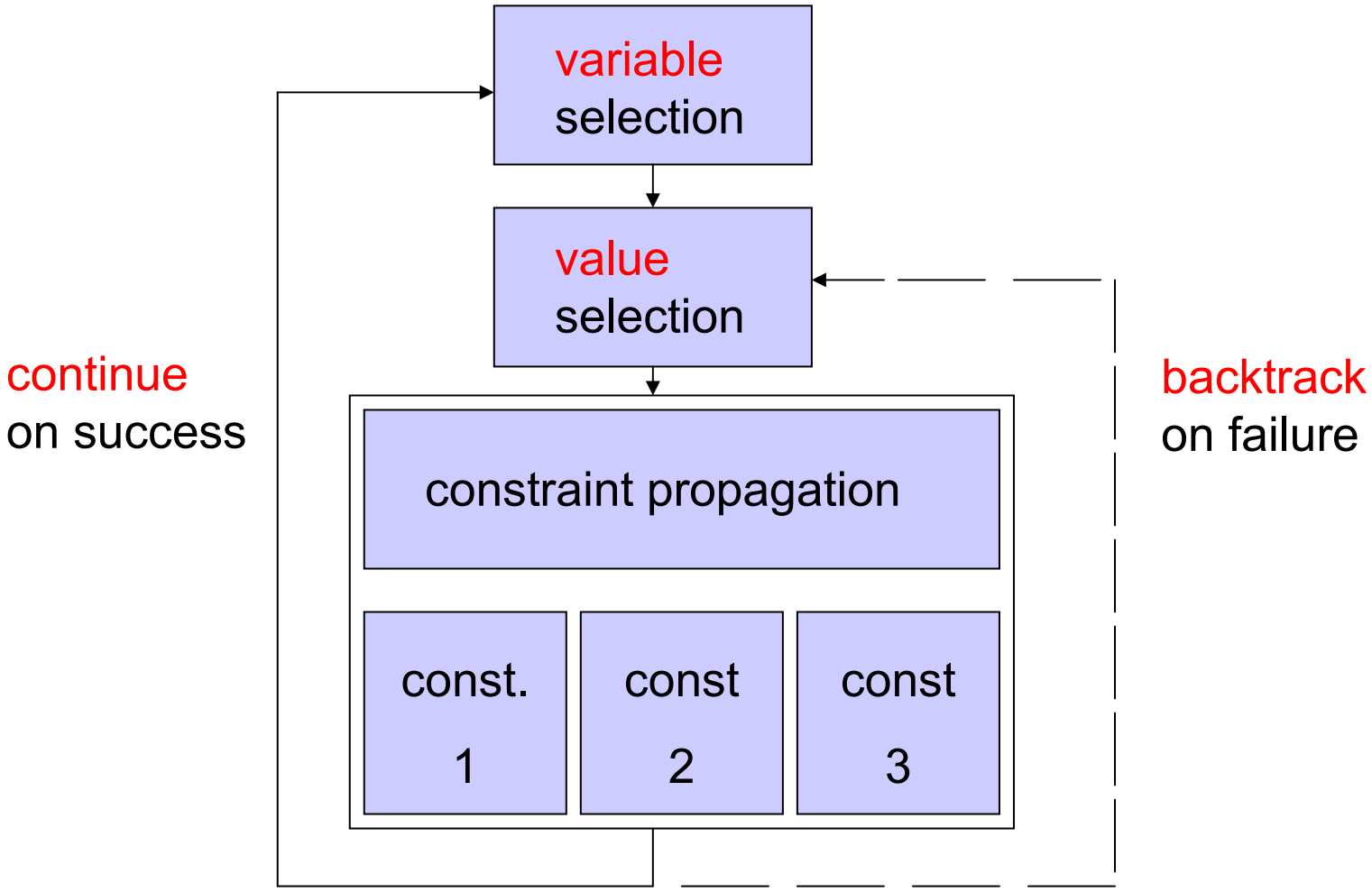
• Propagation algorithms are then **incomplete**, i.e., when propagation is stopped, still some values left in the variable domains can be inconsistent

→ a **SEARCH** step is executed.



- *Branching strategies define the way of partitioning the problem P into easier subproblems P_1, P_2, …, P_n.*
- *To each subproblem: apply again propagation.*
- *New branches can be pruned because of the new information derived from the branching*

The search scheme



CP Preliminaries: dealing with an objective function

- **What about the objective function?**

  As soon as a feasible solution of value Z^* (say) is found, a new ***bounding constraint*** is added so as to impose that further solutions must have a better value:

  $$Z < Z^*$$

  Hence, CP solves a sequence of **feasibility** problems, constrained to improve the objective function value.

  However: Z is in general the sum of the values assumed by several variables → the propagation of the bounding constraint is typically **very weak**

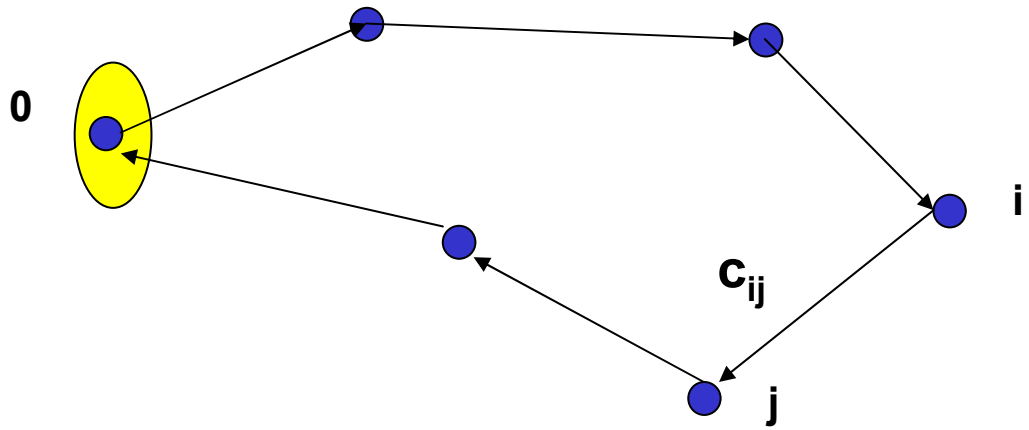
Example: (ATSP)
Asymmetric Traveling SalesmanProblem

Given

- a directed graph **G=(V,A)** with nodes **0,1,…, n-1**
- arc costs c_{ij}

Find

- a **Hamiltonian tour** (= closed circuit passing exactly once through all nodes) of minimum total cost



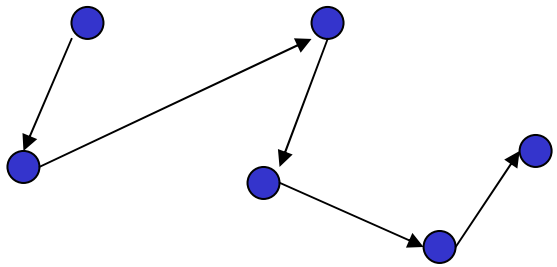
ATSP: a CP model

Given the directed graph **G=(V,A)**, associate to each node **i** a variable X_i whose domain contains the next possible nodes in a simple path.

CP standard modules include the following **path constraint:**

$X_0::D_0, X_1::D_1,..., X_k::D_k$

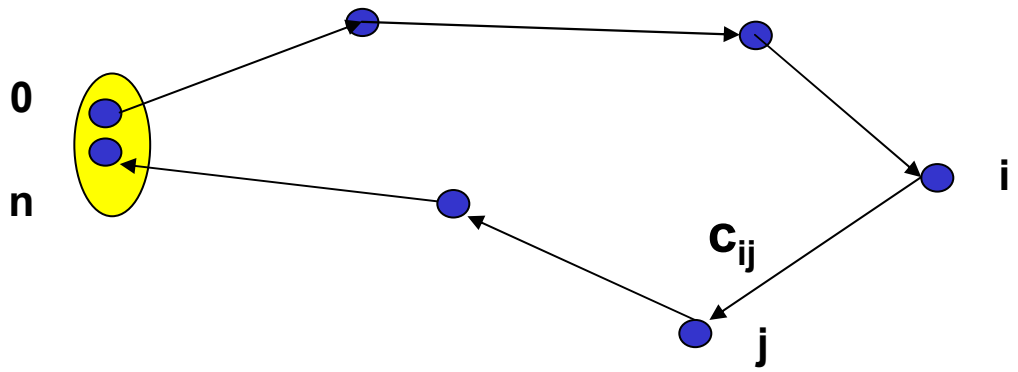
$path([X_0,X_1,\ldots,X_k])$



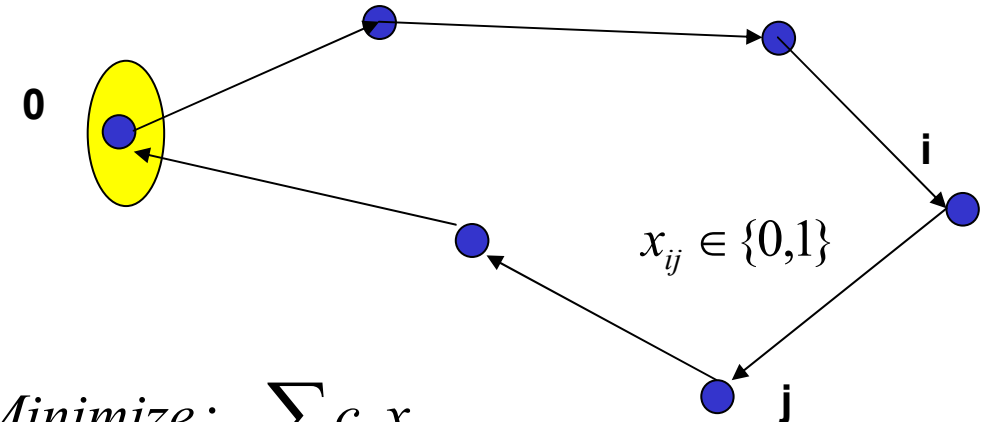
saying that the assignment of variables $X_0,X_1,\ldots,X_k$ has to define a simple path involving all nodes **0,…,k**.

The **Asymmetric Travelling Salesman Problem** (ATSP) can then be formulated through the CP path constraint as follows:

- one of the nodes, say node **0**, is duplicated generating node **n**
- the constraint **path($[X_0,X_1,\ldots,X_n]$)** is imposed



ATSP: an IP model



$x_{ij} = 1 \quad iff \ (i,j) \ in \ the solution$

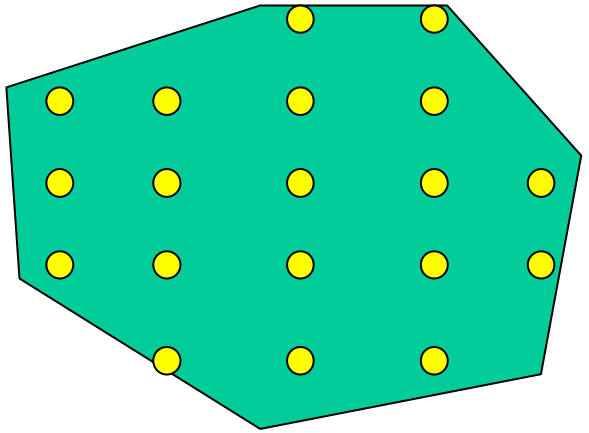
$$Minimize: \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$Subject\ To: \sum_{i\in V} x_{ij} = 1 \quad \forall j \in V \quad (\text{indegree} = 1)$$

$$\sum_{j\in V} x_{ij} = 1 \quad \forall i \in V \quad (\text{outdegree} = 1)$$

$$\sum_{i\in S}\sum_{j\notin S} x_{ij} >= 1 \quad \forall S \subset V : S \neq 0 \quad (\text{no subtours})$$

$$x_{ij} >= 0 \ \text{integer} \quad \forall (i,j) \in A$$



Comparing the CP and IP models

CP- Model:

$X_i :: [v_1, v_2, \ldots, v_n]$ i=0..n-1

path($[X_0, X_1, \ldots, X_n]$)

$C_i :: [c_{i1}, c_{i2}, \ldots, c_{in}]$ i=0..n-1

$C_n = 0$; $X_n = 0$;

$C_0 + \ldots + C_{n-1} = Z$

minimize(Z)

Mapping

IP- Model

$$\min Z = \sum_{i \in V} \sum_{j \in V} c_{ij}\, x_{ij}$$

$$\sum_{i \in V} x_{ij} = 1 \quad j \in V$$

$$\sum_{j \in V} x_{ij} = 1 \quad i \in V$$

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1 \quad S \subset V \quad S \neq \emptyset$$

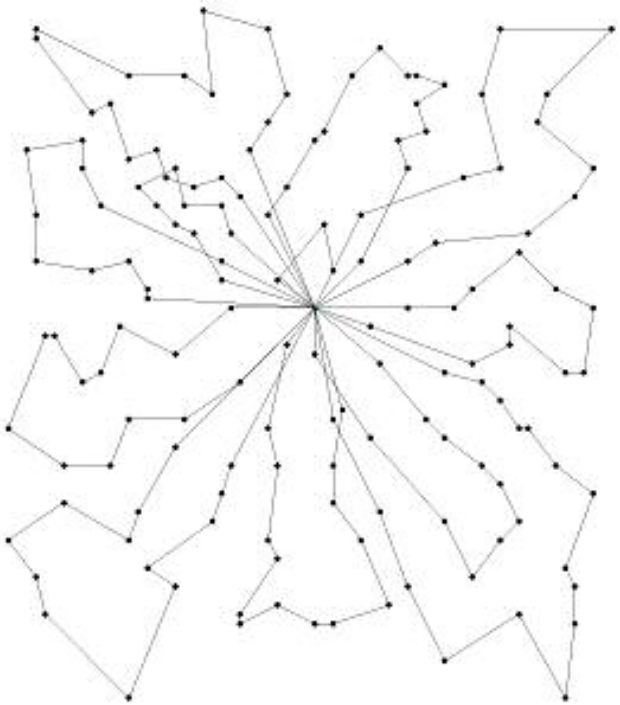
$x_{ij} \geq 0$ and integer

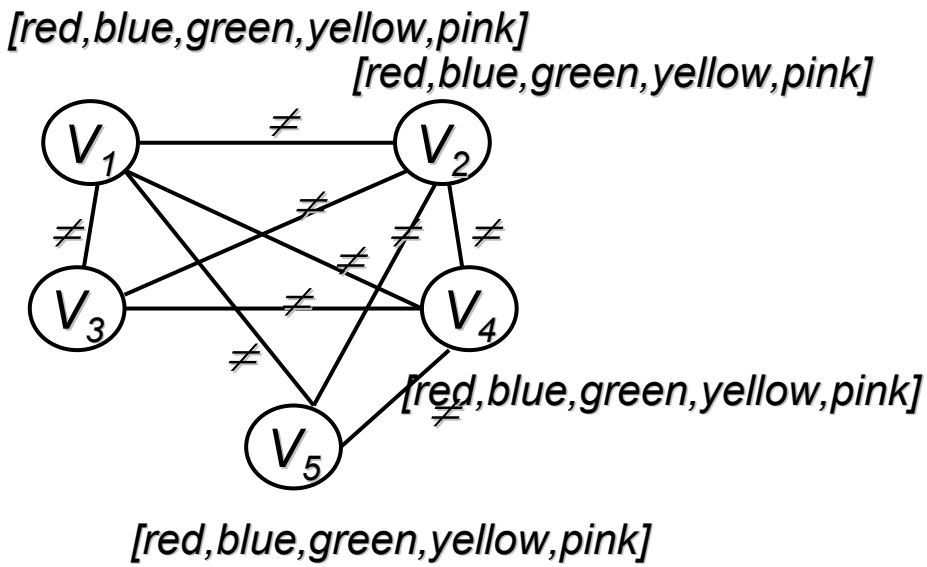
$X_i = v_j$ ⟷ $x_{ij}=1$

path ($[X_0, X_1, \ldots, X_n]$) ⟷ linear constr.s

$C_i = c_{ij}$ ⟷ c_{ij} iff $x_{ij}=1$

Pure IP vs Pure CP





Tight LP relaxation, easy feasibility: IP wins

Almost useless LP relaxation, hard feasibility: CP wins

CP+IP: preliminary results

- Although CP is not competitive for problems like **TSP** and **ATSP**, the addition of an IP optimization component within the CP framework allows for the solution of instances of significantly larger size (**Lodi et al., 2003**)
- Pure CP gets stuck on problems with 15-20 nodes.

TSP and ATSP instances						
Instance	***pure AP***			***AP + Lagrangean relaxation of cuts***		
	Opt	***Time***	***Fails***	***Opt***	***Time***	***Fails***
Gr17	2085	0.39	511	2085	0.49	30
Fri26	937	0.82	725	937	0.71	80
Bays29	2020	4.12	4185	2020	1.20	403
Dantzig42	699	>300	-	699	5.55	1081
RY48P	14854*	>300	-	14422	130.00	50K