Constraint Programming Techniques for Batch Scheduling

Gabriela P. Henning

INTEC (Universidad Nacional del Litoral – CONICET)
Santa Fe – Argentina

August 2005

E-mail: ghenning@intec.unl.edu.ar

Overview

- **Definition of Constraint Satisfaction (CS) and Constraint Programming (CP) problems -** Relevance - Applications that can be tackled
- **CP Phases:** Modeling & Solutions´ Generation
- **Generation of Solutions:** Search algorithms – Consistency Enforcing and Constraint Propagation
- **Domain-specific building-blocks for scheduling problems:** Variables, Constraints, Search Algorithms....
- **Scheduling of a Multistage, Multiproduct Plant**
- **Possible extensions to the basic model**
- **Advantages and disadvantages**

Constraint Satisfaction Problems (CSPs)

A CSP consists of:

- A set of **variables** $X = \{x_1, x_2, ... x_n\}$;
- For each variable x_i, a **finite set** D_i of possible values (its **domain**);
- A set of **constraints** (mathematical and/or symbolic) restricting the values that the variables can simultaneously take.
- A **feasible** solution to a CSP is an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied. In this case, the problem is **satisfiable.**
- If there is no assignment of values to variables from their respective domain from which all variables are satisfied, then the problem is **unsatisfiable**.

CSPs and Constraint Programming

- Formally, a constraint C_{ijk} involving the variables x_i, x_j, x_k, ..., , **specifies any subset of the possible combinations of values of x_i, x_j, x_k**, .. ; i.e $C_{ijk} \subseteq D_i \times D_j \times D_k \times \ldots$, that the constraint allows.

  Example:

  C_{xy}: 2 x = y

  $D_x = \{1, 2, \ldots.. 10\}$
  $D_y = \{1, 2, \ldots.. 15\}$

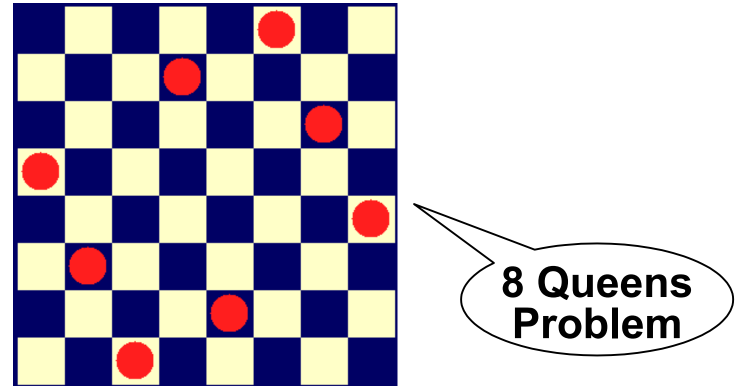
  $\{(1, 2), (2, 4), (3, 6), (4, 8), (5, 10), (6, 12), (7, 14)\}$

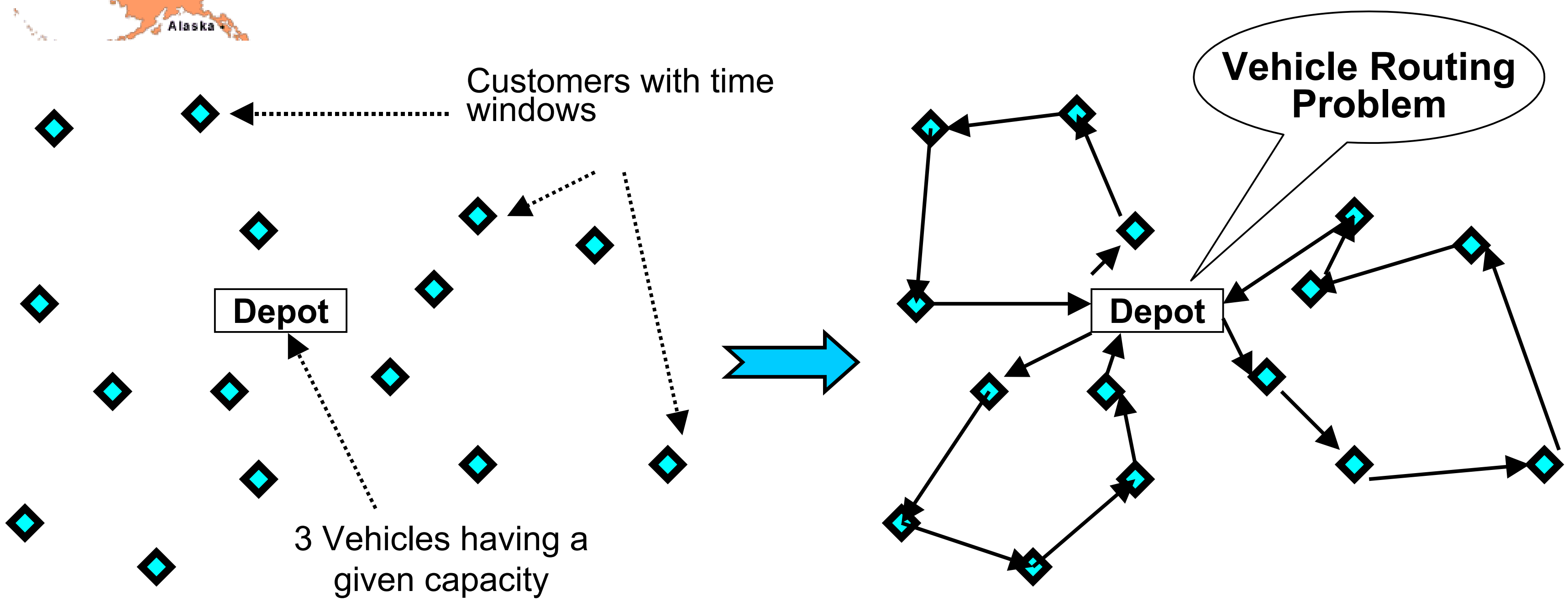
- The implementation of algorithms able to solve CSPs gives rise to **Constraint Programming (CP).**
- **CP is about the formulation of a problem as a CSP and about solving it by means of an appropriate solver** (general and/or domain specific one) .

CSPs Relevance

- Formal model to express problems. Many combinatorial problems can be represented as CSP:
  - **Academic problems:**
    - Graph/Map coloring, N-queens, Cryptarithmetic puzzles,..
  - **Real world problems:**
    - Scheduling of: multiproduct/multipurpose batch plants, flexible manufacturing systems (FMSs), activities in a project.
    - Resource allocation: Warehouse location, Timetabling Problems (Course timetabling, staff/crew rostering,
    - Vehicle Routing, Air/Train Traffic Control….
- Depending on the problem, different types of solutions are sought: Just a feasible solution, all feasible solutions, an optimal solution, a good quality solution, etc....

Variety of Application Areas - Different Problems





Overview

- **Definition of Constraint Satisfaction (CS) and Constraint Programming (CP) problems -** Relevance - Applications than can be tackled
- **CP Phases:** Modeling & Solutions´ Generation
- **Generation of Solutions:** Search algorithms – Consistency Enforcing and Constraint Propagation
- **Domain-specific building-blocks for scheduling problems:** Variables, Constraints, Search Algorithms....
- **Scheduling of a Multistage, Multiproduct Plant**
- **Possible extensions to the basic model**
- **Advantages and disadvantages**

Constraint Programming

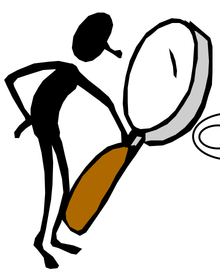
It can be seen as a **two phases approach**:

- **Phase I:** Generation of a **problem representation** ≡ **Modeling**
  It involves:
  - Selection of variables
  - Choice of variable domains
  - Definition of constraints
- **Phase II:** Generation of a/several **problem solution/s**
  - General methods
  - Domain-specific methods
  - Hybrid approaches
- Presence of **built-ins constructs and methods**: Pre-defined variables and constraints, as well as constraint solvers, constraint propagation algorithms and search methods

Phase I: Modeling

- **Different types of variables:**
  - Integers
  - Reals
  - Boolean
  - Symbolic (variables range over non-numeric domains)
  - A combination of the above

- **Different types of constraints defining a constraint nework:**
  - Symbolic constraint satisfaction problems (e.g., Puzzles, qualitative temporal/spacial reasoning)
  - Boolean constraint satisfaction problems (Circuits)
  - Constraint satisfaction problems on reals
  - Logic Constraints
  - ....



Non-trivial task!

Look for an appropriate representation

Each problem can be formalised as a CSP in a number of different ways.

In general, it is difficult to find which representation is better!!

Phase II: Generation of Solutions

- CP implements systematic search procedures that, by fixing the order in which the variables should be chosen, and a way to select a value from a variable domain, supply a proper assignment of values to the problem variables.
- **Search Algorithm = Search Tree + Traversal Algorithm**
- Strategies
  - Backtracking
  - Backjumping
  - Forward Checking
  - MAC – Maintaining Arc Consistency
  - Branch & Bound

  Most of these techniques are included in commercial packages
- Domain Specific Strategies

Overview

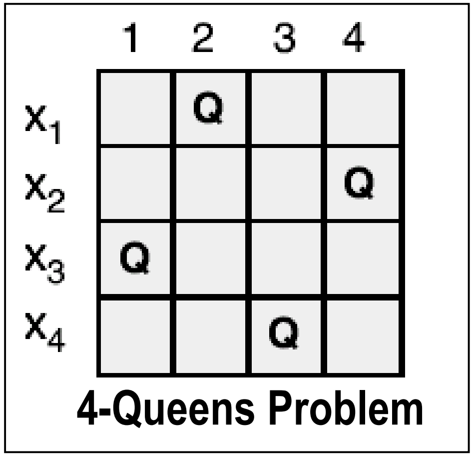
- **Definition of Constraint Satisfaction (CS) and Constraint Programming (CP) problems -** Relevance - Applications than can be tackled
- **CP Phases:** Modeling & Solutions´ Generation
- **Generation of Solutions:** Search algorithms – Consistency Enforcing and Constraint Propagation
- **Domain-specific building-blocks for scheduling problems:** Variables, Constraints, Search Algorithms….
- **Scheduling of a Multistage, Multiproduct Plant**
- **Possible extensions to the basic model**
- **Advantages and disadvantages**

Tree Search

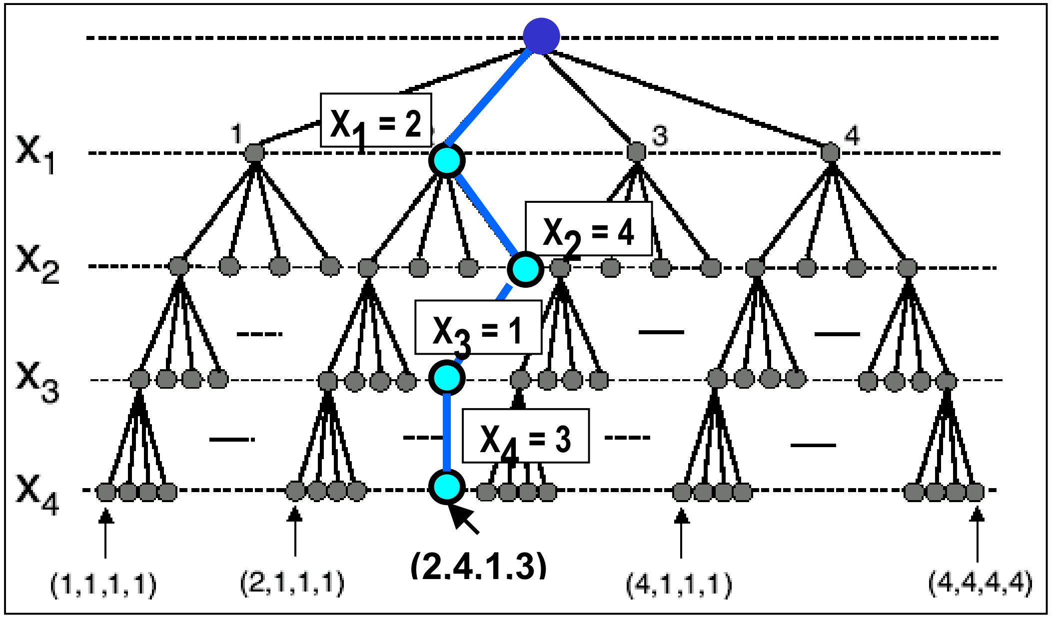
- **State space**: explored as a tree
  - Root: empty
  - One variable per level
  - Successors of a node:

    one successor per value of the variable

    meaning: variable ← value



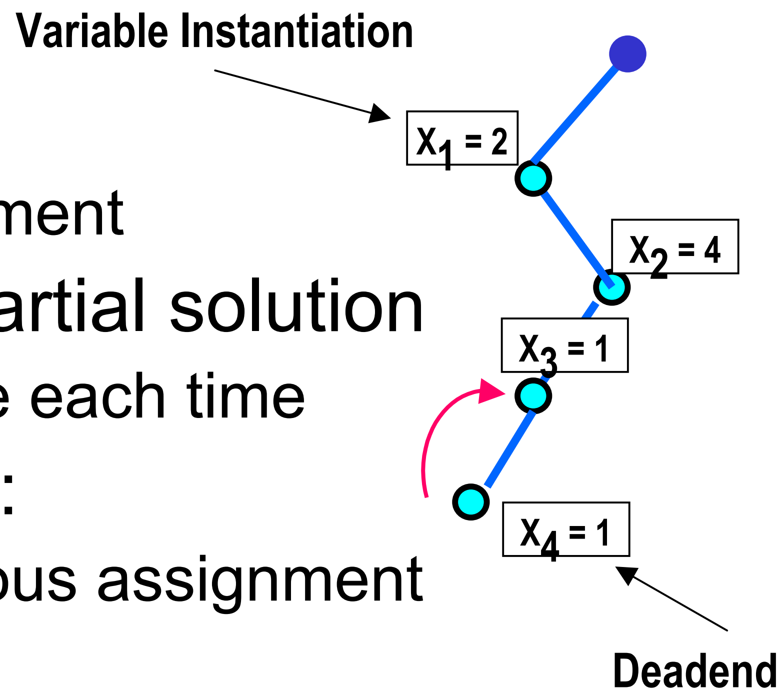
- **Tree:**
  - each branch defines an assignment
  - depth n ≡ number of variables
  - branching factor d ≡ domain size



Backtrack Search

Strategy:

- Build a partial solution:
  - A partial consistent assignment
- Extend consistently the partial solution
  - One new assigned variable each time
- If no consistent extension:
  - Backtrack: change a previous assignment



Variables:

- Past ∈ partial solution (assigned variables)
- Future ∉ partial solution (unassigned variables)

Backtrack Search

Depth-first tree traversal (DFS)

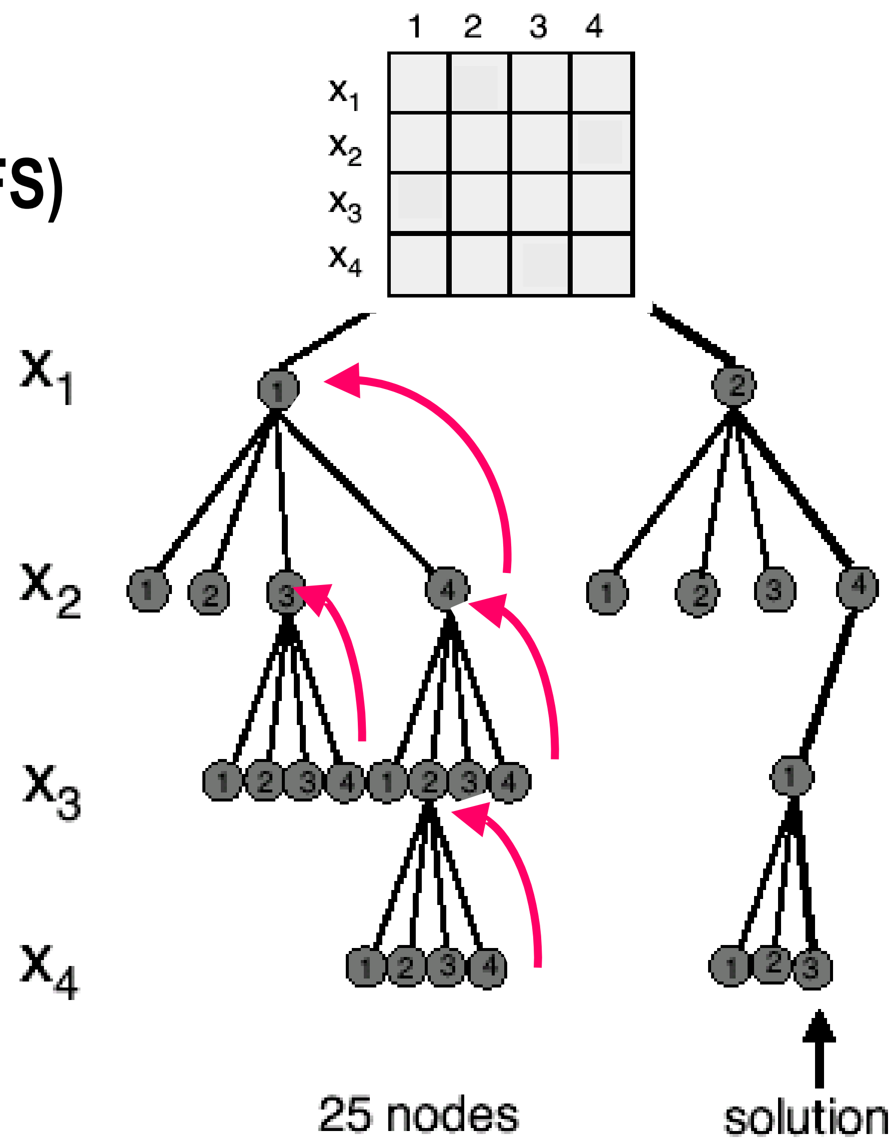
At each node:

- check every completely assigned constraint
- if consistent, continue DFS
- otherwise, prune current branch

  continue DFS

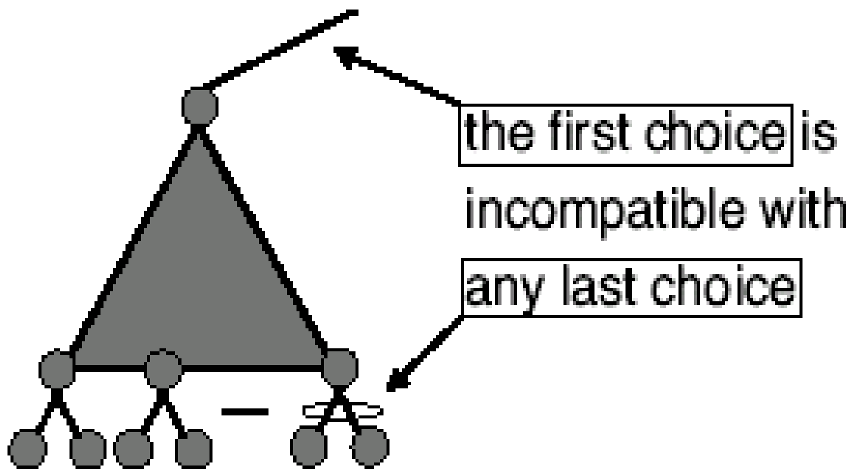
Complexity: O(d^n)

Not efficient!!



Problems with Backtracking

- It only checks the constraints by considering the current variable (the one to be instantiated) and the past ones.
- **Trashing: The same failure can be rediscovered an exponential number of times**



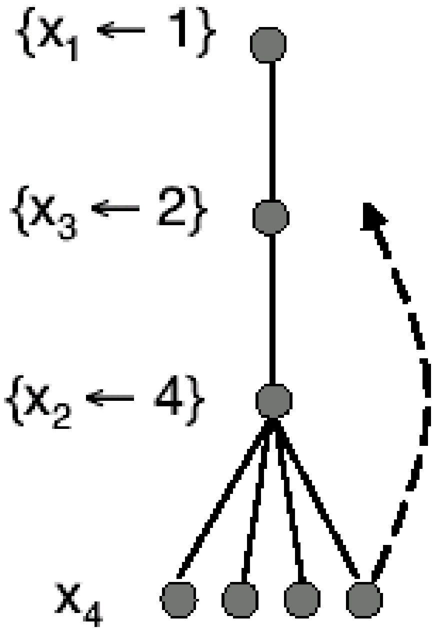
Solutions:

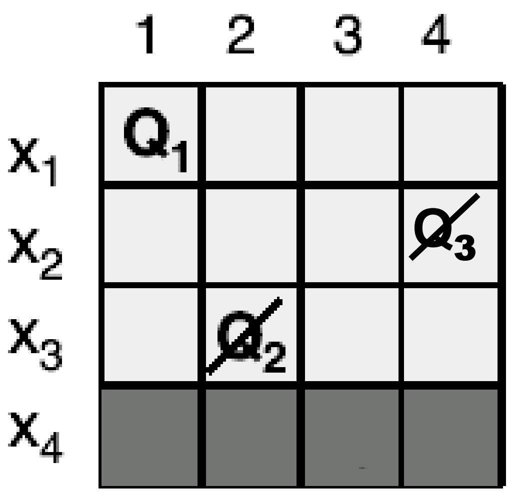
- Check not completely assigned constraints = Consider future variables → **looakahead**
- Non-chronological backtracking → **backjumping**

Backjumping

Non-chronological backtracking:

- Jumps to the last decision responsible for the dead-end
- Intermediate decisions are removed





Lookahead Algorithms

- **Forward checking**:

  When a value is assigned to the current variable, any value in the domain of a future variable which conflicts with this assignment is (temporarily) removed from the domain.

  Advantage: If the domain of a future variable becomes empty, it is known that the current partial solution is inconsistent, then:

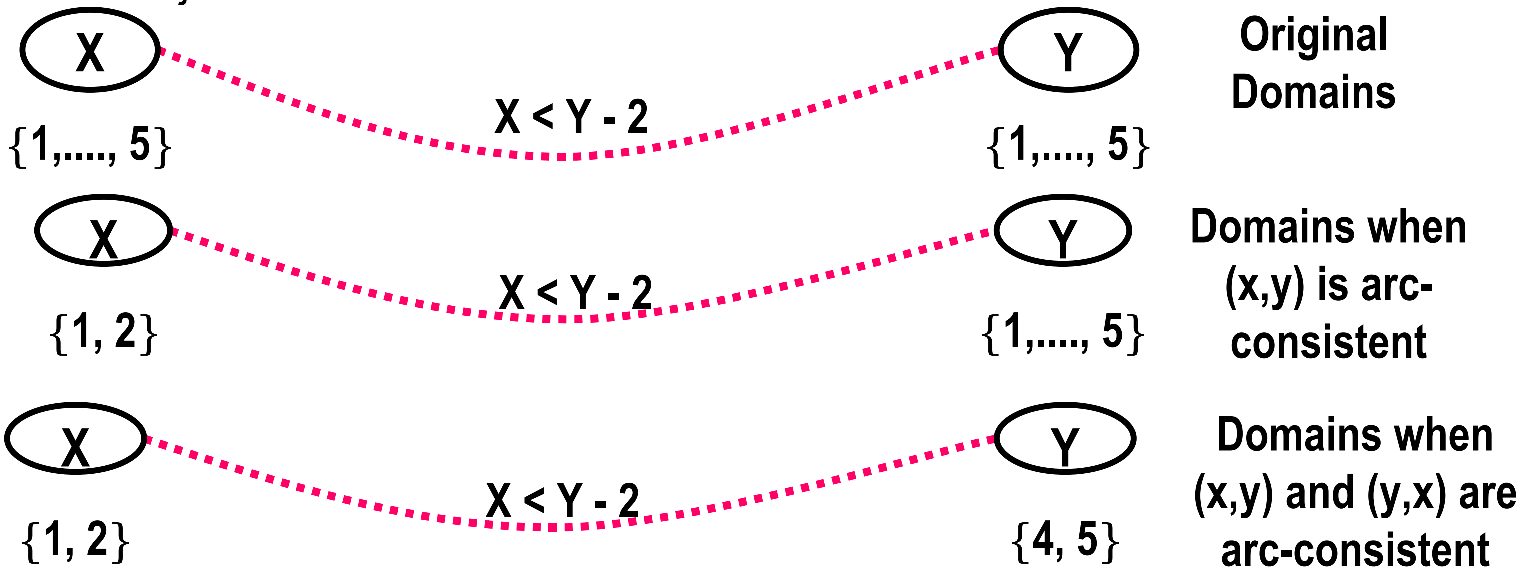
    - another value the current variable is tried, or
    - the algorithm backtracks to the previous variable.

- **Maintaing Arc Consistency (MAC)**: Whenever a new subproblem consisting of the future variables is created by a variable instantiation, the subproblem (constraint network) is made **arc consistent**.

Consistency Enforcing

Arc-Consistency:

- **Binary Constraints**: Given a constraint C_{ij} between the variables x_i and x_j ; then the directed arc $(x_i; x_j)$ is **arc consistent** iff for every value $a \in D_i$, there is a value $b \in D_j$ such that the assignments $x_i = a$ and $x_j = b$ satisfy the constraint C_{ij}.



Consistency Enforcing

- The binary arc-consistency concept can be generalized to consider arbitrary constraints $\rightarrow$ **Hyper-arc Consistency**.
- **Path-Consistency:** A binary constraint C_{ij} is **path-consistent** relative to x_k, iff for every pair $(a_i; a_j) \in C_{ij}$, where a_i and a_j are from their respective domains, there is a value $a_k \in D_k$, such that $(a_i; a_k) \in C_{ik}$ and $(a_k; a_j) \in C_{kj}$.
- A **network is path-consistent** iff for every C_{ij} (including universal binary relations) and for every $k \neq i, j$, C_{ij} is path consistent relative to x_k .
- **Consistency enforcing**: Lookahead capabilities, reduccion in the domains of the variables, adds constraints on pairs of variables $\rightarrow$ **Constraint Propagation!!!!**

Example: Constraint Propagation and Domain Reduction

Variables: x, y

Domains:

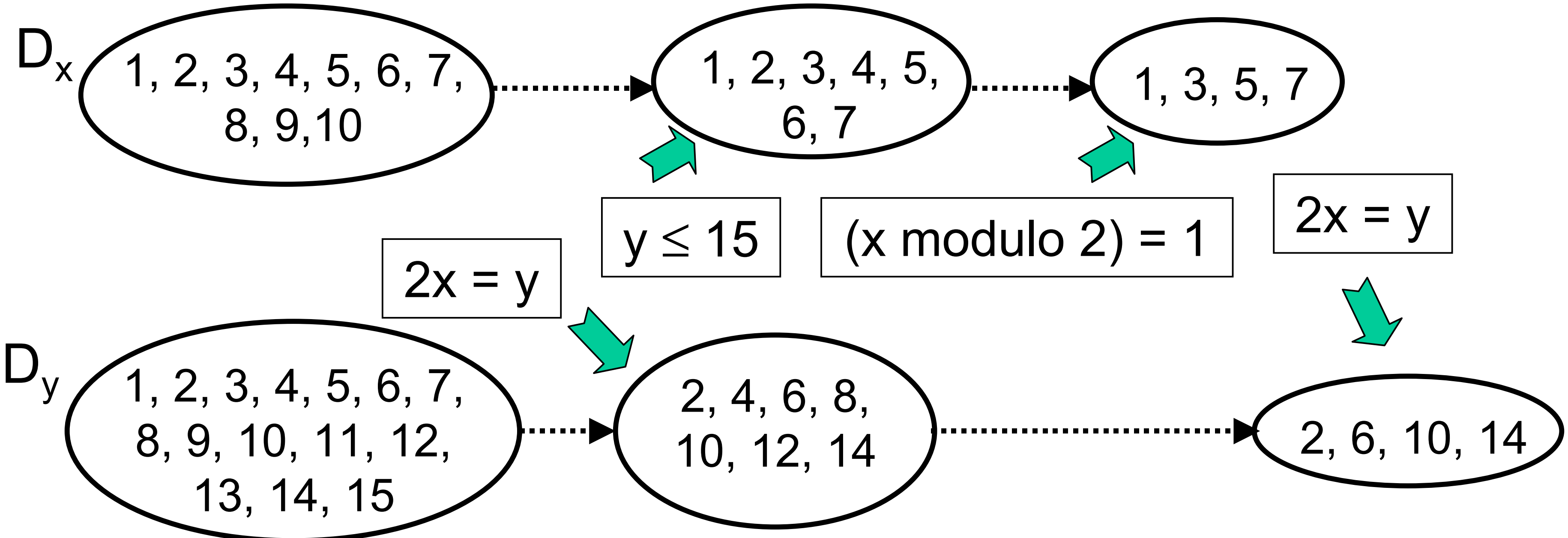
$D_x = \{1, 2, ... 10\}$
$D_y = \{1, 2, ... 15\}$

Constraints:

$2x = y$

$y \leq 15$

(x modulo 2) = 1



Overview

- **Definition of Constraint Satisfaction (CS) and Constraint Programming (CP) problems -** Relevance - Applications than can be tackled
- **CP Phases:** Modeling & Solutions´ Generation
- **Generation of Solutions:** Search algorithms – Consistency Enforcing and Constraint Propagation
- **Domain-specific building-blocks for scheduling problems:** Variables, Constraints, Search Algorithms....
- **Scheduling of a Multistage, Multiproduct Plant**
- **Possible extensions to the basic model**
- **Advantages and disadvantages**

Domain-specific building-blocks

- Problem Variables:
  - Activities
  - Resources
- Specific Constraints:
  - Resource constraints
  - Temporal constraints
  - Global constraints
- Search Algorithms:
  - Edge Finding
  - Others... (Laborie, “Algorithms for...”, Artificial Intelligence, 143, 151-188, 2003)
- Special Constructs

Problem Variables: Activities

- An activity `ActivityA` corresponds to a time interval [`ActivityA.start, ActivityA.end`]
- `ActivityA.start` and `ActivityA.end` are decision variables denoting the start and end time of the activity.
- The duration of an activity may be known in advance or may be a decision variable.

  `ActivityA.duration = ActivityA.end - ActivityA.end`

- Different types of activities can be modelled
  - Processing/Manufacturing: `Task[j,t]; j Job ; t stage`

    [`Task[j,t].start, Task[j,t].end`] ;
  - Start-up, Clean-up, Changeover
  - Pumping & Product movement, Raw Material Delivery
- Activities use or share resources → Compete for resources

Problem Variables: Resources

- Different types of resources, demanded by various kinds of activities, need to be represented:

Non-Renewable Resources	Reservoir Resources
Renewable Resources	Discrete, Cumulative or Sharable Resources Unary Resources

- A **resource constraint** defines how a given activity A will require and affect the availability of a given resource R. It consists of a tuple (A,R,q,TE), where q is the quantity of resource R consumed (if $q < 0$) or produced (if $q > 0$) by activity A and TE is a time extend specifying the time interval where the availability of resource R is affected

Different Types of Resources

- **Reservoir resource:** It is a multi-capacity resource that can be consumed and or produced by activities (e.g. A tank containing an intermediate product, a fuel tank, etc.)
- **Discrete, cumulative or sharable resource:** represents a resource that is used over a certain time interval under the following policy: a certain quantity of the resource is consumed at the start time of the activity and the same amount is released at its end time or earlier. (e.g. Pool of workers, steam, cooling water, etc.).
- **Unary resource:** It is a discrete resource with unit capacity (e.g. Machine/processing units that can perform just one operation at a given time). It imposes that **all the activities demanding the same unary resource are totally ordered**.

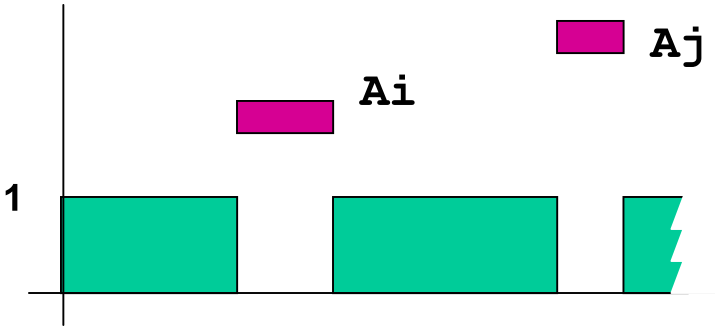
Domain-specific building-blocks

- Problem Variables:
  - Activities
  - Resources
- Specific Constraints:
  - Resource constraints
  - Temporal constraints
  - Global constraints
- Search Algorithms:
  - Edge Finding
  - Others... (Laborie, "Algorithms for...", Artificial Intelligence, 143, 151-188, 2003)
- Special Constructs

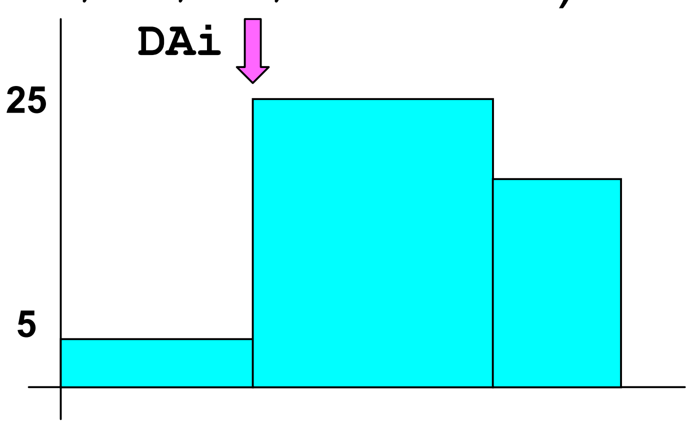
Resource Constraints

General Form: (A,R,q,TE)

(Ai,Ej,-1,FromStartToEnd): States that activity Ai will require one unit of resource Ej between its start and end time. Ai can be a processing activity demanding an equipment unit Ej which is declared as a unary resource.



(DAi,Rk,20,AfterEnd): States that activity DAi will produce 20 units of resource Rk at its end time. DAi can be a raw material delivery activity aiming at replenishing a resource reservoir Rk.



Temporal Constraints

- **Precedence Relationships:**
  Example: Multi-stage, multi-product plant
  - `j` job to be scheduled
  - `t` stage

`task[j,t].end <= task[j,t'].start; ∀ t ∈ T, ∀ j ∈ J,`

`Ord(t')= Ord(t)+1`

Equivalent to: `task[j,t]` **`precedes`** `task[j,t']`

- **Disjunctive constraints on unary resources:**

Let `task[j1,t]` and `task[j2,t]` be two activities that require the same unary resource (e.g. the same processing unit). Then:

`task[j1,t].end <= task[j2,t].start ∨`

`task[j2,t].end <= task[j1,t].start;` which is equivalent to

`task[j1,t]` **`precedes`** `task[j2,t]` or vice versa.

Domain-specific building-blocks

- Problem Variables:
  - Activities
  - Resources
- Specific Constraints:
  - Resource constraints
  - Temporal constraints
  - Global constraints
- Search Algorithms:
  - Edge Finding
  - Others... (Laborie, "Algorithms for...", Artificial Intelligence, 143, 151-188, 2003)
- Special Constructs

Domain Specific Search Strategies

- Algorithms **based on an analysis of activity interactions**. They consider subsets Ω of activities competing for the same resource and perform propagation based on the position of activities in Ω
- Disjunctive Constraints
- Edge-finding
  - An integral part of commercial constraint-based schedulers
  - A fundamental **pruning technique** for scheduling problems associated to renewable resources.
  - Informally speaking, an edge finding algorithm considers one resource at a time, and identifies pairs (Ω, **`A`**) such that task **`A`** cannot precede (follow) any task from Ω in all feasible schedules, and updates the earliest starting time (latest completion time) of task **`A`** accordingly.

Edge-finding

- Let Ω be a subset of activities on a unary resource, and $\mathbf{A} \notin \Omega$ another activity on the same unary resource.
- Let $start_{min}(X)$, $end_{max}(X)$ and $dur_{min}(X)$ respectively denote the minimal start time, maximal end time and minimal duration over all the activities in the set X. Let $\mathbf{A}$.lct be the latest completion time of activity A.
- Most edge-finding techniques can be captured by the following rule:

$$end_{max}(\Omega \cup \mathbf{A}) - start_{min}(\Omega) < dur_{min}(\Omega \cup \mathbf{A}) \Rightarrow$$

$$\mathbf{A}.lct \leq \min_{\Omega' \subseteq \Omega}(end_{max}(\Omega') - dur_{min}(\Omega'))$$

New Bound!

Example of Edge-finding Propagation

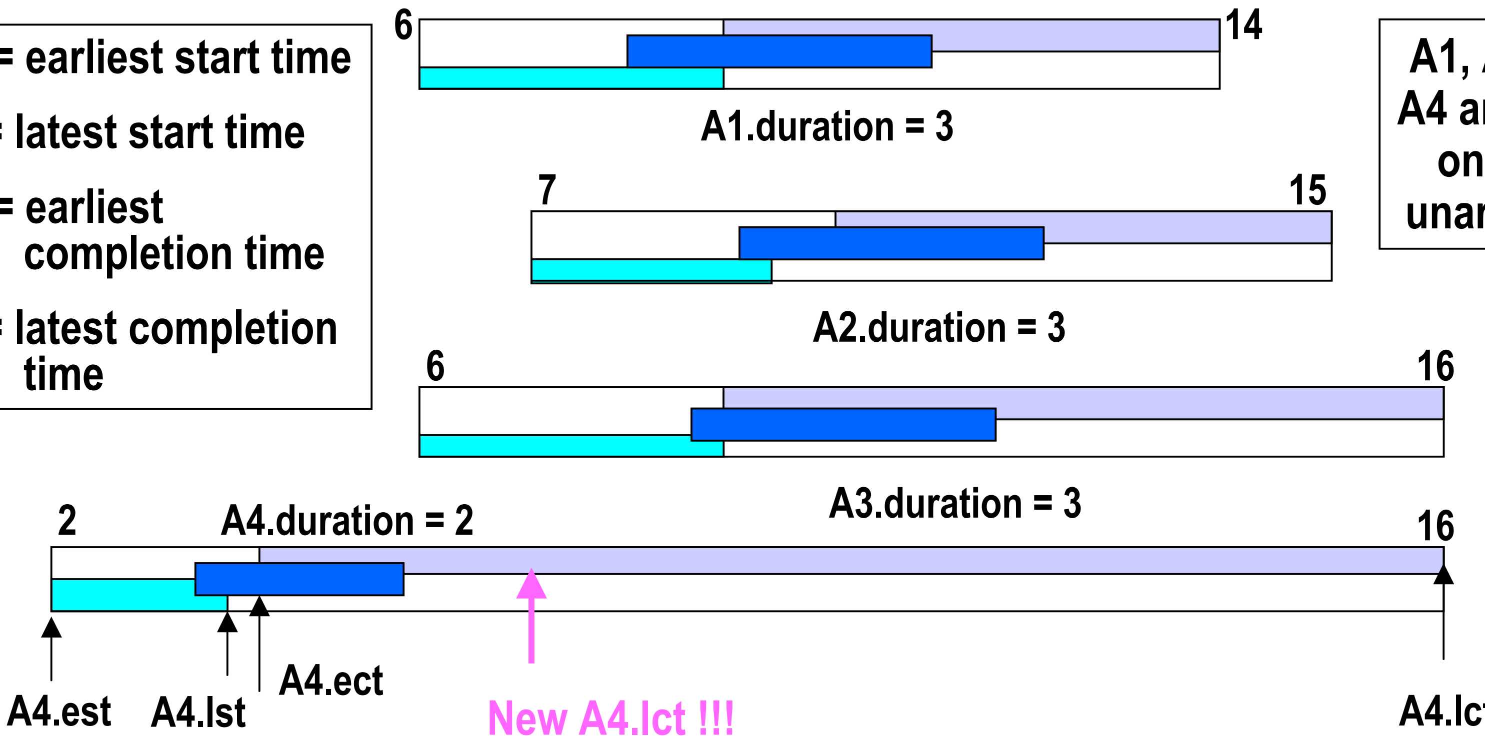
est = earliest start time

lst = latest start time

ect = earliest completion time

lct = latest completion time

A1, A2, A3 and A4 are activities on a certain unary resource



If A = A4 and Ω = {A1, A2 A3}, **the conditions of the propagation rule are satisfied** as $end_{max}(\Omega \cup A) = 16$, $start_{min}(\Omega) = 6$ and $dur_{min}(\Omega \cup A) = 11$ $(16 - 6 < 11)$.

By taking Ω´ = {A1, A2 A3}, **a new upper bound on A4.lct can be computed.**

$A4.lct \leq 16 - 9$; **$A4.lct \leq 7$** !!!!

Domain-specific building-blocks

- Problem Variables:
  - Activities
  - Resources
- Specific Constraints:
  - Resource constraints
  - Temporal constraints
  - Global constraints
- Search Algorithms:
  - Edge Finding
  - Others... (Laborie, "Algorithms for...", Artificial Intelligence, 143, 151-188, 2003)
- Special Constructs

Special Constructs

activityHasSelectedResource(activityA,s,u) is a special construct provided by the OPL language (ILOG, 2004).

- This construct acts like a predicate that evaluates to true (its value is equal to one), when activityA has selected resource u among the set of alternative resources s.
- **activityHasSelectedResource**(activityA,s,u) is in itself a constraint that can be negated.
- Can be employed in higher order constraints as well as in domain-specific search procedures.

Activities can consume and produce (non-renewable resources) or can require and provide (renewable resources) some units of reservoir and discrete resources. There are special constructs to express these ideas:

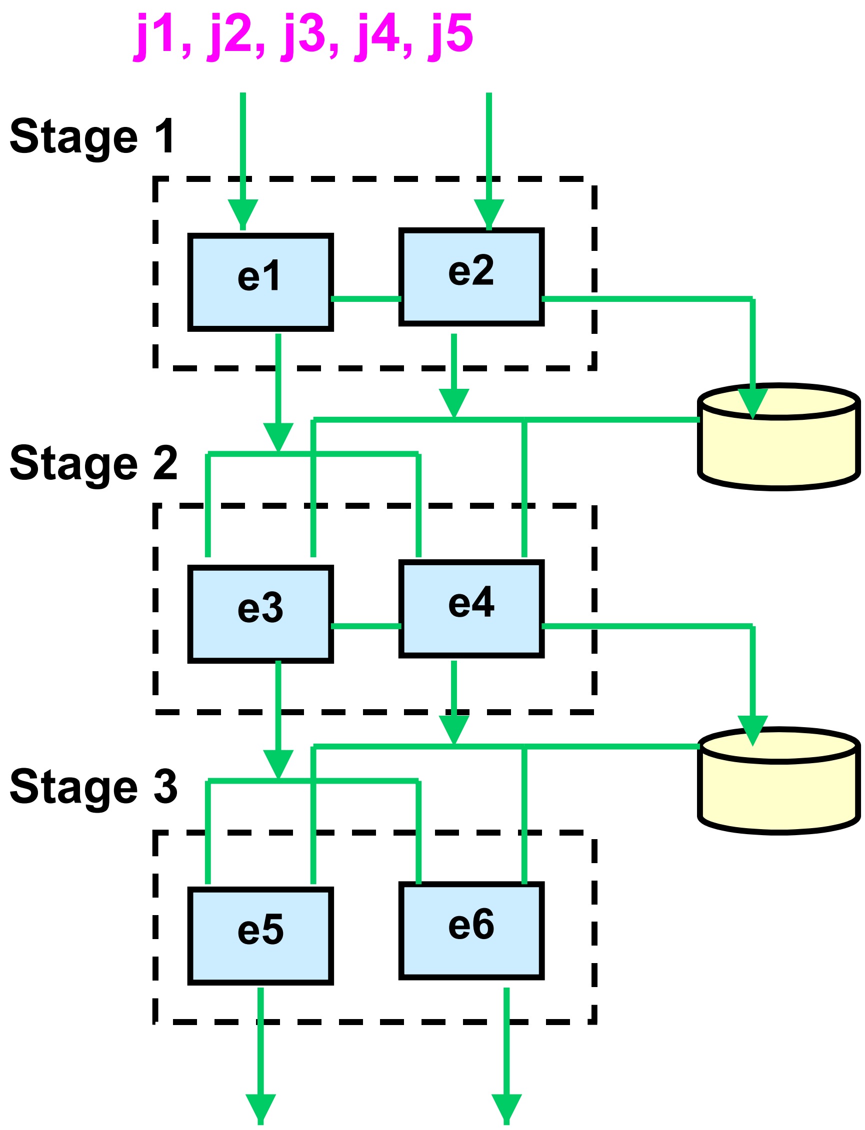
- Example: activityA requires (10) UtilityU.

  Specifies that activityA requires 10 units of UtilityU during its execution.

Overview

- **Definition of Constraint Satisfaction (CS) and Constraint Programming (CP) problems -** Relevance - Applications than can be tackled
- **CP Phases:** Modeling & Solutions´ Generation
- **Generation of Solutions:** Search algorithms – Consistency Enforcing and Constraint Propagation
- **Domain-specific building-blocks for scheduling problems:** Variables, Constraints, Search Algorithms...
- **Scheduling of a Multistage, Multiproduct Plant**
- **Possible extensions to the basic model**
- **Advantages and disadvantages**

Scheduling of a Multistage, Multiproduct Plant



CP Model: Scheduling of a Multistage, Multiproduct Plant

```
enum Jobs...;
enum Equipment...;
enum Stages...;
{Equipment} belongsto[Stages] = ...;
int+ ProcTime[Jobs, Equipment]=...;
scheduleHorizon = 500;
```

Specifies the set of Jobs, units and stages

Sets the equipment units that belong to each stage

Declares the array Processing Time

```
// Variables´ declaration
//Tasks´ Declaration – Resources declaration
Activity Task[j in Jobs, St in Stages];
Activity makespan(0);
UnaryResource tool[Equipment];
AlternativeResources s(tool);
```

Declares a `Task` decision variable for each job j and stage t

Declares a `Makespan` activity having a null duration

Declares equipment as unary resources

```
//Objective Function Definition
minimize
   makespan.end
//Basic Constraints
subject to {
```

CP model stated in the OPL language

CP Model: Scheduling of a Multistage, Multiproduct Plant

```
forall(j in Jobs)
    Task[j,last(Stages)] precedes makespan;
```

Makespan is the last activity

```
forall(st in Stages)
    forall(j in Jobs)
        Task[j,st] requires s;
```

Each task requires one element of the set of alternative resources

```
forall( j in Jobs)
 forall(St in Stages)
   forall(unit in Equipment)
      (unit not in belongsto[St]) =>
       not activityHasSelectedResource(Task[j,St],s,tool[unit]);
```

If the unit does not belong to the stage, the special construct activityHasSelectedResource is negated

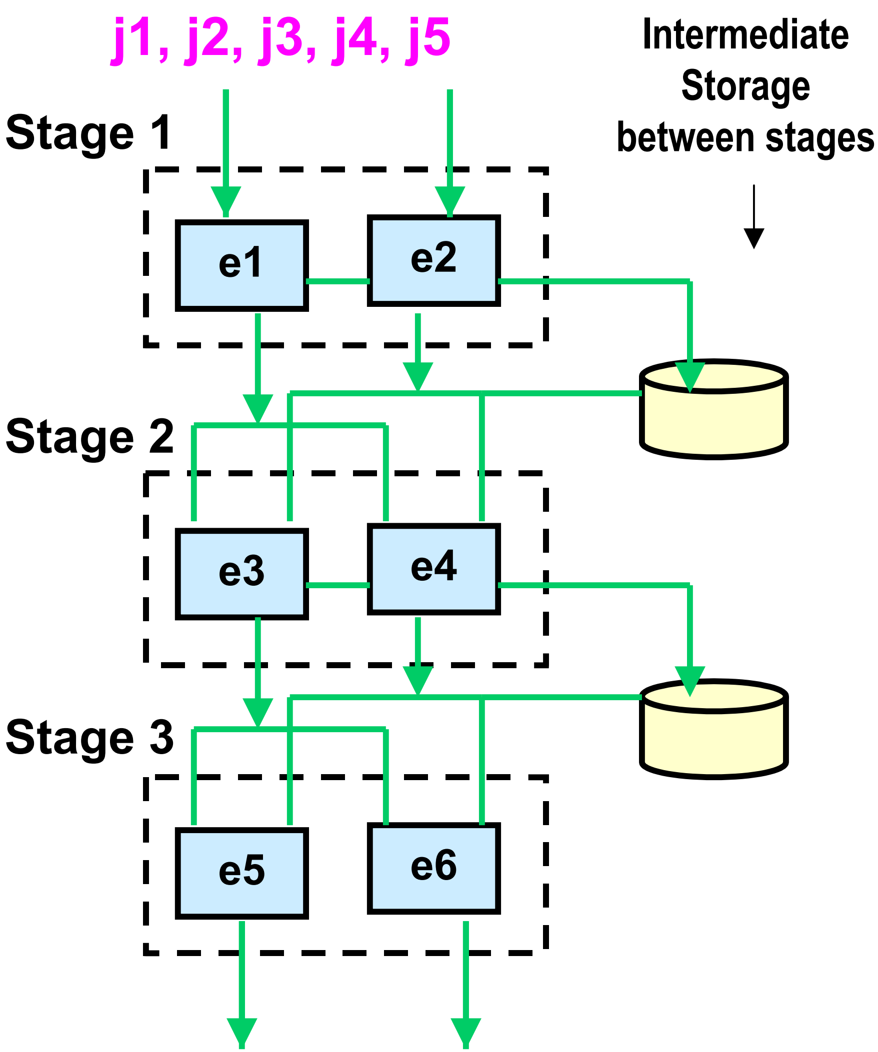
```
forall(St in Stages)
     forall(unit in belongsto[St])
       forall( j in Jobs)
        activityHasSelectedResource(Task[j,St],s,tool[unit]) =>
        (Task[j,St].duration = ProcTime[j,unit]);
```

If the special construct activityHasSelected Resource evaluates to true, the task is assigned the proper duration

```
forall(St in Stages : St<last(Stages))
      forall(j in Jobs)
        Task[j,St] precedes Task[j,next(St)];
};
```

Precedence order of the activities within each job

Example Data



Jobs = {j1, j2, j3, j4, j5};

Equipment = {e1, e2, e3, e4, e5, e6};

Stages = {st1, st2, st3};

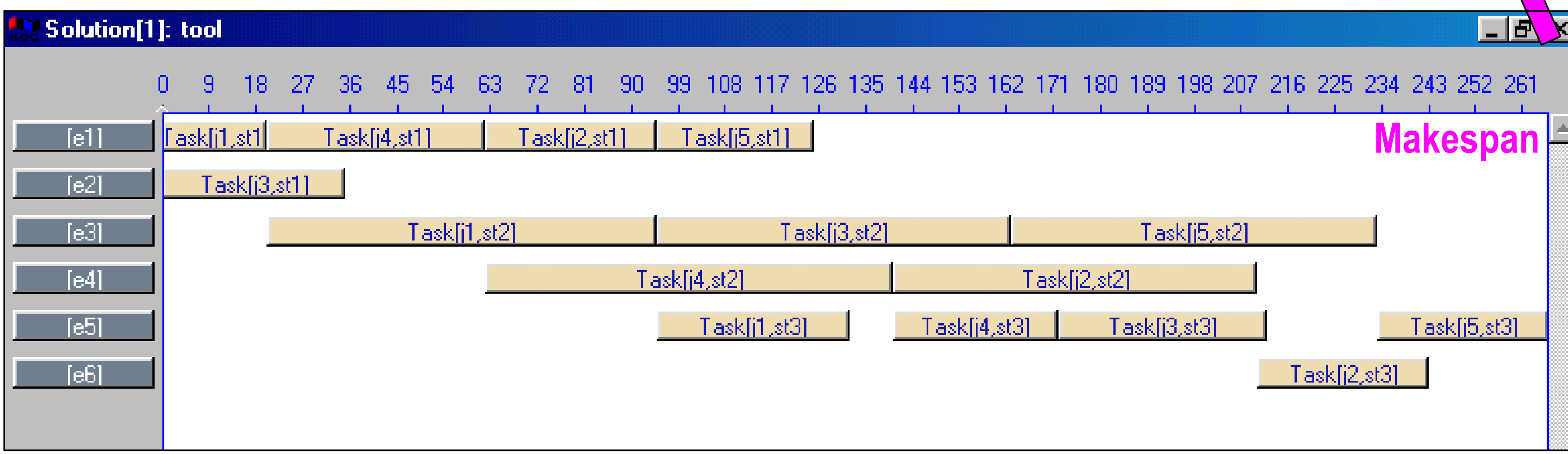
belongsto = #[
    st1:{e1,e2},
    st2:{e3,e4},
    st3:{e5,e6}
    ]#;

ProcTime=[
    [20, 28, 75, 80, 37, 36],
    [33, 31, 71, 70, 35, 33],
    [41, 35, 68, 75, 40, 34],
    [42, 30, 73, 78, 32, 30],
    [30, 33, 70, 74, 33, 35]
  ];

Results: UIS Case (Unlimited intermediate storage)

Solution[1]: Task

	st1	st2	st3
j1	[0 -- 20 --> 20]	[20 -- 75 --> 95]	[95 -- 37 --> 132]
j2	[62 -- 33 --> 95]	[140 -- 70 --> 210]	[210 -- 33 --> 243]
j3	[0 -- 35 --> 35]	[95 -- 68 --> 163]	[172 -- 40 --> 212]
j4	[20 -- 42 --> 62]	[62 -- 78 --> 140]	[140 -- 32 --> 172]
j5	[95 -- 30 --> 125]	[163 -- 70 --> 233]	[233 -- 33 --> 266]

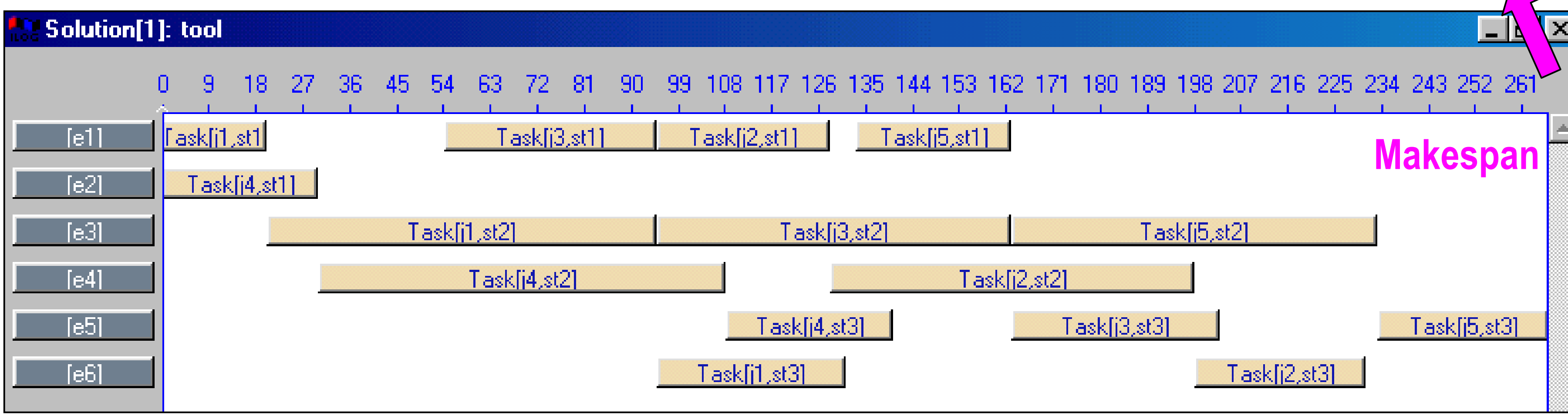


Results for the ZW Case (Zero-Wait)

```
Forall(St in Stages : St<last(Stages))
        forall(j in Jobs)
            Task[j,St].end = Task[j,next(St)].start;
```

Solution[1]: Task

	st1	st2	st3
j1	[0 -- 20 --> 20]	[20 -- 75 --> 95]	[95 -- 36 --> 131]
j2	[95 -- 33 --> 128]	[128 -- 70 --> 198]	[198 -- 33 --> 231]
j3	[54 -- 41 --> 95]	[95 -- 68 --> 163]	[163 -- 40 --> 203]
j4	[0 -- 30 --> 30]	[30 -- 78 --> 108]	[108 -- 32 --> 140]
j5	[133 -- 30 --> 163]	[163 -- 70 --> 233]	[233 -- 33 --> 266]



Overview

- **Definition of Constraint Satisfaction (CS) and Constraint Programming (CP) problems -** Relevance - Applications than can be tackled
- **CP Phases:** Modeling & Solutions´ Generation
- **Generation of Solutions:** Search algorithms – Consistency Enforcing and Constraint Propagation
- **Domain-specific building-blocks for scheduling problems:** Variables, Constraints, Search Algorithms...
- **Scheduling of a Multistage, Multiproduct Plant**
- **Possible extensions to the basic model**
- **Advantages and disadvantages**

Possible Extensions to the Basic Model

- Several objective functions: different performance measures involving earliness, tardiness, a combination of them, number of tardy jobs, etc. (Zeballos & Henning, 2003)
- Consideration of sequence-dependent changeover times (Zeballos & Henning, 2003)
- Inclusion of topology constraints, forbidden unit-order pairs, resource constraints (Zeballos & Henning, 2003)
- Hybrid MIP-CP approaches (Maravelias & Grossmann, 2004; Harjunkoski & Grossmann, 2002; Zeballos & Henning, 2003; Harjunkoski, Jain & Grossmann, 2002).

Incorporation of Changeover Times to the Basic Model

```
enum Jobs...;
enum Equipment...;
enum Stages...;
enum Products...;
{Equipment} belongsto[Stages] = ...;
Products ProductOfJob[1..card(Jobs)] = ...;
Products JobProduct[Jobs];
initialize {
   forall(j in Jobs)
      JobProduct[j] = ProductOfJob[ord(j)+1];
        };
int+ ProcTime[Jobs, Equipment]=...;
int+ TransitionTimeProducts[Products,Products]=...;
scheduleHorizon = 500;

// Variables´ declaration
//Tasks´ Declaration – Resources declaration
Activity Task[j in Jobs, St in Stages] transitionType JobProduct[j];
Activity makespan(0);
UnaryResource tool[Equipment] ](TransitionTimeProducts);
AlternativeResources s(tool);
```

Specifies the set of Products to be manufactured

Maps a Product into each job

Specifies the state (Product) of each job

Transition times depending on product´s sequence are defined (transition matrix).

Tasks are associated to a transitionType that depends on the state (product) of the job

Unary equipment resources are associated with a transition matrix

Incorporation of Changeover Times to the Basic Model

```
Minimize                    //Objective Function Definition
   makespan.end
subject to {                //Basic Constraints
forall(j in Jobs)
    Task[j,last(Stages)] precedes makespan;
forall(st in Stages)
      forall(j in Jobs)
           Task[j,st] requires s;
forall( j in Jobs)
  forall(St in Stages)
     forall(unit in Equipment)
          (unit not in belongsto[St]) =>
           not activityHasSelectedResource(Task[j,St],s,tool[unit]);
forall(St in Stages)
    forall(unit in belongsto[St])
      forall( j in Jobs)
        activityHasSelectedResource(Task[j,St],s,tool[unit]) =>
        (Task[j,St].duration = ProcTime[j,unit]);
forall(St in Stages : St<last(Stages))
    forall(j in Jobs)
       Task[j,St] precedes Task[j,next(St)];
};
```

Example Data – Sequence Dependent Changeover Times

```
Jobs = {j1, j2, j3, j4, j5};

Equipment = {e1, e2, e3, e4, e5, e6};

Stages = {st1, st2, st3};

Products = {P1, P2, P3, P4, P5};

ProductOfJob = [P1, P2, P3, P4, P5];

belongsto = #[
      st1:{e1,e2},
      st2:{e3,e4},
      st3:{e5,e6}
      ]#;

ProcTime=[
    [ 20, 28, 75, 80, 37, 36],
    [ 33, 31, 71, 70, 35, 33],
    [ 41, 35, 68, 75, 40, 34],
    [ 42, 30, 73, 78, 32, 30],
    [ 30, 33, 70, 74, 33, 35]
  ];
```

Products associated to jobs

```
TransitionTimeProducts=[
    [ 0, 8, 5, 6, 1],
    [ 4, 0, 1, 7, 5],
    [ 2, 3, 0, 1, 4],
    [ 3, 4, 4, 0, 5],
    [ 6, 2, 7, 3, 0]
  ];
```

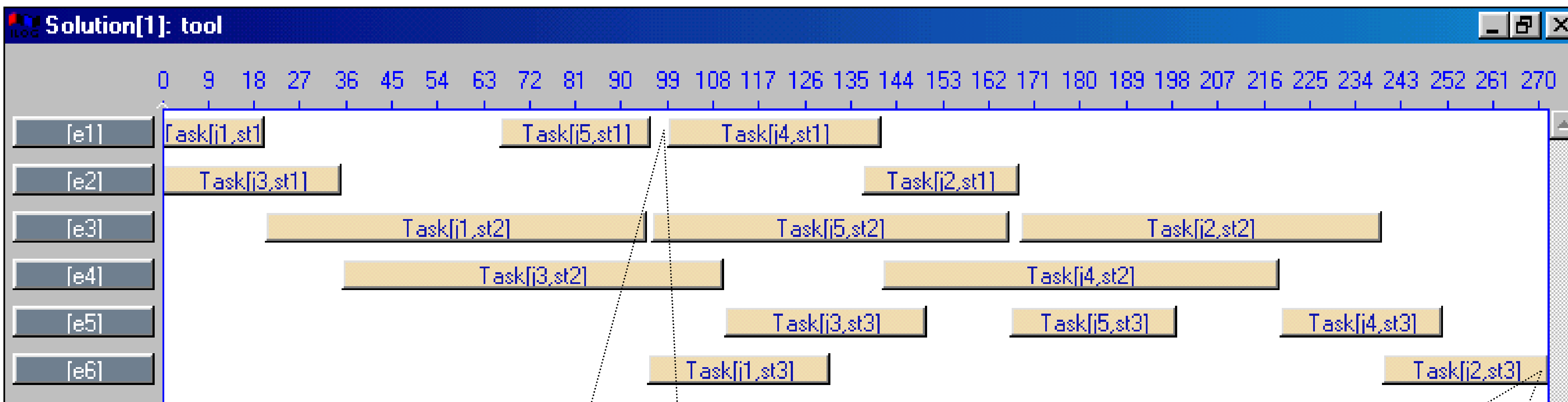
When P5 precedes P4, 3 units of cleaning time are demanded

Results for the NIS Case + Changeover Times

Makespan

Solution[1]: Task

	st1	st2	st3
j1	[0 -- 20 --> 20]	[20 -- 75 --> 95]	[95 -- 36 --> 131]
j2	[137 -- 31 --> 168]	[168 -- 71 --> 239]	[239 -- 33 --> 272]
j3	[0 -- 35 --> 35]	[35 -- 75 --> 110]	[110 -- 40 --> 150]
j4	[99 -- 42 --> 141]	[141 -- 78 --> 219]	[219 -- 32 --> 251]
j5	[66 -- 30 --> 96]	[96 -- 70 --> 166]	[166 -- 33 --> 199]



P5 precedes P4, then 3 units of cleaning time are demanded!!

Malespan has increased from 266 to 272

Overview

- **Definition of Constraint Satisfaction (CS) and Constraint Programming (CP) problems -** Relevance - Applications than can be tackled
- **CP Phases:** Modeling & Solutions´ Generation
- **Generation of Solutions:** Search algorithms – Consistency Enforcing and Constraint Propagation
- **Domain-specific building-blocks for scheduling problems:** Variables, Constraints, Search Algorithms...
- **Scheduling of a Multistage, Multiproduct Plant**
- **Possible extensions to the basic model**
- **Advantages and disadvantages**

Evaluation of CP as a technique for tackling combinatorial problems

- Which technique should be chosen for a given combinatorial problem? (Brailsford et al., 1999)

  Possibilities: CP approaches, OR techniques, local search heuristics (simulated annealing, tabu search, genetic algorithms).
- There are several reasons why a particular technique may be chosen to address a problem. Some of the most important ones are:
  - Easy of implementation
  - Flexibility to handle a variety of constraints that occur in practical problems.
  - Computational time
  - Solution quality
- Unfortunately, the literature contains very few direct comparisons of CP with other approaches. Existing comparisons are partial and incomplete.

Evaluation of CP – Easy of implementation and flexibility

- **For easy of implementation and flexibility, CP scores highly relative to most of the OR techniques**.
  - Constraints can be formulated in a quite natural and intuitive manner.
  - Problem-specific constraints can be added without any need to revise the whole program. The incorporation of extra constraints is not a burden.
  - Formulations tend to benefit from the addition of redundant constraints
  - CP is often the appropriate approach for problems having very different variables and constraints, such as those with integer, logical and choice variables and/or linear, non-linear and logical constraints.

Evaluation of CP – Computational time and solution quality

- CP performance, with respect to solution quality and computation time, as well as the one of other OR techniques, tends to be problem dependent.
- A critical factor in the success of a CP approach is the amount of propagation that the constraints permit. CP is likely to be successful if the nature of the problem is such that a variable instantiation triggers the pruning of many values from the domains of the other variables. Scheduling and time-tabling belong to this category of problems.
- CP performs better on problems that are highly constrained and therefore have a small search space. MIP is superior in problems with a large search space and no strong constraints.
- While CP relies mainly on constraint propagation to restrict the size of the search space, the efficiency of branch and bound based techniques is highly dependent on the bounding scheme.

Suggested Papers – Relevant Weblinks

Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results, P. Laborie, *Artificial Intelligence* 143, 151–188, (2003).
http://www.sciencedirect.com/science/journal/00043702

Constraint satisfaction problems: Algorithms and applications, S. Brailsford, C. Potts, B. Smith, *European Journal of Operational Research* 119 557-581, (1999).
http://www.sciencedirect.com/science/journal/03772217

Program Does Not Equal Program: Constraint Programming and Its Relationship to Mathematical Programming (PDF), Irvin J. Lustig and Jean-Francois Puget, *Interfaces* Vol 31., No. 6, pp. 29-53, December, 2001
http://pubsonline.informs.org/feature/pdfs/0092.2102.01.3106.29.pdf

Constraint Satisfaction and Constraint Programmng. Pedro Meseguer. IBERAMIA-02 Invited Talk, Universidad Autónoma de Barcelona.
http://www.iiia.csic.es/~pedro/Conf-Iberamia-02.pdf

Constraint Programming Techniques for Batch Scheduling

Gabriela P. Henning

INTEC (Universidad Nacional del Litoral – CONICET)
Santa Fe – Argentina

August 2005

E-mail: ghenning@intec.unl.edu.ar