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** \( X = \{x_1, x_2, \ldots, x_n\} \);
- 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 $C_{ijk} \subseteq D_i \times D_j \times D_k \times ...$, that the constraint allows.

Example:

$$C_{xy}: 2x = y$$

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

$$D_y = \{1, 2, \ldots, 15\}$$

• 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

- Map Coloring Problem
- 8 Queens Problem
- Vehicle Routing Problem

- Customers with time windows
- Depot
- 3 Vehicles having a given capacity
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 network:
  - Symbolic constraint satisfaction problems (e.g., Puzzles, qualitative temporal/spacial reasoning)
  - Boolean constraint satisfaction problems (Circuits)
  - Constraint satisfaction problems on reals
  - Logic Constraints
  - ........

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!!

Non-trivial task!
Look for an appropriate representation!
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 $\leftarrow$ value

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

4-Queens Problem

Gabriela Henning – PASI – Iguazú Falls – August 2005
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 $\in$ partial solution (assigned variables)
• Future $\notin$ 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 and 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 → **loookahead**
- 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.

• Maintaining 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 $C_{ij}$.

![Diagram showing arc-consistency](image)

- Original Domains
- Domains when $(x,y)$ is arc-consistent
- Domains when $(x,y)$ and $(y,x)$ are arc-consistent
Consistency Enforcing

• The binary arc-consistency concept can be generalized to consider arbitrary constraints → 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 → Constraint Propagation!!!!
Example: Constraint Propagation and Domain Reduction

Variables: \( x, y \)

Domains:
- \( D_x = \{1, 2, \ldots, 10\} \)
- \( D_y = \{1, 2, \ldots, 15\} \)

Constraints:
- \( 2x = y \)
- \( y \leq 15 \)
- \( (x \text{ modulo } 2) = 1 \)

\( D_x \):
- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

\( D_y \):
- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

Diagram:
- From \( D_x \) to \( D_y \): 
  - \( 2x = y \)
  - \( y \leq 15 \)
  - \( (x \text{ modulo } 2) = 1 \)
- From \( D_y \) to \( D_x \):
  - \( 2x = y \)
  - \( y \leq 15 \)
  - \( (x \text{ modulo } 2) = 1 \)
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 $\rightarrow$ Compete for resources
Problem Variables: Resources

- Different types of resources, demanded by various kinds of activities, need to be represented:
  - Non-Renewable Resources
  - Renewable Resources
    - Reservoir 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
  \[\text{task}[j,t].\text{end} \leq \text{task}[j,t'].\text{start}; \forall t \in T, \forall j \in J,\]
  \[\text{Ord}(t') = \text{Ord}(t) + 1\]
  Equivalent to: \text{task}[j,t] \textit{ precedes} \text{task}[j,t']

• **Disjunctive constraints on unary resources:**
  Let \text{task}[j1,t] \text{ and } \text{task}[j2,t] \text{ be two activities that require the same unary resource (e.g. the same processing unit). Then:}
  \[\text{task}[j1,t].\text{end} \leq \text{task}[j2,t].\text{start} \lor \text{task}[j2,t].\text{end} \leq \text{task}[j1,t].\text{start}; \text{ which is equivalent to }\]
  \text{task}[j1,t] \textit{ precedes} \text{task}[j2,t] \text{ 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 $\Omega$ of activities competing for the same resource and perform propagation based on the position of activities in $\Omega$

• 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 $(\Omega, A)$ such that task $A$ cannot precede (follow) any task from $\Omega$ in all feasible schedules, and updates the earliest starting time (latest completion time) of task $A$ accordingly.
Edge-finding

- Let $\Omega$ be a subset of activities on a unary resource, and $A \not\in \Omega$ another activity on the same unary resource.

- Let $\text{start}_{\min}(X)$, $\text{end}_{\max}(X)$ and $\text{dur}_{\min}(X)$ respectively denote the minimal start time, maximal end time and minimal duration over all the activities in the set $X$. Let $A.lct$ be the latest completion time of activity $A$.

- Most edge-finding techniques can be captured by the following rule:

$$\text{end}_{\max}(\Omega \cup A) - \text{start}_{\min}(\Omega) < \text{dur}_{\min}(\Omega \cup A) \Rightarrow \ A.lct \leq \min_{\Omega' \subseteq \Omega}(\text{end}_{\max}(\Omega') - \text{dur}_{\min}(\Omega'))$$

New Bound!
Example of Edge-finding Propagation

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

\[
\begin{align*}
est &= \text{earliest start time} \\
lst &= \text{latest start time} \\
est &= \text{earliest completion time} \\
lct &= \text{latest completion time}
\end{align*}
\]

If \( A = A4 \) and \( \Omega = \{A1, A2, A3\} \), the conditions of the propagation rule are satisfied as \( \text{end}_{\text{max}}(\Omega \cup A) = 16 \), \( \text{start}_{\text{min}}(\Omega) = 6 \) and \( \text{dur}_{\text{min}}(\Omega \cup A) = 11 \) (16 – 6 < 11).

By taking \( \Omega' = \{A1, A2, A3\} \), 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
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.

- \texttt{activityHasSelectedResource} 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:** \texttt{activityA requires (10) UtilityU}.

  Specifies that \texttt{activityA requires 10 units of UtilityU during its execution}. 

### Special Constructs

- \texttt{activityHasSelectedResource} 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 \texttt{activityA} has selected resource \texttt{u} among the set of alternative resources \texttt{s}.

- \texttt{activityHasSelectedResource} 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:** \texttt{activityA requires (10) UtilityU}.

  Specifies that \texttt{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

Stage 1

Stage 2

Stage 3

j1, j2, j3, j4, j5
CP Model: Scheduling of a Multistage, Multiproduct Plant

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

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

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

- **Enum Jobs...**
- **Enum Equipment...**
- **Enum Stages...**
- `{Equipment} belongsto[Stages] = ...;` sets the equipment units that belong to each stage.
- `scheduleHorizon = 500;` declares the scheduling horizon.

---

Gabriela Henning – PASI – Iguazú Falls – August 2005
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

Jobs = \{j_1, j_2, j_3, j_4, j_5\};
Equipment = \{e_1, e_2, e_3, e_4, e_5, e_6\};
Stages = \{st_1, st_2, st_3\};

belongsto = [#
    st_1:\{e_1, e_2\},
    st_2:\{e_3, e_4\},
    st_3:\{e_5, e_6\}
]#;

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)
Results for the ZW Case (Zero-Wait)

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

Incorporation of **Changeover Times** to the Basic Model

```plaintext
enum Jobs...;
enum Equipment...;
enum Stages...;
enum Products...;

{Equipment} belonsto[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

Gabriela Henning – PASI – Iguazú Falls – August 2005
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 = \{j_1, j_2, j_3, j_4, j_5\};

Equipment = \{e_1, e_2, e_3, e_4, e_5, e_6\};

Stages = \{st_1, st_2, st_3\};

Products = \{P_1, P_2, P_3, P_4, P_5\};

ProductOfJob = [P_1, P_2, P_3, P_4, P_5];

belongsto = #[
  st_1:\{e_1,e_2\},
  st_2:\{e_3,e_4\},
  st_3:\{e_5,e_6\}
]#;

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]
];

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 P_5 precedes P_4, 3 units of cleaning time are demanded.

Products associated to jobs

When P_5 precedes P_4, 3 units of cleaning time are demanded.

Gabriela Henning – PASI – Iguazú Falls – August 2005
Results for the NIS Case + Changeover Times

<table>
<thead>
<tr>
<th>Solution[1]: Task</th>
<th>st1</th>
<th>st2</th>
<th>st3</th>
</tr>
</thead>
<tbody>
<tr>
<td>j1</td>
<td>[0 -- 20 --&gt; 20]</td>
<td>[20 -- 75 --&gt; 95]</td>
<td>[95 -- 36 --&gt; 131]</td>
</tr>
<tr>
<td>j2</td>
<td>[137 -- 31 --&gt; 168]</td>
<td>[168 -- 71 --&gt; 239]</td>
<td>[239 -- 33 --&gt; 272]</td>
</tr>
<tr>
<td>j3</td>
<td>[0 -- 35 --&gt; 35]</td>
<td>[35 -- 75 --&gt; 110]</td>
<td>[110 -- 40 --&gt; 150]</td>
</tr>
<tr>
<td>j4</td>
<td>[99 -- 42 --&gt; 141]</td>
<td>[141 -- 78 --&gt; 219]</td>
<td>[219 -- 32 --&gt; 251]</td>
</tr>
<tr>
<td>j5</td>
<td>[66 -- 30 --&gt; 96]</td>
<td>[96 -- 70 --&gt; 166]</td>
<td>[166 -- 33 --&gt; 199]</td>
</tr>
</tbody>
</table>

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

http://www.sciencedirect.com/science/journal/00043702

http://www.sciencedirect.com/science/journal/03772217

http://pubsonline.informs.org/feature/pdfs/0092.2102.01.3106.29.pdf

Constraint Satisfaction and Constraint Programming. Pedro Meseguer. IBERAMIA-02 Invited Talk, Universidad Autónoma de Barcelona.
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