AIPS'02 Tutorial : Constraint-Based Scheduling in an A.I. Planning and Scheduling Perspective

I. Lecturers

Philippe Laborie, plaborie@ilog.fr, ILOG S.A., France
Wim Nuijten, wnuijten@ilog.fr, ILOG S.A., France

II. Objective

The objective of the tutorial is to give an overview of both the most widely used techniques as well as new emerging techniques in Constraint-Based Scheduling. The tutorial is particularly addressed to A.I. Planning or Scheduling researchers who are interested in a flexible framework in which AI Planning and Scheduling can smoothly be integrated.

III. Abstract

Constraint Programming is a problem-solving paradigm that establishes a clear distinction between two pivotal aspects of a problem: (1) a precise definition of the constraints that define the problem to be solved and (2) the algorithms and heuristics enabling the selection of decisions to solve the problem. It is because of these capabilities that Constraint Programming is increasingly being employed as a tool to solve scheduling problems. Hence the development of Constraint-Based Scheduling as a field of study.

This tutorial provides an overview of the most widely used or emerging Constraint-Based Scheduling techniques. It particularly emphasizes the features of Constraint-Based Scheduling that makes it a good framework for integrating A.I. Planning and Scheduling.

The first part of the tutorial provides an overview of Constraint Programming as well as a model for representing scheduling problems in the Constraint Programming framework. Some classical examples of scheduling problems are given. The end of this introductory part investigates the relations between Scheduling and A.I. Planning. Following the clear distinction between (1) constraint propagation and (2) search space exploration provided by the Constraint Programming paradigm, those two aspects are treated in the second and third parts of the tutorial. The second part is focused on the propagation of resource constraints, which usually are responsible for the "hardness" of a scheduling problem. The notion of a global constraint is introduced as a constraint that enforces a set of local properties on the partial schedule. The different techniques to propagate global resource constraints are then classified, described and compared. The third part of the tutorial presents some classical search techniques in Constraint-Based Scheduling. It describes the most classical branching schemes and control of the search as well as some widely used techniques to deal with large and/or hard problems. In order to show how to put things together, the resolution of two scheduling problems is described in the last part. These examples illustrate the use and the practical efficiency and flexibility of the Constraint-Based Scheduling approach.

IV. Prerequisite knowledge

Some knowledge about Constraint Programming and Scheduling is an advantage but is not required as the first part of the tutorial will recap the main notions. Some basic knowledge about Partial Order Planning may also be useful.

V. About the lecturers

Dr. Philippe Laborie graduated from Ecole Nationale Supérieure des Télécommunications (Paris) in 1992, and received a PhD in Artificial Intelligence from LAAS/CNRS (Toulouse) on the integration of A.I. Planning and Scheduling in 1995. He is one of the developers of the IxTeT Planning system. From 1996-1997, he worked as postdoctoral fellow at Electricité de France (Paris) and INRIA/IRISA (Rennes) on the Supervision and Diagnosis of complex systems (telecommunication and power distribution networks). His main scientific interests include planning, scheduling, supervision and diagnosis of complex systems and more generally, all decision problems dealing with time. Since 1998 he has been at ILOG S.A. in Gentilly, France, where he currently holds the position of Principal Scientist.

Dr. Wim Nuijten is Director of Optimization Technology at ILOG S.A. His main interests lie in using constraint programming, local search, mathematical programming, and their combination to solve scheduling problems. He received his MSc (cum laude) in 1990 and his PhD in 1994 from the Department of Mathematics and Computing Science of Eindhoven University. From 1992-1994 he was a consultant at RIKS in Maastricht. Begin 1995 he joined ILOG as a software developer, where later that year he became the Project Manager of ILOG Scheduler. In the years that followed Dr. Nuijten was a driving force behind making ILOG Scheduler the industrial success it is today. He was closely involved in the introduction and successful application of constraint-based scheduling in several

major companies, amongst which SAP and Oracle. Since mid 1999 he is in his current position and leads a team of about 15 researchers that develop an array of constraint programming products.

VI. Outline

1. Basic principles
1.1. Constraint Programming
- Definition of a CSP [1]
- Constraint propagation [2], [3], [4], [5], [6], [7]
- Search techniques
  Search heuristics
  Search tree exploration: DFS, LDS, ...[8]
- Optimizing objective functions

1.2. Scheduling model [9], [10], [11]
- Typology of activities: non-preemptive, preemptive
- Temporal constraints [12]
- Typology of resources, alternative resources
- Extensions:
  state resources
  resource efficiency
  transition times/costs
- Objective functions

1.3. Examples of scheduling problems
- jobshop [13]
- RCPSP with min/max time lags [14]

1.4. Scheduling and A.I. Planning
- Relations with Partial Order Planning [15], [16]
- Close/not-close status of a resource [17], [18]

2. Propagation of Resource Constraints
2.1. Global constraints
- From local properties to global constraints
- Local properties and truth criteria [19], [20]
- A framework to classify global constraints

2.2. Algorithms based on activity time-windows [21], [22]
- Timetabling constraint [23]
- Disjunctive constraint [24]
- Edge-finding constraint [25], [26]
- Not-first, not-last [27]
- Energetic reasoning [28], [29]

2.3. Algorithms based on relative position of activities
- Precedence energy constraint [30], [31]
- Balance constraint [32], [31]

2.4. Comparison of global constraints [33], [10]

3. Search Techniques
3.1. Branching schemes
- Branching schemes and local properties
- Classical branching schemes:
  Resource allocation
  Setting times
  Rank first/not-first (last/not-last)
  Pair ordering [34], [35]

3.2. Search heuristics
- Slack-based heuristics [36]
- Texture-based heuristics [37]

3.3. Beyond the basic search scheme
- Shaving [38]
- Dichotomizing
- Probing [39], [40]
- Large neighborhood search [41], [42], [43]

4. Putting things together on two examples
4.1. Jobshop [44], [45], [30]
4.2. RCPSP with min/max time lags [14], [31]

5. Conclusion

References

[1] V. Kumar, "Algorithms for Constraint Satisfaction Problems: a Survey," *AI magazine*, vol. 13, no. 1, pp. 32–44, 1992.

[2] R.M. Stallman and G.J. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence*, vol. 9, no. 2, pp. 135–196, 1977.

[3] R. Kowalski, "Algorithm = Logic + Control," *Communication of the ACM*, vol. 22, no. 7, pp. 424–436, 1979.

[4] G.L. Steele, *The Definition and Implementation of a Computer Programming Language Based on Constraints*, Ph.D. thesis, Massachusetts Institute of Technology, 1980.

[5] R. Mohr and T. Henderson, "Arc and path consistency revisited," *Artificial Intelligence*, vol. 28, pp. 225–233, 1986.

[6] J.C. Régin, "A filtering algorithm for constraints of difference in CSPs," in *Proc. 12th National Conference on Artificial Intelligence*, 1994.

[7] C. Bessière and J.C. Régin, "Refining the basic constraint propagation algorithm," in *Proceedings IJCAI'01*, 2001.

[8] W. Harvey and M. Ginsberg, "Limited discrepancy search," in *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, August 1995.

[9] K.R. Baker, *Introduction to Sequencing and Scheduling*, John Wileys and Sons, 1974.

[10] P. Baptiste, C. LePape, and W. Nuijten, *Constraint-Based Scheduling*, Kluwer Academics Publishers, 2001.

[11] ILOG, "ILOG Scheduler 5.2 Reference Manual," 2001, http://www.ilog.com/.

[12] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, no. 1-3, pp. 61–96, may 1991.

[13] A. Jain and S. Meeran, "A state-of-the-art review of job-shop scheduling techniques," Tech. Rep., Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, 1998.

[14] K. Neumann and C. Schwindt, "Project scheduling with inventory constraints," Tech. Rep. WIOR-572, Institut für Wirtschaftstheorie und Operations Research. Universität Karlsruhe, 1999.

[15] D.E. Smith, J. Frank, and A.K. Jonsson, "Bridging the gap between planning and scheduling," *Knowledge Engineering Review*, vol. 15, no. 1, 2000.

[16] X. Nguyen and S. Kambhampati, "Reviving Partial Order Planning," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001, pp. 459–464.

[17] C. Knoblock, "Automatically generating abstractions for planning," *Artificial Intelligence*, vol. 68, pp. 243–302, 1994.

[18] F. Garcia and P. Laborie, *New Directions in AI Planning*, chapter Hierarchisation of the Search Space in Temporal Planning, pp. 217–232, IOS Press, Amsterdam, 1996.

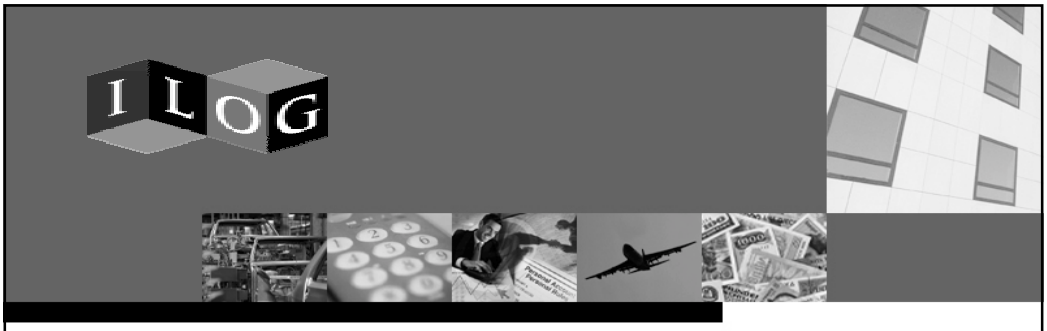
[19] D. Chapman, "Planning for conjunctive goals," *Artificial Intelligence*, vol. 32, pp. 333–377, 1987.

[20] D. Weld, "An Introduction to Least Commitment Planning," *AI Magazine*, vol. 15, no. 4, pp. 27–61, 1994.

[21] W. Nuijten, *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*, Ph.D. thesis, Eindhoven University of Technology, 1994.

[22] U. Dorndorf, T. Phan Huy, and E. Pesch, "A survey of interval capacity consistency tests for time and resource constrained scheduling," in *Project Scheduling - Recent Models, Algorithms and Applications, Kluwer Academic Publ.*, 1999, pp. 213–238.

[23] C. LePape, "Implementation of Resource constraints in ILOG Schedule: A Library for the Development of Constraint-Based Scheduling systems," *Intelligent Systems Engineering*, vol. 3, no. 2, pp. 55–66, 1994.
[24] S. Khambhampati and X. Yang, "On the role of disjunctive representations and constraint propagation in refinement planning," in *KR96*, 1996.
[25] J. Carlier and E. Pinson, "A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem," *Annal of Operation Research*, vol. 26, pp. 269–287, 1990.
[26] P. Baptiste and C. LePape, "Edge-Finding Constraint Propagation Algorithms for Disjunctive and Cumulative Scheduling," in *Proc. 15th Workshop of the UK Planning Special Interest Group*, 1996.
[27] P. Torres and P. Lopez, "On Not-First/Not-Last Conditions in Disjunctive Scheduling," *European Journal of Operational Research*, 1999.
[28] J. Erschler, *Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement*, Ph.D. thesis, Université Paul Sabatier, 1976.
[29] J. Erschler, P. Lopez, and C. Thuriot, "Raisonnement temporel sous contraintes de ressources et problèmes d'ordonnancement," *Revue d'Intelligence Artificielle*, vol. 5, no. 3, pp. 7–32, 1991.
[30] F. Sourd and W. Nuijten, "Multiple-machine lower bounds for shop scheduling problems," *INFORMS Journal of Computing*, vol. 4, no. 12, pp. 341–352, 2000.
[31] P. Laborie, "Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results," in *Proceedings ECP'01*, 2001.
[32] A. Cesta and C. Stella, "A time and resource problem for planning architectures," in *ECP-97*, 1997.
[33] P. Baptiste and C. LePape, "A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling," in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995.
[34] P. Laborie and M. Ghallab, "Planning with sharable resource constraints," in *Fourteenth IJCAI*, 1995, pp. 1643–1649.
[35] A. Cesta, A. Oddi, and S. Smith, "A constraint-based method for project scheduling with time windows," Tech. Rep., CMU RI Technical Report, 2000.
[36] S. F. Smith and C. Cheng, "Slack-based heuristics for constraint satisfaction scheduling," in *Proc. 11th Nat. Conf. on AI*, 1993, pp. 139–144.
[37] C. Beck, A. Davenport, E. Sitarski, and M. Fox, "Texture-based Heuristics for Scheduling Revisited," in *Proceedings AAAI-97*, 1997.
[38] P. Torres and P. Lopez, "Overview and possible extensions of shaving techniques for job-shop problems," in *2nd International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'2000)*, March 2000, pp. 181–186.
[39] H. El Sakkout and M. Wallace, "Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling," *Constraints*, vol. 5, no. 4, pp. 359–388, 2000.
[40] C. Beck and P. Refalo, "A Hybrid Approach to Scheduling with Earliness and Tardiness Costs," in *Third International Workshop on Integration of AI and OR Techniques (CP-AI-OR'01)*, 2001.
[41] G. Pesant and M. Gendreau, "A View of Local Search in Constraint Programming," in *Proc. 2nd International Conference on Principles and Practice of Constraint Programming*, 1996.
[42] P. Shaw, V. Furnon, and B. De Backer, "A Lightweight Addition to CP Frameworks for Improved Local Search," in *International Workshop on Integration of AI and OR Techniques (CP-AI-OR 2000)*, 2000.
[43] F. Focacci, P. Laborie, and W. Nuijten, "Solving scheduling problems with setup times and alternative resources," in *Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 2000, pp. 92–101.
[44] W. Nuijten and C. LePape, "Constraint-Based Job-Shop Scheduling with ILOG Scheduler," *Journal of Heuristics*, vol. 3, pp. 271–286, 1998.
[45] C. LePape and P. Baptiste, "Heuristic Control of a Constraint-Based Algorithm for the Preemptive Job-Shop Scheduling Problem," *Journal of Heuristics*, vol. 5, pp. 305–332, 1999.



Constraint-Based Scheduling in an A.I. Planning & Scheduling Perspective

Philippe Laborie and Wim Nuijten
ILOG
[plaborie, wnuijten]@ilog.fr

Overview

- Basic Principles
- Propagation of Resource Constraints
- Search Techniques
- Putting things together on two examples
- Conclusion

Overview

- Basic Principles
- Propagation of Resource Constraints
- Search Techniques
- Putting things together on two examples
- Conclusion

Basic Principles

Overview

- Constraint Programming
- Scheduling Model
- Examples of Scheduling Problems
- Scheduling and A.I. Planning

Basic Principles

Overview

- Constraint Programming
- Scheduling Model
- Examples of Scheduling Problems
- Scheduling and A.I. Planning

Basic Principles

Constraint Programming

- **Constraint-Based Scheduling =**
  Scheduling + Constraint Programming

- **Scheduling problems arise in situations where a set of** *activities* **has to be processed by a** *limited* **number of** *resources* **in a** *limited* **amount of** *time***.**

Constraint Programming

- **Paradigm to solve combinatorial optimization problems.**
- **One states the problem in terms of** *variables* **and** *constraints* **after which, a** *search* **procedure is used to find a solution.**

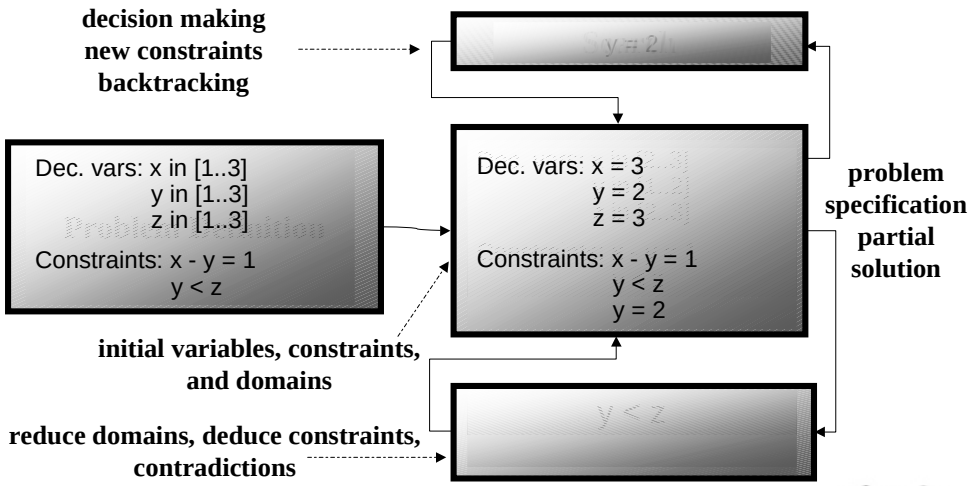
Constraint Programming

- **Each variable has an initial *domain* of possible values (x $\in$ [0,…,4])**
  - Constraints *propagate* to reduce domains

    "x < 3" $\Rightarrow$ x $\in$ [0,…,2]
  - Each constraint has its own propagation algorithm (reusable).
  - As propagation isn't sufficient, a search tree is build.

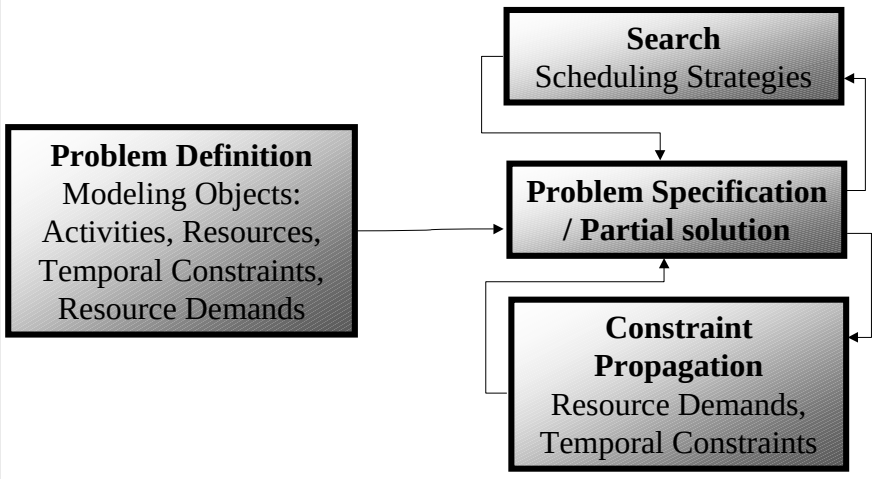
Basic Principles

Constraint Programming



Basic Principles

Constraint-Based Scheduling



Constraint Programming : Properties

- **Besides the separation of *problem definition*, *constraint propagation*, and *search,* the Locality Principle** [Steele, 80] **is important: each constraint must propagate as locally as possible, independent of the existence or non-existence of other constraints.**
- [Stallman & Sussman, 77] **separation between deductive methods (constraint propagation) and search**
- [Kowalski, 79] **distinction between logical representation of constraints and the control of their use: Algorithm = Logic + Control**

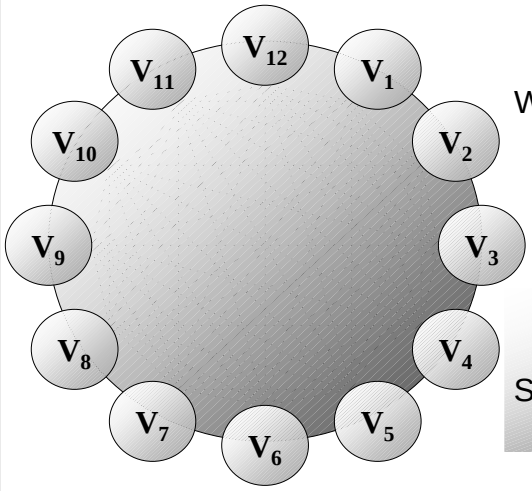
Constraint Programming

- **Global Constraints**
  - One of the strong points of CP tools are the so called *global* constraints (Opposed to constraints like x < y.)
  - Example is the “all-different” constraint where you have n variables which all should take a different value:

$$\forall i, j \in [1,...,n] / i \neq j, (x_i \neq x_j)$$

Basic Principles

Constraint Programming



$O(n^2)$ binary constraints
Weak constraint propagation

One global constraint
(Easier modeling)
Strong constraint propagation
[Regin 94]

Basic Principles

Constraint Programming : Search

- **Search Heuristics**
  - Variable and value selection heuristics.
  - Choose variable to assign a value to and then choose a value for this variable (instantiate the variable).
  - Typical strategy
    - Choose most constrained variable, i.e., a variable that is difficult to instantiate; start with the difficult parts (this before they get even more difficult).
    - Choose least constraining value, i.e., a value that leaves as many values as possible for the remaining uninstantiated variables.

Constraint Programming : Search

- **Variable selection heuristics:**
  - choose variable with smallest remaining domain
  - choose variable with maximal degree in constraint graph (most constraints defined on it)

Constraint Programming : Search

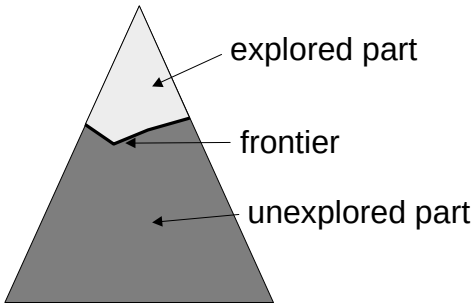
- **Value selection heuristics:**
  - choose value that participates in highest estimated number of solutions. Number of solutions are estimated by counting the solutions to a tree-like relaxation or the original constraint graph.
  - choose minimal value
- **Recall: In general decisions correspond to additional constraints.**

Constraint Programming : Search

- **Search Tree Exploration**
  - Depth First Search (DFS). The traditional search strategy of CP.
  - Best First Search (BFS). Choose the node with the best minimum cost.
  - Limited Discrepancy Search (LDS). Divides the search tree into strips, trying to stick close to the heuristic.
  - … and several others like Depth-Bounded Discrepancy Search, Interleaved Depth-First Search.

Constraint Programming : Search

- **Exploration of the search tree using the Branch & Bound paradigm**

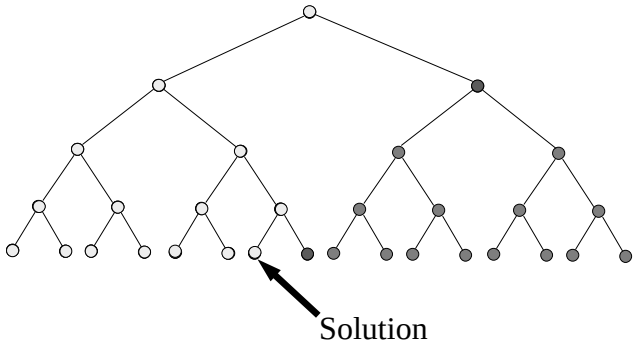


- Set of open nodes is administered.
- Chosing the next open node to expand is what defines the search procedure.

Basic Principles

Constraint Programming : Search

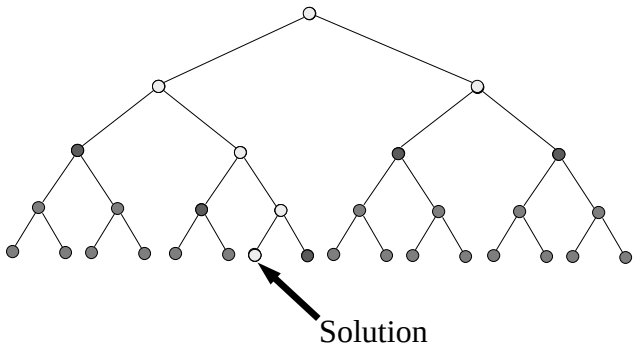
- DFS (Depth First Search)



Basic Principles

Constraint Programming : Search

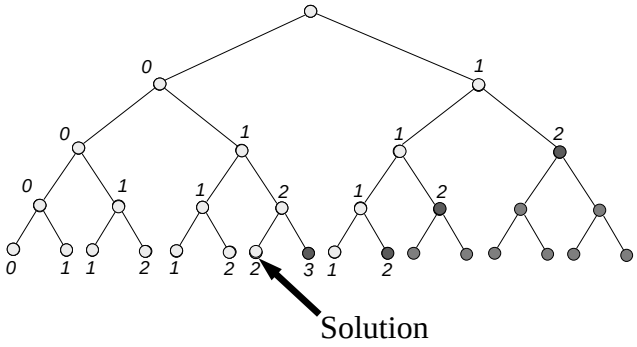
- BFS (Best First Search)



Basic Principles

Constraint Programming : Search

- **LDS (Limited Discrepancy Search)**



Basic Principles

Constraint Programming : Search

- **Incompleteness and completeness**
- **In general in CP both complete and incomplete search is frequently used.**
- **Typical types of incompleteness:**
  - Search heuristic is incomplete.
  - Search tree exploration is incomplete (LDS with truly limited discrepancy).
  - Limits on the search like time limits, limits on the number of choice points, limit on the number of fails.

Constraint Programming : Search

- **Completeness**
- **When doing complete search one can be faced with problems with and without objective functions.**
  - When there's no objective function, one searches "simply" for a solution. (If that fails one has proven the problem is infeasible.)

Constraint Programming : Search

- **Completeness and optimization**
  - In case an objective function is present, one tries to find the best solution. Such optimization can be done in several ways
    - When a solution with cost c is found, restart search to find a solution with cost c-1.
    - Do dichotomization. Besides upper bounds coming from solutions, when being complete one can also calculate lower bounds.
    - When a solution with cost c is found, stay in this solution node, add a "cut" that says the cost needs to be at most c-1, and continue search.

Overview

- Constraint Programming
- Scheduling Model
- Examples of Scheduling Problems
- Scheduling and A.I. Planning

Scheduling Model

- **Scheduling is the "*problem of allocating scarce resources to activities over time*"** [Baker,74]**.**
- **Typology of**
  - Activities
  - Temporal constraints
  - Resources

Scheduling Model: Activities

- **Non-preemptive activities**

Activity *A*

Start time: *s(A)* End time: *e(A)*

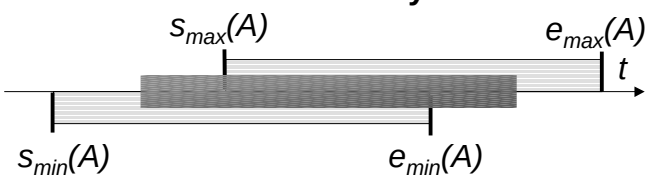
Processing time: *p(A) == e(A)-s(A)*

1 Activity ≡ 3 numerical Variables

Scheduling Model: Activities

- **Notations on current domain of activity time window**

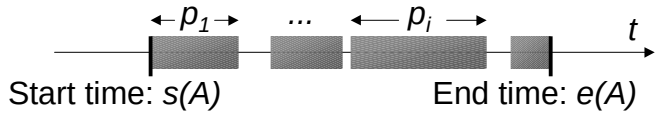
Activity *A*



Scheduling Model: Activities

- **Preemptive activities**

Activity *A*



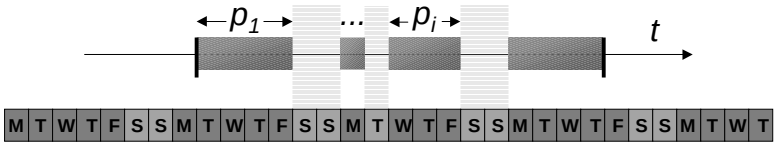
Start time: *s(A)* End time: *e(A)*

Processing time: $p(A) = \Sigma p_i \leq e(A)-s(A)$

Scheduling Model: Activities

- **Preemptive activities**
  - Preemption at fixed dates (break calendar)

Activity *A*

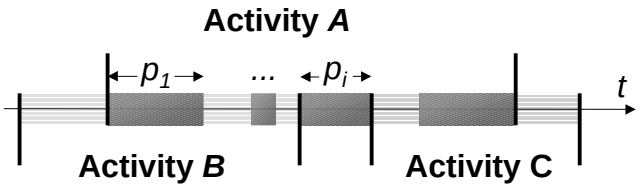


Processing time: $p(A) = \Sigma p_i = e(A)-s(A)$ - *breaks(A)*

1 Activity ≡ 3 numerical Variables

Scheduling Model: Activities

- **Preemptive activities**
  - Preemption by other activities



1 Activity ≡ 1 Variable Set of dates

Scheduling Model: Temporal constraints

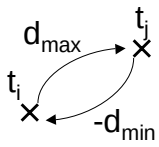
- **Generally expressed as $t_i - t_j \in [d_{min}, d_{max}]$ where:**
  - t_i, t_j : start or end time point of some activities
  - d_{min}, d_{max}: minimal and maximal delay

- **Generic enough to model:**
  - release dates: start(A) ∈ [r,+∞)
  - deadlines: end(A) ∈ (-∞,d]
  - precedence: start(B)-end(A) ∈ [0,+∞)

Scheduling Model: Temporal constraints

- **Simple Temporal Problem (STP)**
  [Dechter & al,91] [Cesta & Oddi,01]

- **Distance Graph representation**



Scheduling Model: Temporal constraints

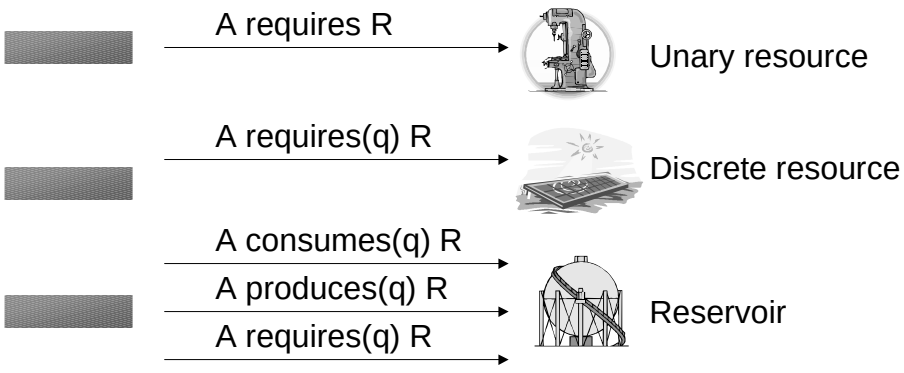
- **Arc consistency (≡ Single Source Shortest Paths) ensures a backtrack-free search**
  - n=#time-points, p=#temporal constraints
  - Memory: O(p)
  - Time: O(n.p) (with negative cycle detection)
- **Path consistency (≡ All Pairs Shortest Paths)**
  - Memory: $O(n^2)$
  - Time: $O(n^2)$
  - Provides all pairs distances (useful for heuristics)

Scheduling Model: Resource Typology

- **Unary resource:**
  - Resource of capacity 1. Two activities requiring the same unary resource cannot overlap.
- **Discrete resource:**
  - Resource of capacity Q. Activities requiring the same discrete resource can overlap provided the resource capacity Q is not exceeded.
- **Reservoir**
  - Resource of capacity Q and initial level L; Activities may consume or produce the reservoir. The level of the reservoir over time must be kept in the interval [0,Q].

Scheduling Model: Resource Usage

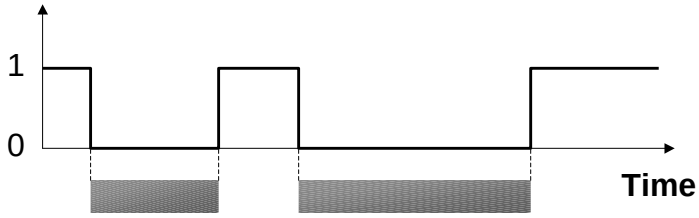
Activities *(A)* **Usage *(U)*** **Resources *(R)***



Basic Principles

Scheduling Model: Resource Usage

Unary Resource Availability

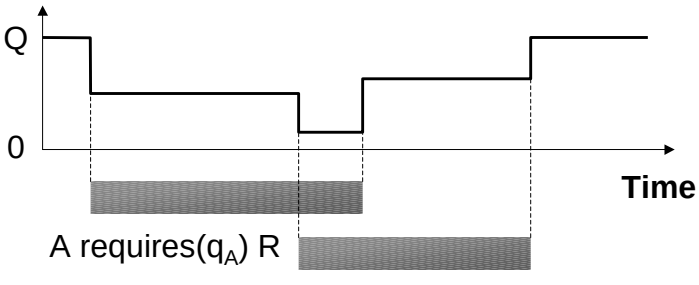


A requires R B requires R

Basic Principles

Scheduling Model: Resource Usage

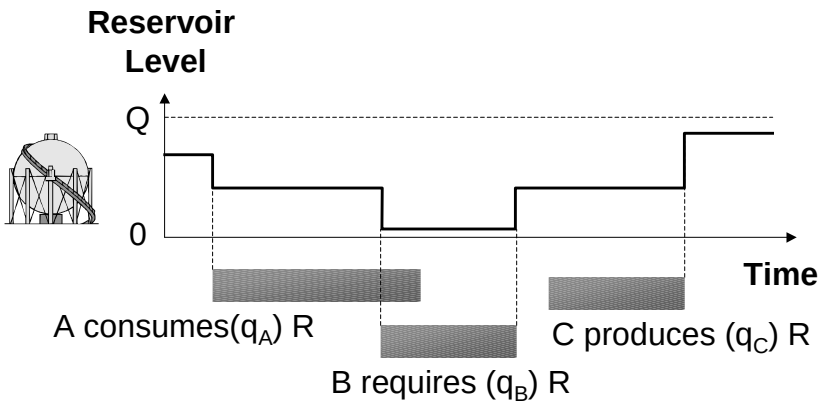
Discrete Resource Availability



A requires(q_A) R

B requires (q_B) R

Scheduling Model: Resource Usage



Scheduling Model: Resource Usage

- **Resource quantities q(U) are decision variables**
- **The same activity may use several resources:**

U_1: *A* requires *Machine1*

U_2: *A* requires(q_2) *Power*

U_3: *A* consumes(q_3) *Fuel*

Scheduling Model: Resource Usage

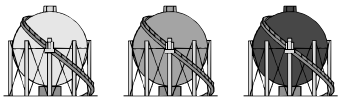
- **Resource usage follow the "constraint protocol".**
- **In particular, they can be used in meta-constraints:**

[A requires Machine1] $\Rightarrow$

[A consumes(q_2) Fuel B] $\vee$ *[B requires(q_3) Power]*

Scheduling Model: Resource Usage

- **Alternative Resource Sets**
  - n equivalent resources S={R_1,…,R_n}



  - *U = [A requires(q) S]* means that
    - One resource in the set S needs to be allocated to activity A. The resource *r(U)*$\in$*S* allocated to A *(its index 1,..,n)* is a decision variable.
    - A will require (or: produce, consume) *q(U)* units of the allocated resource.

Scheduling Model: Resource Usage

- **Alternative Resource Sets**
  - Could be modeled as:

    A requires(q) R_1 ∨ ... ∨ A requires(q) R_n

  - But Alternative Resource Usage allows for:
    - easier to model *(more intuitive)*
    - stronger propagation *(global pruning of the disjunction)*

Scheduling Model: Decision Variables

- **Start, end and processing time of activities:**

  s(A), e(A),p(A)

- **Required quantities of resources:**

  q(U)

- **Allocated resource:**

  r(U)

- **From the standpoint of constraint-programming, the limited capacity of resources imposes some global constraint on the decision variables of the problem**

Scheduling Model: Extensions

- **State Resource**
  - Resource with a finite set of possible states $\{s_1,..,s_n\}$

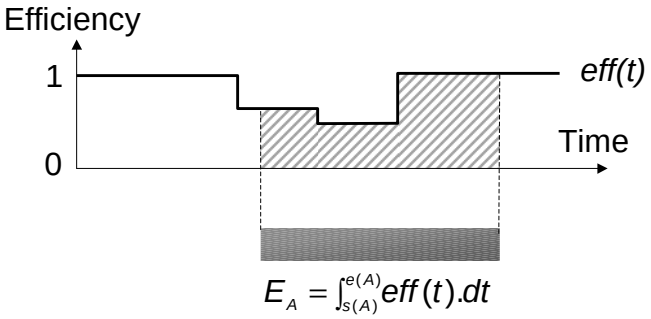
s_1 = 20ºC
s_2 = 150ºC
s_3 = 250ºC
s_4 = 300ºC

  - Activities require the resource be in a particular state s

    U = [A requires(s) R]
  - Activities requiring incompatible state may not overlap

Scheduling Model: Extensions

- **Resource efficiency**
  - The duration of an activity executing on a resource may depend on the efficiency *eff(t)* of the resource



$$E_A = \int_{s(A)}^{e(A)} eff(t).dt$$

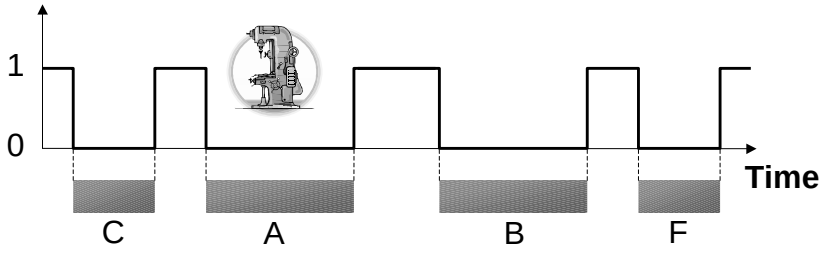
Scheduling Model: Extensions

- **Transition (or setup) times on unary resources**
  - If activity B follows activity A on a unary resource R, then at least $d_{A,B}$ units of time are necessary to setup the resource before starting to process B
  - Similar to a meta-constraint:

$$\forall A,B : [s(B) \geq e(A)] \Rightarrow [s(B) \geq e(A) + d_{A,B}]$$

Scheduling Model: Extensions

- **Transition (or setup) costs on unary resources**
  - If activity B is next to activity A on a unary resource R, then it induces a transition cost $c_{A,B}$.



$$\text{cost} = c_{C,A} + c_{A,B} + c_{B,F}$$

Scheduling Model: Objective functions

- **Often the objective is to minimize a function**

$$f(e(A_1),\ldots,e(A_n))$$

  - Makespan: $f = max(e(A_i))$
  - Total weighted flow time: $f = \Sigma w_i.e(A_i)$
  - Maximum tardiness: $f = max(T_i)$; $T_i = max(0, e(A_i) - \delta_i)$
  - Total weighted tardiness: $f = \Sigma w_i.T_i$
  - Total weighted number of late jobs: $f = \Sigma w_i.U_i$
  - Earliness/Tardiness costs

Scheduling Model: Objective functions

- **Other common objective functions:**
  - Minimum transition costs
  - Minimum peak resource usage
  - Schedule maximum weighted # of activities
  - Schedule on a minimum weighted # of resources

Scheduling Model: Scheduling Problem

- **Given:**
  - A set of resources
  - A set of activities
  - A set of temporal constraints on/between activities
  - A set of resource usage
- **An objective function f**
- **Find an instantiation of the decision variables that**
  - satisfies all the constraints and
  - minimizes f

Scheduling Model: More reality

- **Result of a customer survey over 17 applications using ILOG Scheduler:**
  - Unary Resources in 80% of apps
  - Discrete resources in 80% of apps
  - Reservoirs in 40% of apps
  - State Resources in 35% of apps
  - *Alternative Resources* *in 80% of apps*
  - All with their own constraints and corresponding propagation algorithms

Overview

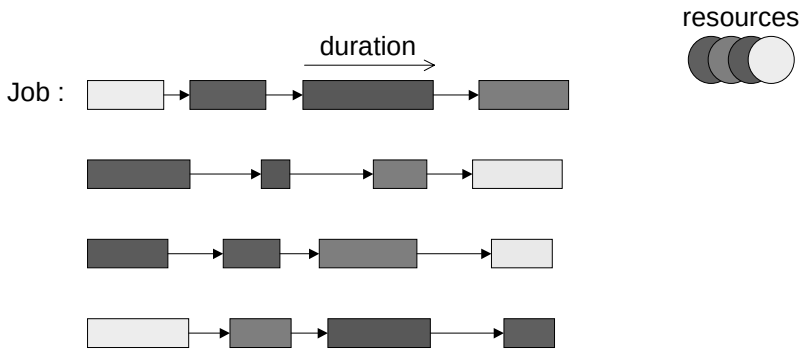
- Constraint Programming
- Scheduling Model
- Examples of Scheduling Problems
- Scheduling and A.I. Planning

Example: Job-Shop Scheduling Problem

- Set of machines.
- Set of jobs, each job is a chain of activities.
- A machine can only process one activity at a time.
- Each activity requires exactly one machine during a given processing time.
- Find a schedule (assignment of start times to activities) that minimizes the makespan.

Basic Principles

Example: Job-Shop Scheduling Problem



Basic Principles

Example: OPL Model

```
int nbMachines = ...;
range Machines 1..nbMachines;
int nbJobs = ...;
range Jobs 1..nbJobs;
int nbActs = ...;
range Acts 1..nbActs;
Machines machineOfAct[Jobs,Acts] = ...;
int+ duration[Jobs,Acts] = ...;

scheduleHorizon = sum(j in Jobs, t in Acts) duration[j,t];
var int makespan in 0..scheduleHorizon;

Activity act[j in Jobs, t in Acts](duration[j,t]);
UnaryResource machine[Machines];
```

Example: OPL Model

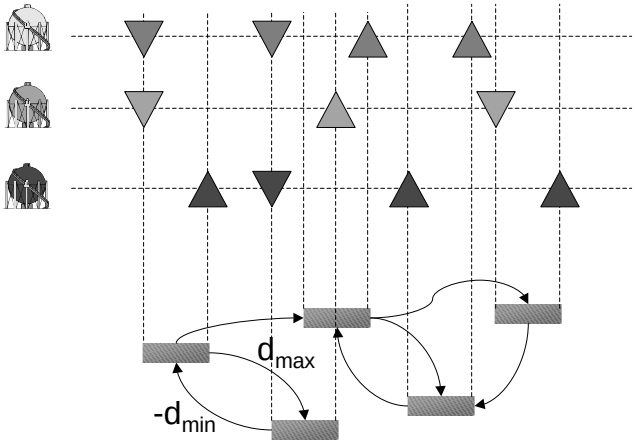
```
minimize
   makespan
subject to {
   forall(j in Jobs) {
      forall(t in 1..nbActs-1)
         act[j,t] precedes act[j,t+1];
      act[j,nbActs] <= makespan
   };

   forall(j in Jobs)
       forall(t in Acts)
          act[j,t] requires machine[machineOfAct[j,t]];
};
```


Example: RCPSPI with time lags

- **Resource-Constrained Project Scheduling Problem with Inventory and min/max time lags** [Neumann & Schwindt,99]
- **Resources:** m reservoirs
- **Activities:** n activities
- **Temporal constraints:** min/max distance graph between activities
- **Resource usage:** each activity produces and/or consumes one or several reservoirs (q constant)
- **Objective function:** minimize makespan

Basic Principles

Example: RCPSPI with time lags



Basic Principles

Example: OPL Model

```
int horizon                    = ...;
int nbActivities               = ...;
int nbReservoirs               = ...;
int+ maxCap[1..nbReservoirs]   = ...;
int+ initLevel[1..nbReservoirs] = ...;

struct TimeLag {
   int before;
   int after;
   int delay;
};
{TimeLag} timelags = ...;

struct Demand {
   int activity;
   int reservoir;
   int quantity;
};
{Demand} demands = ...;
```

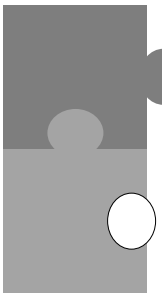
Example: OPL Model

```
var int makespan in 0 .. horizon;
Activity  a[1..nbActivities](1);
Reservoir r[i in 1..nbReservoirs](maxCap[i],initLevel[i]);

minimize
   makespan
subject to {
   forall( t in timelags )
      a[t.before].start + t.delay <= a[t.after].start;
   forall( i in 1..nbActivities )
      a[i].end <= makespan;
   forall( d in demands )
      if d.quantity > 0 then
         a[d.activity] produces(d.quantity)  r[d.reservoir]
      else
         a[d.activity] consumes(-d.quantity) r[d.reservoir]
      endif;
};
```


Overview

- Constraint Programming
- Scheduling Model
- Examples of Scheduling Problems
- Scheduling and A.I. Planning

Scheduling & A.I. Planning



A.I. Planning

Constraint-Based Scheduling
in an A.I. Planning & Scheduling
Perspective

Scheduling & A.I. Planning

- **In Scheduling:**
  - The set of activities to schedule is given as input.
  - Focus on :
    - numerical quantities (time, resources)
    - relative changes of the world
- **In A.I. Planning:**
  - The set of activities is to be generated.
  - Focus on :
    - logical values (predicates, fluents)
    - absolute changes of the world

Scheduling & A.I. Planning

- **STRIPS Planning**
  - A set of propositions $P=\{p_i\}$
  - A set of operators $O=\{o_j\}$
    - Each operator o consists of a 3-uple $(Pre(o),Add(o),Del(o)) \in 2^P \times 2^P \times 2^P$
    - An operator o can be applied to a state $s \in 2^P$ iff $Pre(o) \subset s$.
    - In that case, the execution of operator o on state s leads to a new state: $s' = o.s = s - Del(o) + Add(o)$

Scheduling & A.I. Planning

- **A STRIPS Planning problem is given by:**
  - A set of propositions $P=\{p_i\}$
  - A set of operators $O=\{o_j\}$
  - An initial state $s_0 \subset P$
  - A goal state $s_G \subset P$
- **A solution sequence is a sequence $(o_1,..,o_n)$ of operators that satisfies the constraint that:**
  - o_1 can be applied to s_0, o_2 can be applied to $o_1.s_0$, etc.
  - $s_G \subset o_n. \ldots o_2.o_1.s_0$
- **A partially ordered set of operators is a solution iff all its linearizations are solution sequences**

Scheduling & A.I. Planning

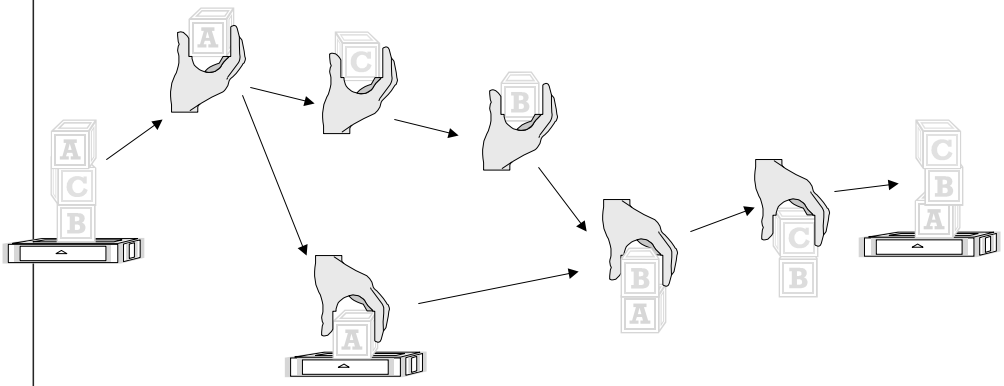
- **Blocks world example**

PUTON(x,y):
Pre ={ free(y), in_hand(x) }
Del ={ free(y), in_hand(x) }
Add={ on(x,y), free(x)}

TAKE(x):
Pre ={ free(x), on(x,y) }
Del ={ free(x), on(x,y) }
Add={ in_hand(x), free(y) }

Scheduling & A.I. Planning

- **Partial Order Planning (POP)**



Scheduling & A.I. Planning

- **Recent advances in both fields have enlarged their relative scope and ambitions**
- **In Scheduling:**
  - Necessity to schedule or not some activities depending on the context: maintenance activity scheduling, alternative processing routes.
- **In A.I. Planning:**
  - Necessity to take into account more complex interactions between actions like time and resources.

Scheduling & A.I. Planning

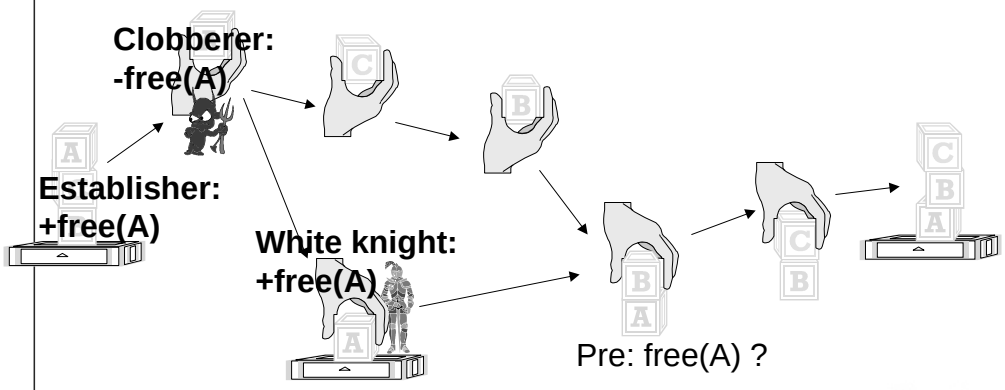
- **A.I. Planning and Scheduling techniques start to converge into a "POPS" (Partial Order Planning and Scheduling) concept**
  - Partial Order Planning is "Reviving" [Nguyen & Khambhampati,01]
  - "Partial Order Scheduling" is most of what Constraint-Based Scheduling is about

Scheduling & A.I. Planning

- **POP and POS work on the same type of structure: a temporal network of operators/activities**
  - POP focus on operator/activity generation constraints
  - POS focus on time and resource constraints
- **The fundamental idea is that in both case, a solution plan or a solution schedule can be characterized by a set of local properties.**
- **These local properties can be enforced by constraint propagation and/or by branching in the search tree.**

Scheduling & A.I. Planning

- **E.g. of such a local property in POP: The Necessary Truth Criterion** [Chapman,87]



Scheduling & A.I. Planning

- In what follow, we identify the local properties that are used in Constraint-Based Scheduling to characterize solution schedules and describe
  1. the propagation (Part 2) and
  2. the branching schemes (Part 3)

  based on these local properties.

Overview

- Basic Principles
- Propagation of Resource Constraints
- Search Techniques
- Putting things together on two examples
- Conclusion

Propagation

Overview

- Global Constraints
- Algorithms based on activity time windows
- Algorithms based on relative position of activities
- Comparison/Complementarity
- Alternative Resources

Propagation

Overview

- Global Constraints
- Algorithms based on activity time windows
- Algorithms based on relative position of activities
- Comparison/Complementarity
- Alternative Resources

Global Constraints

- Recall the OPL model for the JSSP :

```
int nbMachines = ...;
range Machines 1..nbMachines;
int nbJobs = ...;
range Jobs 1..nbJobs;
int nbActs = ...;
range Acts 1..nbActs;
Machines machineOfAct[Jobs,Acts] = ...;
int+ duration[Jobs,Acts] = ...;

scheduleHorizon = sum(j in Jobs, t in Acts) duration[j,t];
var int makespan in 0..scheduleHorizon;

Activity act[j in Jobs, t in Acts](duration[j,t]);
UnaryResource machine[Machines];
```


Global Constraints

```
minimize
   makespan
subject to {
   forall(j in Jobs) {
      forall(t in 1..nbActs-1)
         act[j,t] precedes act[j,t+1];
      act[j,nbActs] <= makespan
   };

   forall(j in Jobs)
       forall(t in Acts)
          act[j,t] requires machine[machineOfAct[j,t]];
};
```

- This looks nice but the real question is :

Global Constraints

- **What's happening inside ?**
  - Most important in the JSSP is the propagation inside the *unary resources* (machines).
  - Key principle is that a resource knows the set of activities that requires it (start, end, processing time).
  - The resource takes care of the $n(n-1)/2$ disjunctive constraints of the form

$$s(A) + p(A) \leq s(A'), \text{ or}$$
$$s(A') + p(A') \leq s(A)$$

Global Constraints

- **What's happening inside ?**
  - When adding an activity requiring a resource, the resource automatically takes care of the relations with all other activities requiring the same resource.
  - Remark: One can add activities during search.
  - Note however that sometimes you need to know that all the activities that will require the resource are known to do any propagation.

Global Constraints

- **What's happening inside ?**
  - We have several types of constraints that all make sure the capacity of the resource is not exceeded at any point. These constraints differ in the strength of the propagation they induce:
    - timetables
    - disjunctive constraint
    - edge-finding
    - precedence energy
    - balance constraint
  - In general: more propagation = more effort.
  - These constraints are global constraints, they "globally" consider the set of activities on a resource.

Global Constraints

- **The scarcity of a resource R impose a global constraint on the set of activities U that use this resource:**

  In a solution schedule, activities using R are scheduled in such a way that for all time t, the resource capacity is never exceeded:

  $$P(U)$$

- **In general, this global constraint can be expressed as a set of local properties:**

  $$\Omega \subset 2^U, \forall \omega \in \Omega, P(\omega)$$

Global Constraints

- **In general, this global constraint can be expressed as a set of local properties:**

  $\Omega \subset 2^U$,

  $\forall \omega \in \Omega$, $P(\omega)$

- **Example: on a unary resource**

  $\Omega = \{\{a,b\}, \{a,b\} \subset U\}$,

  $\forall \{a,b\} \in \Omega$, $[e(a) \leq s(b)] \vee [e(b) \leq s(a)]$

Global Constraints

- **In general, local properties P(ω) can be expressed as a disjunction of constraints:**

  $\Omega \subset 2^U$,

  $\forall \omega \in \Omega$, $p_1(\omega) \vee p_2(\omega) \vee \ldots p_k(\omega)$

- **Example: on a unary resource**

  $\Omega = \{\{a,b\}, \{a,b\} \subset U\}$,

  $\forall \{a,b\} \in \Omega$, $[e(a) \leq s(b)] \vee [e(b) \leq s(a)]$

Global Constraints

- **Remark: same kind of properties in PO Planning**
- **Example: Necessary Truth Criterion [no variables]**

Ω ={(o,p), o$\in$ O, p$\in$ Pre(o)},
$\forall$ (o,p)$\in$ Ω, establish(o,p) $\wedge$ declobber(o,p)

establish(o,p) $\equiv$ $\vee$ {e < o, e $\in$ poss_establishers(o,p)}
declobber(o,p) $\equiv$ $\forall$ c $\in$ clobberer(o,p),
[o<c] $\vee$
[$\vee$ {c<w<o, w $\in$ white_knights(c,o,p)}]

Global Constraints

- **In general, local properties P(ω) can be expressed as a disjunction of constraints:**

$\Omega \subset 2^U$,
$\forall\omega\in$ Ω, $p_1(\omega)\vee$ $p_2(\omega)\vee$... $p_k(\omega)$

- **Constraint Propagation:**
  - For all ω such that none of the constraints $p_2(\omega)$, …, $p_k(\omega)$ is coherent with the current schedule, then, $p_1(\omega)$ can be inferred.
  - Thus ...

Global Constraints

- **In general, constraint propagation rules can be expressed as:**

$$\Omega \subset 2^U,$$
$$\forall \omega \in \Omega, \boxed{C(\omega) \Rightarrow E(\omega)}$$

- **A given local property may be declined into several constraint propagation rules**
- **Framework to classify resource constraint propagation based on what is analyzed in C(ω) and what is inferred in E(ω).**

Global Constraints

$$\forall \omega \in \Omega, C(\omega) \Rightarrow E(\omega)$$

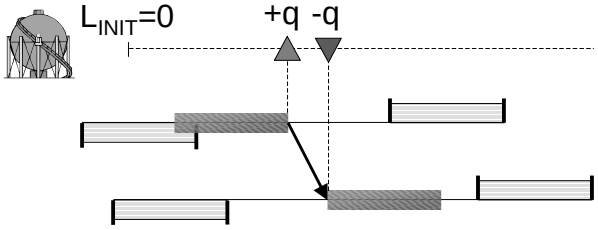
- **Tentative classification:**
  - What's analyzed (C(ω)):
    - Time bounds of activities (s_{min}, s_{max}, e_{min}, e_{max})
    - Precedence relations between activities (A ⟨⟨ B)
  - What kind of reasoning is performed
    - Resource Level Reasoning (Σq)
    - Energetic Reasoning (Σp, Σp.q)
  - What is inferred (E(ω)):
    - New time bounds of activities (s_{min}, s_{max}, e_{min}, e_{max})
    - New not First/Last among ω (¬(A⟨⟨ω))
    - New precedence relations between activities (A ⟨⟨ B)

Global Constraints

- **Different algorithms available to propagate resource capacity** (timetable, disjunctive, edge-finder, balance, etc.)
- **Several algorithms may be used in cooperation on the same resource**
- **A resource knows the set of activities that uses it**
- **As soon as something changes on the resource** (change of activity start/end/processing time, change of resource demand, new resource usage added)**, propagation algorithms are triggered**

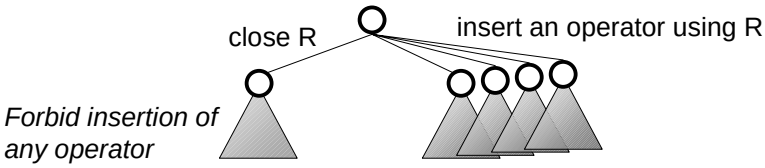
Global Constraints

- **A resource is said to be closed if no additional resource usage will be inserted into the current schedule lower in the search tree.**
- **When a resource is closed, stronger propagation can be inferred.**



Global Constraints

- **In mixed A.I. Planning & Scheduling, a resource can be considered closed if for example :**
  - No operator use this resource; or
  - In hierarchical planning, it corresponds to an already processed abstraction level; or
  - The decision has been taken to close the resource on the current sub-tree :



Overview

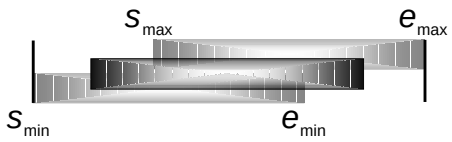
- **Global Constraints**
- **Algorithms based on activity time windows**
- **Algorithms based on relative position of activities**
- **Comparison/Complementarity**
- **Alternative Resources**

Propagation

Algorithms based on time windows

- **Time-tabling**
  - Interesting activities *A* are those for which

    $e_{min}(A) > s_{max}(A)$.



  - Between $s_{max}(A)$ and $e_{min}(A)$ we know A will be executed.

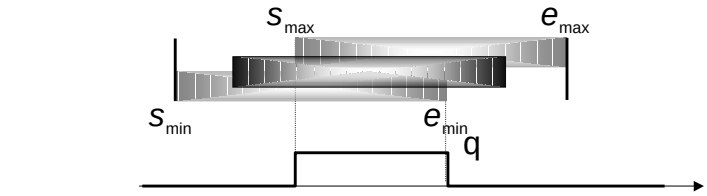
Propagation

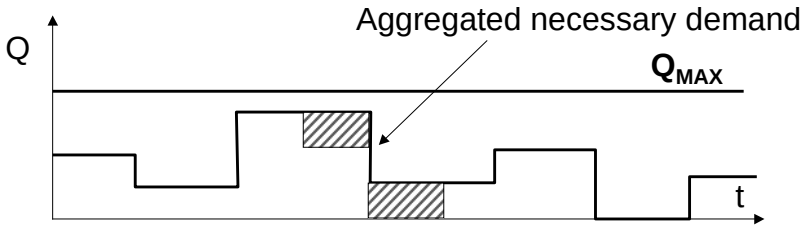
Algorithms based on time windows

- **Time-tabling**
  - All intervals where the demand is (partially) known are administered. Then for each such interval it is assured that :
    - Any activity that surely overlaps with the interval has a demand that is compatible with the already known demand in the interval (demands can be variable)
    - Any activity that has a non-compatible demand with the interval has no overlap with it.

Propagation

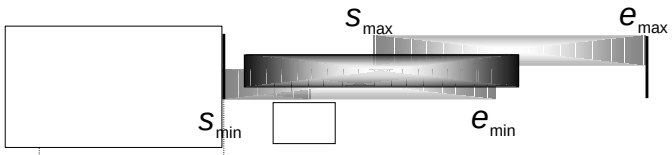
Algorithms based on time windows

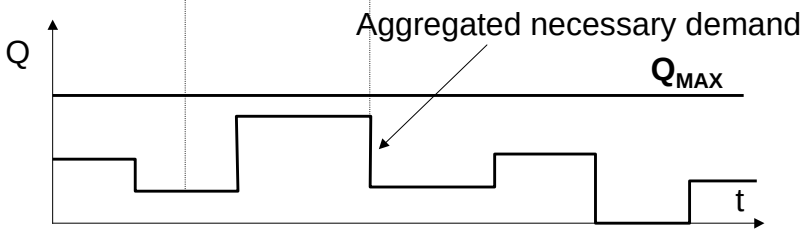




Propagation

Algorithms based on time windows





Propagation

Algorithms based on time windows

- **Time-tabling**

$C(\omega)$	$\Rightarrow$	$E(\omega)$
☑ Time Bounds	☑ Level	☑ Time Bounds
❑ Precedence	❑ Energy	❑ Precedence
		❑ Not First/Last

Propagation

Algorithms based on time windows

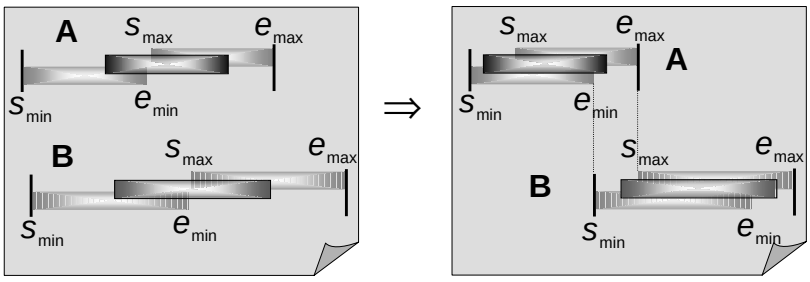
- **Time-tabling**
  - Used for large problems.
  - Worst case $O(n)$ algorithm.
  - Incremental.
  - Smart about when to propagate and when not, group propagation, etc.
  - In practice a very fast algorithm.

Algorithms based on time windows

- **Disjunctive constraint**
  - Considers pairs (A, B) of activities
  - If two activities can not overlap because of the capacity they require, and one, say A, can not be scheduled before the other, then
    - New time bounds are deduced
    - A new precedence constraint is deduced

Algorithms based on time windows

- **Disjunctive constraint on unary resource** [Erschler,76]



$s_{max}(A) < e_{min}(B)$

$e_{max}(A) \leq s_{max}(B)$

$e_{min}(A) \leq s_{min}(B)$

Propagation

Algorithms based on time windows

- **Disjunctive constraint**

$C(\omega)$	$\Rightarrow$	$E(\omega)$
☑ Time Bounds	☑ Level	☑ Time Bounds
❑ Precedence	❑ Energy	☑ Precedence
		❑ Not First/Last

Propagation

Algorithms based on time windows

- **Disjunctive Constraint**
  - Works on unary, discrete, and state resources.
  - More propagation than timetable constraint on unary resources.
  - Low memory cost
  - Needs fake activities to cope with resources not available at certain times
  - Algorithm : O(n)

Propagation

Algorithms based on time windows

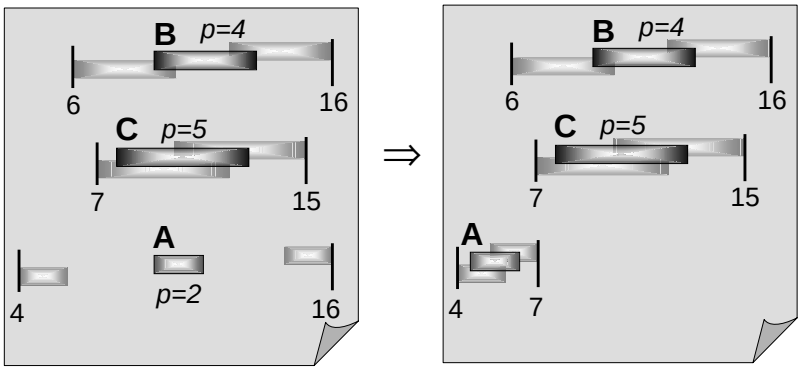
- **Edge-Finding**
  - [Carlier & Pinson,90],
  - [Nuijten,94],
  - [Baptiste & LePape,96]
- **Detecting First Among n Activities on Unary Resource:**

$$\forall \Omega, \forall A, [e_{max}(\Omega \cup \{A\}) - s_{min}(\Omega) < p(\Omega) + p(A)] \Rightarrow [A \prec \Omega]$$

Propagation

Algorithms based on time windows

- **Edge-Finding**



Propagation

Algorithms based on time windows

- **Similar reasoning for detecting Last, Not First, Not Last with respect to a set Ω**
- **Edge-Finding can be extended to discrete resources** [Nuijten,94], [Baptiste & LePape,96]

C(ω)	⇒	E(ω)
☑ Time Bounds	❑ Level	☑ Time Bounds
❑ Precedence	☑ Energy	☑ Precedence
		☑ Not First/Last

Propagation

Algorithms based on time windows

- **Side step: EF in CP**
  - One of the first global constraints used in CP.
  - Until I integrated (my own version of) this algorithm, the performance I had on “non-standard” applications was okay, but the performance on standard Job-Shop instances could be improved ✪
  - Defense was that the the Job-Shop Scheduling Problem doesn’t really exist (which is true in practice).

Algorithms based on time windows

- **Side step: EF in CP**
  - Found that it also boosted performance on the "non-standard" applications.
  - To use it in these applications, extensions and generalizations had to be designed (first step was for cumulative resources).
  - One of the first global constraints in a commercial CP package.
  - Recent survey shows ± 60 % of the scheduling customers of ILOG use (the generalized versions) of it.

Algorithms based on time windows

- **Energetic Reasoning** [Erschler & al, 91]
- **Discrete resource:** $\forall$ **{A,B}** $\subset$ **Acts,**

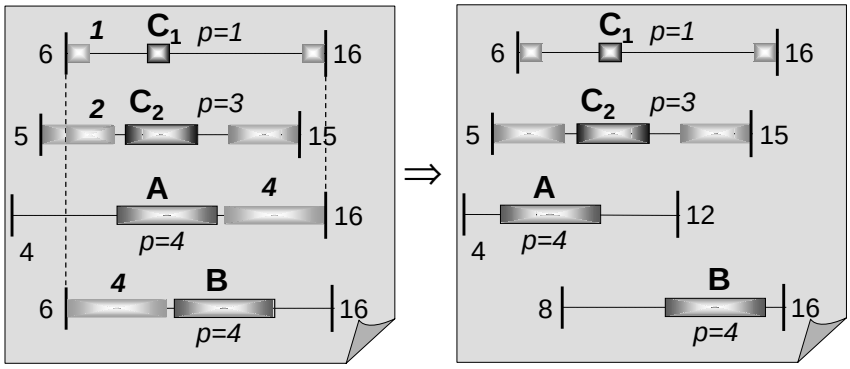
$$[Q_{MAX}.(s_{\min}(B) - e_{\max}(A)) < W(A) + W(B) + \sum_{C \notin \{A,B\}} W^{[s\min(B), e\max(A))}(C)]$$

$$\Rightarrow [s(A) \prec e(B)]$$

Propagation

Algorithms based on time windows

- Energetic Reasoning



Propagation

Algorithms based on time windows

- Energetic Reasoning

C(ω)	⇒	E(ω)
☑ Time Bounds	☐ Level	☑ Time Bounds
☐ Precedence	☑ Energy	☑ Precedence
		☑ Not First/Last

Algorithms based on time windows

- **More Propagation based on time windows:**
  - Preemptive resource constraints: mixed edge-finding [Le Pape & Baptiste 98]
  - Activity cannot be n^{th} [Nuijten & Le Pape 98]
  - Energetic reasoning [Baptiste, Le Pape & Nuijten 99]
  - Multi-machine propagation [Sourd & Nuijten 00]
  - ...

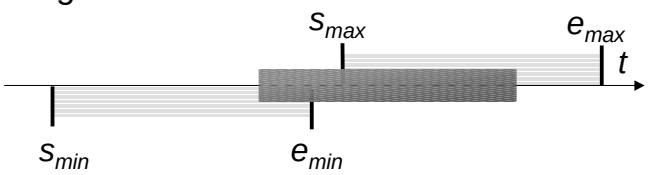
Overview

- **Global Constraints**
- **Algorithms based on activity time windows**
- **Algorithms based on relative position of activities**
- **Comparison/Complementarity**
- **Alternative Resources**

Propagation

Algorithms based on relative positions

- **Algorithms based on analyzing time bounds of activities propagate few (or even nothing) when the problem is not tight.**
- **Example:** On discrete resources, for the time-table constraint, if for all activity of the schedule: $e_{max} - s_{min} \geq 2.p_{min}$, the time-table constraint propagates nothing



Propagation

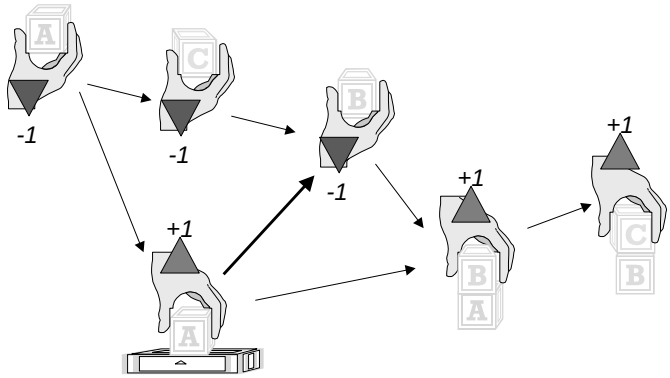
Algorithms based on relative positions

- **These propagation algorithms do not directly take into account precedence relations (but only through their effect on time bounds)**
- **In PO Planning:**
  - Time bounds of operators are loose
  - Extensive usage of precedence relations between operators in the partial plan
- ⇨ **Traditional resource propagation algorithms do not prune the search space enough**

Propagation

Algorithms based on relative positions

- **Example: Blocks world. Limited number of hands (2) modeled as a reservoir of capacity 2 and initial level 2.**



Propagation

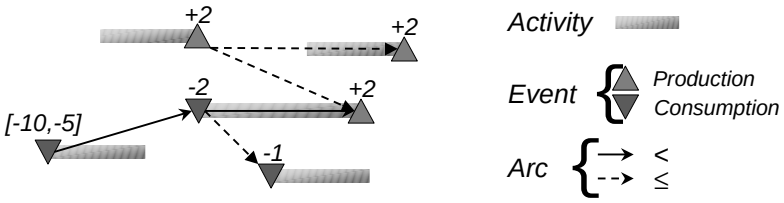
Algorithms based on relative positions

- **Analyze precedence relations on the left side of the propagation rule (C(ω)).**
- **Based on a Precedence Graph that collects and maintains precedence relations between activities:**
  - Initial statement of the problem
  - Precedence constraints inside a planning operator
  - Search decisions (e.g.: causal link, threat, ordering decision on resources)
  - Discovered by propagation algorithms (disjunctive constraint, edge-finding, energetic reasoning, $e_{max} \leq s_{min}$).

Propagation

Algorithms based on relative positions

- **Precedence Graph**
  - Events $x=(t_x, \Delta q_x)$ where the availability of a resource changes
  - Arcs : Two kind of arcs: $t_x<t_y$ and $t_x \leq t_y$

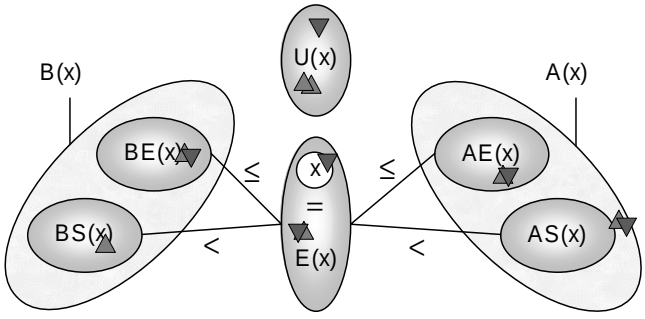


- **Transitive closure is incrementally maintained**

Propagation

Algorithms based on relative positions

- **Partition for an event x in a precedence graph**



- **Traversing a subset Θ(x) has a complexity in $|\Theta(x)|$**

Propagation

Algorithms based on relative positions

- **Energy Precedence Propagation** [Laborie,01]
  - Does not assume the resource to be closed
  - Local Property:

$$\forall X \in U, \forall \omega \subset B(X),$$
$$s(X) \geq \min_{Y \in \omega}(s(Y)) + \sum_{Y \in \omega}(q(Y).p(Y))/Q_{MAX}$$

  - In a partial schedule, we can decline this property to perform constraint propagation

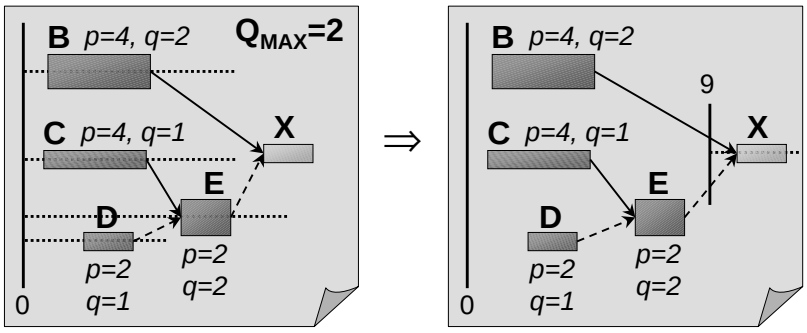
Propagation

Algorithms based on relative positions

- **Energy Precedence Propagation** [Unary,Discrete]

  Discrete resource: $\forall$ X $\in$ U,

$$[\omega \subset B(X)] \Rightarrow [s_{\min}(X) \geq \min_{Y \in \omega}(s_{\min}(Y)) + \sum_{Y \in \omega}(q_{\min}(Y).p_{\min}(Y))/Q_{MAX}]$$



Propagation

Algorithms based on relative positions

- **Energy Precedence Propagation**

C(ω)	⇒	E(ω)
❑ Time Bounds	❑ Level	☑ Time Bounds
☑ Precedence	☑ Energy	❑ Precedence
		❑ Not First/Last

Worst-case complexity: *O(n²)*

Propagation

Algorithms based on relative positions

- **Reservoir Balance Propagation** [Laborie,01]
  - Assumption: when applied to a reservoir, all producers of the reservoir are known in the current schedule
  - Local property: for a given event x in a solution schedule, we can compute the level of the reservoir just before x at date t(x)-ε as:

$$\Lambda_{-}(x) = L_{INIT} + \sum_{y \in BS(x)} q(y)$$

    A schedule is a solution on the reservoir iff

$$\forall x \in U,\ 0 \leq \Lambda_{-}(x) \leq Q_{MAX}$$

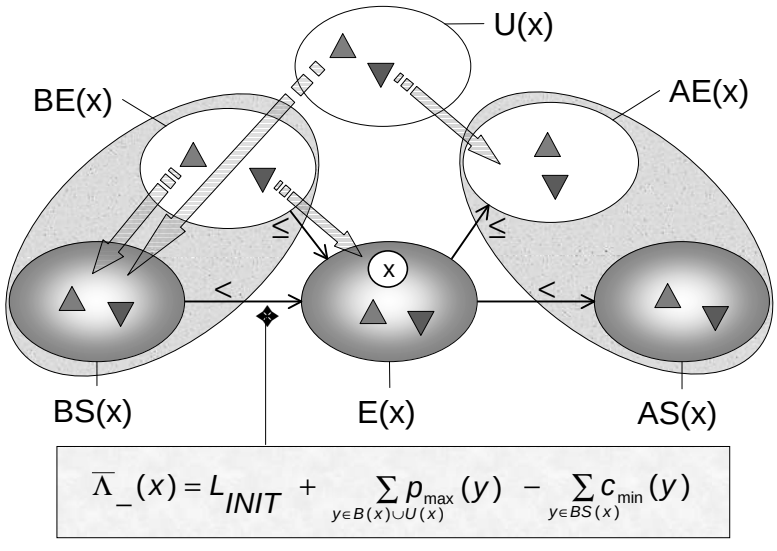
  - In a partial schedule, we can decline this property to perform constraint propagation

Propagation

Algorithms based on relative positions

- **Reservoir Balance Propagation**
  - Given an event *x*, using the graph, we can compute $\overline{\Lambda}_{-}(x)$ an upper bound on the reservoir level at date $t(x)-\varepsilon$ assuming:
    - All the production events *y* that **may** be executed strictly before *x* are executed strictly before *x* and produce their maximal quantity *q(y)*;
    - All the consumption events *y* that **need** to be executed strictly before *x* are executed strictly before *x* and consume their minimal quantity *q(y)*;
    - All the consumption events that **may** be executed simultaneously or after *x* are executed simultaneously or after *x*

Propagation



$$\overline{\Lambda}_{-}(x) = L_{INIT} + \sum_{y \in B(x) \cup U(x)} p_{\max}(y) - \sum_{y \in BS(x)} c_{\min}(y)$$

Propagation

Algorithms based on relative positions

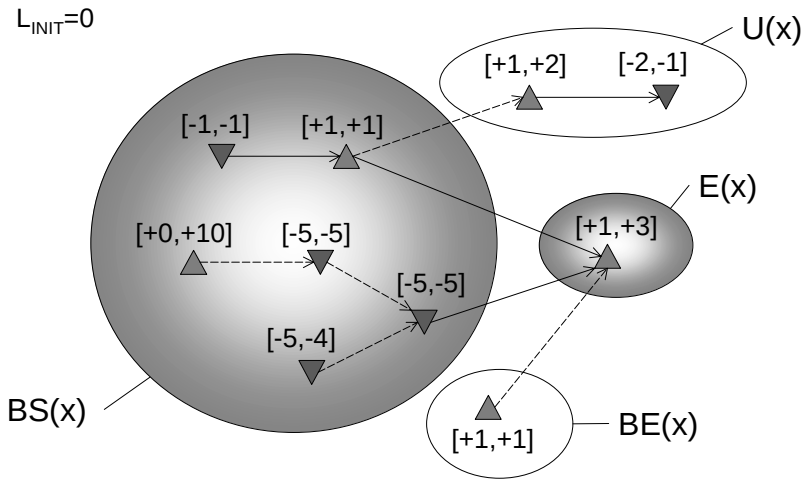
- **Reservoir Balance Propagation**
  - Similar bounds can be computed:
    - Lower bound on the reservoir level at date *t(x)-ε*
    - Lower/Upper bound on the reservoir level at date *t(x)+ε*
  - ✔ For symmetry reason, we focus on $\overline{\Lambda}_{-}(x)$
  - Given $\overline{\Lambda}_{-}(x)$, the reservoir balance algorithm discovers:
    - ✔ Failures
    - New bounds for required quantities of resources q(x)
    - ✔ New bounds for time variables t(x)
    - New precedence relations

Propagation

Algorithms based on relative positions

- **Reservoir Balance Propagation**
  - Discovering failures: As soon as: $\exists x, \overline{\Lambda}_{-}(x) < 0$, no solution exist that satisfy the precedence relations
  - Property: the rule $[\exists x, \overline{\Lambda}_{-}(x) < 0] \Rightarrow [fail]$ is sufficient to ensure the soundness of the search on a reservoir w.r.t. the reservoir underflow.
  - A symmetrical property exists for reservoir overflow based on $\underline{\Lambda}_{-}(x)$.

Propagation



$\Lambda_{-}(x)$= 0 + (2+1+10+1) - (1+5+5+4) = -1 < 0 !!!

Propagation

Algorithms based on relative positions

- **Reservoir Balance Propagation**
  - Safe event: An event *x* is said to be **safe** if and only if

$$\overline{\Lambda}_{-}(x) < Q_{MAX}, \quad \overline{\Lambda}_{+}(x) < Q_{MAX}$$
$$\underline{\Lambda}_{-}(x) \geq 0, \quad \underline{\Lambda}_{+}(x) \geq 0$$

  - Property: If all the events are **safe**, then any instantiation of the time variable that satisfies the precedence constraints also satisfies the reservoir constraint. In other words, the current partial order is safe and the reservoir is "**solved**".

Propagation

Algorithms based on relative positions

- **Reservoir Balance Propagation**
  - Reservoir balance equation:

$$\overline{\Lambda}_{-}(x) = L_{INIT} + \sum_{y \in BS(x)} q_{\max}(y) + \sum_{y \in BE(x) \cup U(x)} p_{\max}(y)$$

  - In case the first part of the equation is such that:

$$L_{INIT} + \sum_{y \in BS(x)} q_{\max}(y) < 0$$

  - It means that some events in $BE(x) \cup U(x)$ will have to be executed strictly before x in order to produce at least:

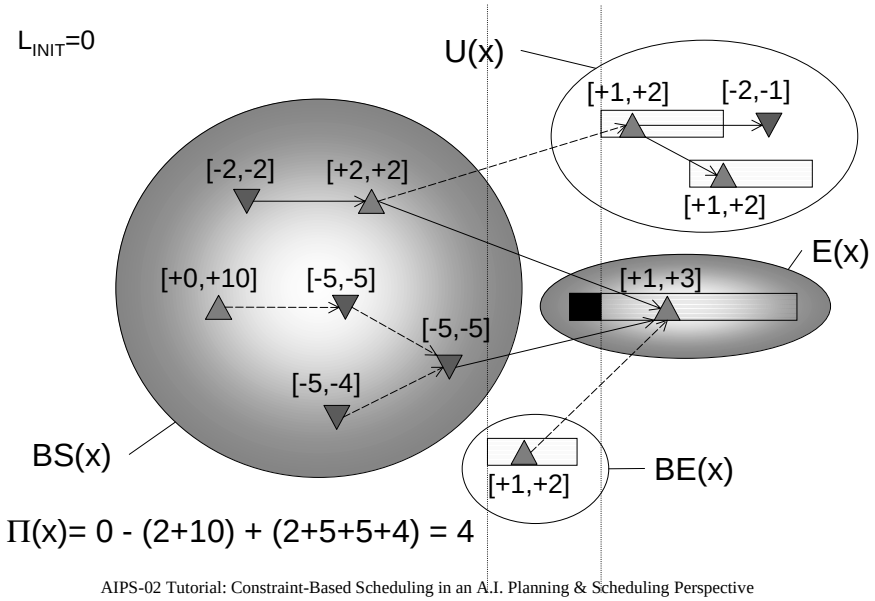
$$\Pi(x) = -L_{INIT} - \sum_{y \in BS(x)} q_{\max}(y)$$

Propagation

Algorithms based on relative positions

- **Reservoir Balance Propagation**
  - Some events $z \in BE(x) \cup U(x)$ will have to be executed strictly before x in order to produce at least $\Pi(x)$
  - Let $(z_1,...,z_i,...,z_n)$ the set of production events in $BE(x) \cup U(x)$ sorted by increasing minimal time
  - Let k be the smallest index such that: $\sum_{i=1}^{k} p_{\max}(z_i) \geq \Pi(x)$
  - Then: $\underline{t}(z_k) < t(x)$

Propagation

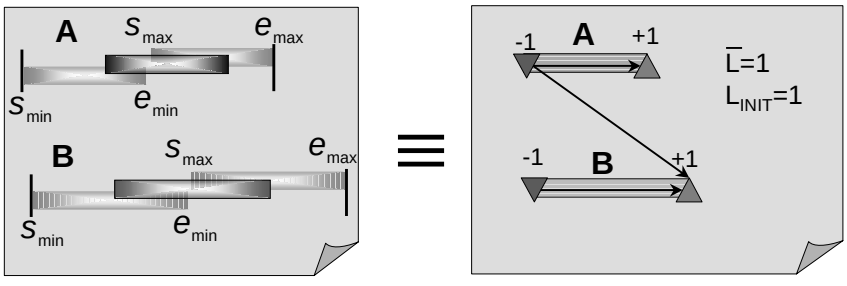


$\Pi(x)= 0 - (2+10) + (2+5+5+4) = 4$

Propagation

Algorithms based on relative positions

- **Reservoir Balance Propagation**
  - Property: On a unary resource, the balance propagation propagates the same as the disjunctive constraint



$s_{max}(A) < e_{min}(B)$

Propagation

Algorithms based on relative positions

- **Reservoir Balance Propagation**
  - In AI Planning applications, if some producers of the reservoir may still be inserted in the partial plan: transform the balance algorithm into a real truth criterion:
    - Some events $z \in BE(x) \cup U(x)$ will have to be executed strictly before x in order to produce at least $\Pi(x)$

      OR

    - Some operator containing a producing event z will have to be inserted such that t(z) < t(x)
  - Such a "truth criterion" is currently being studied

Propagation

Algorithms based on relative positions

- **Reservoir Balance Propagation**

$C(\omega)$	$\Rightarrow$	$E(\omega)$
☑ Time Bounds	☑ Level	☑ Time Bounds
☑ Precedence	☐ Energy	☑ Precedence
		☐ Not First/Last

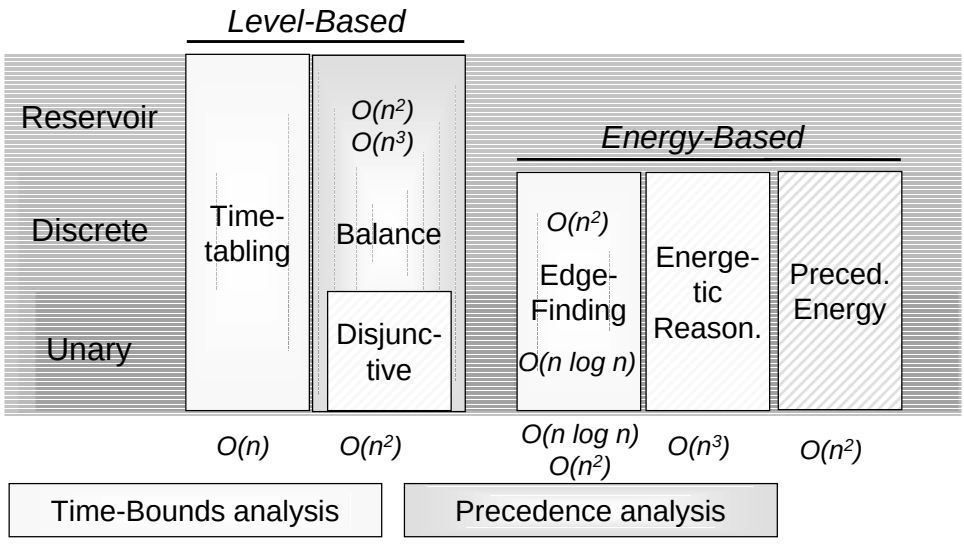
Worst-case complexity: $O(n^2) \rightarrow O(n^3)$

Propagation

Overview

- Global Constraints
- Algorithms based on activity time windows
- Algorithms based on relative position of activities
- Comparison/Complementarity
- Alternative Resources

Propagation



Overview

- Global Constraints
- Algorithms based on activity time windows
- Algorithms based on relative position of activities
- Comparison/Complementarity
- Alternative Resources

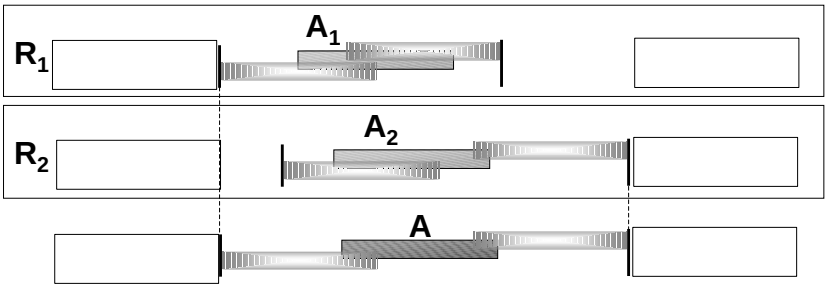
Alternative Resources

- Alternative Resource Set: $S=\{R_1,\ldots,R_n\}$
- U = [A requires(q) S]
- A variable r(U) is introduced to represent the [index of] the resource allocated to A
- Propagate as if A was split into n activities $\{A_1,\ldots,A_n\}$ on the alternative resources
- Alternative Resource Constraint maintains the constructive disjunction between A and $\{A_1,\ldots,A_n\}$

Propagation

Alternative Resources

- **Example**
  - Alternative Resource Set: S={R_1, R_2}
  - U = [A requires(q) S]



Overview

- **Basic Principles**
- **Propagation of Resource Constraints**
- **Search Techniques**
- **Putting things together on two examples**
- **Conclusion**

Search Techniques

Overview

- Branching Schemes
- Search Heuristics
- Beyond the Basic Search Scheme

Search Techniques

Overview

- Branching Schemes
- Search Heuristics
- Beyond the Basic Search Scheme

Search Techniques

Branching Schemes

- **Branching Schemes:** use branching in the search tree to enforce non-deterministic local properties
- **Example on a unary resource :**

  $\Omega = \{\{u,v\}, \{u,v\} \subset U\}$,

  $\forall \{u,v\} \in \Omega, [e(u) \leq s(v)] \vee [e(v) \leq s(u)]$
- **Branching Scheme:**
  - **Select a local property to enforce :**

    $\omega = \{u,v\} \in \Omega$
  - **Branch on :**

    $[e(u) \leq s(v)] \vee [e(v) \leq s(u)]$

Search Techniques

Branching Schemes

- **Classical Branching Schemes :**
  - Resource Assignment → Resource Allocation
  - Event Pair Ordering
  - Sequence
  - Rank First/Not First
  - Setting Times

  → Resource Scheduling

Search Techniques

Branching Schemes

- **Resource Assignment :**
  - Select an unbound variable r(U)
  - Select a value k in the current domain of r(U)
  - Branch on : $[r(U)=k] \vee [r(U)\neq k]$
- **More generally :**
  - Select an unbound variable r(U)
  - Select a subset K of the current domain D of r(U)
    $(K \neq \emptyset, K \neq D)$
  - Branch on : $[r(U)\in K] \vee [r(U)\notin K]$

Search Techniques

Branching Schemes

- **Property :** In a solution for a unary resource, all the pairs of resource usage are ordered.

- **Activity Pair Ordering (Unary Resource) :**
  - Select a not-ordered pair of resource usages : $(u,v) \in U$
  - Branch on : $[e(u) \leq s(v)] \vee [e(v) \leq s(u)]$
  - Needs a precedence graph to maintain the sets of not-ordered pairs of resource usage

Search Techniques

Branching Schemes

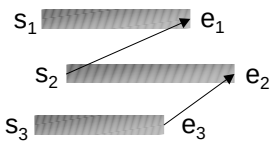
- **Property :** On a capacity resource, a partial schedule where all the events are safe is a solution.

- **Event Pair Ordering (Discrete, Reservoir, etc.) :**
  - Select a critical pair of events (x,y) on a resource R *(event = start or end time-point of an activity using R)*
  - Critical Pair of events : not-ordered pair of event : (x,y) such that neither x nor y is safe (in the sense of the balance constraint)
  - Branch on : $[t(x) < t(y)] \vee [t(x) \geq t(y)]$

Search Techniques

Branching Schemes

- **Property:** in a solution on a discrete resource, there is no Minimal Critical Sets.
- **Minimal Critical Set Ordering** [Laborie & Ghallab,95][Cesta & Oddi,01]
  - Select an MCS on a resource :



$q_1+q_2+q_3 > Q$
$q_1+q_2 \leq Q$
$q_1+q_3 \leq Q$
$q_2+q_3 \leq Q$

  - Branch on the resolvers of this MCS :

$[t(e_1) \leq t(s_3)] \vee [t(e_3) \leq t(s_1)] \vee [t(e_3) \leq t(s_2)]$

Search Techniques

Branching Schemes

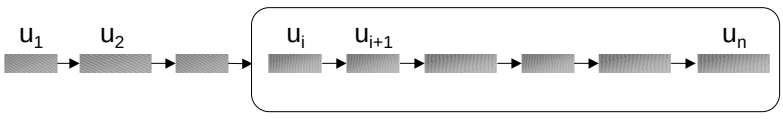
- **Property :** in any solution on a unary resource, each resource usage - but the last one - has a unique resource usage that is executed next to it.

- **Sequence (Unary Resource) :**
  - Select a not-sequenced pair of resource usage : (u,v)
  - Branch on : $[next(u) = v] \vee [next(u) \neq v]$
  - Needs a sequencing constraint to maintain and propagate the domain of next(x)
  - Useful when using transition costs

Search Techniques

Branching Schemes

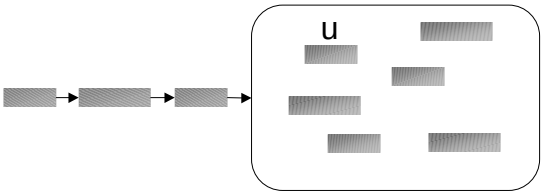
- **Property :** in a solution for a unary resource, there exists a chronological ordering of the resource usages $(u_1,u_2,\ldots,u_n)$ such that for all i, resource usage u_i is ranked first among the set of usages $\{u_i, u_{i+1}, \ldots, u_n\}$.



Search Techniques

Branching Schemes

- **Rank First/Not First (Unary Resource) :**
  - Select an unranked resource usage u



Unranked resource usages

  - Branch on : [rank first(u)] $\vee$ [rank not first(u)]
  - Needs constraint to propagate rank first/not first
- **Similar branching scheme with Rank Last/Not Last**

Search Techniques

Branching Schemes

- **All previous branching schemes are complete**
- **Setting Times (All resources) :**
  - Select an activity a in the schedule
  - Branch on : [s(a) = s_{min}(a)] $\vee$ [s(a) > s_{min}(a)]
  - Dominance rule relying on constraint propagation : instead of posting [s(a) > s_{min}(a)], activity a is marked as non-selectable until its start time has been updated by propagation.

Search Techniques

Overview

- Branching Schemes
- Search Heuristics
- Beyond the Basic Search Scheme

Search Techniques

Search Heuristics

- **Slack-Based Heuristics**
  - Choose a resource where the slack of any subset of potentially conflicting activities on that resource is minimal. Slack of a subset S is
    - $\max_{A \in S} (e_{max}(A)) - \min_{A \in S} (s_{min}(A)) - \text{sum}_{A \in S} p(A)$
    - Sequence that resource / Take a sequencing decision on that resource.
  - Choose pair of unsequenced activities with minimal maximal slack. Maximal slack of A and B is
    - $\max(e_{max}(A) - s_{min}(B),\ e_{max}(B) - s_{min}(A)) - p(A) - p(B)$.
    - Choose sequencing that leaves maximal slack.

Search Heuristics

- **Texture-based Heuristics**
  - Texture measurement: dynamic analysis of a search state to reveal problem structure
  - Heuristic: based on the revealed structure, find “most critical” part of the search state and take a decision to reduce criticality
  - The texture measurement and the heuristic to take decisions are separate.
  - Note: for efficiency reasons one might want to make more than one decision before recomputing texture measurements

Search Heuristics

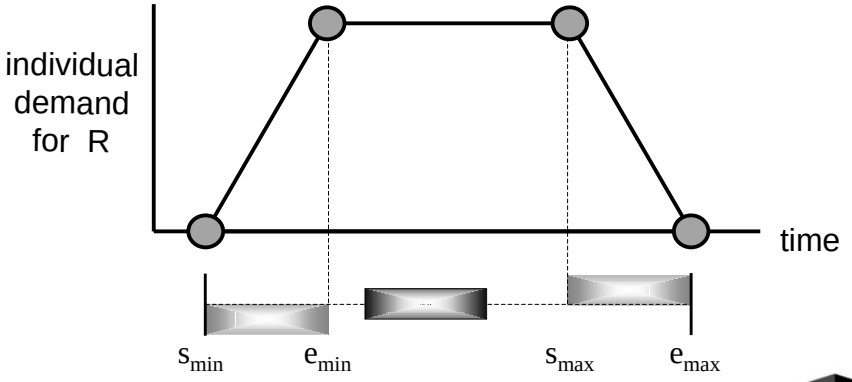
- **Example of Texture-based Heuristics :**

  The SumHeight Heuristic
  - **Texture measurement: a probabilistic estimate of the extent to which activities are competing for each resource and time point (“contention”)**
  - **Find the most critical (highest contention) resource and time point**
  - **Take a decision to reduce contention**

Search Techniques

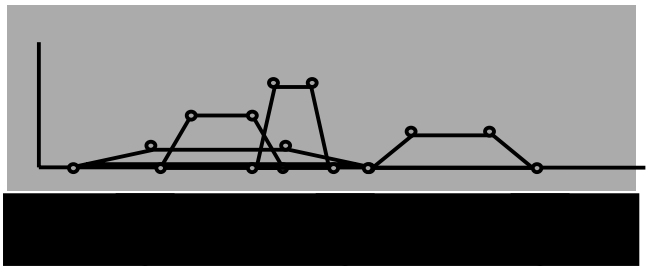
Search Heuristics

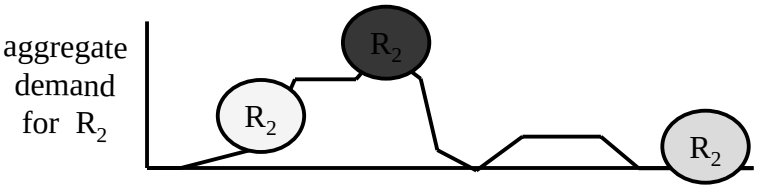
- **The SumHeight Texture Measurement:** Individual Demand



Search Techniques

Search Heuristics





Search Heuristics

- **The SumHeight Decision**
  - Find the resource, R*, and time point, t*, with highest contention
  - Find the 2 activities not sequenced with each other with the highest individual demand for R* at t*
  - Post a precedence constraint between them
  - Complexity:
    - R = # resources, n = # activities per resource
    - O(R.n.log(n)) for texture calculation
    - O(n^2) for finding decision after texture calculation

Overview

- **Branching Schemes**
- **Search Heuristics**
- **Beyond the Basic Search Scheme**

Beyond the Basic Search Scheme

- **Shaving.** Idea:
  - Perform some kind or forward checking that exploit the interactions between resources
  - Use some global propagation algorithms to determine if there exist a schedule that verify a property *p*.
    - E.g. Try posting A ⟨⟨ B on a unary resource ...
  - If no solution schedule verify *p* then ¬*p* must be verified in any solution. Thus it can be inferred.
    - … if global propagation algorithms fail, then infer the opposite constraint B ⟨⟨ A

Beyond the Basic Search Scheme

- **Shaving.** Example of application: Time-Bound Shaving
  - Activity A, Select α such that $e_{min}(A) \leq \alpha \leq e_{max}(A)$
    - If propagating $e(A) < \alpha$ leads to a dead-end: infer $\alpha \leq e(A)$
    - If propagating $e(A) > \alpha$ leads to a dead-end: infer $\alpha \geq e(A)$
  - Perform a dichotomic search on $[e_{min}(A), e_{max}(A)]$ to compute the best adjustments
- **Other shavings:** Precedence, Ranked First/Last activities

Beyond the Basic Search Scheme

- **Shaving. Example (Tested on ILOG Scheduler):**
  - Job-Shop problem **`mt10`** (10×10)
  - Strong constraint propagation (disjunctive, edge-finder, precedence energy)
  - Strong Shaving (time-bounds, precedence, rank first/last). Shave until a fix point is reached.
  - Optimality proof requires:
    - 2 choice points !!!!
    - About 2mn CPU time

Beyond the Basic Search Scheme

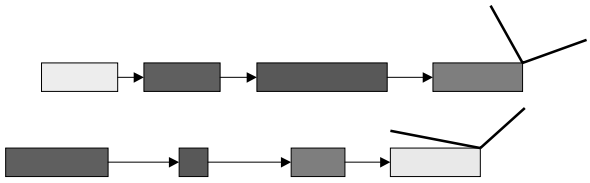
- **Dichotomizing**
  - An optimization problem P, an objective function to minimize f
  - Compute initial bounds of the objective: $f_{min} < f^* \leq f_{max}$
  - ① If $f_{max} = f_{min} + 1$: Optimal solution is $f^* = f_{max}$
  - ② Let $f_{middle} = (f_{min} + f_{max})/2$
  - ③ Try Solving the problem with the constraint $f \leq f_{middle}$
  - ④ If a solution exists : Let $f_{max} = f_{middle}$, goto ①
  - ⑤ If no solution exist : Let $f_{min} = f_{middle}$, goto ①

Beyond the Basic Search Scheme

- **Probing**
  - Main idea: At each node of the search, solve a relaxed version of the current schedule and use this solution to guide the search
    - Relaxed problem: for example only keep temporal constraints [El Sakkout-00] or cost relevant activities [Beck-01]
    - If the solution to the relaxed problem cannot be extended to a solution, identify the discrepancies and branch on the different alternatives to solve these discrepancies.
  - If the relaxed problem can be solved to optimality, it provides a lower bound on the objective function

Beyond the Basic Search Scheme

- **Probing**
  - Example: Job-shop with Earliness/Tardiness Costs of last activity



Beyond the Basic Search Scheme

- During the search, maintain at each node the optimal solution of a linear relaxation of the problem (no resource constraints):

Minimize $\sum_i y_i$ subject to:

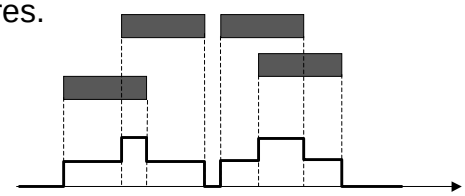
Cost:
$$y_i \geq tc_i \cdot ((x_{i,m} + pt_{i,m}) - dd_i)$$
$$y_i \geq ec_i \cdot (dd_i - (x_{i,m} + pt_{i.m}))$$

Time windows: $s_{\min}(a_{i,j}) \leq x_{i,j} \leq s_{\max}(a_{i,j})$

Precedence: $x_{i,j} + pt_{i,j} \leq x_{i,j+1}$

Beyond the Basic Search Scheme

- Using the relaxed optimal start time (from LP), build textures.



- Identify a peak (where the texture > capacity).
- Select a pair of activities, A and B, at the peak.
- Branch: (A $\rightarrow$ B) $\vee$ (B $\rightarrow$ A)
- New precedence constraints (as well as precedence discovered by constraint propagation) are used to update the LP relaxation

Beyond the Basic Search Scheme

- **Large Neighborhood Search**
  - Idea: From an initial solution and a set of neighboring solutions, create a search tree to find best neighbor while minimizing variable assignments.
  - CP search 'completes' solution

Beyond the Basic Search Scheme

- **Neighborhood exploration**
  - **Current solution ABCDEF=101011**
  - **Leaf nodes:**
    - 001011
    - 111011
    - 100011
    - 101111
    - 101001
    - 101010

Search Techniques

Beyond the Basic Search Scheme

- **Neighborhood exploration**
  - Naïve
    - Test neighboring solution against constraints
    - O(v) instantiations per neighbor
  - [Pesant and Gendreau, 96]
    - Neighborhood exploration by tree search
    - O(v) instantiations per neighbor
  - [Shaw et al., 2000]
    - O(log v) instantiations per neighbor

Search Techniques

Beyond the Basic Search Scheme

- **[Pesant and Gendreau, 96]**
  - Add neighborhood model to basic model
    - Variables V represent neighborhood
    - Interface constraints connect V variables to basic model
  - Interface constraints added anew before each exploration
    - Parameterized by current solution s
  - Explore neighborhood by instantiating V variables
    - Each leaf node a neighbor of s
    - Branch & Bound finds best neighbor

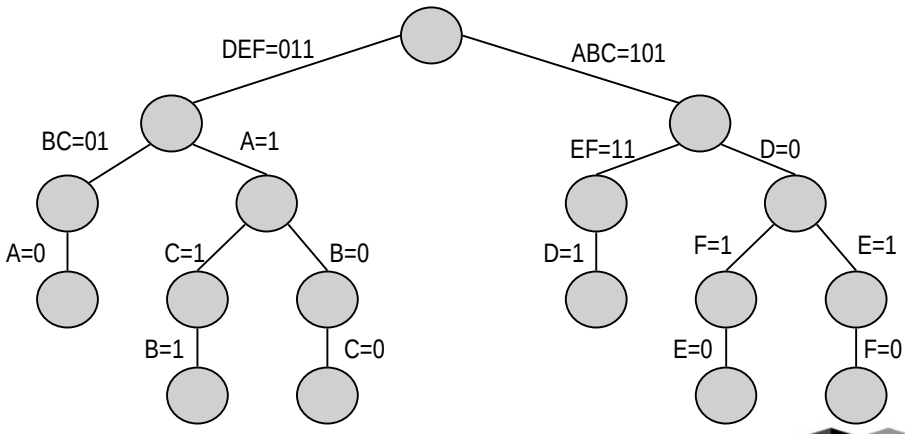
Beyond the Basic Search Scheme

- **[Shaw et al., 2000]**
  - **Current solution ABCDEF=101011**
  - **Leaf nodes:**
    - 001011
    - 111011
    - 100011
    - 101111
    - 101001
    - 101010

Beyond the Basic Search Scheme

[Shaw et al., 2000]

Starting assignment ABCDEF=101001



Overview

- Basic Principles
- Propagation of Resource Constraints
- Search Techniques
- Putting things together on two examples
- Conclusion

Putting things together

Overview

- Job-Shop Scheduling Problem (JSSP)
- RCPSPI with time lags

Overview

- **Job-Shop Scheduling Problem (JSSP)**
- **RCPSPI with time lags**

Job-Shop Scheduling Problem

- **Propagation algorithms:**
  - Precedence constraints
  - Disjunctive constraint on each unary resource
  - Edge-Finding on each unary resource
  - Multi-machine propagation
- **Branching Schemes:**
  - Schedule an activity to be first or last on a resource (among the set of unscheduled activities).
  - Sequence a pair of activities on a resource (A will be before B).

Putting things together

Job-Shop Scheduling Problem

- **Heuristics:**
  - Select unscheduled resource with smallest slack.
  - If the number of "possible firsts" is smaller than the number of "possible lasts" on the resource, decide to schedule an activity first otherwise decide to schedule an activity last. Ties are broken by considering the difference between the two smallest s_{min}'s and the two largest e_{max}'s; the largest difference "wins".
  - If decision is taken to schedule an activity first, chose the activity that has smallest e_{max}, ties are broken by smallest s_{min}. Equivalent rule when scheduling an activity to be last.

Putting things together

Job-Shop Scheduling Problem

- **Heuristics:**
  - Many alternative heuristics exist and are being used.
  - For approximation in [Nuijten & Le Pape, 98], a large neighborhood search is used.
    - Randomly keep a set of ordering decisions of the previous best solution. Probability to keep an ordering is p.
    - Do maximally RS fail limited searches for a better solution (restarts).
    - If no improvement is found in these RS restarts, choose a new random set of ordering decisions.
    - If no improvement is found for n restarts, decrease probability p.
    - Stop if p reaches a given threshold.

Putting things together

Job-Shop Scheduling Problem

- **Results :** Good performance both for optimization and approximation.
- **Approximation:** near 0 MRE (mean relative error) on a thoroughly studied set of 13 JSSP instances.
- **Optimization:** standard 10 x 10, 15 x 10 instances comparable results to the best approaches.
- **Conclusions:**
  - Generality of approach causes little performance loss.
  - Expression of search procedures is powerful enough to express the right search algorithms.

Putting things together

Job-Shop Scheduling Problem

- New lower bounds for FSSP and JSSP instances [Sourd & Nuijten,00].

		LB	UB
rec09	20 x 10	**1520**	1520
rec13	20 x 15	**1836**	1930
yam1	20 x 20	**851**	888
yam3	20 x 20	**842**	893
yam4	20 x 20	**922**	968

- Remark: best known results for preemptive job-shop scheduling [Le Pape & Baptiste, 99].

Putting things together

Overview

- Job-Shop Problem
- RCPSPI with time lags

Putting things together

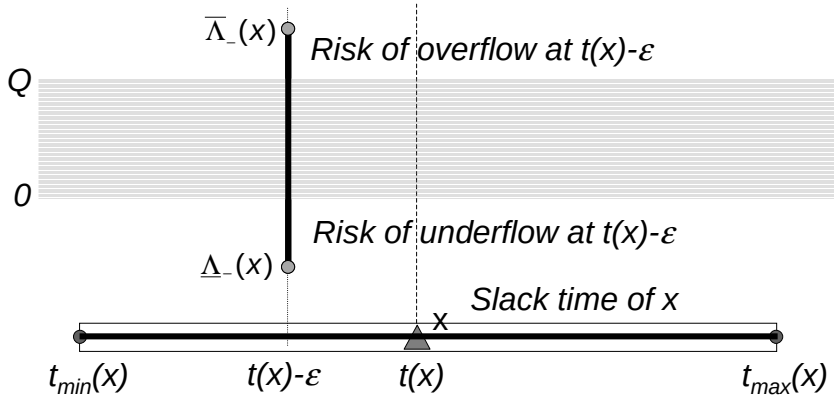
RCPSPI with time lags

- **Propagation algorithms :**
  - Time-table constraint on each reservoir
  - Balance constraint on each reservoir
- **Branching Scheme :**
  - Event Pair Ordering
- **Heuristics :**
  - Based on LB and UB on levels computed by the balance constraint

Putting things together

RCPSPI with time lags

- **Heuristics :**



Putting things together

RCPSPI with time lags

- **Heuristics :**
  - Select an unsafe event x that maximizes

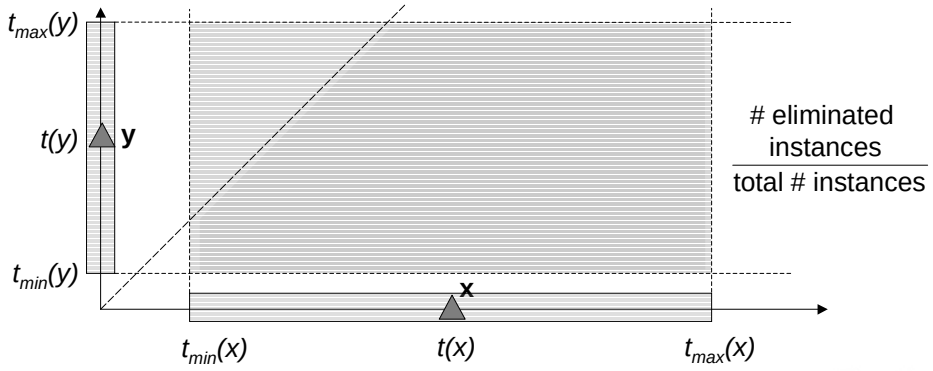
$$crit(x) = \frac{f(risks\ of\ under\ /\ overflow(x))}{slack\ time(x)}$$

  - Selection of an event y unranked w.r.t. x based on temporal commitment of posting t(x)<t(y) and t(x)≥t(y)

Putting things together

RCPSPI with time lags

- **Heuristics :**
  - temporal commitment of posting t(x) < t(y):



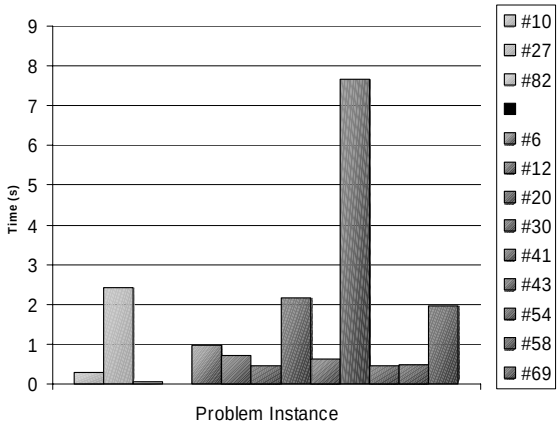
Putting things together

RCPSPI with time lags

- **Search: order pairs (x,y) until all events are safe**
- **Results** [Laborie,01]:
  - Tested on 12 open RCPSPI with time lags [Neumann & Schwindt,99]
  - All problems were closed in less than 10s CPU time (HP-UX 9000/785 workstation)
  - Produce partially ordered schedule instead of fully instantiated ones (more robust).

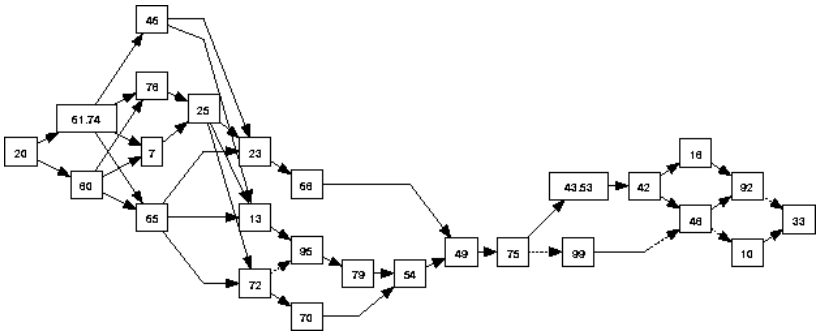
Putting things together

RCPSPI with time lags



Putting things together

RCPSPI with time lags



A part of an optimal solution for #41

Overview

- Basic Principles
- Propagation of Resource Constraints
- Search Techniques
- Putting things together on two examples
- Conclusion

Conclusion

Constraint-Based Scheduling ...

- **Properties that contributed to the industrial success of Constraint-Based Scheduling techniques and tools:**
  - The paradigm clearly separates the constraints (semantics, propagation algorithms) from the search space exploration (branching schemes, heuristics)
  - Each constraint is an autonomous algorithm, the paradigm is extensible: new application-dependent constraints can easily be implemented

Conclusion

Constraint-Based Scheduling ...

- **Properties that contributed to the industrial success of Constraint-Based Scheduling techniques and tools:**
  - A natural and powerful modeling language exists.
  - Effective and efficient (global) propagation algorithms exist both for temporal constraints as for resource constraints.
  - The approach can be used in cooperation with other search techniques: Linear Programming, Local Search

Conclusion

… in an AI Planning and Scheduling Perspective

- **Constraint-Based Scheduling and Partial Order Planning have a lot in common :**
  - A representation of the current plan/schedule as a temporal graph
  - The notion of local property (known as Necessary Truth Criterion in POP, less formalized in CBS)
  - The idea of branching as a non-deterministic way of enforcing local properties
  - The idea of using problem relaxation to propagate constraints and/or to provide good heuristics

Conclusion

Lessons for CBS from POP

- Clear formalization of local properties that need to be enforced in any solution
- Provide necessary (and maybe sufficient) conditions for a partially ordered schedule to be a solution
- Least-commitment problem solving

Conclusion

Lessons for POP from CBS

- Constraint propagation is really a complexity killer: every domain that is pruned, every precedence constraint that is inferred has a huge impact on the size of the search space to be explored
- Efficient algorithms are available for propagating resource constraints and solving scheduling problems with complex resources
- Most of this work is fully compatible with POP
- Developing efficient propagation algorithms on planning predicates should deserve as much attention as designing efficient search heuristics