ICAPS 2006

The English Lake District, Cumbria, UK

Tutorial on Filtering Techniques in Planning and Scheduling

Roman Bartak

Charles University, Czech Republic

TU2

icaps

ICAPS 2006

The English Lake District, Cumbria, UK

TU2

Tutorial on Filtering Techniques in Planning and Scheduling

Roman Bartak

Charles University, Czech Republic

ICAPS 2006
Tutorial onFiltering Techniques in Planning and Scheduling

Table of contents

ICAPS 2006
Tutorial onFiltering Techniques in Planning and Scheduling

Preface

Constraint satisfaction technology plays an important role in solving real-life scheduling problems. As time and resources become more important in nowadays planning, the role of constraint satisfaction is increasing there too. One of the key aspects of constraint satisfaction is using constraints actively to remove infeasible values from domains of variables and consequently to prune the search space. This approach is called domain filtering. The goal of the tutorial is to explain in detail domain filtering techniques used in the context of constraint-based planning and scheduling.

The tutorial is targeted to researchers and practitioners that would like to use constraint satisfaction technology efficiently in solving planning and scheduling problems. Basics of constraint satisfaction and in particular constraint propagation will be explained so no prior knowledge of constraint satisfaction is required. The focus of the tutorial is on existing filtering algorithms for constraints used in modeling time and resource restrictions. The filtering techniques for temporal and resource constraints, namely point and interval algebras, temporal networks, edge-finding, not-first/not-last, and energetic reasoning, will be explained. The details on filtering combing qualitative and quantitative approaches as well as using the objective function will be given. Finally, the implementation of new filtering algorithms will also be explained. The audience will take away a basic understanding of how constraint propagation works with more details on filtering techniques for planning and scheduling constraints.

Instructor

- *Roman Barták*
  Charles University in Prague, Czech Republic

Filtering Techniques in Planning and Scheduling

Roman Barták
Charles University, Prague (CZ)

roman.bartak@mff.cuni.cz

Tutorial outline

- **Introduction**
  - What is domain filtering?
  - What is the context in which domain filtering operates?
- **Domain filtering for P&S**
  - **Temporal (only) reasoning**
    - qualitative approach
    - quantitative approach
  - **Resource reasoning**
    - absolute positioning in time
    - relative positioning in time
    - combining absolute and relative positioning
  - **Reasoning on objective functions**
    - late activities, setup times
- **Conclusions**
  - design of own filtering algorithms

Filtering Techniques in Planning and Scheduling 2

Domain filtering
within constraint satisfaction

Sudoku?

- **Logic-based puzzle,** whose goal is to enter digits 1-9 in cells of 9×9 table in such a way, that no digit appears twice or more in every row, column, and 3×3 sub-grid.

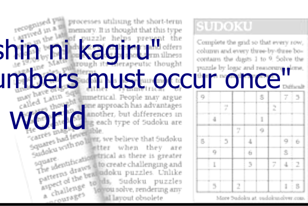
9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

A bit of history

1979: first published in New York
under the name „Number Place"

1986: became popular in Japan
Sudoku – from Japanes "**Su**dji wa **doku**shin ni kagiru"
"the numbers must be single" or "the numbers must occur once"

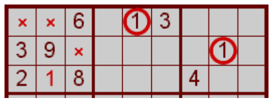
2005: became popular in the western world



Filtering Techniques in Planning and Scheduling

Solving Sudoku

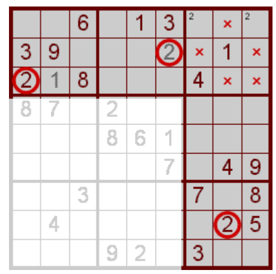
How to find out which digit to fill in?



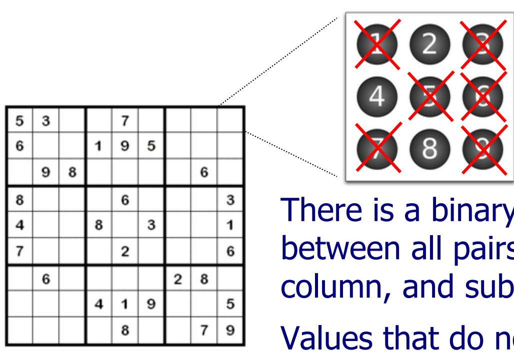
- Use information that each digit appears exactly once in each row and column.

What if it is not enough?

- If rows and columns do not provide enough information then annotate each cell with possible digits that can be filled there.



Sudoku in general



We can see every cell as a **variable** with possible values from **domain** {1,…,9}.

There is a binary inequality **constraint** between all pairs of variables in every row, column, and sub-grid.

Values that do not satisfy any constraint are **pruned** from the domain.

Such formulation of the problem is called a **constraint satisfaction problem.**

Pruning of values – **domain filtering** – is repeated until there is no value to be pruned.

Constraint technology

based on **declarative problem description** via:

- **variables with domains** (sets of possible values) e.g. start of activity with time windows
- **constraints** restricting combinations of variables e.g. end(A) < start(B)

A **feasible solution** to a constraint satisfaction problem is a complete assignment of variables satisfying all the constraints.

An **optimal solution** to a CSP is a feasible solution minimizing/maximizing a given objective function.

Domain filtering

- **Example:**
  - $D_a=\{1,2\}$, $D_b=\{1,2,3\}$
  - $a<b$
  - ↳ Value 1 can be safely removed from D_b.
- Constraints are used **actively to remove inconsistencies** from the problem.
  - inconsistency = value that cannot be in any solution
- This is realized via a procedure FILTER that is attached to each constraint.

Arc-consistency

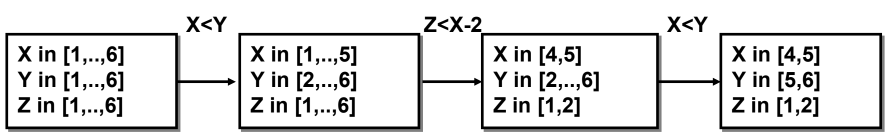
- We say that a constraint is **arc consistent** (AC) if for any value of the variable in the constraint there exists a value for the other variable(s) in such a way that the constraint is satisfied (we say that the value is supported).
  Unsupported values are filtered out of the domain.

- A **CSP is arc consistent** if all the constraints are arc consistent.

Making problems AC

- How to establish arc consistency in CSP?
- Every constraint must be filtered!

Example: X in [1,..,6], Y in [1,..,6], Z in [1,..,6], X<Y, Z<X-2



↳ Filtering every constraint just once is not enough!

- Filtering must be repeated until any domain is changed (AC-1).

Mackworth (1977)

Algorithm AC-3

- Uses a **queue of constraints** that should be filtered.
- When a domain of variable is changed, only the constraints over this variable are added back to the queue for filtering.

```
procedure AC-3(V,D,C)
    Q ← C
    while non-empty Q do
        select c from Q
        D' ← c.FILTER(D)
        if any domain in D' is empty then return (fail,D')
        Q ← Q ∪ {c'∈C | ∃x∈var(c') D'_x≠D_x} – {c}
        D ← D'
    end while
    return (true,D)
end AC-3
```

Filtering Techniques in Planning and Scheduling 11

AC in practice

- Uses a **queue of variables** with changed domains.
  - Users may specify for each constraint when the filtering should be done depending on the domain change.
- The algorithm is sometimes called AC-8.

```
procedure AC-8(V,D,C)
    Q ← V
    while non-empty Q do
        select v from Q
        for c∈C such that v is constrained by c do
            D' ← c.FILTER(D)
            if any domain in D' is empty then return (fail,D')
            Q ← Q ∪ {u∈V | D'_u≠D_u}
            D ← D'
        end for
    end while
    return (true,D)
end AC-8
```

Filtering Techniques in Planning and Scheduling 12

Lhomme (1993)

Arc-B-consistency

- Sometimes, making the problem arc-consistent is costly (for example, when domains of variables are large).
- In such a case, a weaker form of arc-consistency might be useful.

- We say that a constraint is **arc-b-consistent** (bound consistent) if for any bound values of the variable in the constraint there exists a value for the other variable(s) in such a way that the constraint is satisfied.
  - a bound value is either a minimum or a maximum value in domain
  - domain of the variable can be represented as an interval
  - for some constraints (like A<B) it is equivalent to AC

Filtering Techniques in Planning and Scheduling 13

Pitfalls of AC

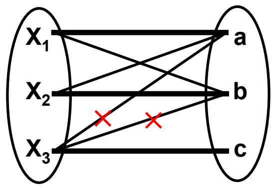
- **Disjunctive constraints**
  - A,B in 1..10, A=1 ∨ A=2
  - no filtering (whenever A≠1 then deduce A=2 and vice versa)
- **Detection of inconsistency**
  - A,B,C in 1..10000000, A<B, B<C, C<A
  - long filtering (4 seconds)
- **Weak filtering**
  - A,B in 1..2, C in 1..3, A\=B, A\=C, B\=C
  - weak filtering (it is arc-consistent)

Filtering Techniques in Planning and Scheduling 14

Régin (1994)

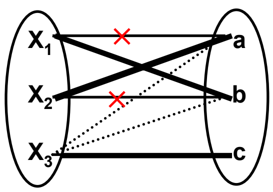
Inside all-different

- a set of binary inequality constraints among all variables $X_1 \neq X_2, X_1 \neq X_3, \ldots, X_{k-1} \neq X_k$
- all_different$(\{X_1,\ldots,X_k\}) = \{(d_1,\ldots,d_k) \mid \forall i\ d_i \in D_i\ \&\ \forall i \neq j\ d_i \neq d_j\}$
- better pruning based on matching theory over bipartite graphs



Initialization:

1. compute maximum matching
2. remove all edges that do not belong to any maximum matching



Propagation of deletions (X1≠a):

1. remove discharged edges
2. compute new maximum matching
3. remove all edges that do not belong to any maximum matching

Filtering Techniques in Planning and Scheduling 15

Prosser et al. (2000)

Meta consistency

Can we **strengthen any filtering technique**?

YES! Let us assign a value and make the rest of the problem consistent.

- **singleton consistency**
  - try each value in the domain
- **shaving**
  - try only the bound values
- **constructive disjunction**
  - propagate each constraint in disjunction separately
  - make a union of obtained restricted domains

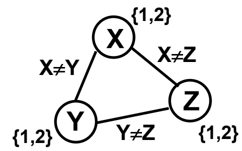
Filtering Techniques in Planning and Scheduling 16

Mackworth (1977)

Path consistency

Arc consistency does not detect all inconsistencies!

Let us look at several constraints together!



- The path $(V_0,V_1,\ldots, V_m)$ is **path consistent** iff for every pair of values $x \in D_0$ a $y \in D_m$ satisfying all the binary constraints on V_0,V_m there exists an assignment of variables $V_1,\ldots,V_{m-1}$ such that all the binary constraints between the neighboring variables V_i,V_{i+1} are satisfied.
- **CSP is path consistent** iff every path is consistent.

Some notes:

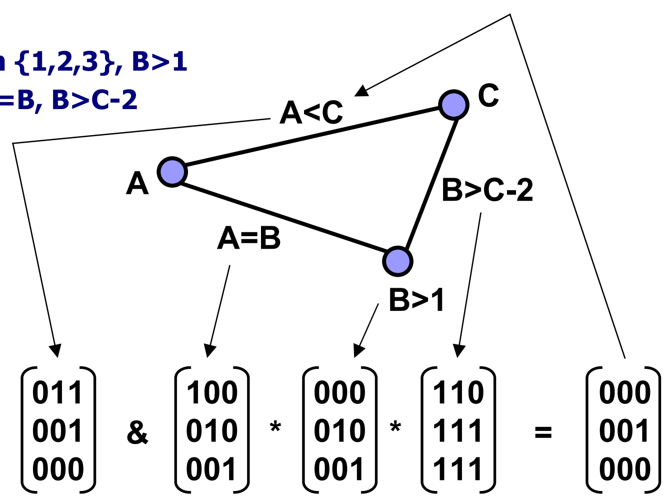
- only the **constraints between the neighboring variables** must be satisfied
- it is enough to explore **paths of length 2** (Montanary, 1974)

Path revision

Constraints represented extensionally via matrixes.
Path consistency is realized via matrix operations

Example:

- **A,B,C in {1,2,3}, B>1**
- **A<C, A=B, B>C-2**



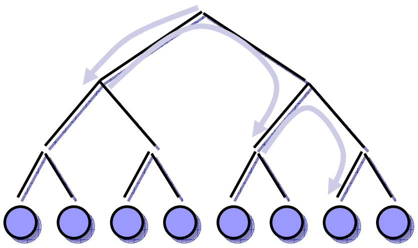
Search / Labeling

Filtering techniques are (usually) incomplete.

↳ We need a **search algorithm** to resolve the rest!

Labeling

- depth-first search
  - assign a value to the variable
  - propagate = make the problem locally consistent
  - backtrack upon failure



- X in 1..5 ≈ X=1 ∨ X=2 ∨ X=3 ∨ X=4 ∨ X=5 (**enumeration**)

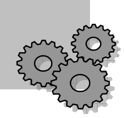
In general, search algorithm resolves remaining disjunctions!

- X=1 ∨ X≠1 (**step labeling**)
- X<3 ∨ X≥3 (**domain splitting**)
- X<Y ∨ X≥Y (variable ordering)

Labeling skeleton

- Search is combined with filtering techniques that prune the search space.
- **Look-ahead technique (MAC)**

```
procedure labeling(V,D,C)
    if all variables from V are assigned then return V
    select not-yet assigned variable x from V
    for each value v from D_x do
        (TestOK,D') ← consistent(V,D,C∪{x=v})
        if TestOK=true then R ← labeling(V,D',C)
        if R ≠ fail then return R
    end for
    return fail
end labeling
```

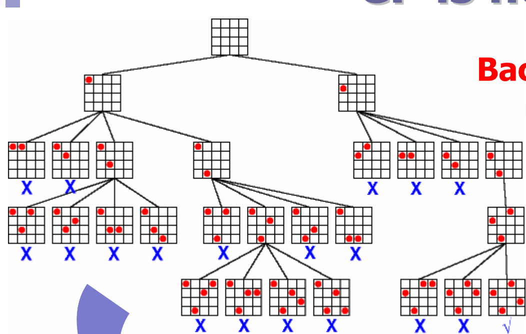


Branching schemes

- **Which variable should be assigned first?**
  - **fail-first principle**
    - prefer the variable whose instantiation will lead to a failure with the highest probability
    - variables with the smallest domain first
    - the most constrained variables first
  - defines the **shape of the search tree**
- **Which value should be tried first?**
  - **succeed-first principle**
    - prefer the values that might belong to the solution with the highest probability
    - values with more supports in other variables
    - usually problem dependent
  - defines the **order of branches** to be explored

CP is not (only) search!

4 queens problem

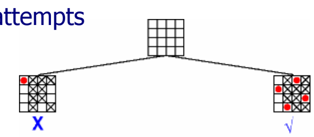


Backtracking is not very good

- 19 attempts

MAC combining search and arc consistency

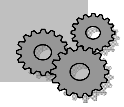
- 2 attempts



Constraint optimization

- **Constraint optimization problem** (COP) = CSP + objective function
- Objective function is encoded in a constraint v=obj(Xs) and the value of v is optimized

Branch and bound technique
find a complete assignment (defines a new bound)
store the assignment
update bound (post the constraint that restricts the objective function to be better than a given bound which causes failure)
continue in search (until total failure)
restore the best assignment



Final notes

- Combination of **depth-first search with arc consistency** is the **mainstream constraint satisfaction technology** used by current constraint solvers.
- However, there exist other constraint satisfaction techniques!
  - stronger consistency notions
    - path consistency, k-consistency, …
  - local search techniques
    - min-conflicts, tabu search, …
  - constraint inference techniques
    - constraints are combined until a solution is obtained

Temporal reasoning

Foundations

What is time?

The mathematical structure of time is generally a set with **transitive and asymmetric ordering operation**.

The set can be continuous (reals) or discrete (integers).

The planning/scheduling systems need to **maintain consistent information about time relations.**

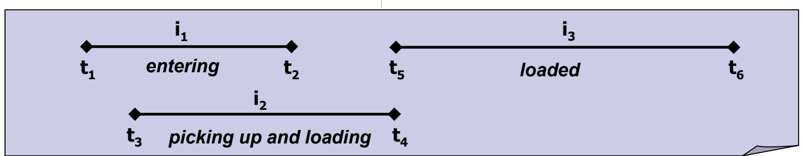
We can see time relations:

- **qualitatively**
  - relative ordering (A finished before B)
  - typical for modeling causal relations in planning
- **quantitatively**
  - absolute position in time (A started at time 0)
  - typical for modeling exact timing in scheduling

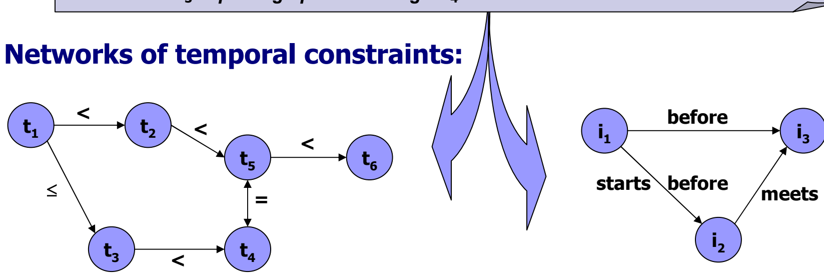
Filtering Techniques in Planning and Scheduling 26

Qualitative approach
example

- Robot starts entering a loading zone at time t_1 and stops there at time t_2.
- Crane starts picking up a container at t_3 and finishes putting it down at t_4.
- At t_5 the container is loaded onto the robot and stays there until time t_6.



Networks of temporal constraints:



Qualitative approach
formally

When **modeling time** we are interested in:

- **temporal references**
  (when something happened or hold)
  - **time points** (instants) when a state is changed
    instant is a variable over the real numbers
  - **time periods** (intervals) when some proposition is true
    interval is a pair of variables (x,y) over the real numbers, such that x<y
- **temporal relations** between temporal references
  - **ordering** of temporal references

Vilain & Kautz (1986)

Point algebra
principles

symbolic calculus modeling relations between instants
without necessarily ordering them or allocating to exact times

There are three possible relations between instants t_1 and t_2:

- $[t_1 < t_2]$
- $[t_1 > t_2]$
- $[t_1 = t_2]$

The relations P = {<,=,>} are called **primitives**.

- A set of primitives, meaning a disjunction of primitives, can describe any (even incomplete) relation between instants:
  - {}, {<}, {=}, {>}, {<,=}, {>,=}, {<,>}, {<,=,>}
    - {} means failure
    - {<,=,>} means that no ordering information is available
- **[t r t‘]** denotes the **relation r** between instants t and t‘

Vilain & Kautz (1986)

Point algebra
operations

- Let R be the set of all possible relations between instants
  - {{}, {<}, {=}, {>}, {<,=}, {>,=}, {<,>}, {<,=,>}}
- Useful operations on R:
  - **set operations** $\cap$, $\cup$
    - describe conjunction ($\cap$) or disjunction ($\cup$) of relations
  - **composition operation •**
    - deduce a new relation based on existing relations (an implied relation)
    - $[t_1\ r\ t_2]$ and $[t_2\ q\ t_3]$ gives $[t_1\ r \bullet q\ t_3]$ using the table

•	<	=	>
<	<	<	P
=	<	=	>
>	P	>	>

- The most useful operations are $\cap$ and • that allow a combination of the existing relation with the implied relation (a filtering rule):
  - $[t_1\ r\ t_2]$ and $[t_1\ q\ t_3]$ and $[t_3\ s\ t_2]$ gives $[t_1\ r \cap (q \bullet s)\ t_2]$

Vilain & Kautz (1986)

Point algebra
consistency

- A **PA (Point Algebra) network** is a directed graph (X,C) where X is a set of instants and each arc (i,j) is labeled by a constraint $r_{i,j} \in R$.
  - If some relation (i,j) is not explicitly mentioned in C then we assume the universal relation P there.
- The PA network (X,C) is **consistent** when it is possible to assign a real number to each instant in such a way that all the relations between instants are satisfied.

Proposition:

A PA network (X,C) is consistent iff there is a set of primitives $p_{i,j} \in r_{i,j}$, such that every triple of primitives verifies $p_{i,j} \in p_{i,k} \bullet p_{k,j}$.

Notes:

To make the PA network consistent it is enough to make its transitive closure, for example using techniques of path consistency.

- The network is inconsistent iff we get the relation {}.
- Path consistency does not produce a minimal network (there could remain some primitives that are not satisfied in any solution).

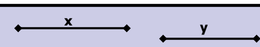
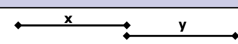
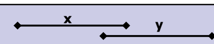
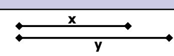
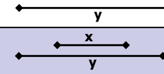
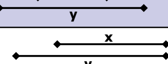
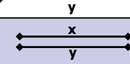
Allen (1983)

Interval algebra
principles

symbolic calculus modeling relations between intervals

(interval is defined by a pair of instants i^- and i^+, $[i^- < i^+]$)

There are thirteen primitives:

x **b**efore y	$x^+ < y^-$	
x **m**eets y	$x^+ = y^-$	
x **o**verlaps y	$x^- < y^- < x^+$ & $x^+ < y^+$	
x **s**tarts y	$x^- = y^-$ & $x^+ < y^+$	
x **d**uring y	$y^- < x^-$ & $x^+ < y^+$	
x **f**inishes y	$y^- < x^-$ & $x^+ = y^+$	
x **e**quals y	$x^- = y^-$ & $x^+ = y^+$	
b′, m′, o′, s′, d′, f′	symmetrical relations	

Allen (1983)

Interval algebra
operations and consistency

- Primitives can again be combined into sets/disjunctions (2^{13} sets).
  - sometimes, only some sets are used in a particular application
    - Example: {b,m,b',m'} can model non-overlapping intervals (unary resource)
  - set operations ∩, ∪ and composition operation •
- An **IA (Interval Algebra) network** (X,C) is **consistent** when it is possible to assign real numbers to x_i^-, x_i^+ of each interval x_i in such a way that all the relations between intervals are satisfied.

Proposition:

An IA network (X,C) is consistent iff there is a set of primitives $p_{i,j} \in r_{i,j}$, such that every triple of primitives verifies $p_{i,j} \in p_{i,k} \bullet p_{k,j}$.

Notes:

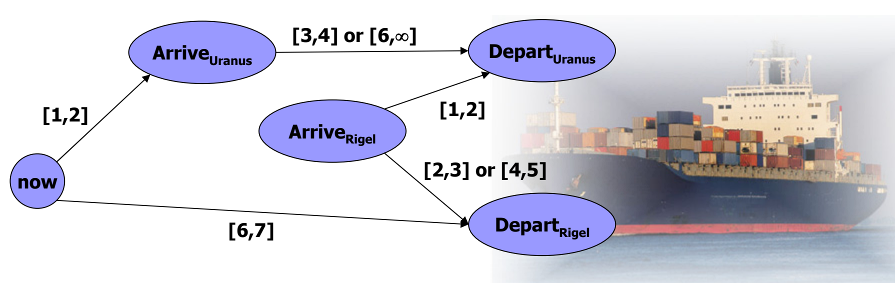
- Consistency-checking problem for IA networks is an NP-complete problem.
- Path consistency is not a complete technique for IA networks but it can detect some inconsistencies.
- Intervals can be converted to instants but with possibly non-binary constraints between instants.

Filtering Techniques in Planning and Scheduling 33

Ghallab et al. (2004)

Qualitative approach
example

- Two ships, Uranus and Rigel, are directing towards a dock.
- The Uranus arrival is expected within one or two days.
- Uranus will leave either with a light cargo (then it must stay in the dock for three to four days) or with a full load (then it must stay in the dock at least six days).
- Rigel can be serviced either on an express dock (then it will stay there for two to three days) or on a normal dock (then it must stay in the dock for four to five days).
- Uranus has to depart one to two days after the arrival of Rigel.
- Rigel has to depart six to seven days from now.



Filtering Techniques in Planning and Scheduling 34

Qualitative approach

formally

- The basic temporal primitives are again **time points**, but now the relations are numerical.
- Simple **temporal constraints** for instants t_i and t_j:
  - unary: $a_i \leq t_i \leq b_i$
  - binary: $a_{ij} \leq t_i - t_j \leq b_{ij}$,

  where a_i, b_i, a_{ij}, b_{ij} are (real) constants

Notes:

- Unary relation can be converted to a binary one, if we use some fix origin reference point t_0.
- $[a_{ij},b_{ij}]$ denotes a constraint between instants t_i a t_j.
- It is possible to use disjunction of simple temporal constraints.

Dechter et al. (1991)

STN

Simple Temporal Network (STN)

- only simple temporal constraints $r_{ij}= [a_{ij},b_{ij}]$ are used
- **operations**:
  - composition: $r_{ij} \bullet r_{jk} = [a_{ij}+a_{jk}, b_{ij}+b_{jk}]$
  - intersection: $r_{ij} \cap r'_{ij} = [\max\{a_{ij},a'_{ij}\}, \min\{b_{ij},b'_{ij}\}]$
- **STN** is **consistent** if there is an assignment of values to instants satisfying all the temporal constraints.
- **Path consistency** is a complete technique making STN consistent (all inconsistent values are filtered out, one iteration is enough). Another option is using all-pairs minimal distance **Floyd-Warshall algorithm**.

Algorithms

- **Path consistency**
  - finds a transitive closure of binary relations r
  - one iteration is enough for STN (in general, it is iterated until any domain changes)
  - works incrementally

one iteration for STN

```
PC(X, C)
    for each k : 1 ≤ k ≤ n do
        for each pair i, j : 1 ≤ i < j ≤ n, , i ≠ k, j ≠ k do
            r_ij ← r_ij ∩ [r_ik • r_kj]
            if r_ij = Ø then exit(inconsistent)
end
```

general

```
PC(C)
    until stabilization of all constraints in C do
        for each k : 1 ≤ k ≤ n do
            for each pair i, j : 1 ≤ i < j ≤ n, i ≠ k, j ≠ k do
                c_ij ← c_ij ∩ [c_ik • c_kj]
                if c_ij = Ø then exit(inconsistent)
end
```

- **Floyd-Warshall algorithm**
  - finds minimal distances between all pairs of nodes
  - First, the temporal network is converted into a directed graph
    - there is an arc from i to j with distance b_{ij}
    - there is an arc from j to i with distance $-a_{ij}$.
  - STN is consistent iff there are no negative cycles in the graph, that is, d(i,i)≥0

```
Floyd-Warshall(X, E)
    for each i and j in X do
        if (i, j) ∈ E then d(i, j) ← l_ij else d(i, j) ← ∞
        d(i, i) ← 0
    for each i, j, k in X do
        d(i, j) ← min{d(i, j), d(i, k) + d(k, j)}
end
```


Dechter et al. (1991)

TCSP

Temporal Constraint Network (TCSP)

- It is possible to use **disjunctions of simple temporal constraints**.
- Operations • and ∩ are being done over the sets of intervals.
- **TCSP** is **consistent** if there is an assignment of values to instants satisfying all the temporal constraints.
- Path consistency does not guarantee in general the consistency of the TCSP network!
- A straightforward **approach** (constructive disjunction):
  - decompose the temporal network into several STNs by choosing one disjunct for each constraint
  - solve obtained STN separately (find the minimal network)
  - combine the result with the union of the minimal intervals

Resource reasoning

Foundations

Resources are used in slightly different meanings in planning and scheduling!

- **scheduling**
  - resource
    = a **machine** (space) for processing the activity
- **planning**
  - resource
    = consumed/produced **material** by the activity
  - resource in the scheduling sense is often handled via logical precondition (e.g. hand is free)

Terminology

Activity is an entity occupying some space (resource) and time while **event** is an instantaneous entity changing the level of resource.

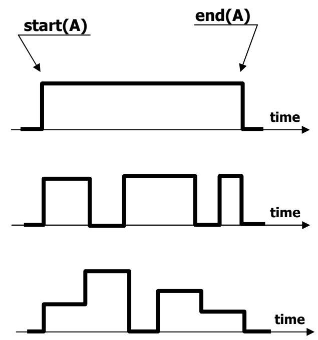
- a typical activity can be seen as a start event and an end event

For activity A we define

- **start(A)**: start time of the activity (activity cannot start before its **release date**)
  - **est(A)** = min(start(A)), earliest start time
- **end(A)**: completion time of the activity (activity must finish before the **deadline**)
  - **lct(A)** = max(end(A)), latest completion time
- **p(A)**: processing time (duration) of the activity
- **cap(A)**: requested capacity of the resource

Classification of activities:

- **non-preemptible** activity A requires cap(A) of the resource for all duration
  - start(A) + p(A) = end(A)
- **preemptible** activity A can be interrupted but it must run for p(A) time units when it requires cap(A) of the resource
  - start(A) + p(A) ≤ end(A)
- **elastic activity** can use different level of the resource at different times, but a total "energy" must be consumed
  - e(A) = p(A).cap(A)

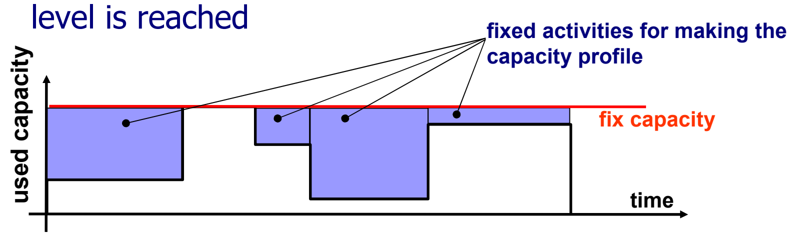


Resource types

- **unary (disjunctive) resource**
  - a single activity can be processed at given time
- **cumulative (discrete) resource**
  - several activities can be processed in parallel if capacity is not exceeded.
- **producible/consumable resource**
  - activity consumes/produces some quantity of the resource
  - minimal capacity is requested (consumption) and maximal capacity cannot be exceeded (production)

Cumulative resources

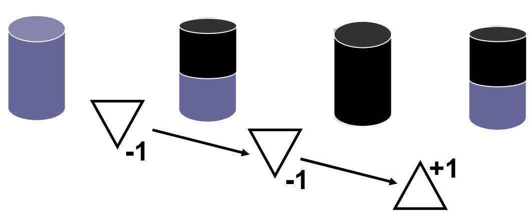
- Each **activity uses some capacity** of the resource – **cap(A)**.
- Activities can be **processed in parallel** if a resource capacity is not exceeded.
- Resource capacity **may vary in time**
  - modeled via fix capacity over time and fixed activities consuming the resource until the requested capacity level is reached



Reservoirs

Producible/consumable resource

- Each event describes how much it increases or decreases the level of the resource.



- Cumulative resource can be seen as a special case of reservoir.
  - Each activity consists of consumption event at the start and production event at the end.

Alternative resources

- **How to model alternative resources for a given activity?**
- Use a **duplicate activity** for each resource.
  - duplicate activity participates in a respective resource constraint but does not restrict other activities there
    - „failure" means removing the resource from the domain of variable res(A)
    - deleting the resource from the domain of variable res(A) means „deleting" the respective duplicate activity
  - original activity participates in precedence constraints (e.g. within a job)
  - restricted times of duplicate activities are propagated to the original activity and vice versa.

Alternative resources

filtering details

- Let A_u be the duplicate activity of A allocated to resource $u \in res(A)$.

$u \in res(A) \Rightarrow start(A) \leq start(A_u)$
$u \in res(A) \Rightarrow end(A_u) \leq end(A)$
$start(A) \geq \min\{start(A_u) : u \in res(A)\}$
$end(A) \leq \max\{end(A_u) : u \in res(A)\}$
failure related to $A_u \Rightarrow res(A) \setminus \{u\}$

Actually, it is maintaining constructive disjunction between the alternative activities.

Resource reasoning

on absolute position in time

Baptiste et al. (2001)

Disjunctive constraint

- How to describe a relation between two non-preemptive activities allocated to the same unary resource?
- Such activities A and B cannot run in parallel, so either A runs completely before B or vice versa

A«B $\vee$ B«A

$\Rightarrow$ **end(A) $\leq$ start(B) $\vee$ end(B) $\leq$ start(A)**

Propagation:

whenever lst(B)<ect(A) then A cannot finish before B starts and hence A must be after B (and vice versa)

Filtering Techniques in Planning and Scheduling 48

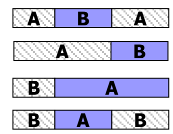
Baptiste et al. (2001)

Disjunctive constraint
preemptive version

- What if the activities **can be preempted**?
- Then the activities A and B can **interleave** during execution (for example, we can start and finish with A while B is processed in the middle), but they still cannot cover more time than available.

Four possibilities:

start(A) + p(A) + p(B) ≤ **end(A)** ∨
start(A) + p(A) + p(B) ≤ **end(B)** ∨
start(B) + p(A) + p(B) ≤ **end(A)** ∨
start(B) + p(A) + p(B) ≤ **end(B)** ∨



Note:
if A cannot be interrupted then the first disjunct can be removed

Baptiste et al. (2001)

Disjunctive constraint
cumulative version

- On a cumulative resource, two activities can overlap provided that they do not consume more than available capacity.
- Extend the previous disjunctions by **cap(A) + cap(B)** ≤ **cap**

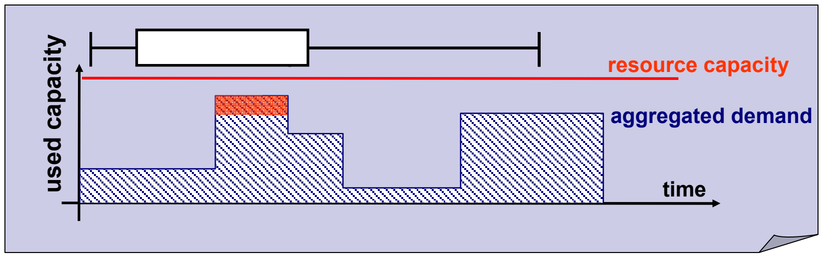
For example (cumulative non-preemptive case):

cap(A) + cap(B) ≤ **cap** ∨
end(A) ≤ **start(B)** ∨
end(B) ≤ **start(A)**

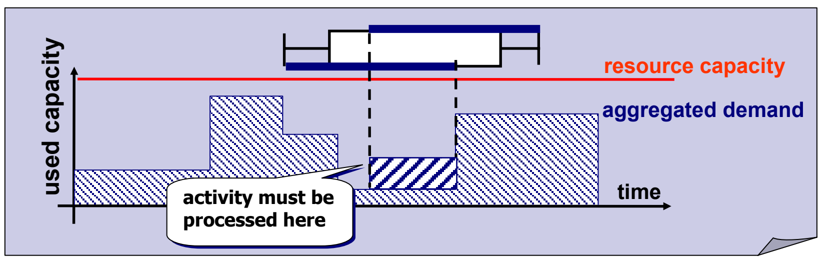
Baptiste et al. (2001)

Aggregated demands

Where is enough capacity for processing the activity?



How the aggregated demand is constructed?



Baptiste et al. (2001)

Timetable constraint

- How to ensure that capacity is not exceed at any time point?*

$$\forall t \sum_{start(A_i) \le t < end(A_i)} cap(A_i) \le cap$$

- **Timetable** for the activity A is a set of Boolean variables **X(A,t)** indicating whether A is processed in time t.

$$\forall t \sum_{A_i} X(A_i, t) \cdot cap(A_i) \le cap$$

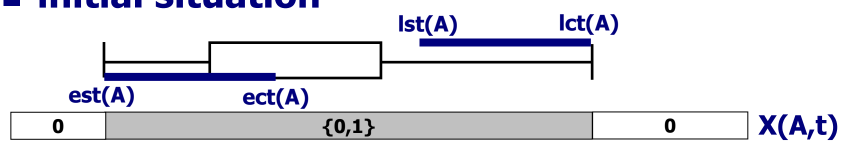
cap=1 in unary resource

$$\forall t, i \;\; start(A_i) \le t < end(A_i) \Leftrightarrow X(A_i, t)$$

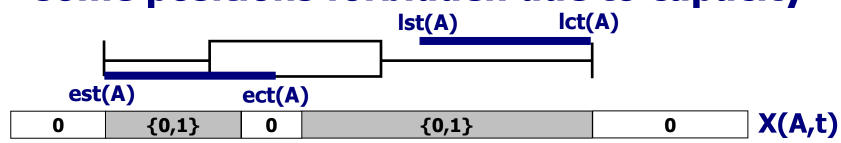
*** discrete time is expected**

Timetable constraint
filtering example

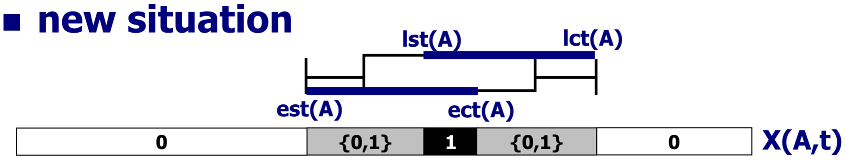
- initial situation



- some positions forbidden due to capacity



- new situation



Baptiste et al. (2001)

Timetable constraint
filtering rules

How to realize filtering through the constraint

$$\forall t,i \;\; start(A_i) \leq t < end(A_i) \Leftrightarrow X(A_i,t)\,?$$

Problem:

t serves as an index and as a variable

start(A) ≥ min{t : ub(X(A,t))=1}
end(A) ≤ 1+max{t : ub(X(A,t))=1}
X(A,t)=0 ∧ t<ect(A) ⇒ start(A)>t
X(A,t)=0 ∧ lst(A)≤t ⇒ end(A)≤t
(lst(A)≤t ∧ t<ect(A) ⇒ X(A,t)=1)

Baptiste et al. (2001)

Timetable constraint

preemptive version

- Capacity restriction is the same as before:

$$\forall t \sum_{A_i} X(A_i,t) \cdot cap(A_i) \le cap$$

- However, time bounds must be adjusted more carefully:

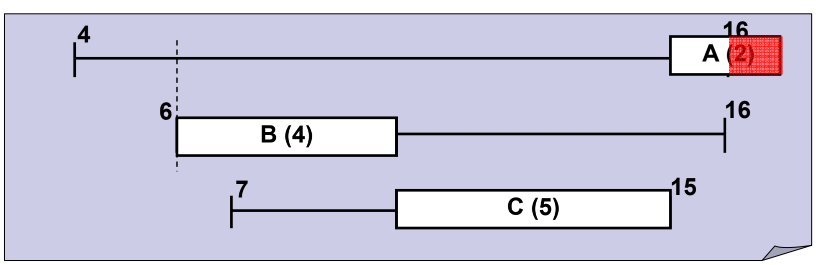
start(A) ≥ min{t : ub(X(A,t))=1}
end(A) ≤ 1+max{t : ub(X(A,t))=1}
end(A) ≥ 1+min{T : |{t : ub(X(A,t))=1 ∧ t ≤ T}| ≥ p(A)}
start(A) ≤ max{T : |{t : ub(X(A,t))=1 ∧ t ≥ T}| ≥ p(A)}

Note:
This propagation is weaker and more time consuming than in the non-preemptive case.

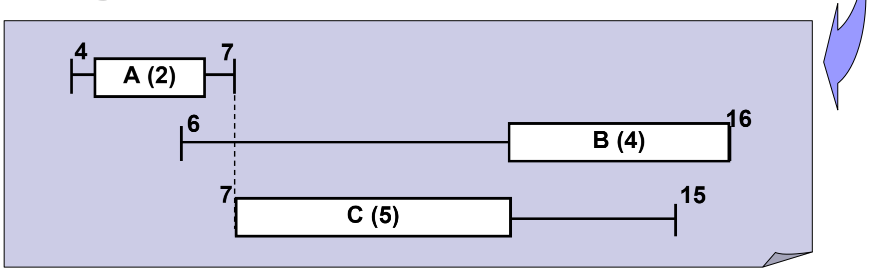
Baptiste & Le Pape (1996)

Edge finding

What happens if activity A is not processed first?



Not enough time for A, B, and C and thus A must be first!



Baptiste & Le Pape (1996)

Edge finding
rules

The rules:

- **p(Ω ∪ {A}) > lct(Ω ∪ {A}) - est(Ω) ⇒ A«Ω**
- **p(Ω ∪ {A}) > lct(Ω) - est(Ω ∪ {A}) ⇒ Ω«A**
- **A«Ω ⇒ end(A) ≤ min{ lct(Ω') - p(Ω') | Ω'⊆Ω }**
- **Ω«A ⇒ start(A) ≥ max{ est(Ω') + p(Ω') | Ω'⊆Ω }**

In practice:

- there are $n.2^n$ pairs (A,Ω) to consider (too many!)
- instead of Ω use so called **task intervals** [X,Y]
  {C | est(X) ≤ est(C) ∧ lct(C) ≤ lct(Y)}
  - time complexity $O(n^3)$, frequently used incremental algorithm
- there are also $O(n^2)$ and O(n.log n) algorithms

Filtering Techniques in Planning and Scheduling 57

Baptiste & Le Pape (1996)

Edge finding
preemptive version

p(Ω ∪ {A}) > lct(Ω ∪ {A}) - est(Ω) means that the activities in Ω ∪ {A} cannot start with an activity different from A and hence we must start with A

- if A is non-preemptive then it must finish before any activity in Ω
- if A is preemptive then its execution can be interrupted and hence A does not necessarily finish before all activities in Ω
- still A must start before all activities in Ω, denote as A<Ω

- **p(Ω ∪ {A}) > lct(Ω ∪ {A}) - est(Ω) ⇒ A<Ω**
- **A<Ω ⇒**
  start(A) ≤ min{ lct(Ω' ∪ {A}) - p(Ω' ∪ {A}) | Ω'⊆Ω }

Symmetrically:

- **p(Ω ∪ {A}) > lct(Ω) - est(Ω ∪ {A}) ⇒ Ω<A**
- **Ω<A ⇒ end(A) ≥ max{ est(Ω' ∪ {A}) + p(Ω' ∪ {A}) | Ω'⊆Ω }**
  Ω<A means that A finishes after all activities in Ω

Filtering Techniques in Planning and Scheduling 58

Nuijten (1994)

Edge finding

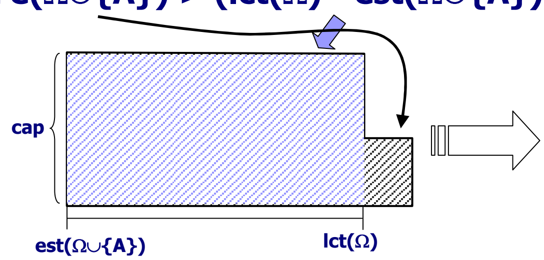
cumulative version

If activities can overlap, we can use energy of activities to describe the occupied space

- $e(A) = cap(A).p(A)$

The rules

- **$e(\Omega \cup \{A\}) > (lct(\Omega \cup \{A\}) - est(\Omega))\ .\ cap \Rightarrow A < \Omega$**
- **$e(\Omega \cup \{A\}) > (lct(\Omega) - est(\Omega \cup \{A\}))\ .\ cap \Rightarrow \Omega < A$**



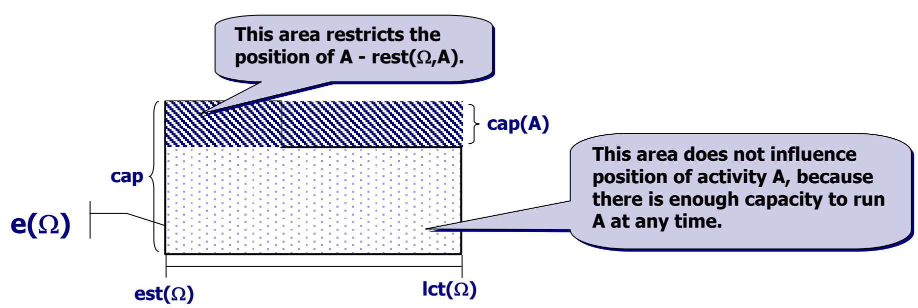
A must finish after all activities in Ω

Nuijten (1994)

Edge finding

cumulative version filtering

How to reduce time window for A if Ω<A?



let $\mathbf{rest(\Omega,A) = e(\Omega) - (cap-cap(A)).(lct(\Omega)-est(\Omega))}$

if $\mathbf{rest(\Omega,A)>0}$ then

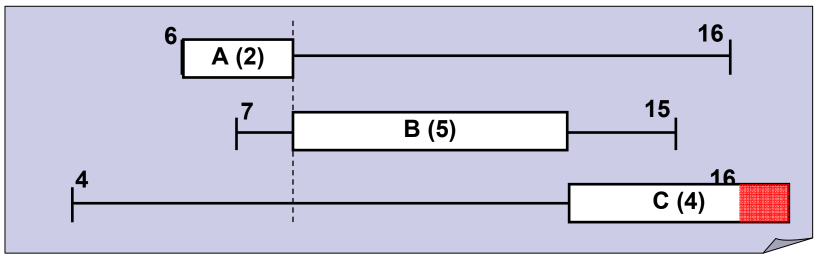
$\Omega < A \Rightarrow \mathbf{start(A)} \geq \mathbf{est(\Omega) + \lceil rest(\Omega,A)/cap(A) \rceil}$

$A < \Omega \Rightarrow \mathbf{end(A)} \geq \mathbf{lct(\Omega) - \lceil rest(\Omega,A)/cap(A) \rceil}$

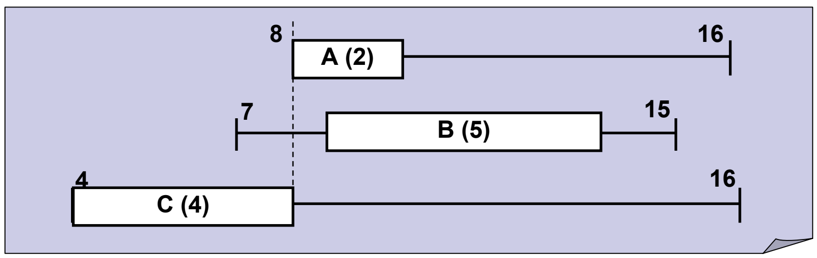
Torres & Lopez (2000)

Not-first/not-last

What happens if activity A is processed first?



Not enough time for B and C and thus A cannot be first!



Filtering Techniques in Planning and Scheduling 61

Torres & Lopez (2000)

Not-first/not-last rules

Not-first rules:

$p(\Omega \cup \{A\}) > lct(\Omega) - est(A) \Rightarrow \neg A \ll \Omega$

$\neg A \ll \Omega \Rightarrow start(A) \geq \min\{ ect(B) \mid B \in \Omega \}$

Not-last (symmetrical) rules:

$p(\Omega \cup \{A\}) > lct(A) - est(\Omega) \Rightarrow \neg \Omega \ll A$

$\neg \Omega \ll A \Rightarrow end(A) \leq \max\{ lst(B) \mid B \in \Omega \}$

In practice:

- can be implemented with time complexity $O(n^2)$ and space complexity $O(n)$

Filtering Techniques in Planning and Scheduling 62

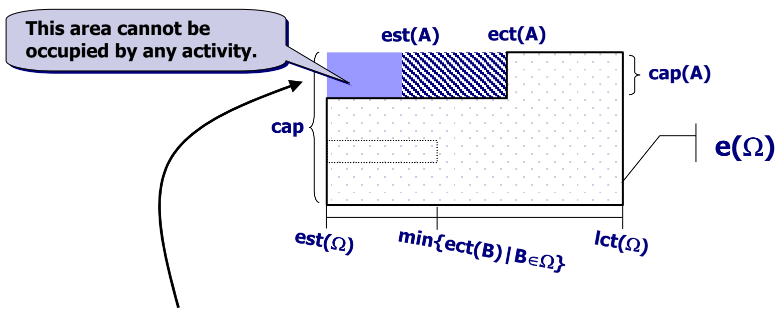
Nuijten (1994)

Not-first

cumulative version

assume that $est(\Omega) \le est(A) < \min\{ ect(B) \mid B \in \Omega \}$

- if A starts in est(A) then no activity in Ω can finish before A starts and hence the area "before" A is wasted

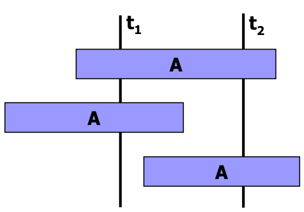


$e(\Omega) + cap(A).(\min\{ect(A),ect(\Omega)\}-est(\Omega)) > cap.(lct(\Omega)-est(\Omega))$
$\Rightarrow start(A) \ge \min\{ ect(B) \mid B \in \Omega \}$

Baptiste et al. (2001)

Energetic reasoning

- **How much time does A spent in interval (t_1,t_2)?**



  - **(t_2-t_1)**
  - **$\max(0,p(A)-\max(0,t_1-est(A))) = p^+(A,t_1)$**
  - **$\max(0,p(A)-\max(0,lct(A)-t_2)) = p^-(A,t_2)$**

- **How much energy does A consume in interval (t_1,t_2)?**
  - **$W(A,t_1,t_2) = cap(A) . \min\{t_2-t_1, p^+(A,t_1), p^-(A,t_2)\}$**
- **How much energy do all activities consume in interval (t_1,t_2)?**
  - **$W(t_1,t_2) = \Sigma_A W(A,t_1,t_2)$**

Baptiste et al. (2001)

Energetic reasoning
overload checking

- **If there exists a feasible schedule then $W(t_1,t_2) \leq cap.(t_2-t_1)$ for any time interval (t_1, t_2).**
  - It means that there must be enough energy to run all activities in any time interval.
  - **$W(t_1,t_2) > cap.(t_2-t_1) \Rightarrow$ failure**
- **Which intervals should be explored?**
  - Explore all non-empty time intervals **(est(A), lct(B)).**
  - This can be done in time $O(n^2)$.

Filtering Techniques in Planning and Scheduling 65

Baptiste et al. (2001)

Energetic reasoning
adjusting time bounds

assume that activity A finishes before t_2

- A consumes $cap(A) \,.\, p^+(A,t_1)$
- all activities in interval (t_1,t_2) consume $W(t_1,t_2) - W(A,t_1,t_2) + cap(A) \,.\, p^+(A,t_1) = LW(A,t_1,t_2)$

$LW(A,t_1,t_2) > cap.(t_2-t_1) \Rightarrow$

- activity A cannot finish before t_2
- overflow energy must be used after t_2
- $end(A) \geq t_2 + (LW(A,t_1,t_2) - cap.(t_2-t_1))/cap(A)$

Filtering Techniques in Planning and Scheduling 66

Laborie (2003)

Energetic reasoning

unary version

assume that A is ordered before B

- interval (est(A), lct(B)) consumes energy $p(A)+p(B)+\Sigma_{C\notin\{A,B\}}W(C, est(A), lct(B))$)

$lct(B) - est(A) < p(A)+p(B)+\Sigma_{C\notin\{A,B\}}W(C, est(A), lct(B))$
$\Rightarrow \neg$ A«B

- **non-preemptive version**
  - $\neg$ A«B means B«A
  - ↳ **end(B) ≤ lct(A) – p(A)**
  - ↳ **est(B)+p(B) ≤ start(A)**
- **preemptive model**
  - $\neg$ A«B means B<A
  - ↳ **start(B) ≤ lct(A) – p(A) – p(B)**
  - ↳ **est(B) + p(A) + p(B) ≤ end(A)**

Resource reasoning

on relative position in time

Relative ordering

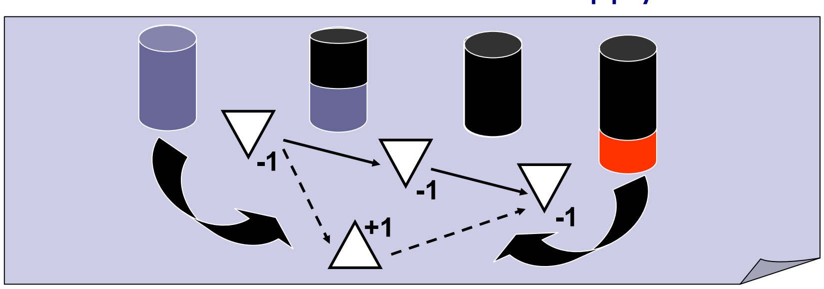
When time is relative (ordering of activities)

- then edge-finding and aggregated demand deduce nothing

We can still use information about ordering of events and resource production/consumption!

Example:

Reservoir: events consume and supply items



Cesta & Stella (1997)

Resource profiles

- Event A „produces" **prod(A)** quantity:
  - positive number means **production**
  - negative number means **consumption**
- **optimistic resource profile (orp)**
  - maximal possible level of the resource when A happens
  - events known to be before A are assumed together with the production events that can be before A

  $$\mathbf{orp(A) = InitLevel + prod(A) + \Sigma_{B \ll A}\, prod(B) + \Sigma_{B?A \wedge prod(B)>0}\, prod(B)}$$
- **pessimistic resource profile (prp)**
  - minimal possible level of the resource when A happens
  - events known to be before A are assumed together with the consumption events that can be before A

  $$\mathbf{prp(A) = InitLevel + prod(A) + \Sigma_{B \ll A}\, prod(B) + \Sigma_{B?A \wedge prod(B)<0}\, prod(B)}$$

***B?A means that order of A and B is unknown yet**

Cesta & Stella (1997)

orp filtering

- **orp(A) < MinLevel ⇒ fail**
  - "despite the fact that all production is planned before A, the minimal required level in the resource is not reached"
- $\mathbf{orp(A) - prod(B) - \Sigma_{B \ll C \wedge C?A \wedge prod(C)>0}\, prod(C) < MinLevel \Rightarrow B \ll A}$

  for any B such that B?A and prod(B)>0
  - "if production in B is planned after A and the minimal required level in the resource is not reached then B must be before A"

Cesta & Stella (1997)

prp filtering

- **prp(A) > MaxLevel ⇒ fail**
  - "despite the fact that all consumption is planned before A, the maximal required level (resource capacity) in the resource is exceeded"
- **prp(A) – prod(B) – $\sum_{B«C \wedge C?A \wedge prod(C)<0}$ prod(C) > MaxLevel ⇒ B«A**

  for any B such that B?A and prod(B)<0
  - "if consumption in B is planned after A and the maximal required level in the resource is exceeded then B must be before A"

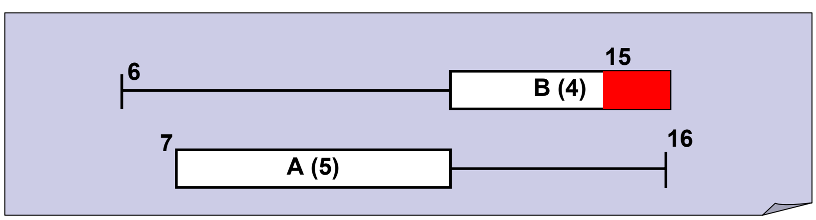
Filtering Techniques in Planning and Scheduling 72

Combined reasoning
on relative and absolute positions

Vilím (2002)

Detectable precedence
from time windows to ordering

What happens if activity A is processed before B?



- Restricted time windows can be used to deduce new precedence relations.
- est(A)+p(A)+p(B)>lct(B) ⇒ B«A

Laborie (2003)

Energy precedence

from ordering to time windows

use energy of activities processed before A to deduce new est(A)

start(A) $\geq$ max{ est(Ω') + $\lceil$e(Ω')/cap$\rceil$ | $\Omega' \subseteq$ {C : C«A} }

for unary resources

start(A) $\geq$ max{ est(Ω') + p(Ω') | $\Omega' \subseteq$ {C : C«A} }

it is enough to explore **Ω(X,A) = {Y | Y « A $\wedge$ est(X) $\leq$ est(Y)}**

start(A) $\geq$ max{ est(Ω(X,A)) + p(Ω(X,A)) | X « A }

```
dur ← 0
end ← est(A)
for each Y∈{ X | X « A } in the non-increasing order of est(Y) do
    dur ← dur + p(Y)
    end ← max(end, est(Y) + dur)
end for
est(A) ← end
```


Laborie (2003)

Balance constraint

generalization and refinement of orp/prp filtering

recall:

orp(A) = InitLevel + prod(A) + $\Sigma_{B«A}$ prod(B) + $\Sigma_{B?A \wedge prod(B)>0}$ prod(B)

maximal possible level of the resource when A happens

new filtering of time windows

If **InitLevel + $\Sigma_{B«A}$ prod(B) < MinLevel** then at least one of the events in {B : B?A $\wedge$ prod(B)>0} must happen strictly before A.

- let $\Delta_{orp(A)}$ **= MinLevel – InitLevel + $\Sigma_{B«A}$ prod(B)**
- order events B?A $\wedge$ prod(B)>0 in non-decreasing t_{min}(B): $B_1,\ldots,B_n$
- let k be the index such that $\Sigma_{i=1,\ldots,k-1}$ **prod(B_i) < $\Delta_{orp(A)}$** $\leq \Sigma_{i=k,\ldots,n}$ **prod(B_i)**
- **t_{min}(B_k) < time(A)**
  (if time(A) $\leq t_{min}(B_k)$ then MinLevel will not be reached right before A)

Reasoning
on objective functions

Objectives in CSP

- Recall that the objective (criteria) function is encoded as an equality constraint:

  v = obj(Xs)

  Example: **makespan = max{end(A_i)}**

  - it is possible to deduce better bounds of v using current domains of Xs
    - **$makespan_{min}$ = max{ect(A_i)}**
  - it is possible to restrict domains of Xs by using the current bounds of v
    - **end(A_i) ≤ $makespan_{max}$**
  - propagation of more complex objectives is typically based on solving a relaxed problem

Baptiste et al. (2001)

Minimizing late activities

objective lower bound

- assume **minimizing the number of late activities** on a single machine
  - each activity A has a due date $\delta(A)$, if A finishes after $\delta(A)$ then A is late
- **preemptive relaxation + continuous relaxation**
  - feasible preemptive schedule if and only if
    $\forall t_1,t_2>t_1\ \Sigma\{p(A) : t_1 \le est(A) \wedge lct(A) \le t_2\} \le t_2-t_1$
    use est(X) for t_1 and lct(X) or $\delta(X)$ for t_2
  - introduce a decision variable x(A) (1 when the activity is on time and 0 otherwise)
  - solve the following LP problem to obtain the lower bound of the number of late activities

min $\Sigma_A(1-x(A))$ under the constraints

$\Sigma_{A\in S(t_1,t_2)}\, p(A) + \Sigma_{A\in P(t_1,t_2)}\, x(A).p(A) \le t_2-t_1$

$est(A)+p(A)>\delta(A) \Rightarrow x(A)=0$

$x(A)\in[0,1]$

$S(t_1,t_2) = \{A : t_1 \le est(A) \wedge lct(A) \le t_2\}$
$P(t_1,t_2) = \{A : t_1 \le est(A) \wedge t_2 < lct(A) \wedge \delta(A) \le t_2\}$
$t_1 \in \{est(A)\}$
$t_2 \in \{lct(A)\} \cup \{\delta(A)\}\ (t_2 > t_1)$

Baptiste et al. (2001)

Minimizing late activities

late/on-time activity detection

- for each possibly late activity B solve the following LP problem to obtain the lower bound of the number of late activities

min $\Sigma_A(1-x(A))$ under the constraints

$\Sigma_{A\in S(t_1,t_2)}\, p(A) + \Sigma_{A\in P(t_1,t_2)}\, x(A).p(A) \le t_2-t_1$

$est(A)+p(A)>\delta(A) \Rightarrow x(A)=0$

$\mathbf{x(B) = 0}$

$x(A)\in[0,1]$

- if the obtained lower bound LB_0 is greater than the existing upper bound for the objective function then the activity must be on time
  - $LB_0 > up(obj) \Rightarrow end(A) \le \delta(A)$
- similarly for testing whether the activity must be late (try x(B) = 1)

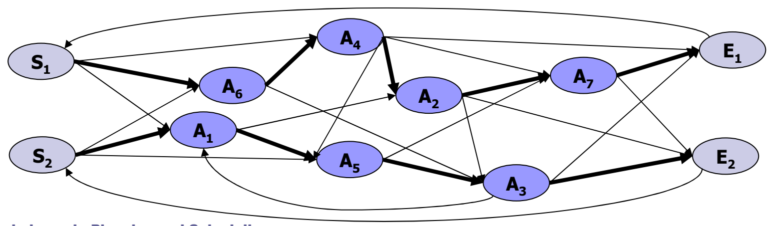
Note:

- $\min(LB_0, LB_1)$ can be used to obtain even better lower bound for the objective function
- similar to singleton consistency (constructive disjunction $x(B)=0 \vee x(B)=1$)

Focacci et al. (2000)

Minimizing setups

- assume **minimizing the setup times/costs** between activities with **alternative unary resources.**
- **path model**
  - **vertices** describe activities (there are 2.m additional vertices S_i and E_i describing the first/last time point in each resource i)
  - **arc** connect vertices A and B if it is possible to allocate A right before B on some common resource
    - arc cost is the minimal setup time/cost between the activities if they are allocated to the same resource
  - the path from S_i to E_i corresponds to a schedule of resource i
  - find m disjoint **paths** between S_i to E_i visiting all vertices and minimizing the overall cost of included arcs



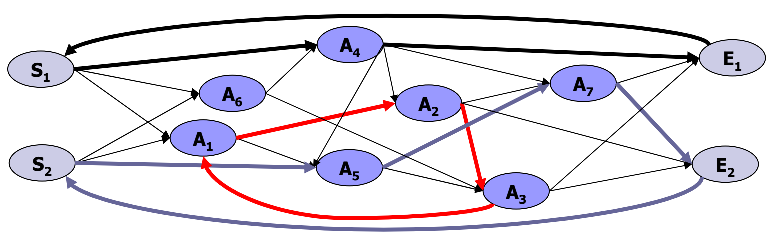
Focacci et al. (2000)

Minimizing setups

filtering rules

relaxation of the path model using an **assignment problem**

- find a set of disjoint sub-tours such that all vertices are visited and the overall cost of included arcs is minimized



- the optimal solution of the assignment problem defines a **lower bound** LB for the overall setup cost/time
- it is possible to use the **reduced cost matrix** for **further pruning**
  - c(A,B) estimates the additional cost added to LB if arc (A,B) is used
  - LB + c(A,B) > up(obj) ⇒ remove arc (A,B) from the path model

Conclusions

Constraint solvers

- It is not necessary to program all the presented techniques from scratch!
- Use existing constraint solvers (packages)!
  - provide **implementation of data structures** for modelling variables' domains and constraints
  - provide a basic **consistency framework** (AC-8)
  - provide **filtering algorithms** for many constraints (including global constraints)
  - provide basic **search strategies**
  - usually **extendible** (new filtering algorithms, new search strategies)

Filtering Techniques in Planning and Scheduling 84

Schulte (2002)

Design of filters

Users can often define code of the FILTER procedures for new constraints.

How to define new filters and integrate them into solvers?

1) decide about the event to evoke the filtering algorithm

- when the domain of involved variable is changed
  - whenever the domain changes (arc-consistency)
  - when minimum/maximum bound is changed (arc-B-consistency)
  - when the variable becomes singleton (constraint checking)
- different events (suspensions) for different variables

  Example:
  filtering for A<B is evoked after change of min(A) or max(B)

2) design the filtering algorithm for the constraint

- the result of filtering is the change of variables' domains
- more filtering procedures for a single constraint are allowed

Example: A<B

- min(A): B in min(A)+1..sup   max(B): A in inf..max(B)-1

Design of filters

integration into solver

It is necessary to specify **when** the filtering algorithm is evoked and **what** global information is available to it.

- Some algorithms are **incremental** – describe how to react to a change in domain of a particular variable.
  - evoke the algorithm after the particular change happens
- Many algorithms for global constraints are proposed as **non-incremental** – filtering is run from scratch independently of the change.
  - evoke the algorithm after any change of constrained variables

Example (installation of filtering rule(s) for A<B)

- arc-B-consistency is identical to full arc-consistency!
- filtering can be realised incrementally

```
less_then(A,B):-
  fd_global(a2b(A,B),no_state,[min(A)]),
  fd_global(b2a(A,B),no_state,[max(B)]).
```

name of the filter with arguments

initial state (local data) of the filter

events when the filter is called

Design of filters
filtering algorithm

The filtering algorithm has access to current domains and it proposes how to restrict the domains (list of changes to domains).

- some solvers provide information about the change that invoked the filter

Example (definition of filtering rule(s) for A<B)

identification of the filter

states before/after running the filtering

list of proposed changes to domains

access to current domains of variables

filtering finished, don't call the filter anymore

```
dispatch_global(a2b(A,B),S,S,Actions):-
   fd_min(A,MinA), fd_max(A,MaxA),
   fd_min(B,MinB),
   (MaxA<MinB ->
        Actions = [exit]
   ;    LowerBoundB is MinA+1,
        Actions = [B in LowerBoundB..sup]).

dispatch_global(b2a(A,B),S,S,Actions):-
   fd_max(A,MaxA),
   fd_min(B,MinB), fd_max(B,MaxB),
   (MaxA<MinB ->
        Actions = [exit]
   ;    UpperBoundA is MaxB-1,
        Actions = [A in inf..UpperBoundA]).
```

Filtering Techniques in Planning and Scheduling 87

Summary

Mainstream constraint satisfaction framework:

- **local consistency** connecting filtering algorithms for individual constraints
  - filters **encapsulate** dedicated reasoning techniques
  - filters are not "run once" codes, but they are **called repeatedly**
  - filters **communicate via domains** of variables
- **search** resolving remaining disjunctions
  - filters are used to **prune the search space**
  - filters may **provide guideline** to search

Filtering Techniques in Planning and Scheduling 88

COMMENTED BIBLIOGRAPHY

Allen, J.F. (1983). Maintaining knowledge about temporal intervals. Communications of the ACM, 21(11): 832-843.
Introduction of interval algebra and description of path-consistency filtering algorithm for handling it.

Baptiste, P. and Le Pape, C. (1996). Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group (PLANSIG).
Description of edge-finding rules for non-preemptive disjunctive scheduling, preemptive and mixed disjunctive scheduling, and non-preemptive cumulative scheduling, and a quadratic algorithm for not-first/not-last rules.

Baptiste, P.; Le Pape, C.; Nuijten, W. (2001). Constraint-based Scheduling: Applying Constraints to Scheduling Problems. Kluwer Academic Publishers, Dordrecht.
A comprehensive text on using constraint satisfaction techniques in scheduling with detailed description of many filtering algorithms for resource constraints.

Barták, R. (2005). Constraint Satisfaction for Planning and Scheduling. In Ionannis Vlahavas, Dimitris Vrakas (eds.): Intelligent Techniques for Planning, Idea Group, 2005, pp. 320-353
An introductory and survey text about constraint satisfaction techniques for planning and scheduling.

Barták, R. and Čepek, O. (2005). Incremental Propagation Rules for a Precedence Graph with Optional Activities and Time Windows. In Proceedings of The 2nd Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA), Volume II, Stern School of Business, New York, 552-560.
Description of incremental algorithms for maintaining a transitive closure of the precedence graph with optional activities and realising the energy precedence constraint on unary resources.

Brucker, P. (2001). Scheduling Algorithms. Springer Verlag.
A comprehensive book on traditional scheduling algorithms including detailed classification of many scheduling problems.

Carlier, J. and Pinson, E. (1994). Adjustment of heads and tails for the job-shop problem. European Journal of Operational Research 78(2), 146-161.
Description of first O(n.log n) algorithm for edge-finding, but this algorithm requires complex data structures.

Caseau, Y. and Laburthe, F. (1995). Disjunctive scheduling with task intervals. LIENS Technical Report 95-25, Laboratoire d'Informatique de l'Ecole Normale Superieure.
Description of incremental $O(n^3)$ algorithm for edge-finding using task intervals.

Cesta, A. and Stella, C. (1997). A Time and Resource Problem for Planning Architectures. Recent Advances in AI Planning (ECP'97), LNAI 1348, Springer Verlag, 117-129.
Description of resource profiles and orp/prp filtering rules.

Dechter, R.; Meiri, I.; Pearl, J. (1991). Temporal Constraint Networks. Artificial Intelligence 49: 61-95.
Introduction of Temporal Constraint Networks and Simple Temporal Problems.

Dechter, R. (2003). Constraint Processing. Morgan Kaufmann.
A comprehensive book on constraint satisfaction techniques, including a detailed description of temporal constraint networks.

Focacci, F.; Laborie, P.; Nuijten, W. (2000). Solving scheduling problems with setup times and alternative resources. In Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS). AAAI Press, 92-101.
Description of path optimization constraint for minimizing setup times/costs in problems with alternative resources.

Ghallab, M.; Nau, D.; Traverso, P. (2004). Automated Planning: Theory and Practice. Morgan Kaufmann.
A comprehensive book on planning, including a description of constraint satisfaction techniques for planning.

Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. Artificial Intelligence 143, 151-188.
Introduction of filtering rules for energy precedence and balance constraints (algorithms are not described).

Lhomme, O. (1993). Consistency techniques for numeric CSPs. In Proc. 13th International Joint Conference on Artificial Intelligence.
Description of arc-B-consistency algorithm.

Mackworth, A.K. (1977). Consistency in Networks of Relations. Artificial Intelligence 8, 99-118.
Description of the basic arc and path consistency algorithms – AC-1, AC-2, AC-3, PC-1, PC-2.

Marriott, K. and Stuckey, P.J. (1998). Programming with Constraints: An Introduction. MIT Press.
A practically oriented book on using constraint satisfaction technology for problem solving.

Martin, P. and Shmoys, D.B. (1996). A new approach to computing optimal schedules for the job-shop scheduling problem. Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization. LNCS 1084, Springer Verlag, 389-403.
Description of alternative formulation of edge-finding rules.

Montanari, U. (1974). Networks of constraints: fundamental properties and applications to picture processing. Information Sciences 7, 95-132.
Introduction and formalization of constraint networks, defining path-consistency and algorithm for PC.

Nuijten, W.P.M. (1994). Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach. PhD thesis, Eindhoven University of Technology.
Description of several filtering algorithms for scheduling problems including the cumulative version of edge-finding and not-first/not-last rules.

Prosser, P.; Stergiou, K.; Walsh, T. (2000). Singleton Consistencies. Proceedings Principles and Practice of Constraint Programming (CP2000), 353-368.
Description and a theoretical study of singleton consistency techniques.

Phan-Huy, T. (2000). Constraint Propagation in Flexible Manufacturing. LNEMS 492, Springer Verlag.
PhD thesis with the description of constraint propagation algorithms for disjunctive scheduling problems.

Régin, J.-Ch. (1994). A filtering algorithm for constraints of difference in CSPs. Proceedings of 12th National Conference on Artificial Intelligence, AAAI Press, 362-367.
Description of the filtering algorithm behind the all-different constraint – based on matching over bipartite graphs.

Schulte, Ch. (2002). Programming Constraint Services. LNAI 2302, Springer Verlag.
A book describing insides of constraint solvers.

Torres, P. and Lopez, P. (2000). On Not-First/Not-Last conditions in disjunctive scheduling. European Journal of Operational Research 127, 332-343.
Description of $O(n^2)$ filtering algorithms for not-first/not-last rules.

Tsang, E. (1995). Foundations of Constraint Satisfaction. Academic Press, London.
A comprehensive book on foundational constraint satisfaction techniques with description of many consistency algorithms and their theoretical study.

Vilain, M. and Kautz, H. (1986). Constraint propagation algorithms for temporal reasoning. In Proceedings of the National Conference on Artificial Intelligence (AAAI), 377-382.
Introduction of point algebra and proof that consistency-checking problem of the IA problem is an NP-complete problem, while PA is a tractable problem.

Vilím, P. and Barták, R. (2002). Filtering Algorithms for Batch Processing with Sequence Dependent Setup Times. Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS). AAAI Press, 312-320.
Description of edge-finding and not-first/not-last algorithms for batch processing with sequence dependent setup times.

Vilím, P. (2002). Batch processing with sequence dependent setup times: New results. In Proceedings of the 4th Workshop on Constraint Programming for Decision and Control (CPDC), Gliwice, 53-58.
Description of edge-finding and not-first/not-last algorithms for problems with sequence dependent setup times, introduction of detectable precedences.

Vilím, P. (2004). O(n log n) Filtering Algorithms for Unary Resource Constraint. Proceedings of CPAIOR 2004. LNCS 3011, Springer Verlag, 335-347.
Description of O(n.log n) algorithms for not-first/not-last rules and detectable precedences.

Vilím, P.; Barták, R.; Čepek, P. (2005). Extension of O(n log n) filtering algorithms for the unary resource constraint to optional activities. Constraints, 10(4): 403-425.
Description of O(n.log n) versions of filtering algorithms for edge finding, not-first/not-last, and detectable precedences and their extension to optional activities.

Wallace, M. (1994). Applying Constraints for Scheduling. In Mayoh B. and Penjaak J. (eds.), Constraint Programming. NATO ASI Series, Springer Verlag.
A survey text on using constraint satisfaction technology in scheduling.