ICAPS05

TU2

Tutorial on Hard and Soft Temporal Constraint Techniques for Scheduling Problems with Preferences and Uncertainty

Francesca Rossi
University of Padova, ITALY

Brent Venable
University of Padova, ITALY

ICAPS 2005
Monterey, California, USA
June 6-10, 2005

CONFERENCE CO-CHAIRS:

Susanne Biundo
University of Ulm, GERMANY

Karen Myers
SRI International, USA

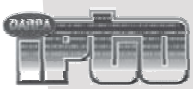
Kanna Rajan
NASA Ames Research Center, USA

Cover design: L.Castillo@decsai.ugr.es

Tutorial on Hard and Soft Temporal Constraint Techniques for Scheduling Problems with Preferences and Uncertainty

Francesca Rossi
University of Padova, ITALY

Brent Venable
University of Padova, ITALY

Tutorial on Hard and Soft Temporal Constraint Techniques for Scheduling Problems with Preferences and Uncertainty

Table of contents

Tutorial on Hard and Soft Temporal Constraint Techniques for Scheduling Problems with Preferences and Uncertainty

Preface

This tutorial will describe the main approaches to modelling and solving soft constraints, and will show how to use soft constraint techniques to model and solve scheduling problems with preferences and uncertainty. In particular, we will focus on scheduling problems that can be described by quantitative temporal constraints where each event duration can be associated to a level of preferences, and where some events may be uncontrollable. Then we will introduce preferences in uncertain scenarios, we will define suitable notions of optimal controllability and corresponding checking algorithms, and we will show that the addition of preferences does not make the problems more difficult. The tutorial is intended for researchers in the area of planning and scheduling who desire to know more about how soft constraint techniques can be useful in their context.

Instructors

- *Francsca Rossi, University of Padova, Italy*
- *Kristen Brent Venable, University of Padova, Italy*

Hard and soft temporal constraint techniques for scheduling problems with preferences and uncertainty

Francesca Rossi, Kristen Brent Venable
University of Padova, Italy

Outline

- Hard Constraints
- Soft Constraints
- Simple Temporal Problems
- Simple Temporal Problems with Preferences
- Simple Temporal Problems with Preferences and Uncertainty

Hard Constraints

Outline

- Search
- Constraint Propagation
  - AC, PC, k-Consistency
- Constraint Propagation + Search
- Adaptive Consistency

Classical (hard) Constraints

(Montanari '74, Dechter 2003)

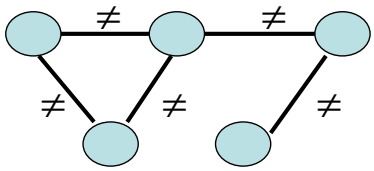
- Variables $\{X_1,\ldots,X_n\}=X$
- Domains $\{D(X_1),\ldots,D(X_n)\}=D$
- Constraints: each constraint c connects a subset of X and is a subset of the Cartesian product of the domains of its variables (allowed combinations)
- Solution: instantiation of all variables to values in their domains such that all constraints are satisfied

Typical questions

- Find a solution
  - Difficult: NP-hard
- Is t a solution?
  - Easy: just check all the constraints

Example: graph coloring

- Variables: nodes of the graph
- Domains: sets of colours
- Constraints: two variables, set of pairs of colours (different colours)



Example: crypto-arithmetic

- SEND+MORE=MONEY
- Variables: the 8 letters
- Domains: [0..9]
- Constraints:
  - $x \neq y$ for all x,y
  - Sum constraint(s)

Example: n queens

- Place n queens in an nxn chessboard such that they do not attack each other
- Variables: x1,...,xn (one per column)
- Domains: [1..n] (row position of a queen)
- Constraints:
  - xi $\neq$ xj for all i,j (no attack on a row)
  - xi-xj $\neq$ i-j (no attack on the SO-NE diagonal)
  - xi-xj $\neq$ j-i (no attack on a NO-SE diagonal)

Example: qualitative temporal reasoning

- Events with a start time and an end time, qualitative constraints relating the intervals for the events
- Variables: pairs of events
- Domains: 13 temporal relations over pairs of intervals (before, after, equal, ...)
- Constraints:
  - Allen's composition table
  - Es.: before x before = before

Example: quantitative temporal reasoning

(Meiri, Dechter, Pearl '90)

- Events with a start time and an end time, quantitative constraints relating the intervals for the events
- Variables: start or end time for each event
- Domains: time points
- Constraints:
  - $a \leq x_i - x_j \leq b$
  - Possibly more than one for each pair of variables

Modelling a constraint problem

- A problem (CSP) may have several different representations
- A solver may have very different complexity over different representations

Search

- Search the space of all complete instantiations to find a solution
- Exponential number of instantiations ➔ generate and test is too costly
- Backtrack search: create a tree of problems, to build while visiting it

Search tree

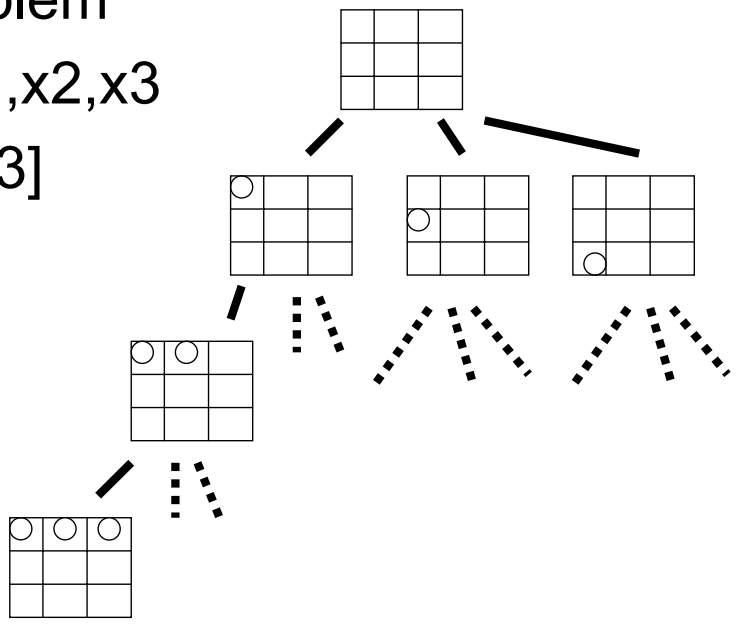
- Tree of constraint problems
- Root: given problem
- P1,...,Pn children of P:
  - P equivalent to P1 ∪ ... ∪ Pn (same set of solutions)
  - Pi has one more instantiated variable
- Leaves: problems with all variables instantiated ➔ easy to check if consistent or not

Backtracking search

- Visit of the tree: from the root problem, move down as much as possible, backtrack when no more values for next variable
- As soon as we find one solved leaf problem, its solution is also a solution for the given problem
- Node with an empty domain ➔ do not search in its subtree

Example

- 3-queen problem
- Variables: x1,x2,x3
- Domains: [1,3]



Choices to make

- Next variable to instantiate
  - Ex.: smallest domain, most constrained
- Next value to try for a variable
  - Ex.: central value in an interval

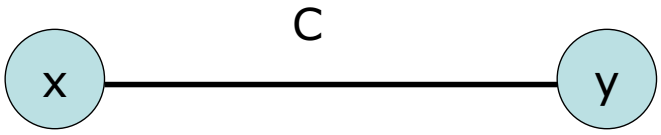
Constraint propagation

- Transform a CSP into an equivalent simpler CSP
- Main idea: remove elements from domains or tuples from constraints if they cannot participate in any solution
- Aim: to obtain a local consistency property

Arc-consistency

(Mackworth '77, Mackworth and Freuder '85)

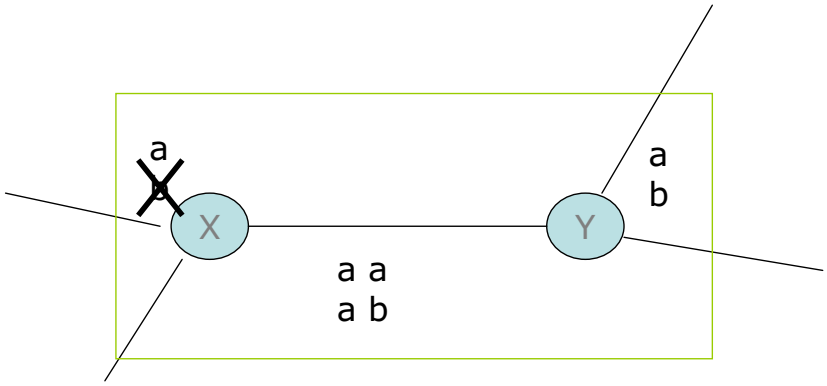
- Take any constraint C over x and y
- For each variable x, and each value v from its domain, there exists a value v' in the domain of y such that (x,y) satisfies C
- And viceversa (from y to x)



Arc-consistency

- If not AC:
  - Delete those elements from the domains of x and y which create the problem
  - Iterate until stability
- Properties:
  - Always terminates if finite domains
  - Order independent
  - Equivalence preserving
- Polynomial time (n^2)

Example



Not AC: no matter what the other constraints are,
X=b cannot participate in any solution.
So we can delete it without changing the set of solutions.

Examples

- N-queen problems: AC
- Graph coloring problems with 2 or more colors: AC

Example

- CSP:
  - x+y = 10, y+z ≤ 3
  - Dx = [1,0], Dy = [5,15], Dz = [-10,10]
- Not AC: for x=10, there is no value for y in [5,15] such that x+y = 10
- Reduced domains: Dx = [1,5], Dy=[5,9], Dz = [-10,-2]
- Now it is AC

AC and search

- AC reduces the domains of the variables ➔ smaller branching factor in the search tree
- Faster to get an empty domain

Example of search+AC

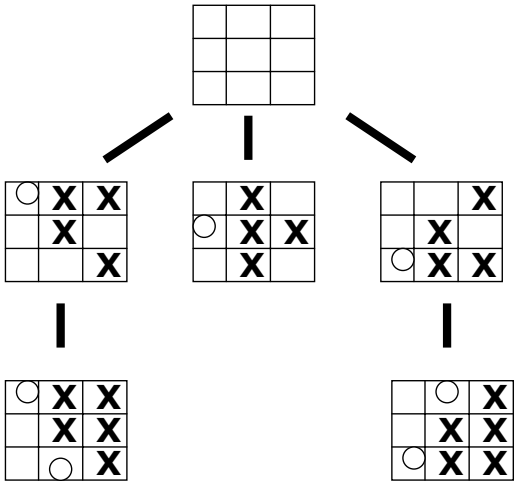
- 3-queen problem
- AC at every node

of the search tree

- Variables: x1,

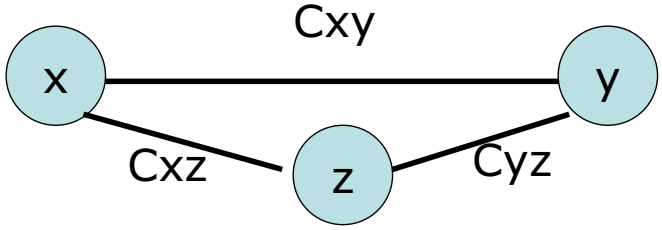
then x2, then x3

- Values: 1,2,3



Path-consistency
(Montanari '74)

- Given any two variables x and y, and any values v_x and v_y from their domains, such that C_{xy} is satisfied, there exists an instantiation for any 3rd variable z, say v_z, such that C_{xz} and C_{yz} are satisfied



Path-consistency

- If not PC:
  - Delete those pairs from C_{xy} which create the problem
  - Iterate until stability
- Properties (same as for AC):
  - Always terminates if finite domains
  - Order independent
  - Equivalence preserving
- Polynomial time (n^3)

Examples

- CSP:
  - x+y = 10, y+z ≤ 3
  - Dx = [1,5], Dy=[5,9], Dz = [-10,-2]
- AC but not PC:
  - For (x=1,z=-2), there is no value for y from [5,9] such that x+y=10 and y+z ≤ 3
- Another example: any graph coloring problem with 3 colors is PC

K-consistency

- Given any k-1 variables $x_1,...,x_{k-1}$, and any k-1 values from their domains, such that all the constraints among them are satisfied, there exists an instantiation for any k^{th} variable x_k such that all the constraints among $x_1,...,x_k$ are satisfied
- K=3 ➔ k-consistency = PC
- K=2 ➔ k-consistency = AC
- Strong k-consistency = i-consistency for i=1,...,k

Constraint propagation + search

- Obtaining local consistency at each step may be costly, but can prune the tree
- PC+AC (strong 3-cons.) is more costly than AC (2-cons), but can prune more
- Strong k-consistency is more costly than strong k-1-consistency for any k>1
- Compromise between pruning power and cost of obtaining strong k-consistency
- Usually AC or variants of it between AC and PC

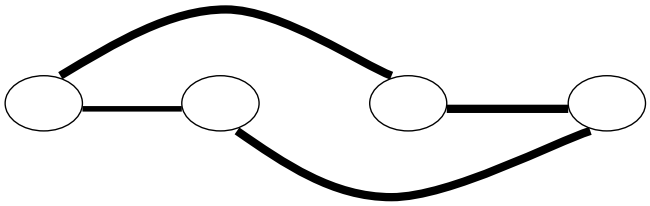
Constraint propagation alone

- Sometimes constraint propagation is enough ➔ no search is needed to find a solution
- Based on properties of the CSP graph or of the constraints

Consistency and width

- Given a linear ordering < of the variables x1,...,xn:
  - Width of a node w.r.t. <: number of previous nodes in < to which it is connected by an arc
  - Width of a graph w.r.t. <: maximum width of its nodes w.r.t. <
  - Width of a graph: minimum width of the graph w.r.t. all <

Example



Node width: 0 1 1 2

➔ graph width w.r.t. < = 2

➔ graph width = 2 (it is not possible to have an ordering with graph width =1)

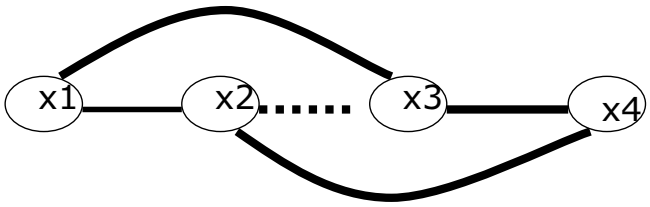
Consistency and width
(Dechter and Pearl '87)

- In general: if the width of the CSP graph is k, then k+1-consistency is enough to solve the CSP
- If the CSP is a tree (width=1), AC is enough to solve the CSP
- If width=2 (some cycles), PC is enough
- If width=n-1 (complete graph), n-consistency is enough (it means solving he problem!)

Adaptive consistency

- Choose a linear ordering of the variables: x1,...,xn
- From xn to x1, take xi:
  - Obtain the solutions of the CSP over xi U frontier(xi)
  - Project this constraint over frontier(xi)
  - Add the constraint to the CSP
- For each i, this means obtaining (directional) local consistency over xi U frontier(xi): for each instantiation of frontier(xi), there exists a value for xi which is compatible
- At the end, a solution can be found by instantiating x1,...,xn taking for each variable a value from its domain which is compatible with the values chosen for the previous variables

Example



For x4: add the constraint between x2 and x3
For x3: change the constraint between x1 and x2
For x2: modify the domain of x1 (AC direzionale)

Then, instantiate:
X1 in its domain
X2 in its domain such that C12 is satisfied
X3 in its domain such that C12, C13, and C23 are satisfied
X4 in its domain such that C23, C24, and C34 are satisfied

Soft Constraints

Outline

- C-semiring framework
- Expressive Power
- Projection, Combination
- Branch & Bound
- Soft Constraint Propagation
- Cuts
- Variable Elimination

Classical constraints are useful for ...

- resource allocation
- scheduling
- vehicle routing
- car sequencing
- timetabling
- graph coloring
- VLSI design
- vision
- Robotics
- ...

However, in most real-life situations we need to express fuzziness, possibilities, preferences, probabilities, costs, ...

Why is this not enough?

- Over-constrained problems
  - it does not make sense to just say that there is no solution
- Optimization problems
  - Also multi-criteria
- Problems with both preferences and hard statements
  - ex.: time-tabling (hard constraints about space and time, soft constraints about teachers)
- Problems with uncertainty

A problem with preferences

- The problem: to decide what to eat at a restaurant, given some choices for drinks and dishes
- The client has some preferences over drinks, dishes, and their combinations
  - Examples:
    - White wine is the most prefred drink
    - Pizza is the most preferred dish
    - Fish is less preferred
    - white wine and fish is the most preferred combination
    - red wine and meat is less preferred
- We look for a combination which “maximizes” the overall preference of the client

Soft Constraints: the C-semiring framework

(Bistarelli, Montanari, Rossi ‘97)

- Variables $\{X_1,\ldots,X_n\}=X$
- Domains $\{D(X_1),\ldots,D(X_n)\}=D$
- C-semiring $\langle A,+,x,\mathbf{0},\mathbf{1}\rangle$:
  - **A** set of preferences
  - **+** additive operator, inducing the ordering: **a≥b iff a+b=a** (idempotent, commutative, associative, unit element 0);
  - **x** multiplicative operator: combines preferences (commutative, associative, unit element 1, absorbing element 0)
  - **0**,**1** respect. bottom and top element
  - x distributes over +

Soft Constraints: the C-semiring framework

- + and x monotone on ≤
- for all a,b in A, a x b ≤ a
- for all a in A, 0 ≤ a ≤ 1
- Lattice <A,≤>
  - + is lub
  - x glb if x idempotent
- When ≤ is a total order and x idempotent:
  - if a ≤ b, then a + b = b and a x b = a, thus +=max and x=min

Soft constraints

- Soft constraint:a pair c=<f,con> where:
  - Scope: con={X^c_1,…, X^c_k} subset of X
  - Preference function :

    f: $D(X^c_1)x\ldots xD(X^c_k) \rightarrow A$

    tuple $(v_1,\ldots, v_k) \rightarrow p$ *preference*
- Hard constraint: a soft constraint where for each tuple (v_1,…, v_k)

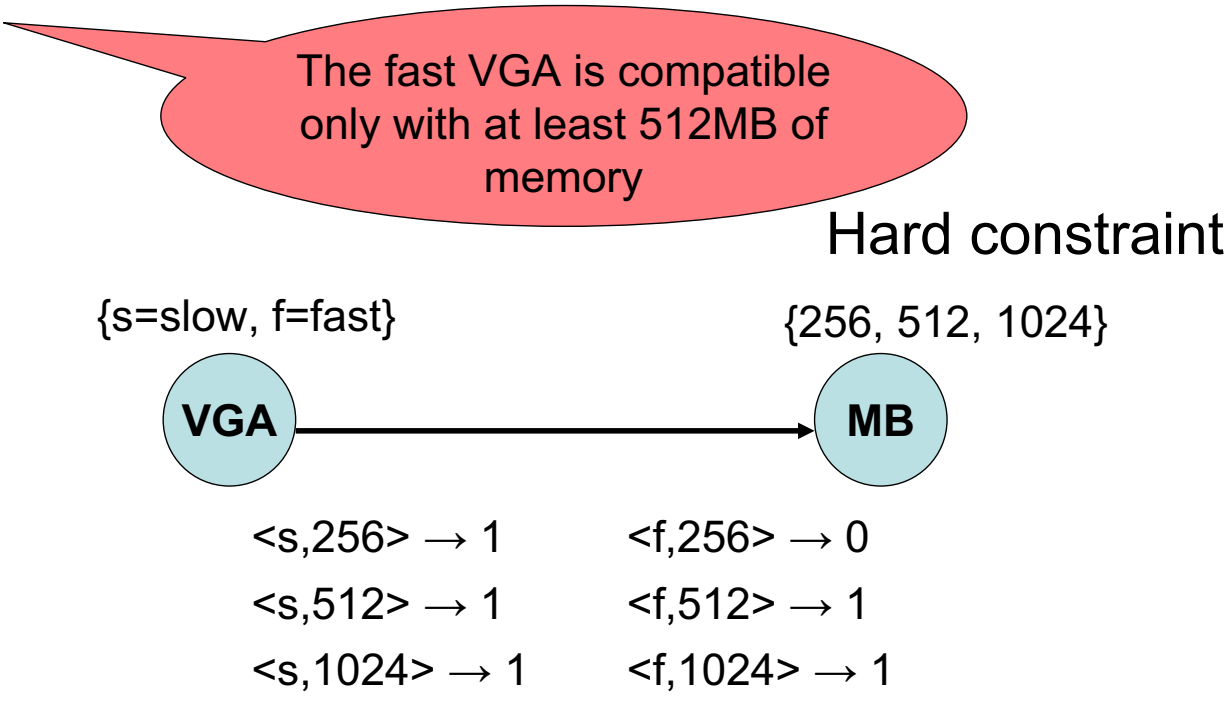
  $f(v_1,\ldots, v_k)=0$ the tuple is allowed

  $f(v_1,\ldots, v_k)=1$ the tuple is forbidden

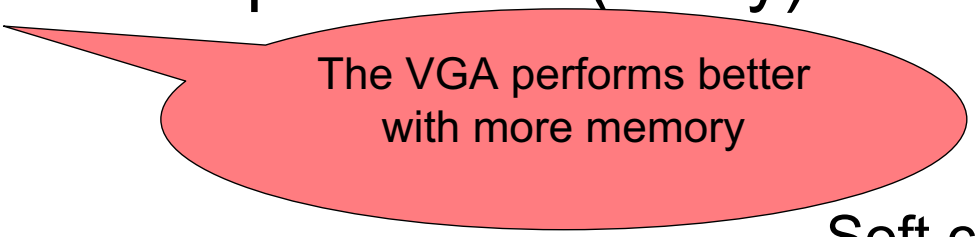
Instances of soft constraints

- Each instance is characterized by a c-semiring <A, +, x, 0, 1>
- Classical constraints: <{0,1},or,and,0,1>
- Fuzzy constraints: <[0,1],max,min,0,1>
- Lexicographic CSPs: <$[0,1]^k$,lex-max,min,0^k,1^k>
- Weighted constraints (N):<N$\cup+\infty$,min,+,$+\infty$,0>
- Weighted constraints (R):<R$\cup+\infty$,min,+,$+\infty$,0>
- Max CSP: weight =1 when constraint is not satisfied and 0 is satisfied
- Probabilistic constraints: <[0,1],max,x,0,1>
- Valued CSPs: any semiring with a total order

Example of hard constraint

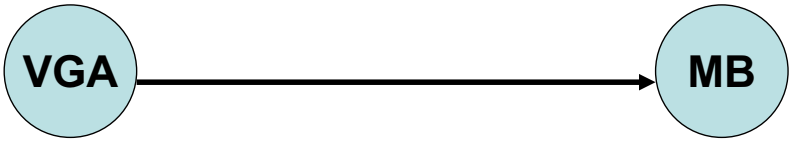


Example of soft (fuzzy) constraint



Soft constraint

{s=slow, f=fast} {256, 512, 1024}



<s,256> → 0.6 <f,256> → 0.1

<s,512> → 0.7 <f,512> → 0.9

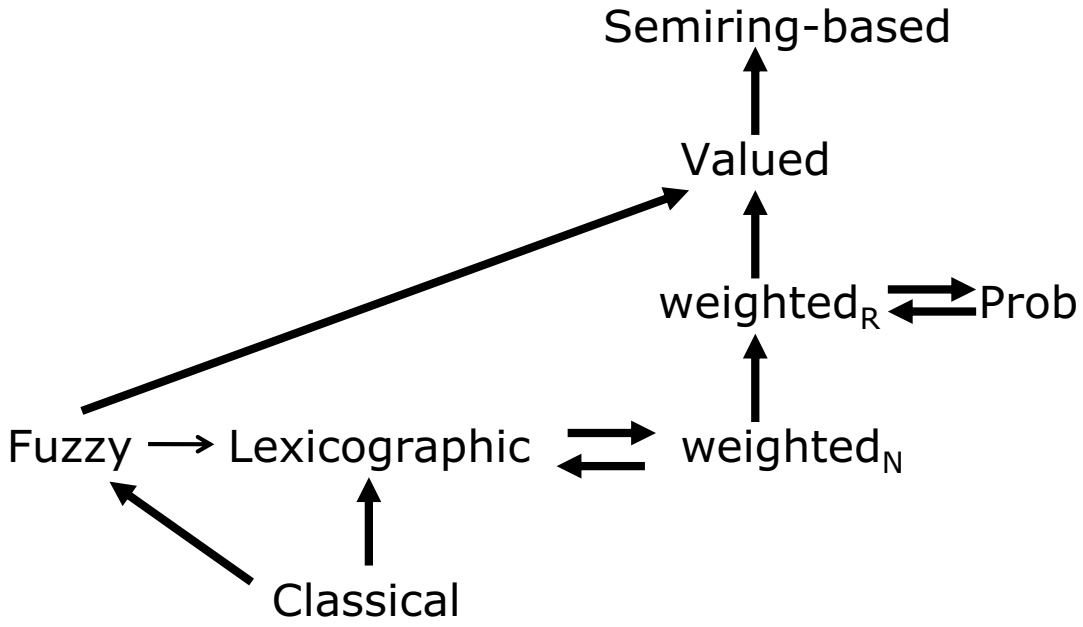
<s,1024> → 0.9 <f,1024> → 1

➔Soft constraints are a quantitative way of expressive preferences

Expressive power

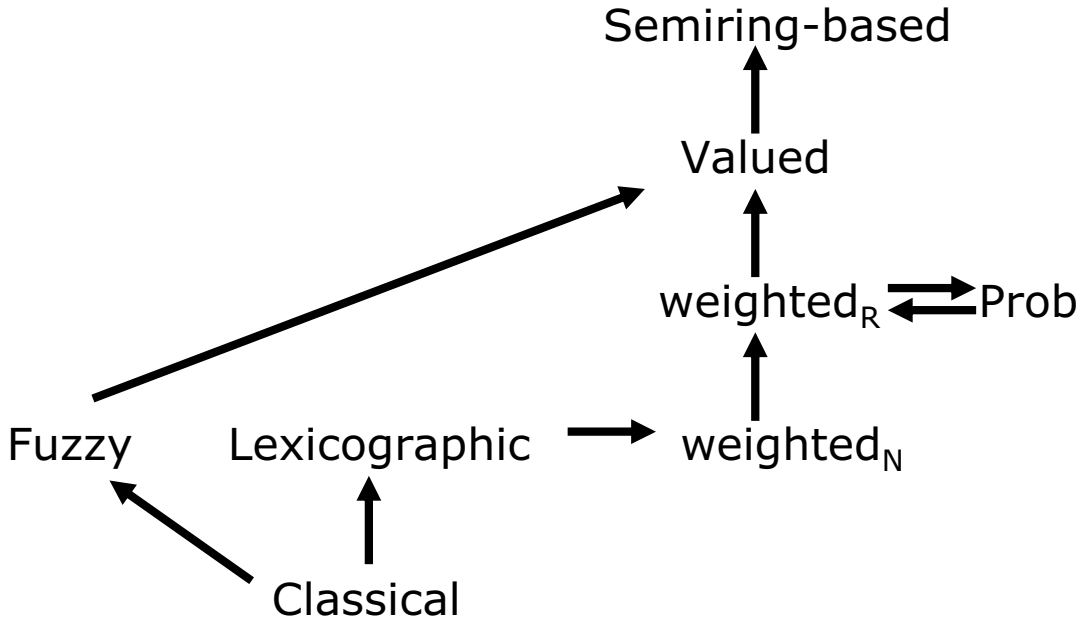
- A ⟶ B iff from a problem P in A it is possible to build in polynomial time a problem P' in B s.t. the optimal solutions are the same (but not necessarily the solution ordering!)
- A ⟶ B iff from a problem P in A it is possible to build in polynomial time a problem P' in B s.t. $opt(P') \subseteq opt(P)$

Expressive power



Expressive power

If interested in maintaining the solution ordering:

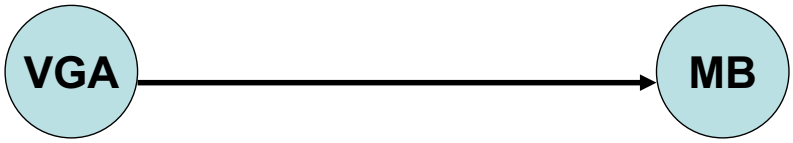


Projection

If c=<f,con>, then $c|_I$ = <f', I ∩ con>

– f'(t') = + (f(t)) over tuples of values t s.t. $t|_{I \cap con}$ = t'

{s=slow, f=fast} {256, 512, 1024}



<s,256> → 0.6 <f,256> → 0.1

<s,512> → 0.7 <f,512> → 0.9

<s,1024> → 0.9 <f,1024> → 1

Projection over VGA:
- s→ max(0.6,0.7,0.9) = 0.9
- f → max(0.1,0.9,1) = 1

Fuzzy constraints: <[0,1],max,min,0,1>

Combination

If ci=<fi,coni>, then

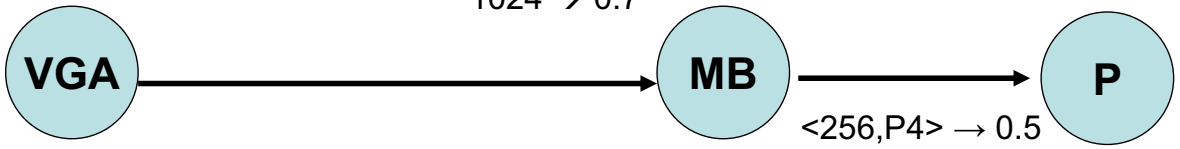
c1 x c2 = <f, con1 ∪ con2>

– f(t) = f1($t|_{con1}$) x f2($t|_{con2}$)

s=slow → 0.2
f=fast → 0.9

256 → 0.5
512 → 0.8
1024 → 0.7

P4 → 0.7
AMD → 0.4



<256,P4> → 0.5

<s,256> → 0.6 <f,256> → 0.1

<512,P4> → 0.7

<s,512> → 0.7 <f,512> → 0.9

<1024,P4> → 0.9

<s,1024> → 0.9 <f,1024> → 1

<256,AMD> → 0.5

<512,AMD> → 0.5

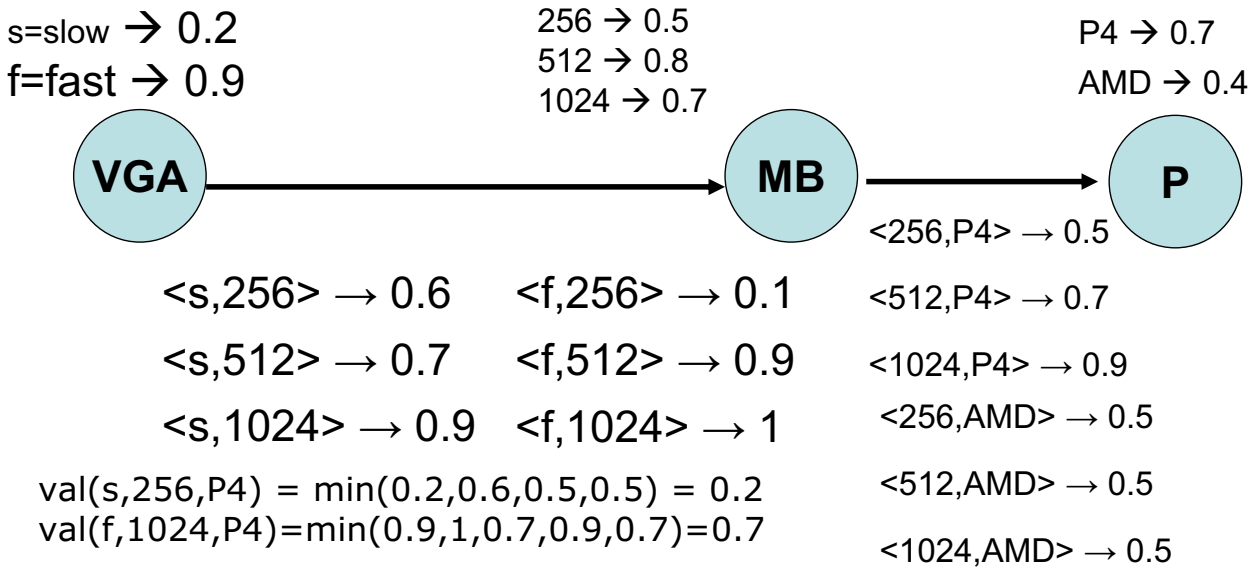
f(s,256,P4) = min(0.2,0.6,0.5,0.5) = 0.2

f(f,1024,P4)=min(0.9,1,0.7,0.9,0.7)=0.7

<1024,AMD> → 0.5

Solutions

- Sol(C) = X(C)
  - an element of A for each tuple of elements of D
- $val(t) = f1(t|_{con1}) \times ... \times fk(t|_{conk})$



Multi-criteria problems

- Main idea: one semiring for each criteria
- Given n c-semirings $S_i = <A_i, +_i, x_i, 0_i, 1_i>$, we can build the c-semiring

$<<A_1,..., A_n>, +, x, <0_1,...,0_n>,<1_1,...,1_n>>$

- + and x obtained by pointwise application of $+_i$ and x_i on each semiring
- A tuple of values associated with each variable instantiation
- A partial order even if all the criteria are totally ordered

Example

- The problem: choosing a route between two cities
- Each piece of highway has a preference and a cost
- We want to both minimize the sum of the costs and maximize the preference
- Semiring: by putting together one fuzzy semiring and one weighted semiring:
  - <[0,1],max,min,0,1>
  - <N, min, +, +∞, 0>
- Best solutions: routes such that there is no other route with a better semiring value
  - <0.8,$10> is better than <0.7,$15>
- Two total orders, but the resulting order is partial:
  - <0.6, $10> and <0.4,$5> are not comparable

Typical questions

- Find an optimal solution
  - Difficult: NP-hard
- Is t an optimal solution?
  - Difficult: NP-hard
- Is t better than t'?
  - Easy: Linear in the number of constraints

How to find optimal solutions

- Classical constraints:
  - depth-first backtracking search
  - Constraint propagation
  - Variable elimination
  - Local search
  - ...
- Is it possible to extend/adapt these techniques to soft constraints?

Branch and bound

- Same scheme as for optimization CSPs
- Lower bound = preference of best solution so far (0 at the beginning)
- Upper bound for each node
- If ub > lb ➔ do not enter the subtree

Upper bounds

- Based on preferences
- t = assignment to past variables (P)
- ub1= x c(t), with c already assigned by t
- ub2 = x ub(cji), where xi in F, xj in P
  - ub(cji) = +cji(vj,a) for a in Di
- ub3 = x (+ cij(a,b))
  - x over xi, xj in F
  - + over a in Di, b in Dj
- Ub = ub1 x ub2 x ub3

Upper bounds weighted constraints (min +)

- t = assignment to past variables
- ub1= + c(t), with c already assigned by t
- ub2 = + ub(cji), for xi in F, xj in P
  - ub(cji) = min cji(vj,a) for a in Di
- ub3 = + (min cij(a,b))
  - + over xi, xj in F
  - min over a in Di, b in Dj
- ub = ub1 + ub2 + ub3

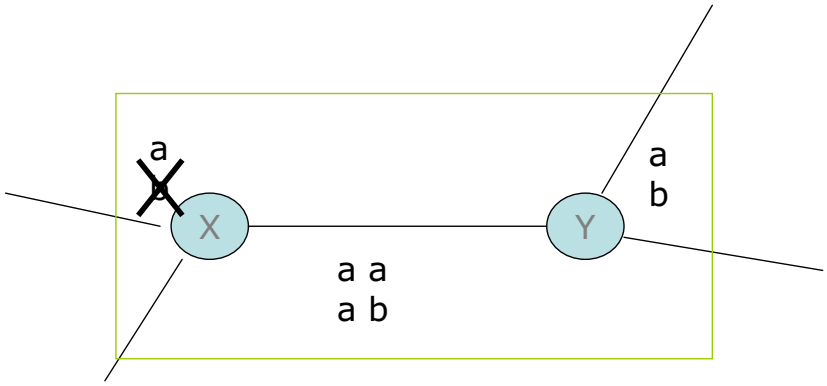
Upper bounds fuzzy constraints (max min)

- t = assignment to past variables
- ub1= min c(t), with c already assigned by t
- ub2 = min ub(cji), for xi in F, xj in P
  - ub(cji) = max cji(vj,a) for a in Di
- ub3 = min (max cij(a,b))
  - min over xi, xj in F
  - max over a in Di, b in Dj
- ub = min (ub1, ub2, ub3)

Constraint propagation

- Constraint propagation (ex.arc-consistency):
  - Deletes an element a from the domain of a variable x if, according to a constraint between x and y, it does not have any compatible element b in the domain of y
  - Iterate until stability
- Polynomial time
- Very useful at each node of the search tree to prune subtrees

Example



No matter what the other constraints are,
X=b cannot participate in any solution.
So we can delete it without changing the set of solutions.

Properties

- Equivalence: each step preserves the set of solutions
- Termination (with finite domains)
- Order-independence

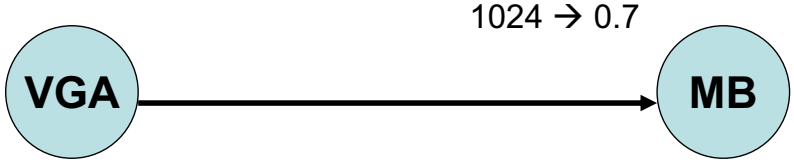
Soft constraint propagation

- Deleting a value means passing from 1 to 0 in the semiring <{0,1},or,and,0,1>
- In general, constraint propagation can change preferences to lower values in the ordering
- Soft arc-consistency: given c_x, c_{xy}, and c_y, compute $c_x := (c_x \times c_{xy} \times c_y)|_x$
- Iterate until stability

Example: fuzzy arc-consistency

s=slow → 0.2
f=fast → 0.9

256 → 0.5
512 → 0.8
1024 → 0.7



$c_x := (c_x \times c_{xy} \times c_y)|_x$

Fuzzy semiring=
<[0,1],max,min,0,1>
→ +=max an x=min

<s,256> → 0.6 <f,256> → 0.1
<s,512> → 0.7 <f,512> → 0.9
<s,1024> → 0.9 <f,1024> → 1

VGA=s → max(min(0.2,0.6,0.5),min(0.2,0.7,0.8),min(0.2,0.9,0.7))= max(0.2,0.2,0.2) = 0.2

VGA=f→ max(min(0.9,0.1,0.5),min(0.9,0.9,0.8),min(0.9,1,0.7))= max(0.1,0.8,0.7)=0.8

Properties

- If x idempotent (ex.:fuzzy,classical):
  - Equivalence
  - Termination
  - Order-independence
- If x not idempotent (ex.: weighted CSPs, prob.), we could count more than once the same constraint ➔ we need to compensate by subtracting appropriate quantities somewhere else ➔ we need an additional property (fairness=presence of -)
  - Equivalence
  - Termination
  - Not order-independence

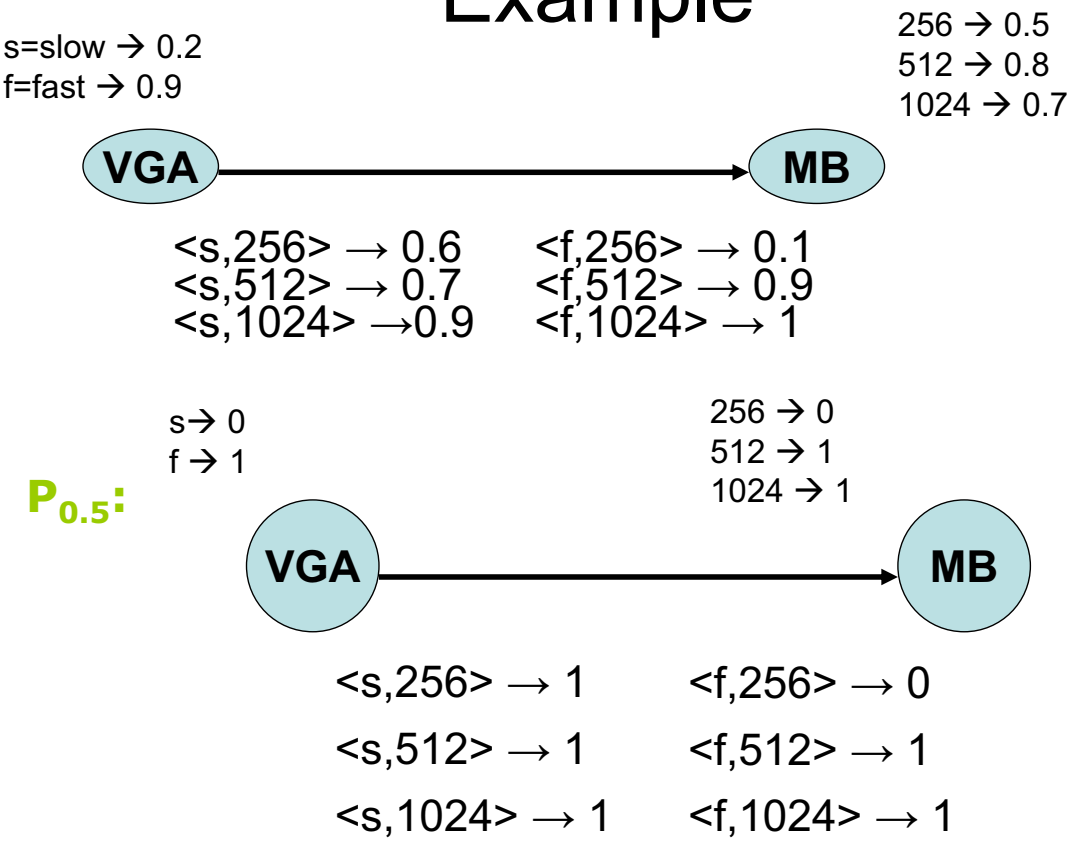
Branch and bound with soft constraint propagation

- Soft constraint propagation lowers the preferences
- ub1, ub2, ub3 can be lower
- more probable that ub < lb
- more probable to prune

Cuts

- Given P soft, and given α in the semiring, build P_α :
  - Same graph as P
  - In each constraint, only the tuples with preference > α

Example



Properties

- t optimal solution of P with preference v
  - For each α<v, P_α consistent and t solution of P_α
  - For each $\alpha \geq v$, P_α inconsistent
- Hold if x idempotent

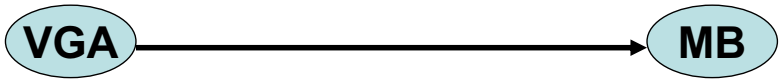
Solution method

- To find the optimal solutions of P:
  - Find the consistent P_α such that, for each $\beta > \alpha$, P_β inconsistent
- The solutions of P_α are the optimal solutions of P
- Just check the P_α with α occurring in P ➔ $O(kd^2)$ calls to a classical constraint solver
  - k= number of constraints
  - d = domain size
- Binary search ➔ $O(\log(kd^2))$ calls

Example

s=slow → 0.2
f=fast → 0.9

256 → 0.5
512 → 0.8
1024 → 0.7



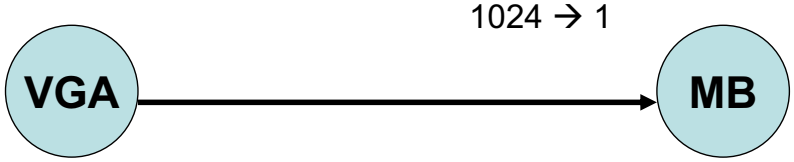
<s,256> → 0.6 <f,256> → 0.1
<s,512> → 0.7 <f,512> → 0.9
<s,1024> →0.9 <f,1024> → 1

Optimal solution: VGA=f,MB=512 (val=0.8)

$P_{0.5}$:

s→ 0
f → 1

256 → 0
512 → 1
1024 → 1



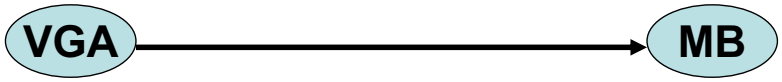
<s,256> → 1 <f,256> → 0
<s,512> → 1 <f,512> → 1
<s,1024> → 1 <f,1024> → 1

VGA=f,MB=512 is a solution, but also VGA=f,MB=1024 is a solution

Example

s=slow → 0.2
f=fast → 0.9

256 → 0.5
512 → 0.8
1024 → 0.7



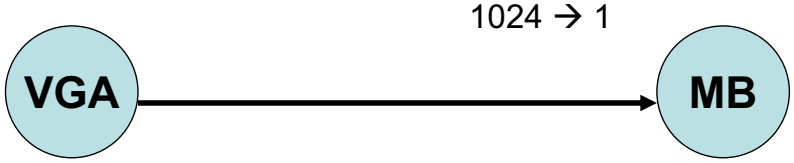
<s,256> → 0.6 <f,256> → 0.1
<s,512> → 0.7 <f,512> → 0.9
<s,1024> →0.9 <f,1024> → 1

Optimal solution: VGA=f,MB=512 (val=0.8)

$P_{0.6}$:

s→ 0
f → 1

256 → 0
512 → 1
1024 → 1



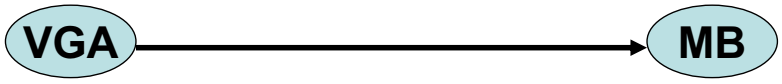
<s,256> → 0 <f,256> → 0
<s,512> → 1 <f,512> → 1
<s,1024> → 1 <f,1024> → 1

VGA=f,MB=512 is a solution, but also VGA=f,MB=1024 is a solution

Example

s=slow → 0.2
f=fast → 0.9

256 → 0.5
512 → 0.8
1024 → 0.7



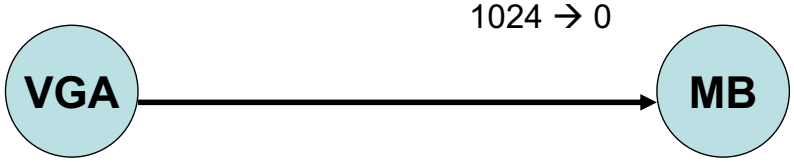
<s,256> → 0.6 <f,256> → 0.1
<s,512> → 0.7 <f,512> → 0.9
<s,1024> →0.9 <f,1024> → 1

Optimal solution: VGA=f,MB=512 (val=0.8)

$P_{0.7}$:

s→ 0
f → 1

256 → 0
512 → 1
1024 → 0



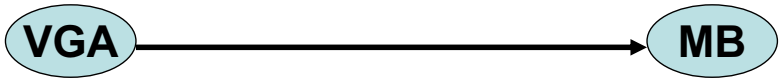
<s,256> → 0 <f,256> → 0
<s,512> → 0 <f,512> → 1
<s,1024> → 1 <f,1024> → 1

Only VGA=f,MB=512 is a solution

Example

s=slow → 0.2
f=fast → 0.9

256 → 0.5
512 → 0.8
1024 → 0.7



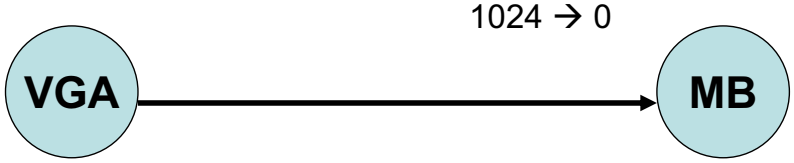
<s,256> → 0.6 <f,256> → 0.1
<s,512> → 0.7 <f,512> → 0.9
<s,1024> →0.9 <f,1024> → 1

Optimal solution: VGA=f,MB=512 (val=0.8)

$P_{0.8}$:

s→ 0
f → 1

256 → 0
512 → 0
1024 → 0



<s,256> → 0 <f,256> → 0
<s,512> → 0 <f,512> → 1
<s,1024> → 1 <f,1024> → 1

No solution

Variable elimination

- Generalization of adaptive consistency to soft constraints
- Choose a linear ordering of the variables: x1,...,xn
- From xn to x1, take xi:
  - Combine the constraints over xi U frontier(xi)
  - Project this constraint over frontier(xi)
  - Add the constraint to the CSP
- At the end, the highest preference of x1 is the preference of the optimal solutions
- An optimal solution can be found by instantiating x1,...,xn taking for each variable an optimal value from its domain which is compatible with the values chosen for the previous variables

Complexity

- As many steps as the number of variables (n)
- At each step, time exponential in the size of Y plus 1 (and space exponential in size of Y)
- n steps to find an optimal solution
- Time: O(n x exp(|Y|) +n)
- But space is the main problem with this method

Other issues

- Global constraints (Regin '94)
  - Specific constraints that occurr often in practice, and specific efficient propagation algorithms for them
- Symmetry breaking
- Local search

References

- [Apt 2003]: K. R. Apt, Principles of Constraint Programming, Cambridge University Press, 2003.
- [Bistarelli,Montanari,Rossi 97]: S. Bistarelli, U. Montanari. F. Rossi, Semiring-based constraint satsifaction and optimization, Journal of the ACM, 44, 2, 1997.
- [Dechter 2003]: R. Dechter, Constraint processing, Morgan Kaufmann, 2003.
- [Dechter,Pearl 87]:R. Dechter, J. Pearl, Network-based heuristics for constraint satisfaction problems, Artificial Intelligence, 34, 1987.
- [Mackworth 77]: A. Mackworth, Consistency in networks of relations, Artificial Intelligence, 8, 1, 1977.

References

- [Mackworth, Freuder 85]: A. Mackworth, E. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, Artificial Intelligence, 25, 1985.
- [Meiri, Dechter, Pearl 90]: I. Meiri, R. Dechter, J. Pearl, Temporal constraint ntworks, Artificial Intelligence, 49, 1990.
- [Montanari 74]: U. Montanari, Networks of constraints: fundamental properties and applications to picture processing, Information Science, 7, 66, 1974.
- [Regin 94]: J.-C. Regin, A filtering algorithm for constraints of difference in CSPs. Proc. AAAI94, 1994.

Simple Temporal Problems

(Dechter,Meiri,Pearl ’91)

Overview

- Temporal Constraint Satisfaction Problems
- Simple Temporal Problems
- Solving STPs
  - Floyd Warshall's algorithm
  - Path consistency

Temporal Constraint Satisfaction Problems

- A TCSP is CSP where
  - Variables → event times
  - Constraints → temporal relations
- Main frameworks
  - Qualitative Point Constraints (Vilain,Kautz, van Beek '89)
    - Relations → Point algebra *{Ø,<,=,>,≥,≤,?,≠}*
    - IxTeT, TimeGraph-II
  - Qualitative Interval Constraints (Allen'83)
    - Variables → time intervals
    - Relations → Allen's Algebra *{before, after, meets, met by, overlaps, overlaps-by, during, contains, equals, starts, started_by, finishes, finished_by}*

Metric Point Temporal Constraints

(Dechter, Meiri, Pearl '91)

- The ones we will consider
- Quantitative
- Variables → timepoints
- Relations → quantitative constraints
- From now on: TCSP = Metric Point TCSP

TCSP formal definition

- Set of **variables** representing timepoints
  $\{X_1, \ldots, X_k\}$
- Each variable has a discrete or continuous **domain** $D(X_i)$
- **Constraint** → set of intervals $I=\{[a_1,b_1],\ldots,[a_k,b_k]\}$
  - Unary: on X_i, $(a_1 \le v_i \le b_1)$ v … v $(a_k \le v_i \le b_k)$, vi $\in D(X_i)$
  - Binary: on (X_i,X_j) , $(a_1 \le v_j - v_i \le b_1)$ v … v $(a_k \le v_j - v_i \le b_k)$, $v_i \in D(X_i)$ $v_j \in D(X_j)$
  - In practice: new variable 'beginning of the world' X_0, $D(X_0)=\{0\}$, and every unary constraint on X_i is rewritten as binary constraint (X_0,X_i).

Example TCSP

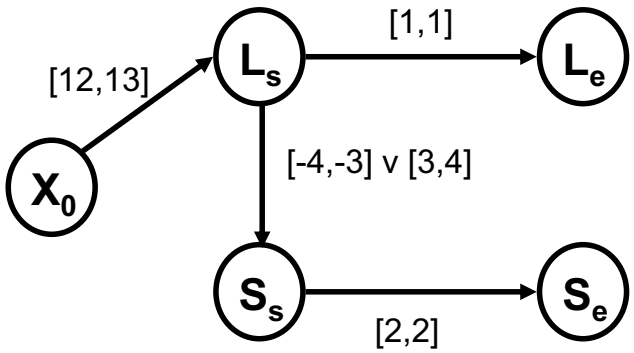
- Alice will go to **lunch any time between noon and 1pm** and usually **stays at lunch for 1 hour**. She can **go swimming for two hours either 3 to 4 hours before lunch,** since then she must shower and drive home**, or 3 to 4 hours after lunch**, since it is not safe to swim too soon after a meal.

- Variables $\{X_0, L_s, L_e, S_s, S_e\}$
- Constraints:
  - $T_{0Ls}=[12,13]$,
  - $T_{LsLe}=[1,1]$,
  - $T_{LsSs}=[-4,-3] \vee [3,4]$,
  - $T_{SsSe}=[2,2]$

Constraint Graph

- Variable (variable name) → Node (label)
- Constraint (X_i, X_J) → Directed edge from X_i to X_J labeled with the intervals

Constraint graph of the example:



TCSP formal definitions (2)

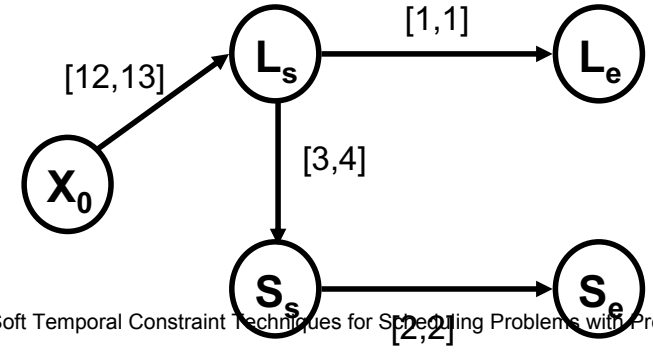
- **Solution** $S=\{v_1,\ldots,v_k\}$ $v_i \in D(X_i)$ consistent with all assignements
- A value $v_i \in D(X_i)$ is **feasible** iff it belongs to at least a solution
- **Minimal domain**: set of all feasible values
- **Minimal constraint** (X_i,X_J): set of feasible values for X_J-X_j
- **Minimal TCSP**: if all its constraints are minimal
- **Decomposable TCSP**: any consistent partial assignment can be extended to a complete solution
- **The class of TCSPs is NP-hard**

Simple Temporal Problems

(Dechter, Meiri, Pearl '91)

- STPs are TCSPs with a single interval on each constraint
- STP constraint on (X_i,X_J) is $T_{iJ}=[a,b]$, meaning $(a \le v_J - v_i \le b)$, $v_i \in D(X_i)$ $v_J \in D(X_J)$

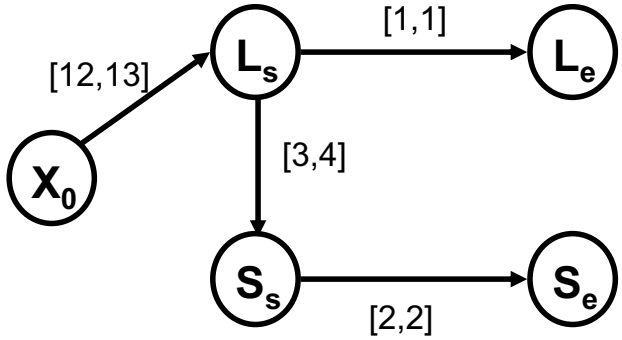
- Example: Alice will go to **lunch any time between noon and 1pm** and usually **stays at lunch for 1 hour**. She can **go swimming for two hours from 3 to 4 hours after lunch**.



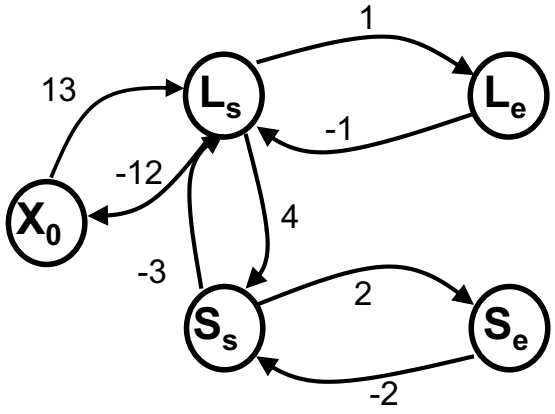
Solving STPs with the Shortest Path Algorithm

- Distance graph of an STP
  - Variables → Nodes
  - Constraint $T_{IJ}=[a,b]$ → two edges:
    - One from X_i to X_J labeled with b
    - One from X_J *to* X_i labeled with $-a$
- Distance matrix of an STP
  - n x n matrix F (n = number of variables),
  - Element *f[i][J]* = label of edge $X_i \rightarrow X_J$
  - Element *f[i][i]=0*, for every *i*
- An STP is consistent iff its distance graph

Example



Constraint Graph



Distance Graph

Distance matrix

	X_0	Ls	Le	Ss	Se
X_0	0	13			
Ls	-12	0	1	4	
Le		-1	0		
Ss		-3		0	2
Se				-2	0

Floyd-Warshall All Pairs Shortest Path Algorithm

1. Input distance matrix n x n
2. For k=1 to k=n
3. For i,j=1 to n
4. *f[i][J]=min{f[i][J],f[i][k]+f[k][J]}*

The distance graph corresponding to the input matrix is acyclic iff after FW, for all i, f[i][i] ≥0

Complexity $O(n^3)$

Example

Before Floyd Warshall

	X_0	Ls	Le	Ss	Se
X_0	0	13	∞	∞	∞
Ls	-12	0	1	4	∞
Le	∞	-1	0	∞	∞
Ss	∞	-3	∞	0	2
Se	∞	∞	∞	-2	0

After Floyd Warshall

	X_0	Ls	Le	Ss	Se
X_0	0	13	14	17	19
Ls	-12	0	1	4	6
Le	-13	-1	0	2	5
Ss	-15	-3	3	0	2
Se	-17	-5	-4	-2	0

The d-graph

- The d-graph is the graph corresponding to the matrix given in output by FW
- If the there are no negative cycles, then the STP corresponding to the d-graph is the minimal network of the original STP in input

- Two solutions of the STP are
  - The earliest: every variable X is assigned the lower bound of the interval on the constraint (X_0,X)
  - The latest : every variable X is assigned the upper bound of the interval on the constraint (X_0,X)

Example

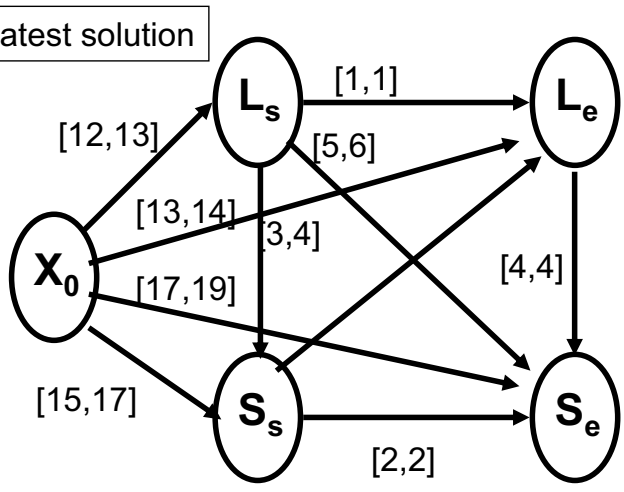
Distance Matrix After Floyd Warshall

	X_0	Ls	Le	Ss	Se
X_0	0	13	14	17	19
Ls	-12	0	1	4	6
Le	-13	-1	0	2	5
Ss	-15	-3	3	0	2
Se	-17	-5	-4	-2	0

Latest solution

Earliest solution

Minimal Network



Solving STPs by enforcing Path Consistency

- Operations on simple temporal constraints
  - Intersection $T^1_{iJ}=[a,b] \oplus T^2_{iJ}=[c,d]$ is a constraint $T^3_{iJ}=[a',b']=[a,b]\cap[c,d]$
  - Composition of $T_{ik}=[a,b] \otimes T_{kj}=[c,d]$ is a constraint $T_{iJ}=[a+c,b+d]$
- A constraint T_{iJ} is path consistent iff
  $$T_{iJ} \subseteq \oplus_{\forall k} (T_{ik} \otimes T_{kJ})$$
- An STP is path consistent if all its constraints are so
- An STP is consistent iff it is path consistent (not true for TCSPs)
- The STP network after path consistency enforcement is the minimal network

PC-2: path consistency enforcement

1. **Input**: STP P
2. Queue Q←{(i,j,k)| i<j, k≠i,j}
3. **while** Q≠Ø do
4. Select and delete a path (i,j,k) from Q
5. **if** $T_{iJ} \neq T_{iJ} \oplus (T_{ik} \otimes T_{kJ})$ **then**
6. $T_{iJ} \leftarrow T_{iJ} \oplus (T_{ik} \otimes T_{kJ})$
7. **if** T_{iJ} =Ø **then exit** (inconsistency)
8. Q← Q ∪ {(i,j,k)| 1≤k≤n, k≠i,j}
9. **end-if**
10. **end-while**

Complexity: $O(n^3r)$ relaxations, $O(n^3r^3)$ arithmetic operations

n= number of variables, r = max size of an interval

Simple Temporal Problems with Preferences

(Khatib,Morris,Morris,Rossi ’01)

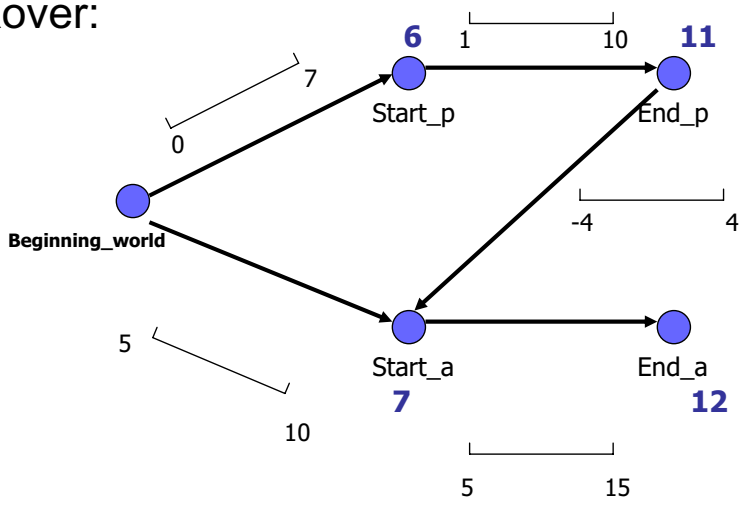
Overview

- Simple Temporal Problems with Preferences
- Tractable Subclass
- Two Solvers for Fuzzy optimal Solutions
- A solver for Pareto optimal Solutions
- Learning local temporal preferences from preferences on solutions
- Utilitarian Optimals
- Disjunctive temporal Problems with Preferences

Simple Temporal Constraints: An Example

Two activities of Mars Rover:

- Taking pictures:
  - 1 < duration < 10
  - 0 < start < 7
- Analysis:
  - 5 < duration < 15
  - 5 < start < 10

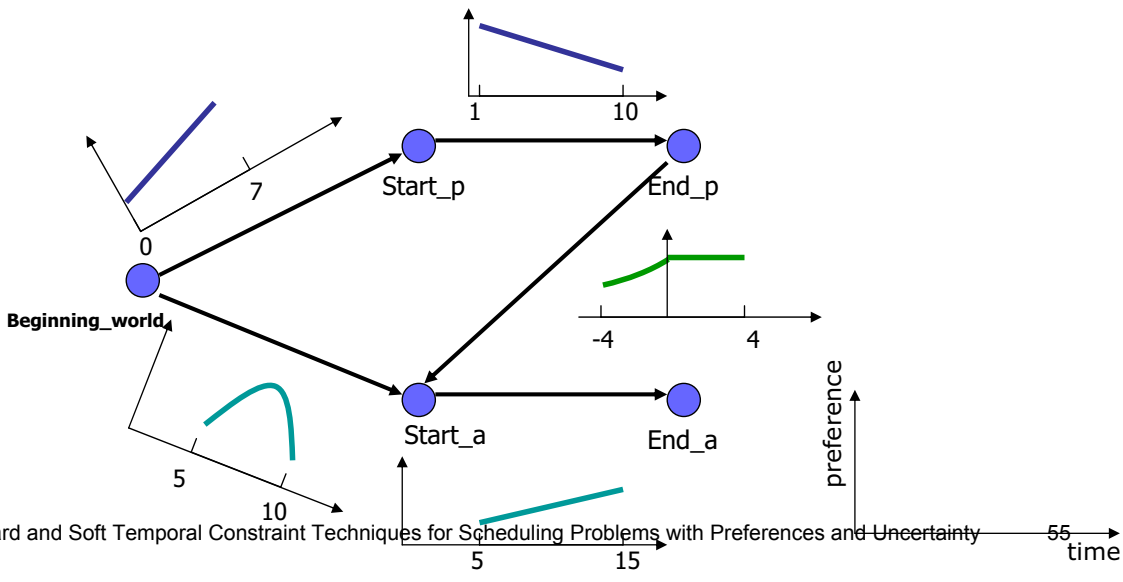


- Additional constraint:
  - -4 < start analysis – end pictures < 4

Introducing preferences

Sometimes hard constraints aren't expressive enough. We may think that:

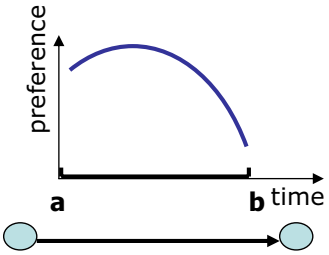
- It's better for the picture to be taken as late as possible and as fast as possible.
- It's better if the analysis starts around 7 and lasts as long as possible.
- It's ok if the two activities overlap but it's better if they don't.



STPP Formalism

- Simple Temporal Problem with Preferences
  - Simple Temporal Problem
    - Set of variables X1,…,Xn;
    - Constraints T={I}, I=[a,b] a<=b;
      - Unary constraint T over variable X : a<=X<=b;
      - Binary constraint T over X and Y: a<=X-Y<=b;
  - C-semiring S=<A, +, x, 0, 1>
    - A set of preference values
    - + compares preference values inducing the ordering on A
    - a<=b if a+b=b , a,b in A
    - x composes preference values

- **Simple Temporal Constraint with Preferences**
  - Binary constraint
  - Interval I=[a, b], a<=b
  - Function f: I ⟶ A



What does solving an STPP mean?

- A **solution** is a complete assignment to all the variables consistent with all the constraints.
- Every solution has a **global preference value** induced from the local preferences.
- **Solving** an STPP means finding an optimal consistent solution , where **optimal** means its global preference is best.

Intractability

The class of STPPs is NP-hard.

Proof: Any TCSP can be reduced to an STPP on the S_{CSP}=<{1,0},v,∧,1,0> semiring in the following way:

For every hard constraint I={$[a_1,b_1]$,…,$[a_k,b_k]$}

write the soft constraint <I',f>

I'=[a1,bk],

f(x) =1 iff ∃j such that x ∈ $[a_J,b_J]$

S is a solution of the TCSP iff it is an optimal solution of the STPP

Tractability conditions

Simple Temporal Problems with Preferences are tractable if:

1) the underlying semiring has an **idempotent** multiplicative operator (**x**).

   For example:

   Fuzzy Semiring **<{x| x in [0,1]}, max, min, 0, 1>**

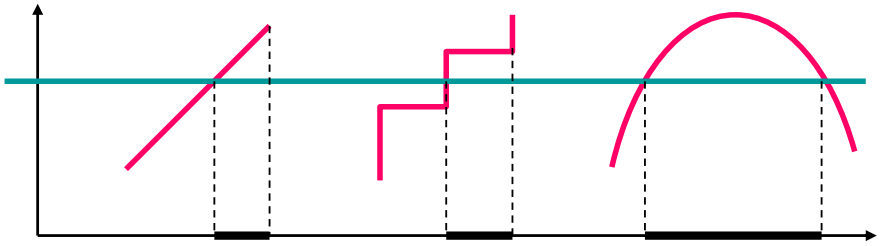
2) the preference functions are **semi-convex**
3) the set of preferences is **totally ordered**

Semi-convex Functions

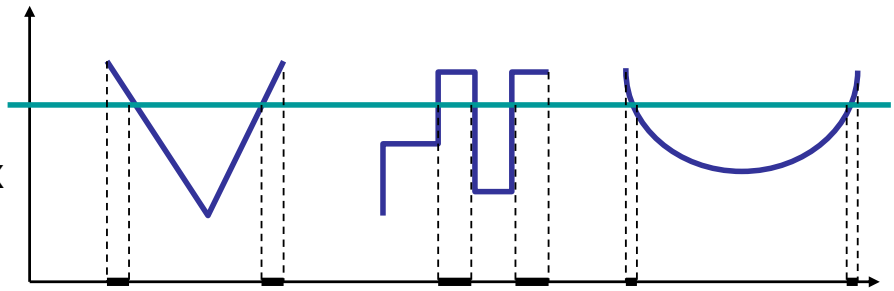
$\forall y\{x \mid f(x) \geq y\}$ is an interval

Examples

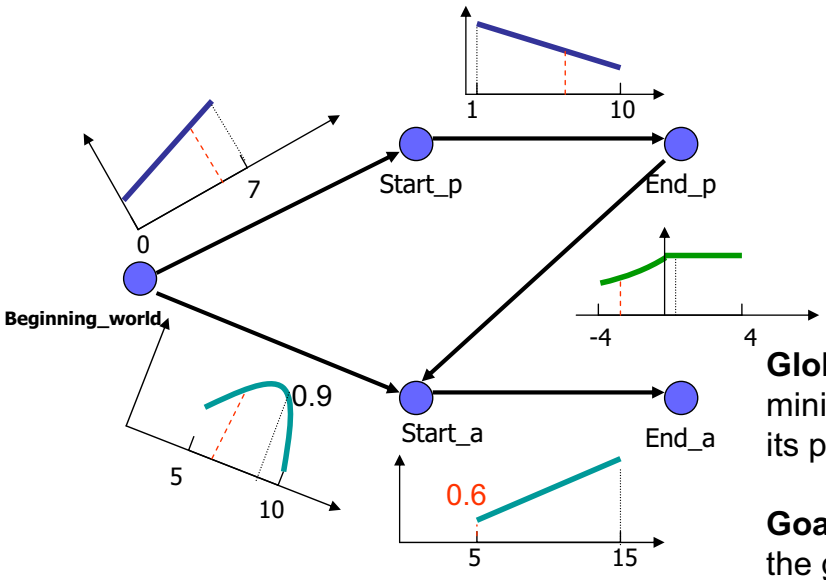
Semi-convex



Not Semi-convex



Solutions of the Rover Example



Fuzzy Semiring
<[0,1], max, min, 0, 1>

Global preference of a solution: minimum of the preferences of its projections

Goal: maximize the global preference

Start_p = 5 End_p= 11 Start_a= 7 End_a=12 global preference =0.6

Start_p = 7 End_p= 8 Start_a= 9 End_a=24 global preference =0.9 ← BEST

Path consistency with preferences

- As with hard constraints, two operations os temporal constraints with preferences:
  - Intersection
  - Composition

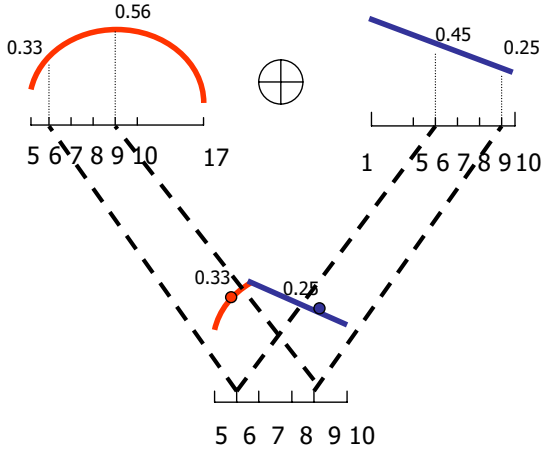
Intersection of Soft Temporal Constraints

If $T_1 = \langle I_1, f_1 \rangle$ *and* $T_2 = \langle I_2, f_2 \rangle$ *defined on* X_i *and* X_j
then $T_1 \oplus T_2 = \langle I_1 \oplus I_2, f_1 \oplus f_2 \rangle$ *is defined on* X_i *and* X_j *and*
$I_1 \oplus I_2 = I_1 \cap I_2$ *and*

$$f_1 \oplus f_2(a) = f_1(a) \times f_2(a) \overset{fuzzy}{=} \min\left(f_1(a), f_2(a)\right)$$

a=6 $\quad$ *min(0.33,0.45)= 0.33*

a=9 $\quad$ *min(0.56,0.25)= 0.25*



Composition of Soft Temporal Constraints

$$If\ \ T_{ik} = \langle I_{ik}, f_{ik} \rangle\ and\ \ T_{kj} = \langle I_{kj}, f_{kj} \rangle\ then$$

$$T_{ij} \otimes T_{kj} = \langle I_{ik} \otimes I_{kj}, f_{ik} \otimes f_{kj} \rangle \quad is\ defined\ \ on\ \ X_i\ and\ \ X_j\ and$$

$$I_1 \otimes I_2 = \{ a = r_1 + r_2 \mid r_1 \in I_1 \ \ r_2 \in I_2 \} and$$

$$f_1 \otimes f_2(a) = \sum \{ f_1(r_1) \times f_2(r_2) \mid a = r_1 + r_2 \}^{fuzzy} = \max \{ \min (f_1(r_1), f_2(r_2)) \mid a = r_1 + r_2 \}$$

If $a=8$

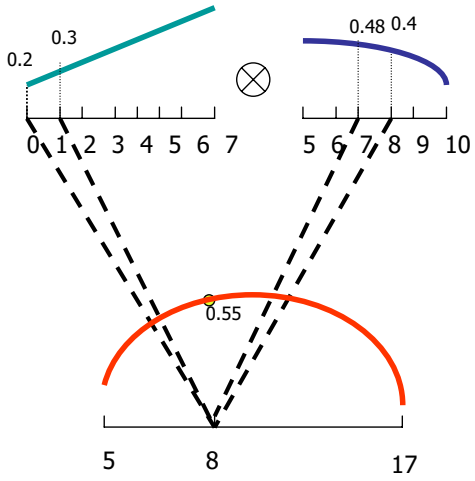
$r1=0$ $r2=8$ $min(0.2,0.4) = 0.2$

$r1=1$ $r2=7$ $min(0.3,0.48) = 0.3$

$r1=2$ $r2=6$ $min(0.4,0.52) = 0.4$

$r1=3$ $r2=5$ $min(0.6,0.55) = 0.55$

$max\{0.2,0.3,0.43,0.55\} = 0.55 = f1 \otimes f2\ (8)$



Path Consistency in STPPs

- A Soft Temporal Constraint, $T_{iJ,}$ is path consistent iff $T_{iJ} \subseteq \oplus \forall k\ (T_{ik} \otimes T_{kJ})$
- An STPP is path consistent if all its constraints are path consistent

Algorithm STPP_PC-2

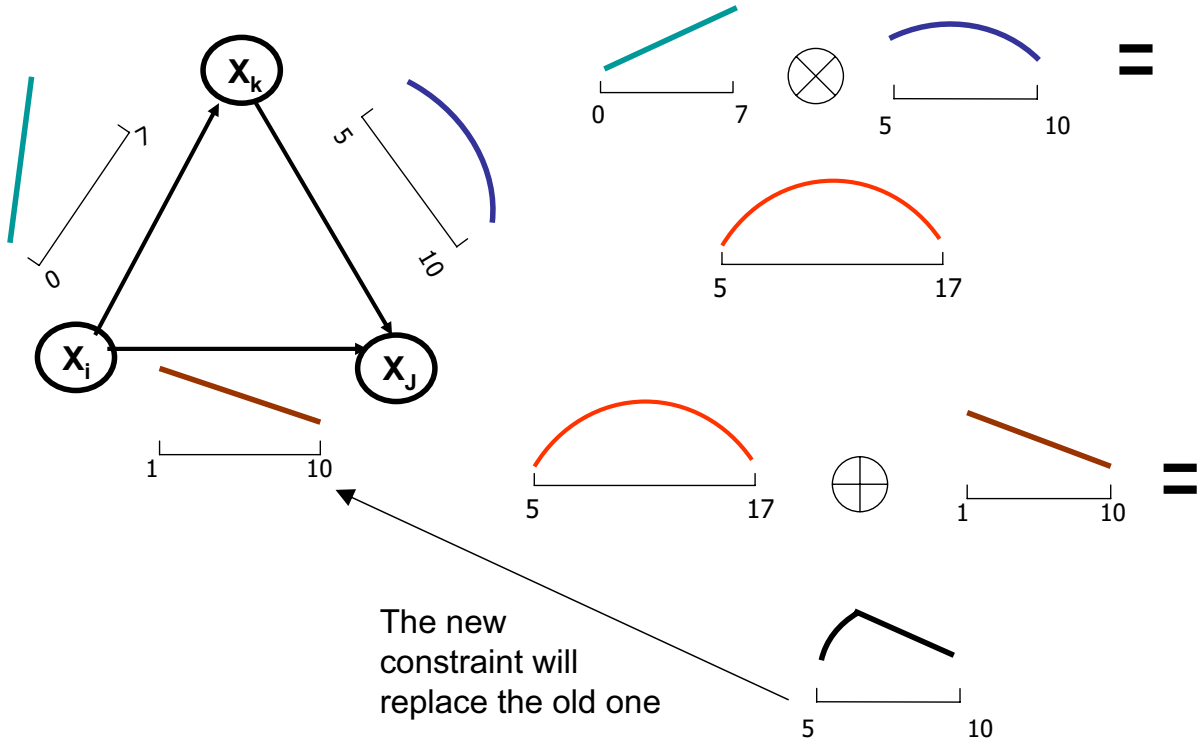
1. Input: **STPP** P
2. Queue Q←{(i,j,k)| i<j, k≠i,j}
3. while Q≠Ø do
4. Select and delete a path (i,j,k) from Q
5. if $T_{iJ} \neq T_{iJ} \oplus (T_{ik} \otimes T_{kJ})$ then
6. $T_{iJ} \leftarrow T_{iJ} \oplus (T_{ik} \otimes T_{kJ})$
7. if $T_{iJ} = \emptyset$ then exit (inconsistency)
8. Q←Q ∪ {(i,j,k)| 1≤k≤n, k≠i,j}
9. end-if
10. end-while
11. Output Path consistent **STPP**

As PC-2 except:

a) The input and the output are STPPs

b) ⊗⊕ are extended to preferences

Path Consistency on STPPs



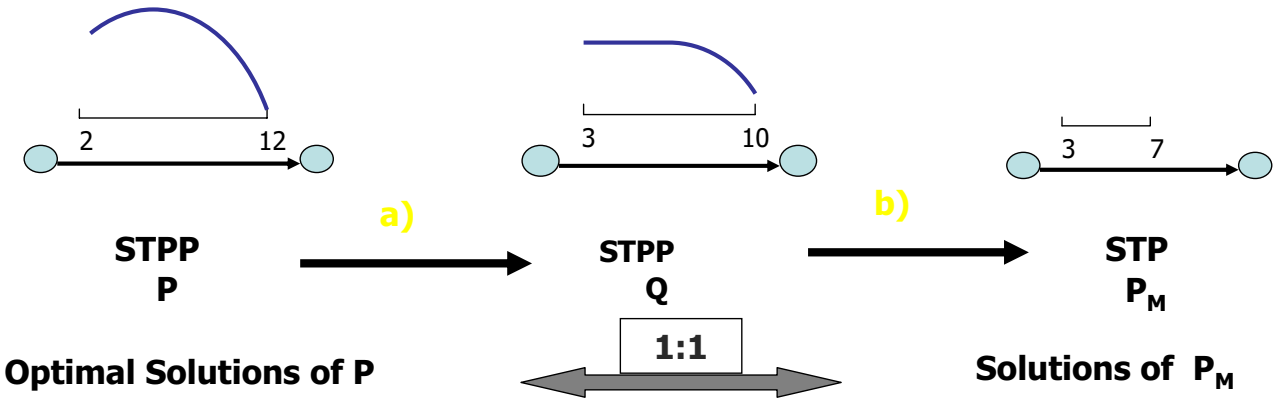
Solving tractable STPPs with path consistency

Given a tractable STPP, path consistency is sufficient to find an optimal solution without backtracking

- Proof:
- **Closure of semi-convex** functions under intersection and composition
- **After** enforcing **path consistency**, if no inconsistency is found, all the preference functions have the **same maximum preference level M**
- The subintervals mapped into M form an STP in minimal form such that **an assignment is a solution of the STP iff it is an optimal solution of the STPP**

Path-Solver

1. **Input** STPP P
2. STPP Q ←STPP_PC-2(P)
3. Build an STP P_M considering only intervals mapped into the best level of preference M
4. **Output** STP P_M



Complexity of Path-solver

- A tractable STPP can be solved in
- $O(n^3r^3)$ relaxations or more precisely
- $O(n^3r^4l)$ arithmetic operations

- n = number of variables
- r = max size of an interval
- l = number of different preference

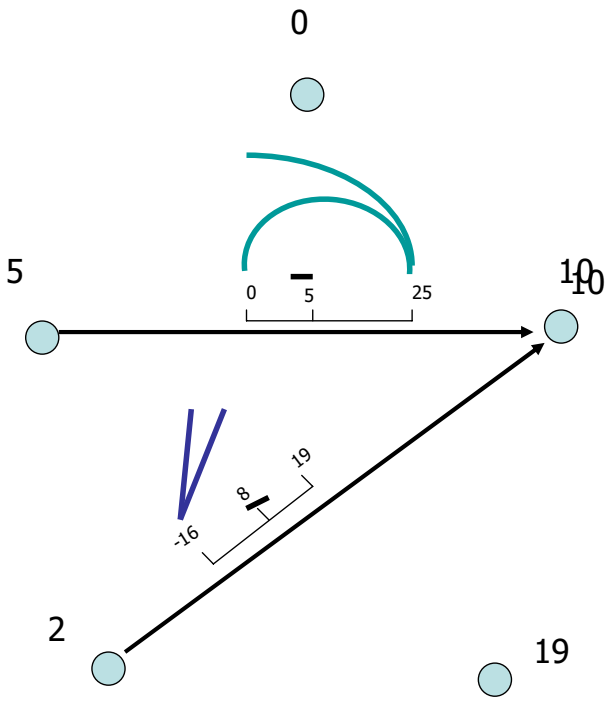
Random Generator of STPPs

- It generates an STPP on the fuzzy semiring and with semi-convex parabolas as preference functions.
  - Why semi-convex parabolas?...

    …Because:
    - they are easily parametrized
    - they are representative of many temporal relations (they include linear)
    - they will be useful for the learning module

A small simulation of the generator

- Parameters:
  - **n**=number of variables
    5
  - **r**=range of the first solution
    20
  - **d**=density
    20%
  - **max**= maximum expansion of the intervals
    25
  - **pa, pb** e **pc**= percentage of perturbation for parabolas
    20% 30% 10%



Experimental Results for Path-solver

Results on randomly generated problems with :

n=30

r=100

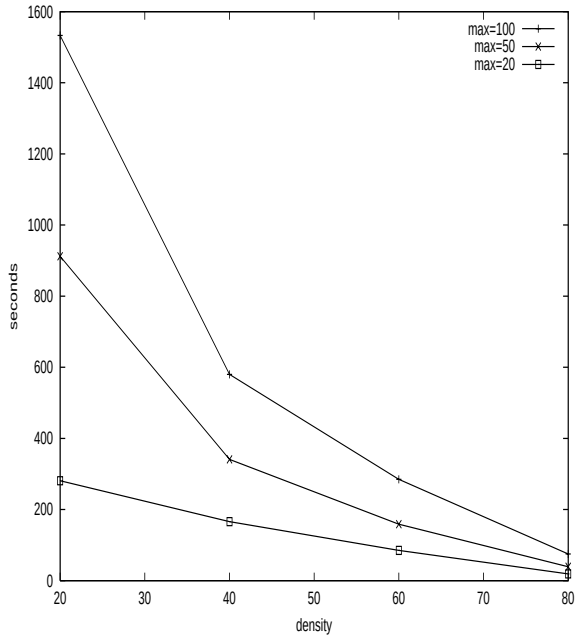
pa=20

pb=20

pc=30

x-axis : density

y-axis : seconds.

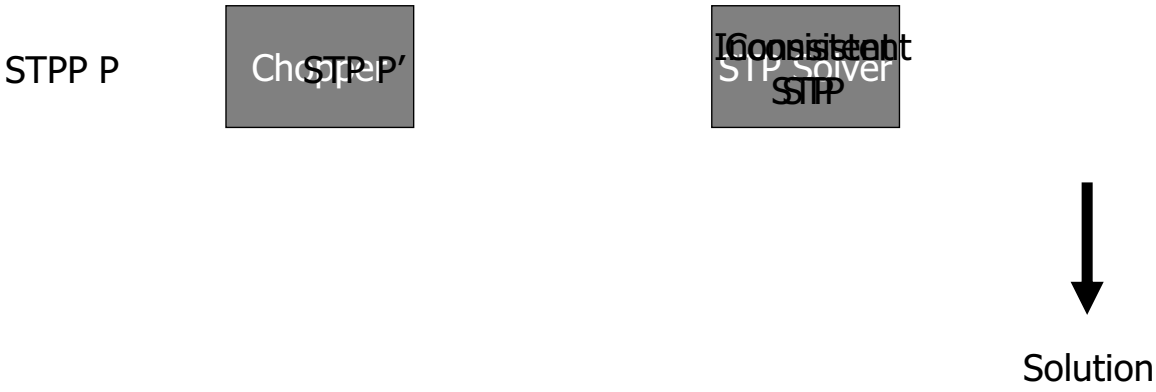


Solving STPPs with a decomposition approach

Given a tractable STPP and a preference level *y*, the intervals of elements with preference above *y* form an STP P_y.

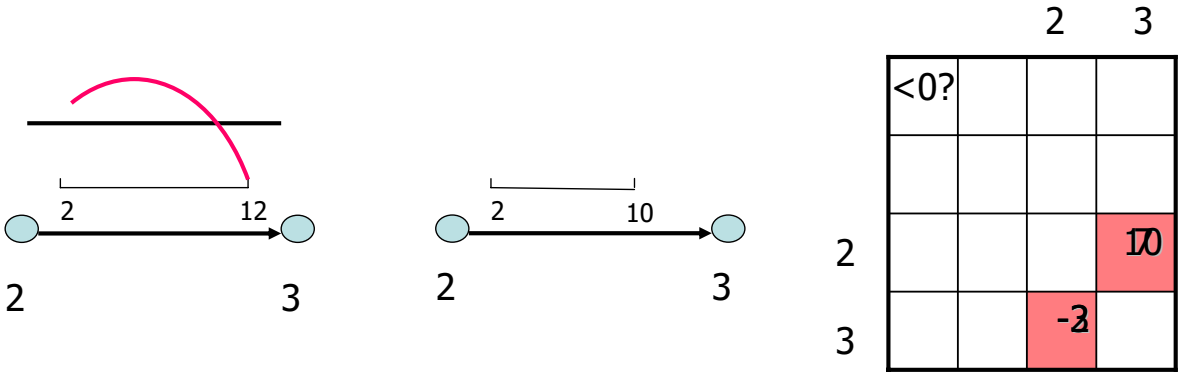
The highest level, *opt*, at which STP P_{opt} is consistent is such that an assignment is a solution of P_{opt} iff it is an optimal solution of the STPP

Basic Iteration of Chop-Solver



Simulation

1) CHOP! 2) SOLVE! 3)CONSISTENCY TEST



Floyd-Warshall's matrix

Chop-Solver Algorithm

```
1)   Input STPP P;
2)   Input Precision;
3)   Real lb=0, ub=1, y=0, n=0;
4)   If STP_y is consistent
5)      y=0.5, n=n+1;
6)      while n<=Precision
7)       if STP_y is consistent
8)         lb=y, y=y+(lb-ub)/2, n=n+1;
9)       else
10)        ub=y, y=y-(lb-ub)/2, n=n+1;
11)     end of while;
12)     return solution;
13)  else exit;
```

Chop-Solver performs a **binary search** of the highest level at which chopping the STPP gives a consistent STP
Precision is the number of steps allowed in the search

Complexity of Chop-solver

- A tractable STPP can be solved using Chop-solver in:
  - O(precision x n^3) if we use Floyd Warshall to solve STPs
  - O(precision x n^3 x r) if we use PC-2 to solve STPs

n =number variables

r = max size of the interval

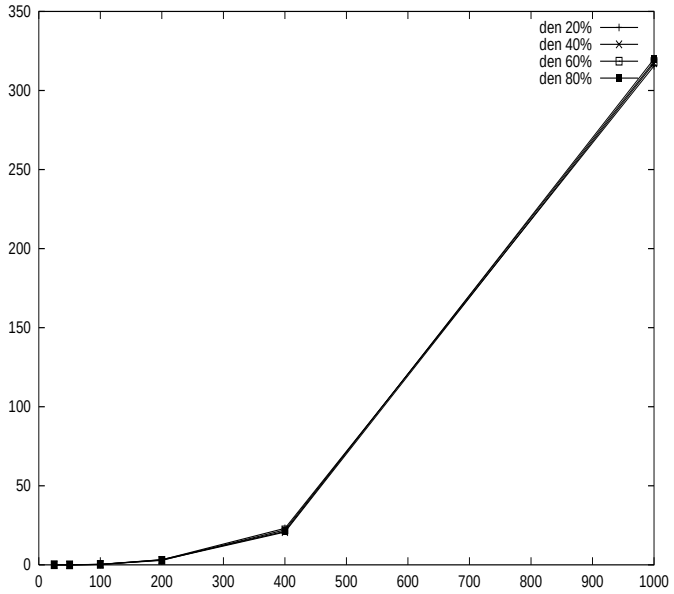
Experimental results for Chop-Solver

X-axis number of variables
Y-axis time in seconds

Fixed parameters:
Range of first solution:100000
Max expansion: 50000
Perturbation on a: 5%
Perturbation on b: 5%
Perturbation on c: 5%

Varying:
Density 20%, 40%, 60% ,80%

Mean on 10 examples



Path-solver vs Chop-solver

	Path-solver	Chop-solver
Constraint representation	discrete	continuous
Performance	slow	very fast

Time to solve a problem with 40 variables, r=100, max=50, pa=pb=10% and pc=5%

Density	Path-solver	Chop-solver
40%	1019.44 sec	0.03 sec
60%	516.24 sec	0.03 sec
80%	356.71 sec	0.03 sec

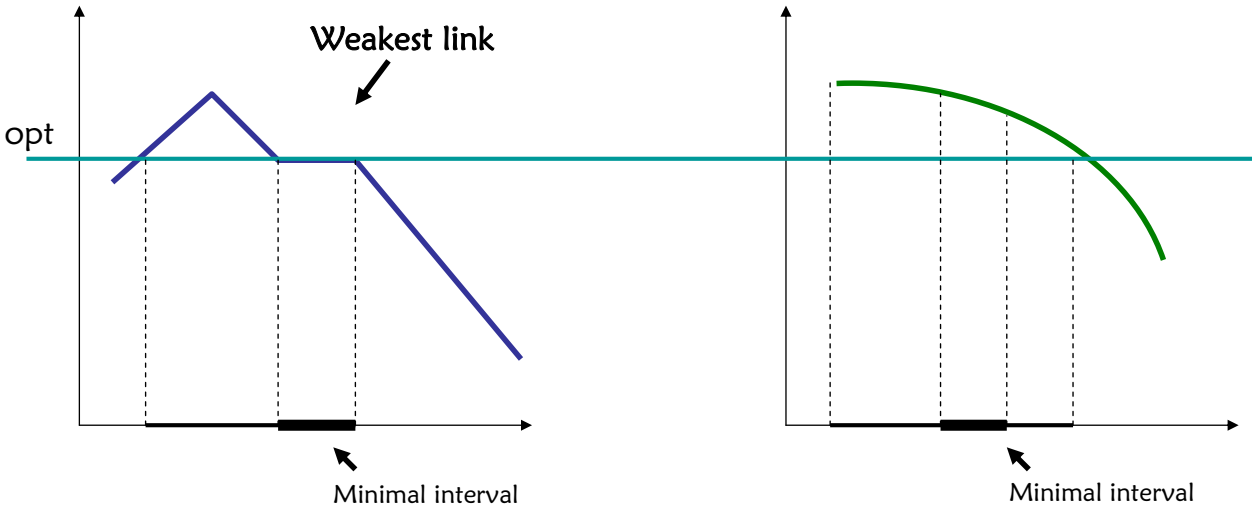
Representational Power	unlimited	limited but useful for learning

Pareto Optimal Solutions

(Khatib, Morris,Morris, Venable '03)

- The WLO+ algorithm finds a Pareto Optimal solution of an STPP
- After applying Chop Solver to the problem it identifies special constraints, the weakest links.
- It modifies the weakest links.
- It reapplies Chop solver.

Weakest Link



A constraint is a weakest link at optimal chopping level opt if the maximum reached by the preference function on the minimal interval is opt. So...

WLO+

1) Apply Chop solver and identify weakest links

2) While there are w.l.

3) Modify the w.l. :

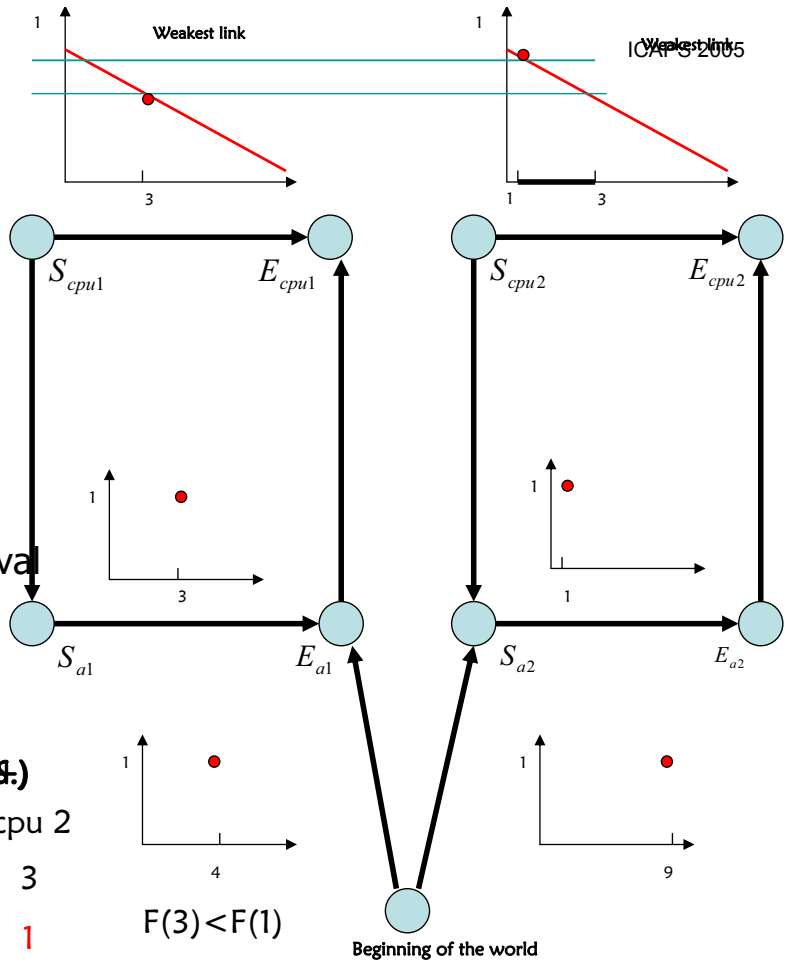
3a) interval fminimal interval

3b) pref. function f y=1

4) Apply Chop Solver

Pareto Optimal solutions (WLO+)

BEa1	BSa2	a1	a2	cpu1	cpu 2
4	9	3	1	3	3
4	9	3	1	3	1



Measures of global preferences

Solution	Complete assignment to variables	A set of projections on all constraints
s	$(s_1, s_2, \ldots, s_n)$	$(s{\downarrow}c_1, s{\downarrow}c_2, \ldots, s{\downarrow}c_k)$
t	$(t_1, t_2, \ldots, t_n)$	$(t{\downarrow}c_1, t{\downarrow}c_2, \ldots, t{\downarrow}c_k)$

Fuzzy s>t iff $\max [\min_i (f_{ci}(s{\downarrow}c_i)), \min_i (f_{ci}(t{\downarrow}c_i))]$

Pareto s>t iff $f_{ci}(s{\downarrow}c_i) \geq f_{ci}(s{\downarrow}c_i)$ for all i=1…k and exists j such that $f_{cj}(s{\downarrow}c_j) > fc_j(s{\downarrow}c_j)$

Utilitarian s>t iff $\sum_i (f_{ci}(s{\downarrow}c_i)) > \sum_i (f_{ci}(t{\downarrow}c_i))$

WLO+: Experimental Results (1)

x-axis: variables

y-axis: seconds

Fixed parameters:

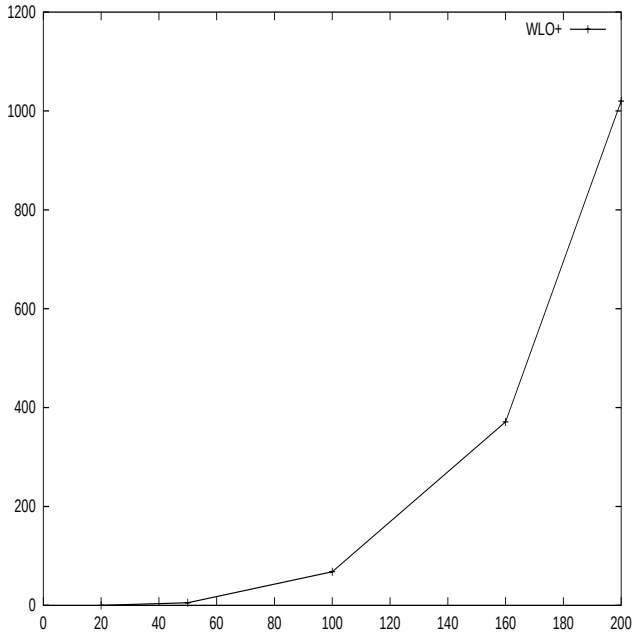
r=50, max=50,

density=no. of variables

Throughout all the experiments:

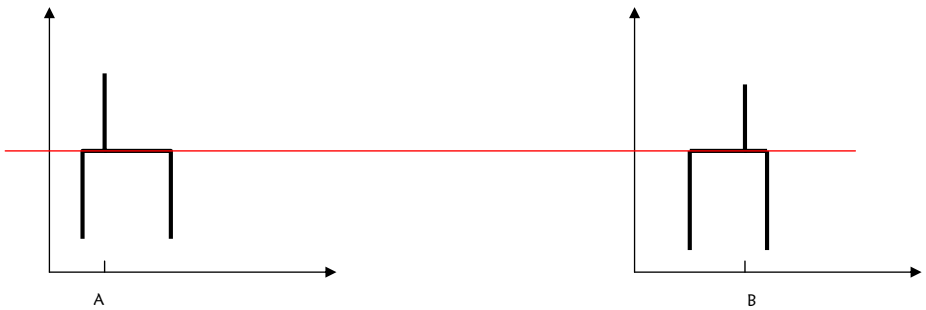
1) 10% of utilitarian improvement
2) 75% of wins

w.r.t. the earliest optimal solution returned by Chop Solver



What if there is no weakest link?

In some cases there is no weakest link.
Example:



Point A inconsistent with Point B

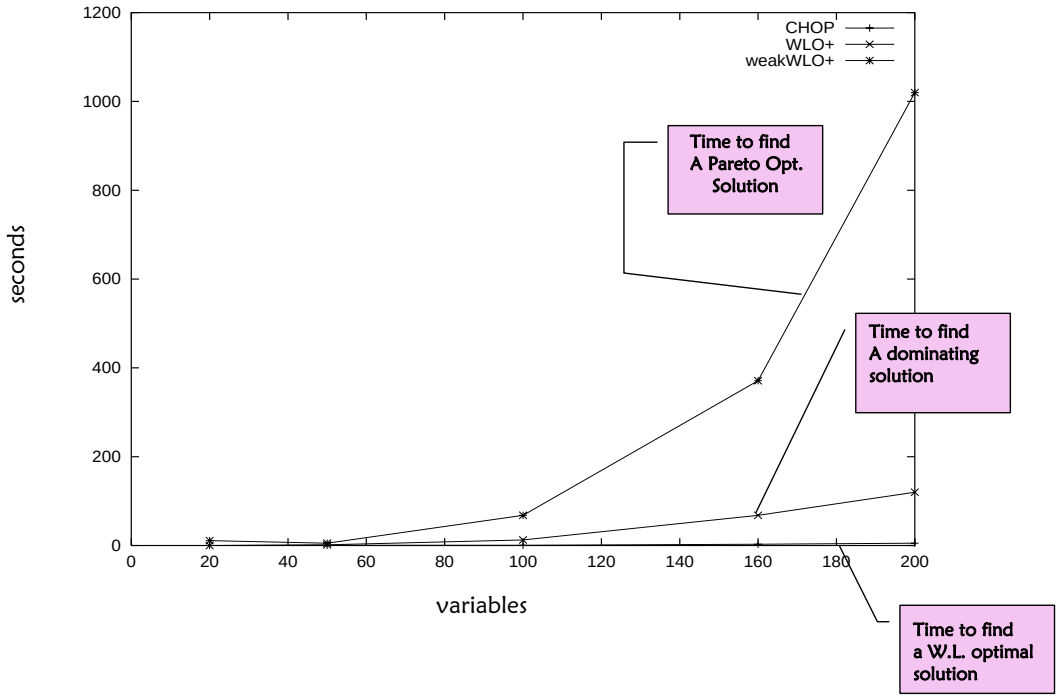
But: continuous domains + convex functions c existence of a weakest link

If the above don't hold:

Weakest link w.r.t. a particular solution: constraint on which the projection of the solution is mapped into chopping level.

Applying WLO+ with the new definition, weakWLO+ , guarantees to find a set of solutions that Pareto-dominate the initial one.

How good is the solution I'll get?



...it depends on how much time I have.

Stratigraphic Egalitarianism and Utilitarian Criteria

(P. Morris et al. '04)

- The set of solutions returned by WLO+ is a subset of the Pareto Optimal solutions
- It can be characterized by the **Stratigraphic Egalitarianism** criterion: given a solution *S* of an STPP let $u_S=<u_s^1,\ldots,u_S^m>$ be the associated vector of preferences (one for each constraint). Solution ***S* SE-dominates solution *S'*** iff for any preference level α:
  - $u_S^i < \alpha$ implies $u_S^i \geq u_{S'}^i$
  - $\exists i$ such that $u_S^i < \alpha$ implies $u_S^i > u_{S'}^i$
  - $u_S^i \geq \alpha$ implies $u_{S'}^i \geq \alpha$
- **Utilitarian criterion** is to maximize the sum of preferences
  - Finding Utilitarian optimal solutions of STPPs with piecewise linear preference functions is tractable
  - The problem is translated into a linear program

Disjunctive Temporal Problems with Preferences (Peintner,Pollack'04)

- Disjunctive Temporal Constraint:
  $(X_1\text{-}Y_1 \in [a_1,b_1]) \vee \ldots \vee (X_n\text{-}Y_n \in [a_n,b_n])$
- Disjunctive Temporal Constraint with Preferences:
  $(X_1\text{-}Y_1 \in [a_1,b_1], f_1) \vee \ldots \vee (X_n\text{-}Y_n \in [a_n,b_n],f_n), \quad f_i: [a_i,b_i] \rightarrow [0,1]$
- Fuzzy Optimization criterion
- Algorithm
  1. For each preference level project a DTP from the DTPP
  2. Search for a consistent DTP such that its solutions have the highest preference value
- Complexity |preferences| x n^3 x (DTP complexity), n=number of variables

Learning Simple Temporal Problems with Preferences

(Khatib et al '02)

Learning local from global

It can be difficult to have precise knowledge on the preference function for each constraint.

Instead it may be easier to be able to tell how good a solution is.

Global information
some solutions + global preference values

Local Information
shape of preference functions

Learning STPPs

- **Inductive Learning**: ability of a system to induce the correct structure of a map ***t*** known only for particular inputs
- **Example**: ***(x,t(x)).***
- **Computational task:** given a collection of examples (training set) return a function ***h*** that approximates ***t***.
- **Approach**: given an error function ***E(h,t)*** minimize modifying ***h***.
- In our context :
  - x → solution
  - t → rating on solutions given by expert
  - Preference function constraint c_i → parabola $a_i x^2+b_i x+c_i$
  - Error E → E($a_1,b_1,c_1,\ldots,a_n,b_n,c_n$)
  - Learning technique → gradient descent

Gradient Descent

- **Define** $h=h_W$ where W=set of internal parameters, thus E=E_W;
- **Initialize** W to small random values (t=0);
- **Update** W according to the Delta rule:

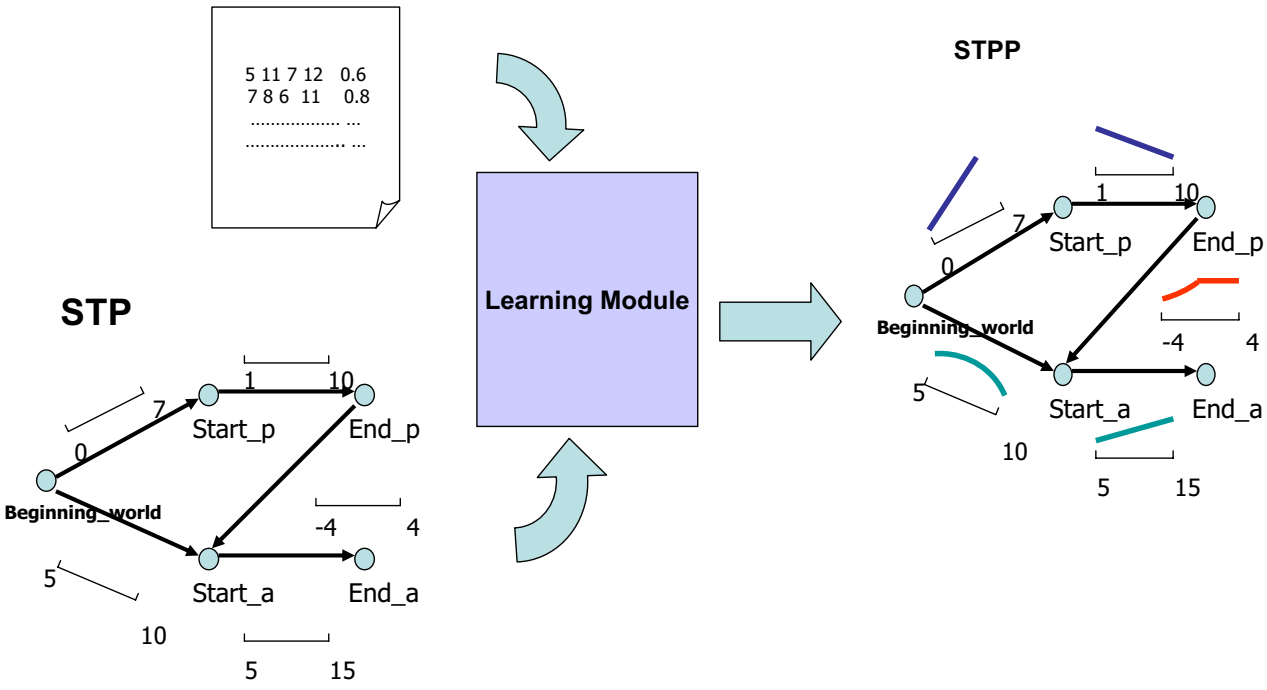
$$W(t+1) = W(t) + \Delta W(t)$$

$$\Delta W(t) = -\eta \frac{\partial E}{\partial W(t)}$$

- **Stop** when satisfied with the level of minimization reached;
- **Test** results on a set of new examples, test set.

The Learning Module

Training set



The Implemented Learning Module

- Works with
  - **Parabolas** ***f(x)=ax2+bx+c*** as preference functions
  - **Fuzzy Semiring** *<[0,1],max,min,0,1>* as underlying structure
  - **Smooth** version of the **min** function
- Performs
  - **Incremental gradient descent** on the sum of squares error

$$E = \frac{1}{2}\sum_{s \in T} (t(s) - h(s))^2$$

$t(s)$ Preference value of solution s in the training set

$h(s)$ Preference value guessed for solution s from the current network

The Learning Algorithm

1) Read a **solution s** and its preference value **t(s)** from the training set
2) Compute the preference value of s, **h(s),** according to the current network
3) Compare h(s) and t(s)
4) Adjust parameters a, b, c, of each preference function of each constraint, in order to make the error smaller
5) Compute the global error; if below threshold, exit, otherwise back to 1)

Adjustment of the parameters

- Delta rule:

$$\tilde{a}_i = a_i - \eta \frac{\partial E}{\partial a_i}(a_1, b_1, c_1, ..., a_v, b_v, c_v) \quad with \quad v = number\ of\ constr.$$

$$\tilde{b}_i = b_i - \eta \frac{\partial E}{\partial b_i}(a_1, b_1, c_1, ..., a_v, b_v, c_v)$$

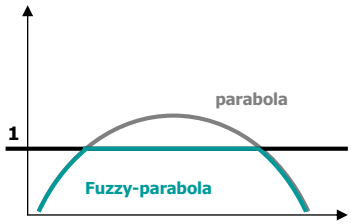
$$\tilde{c}_i = c_i - \eta \frac{\partial E}{\partial c_i}(a_1, b_1, c_1, ..., a_v, b_v, c_v)$$

- Semi-convexity is maintained during all the learning process

$$if \quad \tilde{a} < 0 \quad then \quad \tilde{a} = 0$$

Stopping the learning phase

- Parabolas or fuzzy-parabolas?
  ……both!
- Monitored errors on both kinds of parabolas:
  - Sum of squares error
  - Absolute maximum error
  - Absolute mean error



- Stop criterion:
  - 100 consecutive failure of improving of at least 70% the abs. mean error computed with fuzzy parabolas
- Errors computed on test set for final evaluation:
  - Sum of squares error
  - Absolute maximum error
  - Absolute mean error

Experiments

- ☑ Learning Problems where the **global preference** function is **min**
  - Randomly generated problems
- ☑ Learning problems where the **global preference** function is **not min**
  - Minimize makespan
  - Minimize maximum lateness
  - Minimize resource consumption

Experimental results on randomly generated problems

- Varying parameters:
  - density (D)
  - maximum range of interval expansion (max).
- Fixed parameters :
  - number of variables **n=25**
  - range for the initial solution **r=40**
  - parabolas perturbations **pa=10, pb=10** and **pc=5.**
- Displayed: **absolute mean error (0<ame<1)** on a test set (mean on 30 examples).
  - 357<=iterations<=3812
  - 2′ 31″<=time required<=8′ 18″

Density / Maximum Range	D=40	D=60	D=80	Number of examples of training and test set.
max=20	0.017	0.007	0.0077	500
max=30	0.022	0.013	0.015	600
max=40	0.016	0.012	0.0071	700

An example with maximum lateness

- Problem: **8 activities** to be scheduled in **24 hours**
- Given:
  - Duration intervals for each activity
  - Constraint graph
- Aim: Minimize the ending time of the last activity scheduled.

- Procedure:
  1) Solve the hard constraint problem: **900 solutions**
  2) Rate each solution with a function that gives higher preference to schedules that end sooner: **37 optimal solutions**
  3) Select **200** solutions for the **training set**, 8 optimal solutions, and **300** for the **test set**.
  4) Perform learning: **1545 iterations.**

- Results:
  - Absolute mean error on test set: 0.01
  - Maximum absolute error on test set: 0.04
  - Number of optimal solutions of the learned problem: 252 all rated highly by the original function.
  - Number of unseen optimal solutions recognized by the learned problem: 29.

An example with resource consumption

- Problem: **8 activities** to be scheduled in **24 hours**
- Given:
  - Duration intervals for each activity
  - Constraint graph
- Aim: Minimize the global resource consumption

- Procedure:
  1) Solve the hard constraint problem: **900 solutions**
  2) Rate each solution with a function that gives higher preference to schedules that require less energy: **13 optimal solutions**
  3) Select **200** solutions for the **training set**, 4 optimal solutions, and **300** for the **test set**
  4) Perform learning: **33945 iterations**

- Results:
  - Absolute mean error on test set: 0.02
  - Maximum absolute error on test set: 0.08
  - Number of optimal solutions of the learned problem: 39 all rated highly by the original function.
  - Number of unseen optimal solutions recognized by the learned problem: 9.

Simple Temporal Problems with Uncertainty

(Vidal,Fargier '99)

(Morris, Muscettola, Vidal '01)

Overview

- Simple Temporal Problems with Uncertanty
- Strong Controllability
- Dynamic Controllability
- Weak Controllability

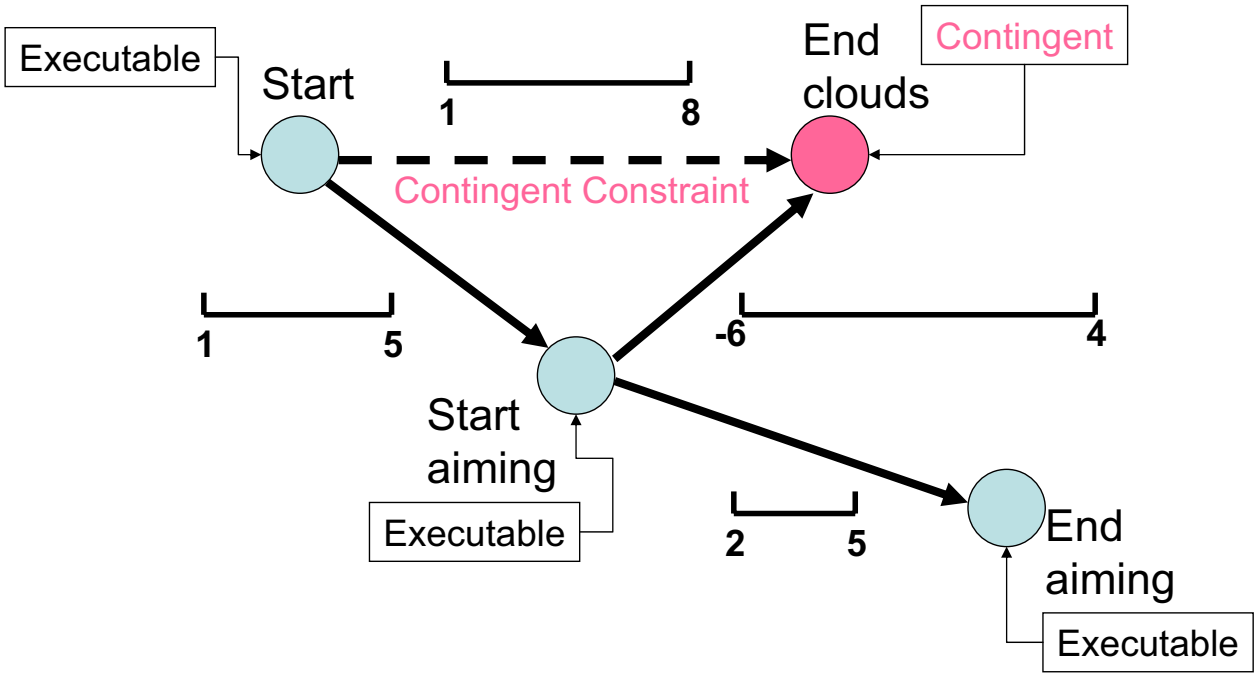
Simple Temporal Problems with Uncertainty

Informally, an STPU is an STP where some of the variables are not under the control of the agent, i.e. the agent cannot decide which value to assign to them.

An STPU:

- Set of ***executable timepoints*** (controllable assignment);
- Set of ***contingent timepoints*** (uncontrollable assignment);
- Set ***requirement constraints*** T_{ij}:
  - Binary
  - Temporal interval I=[a,b] meaning $a \leq X_j - X_i \leq b$
- Set of ***contingent constraints*** T_{hk}:
  - Binary: on an executable X_h and a contingent timepoint X_k
  - Temporal interval I=[c,d] meaning $a \leq X_k - X_h \leq b$

Example: satellite maneuvering



STPUs Definitions

- Given an STPU P
- A **control sequence *d*** is an assignment to the executable timepoints
- A **situation *w*** is a set of durations on contingent constraints (set of elements of contingent intervals)
- A **schedule** is a complete assignment to the variables of P
- A schedule is **viable** if it is consistent with all the constraints. ***Sol(P)*** is the set of all viable schedules of P.
- A **projection P_w** corresponding to situation ***w*** is the **STP** obtained replacing each contingent constraint with its duration in ***w***. ***Proj(P)*** is the set of all projection of P.
- A **viable strategy *S: Proj(P) → Sol(P)*** maps every projection P_w into a schedule including *w*

Controllability

There is a plan that will work whatever happens In the future

Strong Controllability

Dynamic Controllability

I can build a plan while things happen that will be successful.

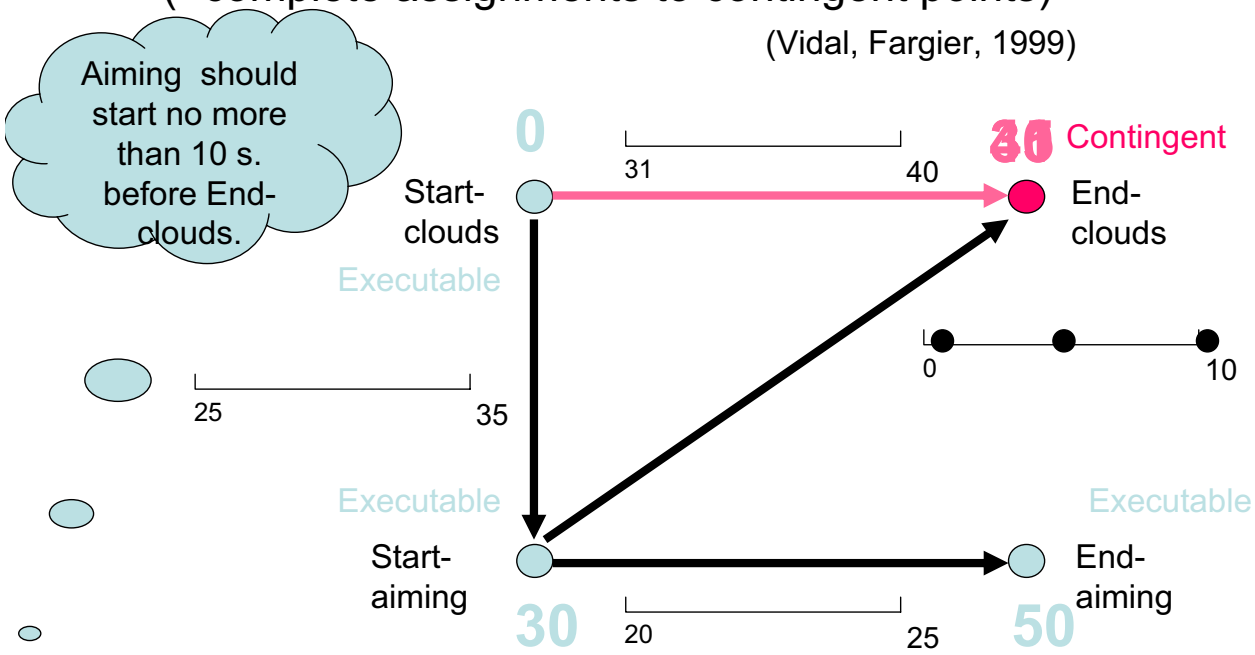
For every possible scenario there is a plan

Weak Controllability

Strong Controllability (SC) of STPUs

- An **STP with Uncertainty** is **Strongly Controllable** if there is an assignment to all the executable time points **consistent** with all the possible scenarios (=complete assignments to contingent points)

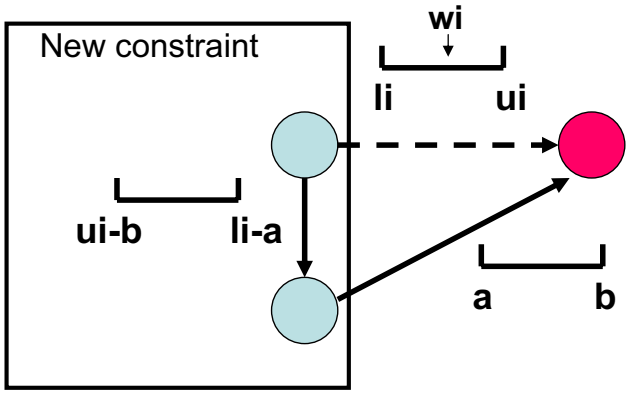
(Vidal, Fargier, 1999)

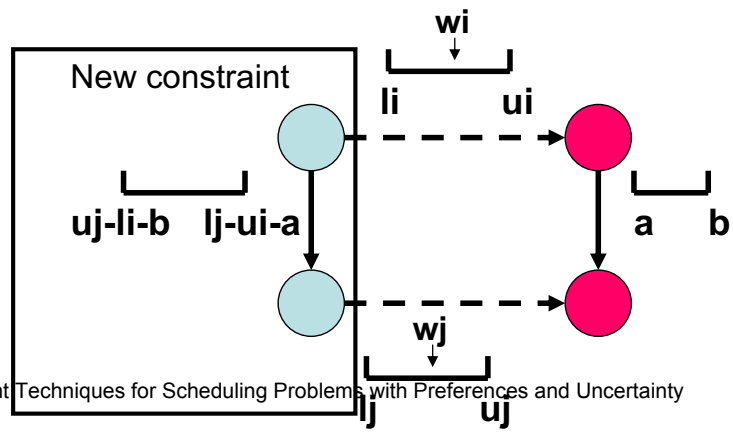


SC algorithm

1. Input STPU Q
2. Rewrite all the constraints involving contingent variables in terms of constraints involving only executable variables
3. From an STPU Q obtain and STP P defined only on the executables of Q
4. **STPU Q SC iff STP P is consistent**
5. Each solution of P is a control sequence of Q consistent with any possible situation *w*
6. Return minimal network of STP P

Complexity $O(n^3)$, n=number of executable variables



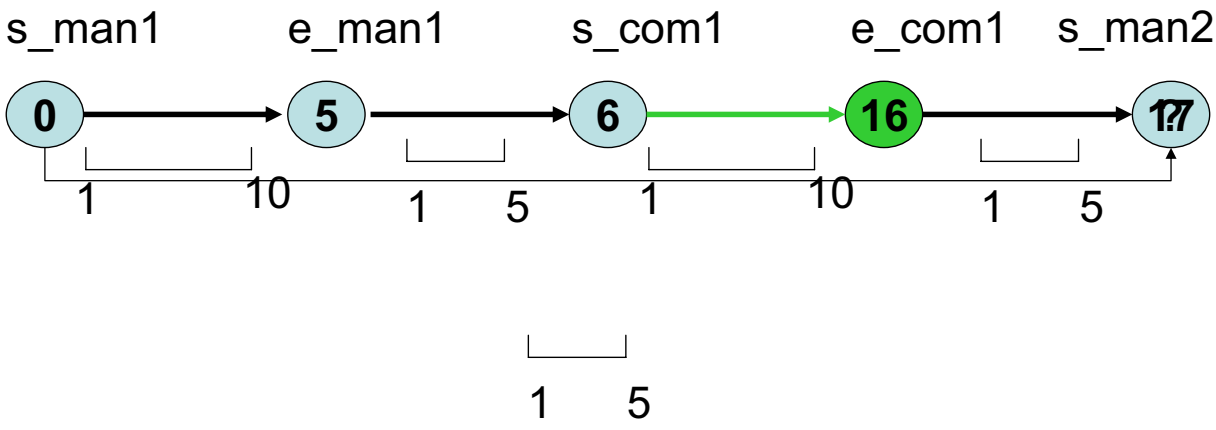


Dynamic Controllability

An **STP with Uncertainty** is **Dynamically Controllable** if there is an online execution strategy that depends only on observed timepoints in the past and that can always be extended to a **complete** schedule whatever may happen in the future.

(Vidal, Fargier, 1999)

Dynamical Controllability: Example

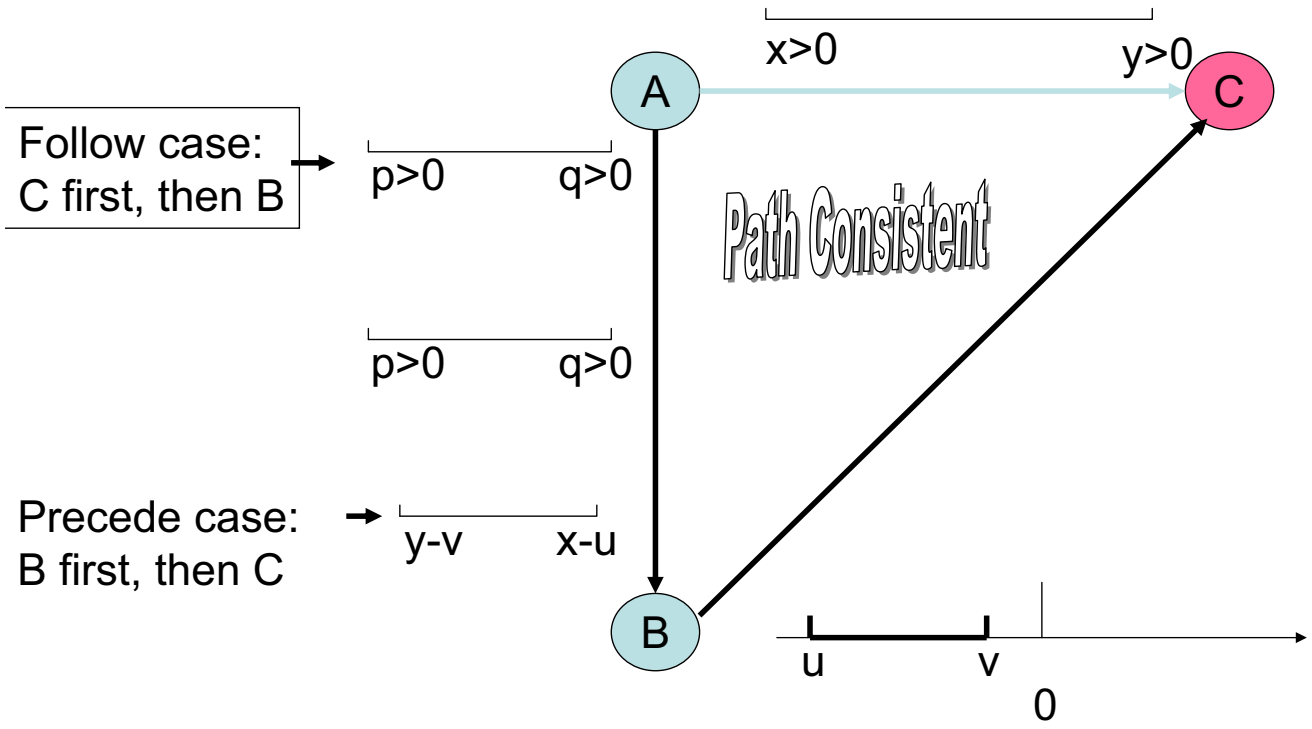


Dynamically Controllable

NOT Dynamically Controllable

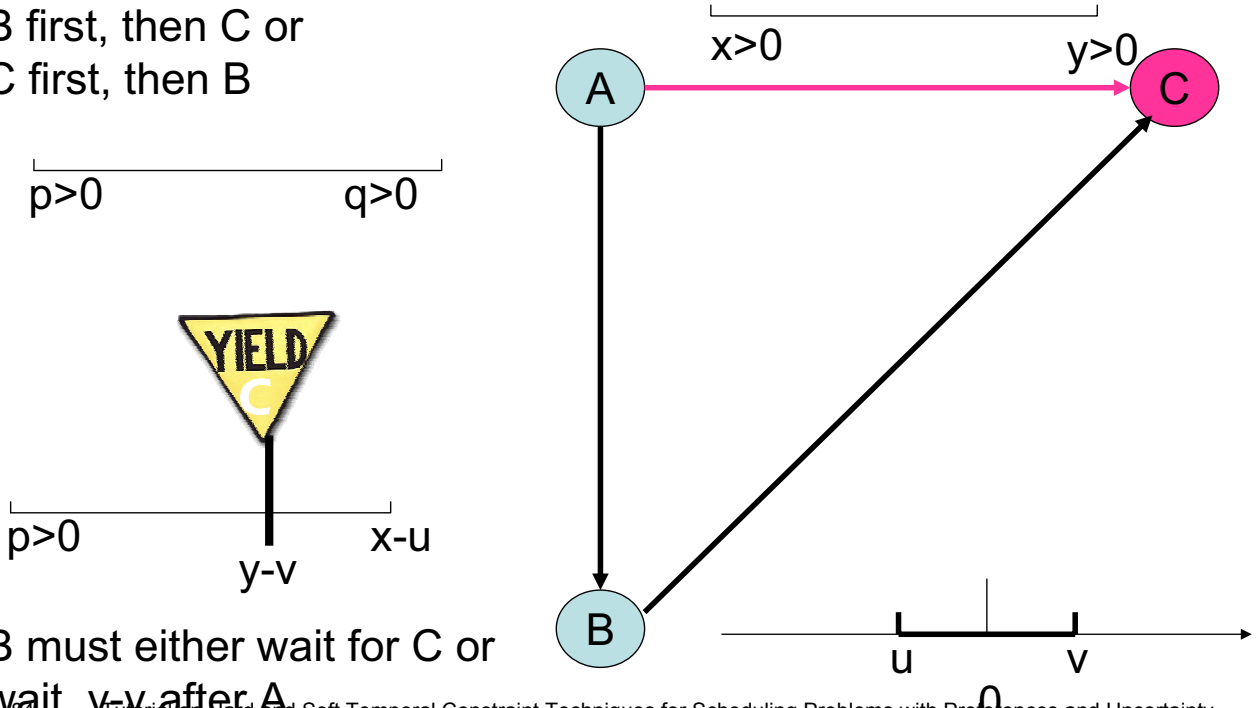
DC of triangular networks (1)

(Morris, Muscettola,Vidal,01)



DC of triangular networks (2)

Unordered case:
B first, then C or
C first, then B



B must either wait for C or wait y-v after A

(Morris, Muscettola,Vidal'01)

Wait regression

- Regression 1:
  - AB constraint has a wait <C,t>
  - Any DB constraint with upper bound u
  - → deduce wait <C,t-u> on AD
- Regression 2:
  - AB constraint has a wait <C,t>
  - A contingent constraint DB with lower bound z
  - → deduce wait <C,t-z> on AD

DC Algorithm

1. **Input** STPU P
2. Until quiescence:
3. If enforcing path consistency on P tightens any contingent interval then exit (not DC)
4. **Select** any triangle ABC, C uncontrollable, A before C
5. Perform triangular **reduction**
6. **Regress** waits
7. **Output** minimal STPU P

Complexity : deterministic polynomial

Weak Controllability

(Vidal,Fargier '99)

- An STPU is weakly controllable if for every situation *w*, the corresponding projection STP P_w is consistent
- Consider STPU Q and the set $Z=\{l_1,u_1\} \times \ldots \times \{l_h,u_h\}$, where l_J,u_J are the lower and upper bound of a contingent constraint in Q:

  →Q is WC iff for every *w'* in *Z*, STP $\mathbf{P_{w'}}$ is consistent
- The WC algorithm tests this property. Exponential.
- Testing WC is Co-NP-complete (Vidal,Fargier'99 and Morris, Muscettola '99)

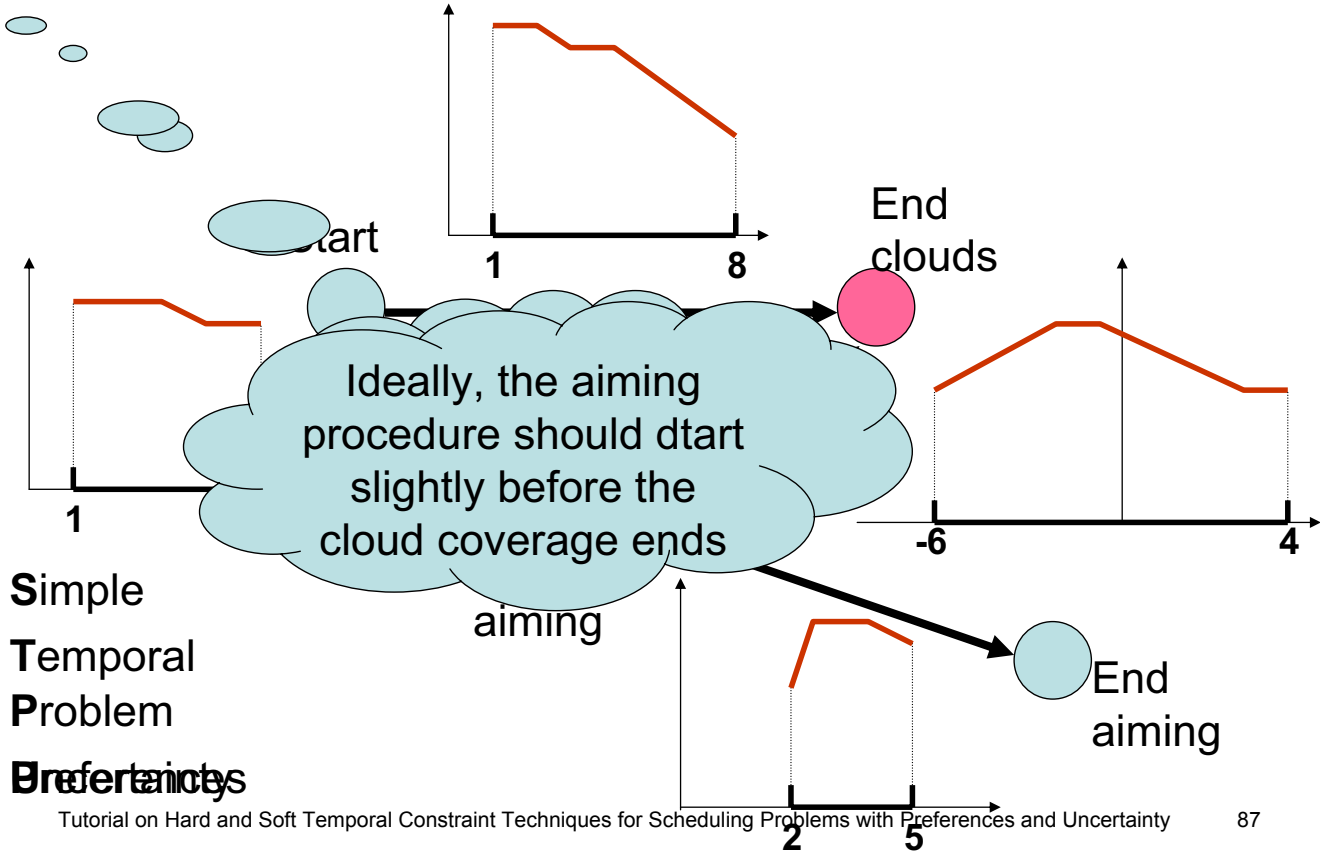
Simple Temporal Problems with Preferences and Uncertainty

(Rossi,Venable,Yorke-Simth '04)

Overview

- Simple Temporal problems wth Preferences
- Optimal and α-Strong Controllability
- Optimal and α-Dynamic Controllability
- Optimal Weak Controllability

Example: satellite manouvering



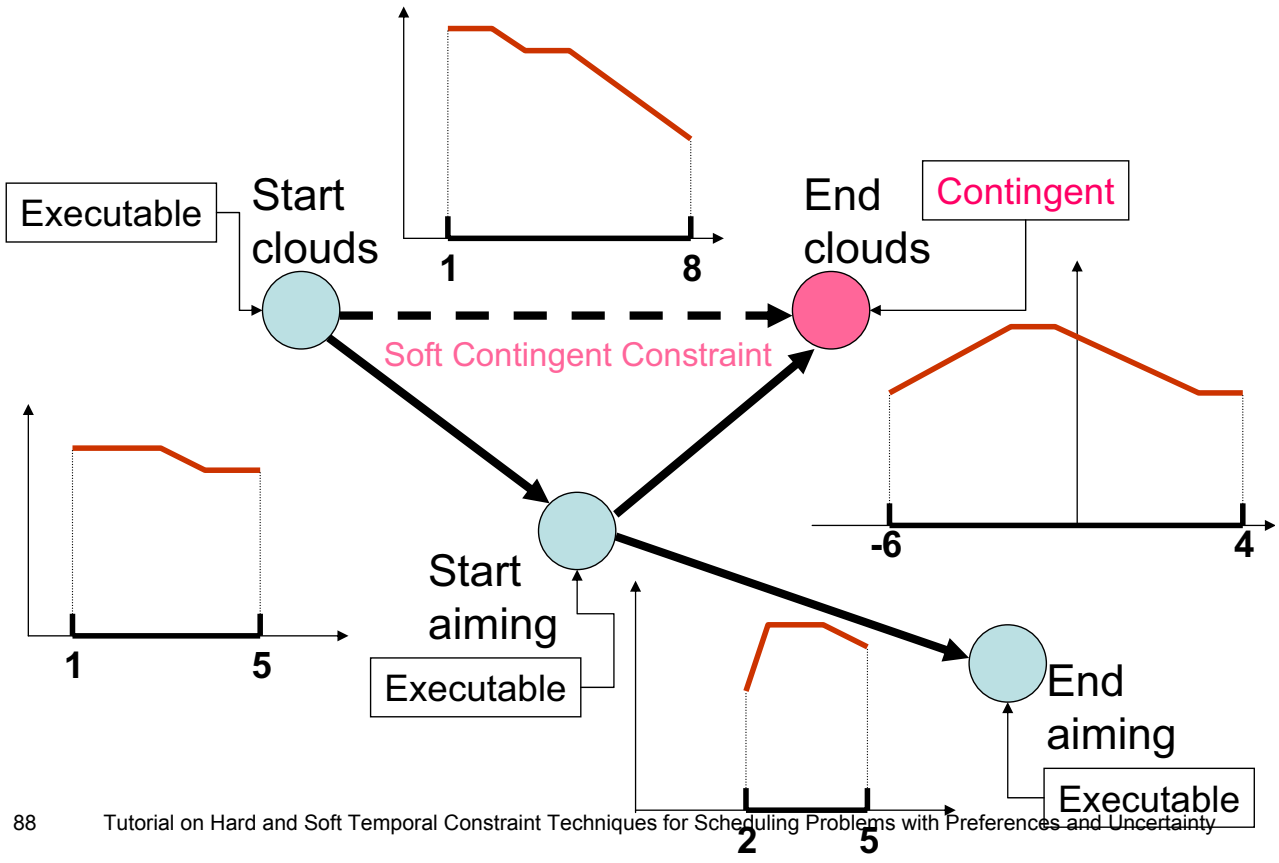
Simple
Temporal
Problem
Preferences
Uncertainty

Simple Temporal Problems with Preference and Uncertainty

An STPPU:

- Set of ***executable timepoints*** (controllable assignment);
- Set of ***contingent timepoints*** (uncontrollable assignment);
- Set of ***soft requirement constraints***:
  - Binary
  - Temporal interval I
  - Preference function f: I →A;
- Set of ***soft contingent constraints***:
  - Binary: on an executable and a contingent timepoint
  - Temporal interval I
  - Preference function f: I→A
- C-Semiring <A,+,x,0,1>

Example: satellite maneuvering



STPPUs Definitions (1)

- Given an STPU P
- A **control sequence *d*** is an assignment to the executable timepoints
- A **situation *w*** is a set of durations on contingent constraints (set of elements of contingent intervals)

STPPUs Definitions (2)

- Given an STPU P
- A **schedule** is a complete assignment to the variables of P
- A schedule T is **viable** if it is consistent with all the constraints.
  - **Preference value** associated to T: **pref(T)=$f_1(T_1)$ x … x $f_n(T_n)$,** where f_i = preference function of i-th constraint and T_i = projection of T on i-th constraint
  - **Sol(P)** is the set of all viable schedules of P
- A **projection *P_w*** corresponding to situation ***w*** is the **STPP** obtained replacing each contingent constraint with its duration in ***w and the corresponding preference value***.
  - **opt(P_w)** is the optimal preference of STPP P_w
  - ***Proj(P)*** is the set of all projection of P
- A **viable strategy *S: Proj(P) → Sol(P)*** maps every projection P_w into a schedule including *w*

Optimal Controllability with Preferences

There is an **Optimal** plan that will work whatever happens In the future

Optimal Strong Controllability

I can build an **Optimal** plan while things happen that will be successful.

Optimal Dynamic Controllability

For every possible scenario there is an **Optimal** plan

Optimal Weak Controllability

α-Controllability with Preferences

There is a plan that will work **whatever happens** in the future that is:
-an **optimal** solution in scenarios with **optimal preference ≤α**
-a solution with **preference ≥α otherwise**

α -Strong Controllability

α-Dynamic Controllability

It is possible to build a plan **while things happen** such that it will lead to:
-an **optimal** solution in scenarios with **optimal preference ≤α**
-a solution with **preference ≥α otherwise**

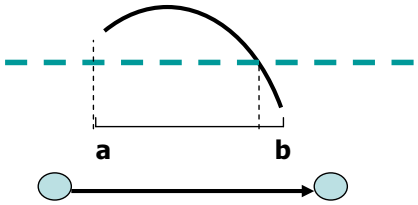
Tractability Assumptions

- On the C-semiring:
  - Idempotent multiplicative operator
  - Totally ordered preference set

  $\Rightarrow$ Fuzzy Semiring <[0,1], max, min, 1, 0>

  $Pref(S)=min(f_1(S_1), \ldots, f_n(S_n))$

  $S_1 > S_2$ *iff* $Pref(S_1)= max(Pref(S_1),pref(S_2))$
- On the preference functions:
  - Semi-convexity



Optimal Strong Controllability (OSC)

- An STPPU P is **Optimally Strongly Controllable** iff there is a viable strategy *S* such that

1. For every P_w and $P_{w'}$, for each executable x, $S(P_w)(x)=S(P_{w'})(x)$. Each executable is assigned the same value in every scenario.
2. For every P_w, $pref(S(P_w))=opt(P_w)$. The schedule obtained is optimal (i.e. there is no other with higher preference) in every scenario.

α−Strong Controllability (α-SC)

- An STPPU P is **α-Strongly Controllable** iff there is a viable strategy *S* such that

1. For every P_w and $P_{w'}$, for each executable x, $S(P_w)(x)=S(P_{w'})(x)$. Each executable is assigned the same value in every scenario.
2. For every P_w, $pref(S(P_w))=opt(P_w)$, if $opt(P_w)\leq\alpha$. Otherwise $pref(S(P_w))\geq\alpha$.
   1. The schedule obtained is optimal in every scenario such that the corresponding projection has optimal preference at most α,
   2. otherwise, for scenarios with an optimal preference higher than α, the solution obtained will have preference at least α.

Best-SC Algorithm

Checks Optimal Strong Controllability and finds the maximum level of α-Strong Controllability

Best-SC

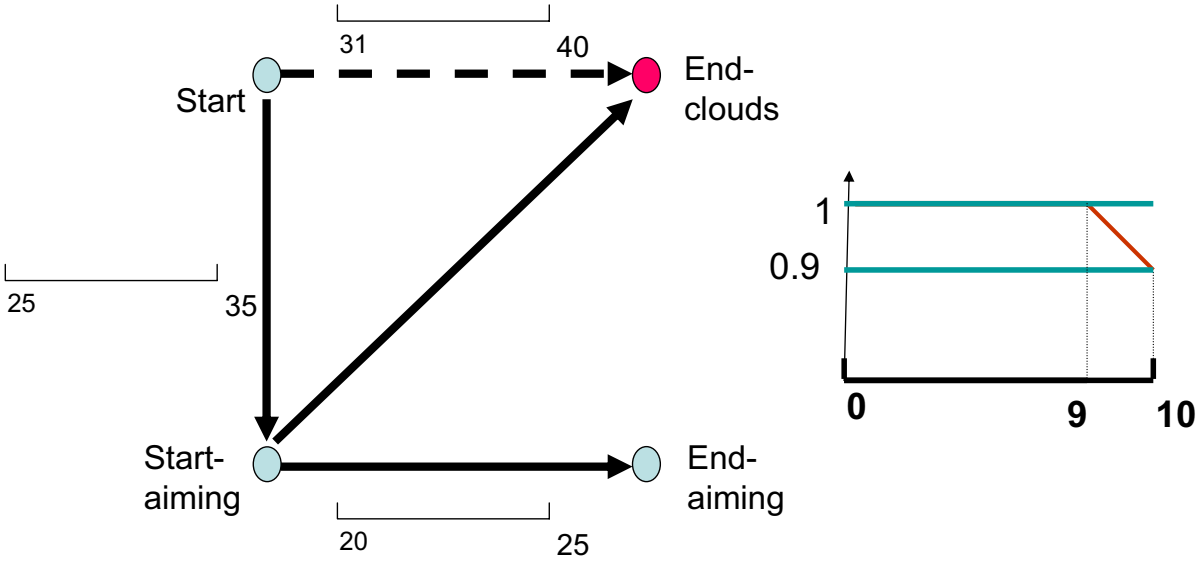
1. From the minimum preference up until inconsistency L_{max} do:
   1. Chop the STPPU P and get STPU Q
   2. Check if Q is Strongly Controllable
   3. Merge the results obtained at all preferences levels
2. If there is a complete assignment *S*, $pref(S)\geq L_{max}$ then $(L_{max}-1)$-SC; else OSC.

OSC-Check step 1: Chopping the STPPU

Chopping a soft constraint at preference β = keeping only elements with preference ≥β. (Hard constraint →all allowed elements have maximum preference)

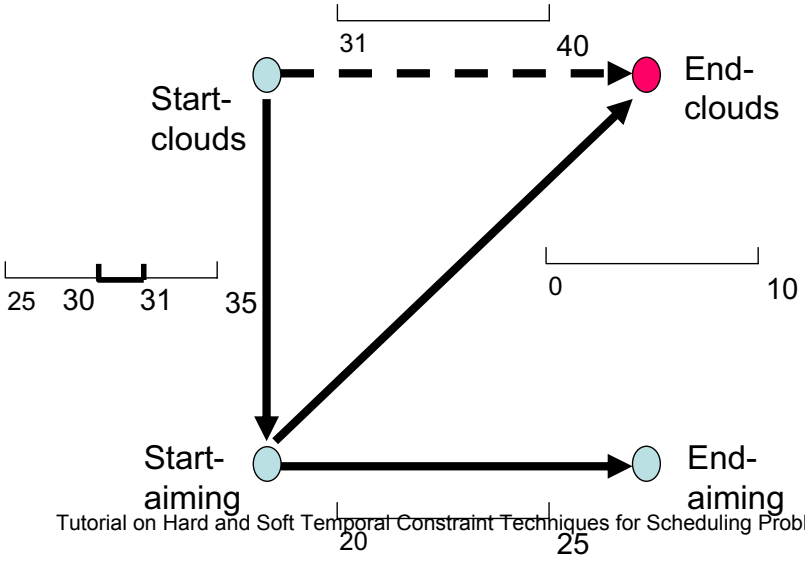
STPPU P

STPPU $Q^{0.9}$ = Chop(P, 0.9)



OSC-Check step 2: enforcing Strong Controllability

1. STPU Q^{β} = Chop(STPPU P, preference β)
2. Enforce Strong Controllability (Vidal et al. ,99) on Q^{β}
   Consider STP T^{β} only on executable variables:
   Q^{β} Strongly Controllable iff T^{β} consistent



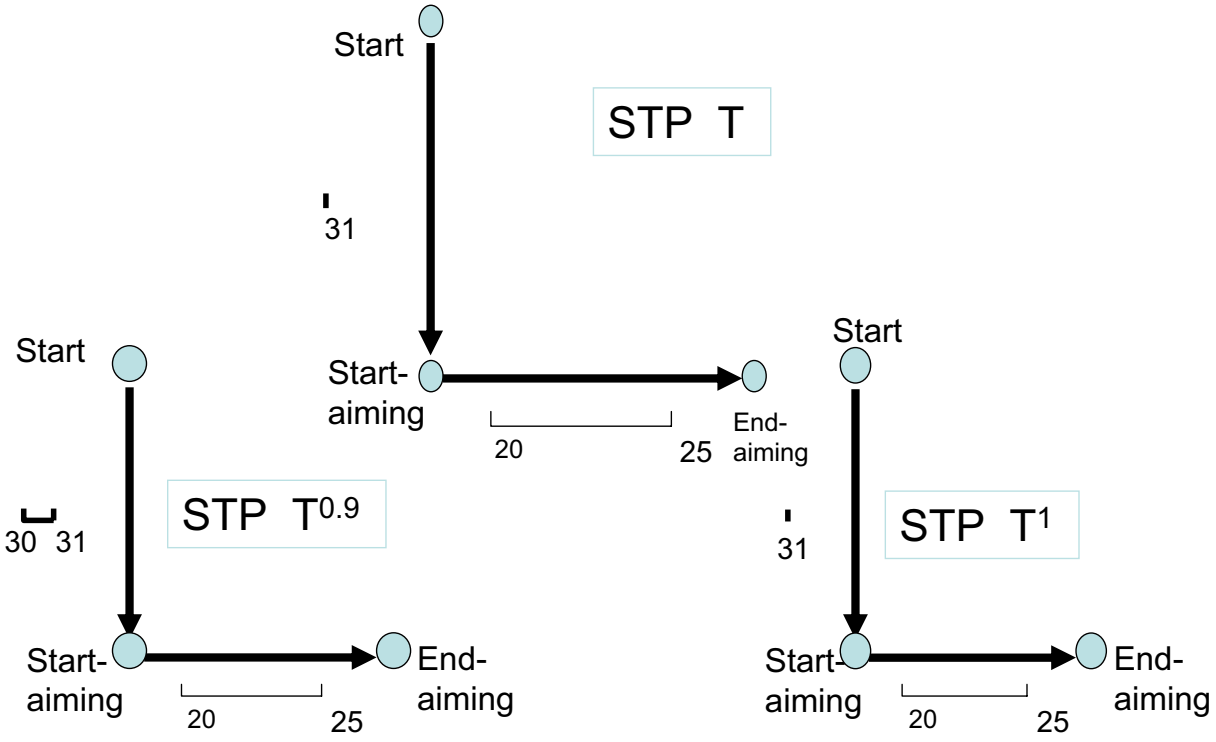
STPU $Q^{0.9}$

Keep only assignments to executables consistent with ALL contingent events

STP $T^{0.9}$

OSC-Check step 3: Merge

Intersect the intervals of the STPs $T^{\beta,}$ for all β such that T^{β} consistent.
Return the resulting STP T if consistent:
All solutions of T are consistent and optimal wit any possible scenario



Complexity of Best-SC

$$|\text{preferences}| \times |\text{variables}|^3 \times |\text{interval size}|$$

SC complexity

Optimal Dynamic Controllability (OSC)

- **Prehistory of executable x given schedule T**, written $\mathbf{T_{<x}}$: is the set of all durations of contingent events which have occurred before x in T
- An STPPU P is **Optimally Dynamically Controllable** iff there is a viable strategy *S* such that

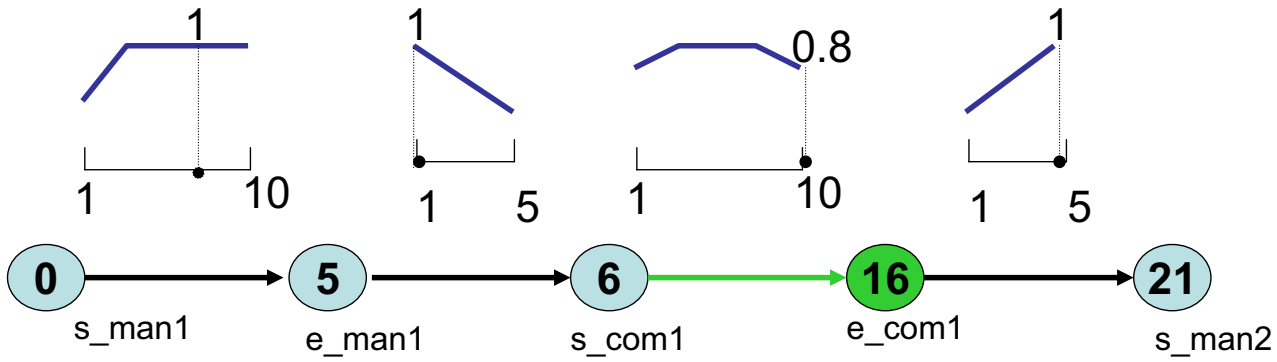
1. For every P_w and $P_{w'}$, for each executable *x*, if $S(P_w)<x=S(P_{w'})<x$, then $S(P_w)(x)=S(P_{w'})(x)$. Each executable is assigned the same value in every scenario in which the same contingent events have occurred before it.
2. For every P_w, $pref(S(P_w))=opt(P_w)$. The schedule obtained is optimal (i.e. there is no other with higher preference) in every scenario.

α−Dynamic Controllability (α−DC)

- An STPPU P is **α−Dynamically Controllable** iff there is a viable strategy *S* such that

1. For every P_w and $P_{w'}$, for each executable *x*, if $S(P_w)<x=S(P_{w'})<x$, then $S(P_w)(x)=S(P_{w'})(x)$. Each executable is assigned the same value in every scenario in which the same contingent events have occurred before it.
2. For every P_w, $pref(S(P_w))=opt(P_w)$, if $opt(P_w)\leq\alpha$. Otherwise $pref(S(P_w))\geq\alpha$.
   1. The schedule obtained is optimal in every scenario such that the corresponding projection has optimal preference at most α,
   2. otherwise, for scenarios with an optimal preference higher than α, the solution obtained will have preference at least α.

Optimal Dynamic Controllability: Example



Optimally Dynamically Controllable

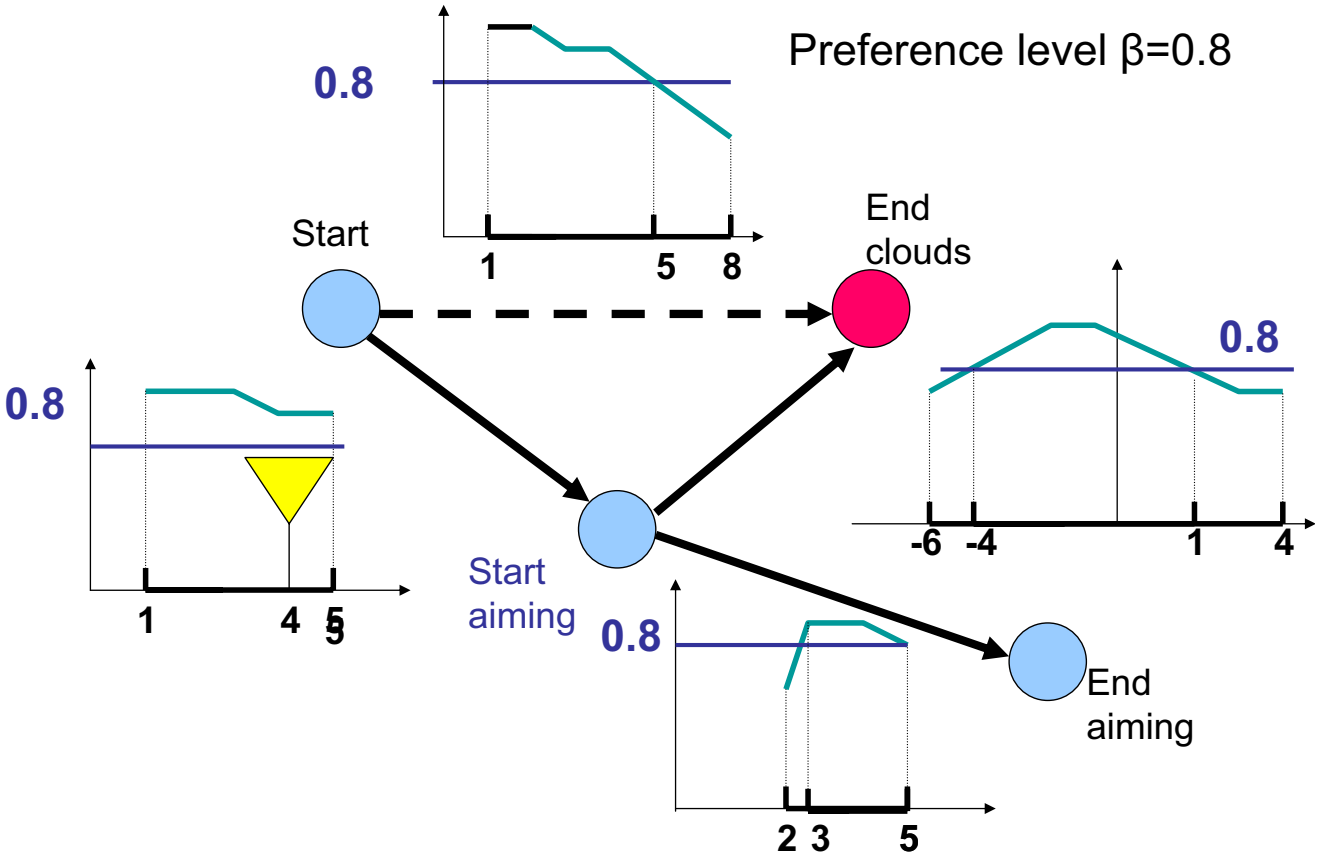
Best-DC Algorithm

Checks Optimal Dynamic Controllability and finds the maximum level of α-Dynamic Controllability

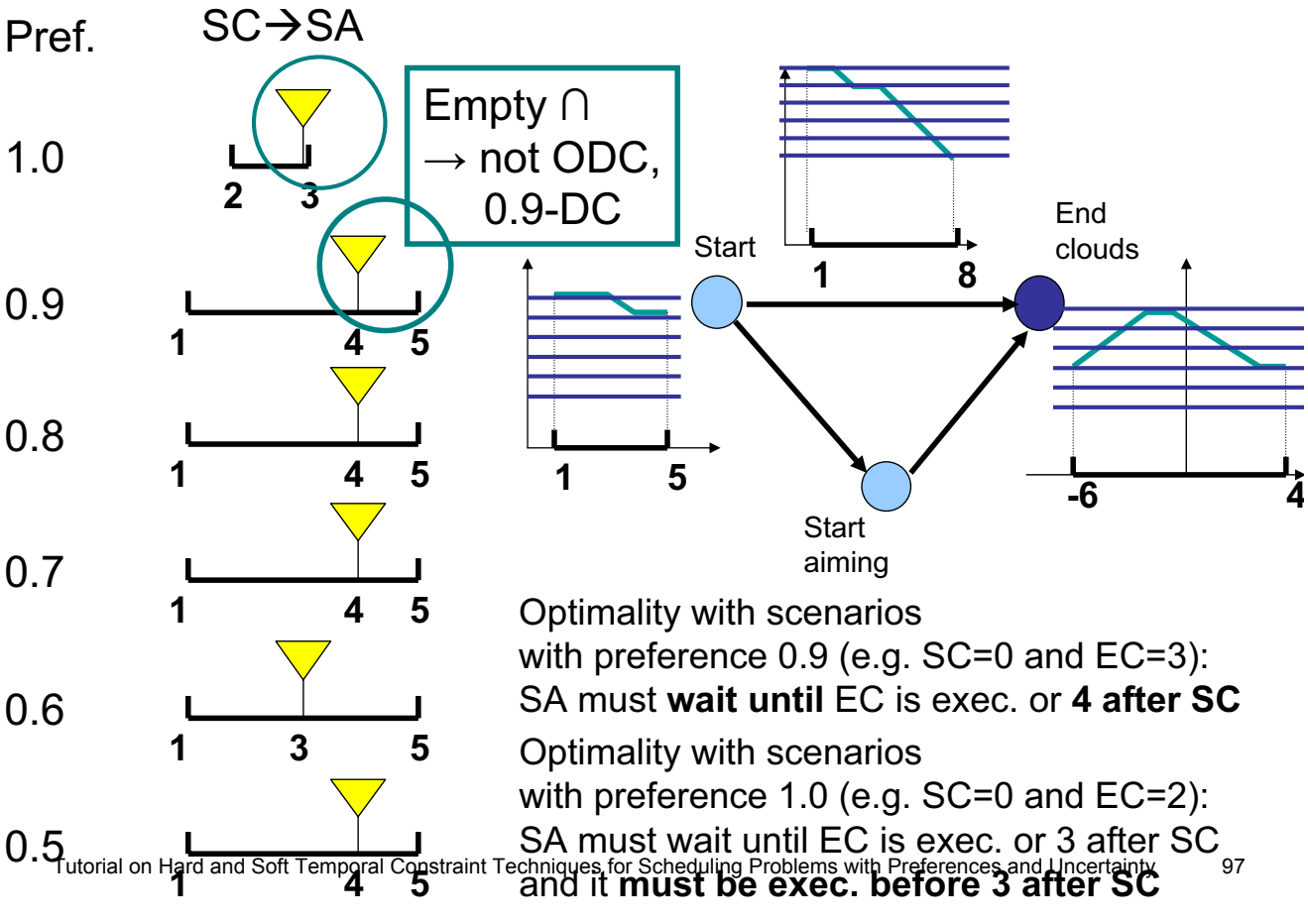
Best-DC

1. From the minimum preference up until inconsistency L_{max} do:
   1. Chop the STPPU P and get STPU Q
   2. Check if Q is Dynamically Controllable
   3. DC-Merge the results obtained at all preferences levels
2. If there is a complete assignment S, $pref(S) \geq L_{max}$ then $(L_{max}-1)$-DC; else ODC.

Best-DC step 1 & 2: example



Best-DC Merge Step exam

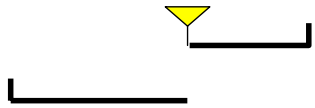


Optimality with scenarios with preference 0.9 (e.g. SC=0 and EC=3): SA must **wait until** EC is exec. or **4 after SC**

Optimality with scenarios with preference 1.0 (e.g. SC=0 and EC=2): SA must wait until EC is exec. or 3 after SC and it **must be exec. before 3 after SC**

ODC-Check step 3: DC-Merge

- For every preference level
  - For every constraint on executables
  - Consider the interval obtained chopping and applying DC
  - Merge the intervals taking:
    - ∩ of what follows waits
    - U of what precedes waits



- The constraints on executables of the resulting STPPU contain only elements that belong to at least one optimal dynamic schedule and the waits to be respected for optimality

Complexity of ODC-Check

|preferences| x DC Complexity

Execution of STPPUs

- as for STPUs (Morris et al. 2001)

0. Perform initial **propagation** from the start timepoint.

1. Immediately execute any executable timepoints that have reached their upper bounds.

2. Arbitrarily pick an executable timepoint TP that is live and enabled and not yet executed, and whose waits, if any, have all been satisfied.

3. Execute TP. Halt if network execution is complete.
Otherwise, **propagate the effect of the execution**.

4. Advance current time, **propagating** the effect of any contingent timepoints that occur, until an executable timepoint becomes eligible for execution under 1 or 2.

5. Go to 1.

Optimal Weak Controllability (OWC)

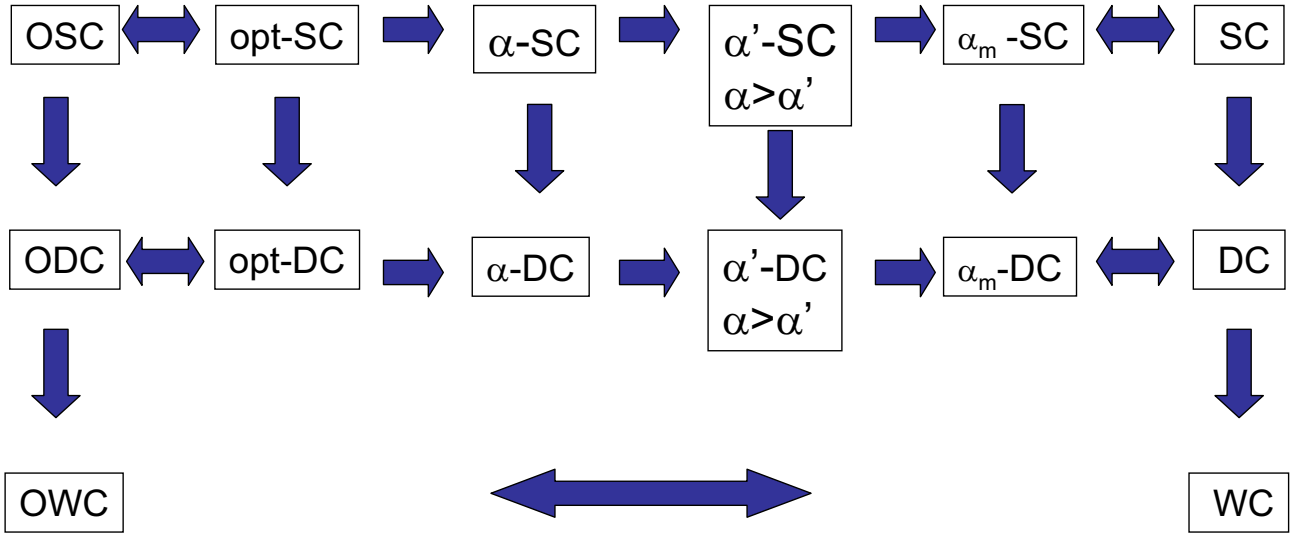
OPTIMALLY WEAKLY CONTROLLABLE
Every scenario has an optimal solution

WEAKLY CONTROLLABLE
Every scenario has a solution

OWC Check (STPPU P)
1. Ignore preferences on P obtaining STPU P'
2. Check if P' is WC

Optimal solution (STPPU P, Scenario s) Polynomial
1. Instantiate s in P obtaining STPP Q
2. Chop-solver(Q)

Comparison among the Controllability Notions



α_m = minimum preference on any constraint

References (1)

- [Allen'83]: J. Allen, Maintaining knowledge about temporal intervals. Communications of the ACM, 26:832--843, 1983.
- [Badaloni, Giacomin '00]: S. Badaloni, M. Giacomin, Flexible temporal constraints. (IPMU 2000).
- [Dubois,Fargier,Prade '95]: D. Dubois, H. Fargier, and H. Prade, Fuzzy constraints in job shop-scheduling. Journal of Intelligent Manufacturing, 6:215--234, 1995.
- [Dubois et al. '03]: D. Dubois, A. HadjAli, and H. Prade, Fuzziness and uncertainty in temporal reasoning. J. UCS, 9(9):1168--, 2003.
- [Dechter,Meiri,Pearl '91]: R. Dechter, I. Meiri, and J. Pearl, Temporal constraint networks. Artif. Intell., 49(1-3):61--95, 1991.
- [Dearden et al. '02]: R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Contingency planning for planetary rovers. Proc. 3rd Intl. Workshop on Planning and Scheduling for Space, 2002.

References (2)

- [Frank et al. ’01]: J. Frank, A. Jonsson, R. Morris, and D. Smith, Planning and scheduling for fleets of earth observing satellites. Proc. i-SAIRAS'01, 2001.
- [Ghallab,Alaoui ’89]: M. Ghallab and A. M. Alaoui. Managing efficiently temporal relations through indexed spanning trees. Proc. IJCAI’89, Morgan Kaufmann, 1989.
- [Khatib,Morris,Morris,Rossi ’01]: L. Khatib, P. H. Morris, R. A. Morris, and F. Rossi, Temporal constraint reasoning with preferences. Proc. IJCAI’01, Morgan Kaufmann, 2001.
- [Khatib et al. ’03] L. Khatib, P. H. Morris, R. A. Morris, and K. B. Venable, Tractable {P}areto optimization of temporal preferences. Proc. IJCAI’03,Morgan Kaufmann, 2003.
- [Morris, Muscettola ’99]: P. Morris, N. Muscettola, Managing temporal uncertainty through waypoint controllability. Proc. IJCAI’99 Morgan Kaufmann, 1999.
- [Morris et al ’04]: P. Morris, R. Morris, L. Khatib, S. Ramakrishnan, and A. Bachmann, Strategies for global optimization of temporal preferences. Proc. CP’04, Springer, 2004

References (3)

- [Muscettola et al.’98] N. Muscettola, P. H. Morris, B. Pell, and B. D. Smith. Issues in temporal reasoning for autonomous control systems. Agents, pp. 362--368, 1998.
- [Morris, Muscettola,Vidal ’01]: P. H. Morris, N. Muscettola, and T. Vidal, Dynamic control of plans with temporal uncertainty. Proc. IJCAI’01, Morgan Kaufmann, 2001.
- [Oddi,Cesta’00]: A. Oddi and A. Cesta, Incremental forward checking for the disjunctive temporal problem. Proc. ECAI 2000, IOS Press, 2000.
- [Peintner,Pollack ’04]: B. Peintner and M. E. Pollack, Low-cost addition of preferences to DTPs and TCSPs. Proc. AAAI’04, AAAI Press / The MIT Press, 2004.
- [Rossi et al. ’02]: F. Rossi, A. Sperduti, K. B. Venable, L. Khatib, P. H. Morris, and R. A. Morris. Learning and solving soft temporal constraints: An experimental study. Proc. CP’02, Springer, 2002
- [Schwalb,Dechter ’93]: E. Schwalb and R. Dechter. Coping with disjunctions in Temporal Constraint Satisfaction Problems. Proc. AAAI’93, AAAI Press/The MIT Press, 1993.

References (4)

- [Vila,Schwalb ’98]: E.Schwalb and L.Vila, Temporal Constraints: A survey. Constraints, 3(2/3):129--149, 1998.
- [Tsamardinos,Pollack’03]: I.Tsamardinos and M.E. Pollack, Efficient solution techniques for disjunctive temporal reasoning problems. Artif. Intell., 151(1-2):43--89, 2003.
- [Rossi,Venable,Yorke-Smith’ 04]: F. Rossi, K.B. Venable nd N. Yorke-Smith, Controllability of Soft Temporal Constraint Problems. Proc. CP’04, Springer, 2004.
- van Beek’89] P.~van Beek, Approximation algorithms for temporal reasoning. Proc. IJCAI’89, Morgan Kaufmann, 1989.
- [Vidal,Fargier ‘99]: T. Vidal and H. Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. J. Exp. Theor. Artif. Intell., 11(1):23--45, 1999.