Boosting the performance of Iterative Flattening Search

Angelo Oddi[1], Nicola Policella[1], Amedeo Cesta[1], Stephen F. Smith[2]

[1] ISTC-CNR, Italian National Research Council
Via San Martino della Battaglia 44, 00185 Rome, Italy
{angelo.oddi, nicola.policella, amedeo.cesta}@istc.cnr.it

[2] Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
sfs@cs.cmu.edu

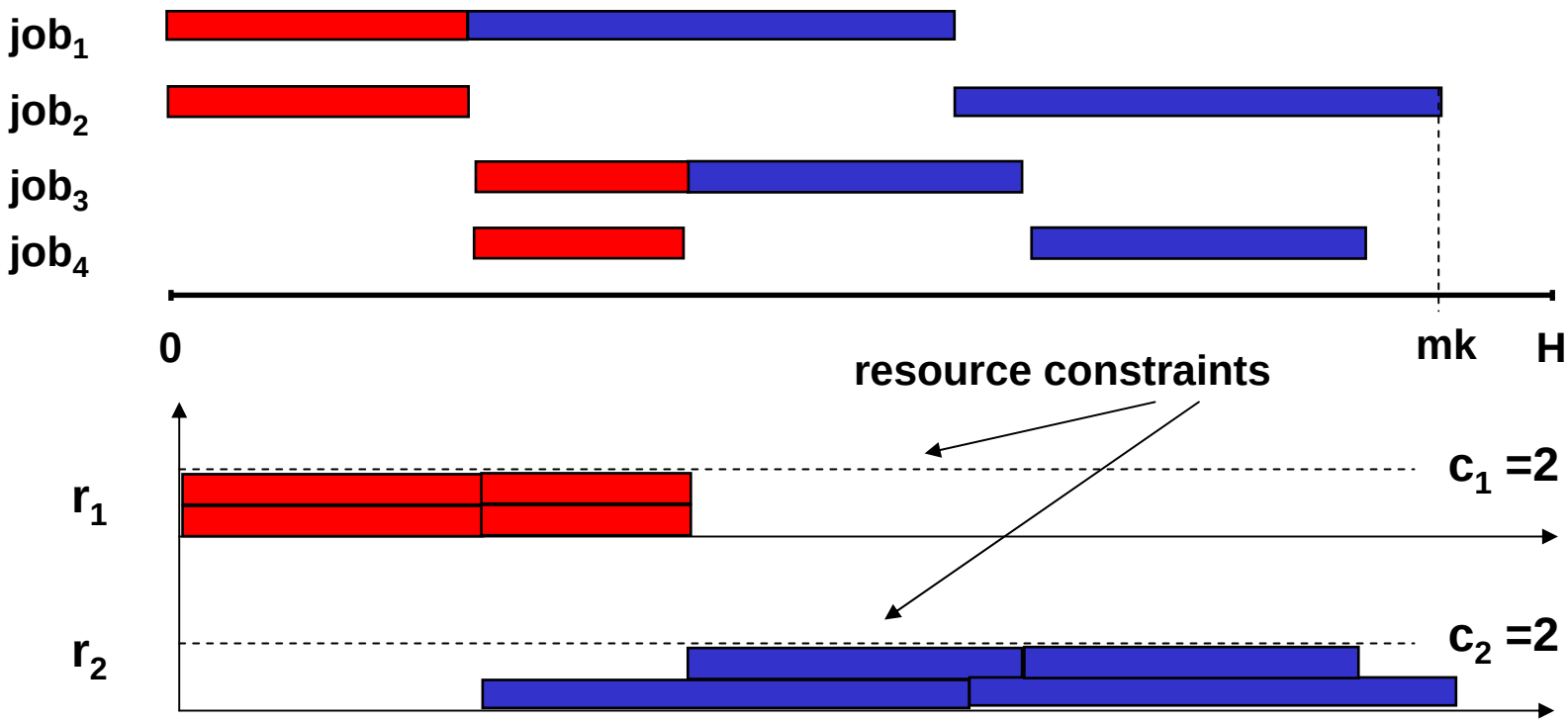
Context

- Iterative Flattening - **iFlat**
  - An iterative improvement search procedure for solving multi-capacitated scheduling problems with *makespan minimization* as the objective
  - The concept of iterative flattening search is quite general and provides a framework for designing effective procedures for scheduling optimization
- Reference works
  - Different *scheduling* algorithms *[authors, AIPS-98, IJCAI-99, Journal of Heuristics 2001]*
  - First version of Iterative Flattening (*iFlat*) *[authors, AAAI-2000]*
  - Improved version of *iFlat* [Michel&Van Hentenryck, (ICAPS-2004)]
  - Variation of the improved version [Godard, Laborie &Nuijten. *Randomized Large Neighborhood Search for Cumulative Scheduling* (ICAPS-2005)]

Outline of the talk

- A reference scheduling problem
- *Basic algorithms*
  - *Profile-based* algorithms
  - *Iterative Flattening*
- Improving Iterative Flattening search
  - Using *Partial Order Schedules* (POSs)
  - iFlat with *tabu-list*
  - Tabu-Search for a *fine-grained* exploration
  - *Loop* integration
- Experimental evaluation
- Conclusions and future work

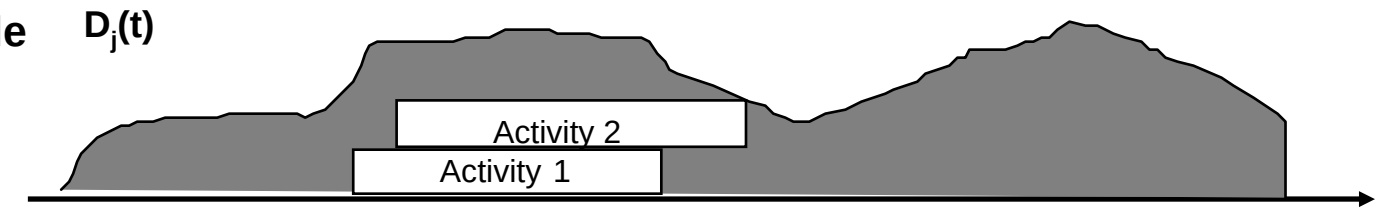
The MCJSSP scheduling problem



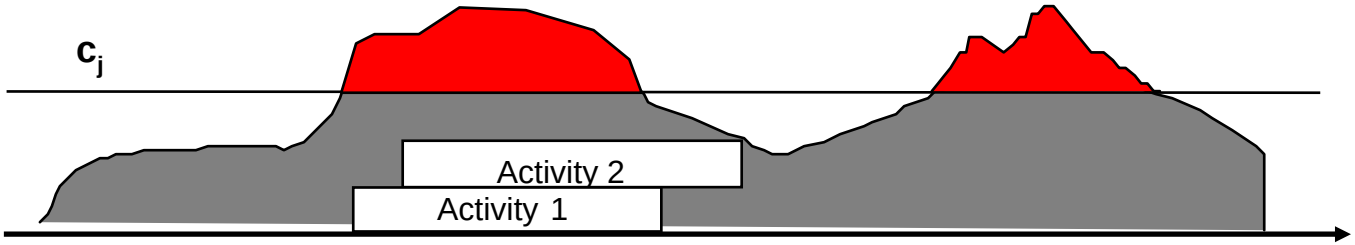
Scheduling problem with **four jobs**, each job has two activities; “red activities” require resource $\mathbf{r_1}$ and “blue activities” require resource $\mathbf{r_2}$

The *profile-based* approach

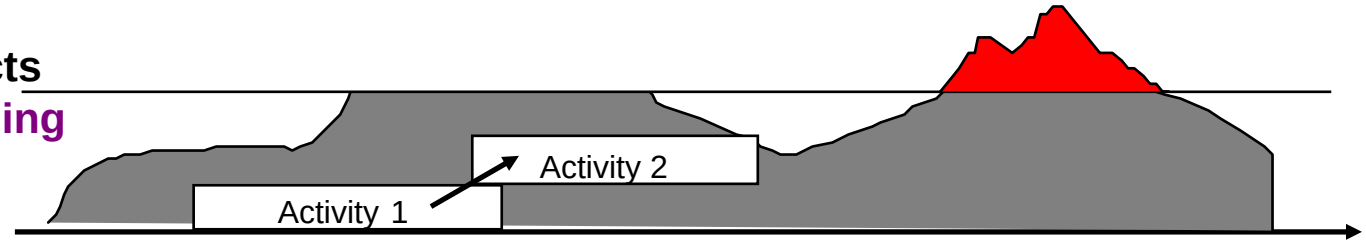
Demand profile for a resource



Conflict detection



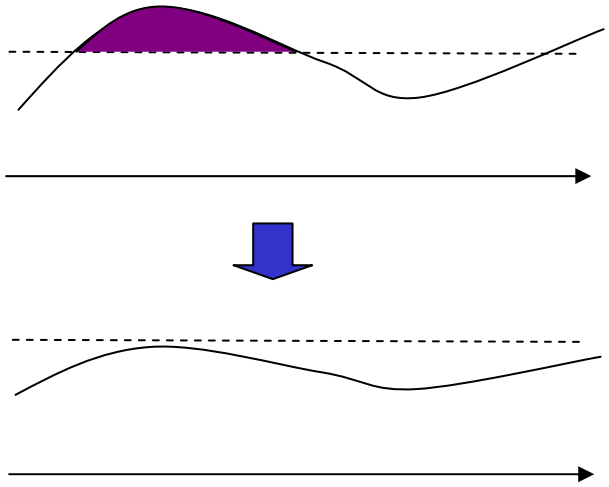
Removing conflicts
Introducing levelling constraints



ISTC CNR
INSTITUTE FOR COGNITIVE SCIENCE & TECHNOLOGY

ESTA: a greedy profile-based algorithm

- Starts with a *time-feasible* solution
- Posts constraints to "stretch" it into a *resource-feasible* solution.



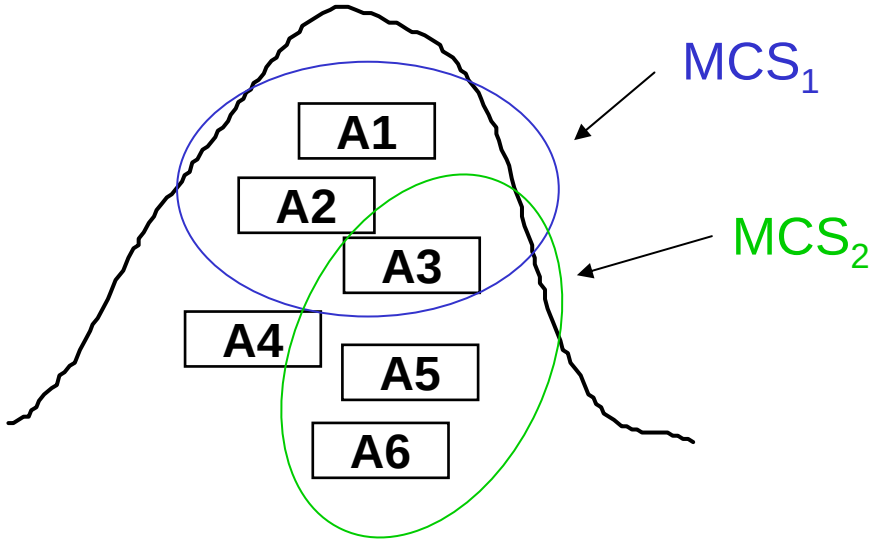
ESTA (*problem*, *horizon*)
post(*horizon*)
loop
- propagate
- compute-minimal-conflicts on resources
- **if** no-conflict
  - **then** return(*solution*)
  - **else if** unsolvable-conflicts
    - **then** return(*fail*)
    - **else**
      - select-conflict
      - select-precedence
      - post(*precedence*)

end-loop

Minimal Critical Set (MCS) analysis

A Minimal Critical Set (MCS) is a resource conflict such that each proper subset is not a resource conflict

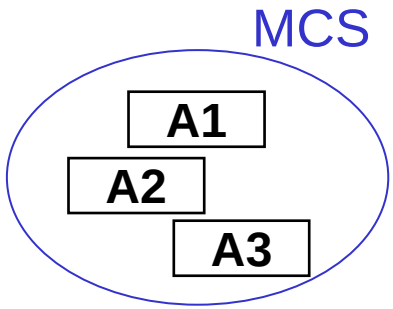
Resource capacity = 2



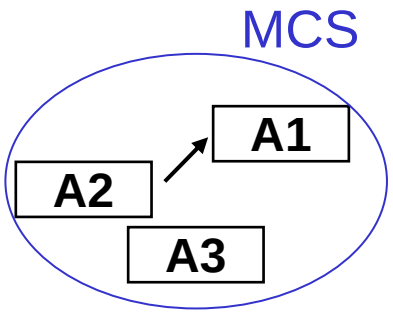
(Approximate computation of MCS *[authors, IJCAI-99]*)

MCS elimination

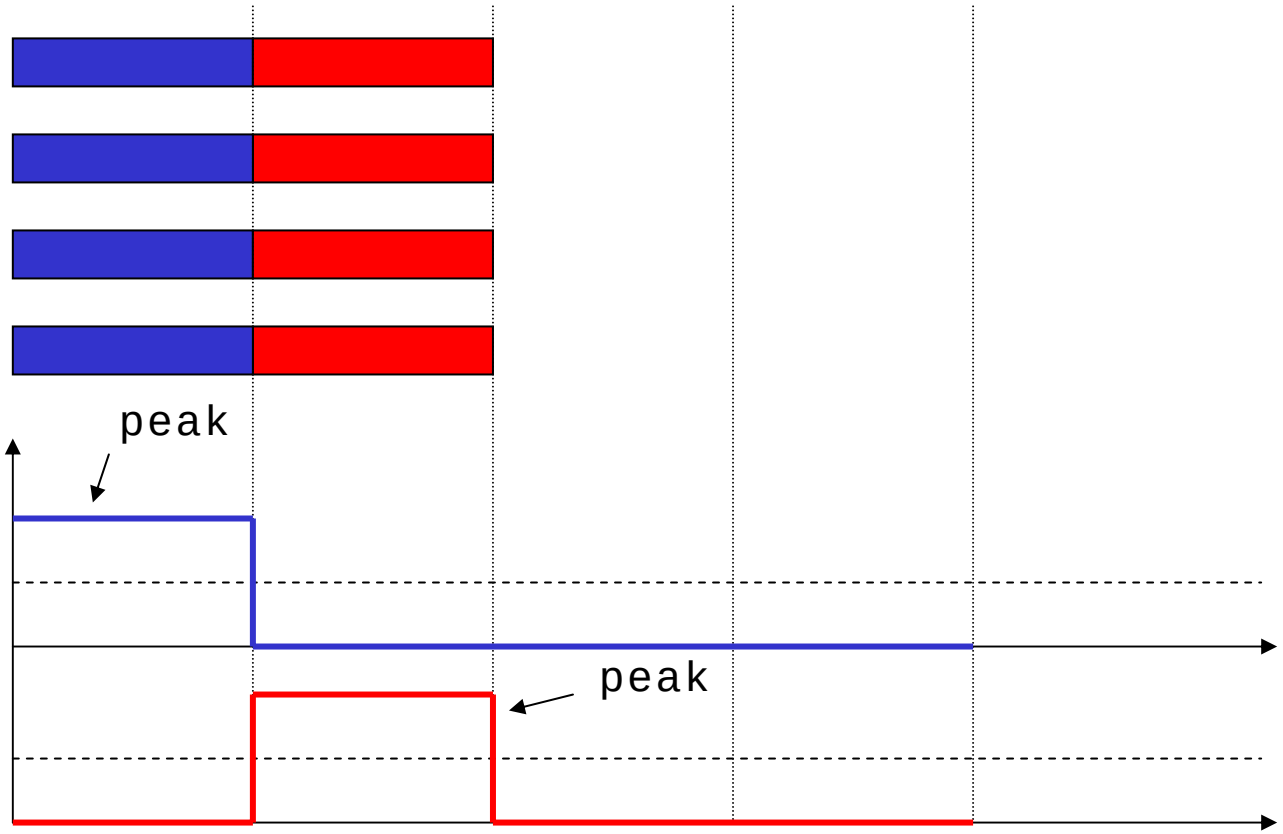
- Variable ordering: which MCS to resolve first
  - Use estimator K [*Laborie&Ghallab '95*] to order MCSs
  - "Select the MCS that is temporally closest to an unsolvable state"



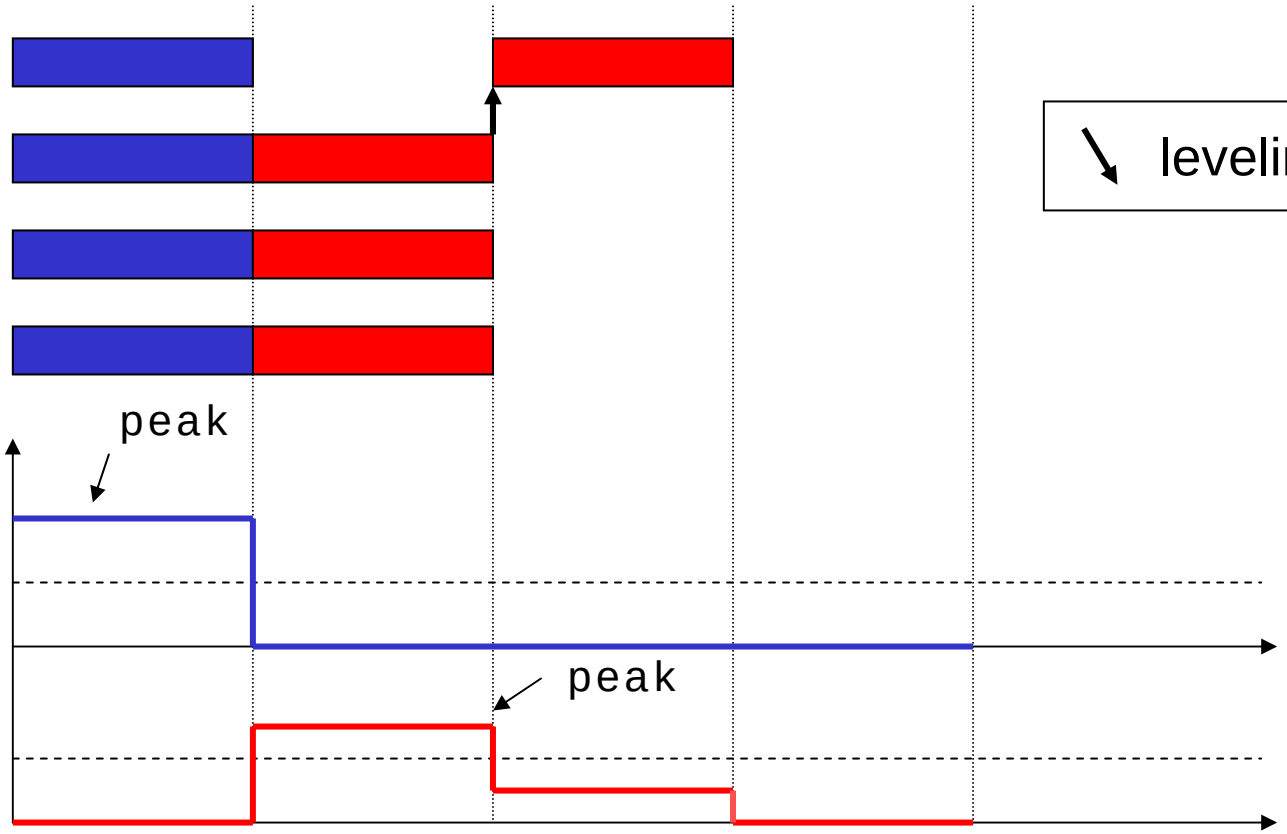
- Value ordering: how to choose the precedence (leveling) constraint
  - Use *slack-based* heuristics [*Smith&Cheng '93*]



Greedy strategy: example (1)

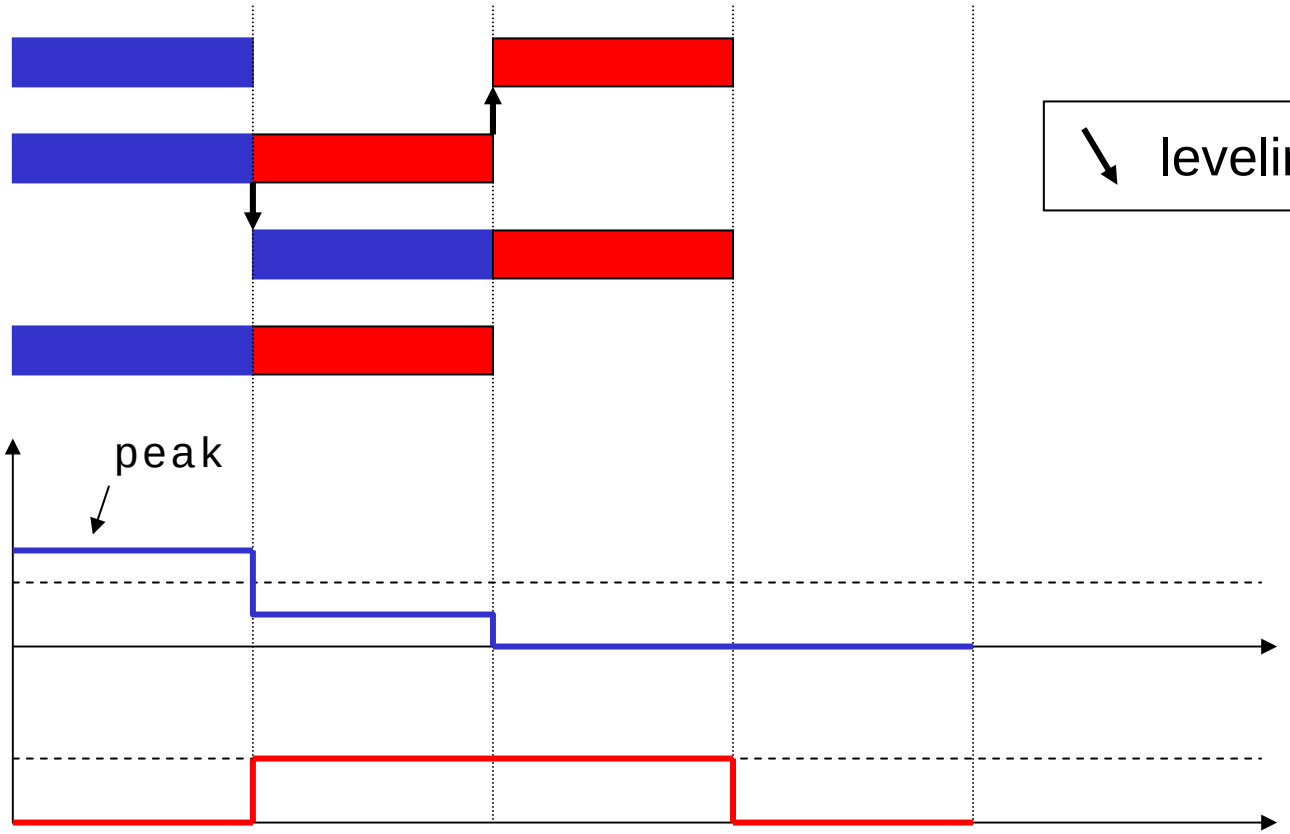


Greedy strategy: example (2)



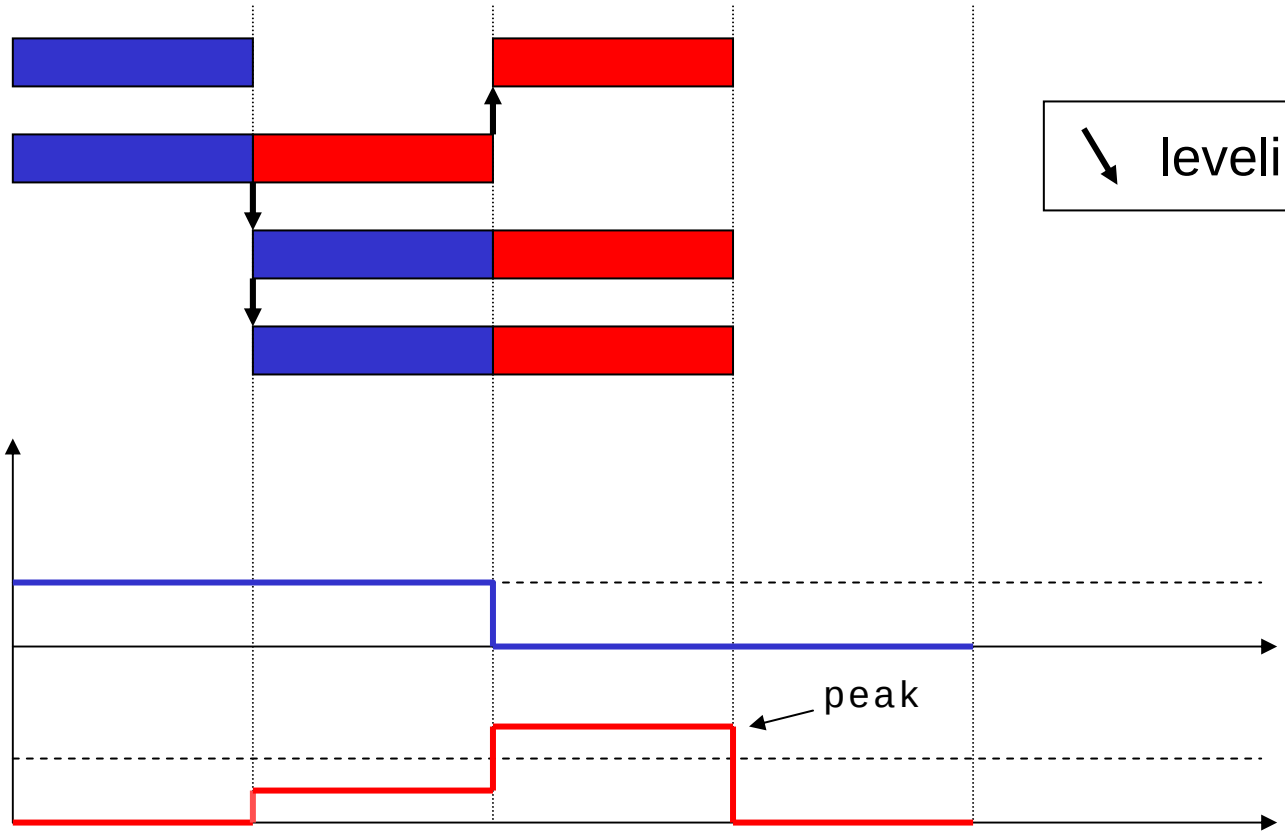
leveling constraint

Greedy strategy: example (3)

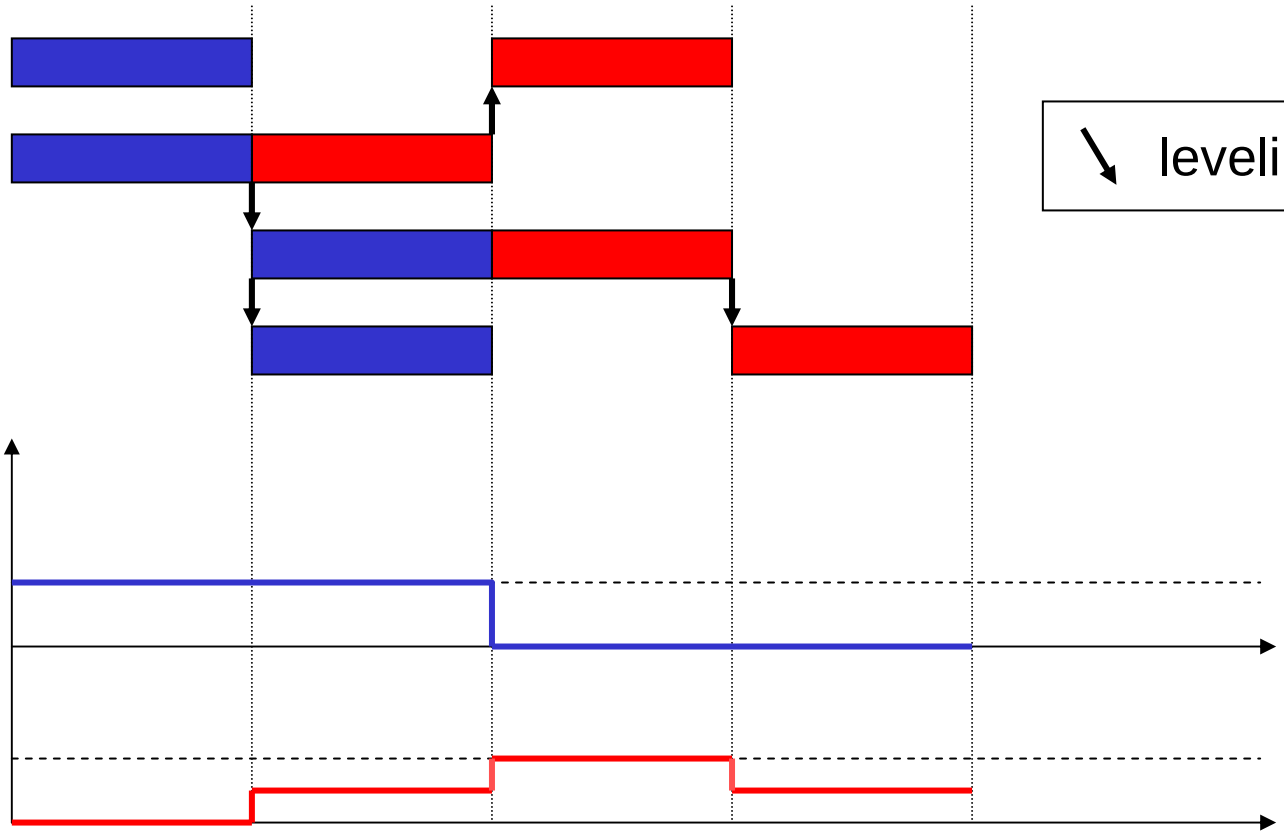


↘ leveling constraint

Greedy strategy: example (4)

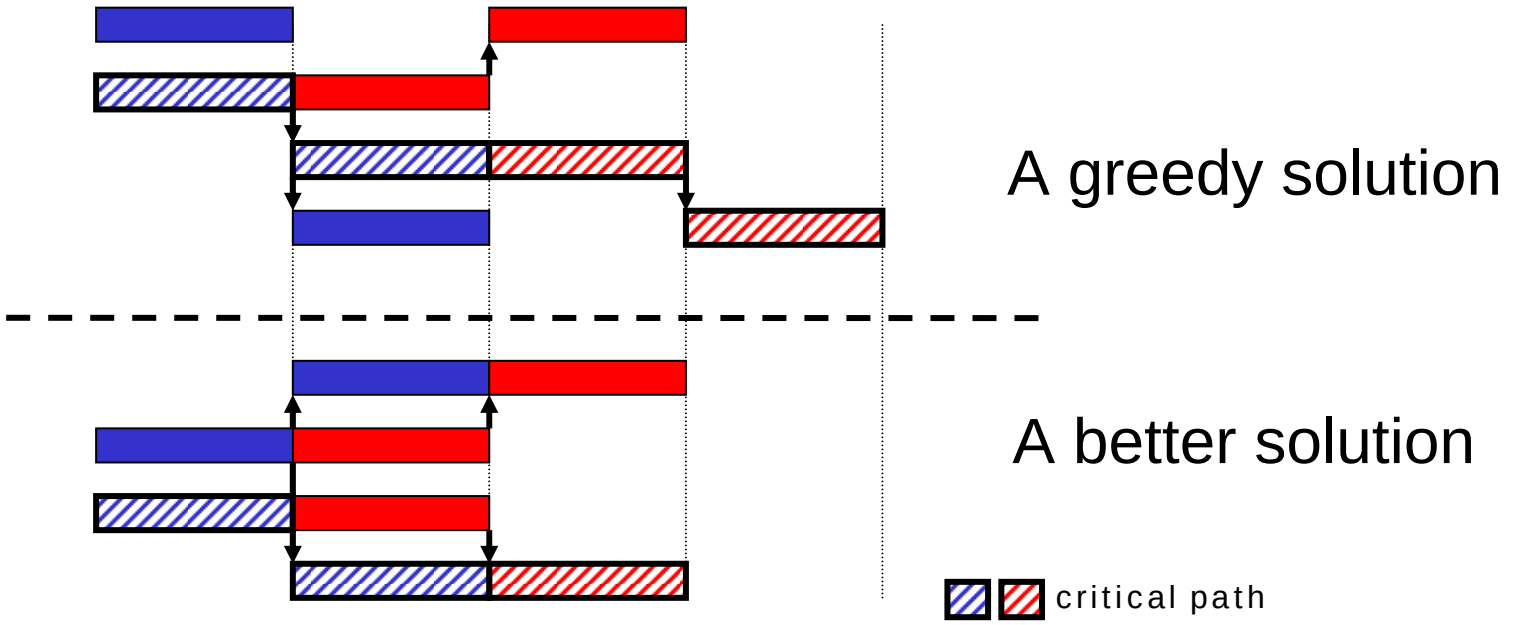


Greedy strategy: example (5)



↘ leveling constraint

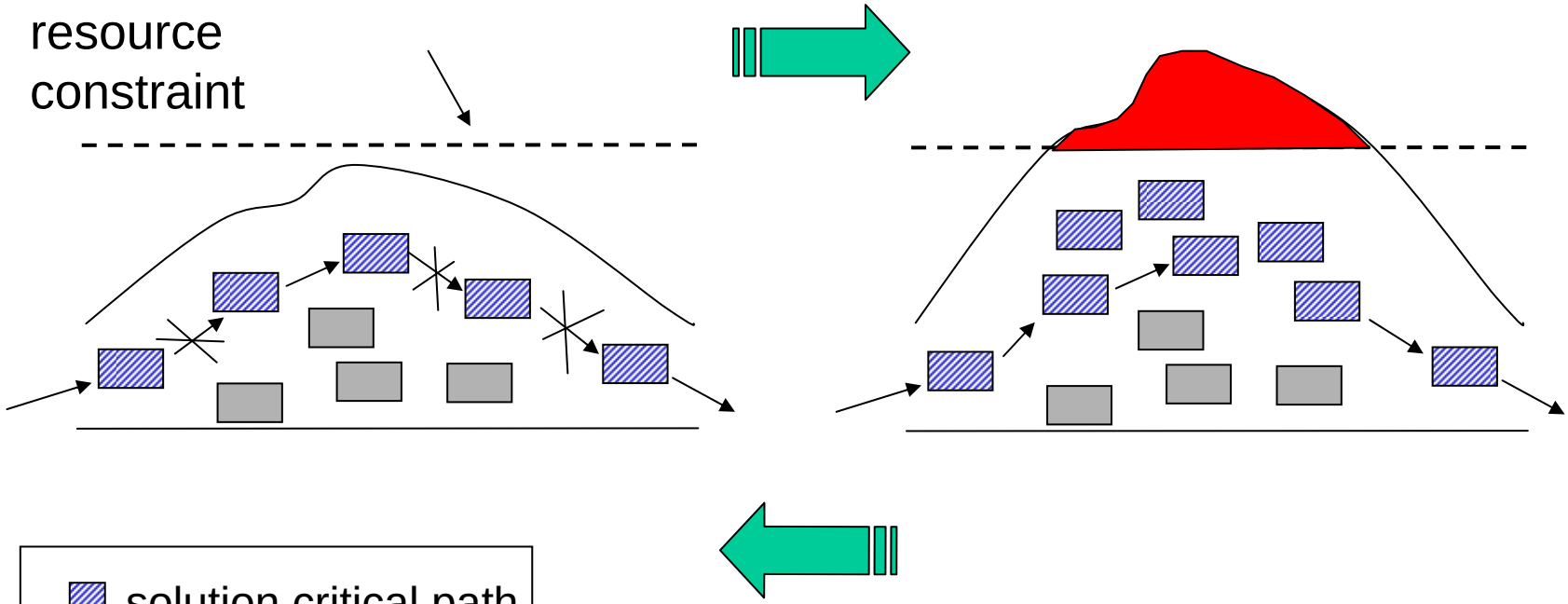
Finding better solutions



- A Greedy solution is not necessarily optimal
- A better solution will necessarily have a shorter critical path
- Implies change to one or more constraints along critical path

Iterative Flattening

random perturbation on
the solution critical path



solution critical path

random removal of
a leveling constraint

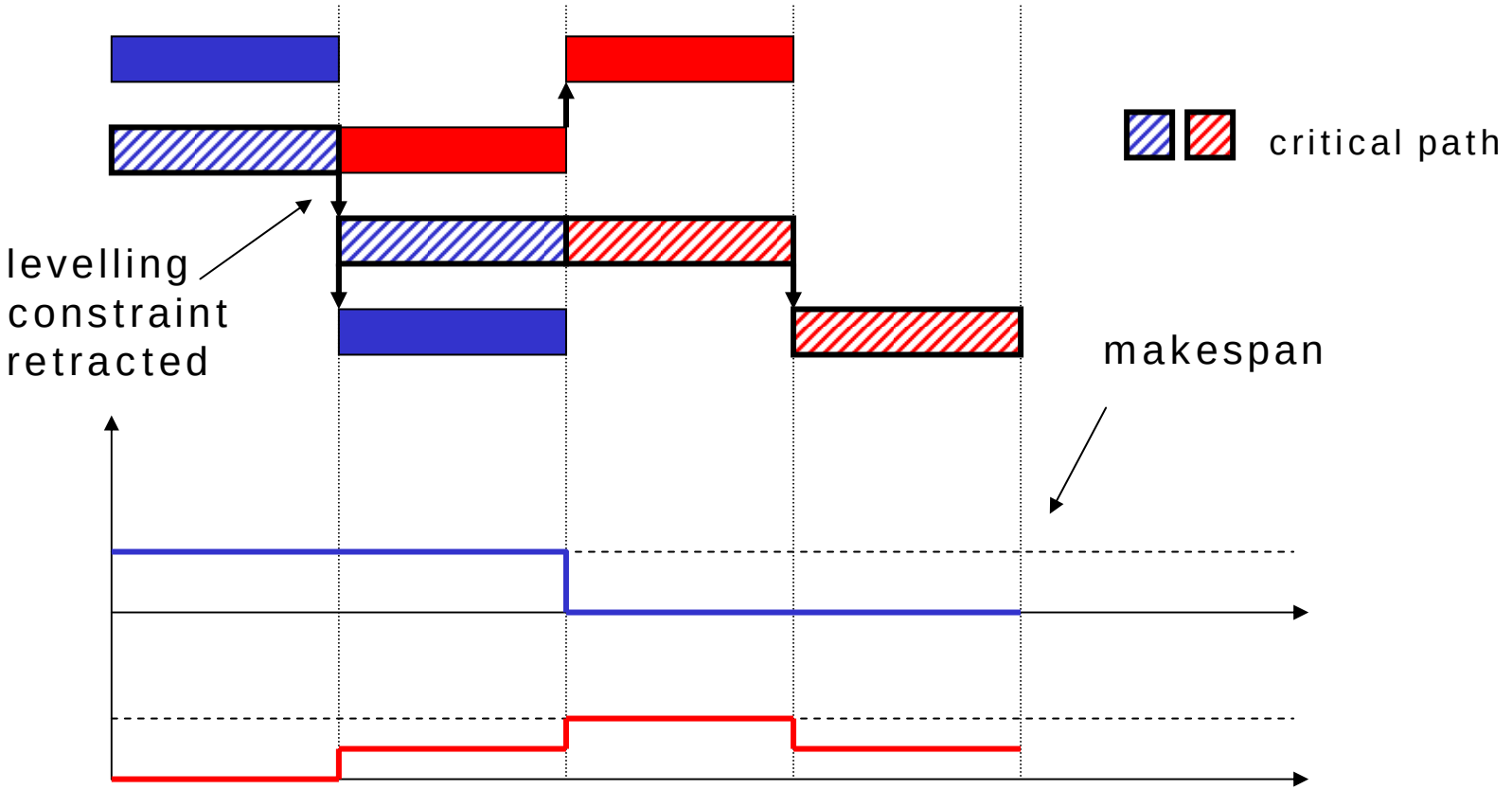
resource leveling (greedy algorithm)

The iFlat algorithm

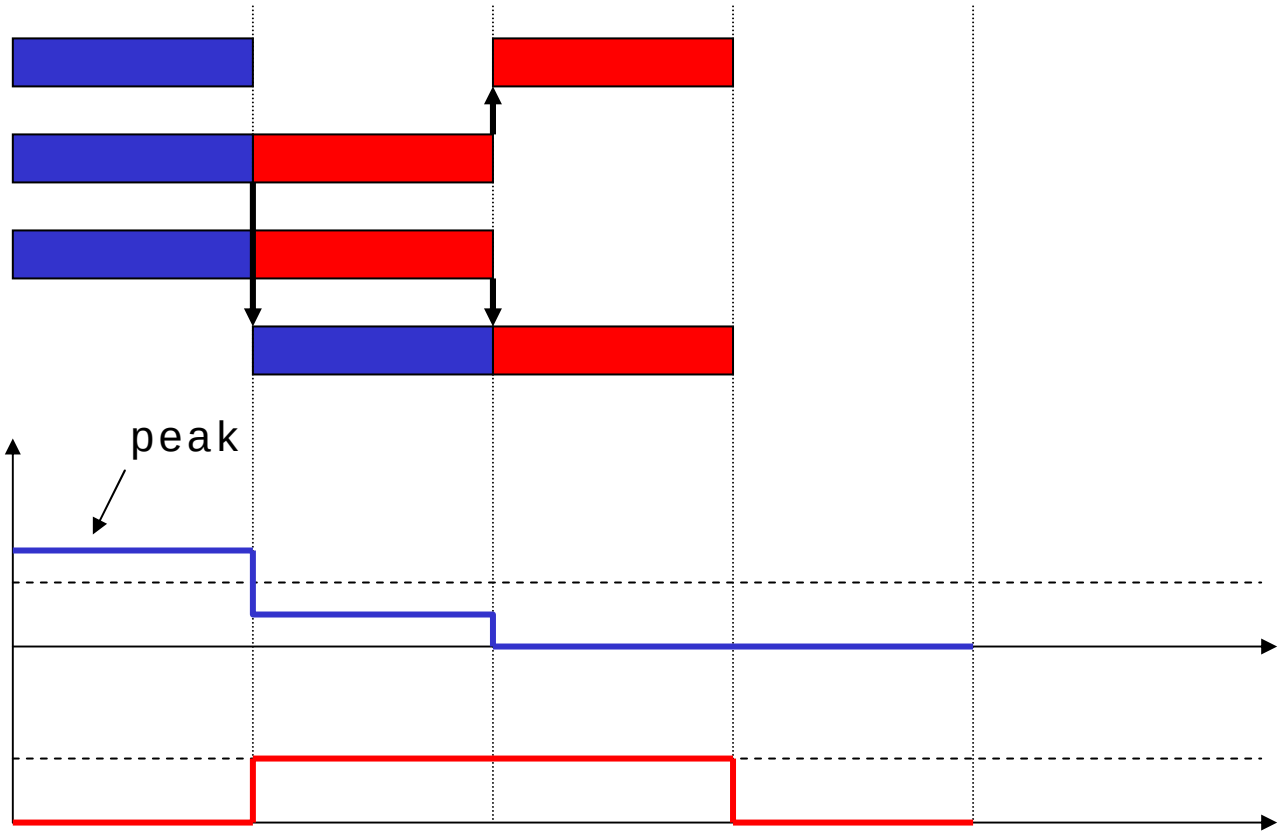
IFlat (*Solution*, P_{rem}, *MaxFail*) **{**

while (the makespan is improved within *MaxFail* iterations*)* **{**
Randomly retract a percentage P_{rem} of leveling constraints on the solution critical path /***removal step***/
Re-apply the ESTA to level (flatten) the new introduced resource conflicts /***flattening step***/
}
return(*Solution*);
}

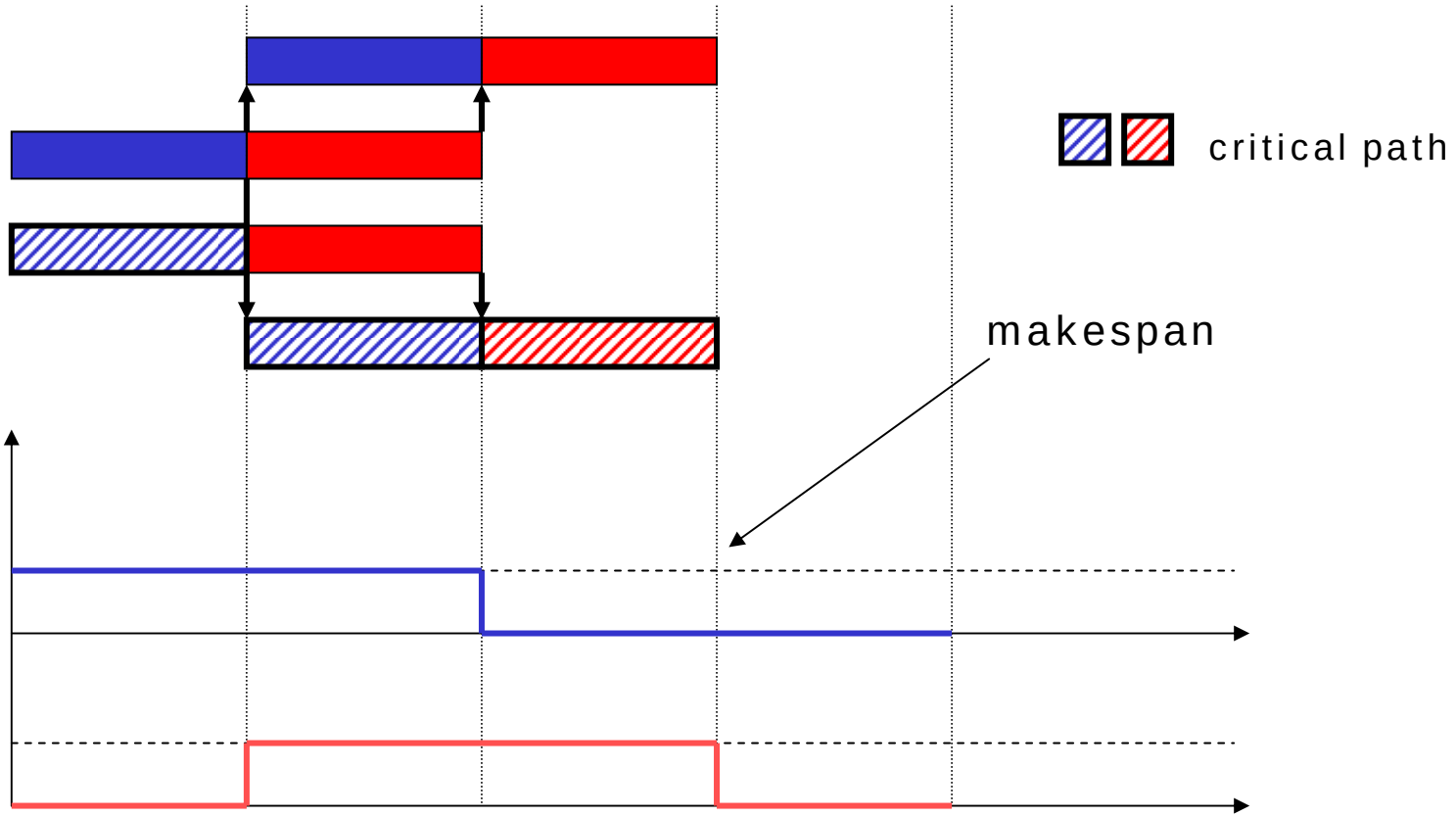
iFlat cycle: critical path analysis



iFlat cycle: shrinking step



iFlat cycle: Flattening step



The improved iFlat: *IFlatRelax (iFlatx)*

IFlatIRelax (*Solution*, P_{rem}, *MaxFail, MaxRelaxations*) **{**

while (the makespan is improved within *MaxFail* iterations*)* **{**

for (*i = 1* **to** *MaxRelaxations)* **{**

Randomly retract a percentage P_{rem} of leveling constraints on the solution critical path /***removal step***/

}

Re-apply the ESTA to level (flatten) the new introduced resource conflicts /***flattening step***/

}

return(*Solution*);

}

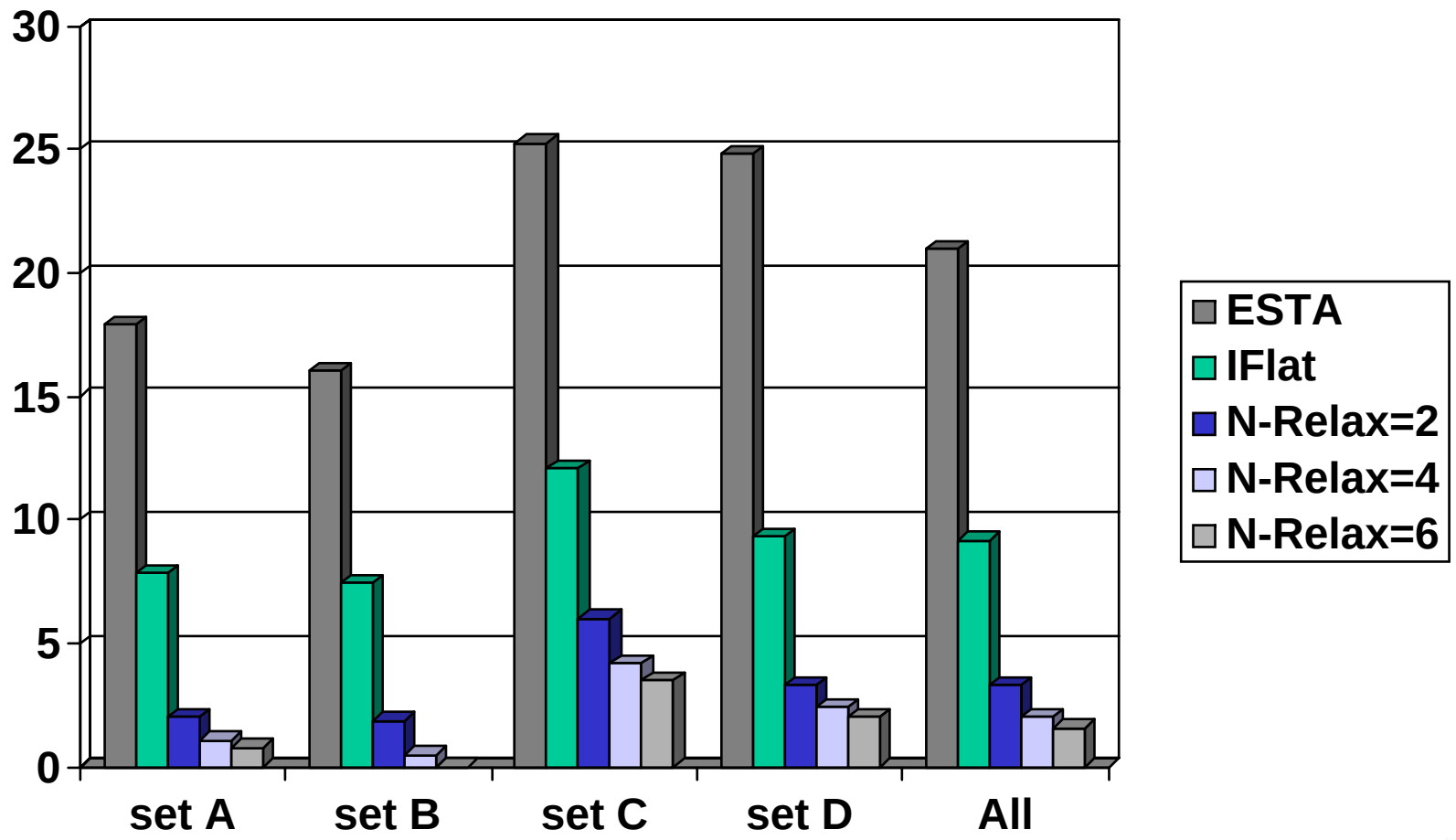
ISTC CNR INSTITUTE FOR COGNITIVE SCIENCE & TECHNOLOGY

MCJSSP: experimental setting

- The benchmark set is partitioned in four subsets of 20 problems:
  - **Set A**: (LA1-10) 100 - 225 activities
  - **Set B**: (LA11-20) 200 - 300 activities
  - **Set C**: (LA21-30) 300 - 600 activities
  - **Set D**: (LA31-40) 450 - 900 activities

- *IFlatIRelax* is implemented in COMET on a Pentium 4 2.4 Ghz [*Michel&Van Hentenryck*, ICAPS-2004]
  - P_{rem} = *20%*, *MaxFail = 5000*
  - **Set A** and **B**: *NumRestarts=100*
  - **Set C** and **D**: *NumRestarts=20* (*10* in some cases)

Makespan: $\Delta UB_{\%}$ from the best UBs



Extending iterative flattening

- The concept of iterative flattening search is quite general and provides an interesting new basis for designing more effective procedures for scheduling optimization

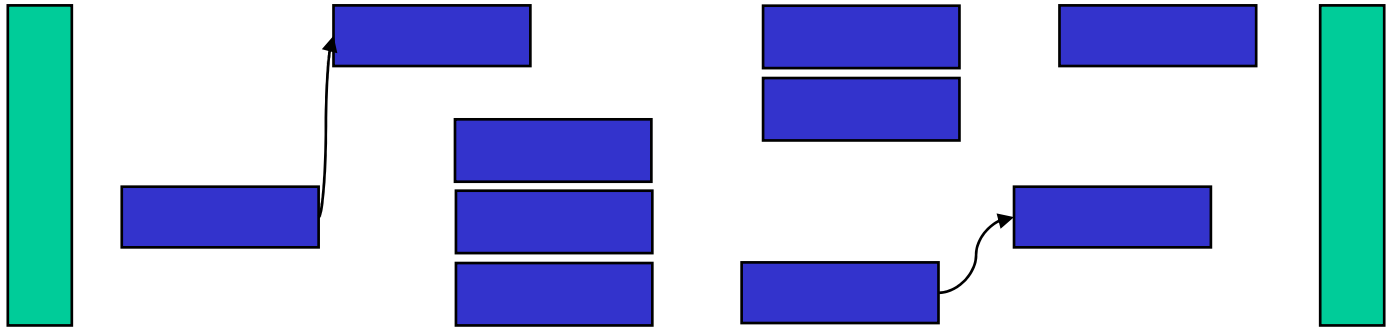
- In the following we describes three possible extensions based on three *drawbacks* identified in the iterative flattening search:
  - *POS schedules* -- A first potential shortcoming is the *lack of temporal flexibility* in the initial solution provided to seed iFlat
  - *Tabu-list* -- A second possible drawback stems from the simple manner in which precedence constraints are selected for retraction, which can lead to *repeated selection of the same constraints*
  - *Tabu Search* -- A third possible drawback is the lack of an ability to conduct a *fine-grained search* when a near-optimal solution is generated by iFlat

A more flexible input solution

- *The basic intuition*: the greater the time flexibility, the higher the probability that new start times for relaxed activities can be found on a given relax-and-flatten cycle that reduce the overall makespan
- We pursue the idea of constructing Partially Ordered Schedules (POSs), such that, each activity retains a set of feasible start times and each time feasible schedule is a feasible solution
  - Activities which require the same resource units are linked via precedence constraints into precedence chains
  - Each posted constraint represents a producer-consumer relation. Each time an activity terminates its execution (producer), it passes its resource unit(s) on to its successors (consumer) and execution continues to move forward
  - In this way, the resulting network of chains can be interpreted as a flow of resource units through the schedule

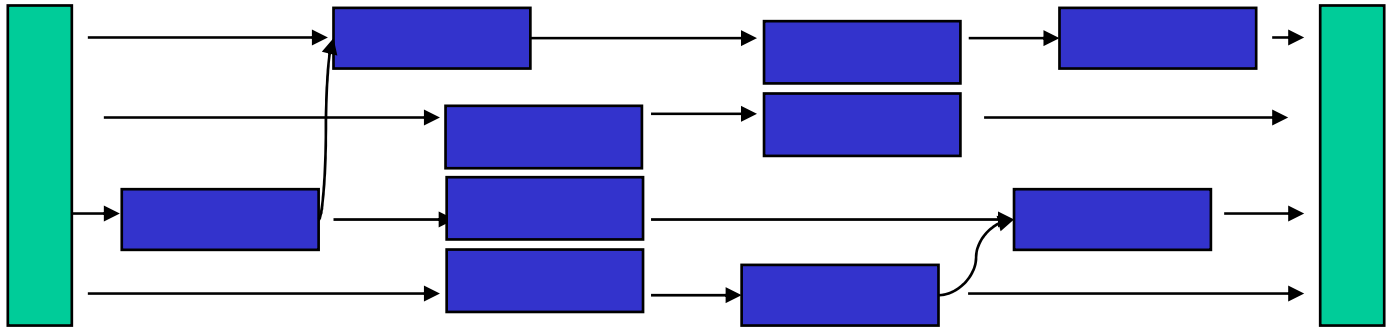
A POS-form solution

Fixed-time solution



C=4

POS-form solution



C=4

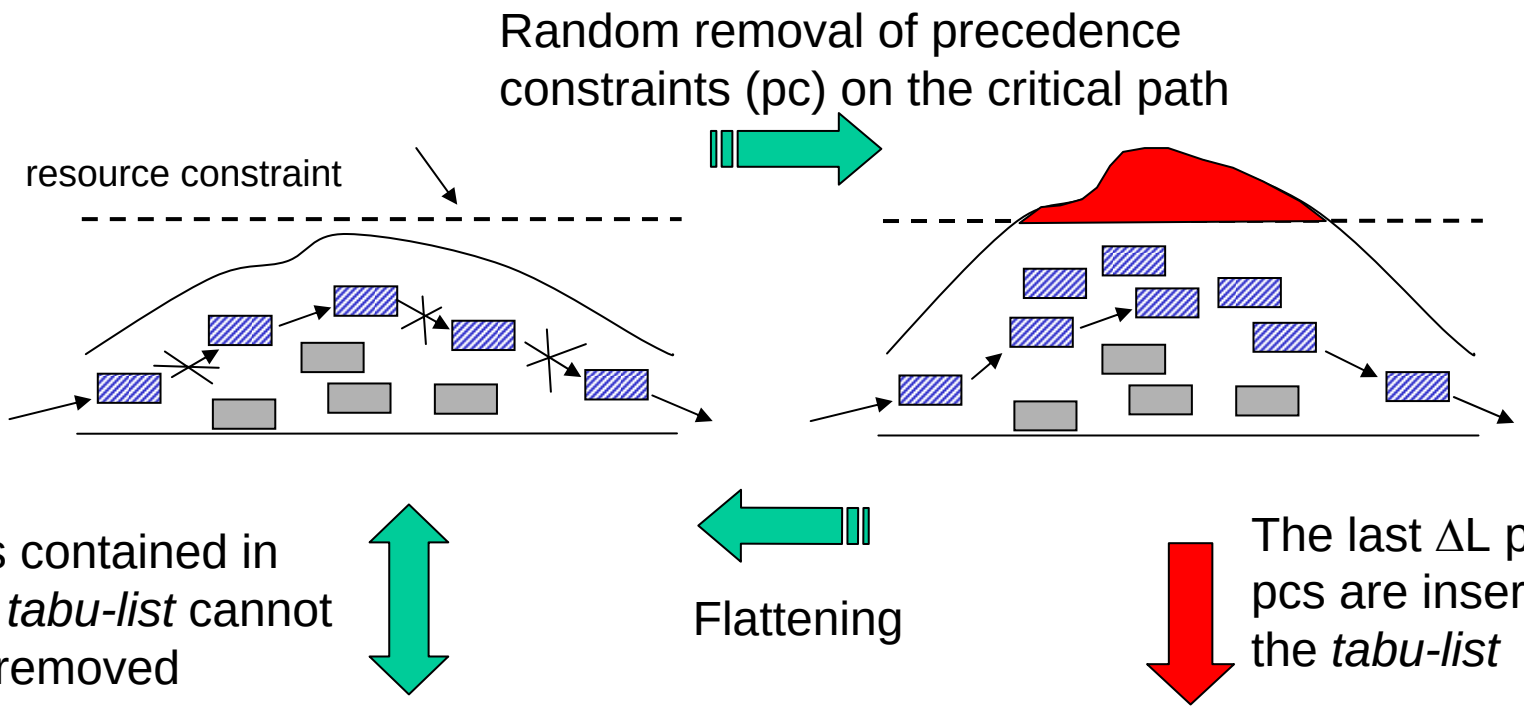
source

sink

Iterative flattening with tabu-list

- A new strategies for selecting candidate decisions within the iFlat *removal phase*
- We propose the use of a *tabu-list* mechanism for avoiding to turn-back to previously explored solutions.
  - At each flattening cycle, a subset of the posted precedence constraints are inserted in the *tabu-list*
  - Precedence constraints contained in the *tabu-list* cannot be retracted during the removal phase
  - Two additional search parameters: the number of precedence constraints ΔL inserted in the tabu list at each flattening cycle and the maximal length of the list L.

IFlatx with tabu-list



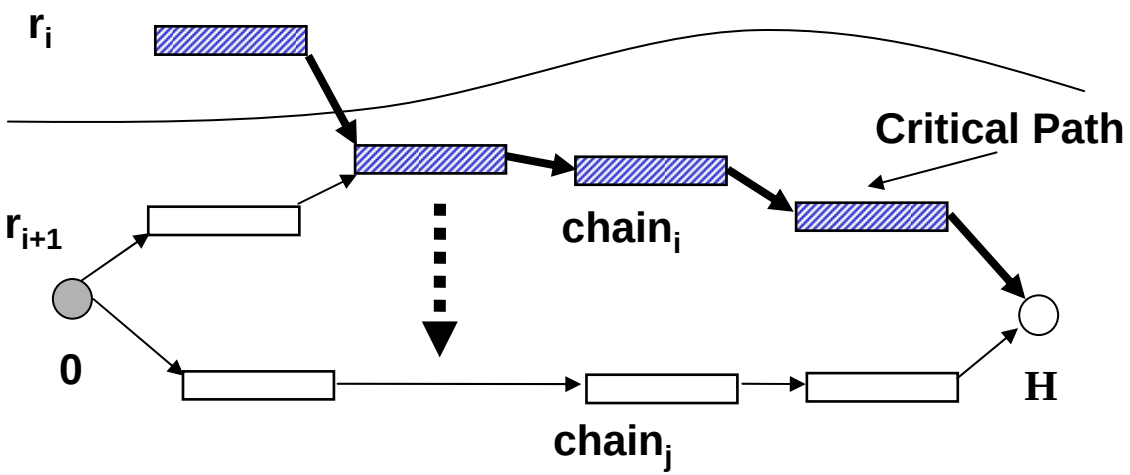
Tabu-list (fifo) with max length L

Tabu Search

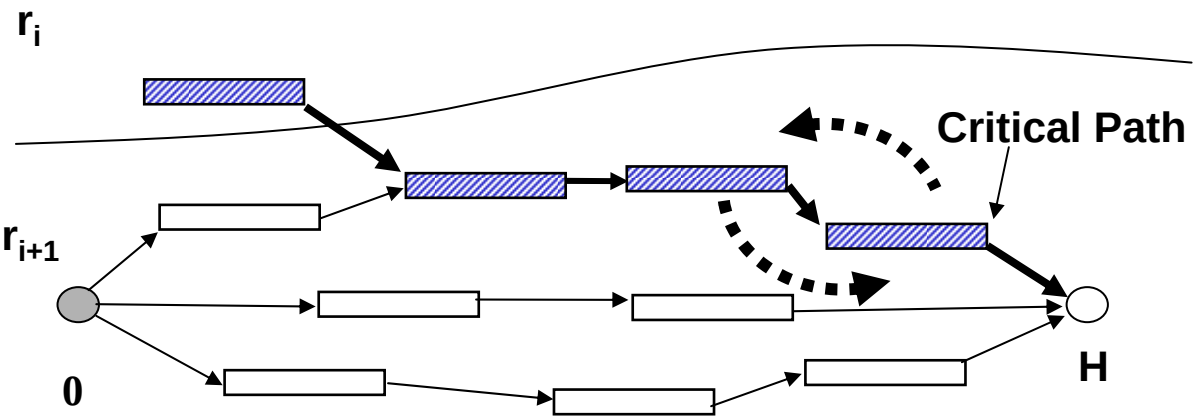
- Tabu search is a meta-heuristic approach to find a near-optimal solution of combinatorial optimization problems
- It needs a fundamental notion called the *move*. The move is a function which transforms a solution into another
- For any solution *S*, a subset of moves *m* applied to *S* is given. This subset of moves induces a subset of solution called the *neighborhood* of *S*
- Tabu search starts from an initial solution S_0, and at each step *i* the neighborhood of a given solution is searched in order to find a *neighbor* S_i that has the best value of a fixed objective function
- In order to prevent cycling, it is not allowed to turn back to chosen solutions visited in the previous *MaxSt* steps. Where *MaxSt* is the max length of the so-called *tabu list* which is a queue with limited length.

Two types of move

Vertical move



Horizontal move

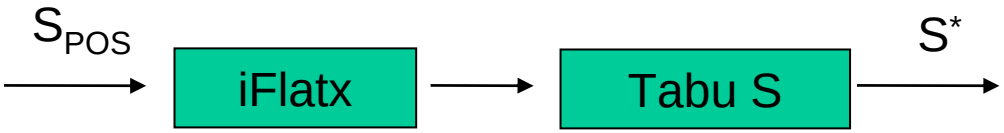


iFlat extensions

POS

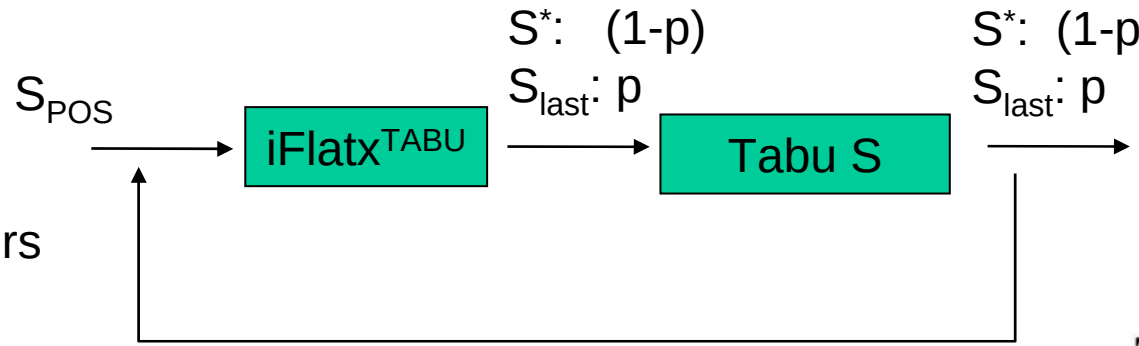
POS-TABU

POS-TABU-SEARCH



S^* is the best solution found by the component strategy, S_{last} is the last one

LOOP



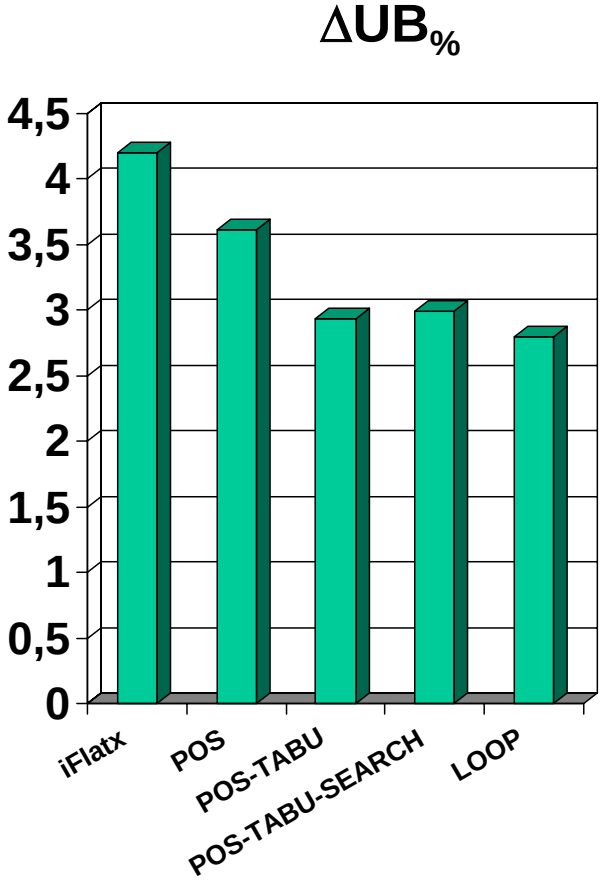
p is a noise parameters in [0,1]

Comparing iFlatx extensions

- Benchmaks: set A, B, C, D
- *iFlatx* extensions:
  - POS: *Iflatx* with a POS-form input solution
  - POS-TABU: *pos* with the *tabu-list*
  - POS-TABU-SEARCH: the best solution found by *pos* is serialized with the *tabu-search* algorithm
  - LOOP: interleaves *pos-tabu* with *tabu-search*
- A two-step evaluation: preliminary and intensive

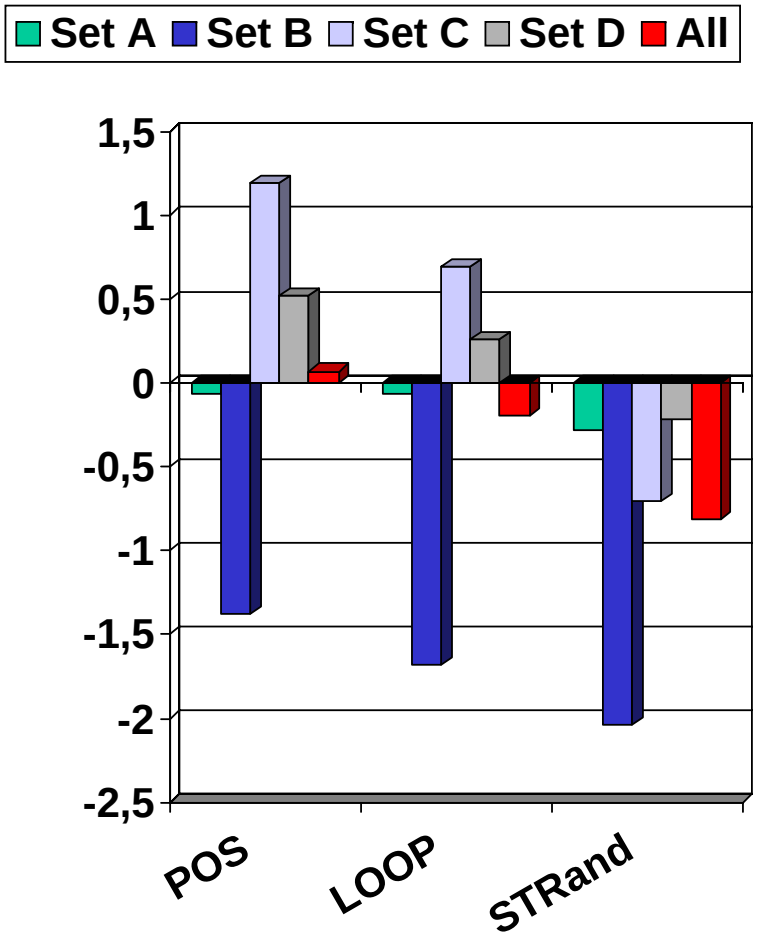
Preliminary evaluation

- $\Delta UB_{\%}$ is the percentage deviation from the Lawrence upper bounds
- Only **Set C**, *Tout = 1000 sec*
- P_{rem} *= 0.2*, *MaxFail=400* and *MaxRelaxations = 6*
- When a tabu-list is used its length *l = 16* and Δl *= 16*
- The Tabu Search parameters: tabu-list's length = *9; init-move= 'vertical'; maxintrlv = 1; maxiter = 50*
- The noise value for the *Loop* integration was set to *p = 0.2*



Intensive evaluation

- $\Delta UB_{\%}$ for Set A, B, C and D (80 instances)
- *Tout = 8000* sec
- P_{rem} *= 0.2*, *MaxFail = 1000*, *MaxRelaxations = 6*
- The same parameters for the tabu list, tabu-search and the loop integration
- An additional comparison with the results shown in [Godard, Laborie, and Nuijten. Randomized Large Neighborhood Search for Cumulative Scheduling (ICAPS-2005)]: **STRand**



Conclusions

- Iterative Flattening (iFlat) is a iterative improvement search procedure for solving multi-capacitated scheduling problems with makespan minimization as the objective
- The approach is quite general and is applicable to a range of cumulative scheduling problems
- Previous work have extended or proposed variations of the original iFlat procedure. In both cases these new procedures were able to find *new optimal solutions* or to improve known upper-bounds for problem instances in the MCJSSP

Conclusions

- We recently started some further investigation on variants of iFlat. These extensions were motivated by three potential limitations:
  - the lack of flexibility in the initial seed solution
  - the potential for repeatedly searching the same solution subspace
  - The inability of iFlat to explore the close neighborhood of a near-optimal solution
- The proposed extensions were found to significantly improve the performance of the reference strategies (IFlatRelax)
- Further study will be necessary to clearly understand the effectiveness of the algorithms proposed, especially with regard to the best results available (STRand)
- However, we believe that the proposed extensions are quite general and can be also usefully used within the STRand algorithm

Future work

- One particular interest is investigation of a more sophisticated *tabu-list* mechanism, which biases the tenure value according to the estimated quality of a given constraint
- Another general focus will be exploration of alternative approaches to integrating iterative flattening and tabu search. In this regard, we believe a *Back Jumping Tracking* schema (Nowicki & Smutnicki 1996), where search is restarted from promising solutions accumulated during the search, holds particular promise
- A third direction of research is the resolution of more complex scheduling problems, like the *Resource Constraint Scheduling Problem* (RCPSP)