ICAPS 2006

The English Lake District, Cumbria, UK

TU3

Tutorial on Meta-heuristics for Solving Scheduling Problems

Angelo Oddi

Institute of Cognitive Science and Technology Italian National Research Council (ISTC-CNR), Italy

ICAPS 2006

The English Lake District, Cumbria, UK

TU3

Tutorial on Meta-heuristics for Solving Scheduling Problems

Angelo Oddi

Institute of Cognitive Science and Technology Italian National Research Council (ISTC-CNR), Italy

ICAPS 2006
Tutorial on Meta-heuristics for Solving Scheduling Problems

Table of contents

4.2 Iterative Random Sampling

4.3 Basic Local Search

4.4 Composite strategies

5. Conclusions

6. References

Angelo Oddi

---

http://icaps06.icaps-conference.org/

ICAPS 2006
Tutorial on Meta-heuristics for Solving Scheduling Problems

Preface

Despite much progress has been made in finding exact and provably optimal solutions to scheduling problems, many hard scheduling problems are still not solved exactly and require heuristic methods. In addition, reaching optimal solutions is in some cases meaningless, as in practice we are often dealing with models that are rough simplifications of the working domain.

This tutorial introduces methods for solving scheduling problems that combine heuristic search and constraint reasoning. Specifically, its goal is to explain how heuristic search, in combination with constraint reasoning techniques, has emerged as a robust methodology to quickly produce good-quality solutions for a variety of scheduling problems. The focus is first put on greedy algorithms based on temporal flexibility heuristics and to local search approaches to scheduling optimization, including neighborhood structures and heuristics for improving search efficiency. Secondly, some meta-heuristics will be described, i.e., high level procedures that coordinate and combine simple heuristics to find solutions that are of better quality than those found by the simple heuristics alone. The tutorial is targeted to researchers and practitioners that would like to use meta-heuristic techniques for solving scheduling problems. No prior knowledge on meta-heuristics or constraint reasoning is required.

Angelo Oddi

Instructor

- *Angelo Oddi is a research scientist at the Institute of Cognitive Science and Technology of the Italian National Research Council (ISTC-CNR). He received his Master Degree in Electronic Engineering from University of Rome "La Sapienza" in 1993 and his PhD in Medical Computer Science from the same university in 1997. He has been visiting scholar at the Intelligent Coordination and Logistics Laboratory of the Robotics Institute at Carnegie Mellon University in 1995-6. His work focuses on the application of Artificial Intelligence techniques for scheduling, automated planning and constraint reasoning. In particular, he has proposed several algorithms for temporal reasoning and developed both local search and randomized approaches for schedule optimization. Regarding his professional activities, he has published more than 40 papers, both in journals and in proceedings of international conferences, and has a wide experience in the design of intelligent systems for real world applications. In particular, he has been involved in several projects financed by the Italian and European Space Agencies (ASI/ESA) concerning the development of intelligent mission planning support software.*

Meta-heuristics for solving scheduling problems

Angelo Oddi
ISTC-CNR, Institute for Cognitive Science and Technology, Italian National Research Council
angelo.oddi@istc.cnr.it

Outline

- Introduction
  - Scheduling
  - Meta-heuristics
  - Constraint-based reasoning
- Basic elements
- Constructive methods
- Meta-heuristics
- Conclusions

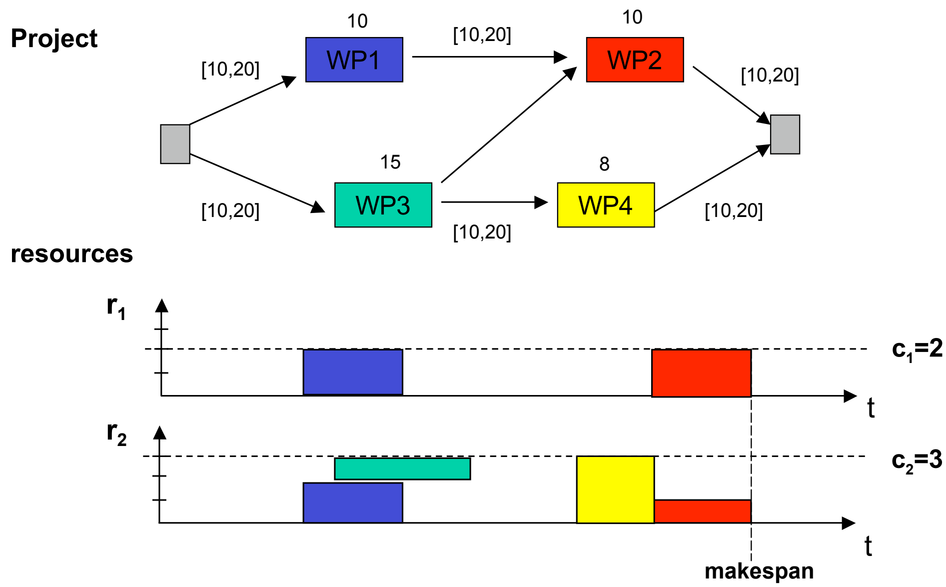
Scheduling Problems

- Scheduling problems arise in many application fields: manufacturing, transportation, communication, project management, etc
- Scheduling is an important tool for manufacturing and service industries, where it can have a major impact on the productivity of a process
- In manufacturing, the purpose of scheduling is to minimize the production time and costs, by telling a production facility what to make, when, with which staff, and on which equipment
- Similarly, scheduling in service industries, such as airlines and public transport, aims to maximize the efficiency of the operation and reduce costs

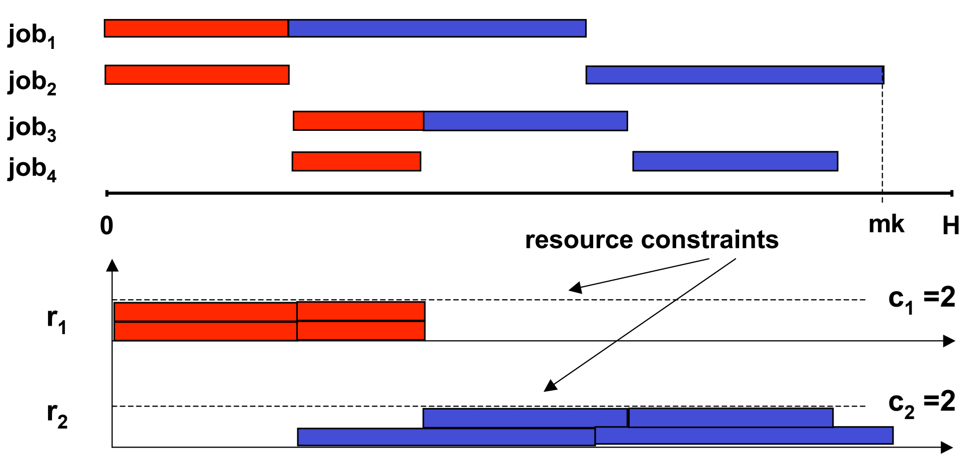
Scheduling

- A scheduling problem is generally formulated as a set of **resources** (machines, channels, money, etc) and a set of **activities** (or jobs) which use the resources. The problem is to find a temporal assignment to all the activities of the plans which is consistent with all the time and resource constraints
- In few words, scheduling is the *“problem of allocating scarce resources to activities over time”* [Baker 74]
- Within this tutorial we mainly refer to a quite general class of scheduling problems, the so-called *Resource Constraint Scheduling Problem with Time Windows (RCPSP/max)* [Neumann&Schwindt 97]. We also consider subclasses of RCPSP/max

An example of RCPSP/max

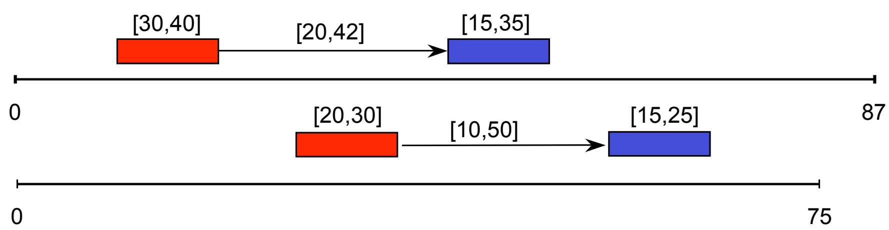


The MCJSSP scheduling problem



An example of scheduling problem [Cesta Oddi&Smith 00] with **four jobs**, each job has two activities; “red activities” require resource $\mathbf{r_1}$ and “blue activities” require resource $\mathbf{r_2}$.

Job Shop Deadline Scheduling Problem



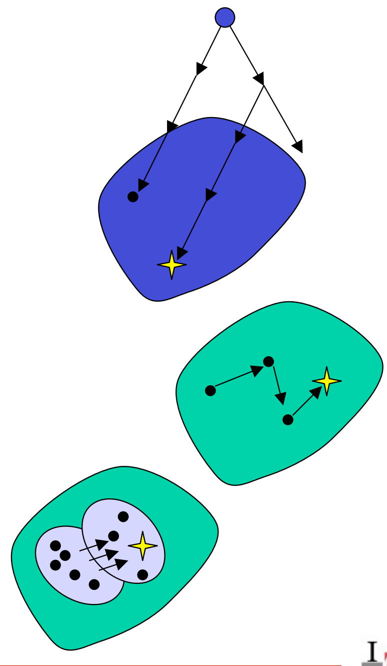
- An example of scheduling problem [Smith&Cheng 94] with two jobs, each job has two activities."Red activities" require resource $\mathbf{r_1}$ and "blue activities" require resource $\mathbf{r_2}$
- Each activity requires only one resource and at any instant a resource can execute only one activity. The processing time is an interval of possible values
- Between a couple of successive activities there is an interval of temporal separation
- Each job has a ready time time and a deadline

Meta-heuristics

- Two fundamental goals in computer science are finding algorithms with provably *good run times* and with provably *good or optimal solution quality*
- A **heuristic** is an algorithm that reaches one or both of these goals
- A **meta-heuristic** is a solving method combining given component procedures — usually heuristics themselves — in a *hopefully* effective and efficient way
- At least two *motivations* for using meta-heuristics:
  - Despite much progress has been made in finding exact and provably optimal solutions to scheduling problems, many hard scheduling problems are still not solved exactly and require heuristic methods
  - Reaching optimal solutions is in some cases meaningless, as in practice we are often dealing with models that are rough simplifications of the working domain

Different heuristic methods

- *Constructive* methods
  - Greedy algorithms
    - Deterministic priority rules
  - Bounded Systematic Search
    - Branch&Bound
    - Limited Discrepancy Search (LDS)
  - Random Sampling
- *Improving* methods [Blum&Roli 03]
  - Local Search (trajectory-based)
    - Tabu search
    - Simulated Annealing
    - …
  - Population-based
    - Ant Colonies Optimization
    - Genetic Algorithms
    - ….

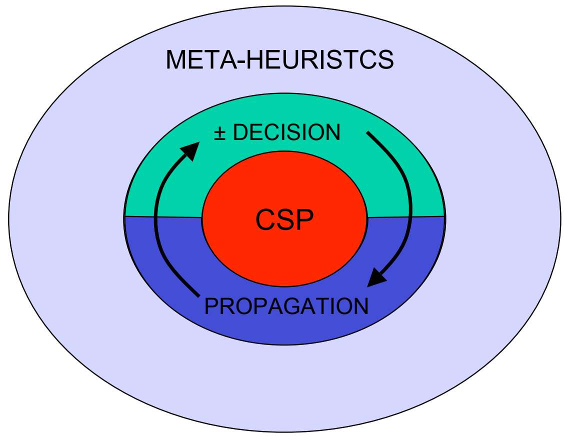


Constraint-based reasoning

- Problem solving = *reasoning with constraints*
  - This solving paradigm clearly separates the *constraints* (semantics, pruning algorithms) from the *search space exploration* (branching schemes, heuristics)
    - *What* to solve
    - *How* to solve
- A Constraint Satisfaction Problem (CSP) is defined as:
  - *A set of variables* representing elementary decisions
  - A set of *constraints* on the decision variables
- A Constraint Optimization Problem (COP) is defined as:
  - A CSP
  - An objective function on the set of solutions
- A *solution* is un a set of elementary decisions which satisfies all the constraints
- An *optimal solution* is a solution which minimize the objective

The reference framework

- A **meta-heuristic** can be seen as a combinations of basic heuristic methods
- A basic heuristic method applies two steps:
  - A *propagation* (*inference)* method on the set of decision variables, which prunes a subset of infeasible choices
  - *A decision* method
- A single *decision* can:
  - *Add* a constraint
  - *Retract* a constraint



Contribution of the tutorial

- We define a **scheduling problem** as:
  - a Constraint Satisfaction Problem (CSP) or
  - a Constraint Optimization Problem (COP)
- We propose meta-heuristic schemas which combine basic **constructive** and **local search methods** within a **CSP** reference framework
- The definition of **meta-heuristic** schemas is driven by two key concepts:
  - The definition, representation and use of **control knowledge** to drive the search (better if domain independent)
  - The balancing between **intensification** and **diversification**:
    - *Intensification* means to search carefully and intensively around good solutions found in the past search
    - *Diversification***,** on the contrary, means to guide the search to unvisited regions of the search space

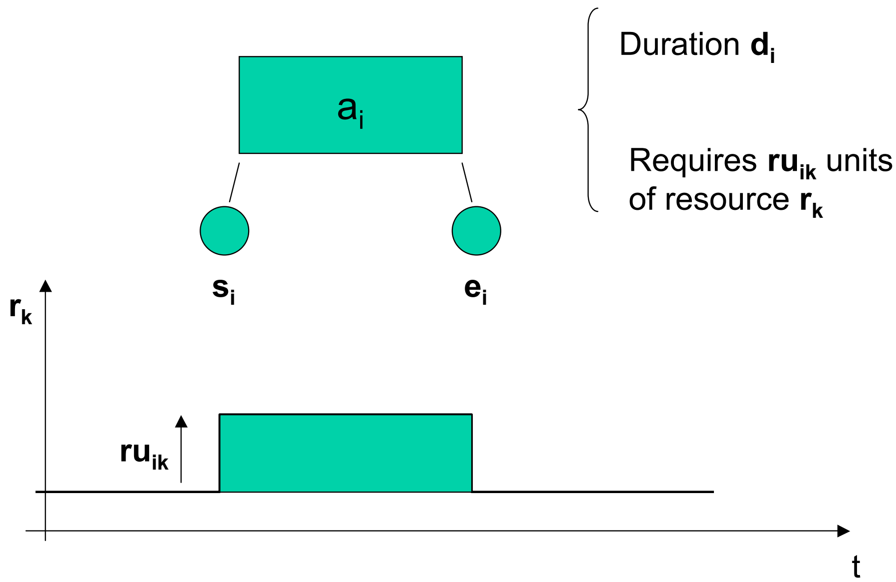
Outline

- Introduction
- Basic elements
  - A reference scheduling problem
  - Complexity
  - Constraint-based scheduling
- Constructive methods
- Meta-heuristics
- Conclusions

A reference scheduling problem

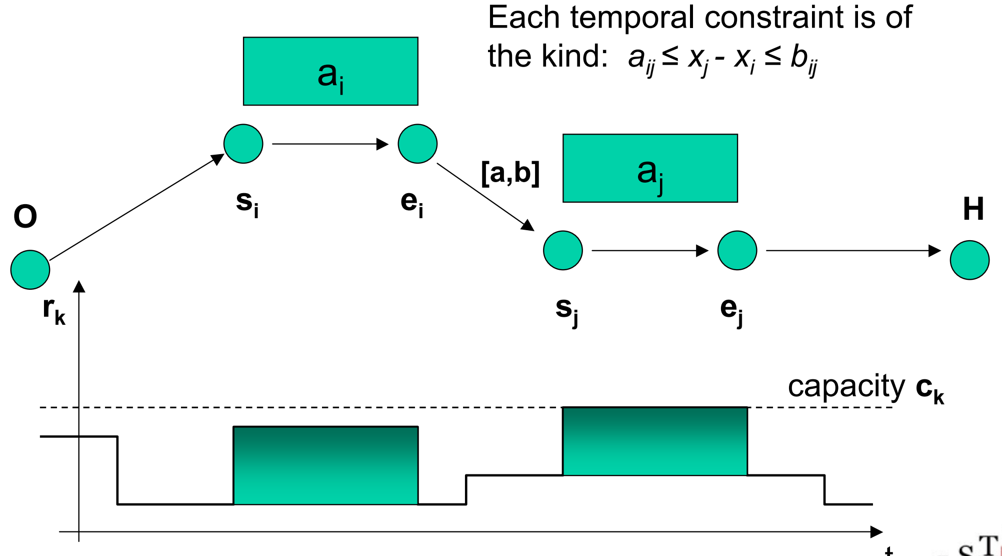
- Activities
  - Non-preemptive
  - Constant resource usage
  - Start-Time, End-Time
- Temporal constraints
  - Simple Temporal Constraints (STP) [Dechter&al 91]
- Resources constraints
  - Discrete resources

Scheduling problem: activity



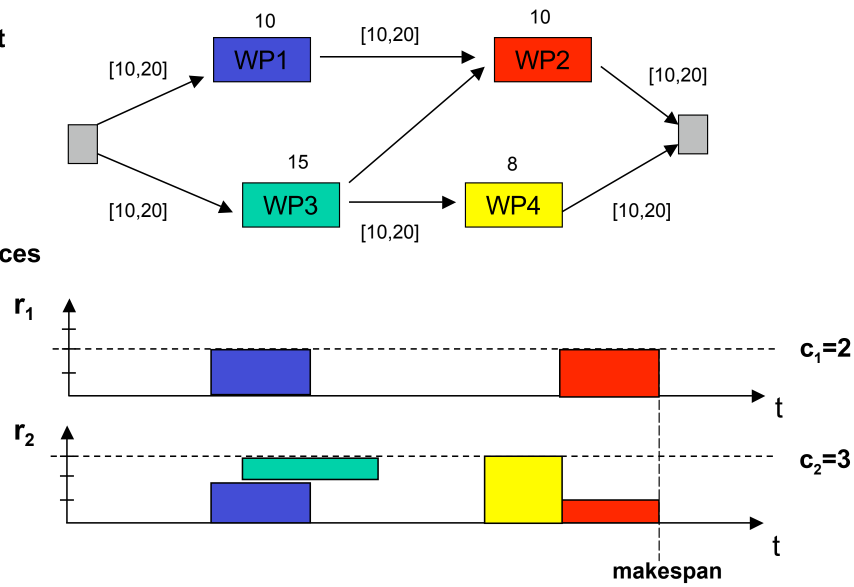
Scheduling model: constraints

Each temporal constraint is of the kind: $a_{ij} \leq x_j - x_i \leq b_{ij}$



A reference scheduling problem

Project

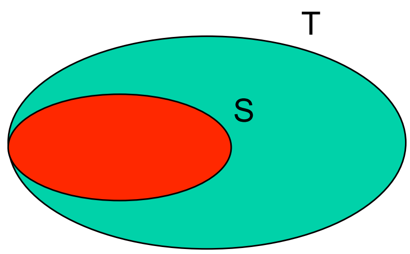


A reference scheduling problem

- A set of activities A= {a_1, a_2,…,a_n}, each activity a_i
  - has a fixed duration $dur(a_i)$
  - requires the use of ru_{ik} units of resource r_k during its execution
- A set of resource R, each resource r_k has an integer capacity $c_k \geq 1$
- There are a set of temporal constraints between pairs (a_i,a_j):
  $lb_{ij} \leq st(a_j)\text{-}st(a_i) \leq ub_{ij}$
- For each time t, the total amount of resource required by the set of activity in execution must be less or equal to c_k
- A *solution* is an assignment to all activity start times, which satisfies both the temporal and the resource constraints
- An *optimal solution* is a solution with minimal makespan

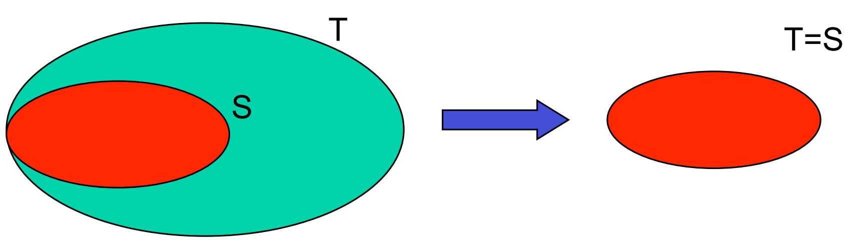
Schedules and Solutions

- A schedule is an assignment to each activity start-time
- A time feasible schedule satisfies all the temporal constraint. Let T be the set of all time feasible schedules
- A feasible schedule satisfies both the temporal and the resource constraints. Let S be the set of all feasible schedules.
- An earliest start schedule (or semi-active schedule) is a feasible schedule in which each activity is allocated in its earliest start time.



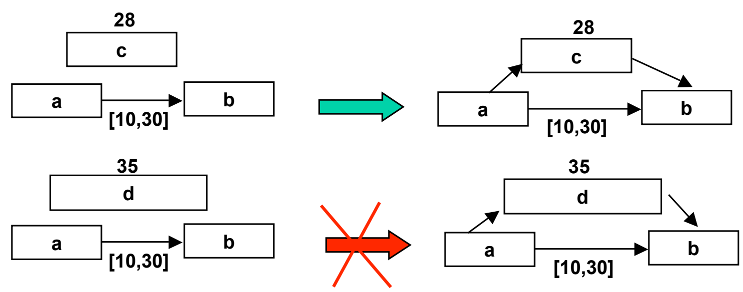
Partial Order Schedule (POS)

- We also pursue the idea of constructing Partially Ordered Schedule (POS) instead of fixed-time schedule
- *A POS can be defined as a set of additional precedence constraints imposed on the original problem, such that each activity retains a set of feasible start times and each time feasible schedule is also a feasible solution*
- Algorithms to make POS schedules are described in the following as components of meta-heuristics procedures
- Fixed-time solutions and POS are equivalent solutions, that is exist a polynomial transformation to convert one into another



Complexity

- Finding a feasible solutions
  - *NP-hard* [Bartusch&al 88] when there are maximal temporal constraints: $a_{min} \leq x_j - x_i \leq b_{max}$
  - *P* otherwise: $a_{min} \leq x_j - x_i \leq +\infty$
- Finding *makespan optimal* solutions
  - The problem is *NP-hard* [Garey&Johnson 79] in both cases
- The existence of *maximum separation constraints* makes the problem particularly hard, *intuition*:

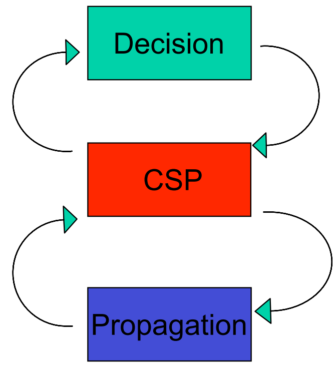


Constraint Satisfaction Problem

- An instance of CSP [Montanari 74] involves
  - a set of Decision Variables $X = \{X_1, X_2, \ldots, X_n\}$
  - a Domain of possible values D_i for each variable
  - a set of Constraints $C = \{C_1, C_2, \ldots, C_q\}$, such that $C_j \subseteq D$, with $D = D_1 \times D_2 \times \ldots \times D_n$
- A solution is an assignment of domain values to all variables consistent with all the constraints C_j
- Given an objective function $f: D \rightarrow Z^+$, an optimal solution is a solution which minimize f

Constraint-based problem solving

- A solution method applies two basic steps:
  - A *propagation* (*inference)* method on the set of decision variables, which prunes a subset of infeasible choices
  - *A decision (heuristic)* method
- A *solution* is generated by interleaving propagation and decision steps

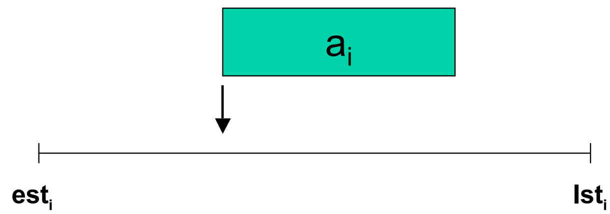


Two examples of decision variables

- Start times (ST)
  - A decision variable is the *start-time* of a generic activity a_i
  - A value is a specific start time assignment
  - A *solution* is an assignment to all activity start times which satisfy both time and resource constraints
- Minimal Critical Set (MCS)
  - A decision variable is a *subset of activities* competing for the same resource requiring more than the resource capacity (conflict) and such that each subset requires no more than the available capacity
  - A value is a single precedence constraints (a_i, a_j)
  - A *solution* is a set of additional precedence constraints imposed on the original problem, which admits at least one start-time solution calculable in polynomial time

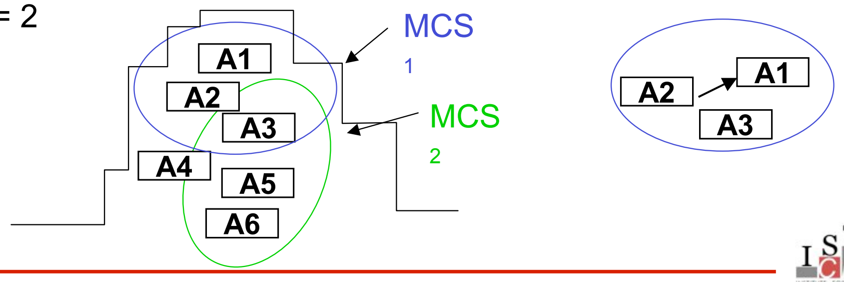
Two examples of decision variables

Set **Start-time**



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

Capacity = 2



An algorithm for solving a CSP

Solve(*CSP***)**
    Propagate(*CSP*)
    if ($\exists D_i = \varnothing$)
        then Return(*failure*)
        else if (*all decision variables* X_i *are set*)
            then Return(*solution*)
            else
                $X_i \leftarrow$ Select-decision-variable(*CSP*)
                $v \leftarrow$ Choose(select-value(D_i))
                set(v,X_i)
                Solve(*CSP*)

Two basic components

- Given the representation of the scheduling problem as a *CSP = ⟨X, D, C⟩: decision variables X, domains D and constraints C*
- Two basic components characterize each solving algorithm:
  - **Heuristics**
    - *Variable* ordering
    - *Value* ordering
  - The set of **propagation** functionalities to remove infeasible elements from the domains D_i . Two different kind of constraints:
    - Temporal constraints
    - Discrete resource constraints

Heuristics

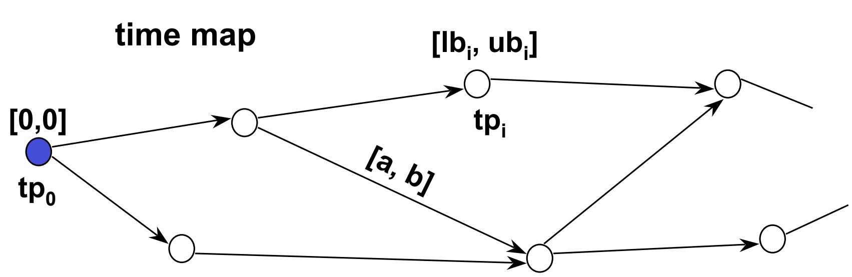
- *Variable* ordering
  - Choose the **most constrained variable**, the variable that is difficult to instantiate: *"start with the difficult part of the problem before it get even more difficult!"*
    - Choose the variables with smallest remaining domain
    - Choose the variables with maximal degree in the constraint graph representation
- *Value* ordering
  - Choose the **least constrained value**, the value that leaves as many values as possible for the remaining not instantiated variables
    - Choose the value that participates in the highest number of estimated solutions (e.g., number of solutions can be estimated on relaxations of the original problem)
    - Choose minimal values

Propagation

- Propagation procedures make explicit constraints which are implicitly contained in the current solution and prune subset of infeasible choices for the decision variables
- In the following we briefly describe a set of procedures to propagate temporal and resource constraints
  – Temporal constraints are *binary constraints* among the activities and the set of temporal constraints imposed on a scheduling problem represents a so-called *Simple Temporal Problem (STP) [Detcher&al 91]*
  – Whereas resource constraints are *n-ary constraints* imposed on subsets of activities. Propagation methods are able to deduce new time bounds and/or new precedence constraints

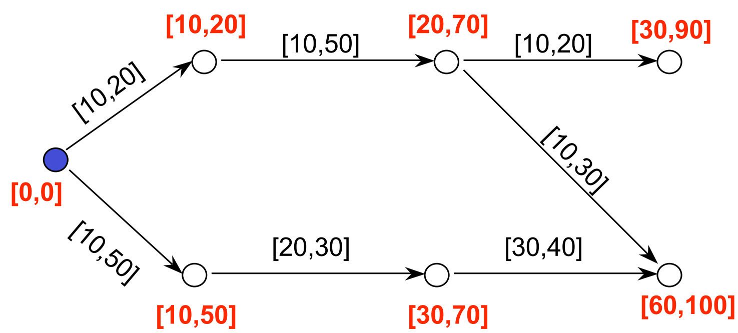
Simple Temporal Problem (STP)

- *A Simple Temporal Problem* is a special case of CSP. It is a set of *n* **variables** (time points) $\{tp_i\}$ with domain $[lb_i, ub_i]$ and a set of **constraints** $\{a \leq tp_j - tp_i \leq b\}$
- There is an additional time point $\mathbf{tp_0}$ called time origin with domain *[0,0]*
- The problem is *consistent* when an instantiation of the variables $\{tp_i\}$ exists such that satisfies all the constraints
- A *time-map* represents a *Simple Temporal Problem*



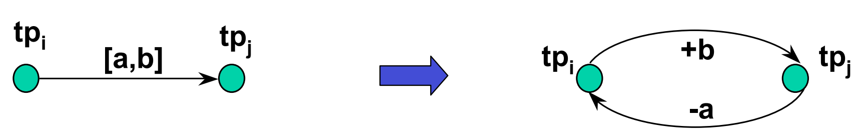
An example of time-map

- Each tp_i has domain $[lb_i,ub_i]$
- All the constraints are of the kind $a_{ij} \leq tp_j - tp_i \leq b_{ij}$
- The time point $\boldsymbol{tp_0}$, called time *origin,* has domain *[0,0]*



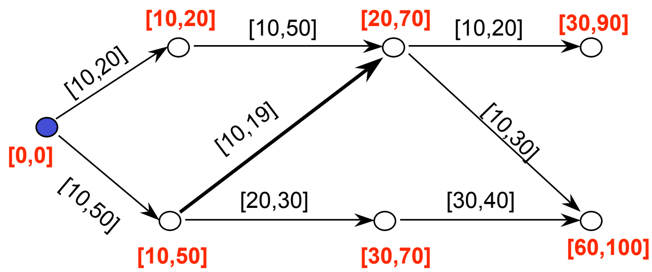
STP as a Shortest Paths Problem

- STP problem can be reduced to a shortest paths problem on a graph $G_d(V,E)$, where V is the set of time points and E is the set of labelled edges such that:



- An STP is <u>inconsistent</u> **iff** G_d contains at least a cycle with negative length [Dechter&al 1991].

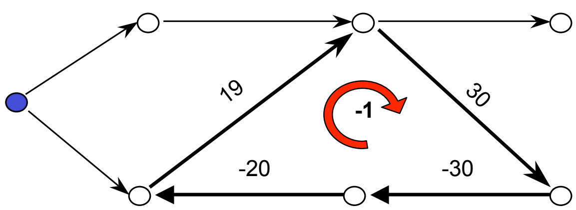
Inconsistency and negative cycles



Time-map

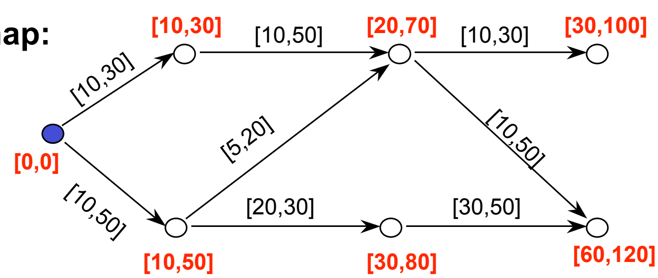
The new constraints [10,19] makes the time-map inconsistent, induces a negative cycle of length -1

$G_d(V,E)$



Shortest Path Trees (SPT)

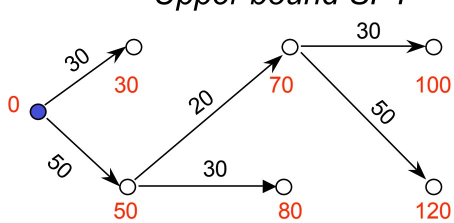
Time-map:



Each tp_i has a domain $[lb_i, ub_i]$

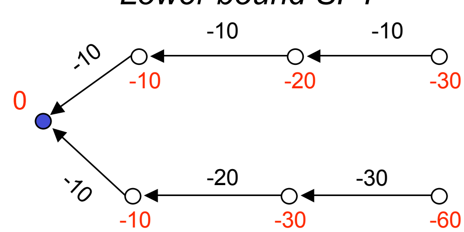
$G_d(V,E)$:

Upper bound SPT



$\boxed{ub_i = d_{0i}}$

Lower bound SPT

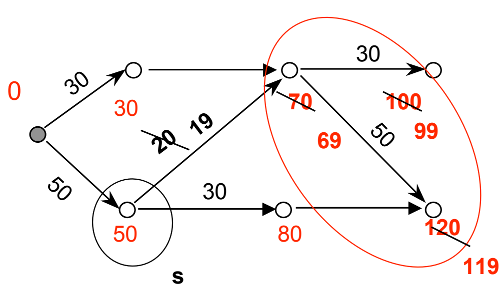


$\boxed{lb_i = - d_{i0}}$

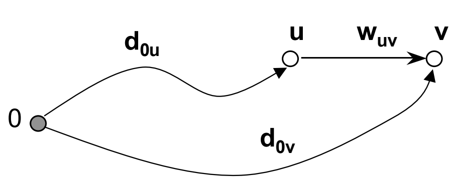
About the Simple Temporal Problem

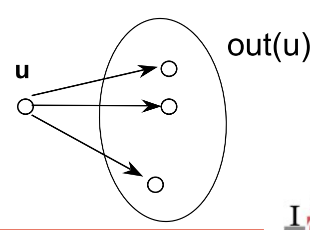
- The simple temporal problem is a constraint component of many scheduling and temporal reasoning problems
- **Shortest paths** [Pallottino 84] and **negative cycles** [Cherkassky&Goldberg 96] analysis are sources of information for driving the search for solving scheduling problems
- Typical procedures for managing time-maps include:
  - Computation from scratch of the bounds $[lb_i, ub_i]$ - include **consistency checking**
  - **Incremental** insertion and removal of temporal constraints improve computation efficiency
  - Computation of the **minimal network** - the tightest set of constraints $a_{ij} \leq tp_j - tp_i \leq b_{ij}$ which holds the same set of solutions

Incremental insertion of constraints

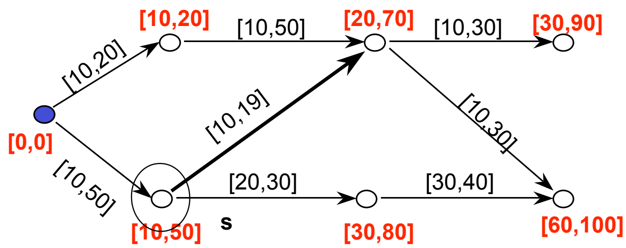


Q = {s}
while (Q <> nil) **{**
  u = Dequeue(Q)
  foreach v $\in$ out(u)
    if ($d_{0v} > d_{0u} + w_{uv}$) {
      $d_{0v} = d_{0u} + w_{uv}$
      $p_v = u$
      if ($d_{0v} + d_{v0} < 0$) exit(failure!)
    }
}

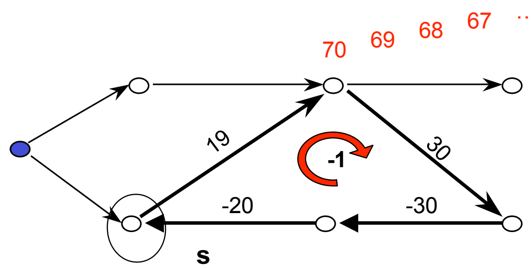




Improving consistency checking



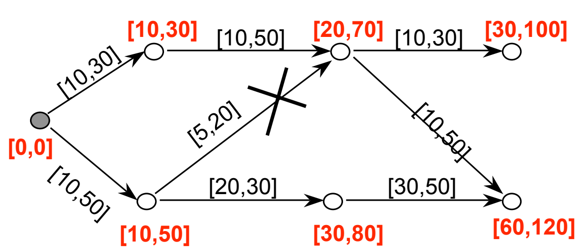
if ($lb_i > ub_i$) exit(failure!)

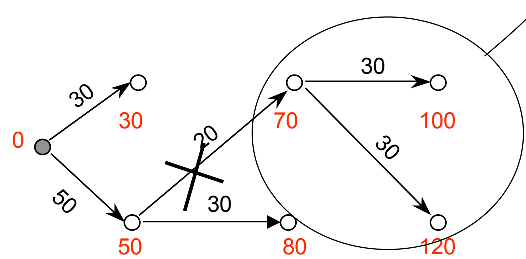


if ($d_{0v} + d_{v0} < 0$) exit(failure!)

If the node *s* is inserted twice in *Q* then the STP is inconsistent

Incremental removal of constraints





Upper bound SPT

Q = {.}
while (Q <> nil) **{**
  u = Dequeue(Q)
  foreach v $\in$ out(u)
    if ($d_{0v} > d_{0u} + w_{uv}$) {
      $d_{0v} = d_{0u} + w_{uv}$
      $p_v = u$
      ~~if ($d_{0v} + d_{v0} < 0$) exit(failure!)~~
    }
}

Minimal Network

- For each tp_i the bounds $[lb_i, ub_i]$ can be calculated with *Single Source Shortest Path* (SSSP) algorithms (e.g., Bellman-Ford [Cormen&al 90])
- A stronger way to deal with STP problems is to consider *All-Pairs Shortest Paths* algorithms (APSP) [Cormen&al 90] in order to calculate the set of all shortest path distances $\boldsymbol{d_{ij}}$ between each pair of time points $(tp_i\ tp_j)$
- Given an STP, if any constraints $a \leq tp_j - tp_i \leq b$ is replaced with the constraints $\boldsymbol{-d_{ji} \leq tp_j - tp_i \leq d_{ij}}$, the latter set of constraints is called ***minimal network*** and represents an equivalent STP, that is a problem with the same set of solutions of the first one.
- There are at least two advantages in using APSP Vs. SSSP algorithms:
  - Performing incremental **consistency checking** in O(1):
    if $(d_{ij} + w_{ij} < 0)$ *exit(fail!)*
  - Discovering in *O(1)* the mutual temporal position of each pair of time points (tp_i, tp_j)

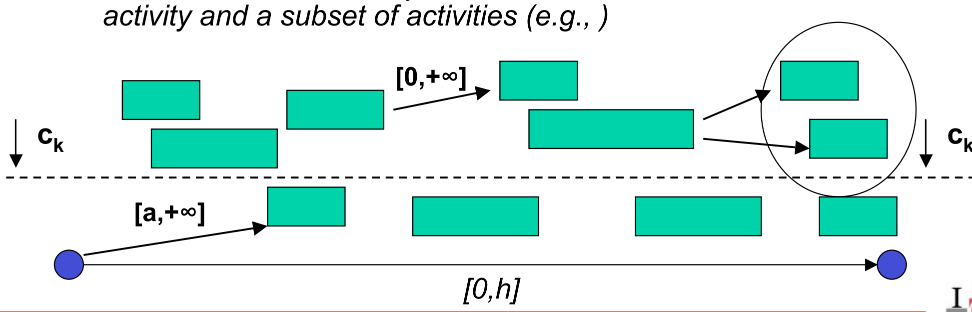
SSSP Vs. APSP algorithms

- Scratch propagation
  - *O(n.e)* Bellman-Ford [Cormen&al 90]
  - *O(n.e)* [Goldberg&Radzic 93]
- Incremental insertion
  - *O(n.e)* [Cesta&Oddi 96]
  - O(min(m,k.Δ).log(n)) [Frigioni&al 03]
- Incremental removal
  - *O(n.e)* [Cesta&Oddi 96]
  - *O(e + n.logn)* [Oddi 97]
  - [Frigioni&al 03]
- Memory: $O(n+e)$

- Scratch propagation
  - $O(n.e + n^2 log(n))$ Johnson [Cormen&al 90]
- Incremental insertion
  - $O(n^2)$ [Ausiello&al 91, Cesta&Oddi 01]
  - [Demetrescu&al 04]
- Incremental removal
  - $O(n.e + n^2 log(n))$ [Cesta&Oddi 01]
  - [Demetrescu&al 04]
- Memory: $O(n^2)$

Propagation: resource constraints

- Resource constraints are *n-ary constraints* imposed on a subset of activities
- Under the hypothesis that we consider a reference event *O*, all the synthesized (explicit) constraints can be represented with the form $a \leq tp_j - tp_i \leq b$, *in particular:*
  - Absolute constraints have the form $a \leq tp_i - O \leq b$
  - *Relative constraints* has the form $a \leq tp_j - tp_i \leq b$
  - *Set of constraints* $\{a \leq tp_j - tp_i \leq b\}$ *represents the relation between an activity and a subset of activities (e.g.,)*



Propagation: resource reasoning

- *Absolute time positions algorithms (ATPA)* - considers the absolute position of the activities can infer time bounds or precedence relations, among the literature proposals:
  - **Time-tabling** [Le Pape 94]: new time-bounds, $O(n^2)$
  - **Disjunctive constraints** (unary resource) [Ershler 76]: new precedence $O(n^2)$
  - **Edge-finding** [Carlier&Pinson 90, Nuijten 94, Baptiste&Le Pape 96]: new time-bounds, $O(n^2)$ and new precedence $O(n^3)$
  - **Energetic reasoning** [Ershler&al 91]: new time-bounds, new precedence $O(n^3)$
- *Relative time positions algorithms (RTPA)* - consider both the absolute positions and the precedence relations among the activities, among the literature proposals:
  - **Energy precedence propagation** [Laborie 03]: new time-bounds, $O(n^2)$
  - **Reservoir Balance Propagation** [Laborie 03]: new time-bounds, $O(n^2)$; new precedence $O(n^3)$

Outline

- Introduction
- Basic principles
- Constructive methods
  - Premise
  - Precedence constraint posting algorithms
  - Start-time based algorithms
  - POS schedules
- Meta-heuristics
- Conclusions

Constructive methods: premise

- Within a meta-heuristic search schema a constructive method is needed to generate a *first input solution* or used as *component procedure*
- In both cases the complexity of the constructive methods must be **polynomial**. A solution can be found without retraction of previous decisions or by performing a limited amount of search to revise previous decisions. In the latter case the computational effort have to be a *constant factor* times the effort spent for finding a first solution or a first dead-end
- As a consequence, when finding a feasible solution to a scheduling problem is not a polynomial task, the constructive method have to perform some constraint violations on the input problem and in general the solution is only partial feasible

Constructive methods

- A solution can be found through two modalities:
  - A *single path* of decisions, such that at each step there are two possibilities: relax or not some constraints in order to take the next decision
  - Apply a search strategy to revise the previous decisions before to go to the first modality. Mixed modalities are also possible.
- In principle to revise the previous decisions we can apply several possible **search strategies** like:
  - Depth First Search (DFS)
  - Best First Search (BFS)
  - Limited Discrepancy Search (LDS) [Harvey&Ginsberg 95]
- Within the previous schemas we can adopt different **branching schemas** like:
  - start(A) = est(A) **or** start(A) > est(A)
  - A *before* B **or** B *before* A

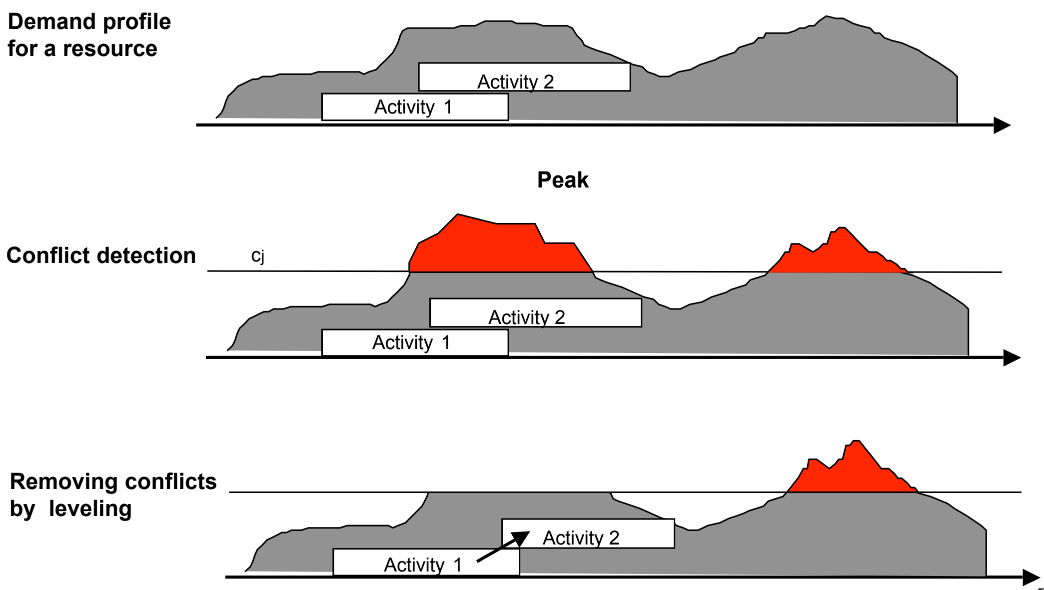
Constructive methods

- We describe four different algorithms to generate solutions:
  - A precedence posting algorithm based on the so-called profile based approach [Cesta&al 01]
  - A precedence posting algorithm to generate relaxed solutions [Oddi&Cesta 97]
  - *Start-time based* algorithms, inspired to the one presented in [Le Pape 94]
  - A polynomial algorithm to convert a *start-time* based solution into precedence based one (*Chaining*) [Policella&al 04]
- Modified versions of these algorithms are also used as component of the meta-heuristic strategies described in the following sections

ESTA Algorithm

- A constraint-based algorithm for solving RCPSP/max problem:
  - temporal constraints in a STP constraint network
  - resource capacity constraints
- Uses early start time resource profiles to detect conflicts in resource usage
- Remove conflicts by posting precedence constraint between pair of activities (a_j, a_j)

The profile-based approach



A greedy algorithm

ESTA(problem, maxHorizon)
Create-CSP-representation
Post(maxHorizon)
loop
    Propagate(CSP)
    Compute-conflicts(CSP)
    if *no-conflict* **then** Return(*solution*)
        else if *unresolvable-conflicts* **then** Return(*fail*)
            else
                Select-conflict
                Select-precedence
                Post(precedence)
end-loop

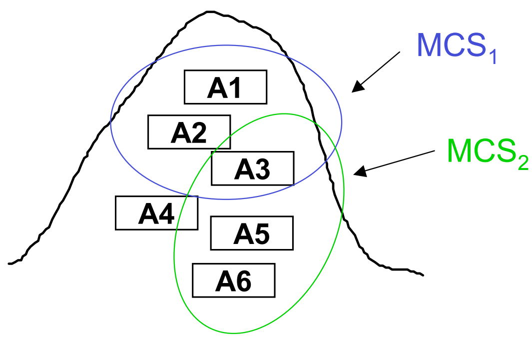
Compute Conflicts

1. *Analyze the Earliest Start Solution*
2. *Compute Conflicts (Peaks):* there is a conflict peak on resource r_k at time t if the resource requirement of the activities scheduled in t exceeds the resource capacity c_k of r_k
3. *Compute MCSs on Peaks*: a Minimal Critical Set (MCS) is a conflict such that each of its proper subsets is not a conflict

Minimal Critical Set (MCS) analysis

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

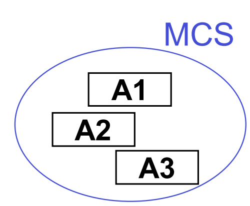
Resource capacity = 2



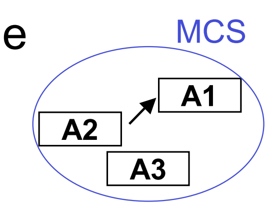
(Approximate computation of MCS *[authors, IJCAI-99]*)

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*]



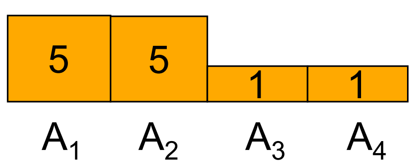
Texture-based heuristics

- The previous proposal represent one possible choice. Other heuristic methods might be defined, a guiding idea might be the so-called *texture-based heuristics* [Beck&Fox 00]
- A *texture measurement* is an analysis of the search state to reveal problem structure
- An heuristic is based on the revealed structure in order to find the most critical part of the current solution and take a decision
- The texture measurement and the heuristic to take decisions are separate

Why MCSs?

Capacity = 8

Peak



Pairs of activities:
$\{A_1\ A_2\}$ $\{A_1\ A_3\}$ $\{A_1\ A_4\}$ $\{A_2\ A_3\}$ $\{A_2\ A_4\}$ $\{A_3\ A_4\}$

MCSs:
$\{A_1\ A_2\}$

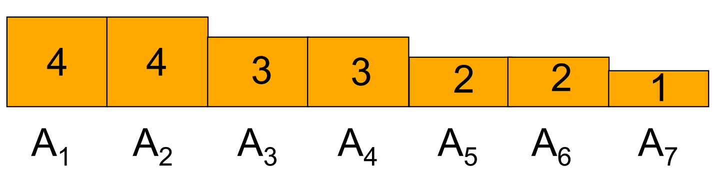
Approximate Computation of MCSs

- The number of minimal critical sets is exponential in the general case
- Proposal: sampling them with an approximate analysis on peaks
  - Linear sampling
  - Quadratic sampling
  - Bounded lexicographic sampling

Linear and quadratic sampling

Resource capacity = 6

Peak



MCSs

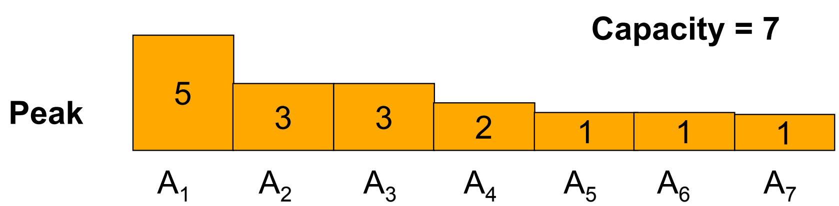
Linear Sampling:

$\{A_1\ A_2\}$
$\{A_2\ A_3\}$
$\{A_3\ A_4\ A_5\}$
$\{A_4\ A_5\ A_6\}$

Quadratic Sampling:

$\{A_1\ A_2\}\ \{A_1\ A_3\}\ \{A_1\ A_4\}$
$\{A_2\ A_3\}\ \{A_2\ A_4\}$
$\{A_3\ A_4\ A_5\}\ \{A_3\ A_4\ A_6\}\{A_3\ A_4\ A_7\}$
$\{A_4\ A_5\ A_6\}$

Bounded lexicographic sampling



Lexicographic order induced by the total order imposed on the peak activities *$(A_i[ru_i] < A_j[ru_j] \Leftrightarrow ru_i \geq ru_j)$*:

$$\left.\begin{array}{l}\{A_1[5], A_2[3]\}\\ \{A_1[5], A_3[3]\}\\ \{A_1[5], A_4[2], A_5[1]\}\end{array}\right\}\ \textbf{Selected MCSs}$$

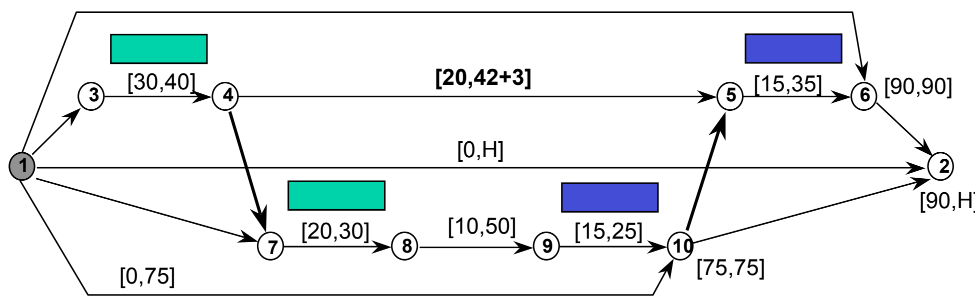
$\{A_1[5], A_4[2], A_6[1]\}$
$\{A_1[5], A_4[2], A_7[1]\}$
$\{A_2[3], A_3[3], A_4[2]\}$

Finding a relaxed solution

- The ESTA algorithm can be also used to find a *relaxed solution* when a *dead-end* is found during the search
- An *unsolvable* MCS represents a dead-end, that is a MCS where does not exist the possibility to post any precedence constraint to solve the conflict
- The introduced sampling techniques can be used to analyze a subset of unsolvable MCSs and choose one where it possible to post a precedence constraint with the minimal amount of violation of the temporal constraints
- When a new precedence is posted, constraints can be violated by *canceling induced negative cycles* [Cherkassky&Goldberg 96, Oddi&Cesta 97]
- Let see an example on a very small problem …

Heuristic Resolution by Relaxation

- Insertion of the ordering constraints *(10,5)* creates two negative cycles on G_d with length *-3*
- These cycles can be canceled by relaxing of *3* time units the constraints *(1,6)* (deadline) and *(4,5)* (separation constraint).



A start-time based algorithm

- A solution S = ($\langle a_1, est_1\rangle, \langle a_2, est_2\rangle, \ldots, \langle a_n, est_n\rangle$)
- Iteratively selects activities on the basis of the following priority rules (value ordering): selects an activity which has the minimal feasible est_i with regard to the current resource load profiles (uses the values lft_i to break ties)
- Uses an SSSP-based algorithm to update the variables est_i, eft_i, lst_i, and lft_i each time a new activity is added to the current schedule *S*
- The algorithm *branches* with the rule:
  – $s(a_i) = est_i$ **or** $s(a_i) > est_i$

SetStartTimes algorithm

```
SetStartTimes(U,F_S,S)
    U : unselected activities {a_i1 , a_i2 ,… a_ik }
    F_S : failed choices ⟨a_i ,est_i⟩ on the current (partial) solution S
    S : current solution (⟨a_1 ,est_1⟩, ⟨a_2 ,est_2⟩, …, ⟨a_n ,est_n⟩)
  if Propagate(U,S)
    then if (U =∅)
            then Return(S)
            else if aSelectableActivityExists(U,F_S)
                    then a_i ← SelectAcyivity(U,F_S)
                         S ← S ∪ {⟨a_i ,est_i⟩}
                         SetStartTimes(U - {a_i},F_S,S)
                         Pop(S)
                         F_S ← F_S ∪ {⟨a_i ,est_i⟩}
                         SetStartTimes(U,F_S,S)
```

- *Propagate(U)*: updates the variables est_i, eft_i, lst_i, and lft_i. Returns T when the current solution is time feasible, F otherwise
- *aSelectableActivityExists(U,F_S)*: returns T when $\exists a_i$: $a_i \in U$ and $\langle a_i ,est_i\rangle \notin F_S$
- *SelectAcyivity(U, F_S)*: selects an activity $a_i \in U$ which has the minimal feasible est_i with regard to the current resource load profiles. Uses the values lft_i to break ties
- The algorithm starts with SetStartTimes(A, ∅, ∅)

An algorithm to generate POSs

- We describe an algorithm which transforms a *fixed-times* schedule into a Partially Ordered Schedule
- We remember that within a POS, each activity retains a set of feasible start time and each time feasible schedule is also a feasible solution
- The algorithm is described in [Policella&al 04] as a way to provide a basis for responding to unexpected disruptions in a schedule and to improve its *robustness* (it can be seen as a way to enforce *backtracking free* solutions)
- In the following POS schedules are used as a standard way to represent solutions within different Local Search procedures

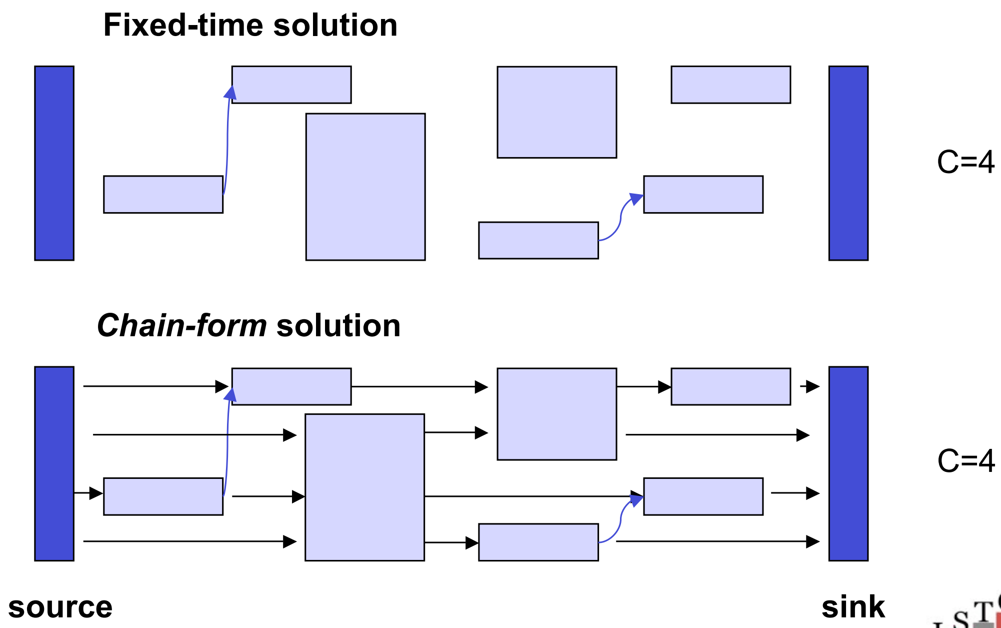
An algorithm to generate POSs

- Within a POS, 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

Chaining algorithm

Chaining (*S*)
    in: a fixed-time schedule *S*
    out: a partial order schedule *POS*
  for all resources r_k **do**
        Sort the activities A_k *by start-times*
        while $A_k \neq \emptyset$ **do**
            $c \leftarrow$ Select-Consumer(A_k)
            while (*c needs more resource units*) **do**
                $p \leftarrow$ Select-Producer(c)
                $POS \leftarrow POS \cup \{(p,c)\}$
  Return(*POS*)

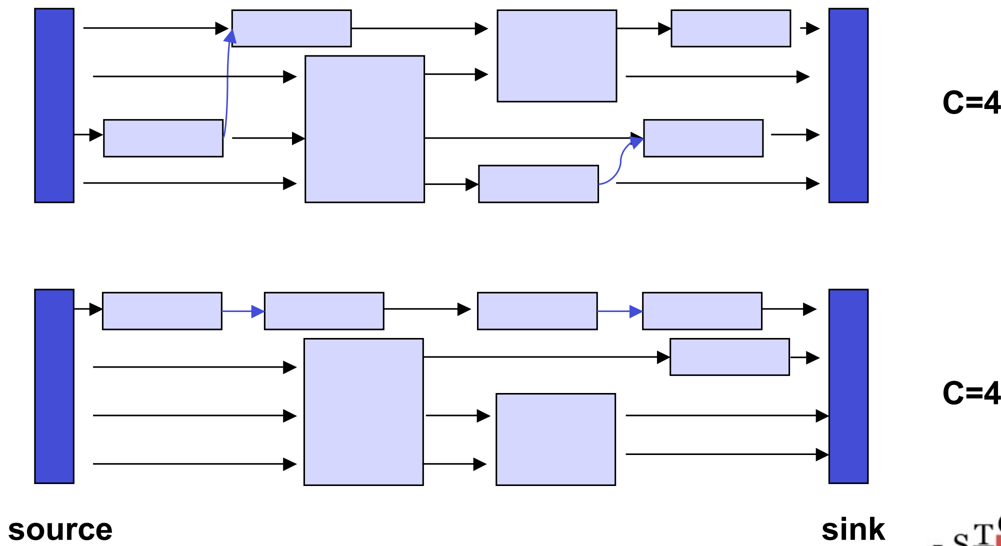
Chaining algorithm



Properties of the *chain-form*

- The resulting partial order is a POS
- Since only simple precedence constraints already contained in the input solution are added, the makespan on the output solution will not be greater than the original one
- Given a fixed-time schedule, generally it has a set of *chain-form* solutions

Two different chain-forms



Outline

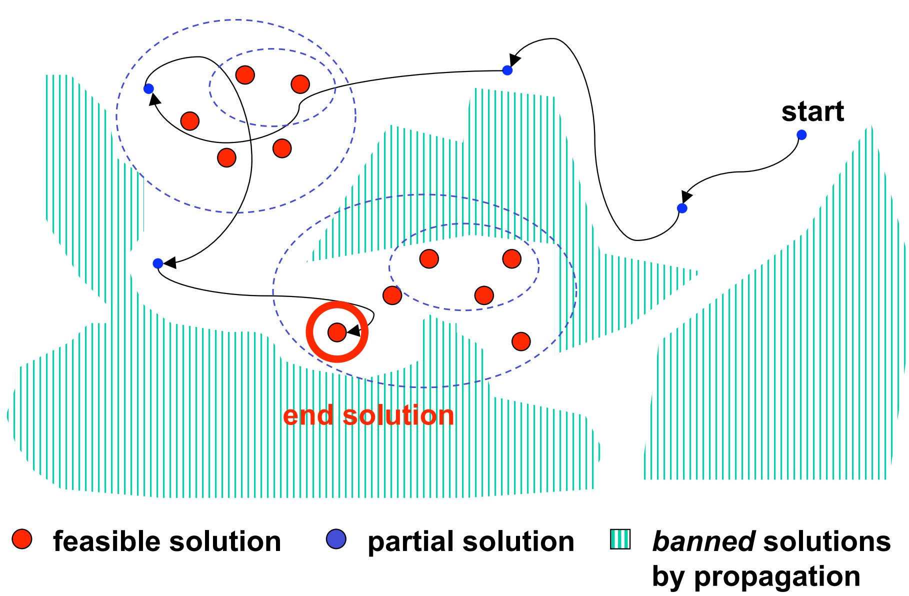
- Introduction
- Basic Principles
- Meta-heuristics
  - Premise
  - Iterative Random Sampling
  - Basic Local Search
    - Tabu Search
    - Iterative Flattening
  - Composite strategies
    - Back-Jump Tracking (BJT)
    - Greedy Randomized Adaptive Search (GRASP)
    - Variable Neighborhood Search (VNS)
    - Iterated Local Search (ILS)
- Conclusions

Guiding ideas

- A meta-heuristic schema combines basic *constructive* and *local search methods* within a *CSP* reference framework
- The definition of a *meta-heuristic* schema is driven by two key concepts:
  - The definition, representation and use of *control knowledge* to drive the search
  - The *balancing* between *intensification* and *diversification*:
    - *Intensification* means to search carefully and intensively around good solutions found in the past search
    - *Diversification*, on the contrary, means to guide the search to unvisited regions of the search space (a strategy for escaping from a local minima)

Guiding ideas

Search space: balancing between *intensification* and *diversification*



Iterative Random Sampling

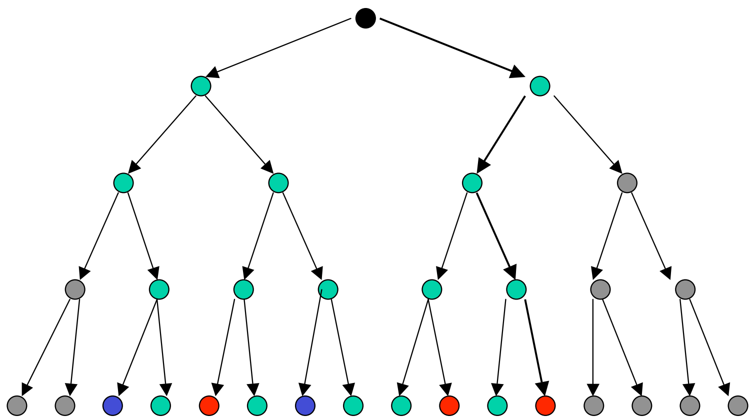
- Iterative Random Sampling is a meta-heuristic technique for solving combinatorial problems (both for optimization or constraint satisfaction)
- Given a CSP formulation of the problem, the algorithm iteratively generates solutions by a *greedy random sampling procedure*
- A greedy random sampling procedure iteratively applies the following steps:
  - Propagation
  - *Random* variable ordering
  - *Random* value ordering
- The algorithm stops either with the best solution found or with a failure

Iterative Random Sampling

IterativeRandomSampling(*CSP*)
  in: CSP = $\langle X, D, C\rangle$, *termination conditions*
  out: best solution *S**
    S* ← Ø
    while (a *termination condition not met*) **do**
      S ← GreedyRandomSampling(CSP)
      UpdateBestSolution(S,S*)

GreedyRandomSampling(*CSP***)**
  Propagate(*CSP*)
  if ($\exists D_i = \emptyset$) **then** Return(*failure!*)
    else if (*all decision variables* X_i *are set*) **then** Return(*solution*)
      else
        X_i ← RndSelectDecisionVariable(*CSP*)
        v ← RndSelectValue(D_i))
        set(v,X_i)
        Solve(*CSP*)

Iterative Random Sampling

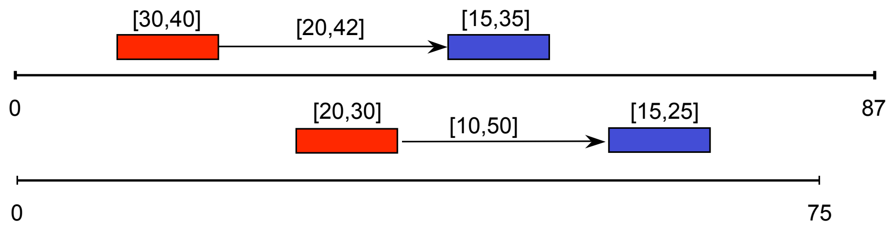


partial or feasible solution
banned by propagation
optimal solution
optimal solution

Iterative Random Sampling

- In the following we propose two examples:
  - An Iterative Random Sampling (IRS) procedure for solving Job Shop Scheduling Problems with maximal temporal constraints [Oddi&Smith 97]
  - IRS for RCPSP/max: an optimization problem [Cesta Oddi&Smith 02]
- Two emerging observations
  - Both the procedure use stochastic search as a means of efficiently solving scheduling problems with deadlines and complex metric constraints
  - A key idea underlying the described approach is to *heuristically bias random choices* in a dynamic fashion, according to how (or how poor) the available search heuristics discriminate among several alternatives

Job Shop Deadline Scheduling



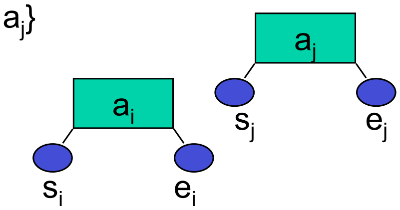
- Scheduling problem with two jobs, each job has two activities. “Red activities” require resource $\mathbf{r_1}$ and “blue activities” require resource $\mathbf{r_2}$
- Each activity requires only one resource and at any instant a resource can execute only one activity. The processing time is an interval of possible values
- Between a couple of successive activities there is an interval of temporal separation
- Each job has a ready time time and a deadline

JSDSP: problem definition

- A set of jobs $J=\{j_1...j_n\}$
- A set of resources $R=\{r_1...r_n\}$
- The execution of a job_i requires the processing of a sequence n_i activities.
- Each activity a_{ij} in a job_i can requests only one resource and a resource is requested only once in a job.
- Each activity a_{ij} has a time processing constraint:
  $lbp_{ij} \leq end(a_{ij})\text{-}start(a_{ij}) \leq ubp_{ij}$
- Between the set of activities $\{a_{i1}...a_{in}\}$ in a job_i are posted a set of separation constraints:
  $lbp_{ij} \leq start(a_{i(k+1)}) - end(a_{ik}) \leq ubp_{ij}$ $k=1..(n-1)$
- Each job_i has an associated ready time r_i and a deadline d_i .

JSDSP: propagation

Given a conflict (decision variables) (a_i, a_j),
two possible values: {a_j *before* a_i **,** a_i *before* a_j}



On the basis of the distance d(x,y) on the time-map, the following propagation rules are applicable:

1. unsolvable conflict: $d(e_i,s_j) < 0$ *and* $d(e_j,s_i) < 0$: *no values!*
2. solvable conflict:
   a. $d(s_i,e_j) >= 0$: {a_j *before* a_i}
   b. $d(e_j,s_i) >= 0$: {a_j *before* a_i}
   c. $d(e_i,s_j) >= 0$ **and** $d(e_j,s_i) >= 0$: {a_j *before* a_i **,** a_i *before* a_j}

Random Variable and Value Ordering

- $\alpha-\beta$ *heuristics* [Cheng&Smith 94, Oddi&Smith 97]
- Heuristic-Biased Stochastic Sampling (*HBSS)* [Bresina 96]

α-β value ordering

- This heuristic [Cheng&Smith 94] selects the conflict which can be resolved with the *“minimum temporal commitment”* on the current solution.
- It selects the conflict (a_i, a_j) with the minimun value of the function:

$$TC((a_i, a_j)) = min\{bd_{ij}, bd_{ji}\}$$

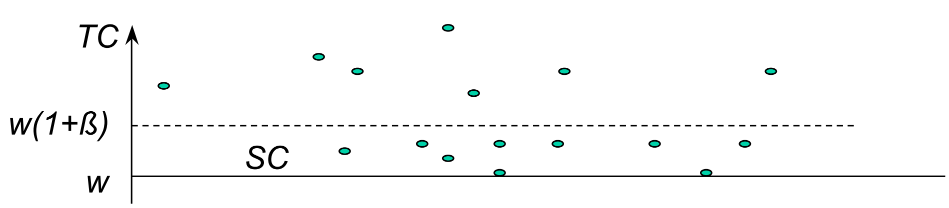
Where:

$$bd_{ij} = \frac{d(e_i,s_j)}{S^{0.5}} \quad bd_{ji} = \frac{d(e_j,s_i)}{S^{0.5}} \qquad S = \frac{min\{d(e_i,s_j),d(e_j,s_i)\}}{max\{d(e_i,s_j),d(e_j,s_i)\}}$$

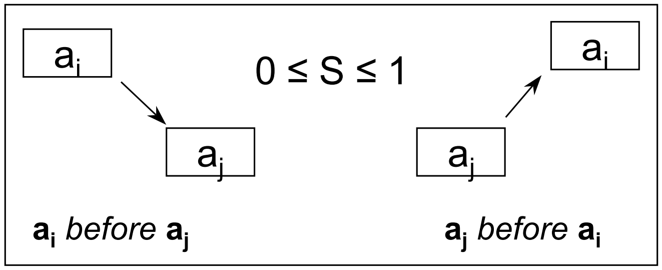
α-β variable ordering

Random Variable ordering:

1. $w = min\{TC((a_i, a_j))\}$
2. $SC=\{(a_i, a_j): w\ TC((a_i, a_j))\ w(1+ß)\}$
3. *Random select a conflict* (a_i, a_j) *in the set SC*

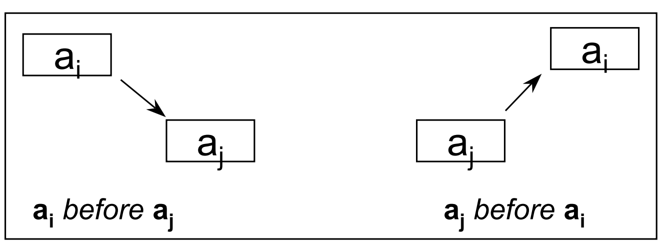


α-β variable ordering



S measures the "*similarity*" in the commitment of the two possible ways to solve a conflict **(a_i, a_j)**
When **S=1** ($0 \leq S \leq 1$) both the ordering choices make the same temporal commitment on the current solution

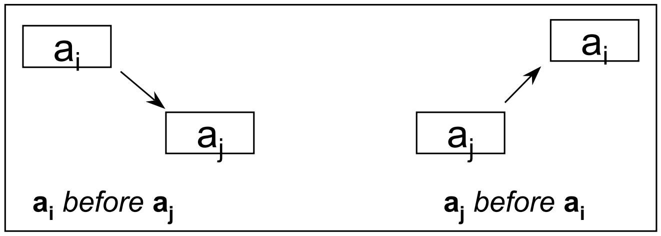
α-β variable ordering



Value Ordering: Following the same criterion [Cheng&Smith 94] of leaving the maximum degree of temporal flexibility, the selection of a precedence constraint is done as follows:

- (a_j *before* a_i), **when $bd_{ij} > bd_{ji}$**
- (a_i *before* a_j), **otherwise**

α-β value ordering



Random value ordering on *(a_i, a_j):*

- *deterministic* choice $U[0,1] + \alpha \geq S$;
- the opposite choice **otherwise**

α $(0 \leq \alpha \leq 1)$ is a *threshold* to avoid to exchange precedence constraint choices in conflicts with small values of similarity *S*

HBSS variable and value ordering

Variable ordering:

- Let *C* is the set of resolvable conflicts
- Sort the set C according to the *TU* values and assign and index *r=1,2,3...*(rank) to its elements, so that the conflict with minimum value has rank *r =1*
- Randomly select an element in *C*, where the probability to get a choice with rank *r* is: $P(r)=Fb(r) / (Fb(1) + Fb(2)...+ Fb(c))$
(*Fb* is one of the following bias functions: $1/r^2$, $1/r^3$ or $exp(-r)$)

Value ordering:

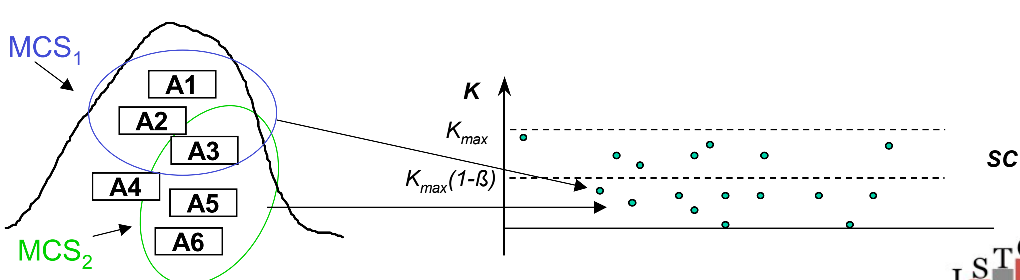
- Assign the rank *r=1* to the decision proposed by the "deterministic heuristic" proposed in Cheng & Smith AAAI-94 and the value *r=2* to the complementary decision
- Randomly select one ot the two possible choices, where the probability to get a choice with rank *r* is:
  $P(r)=Fb(r) / (Fb(1) + Fb(2))$

IRS for RCPSP/max

- *RCPSP/max:* a complex optimization problem
- ISES an iterative sampling framework
  - an heuristically guided greedy algorithm to solve
  - approximate computation of MCSs to avoid computational burden
  - use of randomization, linked to heuristic bias
- The ISES algorithm
  - Basic randomized greedy strategy
  - Meta-heuristic schema for optimization
- Experimental Evaluation

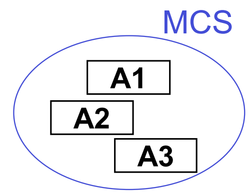
Randomized basic search procedure

- To expand the search and take advantage of multiple executions of the basic greedy algorithm: we insert a random choice in the MCSs selection
- After ordering MCSs according to estimator K, we consider values in the interval $K_{max}\ (1-\beta) \leq K(MCS) \leq K_{max}$ as equivalent. Where $\beta \in [0,1]$ is an *acceptance* band
- In this way the resolution procedure is transformed into a random search process *biased with heuristic information*

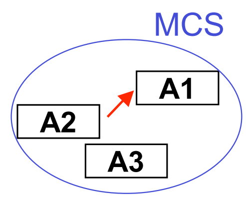


Random MCS elimination

- Variable ordering: which MCS to resolve first
  - Use estimator K [*Laborie&Ghallab 95*] to order MCSs
  - Randomly select an MCS within the set SC(β)



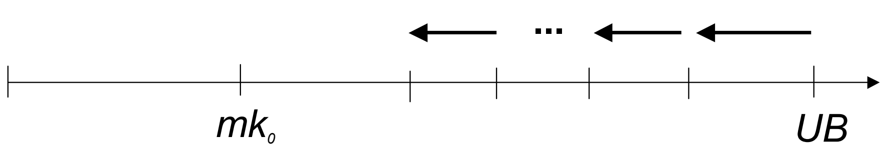
- Value ordering: how to choose the precedence (leveling) constraint
  - Use slack-based heuristics [*Smith&Cheng 93*]



The ISES algorithm

Multi-Pass ISES Procedure (*NbrRestarts*, *UpperBound*)

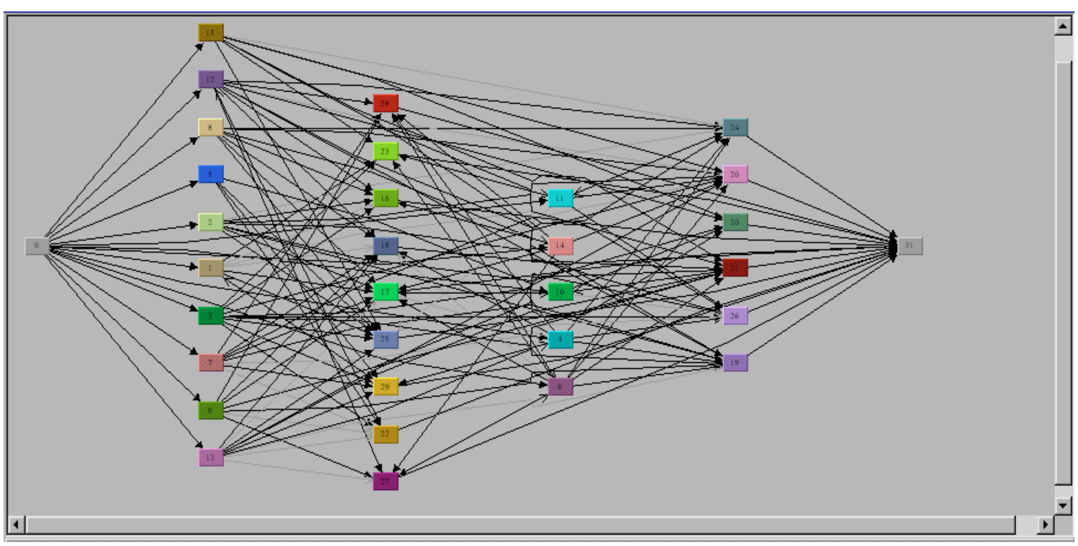
if (a solution with *maxHorizon* = *UpperBound* exists) **then**
repeat
  Apply randomized ESTA *NbrRestarts* times
  Set *maxHorizon* to the best makespan found
until (the makespan is improved)
Return(*best solution*)



Experimental evaluation

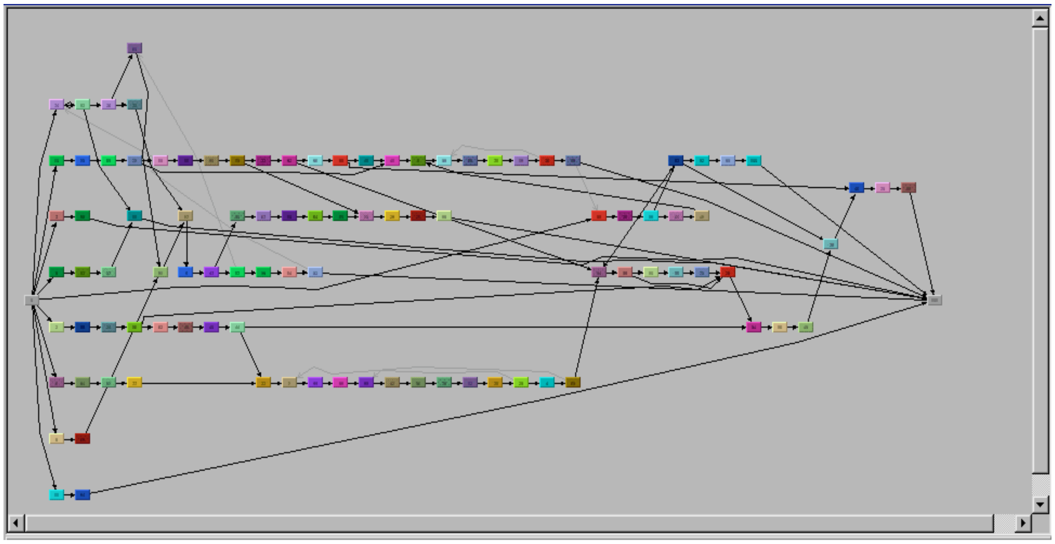
- From the OR repository of *RCPSP/max* problems we have selected two sets of problems.
  - **Problem set A**. [Kolisch&al. 98]. 3 sets of *270* problems each, named *J10*, *J20* and *J30*, with problems of *10*, *20* and *30* activities respectively
  - **Problem set B**. [Schwindt 98]. It consists of *1080* problems of *100* activities
- In all experiments,
  - *NbrRestarts* = *10*.
  - *Acceptance Band* β = *0.1* (i.e., *MCSs* within *10%* of highest ranked considered as equivalent)
  - *UpperBound* = $5\ mk_0$

Set A (a sample network)



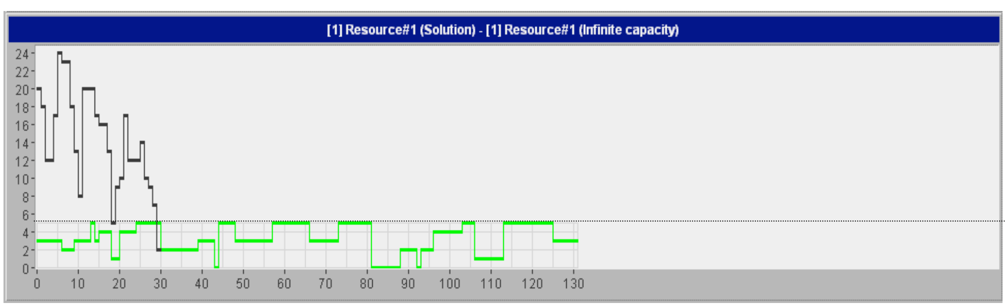
J30

Set B (a sample network)



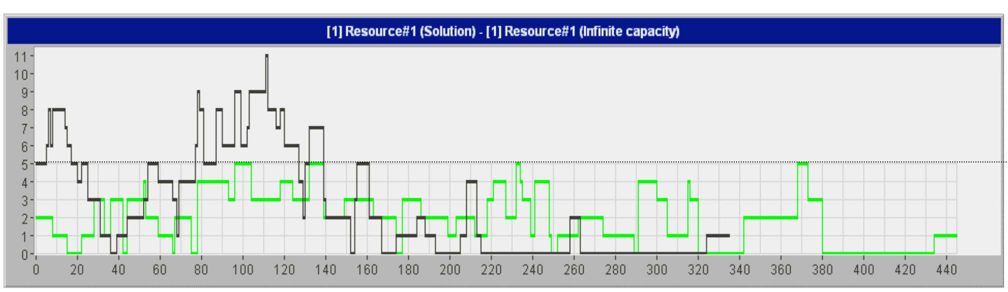
J100

Comparing resource profiles



Set A (J30)

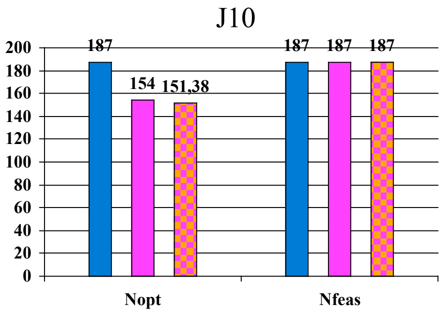
C=5

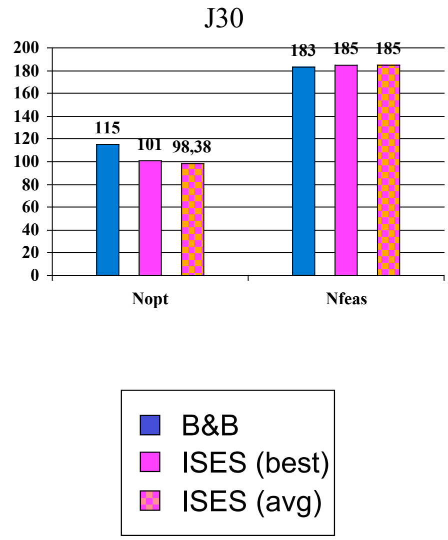


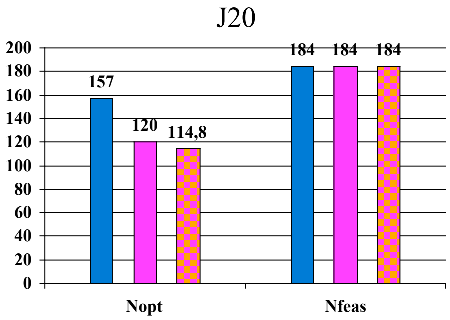
Set B (J100)

C=5

Experimental Results on Set A



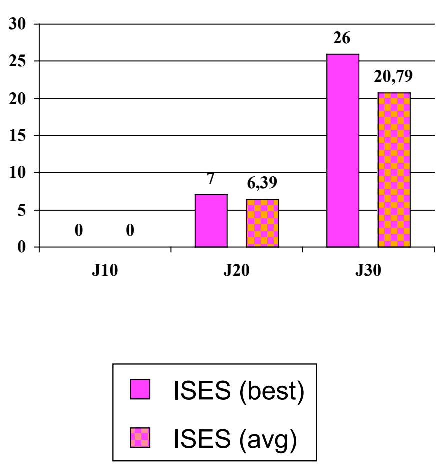




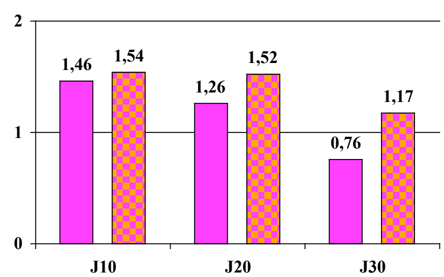
(CPU bound 100 seconds)
(ISES with quadratic sampling)

Experimental Results on Set A (2)

Number of improved solutions (wrt B&B)



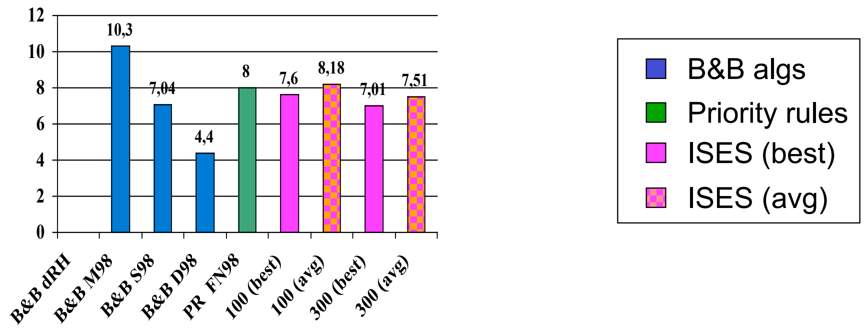
Average and best deviation (wrt B&B)



(CPU bound 100 seconds)
(ISES with quadratic sampling)

Experimental Results on Set B

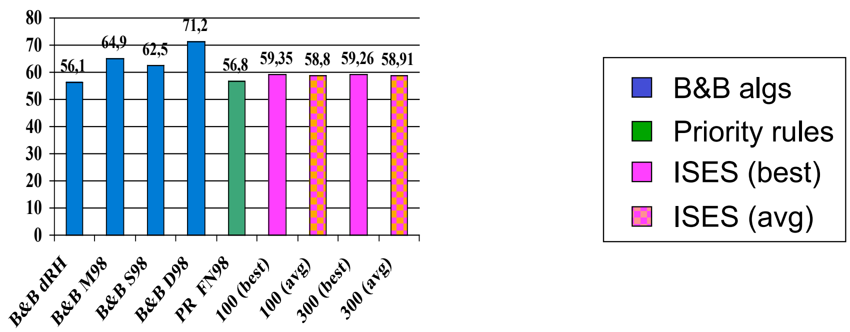
Lower bound percentage deviation



(ISES with linear sampling)

Experimental Results on Set B (2)

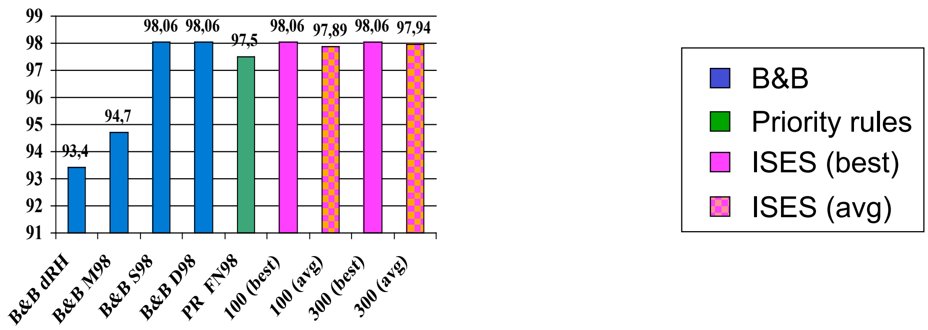
Percentage of optimal solutions



(ISES with linear sampling)

Experimental Results on Set B (3)

Percentage of feasible solutions



(ISES with linear sampling)

Overview

- Introduction
- Basic Principles
- Meta-heuristics
  - Premise
  - Iterative Random Sampling
  - Basic Local Search
    - Tabu Search
    - Iterative Flattening
  - Composite strategies
    - Back-Jump Tracking (BJT)
    - Greedy Randomized Adaptive Search (GRASP)
    - Variable Neighborhood Search (VNS)
    - Iterated Local Search (ILS)
- Conclusions

Tabu Search: introduction

- Tabu search is a meta-heuristic approach to find a *near-optimal* solution $s^* \in S$ of combinatorial optimization problems
- It needs a fundamental notion called the *move*. A move is a function which transforms a solution into another $m: S \to S$
- A move m induces the so-called *neighborhood structure N* of a solution s, which is a function $N: S \to 2^S$. $N(s)$ is the *neighborhood* of s
- Tabu search starts from an initial solution s_0, and at each step i the neighborhood $N(s_i)$ of a given solution is searched in order to find a neighbor s_{i+1} with the best value $f_{obj}(s_{i+1})$
- In order to prevent cycling, it is not allowed to turn back to the previous visited *MaxSt* solutions. Where *MaxSt* is the max length of the so-called *tabu-list* which is a queue with limited length

Tabu Search template

Tabu Search(*CSP*)
  in: s_0, *MaxSt, termination conditions*
  out: best solution *s**
    Tabu-list ← Ø
    while (*termination conditions not met*) **do**
        S ← ChooseBestOf (*N(s)* \ *Tabu-list*)
        UpdateBestSolution(*s,s**)
        Update(*Tabu-list*)

Tabu Search: improving efficiency

- Generally the implementation the short time memory mechanism as a *tabu-list* that contain the previous *MaxSt* complete solutions is not practical for efficiency reasons
- The most common solution is to store solutions attributes in the *Tabu-list*. Attributes are usually components of solutions, moves, or differences between two solutions
- The set of attributes and the corresponding tabu lists define the tabu conditions, which are used to filter the neighborhood of a solution and generate the allowed set of solutions

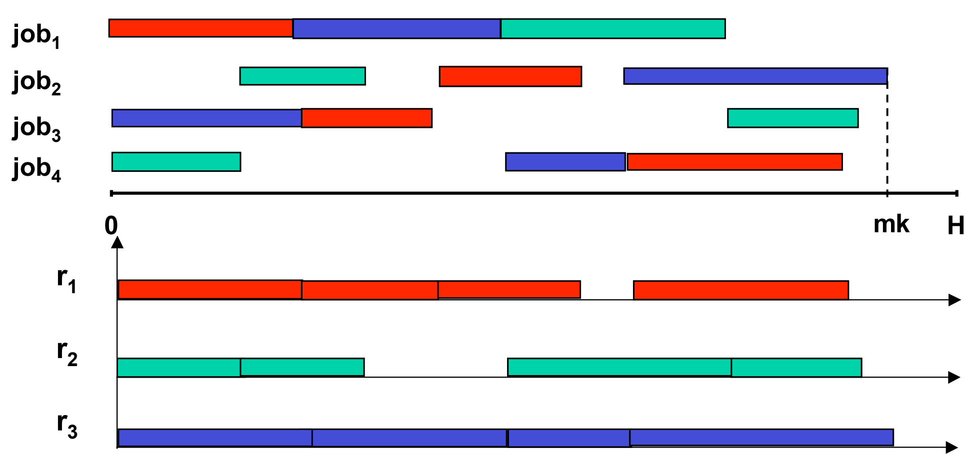
Tabu Search: improving efficiency

- Storing attributes instead of complete solutions is much more efficient, but it introduces a *loss of information*
- In fact, forbidding an attribute means assigning the tabu status to probably more than one solution. Thus, it is possible that unvisited solutions of good quality are excluded from the allowed set
- To overcome this problem, *aspiration criteria* are defined, which allow to include a solution in the allowed set even if it is forbidden by tabu conditions
- Aspiration criteria define the aspiration conditions that are used to construct the allowed set
- The most commonly used aspiration criterion selects solutions which are better than the current best one

Two different algorithms for scheduling

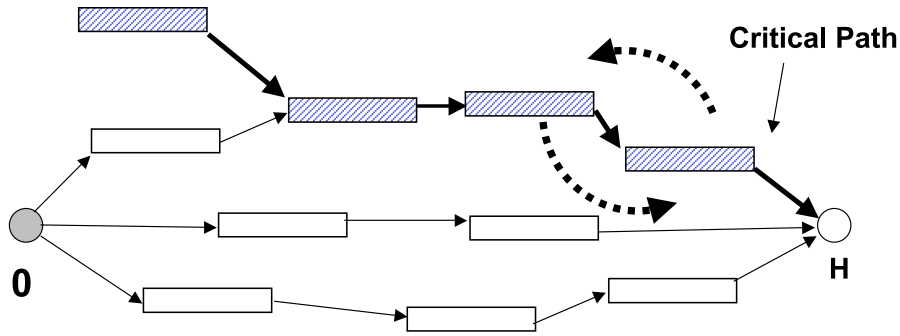
- One for solving a classical optimization problem: the Job Shop Scheduling Problem (JSSP)
- Another one for solving the Job Shop Scheduling Problem (JSSP) with relaxable metric constraints, an *NP-hard* satisfaction scheduling problems

The JSSP scheduling problem



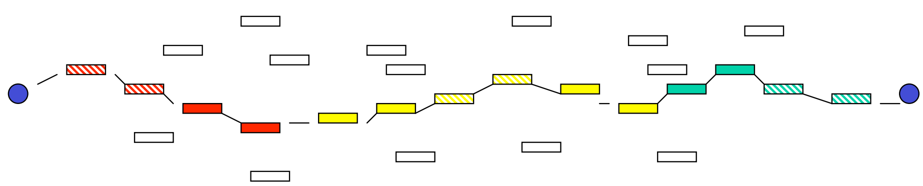
Scheduling problem with *four jobs*, each job has three activities; "red activities" require resource $\mathbf{r_1}$, "green activities" require resource $\mathbf{r_2}$, and "blue activities" require resource $\mathbf{r_3}$

Move



- A move is defined as the swap of two consecutive activities (a_i,a_j) on the critical path which require the same resource
- In addition, (a_i,a_j) is the last pair or the first pair of activities within the so-called critical path ***blocks*** [Grabowsky&al 86]

Critical path and *blocks*



- The set of activities on the critical path can be partitioned in a set of *nb* blocks $B_1, B_2, \ldots B_{nb}$
- Each block B_i is a sequence of activities on the critical path which require the same resource
- (a_i,a_j) is either
  - the last pair or the first pair of activities within the blocks $B_2, \ldots B_{nb-1}$, or
  - the first pair of B_{nb} or the last pair of B_1

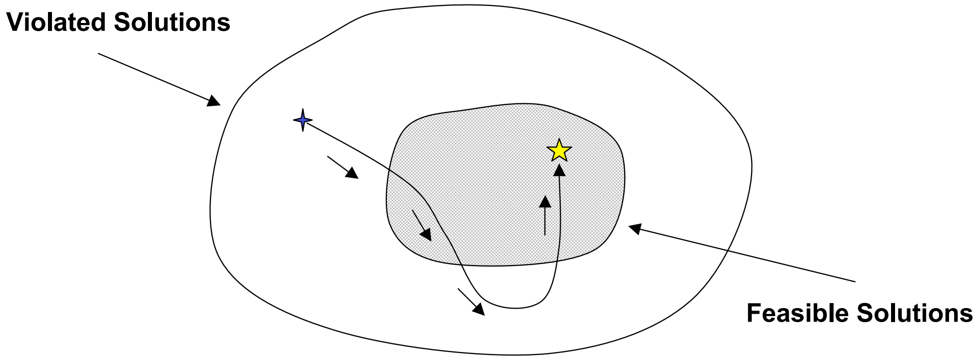
Main properties of *N(s)*

- Properties of N(s):
  - All the elements in the set N(s) are feasible solutions, that is are represented with dag graphs
  - For any processing order obtained by swapping the pair (a_i, a_j), which does not hold the previous block condition, the corresponding makespan is greater or equal to each makespan associated to the solutions in N(s)
  - If N(s) is empty, then s is an optimal processing order
- We observe that the main effect of using the additional condition of blocks is the reduction of the size of *N(s)* with great benefits in the computational efficiency

A problem with violable constraints

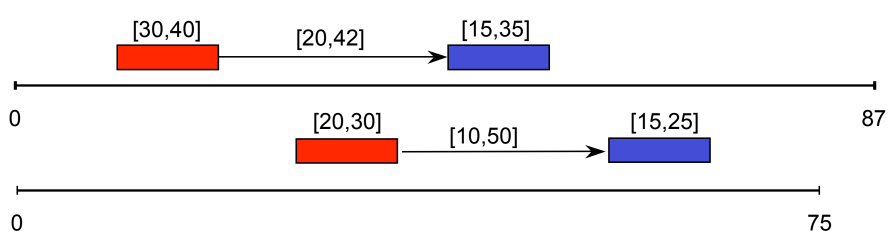
- A Relaxable Metric Scheduling Problem (**RMSP**) extends the classical Job-Shop Scheduling Problem with the use of complex temporal metric constraints and with the possibility of making the distinction between *relaxable* and *not- violable* constraints
- We briefly describe a *tabu-search* which uses the idea of relaxing some temporal constraints to *navigate* the search space and to find a solution where there are no violations or only the constraints classified as *relaxable* are violated
- Two motivation for RMSP
  - It may be the case that there is no solution which satisfies all the original constraints, but at the same time some constraints are relaxable and the only possible solution is to find an agreement on the operated violations
  - In many practical applications a scheduling problem can be defined where it is possible to relax some of the temporal constraints in order to find a solution, even if this is not strictly desirable

Finding a feasible solution



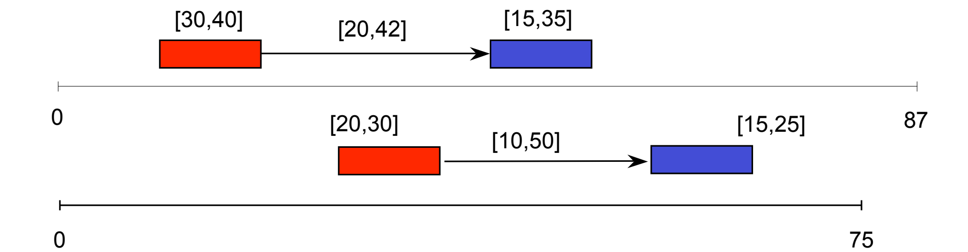
- **Violated Solution**: a solution where are relaxed *violable* and *not-violaable* constraints
- **Feasible Solution**: a solution where are relaxed only *violable* constraints

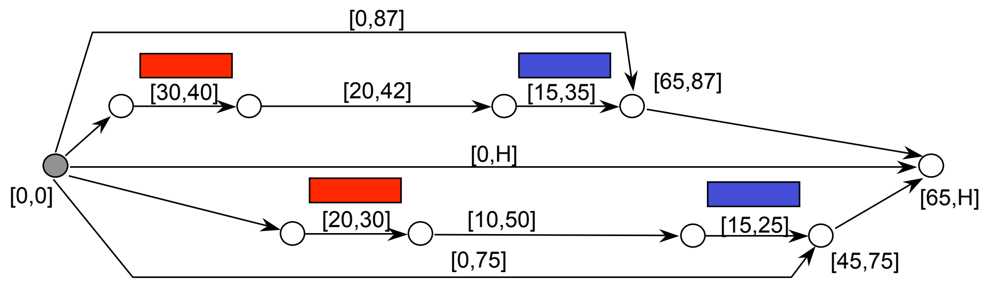
An example of scheduling problem



- Scheduling problem with two jobs, each job has two activities. "Red activities" require resource r_1 and "blue activities" require resource r_2
- Each activity requires only one resource and at any instant a resource can execute only one activity. The processing time is an interval of possible values
- Between a couple of successive activities there is an interval of temporal separation
- Each job has a ready time time and a deadline

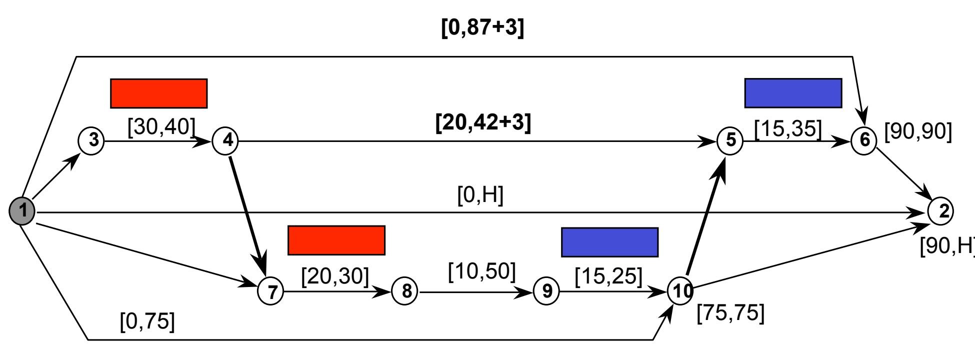
Time Map





Heuristic Resolution tion

- Insertion of the ordering constraints *(10,5)* creates two negative cycles on G_d with length -3.
- These cycles can be canceled by relaxing of 3 time units the constraints *(1,6)* (deadline) and *(4,5)* (separation constraint).



Tabu Search with STP: definition of move

- Tabu search "navigates" in the space of the violated solutions.
- The definition of **move** implies relaxation and restoration of temporal constraints.
- The objective function is a linear function where the components represent the sum of the relaxation values on specific types of constraint.
- Given a current solution, a move **m** on a resource **r** is defined as:
  - a couple of activities ($\mathbf{a_i,a_j}$) which have to be swapped;
  - a set of violations of the time constraints;
  - a set of possible restorations operated after the swap.
- Previous definition does not specify methods to violate and restore constraints, this is a matter of the specific heuristics adopted. Instead the previous definition is quite general to be applied with the STP temporal model.

Remarks on the definition of *move*

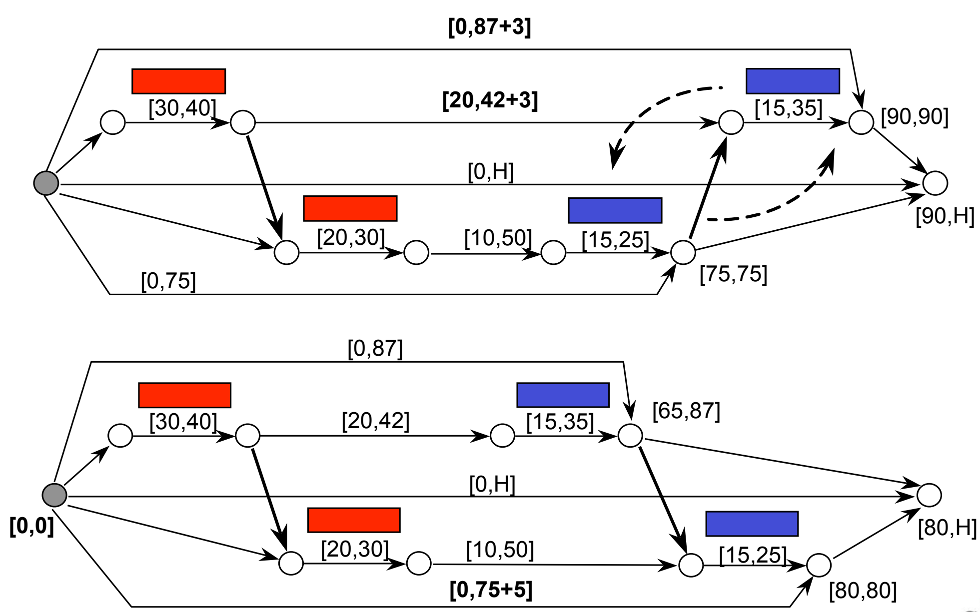
- A move **m** can create a circular chain of activities which induces a time inconsistency which can be resolved only by introducing negative durations.
- As it is proved in *[Oddi&Cesta 97]* to avoid the previous problem it is sufficient to choose moves which hold the following conditions:
  - ($\mathbf{a_i,a_j}$) are consecutive on the resource **r**;
  - ($\mathbf{a_i,a_j}$) are on a shortest path in the graph G_d.

The objective function

- We assume there are two types of constraints considered in the tabu search procedure: deadlines and separation constraints. Deadlines are relaxable, separation constraints are not relaxable. A solution is feasible if there is no violation on the separation constraints
- The **objective function** has two weighted components: the sum   of the relaxation on the deadline constraints (**RLX_{dl}**) and the sum on the separation constraints between activities (**RLX_{sp}**).
  With the values assigned to the values α_{dl} and α_{sp}, it is possible to focus the tabu search on a specific type of constraints

$$F_{obj}(S)= \alpha_{dl}\, RLX_{dl}(S) + \alpha_{sp}\, RLX_{sp}(S)$$

Example: feasible solution after the swap



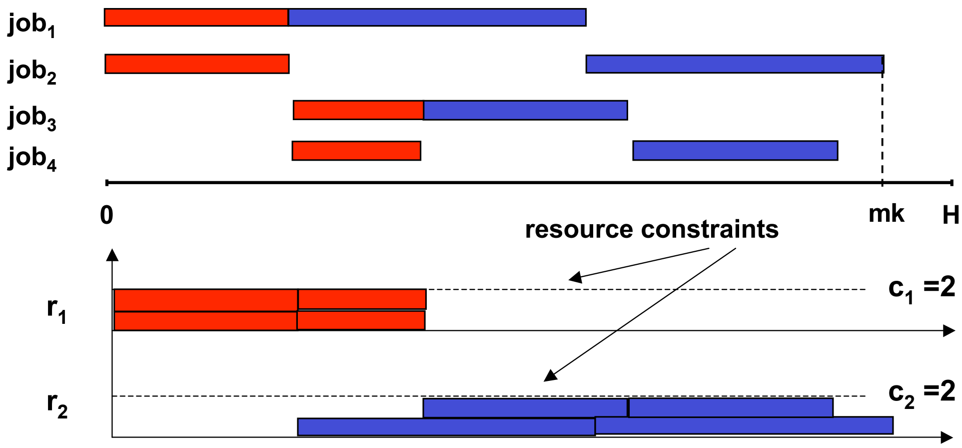
Outline

- Introduction
- Basic Principles
- Meta-heuristics
  - Premise
  - Iterative Random Sampling
  - Basic Local Search
    - Taboo Search
    - Iterative Flattening
  - Composite strategies
    - Back-Jump Tracking (BJT)
    - Greedy Randomized Adaptive Search (GRASP)
    - Variable Neighborhood Search (VNS
    - Iterated Local Search (ILS)
- Conclusions

Iterative Flattening Search

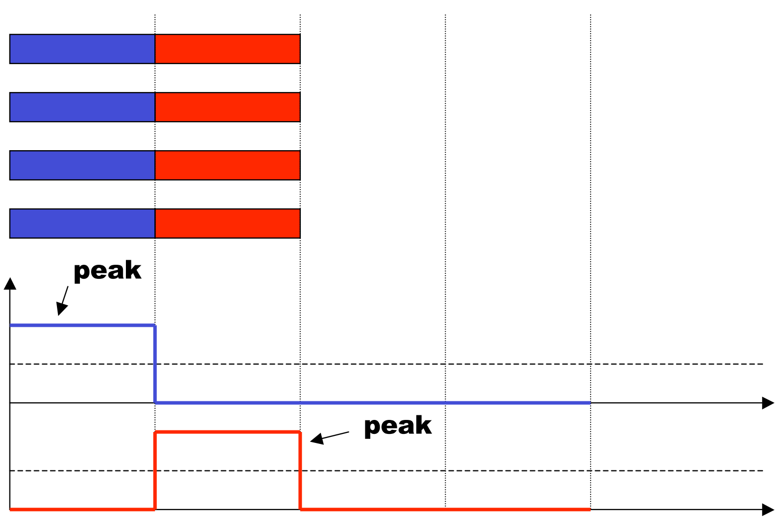
- Premise
  - *Iterative Flattening* is a iterative improvement search procedure for solving multi-capacitated scheduling problems with *makespan minimization* as the objective
  - The concept of iterative flattening search is quite general and provides a framework for designing effective procedures for scheduling optimization
- Three different algorithms:
  - **iFlat** - first version of Iterative Flattening [Cesta,Oddi &Smith 00]
  - **iFlatRelax** - improvement of iFlat [Michel&Van Hentenryck, 04]
  - **STRand** - variation of iFlat [Godard, Laborie &Nuitjen 05]

The MCJSSP scheduling problem

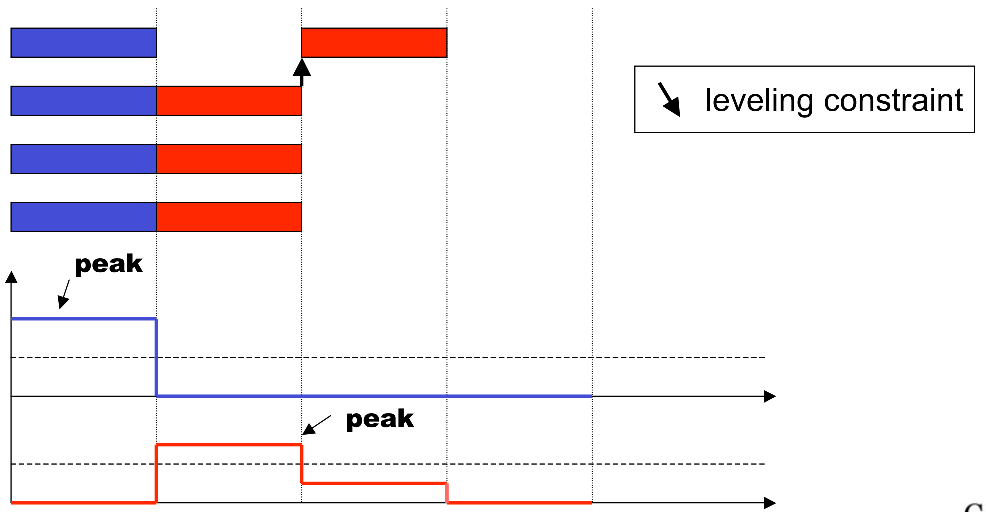


Scheduling problem with **four jobs**, each job has two activities; “red activities” require resource $\mathbf{r_1}$ and “blue activities” require resource $\mathbf{r_2}$

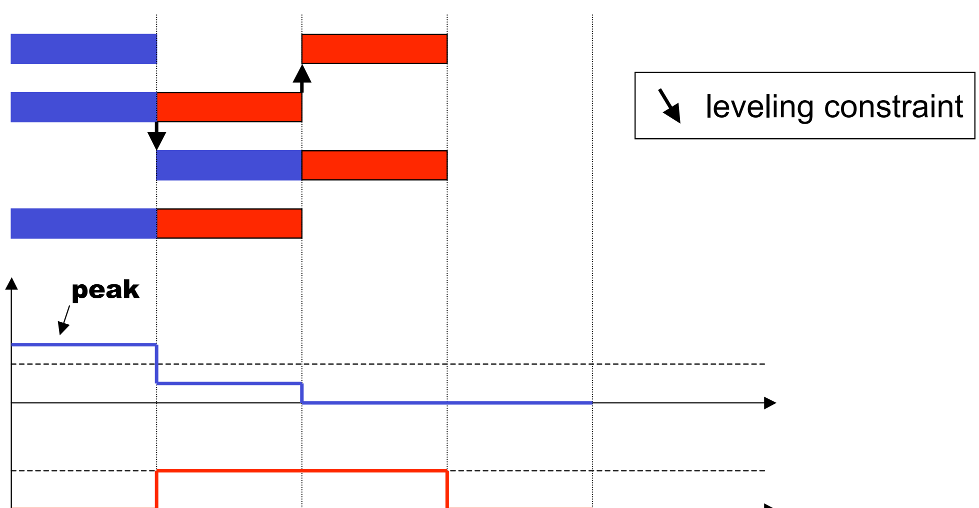
Greedy strategy: example (1)



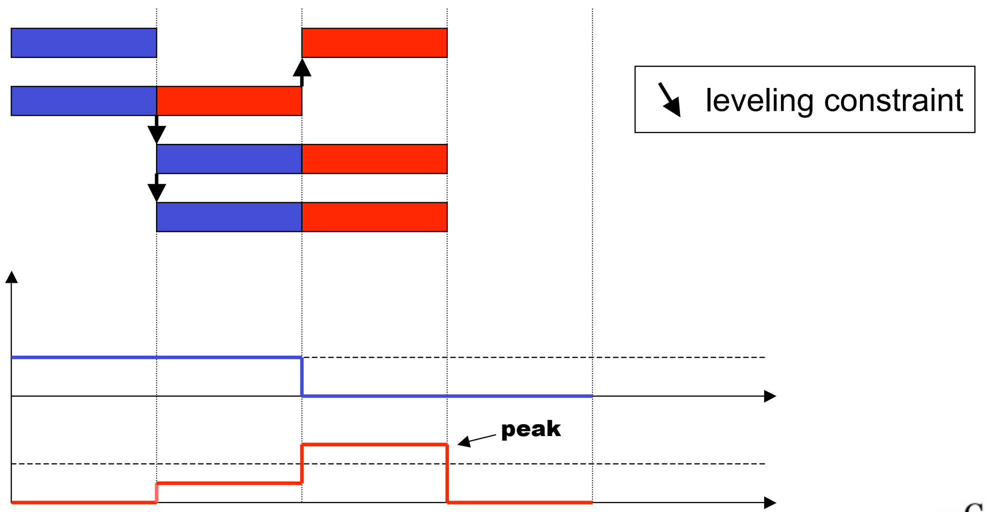
Greedy strategy: example (2)



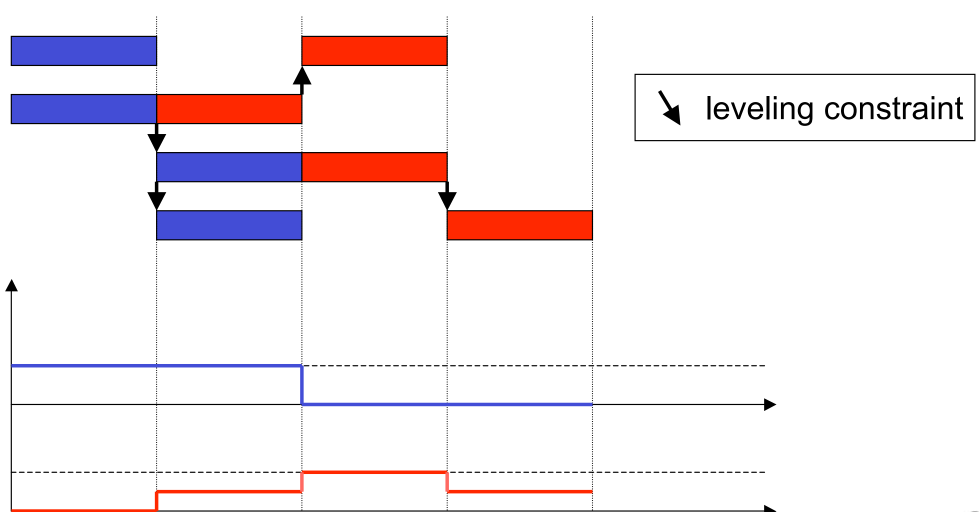
Greedy strategy: example (3)



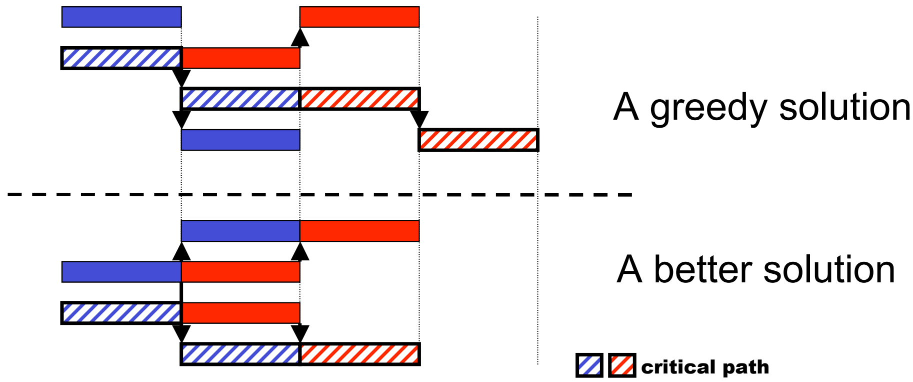
Greedy strategy: example (4)



Greedy strategy: example (5)

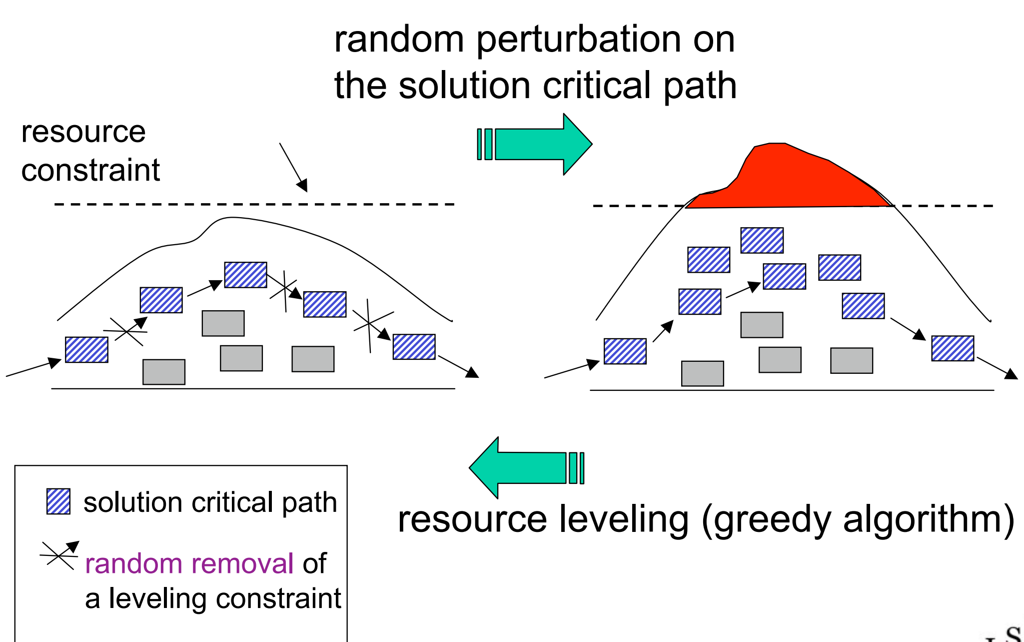


Finding better solutions



- 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



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**)

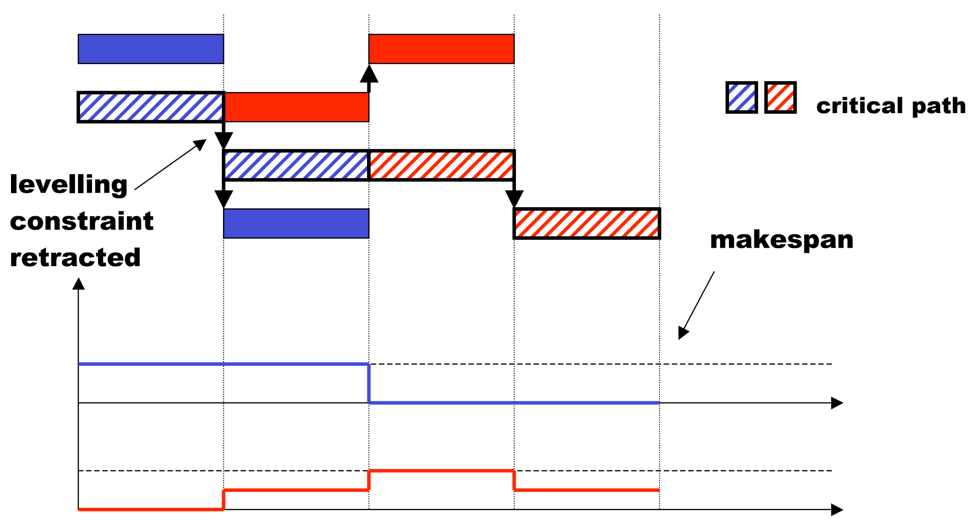
Re-apply the $ESTA_M$ to level (flatten) the new introduced resource conflicts (**flattening step**)

}

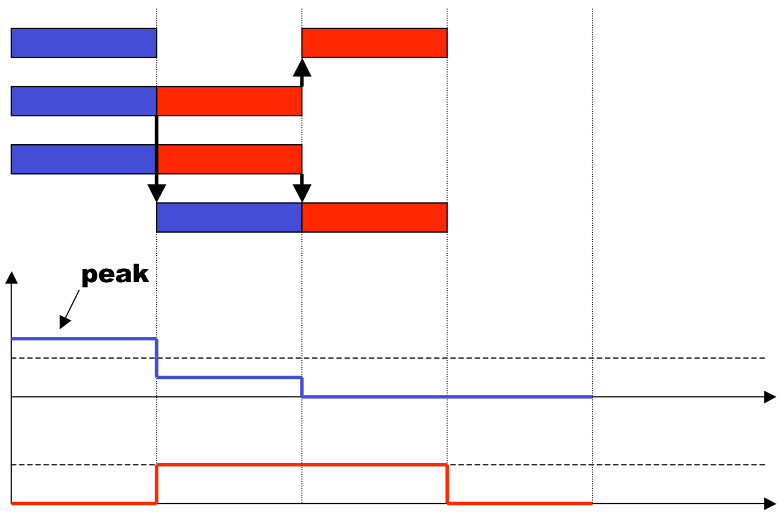
return(*Solution*);

}

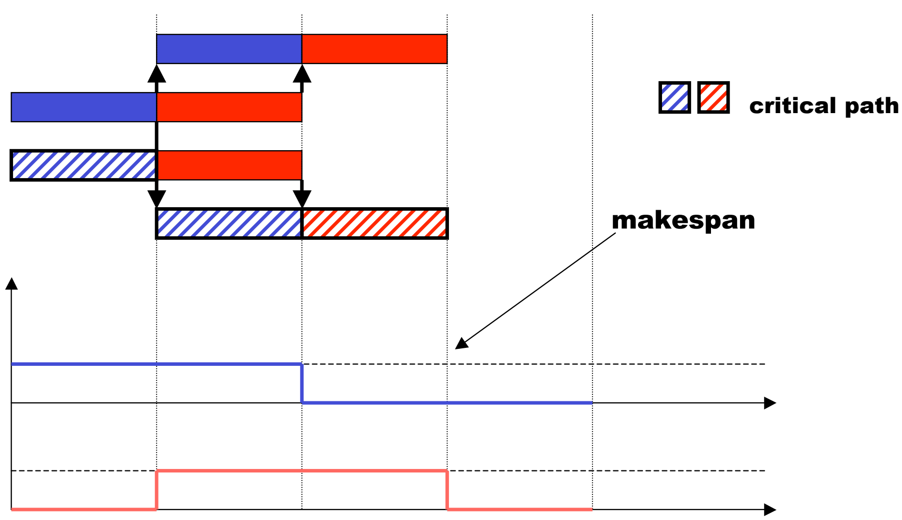
iFlat cycle: critical path analysis



iFlat cycle: shrinking step



iFlat cycle: Flattening step



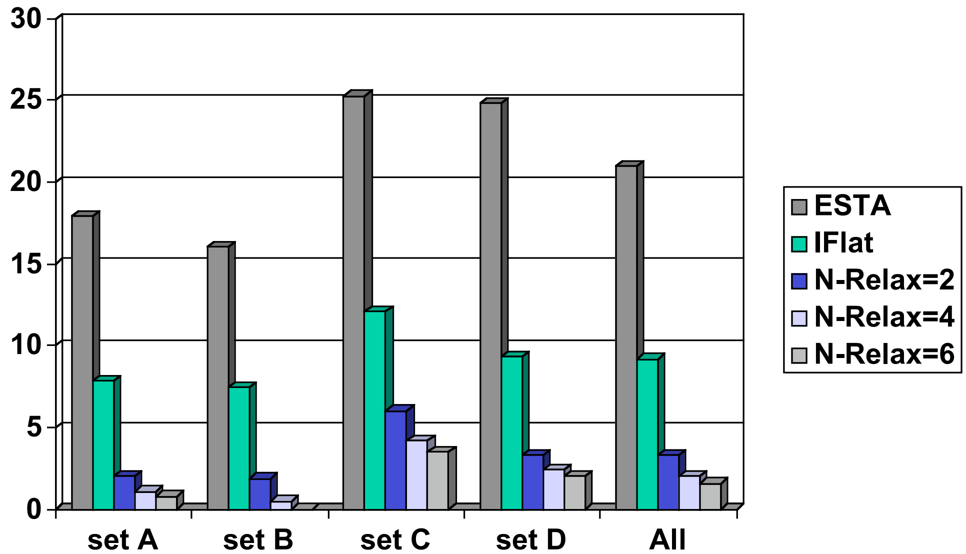
The improved iFlat: *IFlatRelax*

<u>IFlatIRelax</u> (*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 $ESTA_M$ to level (flatten) the new introduced resource conflicts (**flattening step**)
}
return(*Solution*)

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* 04]
  - P_{rem} *= 20%*, *MaxFail = 5000*
  - **Set A** and **B**: NumRestarts=100
  - **Set C** and **D**: NumRestarts=20 (10 in some cases)

Makespan: Δ% from the best UB



A variation of iFlat: STRand

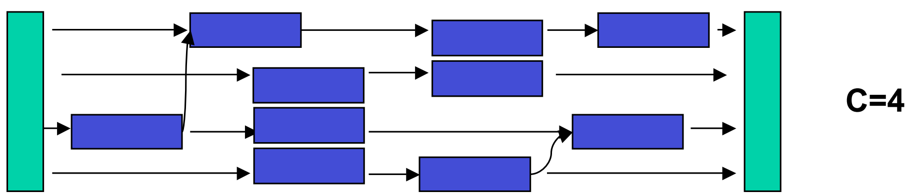
STRand (S_{POS}, P_{rem}, *MaxFail*)

while (the makespan is improved within *MaxFail* iterations) **do**
- Randomly retract a percentage P_{rem} of precedence constraints on the solution in *POS-form* (**removal step**)
- Make a start-time solution *S* (e.g., apply *SetStartTime*) (**flattening step**)
- Generate a new POS-form from *S*
- UpdateBestSolution(S^*_{POS})

end-while
Return(S^*_{POS})

About STRand

- The removal step is performed on a POS-form solution, such that each precedence constraint is a candidate to be removed
- The removal is performed in one step
- The flattening algorithm is a start-time based algorithm (ESTA is precedence constraint based)
- The new fixed-time solution is converted to a *POS-form* before the next removal step



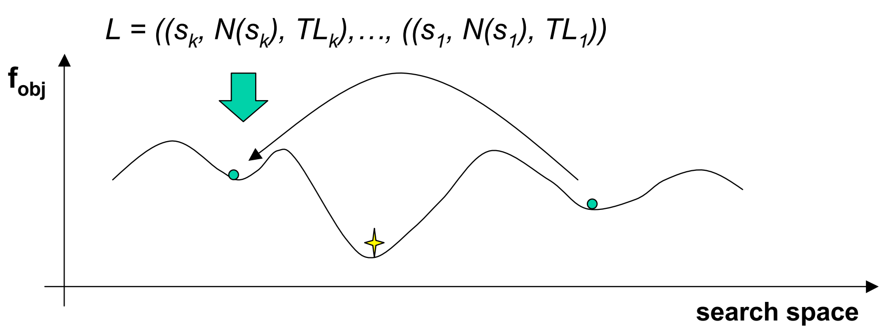
Interleaving Diversification & Intensification

- Back-Jump Tracking (BJT)
- Greedy Randomized Adaptive Search (GRASP)
- Variable Neighborhood Search (VNS)
- Iterated Local Search (ILS)

Back-Jump Tracking

- Given a local search strategy (e.g., tabu-search), BJT works as a *diversification* mechanism by including a further *long-term memory mechanism* to the local search strategy (see for example [Glover 90])
- The idea: during the local search a list of search contexts *(s, state(s))* is stored in a list with max length l_{max}
- Each time the basic local search is *trapped* in a local minima, the search jumps to the last context stored in the list
- For example, for in the case of Tabu Search, a context is given by the *3-tupla (s, N(s), tabu-list)*

Back-Jump Tracking



- In *[Nowicki&Smutnicki 96]* the basic strategy for solving JSSP instances (see above) is used within a BJT strategy: TSAB algorithm
- The use of the BJT strategy gave very valuable results. TSAB is one of the most effective compromise between quality and computational efficiency, included scalability (problem sizes up to 2000 activities)

Greedy Randomized Adaptive Search

- GRASP [Festa&Resende 02] is a meta-heuristic which combines random constructive heuristics and local search
- The idea: the procedure iteratively composes two phases: *construction* and *improvement*. The best solution found is returned upon termination
- GRASP can be effective when the following conditions are satisfied:
  - The construction mechanism samples the *most promising* regions of the search space
  - The constructed solutions belong to basin of attractions of different locally minimal solutions

GRASP

GRASP(*CSP*)
  in: CSP = ⟨*X, D, C*⟩, *termination conditions*
  out: best solution *S**
    S* ← Ø
    while (*termination conditions not met*) **do**
      S ← GreedyRandomSampling(*CSP*)
      *S** ← ApplyLocalSearch(*S*)
      UpdateBestSolution(*S,S**)

GreedyRandomSampling(*CSP***)**
  Propagate(*CSP*)
  if ($\exists D_i = \varnothing$) **then** Return(*failure)*
    else if (*all decision variables* X_i *are set)* **then** Return(*solution*)
      else RCL ← GenerateRestrictedCanditateSet(CSP)
        X_i ← RndSelectDecisionVariable(*RCL*)
        v ← RndSelectValue(D_i))
        set(v,X_i)
        GreedyRandomSampling(*CSP***)**

Variable Neighborhood Search (VNS)

- A meta-heuristic proposed in [Hansen&Mladenovic 01], the basic idea is to apply a strategy based on dynamically changing of *neighborhood structures*
- The strategy considers a set N_k $(k=1 \dots k_{max})$ of neighborhood structures, given a solution *s,* VNS applies the following steps until a termination conditions is met:
  - $k = 1$
  - while ($k < k_{max}$)
    - A solution *s'* is randomly selected in $N_k(s)$
    - s'' ← LocalSearch(s')
    - If s" improves s then s'' replaces *s* and k= k+1; otherwise k=1
- The process of changing neighborhoods in case of no improvements corresponds to a *diversification* mechanism. A "bad place" on the search landscape given by one neighborhood could be a "good place" for another one, that is a place where a good local minimum can be reached

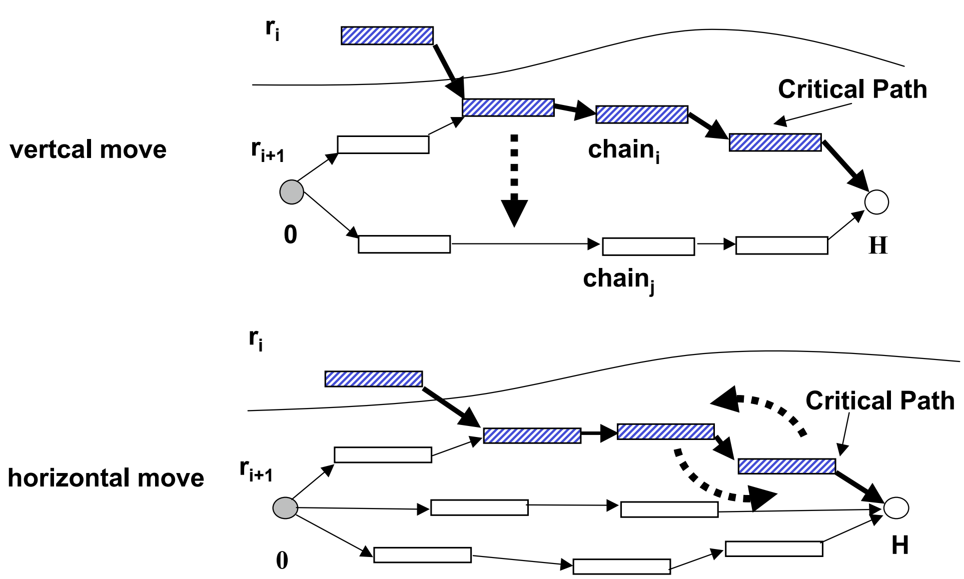
Variable Neighborhood Descent

- A similar strategy to VNS is *Variable Neighborhood Descent* (VND), which basically tries in sequence the neighborhood structures N_k $k=1, \dots, k_{max}$ until find an improved solution or a termination condition is met
- Given a solution *s*, VND applies the following steps until a termination conditions is met:
  - $k = 1$
  - while ($k < k_{max}$)
    - s'' ← LocalSearch(s')
    - If s" improves s then s'' replaces *s* and k= k+1; otherwise k=1

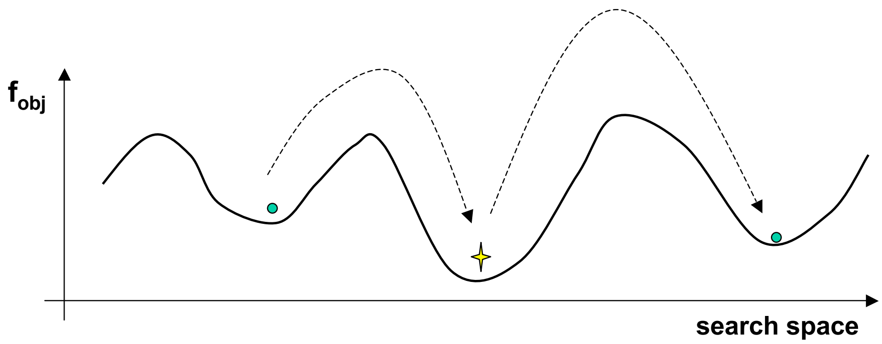
Solving MCJSSP instances

- A variation of the VND strategy TSAB-MC is proposed in *[Oddi 98]* for solving MCJSSP instances as an extension of the algorithm TSAB proposed in *[Nowicki&Smutnicki 96]*
- The procedure is evaluated on 28 MCJSSP instances (see above) taken from *[Nuijten&Aarts 96]* (included 3 instances of modified JSSP from *[Muth&Thompson 63])*
- *The algorithm produced quite comparable results to the one presented in [Nuijten& Aarts 96]*
- In the following we just give a sketch of the two different neighborhood structures proposed

Two types of move



Iterated Local Search



Iterated Local Search

Iterated Local Search(*CSP*)
in: CSP = ⟨*X, D, C*⟩, *termination conditions*
out: best solution *S**
s_0 ← GenerateInitialSolution(CSP)
s* ← LocalSearch(s_0)
while (*termination conditions not met*) **do**
s' ← Perturbation(*s**)
s'* ← LocalSearch(*s'*)
s* ← UpdateBestSolution(s*,s*')
end-while

Iterated Local Search

- Iterated Local Search (ILS) [Lourenco&al 02] is a general meta-heuristic schema. It applies a local search to an initial solution until finds a local optimum; then it perturbs the solution and it restarts the local search
- We observe the importance of the perturbation: a too small perturbation might be unable to escape from a local minimum; on the other hand, a too strong one would make the algorithm similar to random restart local search
- *The acceptance criterion acts as a counterbalance*, as it filters and gives feedback to the perturbation action, depending on the characteristic of the new local minimum (new local minimum should be closer to s than a local minimum produced by random restart)

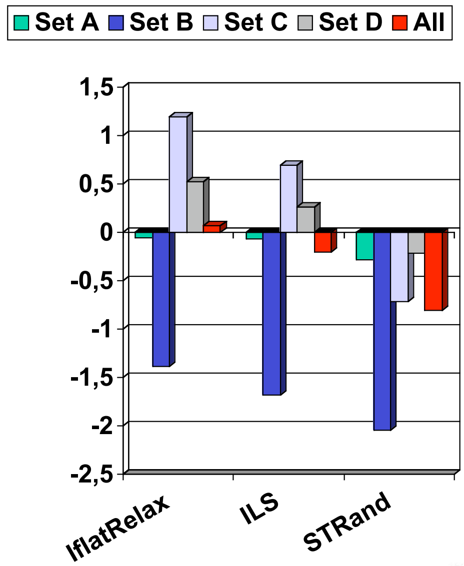
A variation of ISL for MCJSSPs

Iterated Local Search(*MCJSSP*)
in: MCJSSP, *termination conditions*
out: best solution *s**
s ← GenerateInitialSolution(MCJSSP)
while (*termination conditions not met*) **do**
1. Apply IFlatRelax, with probability *p* returns the last solution found with probability *(1-p)* the best solution found
2. Apply the previously described tabu search TSAB-MC with probability *p* returns the last solution found, with probability *(1- p)* the one
3. UpdateBestSolution(s*)

end-while

Comparing meta-heuristic strategies

- MCJSSP benchmarks - sets A, B, C and D (80 instances)
- **ΔUB$_{\%}$** with respect to sets A, B, C and D
- *Three different meta-heuristics*
  - iFlatRelax
  - ILS
  - *STRand*



Outline

- Introduction
- Basic principles
- Constructive methods
- Meta-heuristics
- Conclusions

Conclusions

- Meta-heuristic strategies are successful in practice, they provide a methodology *to trade* problem generality, computational costs (*time* and *space*) and solution quality
- Effective meta-heuristic strategies relay on:
  - efficient *core component algorithms* (e.g., temporal reasoning algorithms), and *incremental* algorithms can play an important role in solving real world problems
  - the balancing between *intensification* and *diversification*
  - the representation of *heuristic control knowledge* to drive the search within the component procedures and to control the composite schema

ISTC CNR INSTITUTE FOR COGNITIVE SCIENCE & TECHNOLOGY

Conclusions

- We describe meta-heuristic schemas which combine basic constructive and local search methods within a CSP reference framework
- CSP solving paradigm clearly separates the *constraints* (semantics, pruning algorithms) from the *search space exploration* (branching schemes, heuristics) giving valuable benefits both from an algorithmic and implementative point of view
- Despite we have presented a set of search procedures for solving multi-capacitated scheduling problems with makespan minimization as the objective, many of the proposed procedures are applicable to a wider range of scheduling problems

ISTC CNR INSTITUTE FOR COGNITIVE SCIENCE & TECHNOLOGY

References

- [Ausiello&al 91] G. Ausiello, G. Italiano, A. Marchetti-Spaccamela and U. Nanni. "Incremental Algorithms for Minimal Length Paths". Journal of Algorithms 12:615-638. (1991)
- [Baker 74] K.R. Baker, Introduction to Sequencing and Scheduling, John Wileys and Sons, 1974.
- [Baptiste&al 01] P. Baptiste, C. LePape, and W. Nuijten, Constraint-Based Scheduling, Kluwer Academics Publishers, 2001.
- [Baptiste&Le Pape 96] P. Baptiste and C. Le Pape, *Edge-Finding Constraint Propagation Algorithms for Disjunctive and Cumulative Scheduling*, in Proc. 15th Workshop of the UK Planning Special Interest Group, 1996.
- [Baptiste&Pape 95] P. Baptiste and C. Le Pape, A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling," in Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1995.
- [Bartusch&al 88] Bartusch M., Mohring R.H., Radermacher F.J., *Scheduling Project Networks with Resource Constraints and Time Windows*, Annals of Operation Research 16, 201-240
- [Beck&al 97] C. Beck, A. Davenport, E. Sitarski, and M. Fox, Texture-based Heuristics for Scheduling Revisited," in Proceedings AAAI-97, 1997.
- [Beck&Fox 00] J. Christopher Beck, Mark S. Fox: Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. Artif. Intell. 117(1): 31-81 (2000)
- [Beck&Fox 97] C. Beck and M. Fox, *Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics*, Artificial Intelligence, 117 (2000) 31-81.
- [Beck&Refalo 01] C. Beck and P. Refalo, *A Hybrid Approach to Scheduling with Earliness and Tardiness Costs*, in Third International Workshop on Integration of AI and OR Techniques (CP-AI-OR'01), 2001.

References

- [Bellman 58] R.E. Bellman. "*On a Routing Problem*". Quart. Appl. Math., 16:87-90 (1958)
- [Bessier&Regin 01] C. Bessiere and J.C. Regin, Refining the basic constraint propagation algorithm," in Proceedings IJCAI'01, 2001.
- [Blum&Roli 03] Blum, C., Roli, A., *Meta-heuristics in combinatorial optimization: Overview and conceptual comparison*. ACM Comput. Surv. 35 (2003) 268–308
- [Carlier&Pinson 90] J. Carlier and E. Pinson, A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem, Annal of Operation Research, vol. 26, pp. 269--287, 1990.
- [Cervoni&al 94] Cervoni, R., Cesta, A., Oddi, A., *Managing Dynamic Temporal Constraint Networks*, in Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS94), *Chicago, IL, June 13-15,* AAAI Press: Menlo Park, CA, 1994.
- [Cesta,Oddi &Smith 00a] A. Cesta, A. Oddi, and S. Smith, *A constraint-based method for project scheduling with time windows*, Tech. Rep., CMU RI Technical Report, 2000.
- [Cesta,Oddi &Smith 00b] Cesta, A., Oddi, A., Smith, S.F., *Iterative flattening: A scalable method for solving multicapacity scheduling problems.* In: AAAI/IAAI, Seventeenth National Conference on Artificial Intelligence. (2000) 742–747
- [Cesta,Oddi &Smith 98] Cesta, A., Oddi, A., Smith, S., *Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems.* In: Proceedings of the Fourth Int. Conf. on Artificial Intelligence Planning Systems (AIPS-98) (1998).
- [Cesta&Oddi 01] A. Cesta and A. Oddi. "*Algorithms for Dynamic Management of Temporal Constraints Networks*". Technical Reports 2001.
- [Cesta&Oddi 96] Cesta, A., Oddi, A., *Gaining Efficiency and Flexibility in the Simple Temporal Problem*, in L. Chittaro, S. Goodwin, H. Hamilton, A.Montanari, Proc. 3rd Int. Workshop on Temporal Representation and Reasoning *(TIME-96)*, Key West, FL, May 19-20, 1996, IEEE Computer Society Press:May 1996

References

- [Cesta&Stella 97] A. Cesta and C. Stella, *A time and resource problem for planning architectures*, in ECP-97, 1997.
- [Cherkassky&al 96] B.V. Cherkassky, A.V. Goldberg and T. Radzic. “*Shortest Paths Algorithms: Theory and Experimental Evaluation*”. Mathematical Programming, 73:129–174 (1996)
- [Cherkassky&Goldberg 96] *Negative-Cycle Detection Algorithms* (1996) Boris V. Cherkassky, Andrew V. Goldberg European Symposium on Algorithms.
- [Cormen&al 90] T.H. Cormen, C.E. Leiserson and R.L. Rivest. “*Introduction to Algorithms*”. MIT Press, Cambridge, MA (1990)
- [Dechter&al 91] Dechter, R., Meiri, I., Pearl, J., *Temporal constraint networks*. Artificial Intelligence 49 (1991) 61–95
- [Demetrescu&al 04] Demetrescu C, Emiliozzi S., Italiano G. F., *Experimental analysis of dynamic all pairs shortest path algorithms,* SODA 2004: 369-378
- [Demetrescu&Italiano 01], Demetrescu C., and Italiano G. F., *Fully Dynamic All Pairs Shortest Paths with Real Edge Weights*, FOCS 2001: 260-267
- [Dijkstra 59] E.W. Dijkstra. “*A Note on Two Problems in Connection with Graphs*”. Numer. Math., 1:269-271 (1959)
- [Dorndorf&al 99] U. Dorndorf, T. Phan Huy, and E. Pesch, A survey of interval capacity consistency tests for time and resource constrained scheduling," in Project Scheduling - Recent Models, Algorithms and Applications, Kluwer Academic Publ., 1999, pp. 213--238
- [Erschler 91] J. Erschler, P. Lopez, and C. Thuriot, Raisonnement temporel sous contraintes de ressources et probl emes d'ordonnancement, Revue d'Intelligence Artificielle, vol. 5, no. 3, pp. 7--32, 1991.

References

- [Erschler 76] J. Erschler, *Analyse sous contraintes et aide a la decision pour certains probl emes d'ordonnancement*, Ph.D. thesis, Universit e Paul Sabatier, 1976.
- [Festa &Resende 02] P. Festa and M. G. C. Resende. *GRASP: An annotated bibliography*. In C. C. Ribeiro and P. Hansen,editors, Essays and Surveys on Metaheuristics, pages 325ñ367. Kluwer Academic Publishers, 2002.
- [Focacci&al 00] F. Focacci, P. Laborie, and W. Nuijten, *Solving scheduling problems with setup times and alternative resources*, in Fifth International Conference on Artificial Intelligence Planning and Scheduling, 2000, pp. 92--101.
- [Ford&Fulkerson 62] L.R. Ford and D.R. Fulkerson. “*Flows in Networks*”. Princeton Univ. Press (1962)
- [Frigioni&al 03] D. Frigioni, A. Marchetti-Spaccamela and U. Nanni. “*Fully dynamic shortest paths in digraphs with arbitrary arc weights*”. J. Algorithms 49(1): 86-113 (2003)
- [Garcia&Laborie 96] F. Garcia and P. Laborie, *New Directions in AI Planning, chapter Hierarchisation of the Search Space in Temporal Planning*, pp. 217--232, IOS Press, Amsterdam, 1996.
- [Garey&Johnson 79] M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.
- [Gerevini&al 96] A. Gerevini, A. Perini and F. Ricci. “*Incremental algorithms for managing temporal constraints*”. Proc. 8th IEEE Conf. on Tools with Artificial Intelligence. (1996)
- [Glover 89] Glover F., Tabu Search - Pat I, ORSA J. Computing, 1,3 (1989), 190-206
- [Glover 90] Glover F., Tabu Search - Pat II, ORSA J. Computing, 2,1 (1990), 4-32

References

- [Godard&al 05] Godard, D., Laborie, P., Nuitjen, W., *Randomized Large Neighborhood Search for Cumulative Scheduling*. In: Proceedings of the International Conference on Automated Planning & Scheduling, ICAPS 2005: 81-89
- [Goldberg 01] A.V. Goldberg. "*A simple shortest path algorithm with linear average time*". In Proceedings of the 9th European Symposium on Algorithms (ESA '01), pages 230-241 (2001)
- [Goldberg&Radzic 93] A.V. Goldberg and T. Radzic. "*A Heuristic Improvement of the Bellman-Ford Algorithm*". Applied Math. Let., 6:3-6 (1993)
- [Grabowsky&al 86] Grabowsky J., Nowicki E., Zdrzalka S., *A Block Approach for Single Machine Scheduling with Release Date and Due Date*. European Journal of Operational Research, 26: 278-285, 1986.
- [Hansen& Mladenovic 01] P. Hansen and N. Mladenovic. Variable neighborhood search: Principles and applications. European Journal of Operational Research, 130:449-467, 2001.
- [Harvey&Ginsberg 95] W. Harvey and M. Ginsberg, *Limited discrepancy search*, in In Proceedings of the Fourteenth International Joint Conference on Arti cial Intelligence, August 1995.
- [Italiano 86] G.F. Italiano. "*Amortized efficiency of a path retrieval data structure".* Theoretical Computer Science, 48:273-281. (1986)
- [Junker 00] Junker, U., *Preference-Based Search for Scheduling*. In: Proceedings of the 17th National Conference on Artificial Intelligence. (2000) 904–909
- [Kumar 92] V. Kumar, *Algorithms for Constraint Satisfaction Problems: a Survey*, AI magazine, vol. 13, no. 1, pp. 32-44, 1992.
- [Laborie 03] P. Laborie. "*Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results*". Artificial Intelligence. 143(2):151 – 188. (2003)

References

- [Laborie&Ghallab 95] P. Laborie and M. Ghallab, *Planning with sharable resource constraints*, in Fourteenth IJCAI, 1995, pp. 1643--1649.
- [Lawrence 84] Lawrence, S., *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement).* Technical report, Graduate School of Industrial Administration, Carnegie Mellon University (1984)
- [Le Pape&Baptiste 99] C. Le Pape and P. Baptiste, *Heuristic Control of a Constraint-Based Algorithm for the Preemptive Job-Shop Scheduling Problem*, Journal of Heuristics, vol. 5, pp. 305--332, 1999.
- [LePape 94] C. Le Pape, *Implementation of Resource constraints in ILOG Schedule: A Library for the Development of Constraint-Based Scheduling systems*, Intelligent Systems Engineering, vol. 3, no. 2, pp. 55--66, 1994.
- [Lourenco&al 02] H. R. Lourenco, O. Martin, and T. Stutzle. Iterated local search. In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, volume 57 of International Series in Operations Research & Management Science, pages 321-353. Kluwer Academic Publishers, Norwell, MA, 2002.
- [Michel&Hentenryck 04] Michel, L., Van Hentenryck, P., *Iterative relaxations for iterative flattening in cumulative scheduling*. In: ICAPS. (2004) 200–208
- [Michel&VanHentenryck 03] L. Michel and P. Van Hentenryck. "*Maintaining Longest Paths Incrementally*". Proc. CP-03. (2003)
- [Mohr&al 86] R. Mohr and T. Henderson, *Arc and path consistency revisited*, Artificial Intelligence, vol. 28, pp. 225-233, 1986.
- [Montanari 74] Montanari, U. (1974). Networks of Constraints: Fundamental Properties and Applications to Picture Processing. Information Sciences, 7 , 95–132.
- [Muth&Thompson 63] J.F. Muth and G.L. Thompson. Industrial scheduling. Englewood Cliffs, 1963.

References

- [Neumann&Schwindt 97] Neumann K., Schwindt C., *Activity-on-node Networks with Minimal and Maximal Time Lags and Their Application to Make-to-Order Production*, Operation Research Spektrum, 19, 205-217
- [Neumann&Schwindt 99] K. Neumann and C. Schwindt, *Project scheduling with inventory constraints*, Tech. Rep. WIOR-572, Institut fur Wirtschaftstheorie und Operations Research. Universitat Karlsruhe, 1999.
- [Nowicki&Smutnicki 96] Nowicki, E., Smutnicki, C., *A Fast Taboo Search Algorithm for the Job Sho Problem*. Management Science 42 (1996) 797–813
- [Nuijten 94] W. Nuijten, *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*, Ph.D. thesis, Eindhoven University of Technology, 1994.
- [Nuijten&Aarts 96] Nuijten,W., Aarts, E., *A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling*. European Journal of Operational Research 90 (1996) 269–284
- [Nuijten&Le Pape 98] W. Nuijten and C. LePape, *Constraint-Based Job-Shop Scheduling with ILOG Scheduler*, Journal of Heuristics, vol. 3, pp. 271--286, 1998.
- [Oddi 98] Angelo Oddi, *Solving Multiple Capacitated Scheduling Problems with Tabu Search*. ECAI 1998: 520-521
- [Oddi&Cesta 97] A. Oddi, A Cesta, *A Tabu Search Strategy to Solve Scheduling Problems with Deadlines and Complex Metric Constraints*. ECP 1997: 351-363
- [Oddi&Smith 97] Oddi A., Smith S.F., *Stochastic Procedures for Generating Feasible Schedules*, in Proceedings AAAI-97, 1997.
- [Pallottino 84] S. Pallottino. *Shortest-Path Methods: Complexity, Interrelations and New Propositions*. Networks, 14:257-267 (1984)

References

- [Pesant&Gendreau 96] G. Pesant and M. Gendreau, *A View of Local Search in Constraint Programming*, in Proc. 2nd International Conference on Principles and Practice of Constraint Programming, 1996.
- [Policella&al 04] Policella, N., Smith, S.F., Cesta, A., Oddi, A., *Generating robust schedules through temporal flexibility*. In: ICAPS. (2004) 209–218
- [Ramalingam&Reps 96] G. Ramalingam and T. Reps. "*On the computational complexity of dynamic graph problems*". Theoretical Computer Science, 158(1-2):233-277 (1996)
- [Regin 94] J.C. Regin, *A filtering algorithm for constraints of difference in CSPs*, in Proc. 12th National Conference on Artificial Intelligence, 1994.
- [Sakkout&Wallace 00] H. El Sakkout and M. Wallace, *Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling*, Constraints, vol. 5, no. 4, pp. 359--388, 2000.
- [Smith&Cheng 93] S. F. Smith and C. Cheng, *Slack-based heuristics for constraint satisfaction scheduling*, in Proc. 11th Nat. Conf. on AI, 1993, pp. 139--144.
- [Sourd&Nuijten 00] F. Sourd and W. Nuijten, *Multiple-machine lower bounds for shop scheduling problems*, INFORMS Journal of Computing, vol. 4, no. 12, pp. 341--352, 2000.
- [Tarjan 72] R.E. Tarjan. "Depth first search and linear graph algorithms". SIAM Journal of Computing, 1(2):146,160. (1972)
- [Tarjan 83] R.E. Tarjan. "*Data Structures and Network Algorithms*". Society for Industrial and Applied Mathematics (1983)
- [Torres&Lopez 00] P. Torres and P. Lopez, *Overview and possible extensions of shaving techniques for job-shop problems*, in 2nd International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CPAI-OR'2000), March 2000, pp. 181--186.

References

- [Torres&Lopez 99] P. Torres and P. Lopez, *On Not-First/Not-Last Conditions in Disjunctive Scheduling*, European Journal of Operational Research, 1999.
- [Tsang 93] Tsang, E. (1993). Foundations of Constraint Satisfaction. Academic Press.
- [Warshall 62] S. Warshall. "*A theorem on Boolean matrices*". Journal of ACM, 9(1):11-12. (1962)

About the author

Angelo Oddi is a research scientist at the Institute of Cognitive Science and Technology of the Italian National Research Council (ISTC-CNR). He received his Master Degree in Electronic Engineering from University of Rome ``La Sapienza'' in 1993 and his PhD in Medical Computer Science from the same university in 1997. He has been visiting scholar at the Intelligent Coordination and Logistics Laboratory of the Robotics Institute at Carnegie Mellon University in 1995-6. His work focuses on the application of Artificial Intelligence techniques for scheduling, automated planning and constraint reasoning. In particular, he has proposed several algorithms for temporal reasoning and developed both local search and randomized approaches for schedule optimization. Regarding his professional activities, he has published more than 40 papers, both in journals and in proceedings of international conferences, and has a wide experience in the design of intelligent systems for real world applications. In particular, he has been involved in several projects financed by the Italian and European Space Agencies (ASI/ESA) concerning the development of intelligent mission planning support software.