Constraint-Based Scheduling

CHR Summer School 2013, Berlin
Dr. Armin Wolf

What is Constraint-Based Scheduling?

The allocation of actions to resources and in time such that
• the availability of resources and their capacities are respected
• temporal restrictions of the actions are considered: due/release dates, durations
• temporal relations between actions are satisfied: precedences, etc.

Constraint-Based Scheduling (all quantities have integer values)
uses local/global Operations Research methods for different kinds of problem classes:
• Two important problem classes: disjunctive and cumulative scheduling
• Local methods:
  • Forbidden Regions
  • Detectable Precedences
• Global methods:
  • Timetabling
  • Overload Checking
  • Edge Finding
  • Not-First/Not-Last Detection
➤ Performs "Pruning"/"Filtering", i.e. removal of inadmissible values
Things that may be scheduled I

Processes (Jobs)
• (standardized) work flows
  • consisting of activities
  • which require resources

Activities (Tasks)
• process steps, courses, treatments, pre- and post-processing, ...
• either preemptive or non-preemptive
• having minimal/maximal duration
• to be performed in a time slot
• require in general different resources

Things that may be scheduled II

Resources
• machines, tools, devices, persons, rooms, consumable products, energy, ...
• in general of restricted capacities
• are exclusive, alternative or cumulative
• are either consumed (e.g. water, energy) or re-usable (machines, persons)

Spatio-Temporal Relationships
• sequences/orders of activities
• minimal/maximal gaps between activities (e.g. between usage and cleaning)
• (sequence-dependent) set-up times (and costs)
• resource-dependent durations (e.g. slow and fast devices)
Modeling of Activities I

Task $t$

- is in general a non-preemptive activity
- has in general a variable start time $s = t.\text{start} \in S$, $\text{est} := \min(S)$, $\text{lst} := \max(S)$
- has in general a variable duration $d = t.\text{duration} \in D$
- has in general a variable end time $e = t.\text{end} \in E$
- satisfies the constraint $t.\text{start} + t.\text{duration} = t.\text{end}$

$\min(E)$ $\max(E)$

- N.B.: In general the value sets $S, D, E$ of the variables $s, d, e$ are called the domains of these variables: $\text{Dom}(s)$, $\text{Dom}(d)$, $\text{Dom}(e)$.

Modeling of Activities II

Task $t$

- has in general a variable capacity $t.\text{capacity} \in C$
- has in general a variable resource (identifier) $t.\text{resourceId} \in R$
- may be optional: $t.\text{duration} \in \{0, p, \ldots, q\}$ $0 < p \leq q$

→ useful to implement tasks on alternative resources:

A task on alternative resources is modeled by optional task with the same start times and the constraint that exactly one optional task must be mandatory.
Modeling of Resources I

Resource $r$
- is either exclusive with capacity $C_r = 1$
- or alternative exclusive $\Rightarrow$ collection/pool of congeners
- or cumulative with capacity $C_r > 1$
- or alternative cumulative $\Rightarrow$ collection/pool of congeners

$\Rightarrow$ The capacity constraint has to be satisfied: $\forall t \forall \tau : \sum_{t_{\text{start}}}^{t_{\text{end}}} t_.capacity \leq C_r$

Modeling of Resources II

Reservoirs of consumer goods /commodities
- Here: "at-once" production/consumption at schedulable events $C_i$
- Can be modeled as a cumulative resource with fixed start/end activities $\Rightarrow$ Exercise!
- Specialized pruning algorithms are also available (e.g., "envelope computation")
Disjunctive Scheduling

- means temporal "non-overlapping" of activities
  - either on exclusive resources
  - or on cumulative resources where the activities’ resource consumption is \( > \frac{C_r}{2} \)

Cumulative Scheduling

- means that activities may overlap in time
  - generalizes disjunctive scheduling (see capacity constraint before)
  - Operations Research methods are also applicable for disjunctive scheduling: \( C_r = 1 \)

Modeling of Multi-Resources Activities

**Example: Cleaning of Devices**

- allocates the device to be cleaned **and** requires a cleaning worker at the same time

\( \Rightarrow \) model: two tasks with identical times:
- the tasks: `cleanOnDevice` and `cleanByPerson` with
  - `cleanOnDevice.start = cleanByPerson.start`
  - `cleanOnDevice.duration = cleanByPerson.duration`
  - `cleanOnDevice.end = cleanByPerson.end`

\( \Rightarrow \) however on different/disjoint resources:
- `cleanOnDevice.resourceID \in \text{Devices}`
- `cleanByPerson.resourceID \in \text{Employees}`
- \( \text{Devices} \cap \text{Employees} = \emptyset \)
Modeling the Usage of Common Resources

Example: Choice within a Set of Tools
- Different kinds of work require specific-purpose tools (some of them are more general)
- However, which tools should be considered together?
  - naive: there are in one big pool
  - more sophisticated: consider the finest partition

The reason: reduction of algorithmic complexity: \( p_1^n + \ldots + p_n^n < (p_1 + \ldots + p_n)^n \)

Further Modeling Aspects

Temporal Relationships
- \( y - x \leq d \)
  - represented by a distance graph
  - computation of all minimal distances with the Floyd-Warshall algorithm \( O(n^3) \)
  - (sequence-dependent) set-up times and costs
  - resource-dependent processing times
  - modeling by use of "element" constraint:
    \[
    \text{element}(v, [\text{val}_0, \ldots, \text{val}_k], i) \leftrightarrow v = \text{val}_i
    \]
    \[
    \text{element}((\text{duration}, [\text{dur}_0, \ldots, \text{dur}_{n-1}]), \text{resourceID}) \leftrightarrow \text{duration} = \text{dur}_{\text{resourceID}}
    \]
  - etc.
Forbidden Regions in Disjunctive Scheduling
... and for some tasks of Cumulative Scheduling as well

The rule:

- Any start time of a task $j$ is infeasible if another task $i$ cannot be scheduled before nor after $j$.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\min(S_i) + d_i - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j$</td>
<td>$\min(S_j), \max(S_j) - d_j + 1$</td>
</tr>
<tr>
<td></td>
<td>$\max(S_j)$</td>
</tr>
<tr>
<td></td>
<td>$\max(S_j) + d_j$</td>
</tr>
</tbody>
</table>

- **local** pruning rule with **quadratic complexity** (in the number of tasks)
- removes in general “inner” values of the start time domains

Detectable Precedencies I in Disjunctive Scheduling
... and for some tasks of Cumulative Scheduling as well

The rule:

- If a task $q$ cannot be scheduled after a task $p$ then $q$ must be before $p$

- More formally: $\text{ect}_p > \text{lst}_q \Rightarrow q << p$

- **local** pruning with **quadratic time complexity** (in the number of tasks)
Detectable Precedencies II

Generalization:

• If all tasks of a task set $\Omega$ are before $p$ then $p$ cannot start before the earliest completion time of $\Omega$

$$\Omega = \{t \mid t \ll p\} \Rightarrow \text{est}_p^* = \max(\text{est}_p, \text{ect}_\Omega)$$

• More formally:

$$\Omega \subseteq \Theta \
\Rightarrow \text{est}_p^* = \max_{\Theta \in \Omega}(\text{est}_p + d_\Theta)$$

$$\text{ect}_\Omega := \max_{\Theta \in \Omega}(\text{est}_\Theta + d_\Theta)$$

$$\text{est}_\Theta := \min_{t \in \Theta} \text{est}_t \text{ and } d_\Theta := \sum_{t \in \Theta} d_t$$

Detectable Precedences III

• Generalized pruning is possible in $O(n \log n)$ time where $n$ is the number of tasks

Petr Vilím: $O(n \log n)$ Filtering Algorithms for Unary Resource
In proceedings of CPAIOR 2004, Nice, France, April 2004,

• Symmetric pruning of the latest completion times is analogous.
Notations

For any task $i$

- it is assumed that the task has fixed duration $d_i$ and fixed capacity $c_i$ (either $=1$ or $>1$)
- and for $i$, variable start time $s_i \in S_i$ let
  - $\text{est}_i := \min(S_i)$ be its earliest start time (a.k.a. release date)
  - $\text{lct}_i := \max(S_i)$ be its latest start time
  - $\text{lct}_i := \min(S_i) + d_i$ be its earliest completion time
  - $\text{ect}_i := \max(S_i) + d_i$ be its latest completion time (a.k.a. due date)
  - $e_i := d_i c_i$ be its energy

For any set of tasks $\Omega$ let

- $\text{est}_\Omega := \min_{i \in \Omega} \text{est}_i$, $\text{lct}_\Omega := \max_{i \in \Omega} \text{lct}_i$, $e_\Omega := \sum_{i \in \Omega} e_i$
- if $\Omega$ is the empty set let $\text{est}_\Omega := +\infty$, $\text{lct}_\Omega := -\infty$ and $e_\Omega := 0.$

Cumulative Scheduling – Definition

We focus on Cumulative Scheduling because it is a generalization of Disjunctive Scheduling

A **Cumulative Scheduling Problem (CuSP)** is defined by

- a set of tasks $T$
- a resource with capacity $C$

A solution of the CuSP is a schedule that assigns a start time $s_i$ to each task $i$ such that

\[
\forall i \in T : \text{est}_i \leq s_i \leq s_i + d_i \leq \text{lct}_i
\]

\[
\forall \tau : \sum_{i \in \tau, s_i \leq \tau < s_i + d_i} c_i \leq C
\]
Cumulative Scheduling – E-Feasibility

**E-Feasibility** – a necessary condition for a solution

- For any non-empty subset $\Omega$ of the tasks in $T$ the available energy must be greater or equal to the required energy:

$$\forall \Omega \subseteq T, \Omega \neq \emptyset : C(lct_\Omega - est_\Omega) \geq e_\Omega$$

**How fast can E-Feasibility be checked?**

- naive approach: consideration of all subsets: $O(2^n)$ where $n = |T|$.
- better approach: consideration of task sets $\Omega$ only including all tasks within the intervals $[est_{\Omega}, lct_{\Omega}]$, so called **task intervals**: $O(n^2)$:

For any two tasks $p, q$ in $T$ the set

$$\{ j | est_j \leq est_p \text{ and } lct_j \leq lct_q \}$$

is called a **task interval**

- best approach: $O(n \log n)$: "sweeping" over task intervals

Cumulative Scheduling – Timetabling I

**Compulsory Parts**

- If $|lct_i - d_i < est_i + d_i$ holds for a task $i$ then the task $i$ has a (non-empty) **compulsory part**, i.e.

  the task $i$ always occupies capacity $c_i$ within the interval $[lct_i - d_i, est_i + d_i]$

**Timetabling**

- The timetable $TT$ of the CuSP is the aggregation of all compulsory parts of the tasks in $T$

- The CuSP has not any solution if the timetable exceeds the capacity $C$ at some time $\tau$
Cumulative Scheduling – Timetabling II

• Timetable $TT$ represent the capacity profile occupied in each feasible schedule.

$\Rightarrow$ There is not any feasible schedule if this profile exceed the capacity limit:

• A necessary condition for feasibility/satisfiability computable in $O(n \log n) \Rightarrow$ Exercise!

Cumulative Scheduling – Edge Finding I

The Main Idea
Discover a set of tasks $\Omega$ and a task $i$ not in $\Omega$ such that in any solution, all the tasks in $\Omega$ end before the end of $i$ – denoted by $\Omega \prec i$.

Then the earliest start time of $i$ can be adjusted by the following rule:

$\Omega \prec i \Rightarrow \text{est}_i \geq \text{est}_i + \frac{1}{c_i} \text{rest}(\Theta, c_i)$

for all $\Theta \subseteq \Omega$ such that $\text{rest}(\Theta, c_i) > 0$, where

$\text{rest}(\Theta, c_i) = \begin{cases} \epsilon_i - (C - c_i)(\text{lct}_{\Theta} - \text{est}_{\Theta}) & \text{if } \Theta \neq \phi \\ 0 & \text{otherwise.} \end{cases}$

What is the strongest possible update of $\text{est}_F$?
Cumulative Scheduling – Edge Finding II

The possible updates

- The tasks in $\Omega = \{A, B, C, D, E\}$ they are not obviously ending before $F$ ends.
- However, if we assume that $F$ ends at or before a task in $\Omega$ then E-Feasibility is violated in the interval $[\text{est}(F), \text{lct}(A, C, D)]$ thus the condition $\Omega < F$ is satisfied.
- A first possible update with $\Theta = \Omega$ results in

A second – even better – update is possible with $\Theta = \{B, E\}$ independently whether the first update was performed or not!

$\Rightarrow$ This allows a lazy iterative updating; the best updates will be found eventually

Cumulative Scheduling – Edge Finding III

The possible updates (cont.)
Cumulative Scheduling – Edge Finding IV

**The relation Ω < i**

- Let a set of tasks Ω and a task i not in Ω be given. The “edge finding” relation holds in two cases:
  1. \( \text{est}_i + d_i \geq \text{lct}_{Ω} \Rightarrow Ω < · i, \)
  2. \( e_{Ω∪\{i\}} > C(\text{lct}_{Ω} - \text{est}_{Ω∪\{i\}}) \Rightarrow Ω < · i. \)

- The first case is proved by contradiction (see example above)
- The second case holds obviously

Cumulative Scheduling – Edge Finding V

**Computational Results**

- There are algorithms which perform updates (not always the strongest) in \( O(n^2) \) time and \( O(n) \) space, e.g.


- Strongest updates require at most \( n-1 \) iterations

- Updating the lcts of the tasks with respect to edge finding is performed symmetrically: \( -\text{lct} \rightarrow \text{est}; -\text{est} \rightarrow \text{lct} \)
Cumulative Scheduling – Not-First/Not-Last Filtering I

The Main Idea
Discover a set of tasks \( \Omega \) and a task \( i \) not in \( \Omega \) such that in any solution \( i \) cannot be the first (resp. last) in \( \Omega \cup \{i\} \) (otherwise there will be an overload).
Then the earliest start time (resp. latest completion time) of \( i \) can be updated to the earliest completion time (resp. latest start time) of all tasks within the set \( \Omega \):

Consider the following example:
There task D cannot be the last regarding \( \Omega = \{A, B, C\} \).

Cumulative Scheduling – Not-First/Not-Last Filtering II

The Pruning Rules
- More formally the rules for an update are:
  \[
  \forall \Omega \subset T, \forall i \in T \setminus \Omega:
  \begin{align*}
  \text{est}_i < \min_{j \in \Omega} (\text{ect}_j, e_{i,\Omega}) + c_i \cdot \left( \min_{j \in \Omega} (\text{lct}_j, \text{est}_j) \right) \Rightarrow \text{est}_i \geq \min_{j \in \Omega} (\text{ect}_j) \\
  \text{(Not - First)}
  \end{align*}
  \]
  \[
  \begin{align*}
  \text{max}_{i \in \Omega} (\text{lct}_i) < \text{lct}_i + c_i \cdot (\text{lct}_i - \text{max}_{j \in \Omega} (\text{lst}_j, \text{est}_j)) \Rightarrow \text{lct}_i \leq \text{max}_{i \in \Omega} (\text{lst}_i) \\
  \text{(Not - Last)}
  \end{align*}
  \]
- What are the best updates and how to compute them?
Cumulative Scheduling – Not-First/Not-Last Filtering III

Computational Results

- There are algorithms which perform updates (not always the strongest) in $O(n^2 \log n)$ time, e.g.


- Strongest updates require at most $n-1$ iterations

Constraint-Based Energy Management

Load Balancing in Production
- e.g. peak reduction
  - by the use of a "dummy" task
  - requiring (maximal) capacity over the scheduling horizon

time optimized $\Rightarrow$ high variations / peak loads
energy optimized $\Rightarrow$ marginal time extension
Cost Reduction Based on Time-Variable Energy Tariffs

Motivation

- time-variable energy prices offers the opportunity to shift loads into cheap periods

Goal

Development of a constraint-based model for energy cost-optimized Scheduling
- for shiftable capacity loads
- under a time-variable capacity tariff
- over a fixed scheduling horizon
Consumers

Each Consumer $V$ is defined by
- its start time: $\text{start}(V)$
- its end time: $\text{end}(V)$
- its duration: $\text{duration}(V)$

Load Profiles of Electrical Consumers (e.g. Dish Washers)

- Load profiles (capacity usages) are device-dependent and usage-specific ("mode/program")
- approximation by tasks with piecewise-constant capacities
Application Scenario: Reduction of Energy Costs in Households

Cost-Optimized Scheduling of Appliances in Households

- consideration of day-ahead energy prices (from EEX)
- real (measured) device-specific load profiles
- individual and process-specific constraints (time, capacity, etc.)


Consumers and its Tasks I

Each consumer $V$ consists of a consecutive sequence of tasks: $[A^V_{1}, \ldots, A^V_{k}]$
Consumers and its Tasks II

Each task is defined by its (variable) start and end times, its (fixed) duration, its fixed consumption and its cost (depending on time).

Tasks of a consumer are executed consecutively. Breaks are tasks with consumption = 0. Consecutive tasks with the same consumption may be combined to one task.

Constraints

- Temporal: start(A) + duration(A) = end(A), end(A) = start(A_{i+1}), ...
- (Locally consistent)
- User-defined: off-times, sequences, non-overlapping, capacity restrictions, etc.
- For cost-calculation: consumption costs for consumers and their tasks
**Off-Time Periods**

A consumer $V$ is not schedulable within its **off-times periods** (pre-defined time periods)

- The domains $\text{Dom}(\text{start}(V))$ and $\text{Dom}(\text{end}(V))$ has to be defined accordingly:

$$
\begin{align*}
[a - \text{duration}(V); b + 1] \subseteq \text{Dom}(\text{start}(V)) \\
[a + 1; b + \text{duration}(V)] \subseteq \text{Dom}(\text{end}(V))
\end{align*}
$$

**Capacity Restrictions**

The total capacity request (power request) of all consumers must not exceed a given capacity limit at any time (e.g. a limit is always given by the main fuse)

$$
\sum_{A \in \{A_1, \ldots, A_m\}} \text{consumption}(A) \leq \text{limit}
$$

This restriction will satisfied by the proper use of a **Cumulative** constraint:
Alternative Modeling of Consumers

How to model consumers with tasks?

- presented approach: consecutive sequence of tasks
- alternative (or additional) approach: horizontal stacking of tasks

What is the advantage of the alternative approach (remark: timetabling)?

Non-Overlapping (Mutual Temporal Exclusion)

Exclusive Usage of Resources (Locations, Devices, Humans, etc.)

- Repeated activations of the same device (washing)
- Operation of different devices by one operator (ironing, vacuum cleaning)

→ 2 consumers (i.e. their tasks) are never active at the same time:

\[(\text{end}(V_i) \leq \text{start}(V_j)) \lor (\text{end}(V_j) \leq \text{start}(V_i))\]

→ proper use of "parallel" tasks (with capacity 1) on Exclusive Resource constraints (for disjunctive scheduling; effective pruning algorithms)
Sequencing of Activities

Specific Workflows
- require a fixed sequence of activities (consumers), e.g. washing, drying, ironing, in particular with specific gaps
- Modeling by the use of Sum constraints (locally consistent):
  - \( \text{end}(V_1) + \text{delay}(V_1, V_2) = \text{start}(V_2) \)
  - with \( \text{Dom}(\text{delay}(V_1, V_2)) \subseteq [0; T - \text{duration}(V_1) - \text{duration}(V_2)] \)

\[ \begin{array}{c}
0 & \text{end}(V_1) & \text{delay}(V_1, V_2) & \text{start}(V_2) & T \\
V_1 & \rightarrow & \rightarrow & \rightarrow & V_2
\end{array} \]

- Special cases
  1. unrestricted delay:
     - if \( \text{Dom}(\text{delay}(V_1, V_2)) = [0; T - \text{duration}(V_1) - \text{duration}(V_2)] \)
     - then \( \text{end}(V_1) \leq \text{start}(V_2) \)
  2. without delay: if \( D(\text{delay}(V_1, V_2)) = 0 \) then \( \text{end}(V_2) = \text{start}(V_2) \)

Shifts I

Settings:
- the scheduling horizon is split into "shifts" of the same duration (e.g. days of a week)
- Consumers may be in several shifts, then let
  - \( \text{PERIODS} \) – the number of shifts
  - each shift be identified by a number – from 0 to \( \text{PERIODS} - 1 \)
  - \( \text{SHIFTDURATION} \) – the fixed duration of each shift
- For any consumer \( V \) introduced the following additional variables
  - \( \text{shiftnumber}(V) \) – a variable with \( \text{Dom}(\text{shiftnumber}(V)) \subseteq [0; \text{PERIODS} - 1] \)
  - \( \text{shiftstart}(V) \) – a variable with \( \text{Dom}(\text{shiftstart}(V)) \subseteq [0; \text{SHIFTDURATION} - 1] \)
  - \( \text{shiftend}(V) \) – a variable with \( \text{Dom}(\text{shiftend}(V)) \subseteq [1; \text{SHIFTDURATION}] \)
Shifts II

Constraints:
- A consumer \( V \) must start and end within a specific shift:
  - \( T_0 + \text{shiftnumber}(V) \times \text{SHIFTDURATION} + \text{shiftstart}(V) = \text{start}(V) \)
  - \( T_0 + \text{shiftnumber}(V) \times \text{SHIFTDURATION} + \text{shiftend}(V) = \text{end}(V) \)
- Several consumers \( \{V_1, \ldots, V_n\} \) should start/end in the same shift:
  - \( \text{shiftnumber}(V_1) = \ldots = \text{shiftnumber}(V_n) \) (usage of one common variable is most efficient)
- Several consumers \( \{V_1, \ldots, V_n\} \) should start/end in (pairwise) different shifts:
  - \( \text{allDifferent} (\text{shiftnumber}(V_1), \ldots, \text{shiftnumber}(V_n)) \)
  (N.B.: infeasible if \( n > \text{PERIODS} \) holds)

![Shifts II Diagram]

Cost Calculation I

Consideration of a Time-Variable Tariff
- Energy price depends only on time
- Energy price is independent from used amount of energy

CP-Model: for each task \( A \) an **Element** constraint is generated:

\[ \text{Element}\left(\text{cost}(A), C^A_0, \ldots, C^A_{\text{duration}(A)}, \text{start}(A)\right) \]

where
- \( C^A_i \) are the **costs of task\( A \) if started at time point \( i \).**
  - All these values are computable in advance on the basis of the given tariff
  - Exercise: How to compute these values efficiently over the scheduling horizon?

It holds:

\[ \text{cost}(A) = C^A_{\text{start}(A)} \]

![Cost Calculation I Diagram]
Cost Calculation II

**Total Cost of all Tasks** \( \langle A_1, \ldots, A_m \rangle \):

\[
\sum_{i=1}^{m} \text{cost}(A_i) = \text{totalcost}
\]

**Objective:** minimize the value \( \text{totalcost} \) under consideration of all defined constraints: the CP becomes a COP.

**Pros & Cons** of the approach with an element constraint per each task:
- **Pro:** propagation in both directions – \( \text{start}(A) \leftrightarrow \text{cost}(A) \)
- **Con:** the other tasks of a consumer are not (directly) considered

→ **Integrating Approach:** introduction of redundant Element constraints for consumers consisting of several tasks.

Cost Calculation III

**Extended Model**

- one **Element** constraint for each consumer \( V \) with \( k > 1 \) tasks:

\[
\text{Element}(\text{cost}(V), [C^{V}_{i1}, \ldots, C^{V}_{ik}], \text{duration}(V), \text{start}(V))
\]

Where

- \( C^{V}_{ij} \) are the costs of consumer \( V \) if started at time point \( i \)
- \( C^{V}_{ij} = C^{A_{i1}} + \ldots + C^{A_{ik}} \)
  where \( i_1, \ldots, i_k \) are the start times of the tasks \( A_{i1}, \ldots, A_{ik} \) if \( V \) is started at time point \( i \)

Finally it holds

\[
\sum_{i=1}^{m} \text{cost}(A_i) = \sum_{j=1}^{n} \text{cost}(V_{ij}) = \text{totalcost}
\]
Cost Calculation IV

Further Information on Cost Aware Scheduling


Optimization

In General

- Only the start times of the consumers are labeled → the labeling of the other variables are determined accordingly (shifts, costs, etc.)

Branching & Bounding

- starts with a trivial lower cost bound (how to compute?) and a first solution yielding an upper bound of totalcost
- bounding of totalcost with decreasing values until the smallest value of totalcost is found:
- Search strategy:
  - uses dichotomic bounding (instead of monotonic bounding)
  - uses local consistency of the Sum and redundant Element constraints
  - selects cost(V) such that the one with greatest domain is considered first
  - selects the smallest value of cost(V) first (minimal local cost contribution)
The End

... & Another Tutorial on Constraint-Based Scheduling
http://www.math.unipd.it/~frossi/cp-school/lepape.pdf

Many thanks for your attention!

Contact

Fraunhofer Institute for
Open Communication Systems FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany
www.fokus.fraunhofer.de

Dr. Armin Wolf
Phone: +49 (30) 34 63 -7469
armin.wolf@fokus.fraunhofer.de