ARTIST2

TORSCHE
Scheduling Toolbox for Matlab

Přemysl Šůcha and Michal Kutil
Czech Technical University, Department of Control Engineering
Karlovo náměstí 13, 121 35 Prague 2, Czech Republic
{suchap,kutilm}@fel.cvut.cz
http://dce.felk.cvut.cz/sucha
http://www.tim.cz

Session Outline

- TORSCHE Introduction
- TORSCHE Quick Start
- Iterative Algorithms Scheduling
- Outlook

TORSCHE Introduction

TORSCHE (Time Optimization of Resources, SCHEduling) is Matlab based toolbox for Scheduling.

Aim of the toolbox:

- rapid prototyping of scheduling algorithms
- co-design of control and scheduling problems
- repository of off-line and on-line scheduling algorithms
- open for new algorithms
- demonstration tool for education

TORSCHE – Problem Representation

Maltlab objects are used to represent large variety of scheduling problems.

Main objects of the toolbox:

- ***task*** - parameters of task
- ***taskset*** - object encapsulating set of tasks
- ***problem*** - specification of problem (Błażewicz notation)
- ***graph*** - representation of graph

TORSCHE – Algorithms

The toolbox algorithms structured into several groups.

Groups of the toolbox function:

- functions for manipulation with objects of the toolbox
- scheduling algorithms (List scheduling, EDD, ...)
- graph algorithms (Floyd's algorithm, ...)
- supplementary algorithms (ILP, MIQP, ...)
- GUIs (graphedit, ...)

Session Outline

- TORSCHE Introduction
- ***TORSCHE Quick Start***
- Iterative Algorithms Scheduling
- Outlook

TORSCHE Quick Start

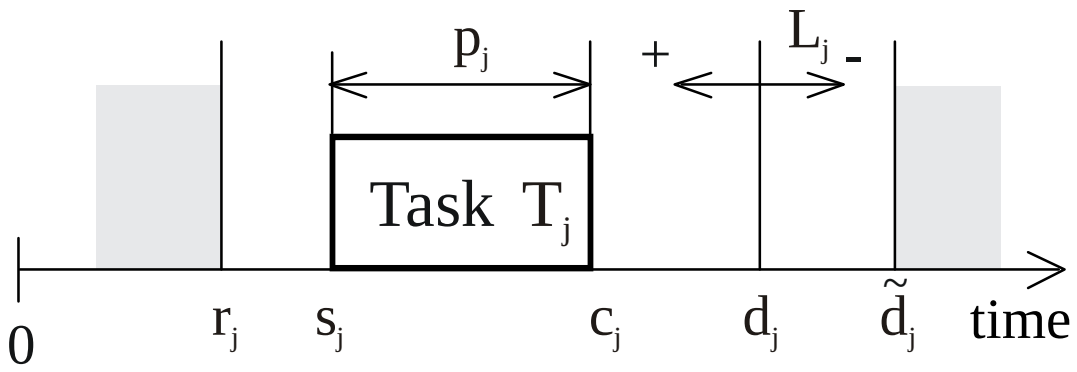
Steps to solve scheduling problems:

1. definition of tasks
2. definition of taskset
3. problem definition
4. scheduling

Example – Horn's algorithm

Problem: 1|pmtn,rj|Lmax

T = { t1, t2, t3}
p = { 5, 2, 3}
r_j = { 1, 0, 5}
d_j = {12, 11, 9}



Objective: minimize maximum lateness $L_{max} = \max\{L_j\}$

Algorithm: Horn's algorithm [Horn74]

Definition of Tasks

Task is defined by command ***task***, for example:

```
>> t1 = task('task1', 5, 1, inf, 12)
Task "task1"
 Processing time: 5
 Release time:    1
 Due date:        12
```

This command defines task with **name** "task1", **processing time** 5, **release time** 1, without **deadline** (inf) and **due date** 12. Other tasks can be defined in the same way:

```
>> t2 = task('task2', 2, 0, inf, 11);
>> t3 = task('task3', 3, 5, inf, 9);
```

Definition of Taskset

Set of tasks is created by command ***taskset*** :

```
>> T = taskset([t1 t2 t3])
Set of 3 tasks
```

For short:

```
>> T = [t1 t2 t3]
Set of 3 tasks
```

Problem Definition

Classification of deterministic scheduling problems:

- notation proposed by [Graham79] and [Błażewicz83]
- special problems, not specified by the notation (e.g. m-DEDICATED)

```
>> p = problem('1|rj,pmtn|Lmax')
1|pmtn,rj|Lmax
```

Scheduling

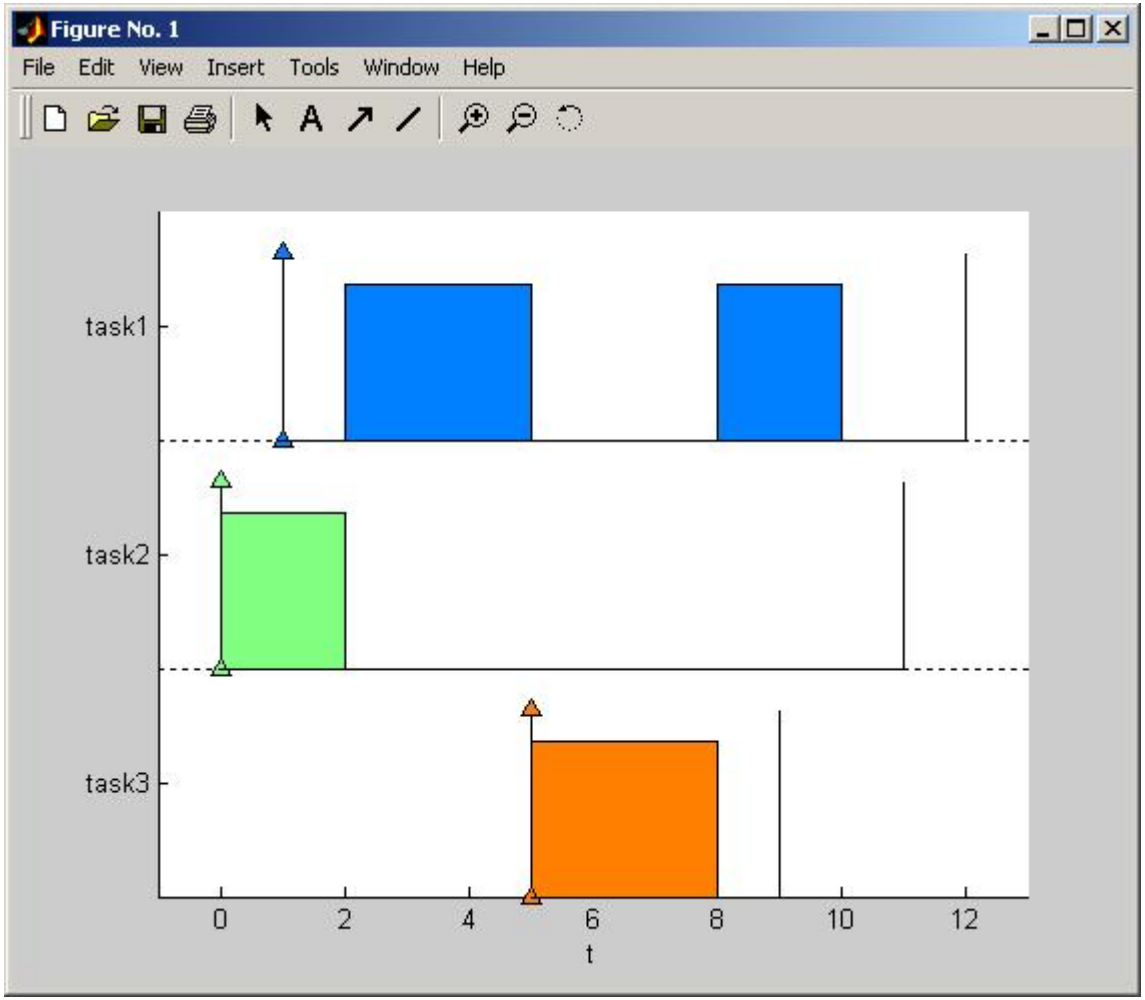
Now we can run the scheduling algorithm, for example Horn's algorithm:

```
>> TS = horn(T,p)
Set of 3 tasks
 There is schedule: Horn's algorithm
   Solving time: 0.016s
```

Graphical representation of the schedule (Gantt chart) can be displayed using command ***plot*** :

```
>> plot(TS,'proc',0)
```

Schedule – Gantt Chart



Session Outline

- TORSCHE Introduction
- TORSCHE Quick Start
- ***Iterative Algorithms Scheduling***
- Outlook

Iterative Algorithms Scheduling

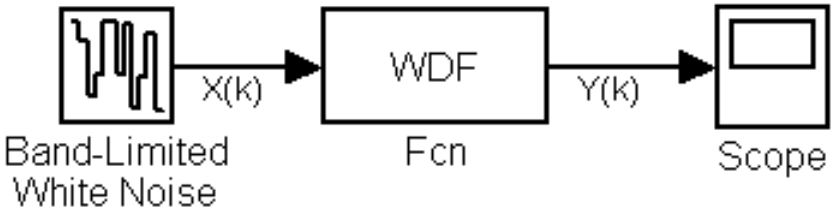
Cyclic scheduling – tasks are repeated in K iterations

Periodic schedule – tasks are repeated periodically with constant period w

Objective – to find a periodic schedule with minimal period w

Example – Cyclic Scheduling

Wave Digital Filter (WDF):




```
for k=1 to N do
   a(k) =X(k) + e(k-1)        %T1
   b(k) = a(k) - g(k-1)       %T2
   c(k) = b(k) + e(k)         %T3
   d(k) = gamma1 * b(k)       %T4
   e(k) = d(k) + e(k-1)       %T5
   f(k) = gamma2 * b(k)       %T6
   g(k) = f(k) + g(k-1)       %T7
   Y(k) = c(k) - g(k)         %T8
end
```

Hardware: one addition and one multiplication unit on a FPGA architecture with floating-point units

floating-point unit	processing time [clk]	latency [clk]
addition (+)	1	1
multiplication (*)	3	3

Problem Statement

Dedicated tasks – tasks are assigned to specified processor (i.e. floating-point unit)

Instance representation: Cyclic Data Flow Graph (CDFG)

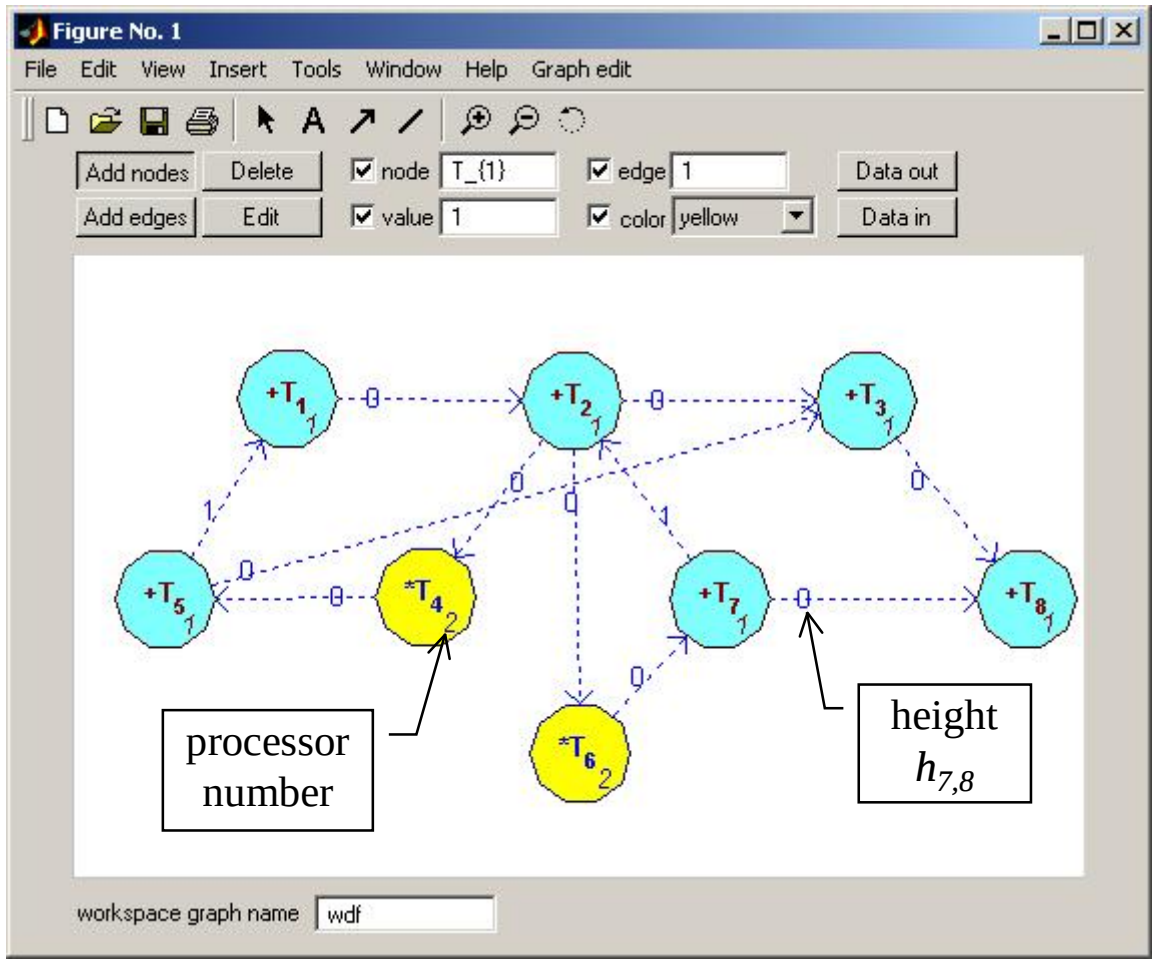
single operation ≈ task ≈ node in the CDFG

precedence constraints between operations ≈ edges weighted by height h_{ij} (dependence distance)

```
>> load('g:\wdf.mat');
>> graphedit(wdf)
```

Cyclic Data Flow Graph of WDF

```
for k=1 to N do
   a(k) =X(k) + e(k-1)  %T1
   b(k) = a(k) - g(k-1) %T2
   c(k) = b(k) + e(k)   %T3
   d(k) = gamma1 * b(k) %T4
   e(k) = d(k) + e(k-1) %T5
   f(k) = gamma2 * b(k) %T6
   g(k) = f(k) + g(k-1) %T7
   Y(k) = c(k) - g(k)   %T8
end
```



Parameters of Processors

Parameters of processors (floating-point units):

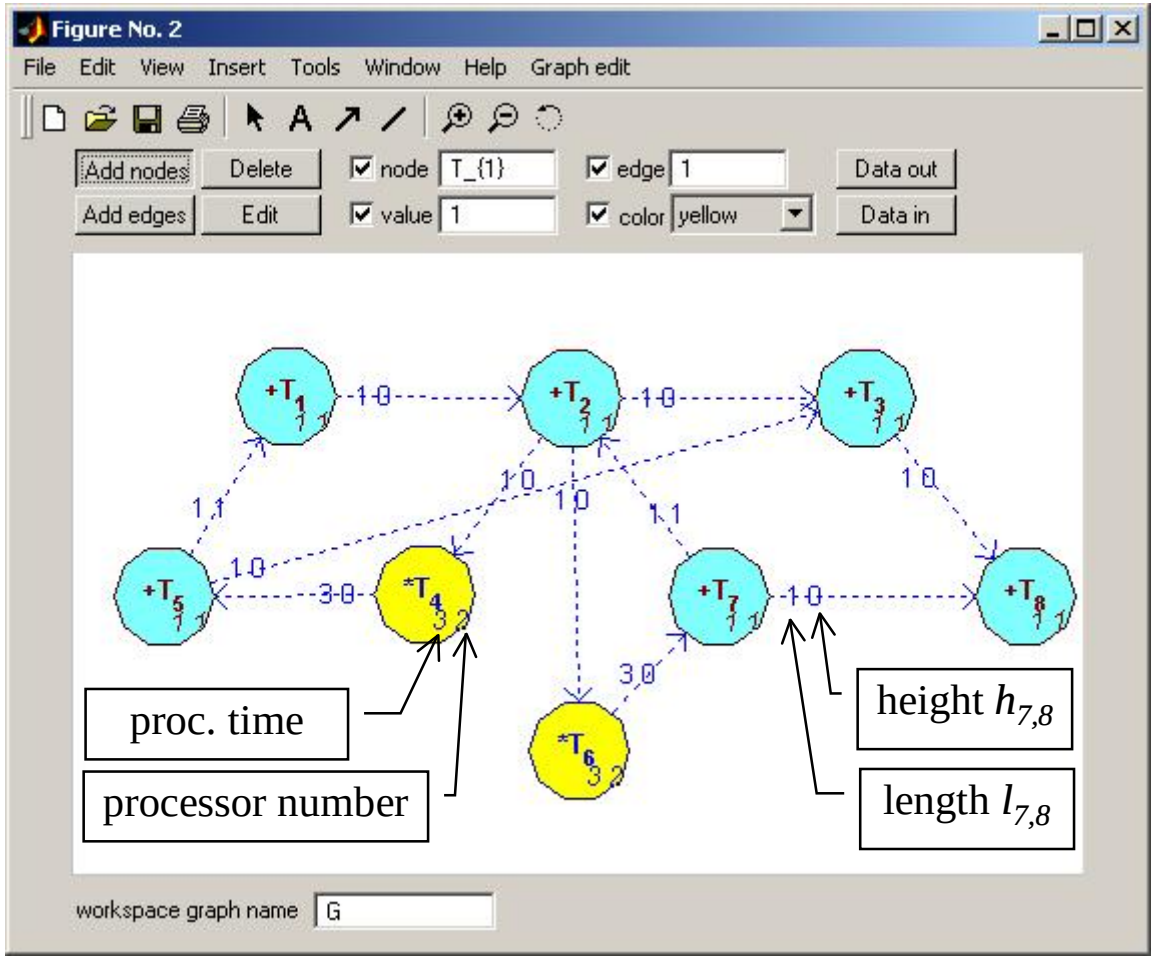
- processing time – processor occupation time
- latency – length l_{ij} (minimal distance between tasks), i.e. processing in pipeline

Scheduling problem is represented by graph G where edges are weighted by couple (l_{ij} , h_{ij}).

```
>> UnitProcTime=[1 3];
>> UnitLattency=[1 3];
>> G = cdfg2LHgraph(wdf,UnitProcTime,UnitLattency);
>> graphedit(G)
```

Note: Floating point units are considered non-pipelined only for simplicity reasons.

Graph *G* of WDF



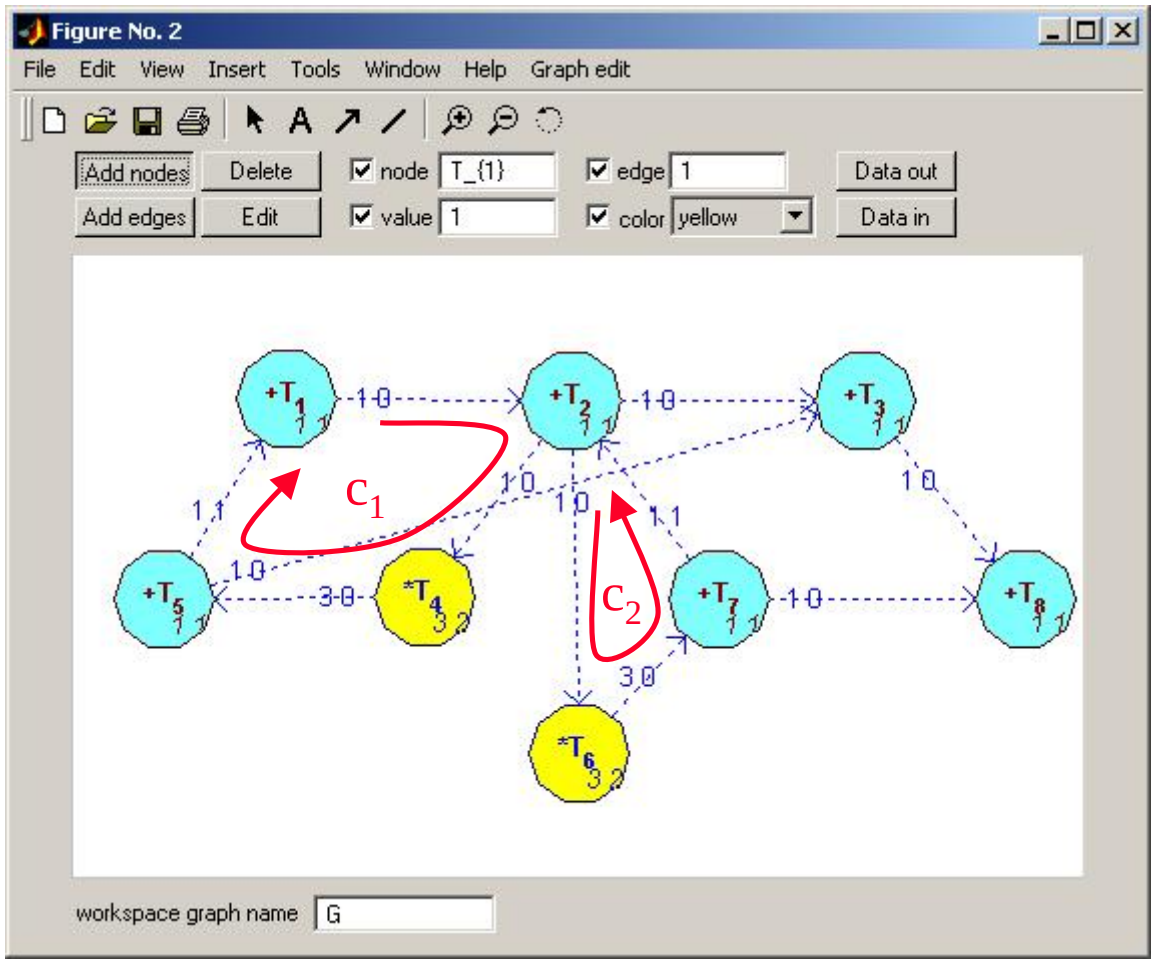
Critical Circuit

- critical circuit in graph G determines minimal feasible period w with respect to precedence constraints
- problem assumes graph G where edges are weighted by a couple of constants length l_{ij} and height h_{ij}
- objective is to find the critical circuit ratio defined as:

$$r = \max_{c \in C(G)} \frac{\sum_{e_{ij} \in c} l_{ij}}{\sum_{e_{ij} \in c} h_{ij}}$$

  where C is a circuit of graph G.
- circuit C of graph G with maximal circuit ratio r is the critical circuit

Critical Circuit - WDF



Critical Circuit Ratio

Graph G contains two circuit c_1 and c_2 with circuit ratio:

$r(c_1) = (1+1+3+1)/(0+0+0+1) = \mathbf{6}$

$r(c_2) = (1+3+1)/(0+0+1) = 5$

Critical circuit is c_1 , therefore period $w \geq 6$. Critical circuit ratio can be evaluated in the toolbox using command:

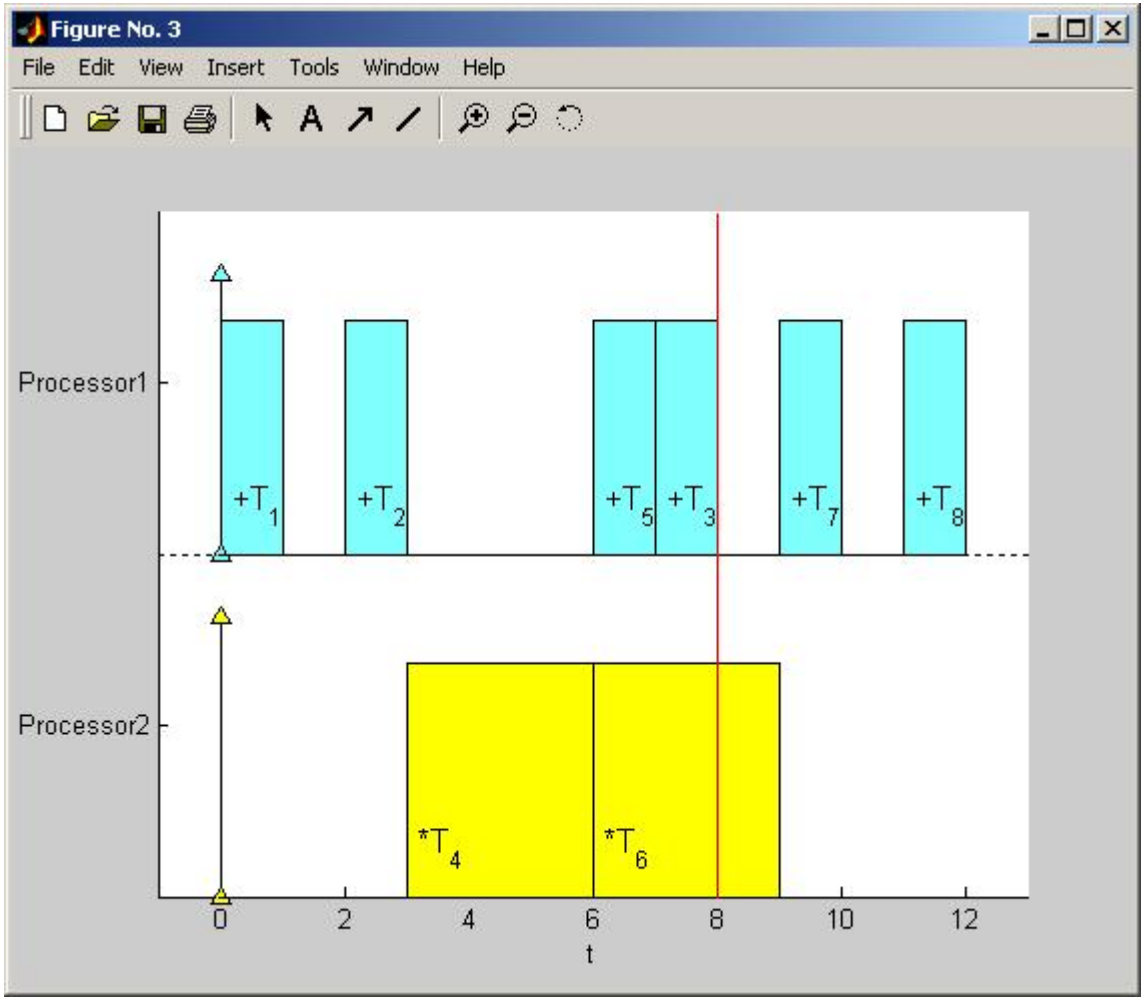
```
>> critical_circuit_ratio(G)
ans =
    6.0000
```

Solution

Graph G can be directly transformed to the taskset:

```
>> T=taskset(G)
Set of 8 tasks
 There are precedence constraints
>> p=problem('m-DEDICATED');
>> schoptions=schoptionsset('ilpSolver','glpk');
>> TS=mdcycsch(T, p, 1, schoptions)
Set of 8 tasks
 There are precedence constraints
 There is schedule: MONOCYCSCH-ILP based algorithm (integer)
   Tasks period: 8
   Solving time: 0.126s
   Number of iterations: 4
>> plot(TS,'prec',0)
```

Resulting Schedule



Solution Summary

```
>> load('g:\wdf.mat');
>> graphedit(wdf)
>> UnitProcTime=[1 3];
>> UnitLattency=[1 3];
>> G = cdfg2LHgraph(wdf,UnitProcTime,UnitLattency);
>> T=taskset(G);
>> p=problem('m-DEDICATED');
>> schoptions=schoptionsset('ilpSolver','glpk');
>> TS=mdcycsch(T, p, 1, schoptions);
>> plot(TS,'prec',0);
```

Session Outline

- TORSCHE Introduction
- TORSCHE Quick Start
- Iterative Algorithms Scheduling
- ***Outlook***

Outlook

Currently we are working on:

- automatic code generation for Handel C and TrueTime
- real-time schedulability analysis
- new graph and optimization algorithms

More Materials

TORSCHE Scheduling Toolbox for Matlab with a complete documentation can be downloaded at:

http://rtime.felk.cvut.cz/scheduling-toolbox/