Introduction to Combinatorial Optimization

Zdeněk Hanzálek
hanzalek@fel.cvut.cz
Acknowledgments to: Přemysl Šůcha, Jan Kelbel
Jiří Trdlička, Michal Kutil, Zdeněk Baumelt, Roman Čapek

CTU FEE Department of Control Engineering

February 12, 2013

Table of Contents

1. Course organization
2. Application examples
3. Graph Theory Overview

Lectures

1. Introduction of Basic Terms, Example Applications.
2. Integer Linear Programming - Algorithms.
3. Problem Formulation by Integer Linear Programming.
4. Shortest Paths. Test I.
5. Flows and Cuts - Problem Formulation and Algs. Bipartite Matching.
6. Multicommodity Network Flows.
7. Knapsack Problem, Pseudo-polynomial and Approximation Algs.
8. Traveling Salesman Problem and Approximation Algorithms. Test II.
9. Monoprocessor Scheduling.
10. Scheduling on Parallel Processors.
11. Project Scheduling with Time Windows.
12. Constraint Programming.

1 - motivation
4,5,6 - mostly polynomial complexity
7,8,9,10,11 - NP-hard problems
2,3,12 - declarative programming techniques

Seminars

1. Introduction to the Experimental Environment and Optimization Library
2. Integer Linear Programming
3. Applications of Integer Linear Programming
4. Individual Project I - Assignment and Problem Classification
5. Shortest Paths
6. Individual Project II - Related Work and Solution
7. Applications of Network Flows and Cuts
8. Individual Project III - Consultation
9. Scheduling
10. Test III
11. Individual Project IV - presentation of code, results and written report
12. Credits

1,2,3,5,7,9 - exercises 1-6

4,6,8,11 - individual project - consultation and reporting

10 - test III - programming exercise to be finished in limited time

Course Organization

A4M35KO site: https://moodle.dce.fel.cvut.cz

- Courses
- Seminars
- Project
- Classification
- Literature

What is Combinatorial Optimization?

Optimization is a term for the mathematical discipline that is concerned with the minimization/maximization of some objective function subject to constraints or decision that no solution exists.

Combinatorics is the mathematics of discretely structured problems.

Combinatorial optimization is an optimization that deals with discrete variables.

It is very similar to **operation research** (a term used mainly by economists, originated during WW II in military logistics).

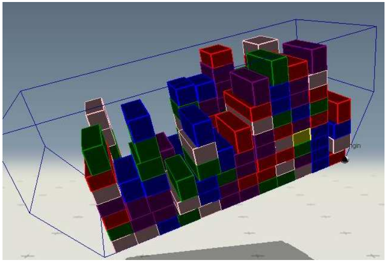
Application Areas

Many real-life problems can be formulated as combinatorial optimization problems.

Typical application areas:

- Production (production speed up, cost reduction, efficient utilization of resources...)
- Transportation (fuel saving, reduction of delivery time...)
- Employees scheduling (reduction of human resources...)
- Hardware design (acceleration of computations...)
- Communication network design (end-to-end delay reduction...)
- ...

Container Loading



Goal:

- To store as much cargo as possible in a container.

Constraints:

- size of the container
- sizes of the boxes
- loading process
- stability, orientation of the boxes
- requested order of the boxes when unloading the cargo

Can be formalized as a **3-D knapsack**.

Scheduling of Human Resources

Assignment of shifts to employees

Goal:

- create acceptable shift schedule,
  so that all required shifts are assigned

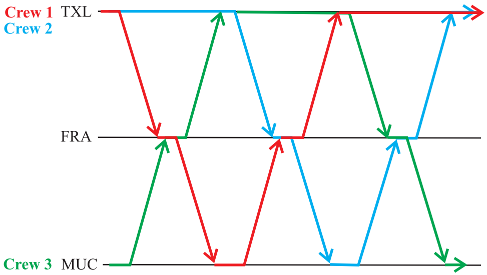
Constraints:

- qualification of employees
- labor code restrictions (e.g. at least 12 hours of rest during 24 hours)
- collective agreement restrictions (e.g. maximum number of night shifts in a block)
- employees demands (e.g. required day-off)
- fair assignment of shifts (same number of weekend shifts)

Scheduling of Human Resources

Scheduling of human resources is often formalized as a **matching in a bipartite graph**

The problem becomes harder when we consider the geographic position of the employees (e.g. stewards and pilots in an airline company).



Storage System in Automated Warehouse

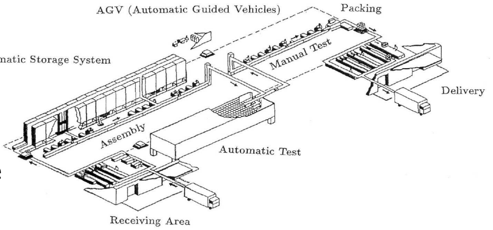
Automated warehouse is used as a in-process store connecting parts of the factory. When overloaded, it can become a bottleneck.

Goal:

- reduce the length of vehicle trips

Constraints:

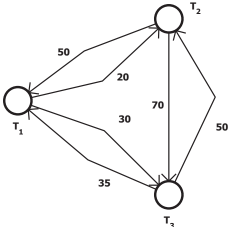
- given tasks
- warehouse parameters
- vehicle can transport 1 *container* in the given time
- tasks are added on-line



Storage System in Automated Warehouse

The task is determined by the start position and destination position.
The problem is represented by the directed graph.
The nodes of the graph represent the transport tasks.
Edge (i,j) means the possibility to perform task j right after i.
Edge cost represents the cost of the trip from i to j.

Can be formulated as
Asymmetric Traveling Salesman Problem.



Vehicle Routing

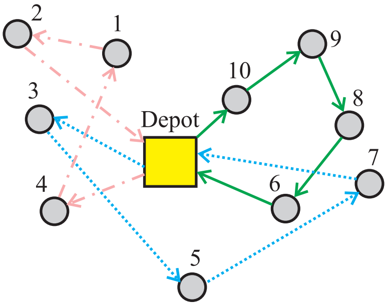
Goods, customers and fleet of cars.

Goal:

- fulfill demands of customers
- minimize transportation cost

Constraints:

- payload capacity
- time windows
- traffic jams
- shifts, breaks



Surface Mount Technology

The placement machines are scarce resource of the Printed Circuit Board (PCB) production, due to their high cost.

Goal:

- Maximize production speed

Constraints:

- Assembly line configuration
- Description of produced PCB

Problem can be divided into two subproblems.

Surface Mount Technology

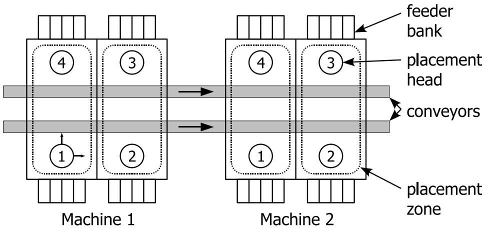
A) Allocation of the components to the placement heads
Can be formulated as a **Partition Problem**

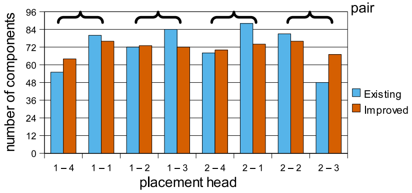
Input:

- types of SMT components
- number of components of a given type
- precedence relations among some components
- machine parameters

Output:

- allocation of components to the placement heads





Surface Mount Technology

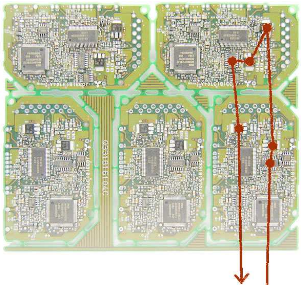
B) Sequencing for a given head can be formulated as a (capacitated multi-trip) **Traveling Salesman Problem**

Input:

- allocation of components to the assembly head
- position of components on the PCB

Output:

- assembly sequence
- estimation of operation time



Steel Mill Slab Design Problem

A steel plant produces slabs which are later divided and processed into final products

Goal:

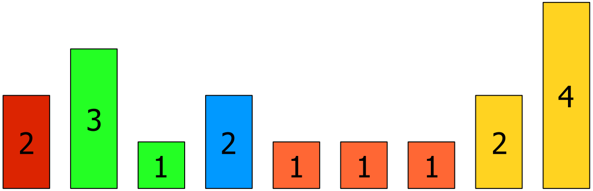
- Minimize the amount of the steel slabs needed for realization of all jobs

Constraints:

- n different sizes of slabs
- size of each job
- color of job determines processing of a slab
- at most p jobs of different colors can be made from one casting

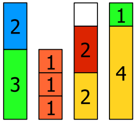
Steel Mill Slab Design Problem

Example



- Slab sizes: $\{3, 5\}$, $n = 2$
- Jobss $1 \dots 9$
- Colors $1 \dots 5$
- Maximum number of colors on each slab $p = 2$

Result:



Routing in Wireless Sensor Network - Specification

Vast area monitoring using autonomous devices equipped with:

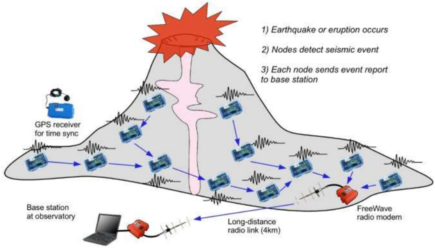
- own power supply
- wireless short range communication
- temperature sensors

Goal:

- create routing tables
- minimize energy consumption

Constraints:

- capacity of each communication link
- limited transmitter performance
- maximal allowed end-to-end delay of communication
- memory capacity of devices


1) Earthquake or eruption occurs
2) Nodes detect seismic event
3) Each node sends event report to base station
GPS receiver for time sync
Base station at observatory
Long-distance radio link (4km)
FreeWave radio modem

Routing in Wireless Sensor Network - Formalization

Can be formalized as a **Multi-Commodity Network Flows problem**.

Network represented by a graph (devices = vertices, links = edges)

Constant communication delay for all links (TDMA period,...)

Communication demand = commodity flow:

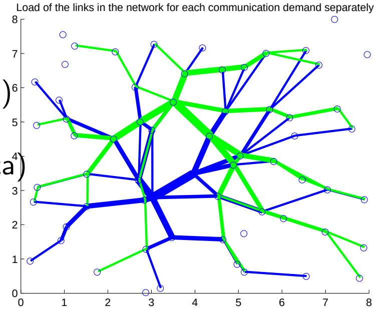
- source devices and destination devices
- volume of demand (quantity of data per time unit)
- deadline (maximum number of hops)

Communication link:

- capacity (quantity of data per time unit)
  - relates to the number of TDMA slots)
- price (energy to transfer one unit of data)

Other variants:

- various delay on links
- indivisible flows
- maximize the network lifetime (minimize energy consumption)
- distributed version



Routing in Wireless Sensor Network - Distributed Problem

In this application, the centralized approach is not useful:

- inputs (link capacity,...) and outputs (volume of flow in each link) have a local nature
- adding/removing of a device in centralized algorithm needs:
  - communication of inputs
  - centralized computation
  - communication of outputs
  - switching of network configuration

Routing in Wireless Sensor Network - Distributed Problem

In this application, the centralized approach is not useful:

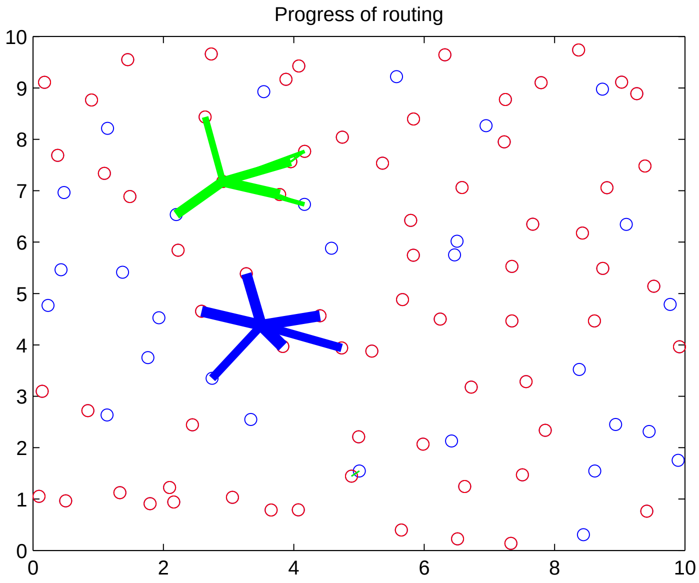
- inputs (link capacity,...) and outputs (volume of flow in each link) have a local nature
- adding/removing of a device in centralized algorithm needs:
  - communication of inputs
  - centralized computation
  - communication of outputs
  - switching of network configuration

Distributed algorithm - same code in each device:

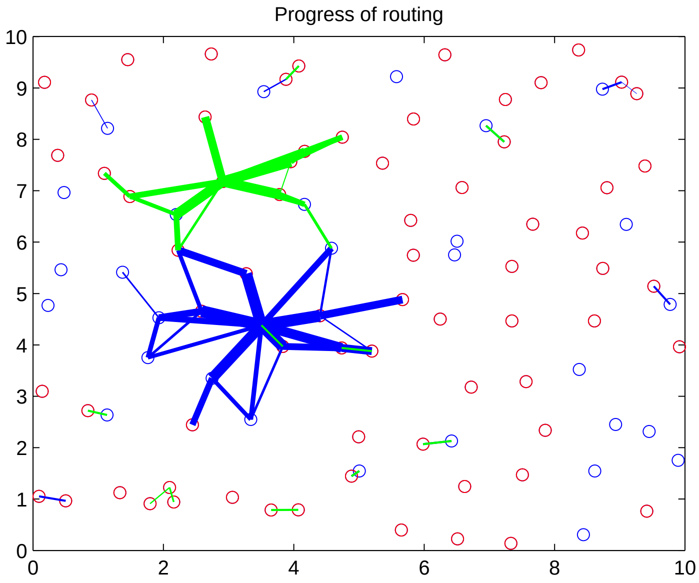
```
Input: capacities and prices of incident links, source and sink nodes,...
Output: volume of flow on incident links,...
while consensus_not_reached do
    do local optimization;
    communicate border variables with neighbors;
end
```

Problems in distributed algorithms - convergence and termination

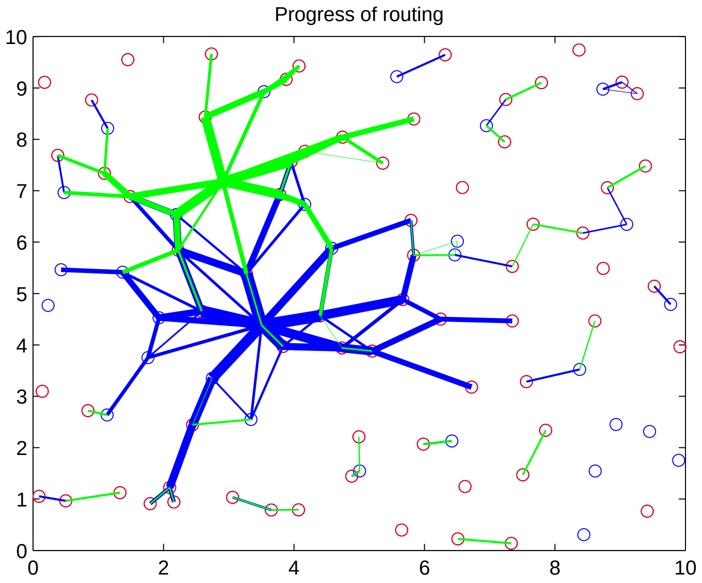
Routing in Wireless Sensor Network



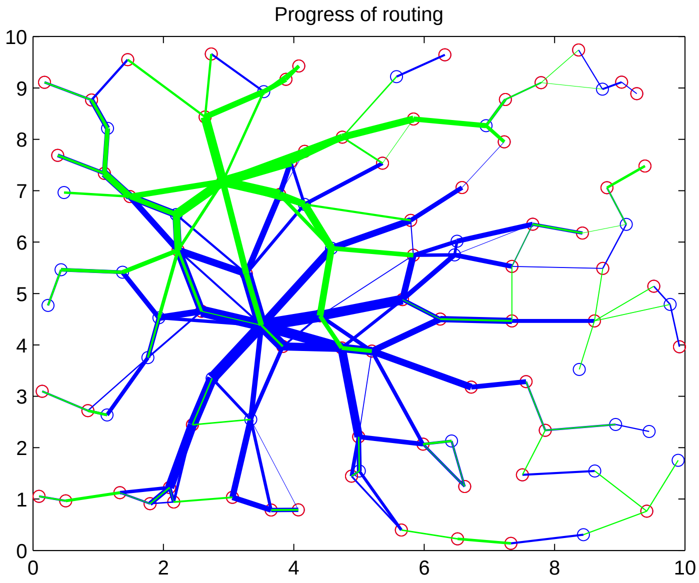
Routing in Wireless Sensor Network



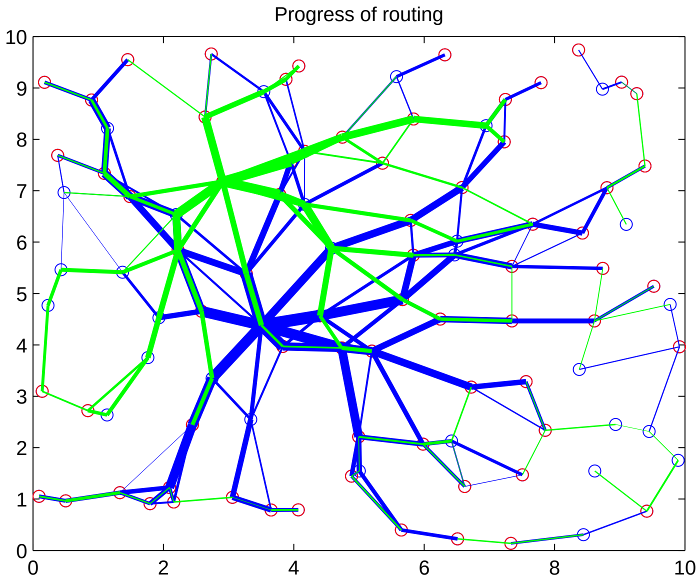
Routing in Wireless Sensor Network



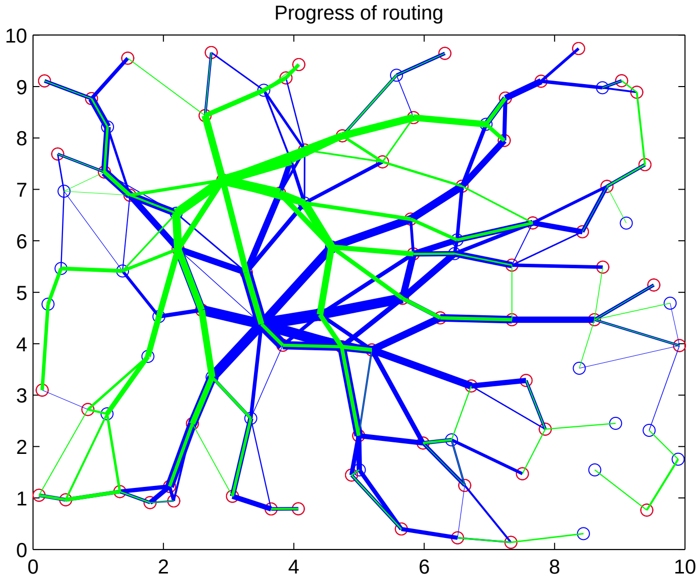
Routing in Wireless Sensor Network



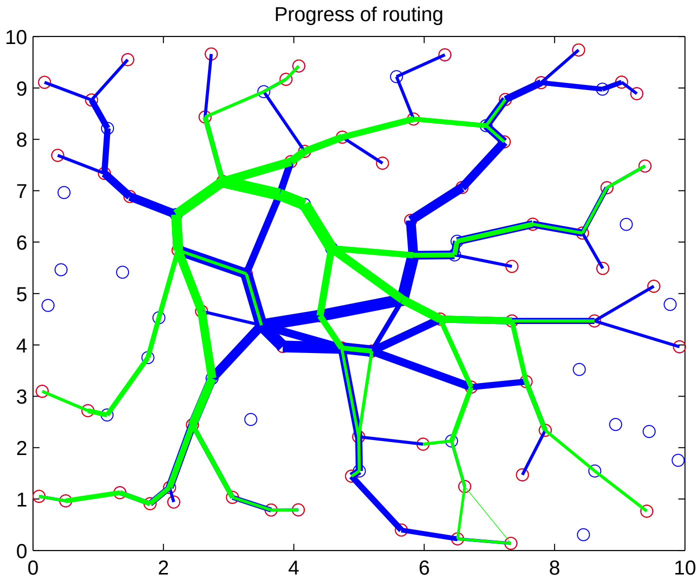
Routing in Wireless Sensor Network



Routing in Wireless Sensor Network



Routing in Wireless Sensor Network



Crossroads Design - Specification

A city is represented by crossroads and streets. Traffic flows are determined by sources, destinations and the number of cars per time unit.

Goal:

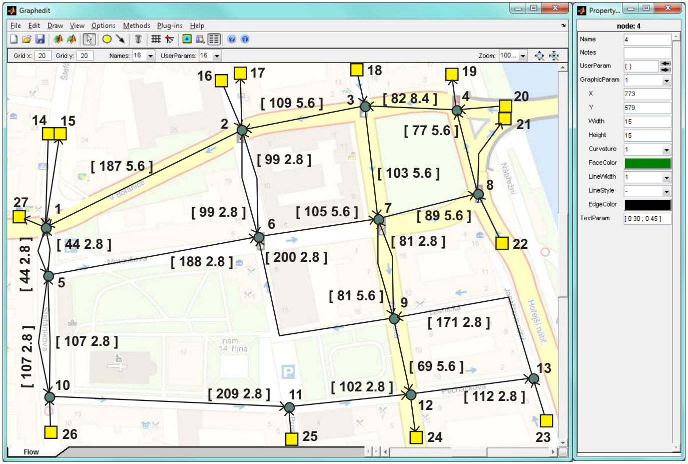
- For every crossroad find out a volume of the flow in left/straight/right direction = routing (from such information, we can determine the capacities of lanes and setting of traffic lights)
- We assume optimal behavior of the system = minimization of the transportation cost (i.e. we minimize traveled distance or fuel consumption or transportation time).

Constraints:

- road capacities (determined by the number of lanes and speed limits)
- one-way roads

Can be formalized as a **Multi-Commodity Network Flows Problem**

Crossroads Design - Specification



Grid x: 20 Grid y: 20 Names: 16 UserParams: 16 Zoom: 100...

node: 4

Name	4
Notes	
UserParam	{ }
GraphicParam	1
X	773
Y	579
Width	15
Height	15
Curvature	1
FaceColor	🟩
LineWidth	1
LineStyle	-
EdgeColor	⬛
TextParam	[0 30 ; 0 45]

source	14	14	14	14	16	16	16	16	18
destination	24	17	21	27	24	21	15	27	24
cars per minute	4.9	1.9	3.1	30.1	10.1	12.1	1.2	6.6	32.7

Lacquer Production Scheduling

Made-to-order lacquer production, where jobs are determined by type of lacquer, quantity and delivery date.

Goal:

- minimize tardiness (delivery date overrun)
- minimize storage costs

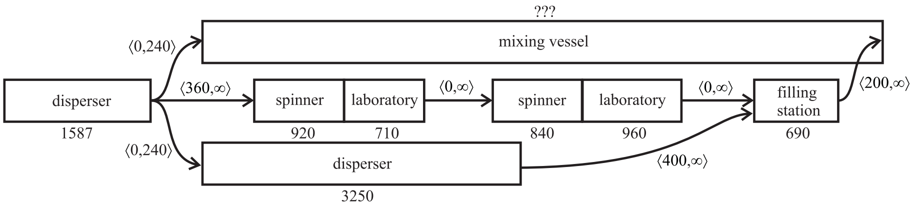
Constraints:

- batch production of various kinds of lacquer
- varying production process/time for different kinds
- time constraints between start times and/or completion times of operations
- working hours (processing times of some operations exceed working hours)
- preparation (*set-up time*)

Can be formalized as $PS\,|temp, o_{ij}, tg|\,C_{max}$

Lacquer Production Scheduling

There can be time constraints on operations. We must consider:
(1) minimal delay between the end of one operation and the start of the next one (e.g. minimal delay needed to dissolve an ingredient into the lacquer)
(2) maximal delay between the end of one operation and the start of the next one (e.g. the lacquer can solidify).

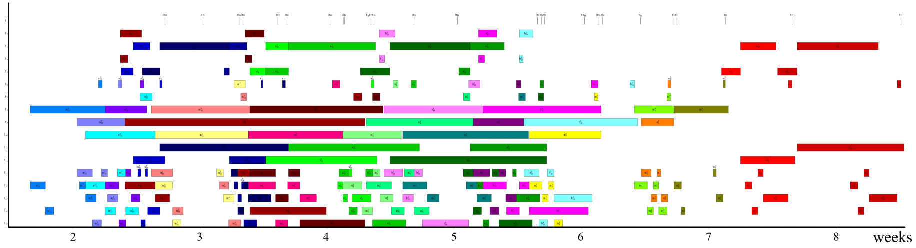


Plus, we have take into account that the processing time on some resources (reservoirs) depends on the start or completion of different operations.

Lacquer Production Scheduling

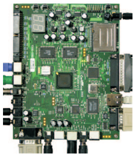
Example

- production of 29 jobs
- 3 types of lacquer
- 9 weeks time horizon



Configurable HW for Specific DSP Application Design

Special computational units (adder, multiplier) on FPGA.
Iterative Digital Signal Processing (DSP) algorithm consists of atomic operations executed on these units.



Goal:

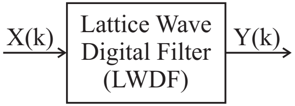
- maximize computational speed
- minimize amount of used gates and interconnects

Constraints:

- precedence relations between atomic operations (addition, multiplication...)
- limited access to the circuit memory
- limited amount of circuit memory
- number of available processing units on FPGA

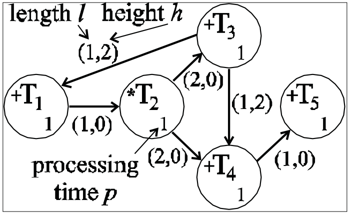
Configurable HW for Specific DSP Application Design

An example application is a simple digital filter LWDF. $X(k)$ are samples of input signal, $Y(k)$ are samples of output signal.



```
for k=1 to N do
    T_1:  $a(k) = X(k) - c(k-2)$
    T_2:  $b(k) = a(k) * \alpha$
    T_3:  $c(k) = b(k) + X(k)$
    T_4:  $d(k) = b(k) + c(k-2)$
    T_5:  $Y(k) = X(k-1) + d(k)$
end
```

LWDF filter algorithm



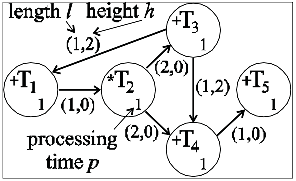
Atomic operation dependencies

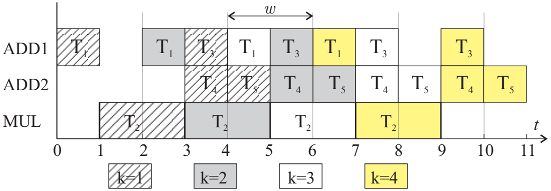
Configurable HW for Specific DSP Application Design

Can be formalized as a cyclic extension of $PS\,|temp|\,C_{max}$

Used hardware features

unit	count $[-]$	computation time $[clk]$
ADD	2	1
MUL	1	2



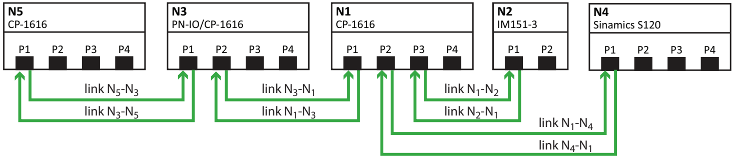


Message Scheduler for Profinet IO IRT - Specification

Profinet IO IRT is an Ethernet-based hard-real time communication protocol, which uses static schedules for time-critical data. Each node contains a special hardware switch that intentionally breaks the standard forwarding rules for a specified part of the period to ensure that no queuing delays occur for time-critical data.

Goal:

- Find the shortest makespan (length of the schedule) for time critical messages.



line	$N_1 \to N_3$	$N_1 \to N_4$	$N_1 \to N_2$	$N_2 \to N_1$	$N_3 \to N_1$	$N_4 \to N_1$	$N_3 \to N_5$	$N_5 \to N_3$
line delay [ns]	4875	5130	5862	3841	4875	4895	4875	4875

Message Scheduler for Profinet IO IRT - Specification

Constraints:

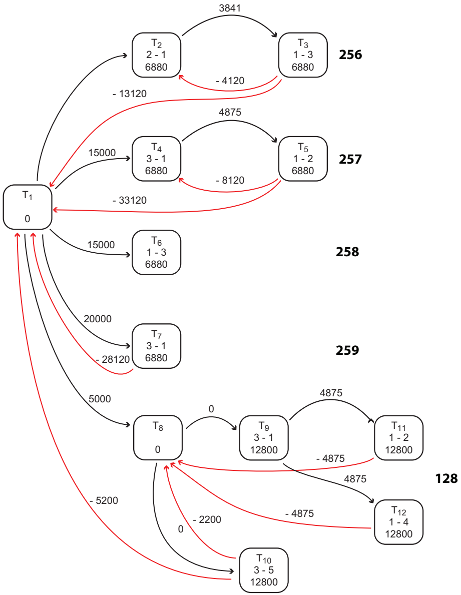
- tree topology $\Rightarrow$ fixed routing
- release date r - earliest time the message can be sent
- deadline $\widetilde{d}$ - latest time the message can be delivered
- maximal allowed end-to-end time delay

ID	source $\rightarrow$ target	length [ns]	r [ns]	$\widetilde{d}$ [ns]	end2end delay [ns]
256	$N_2 \rightarrow N_3$	5760	5000	20000	11000
257	$N_3 \rightarrow N_2$	5760	15000	40000	15000
258	$N_1 \rightarrow N_3$	5760	15000	–	–
259	$N_3 \rightarrow N_1$	5760	20000	35000	–
128	$N_3 \rightarrow \{N_1,N_2,N_4,N_5\}$	11680	5000	{–,–,–,18000}	{–,17675,17675,15000}

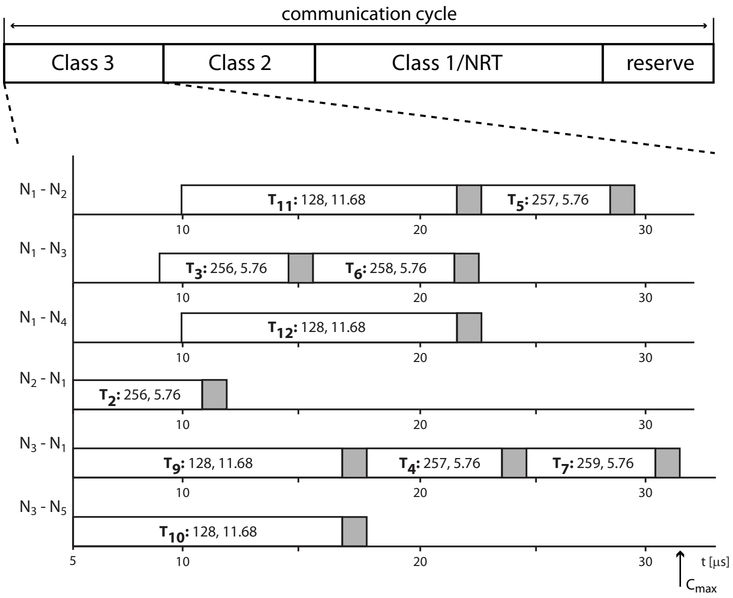
Message Scheduler for Profinet IO IRT - Formalization

Can be formulated as $PS\,|temp|\,C_{max}$ problem.

- task = message on a given line
- positive cost edge = r, precedence relations
- negative cost edge = $\widetilde{d}$, end-to-end delay
- unicast message = chain of tasks (assuming positive edges)
- multicast message = out-tree of tasks (assuming positive edges)



Message Scheduler for Profinet IO IRT - Result



Convincing Arguments

Explain the typical goals of optimization:

- increase the volume of the production (shorter production-line cycle)
- cost reduction (fuel saving, less machines)
- risk reduction (error elimination due to automated creation of production schedule)
- lean manufacturing (supply and stores reduction, outgrowths reduction when delay in supply)
- increase of the flexibility (faster reaction to structure or constraint change)
- user-friendly solutions (balanced schedule for all employees)

How does this course cover the important skills?

An engineer is usually hired to systematically solve the problem.

Skill	Example
Business case	Attract a customer
Specification	Production scheduling
Formalization	Flow-shop
Algorithms	Johnson's algorithm
Prototype solution	Matlab OPL, ...
Implementation	C#, dB, ...

How does this course cover the important skills?

An engineer is usually hired to systematically solve the problem.

Skill	Example	Lectures
Business case	Attract a customer	-
Specification	Production scheduling	Application examples
Formalization	Flow-shop	Formulation of the opt. problem
Algorithms	Johnson's algorithm	Pseudocode iteration with data
Prototype solution	Matlab OPL, ...	-
Implementation	C#, dB, ...	-

How does this course cover the important skills?

An engineer is usually hired to systematically solve the problem.

Skill	Example	Lectures	Seminars
Business case	Attract a customer	-	Project?
Specification	Production scheduling	Application examples	Project Exercises
Formalization	Flow-shop	Formulation of the opt. problem	Project Exercises
Algorithms	Johnson's algorithm	Pseudocode iteration with data	Project
Prototype solution	Matlab OPL, ...	-	Project Exercises
Implementation	C#, dB, ...	-	Project?

How does this course cover the important skills?

An engineer is usually hired to systematically solve the problem.

Skill	Example	Lectures	Seminars	Exam
Business case	Attract a customer	-	Project?	-
Specification	Production scheduling	Application examples	Project Exercises	Project
Formalization	Flow-shop	Formulation of the opt. problem	Project Exercises	Project Exam
Algorithms	Johnson's algorithm	Pseudocode iteration with data	Project	Test I, II Exam
Prototype solution	Matlab OPL, ...	-	Project Exercises	Project Test III
Implementation	C#, dB, ...	-	Project?	-

Algorithms are used to solve the problems

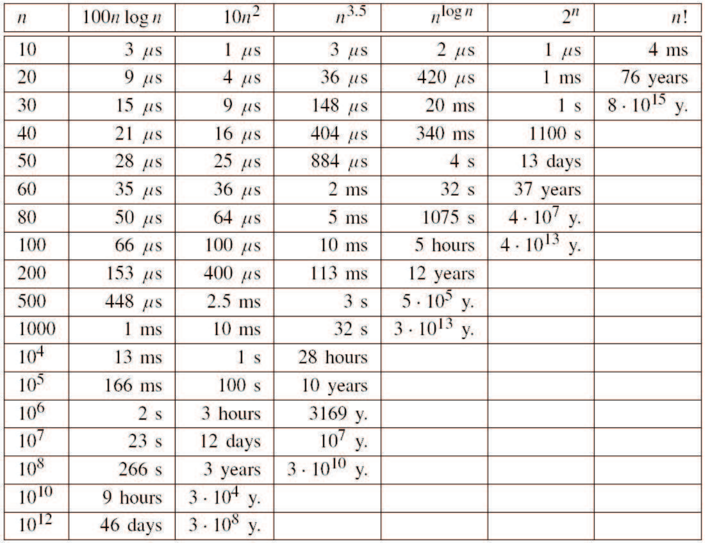
Combinatorial optimization uses combinatorial algorithms

Many problems can be formalized by:

- constraints
- optimization criterion

It is not always easy to find the optimal solution efficiently.
In the case of exhaustive search while enumerating all solutions, the computation time for bigger instances can be enormous.
For example, the permutation Flow-shop problem has complexity of $n!$, where n is the number of jobs.

Time Complexity of Algorithms



n	$100n \log n$	$10n^2$	$n^{3.5}$	$n^{\log n}$	2^n	$n!$
10	3 μs	1 μs	3 μs	2 μs	1 μs	4 ms
20	9 μs	4 μs	36 μs	420 μs	1 ms	76 years
30	15 μs	9 μs	148 μs	20 ms	1 s	$8 \cdot 10^{15}$ y.
40	21 μs	16 μs	404 μs	340 ms	1100 s	
50	28 μs	25 μs	884 μs	4 s	13 days	
60	35 μs	36 μs	2 ms	32 s	37 years	
80	50 μs	64 μs	5 ms	1075 s	$4 \cdot 10^7$ y.	
100	66 μs	100 μs	10 ms	5 hours	$4 \cdot 10^{13}$ y.	
200	153 μs	400 μs	113 ms	12 years		
500	448 μs	2.5 ms	3 s	$5 \cdot 10^5$ y.		
1000	1 ms	10 ms	32 s	$3 \cdot 10^{13}$ y.		
10^4	13 ms	1 s	28 hours			
10^5	166 ms	100 s	10 years			
10^6	2 s	3 hours	3169 y.			
10^7	23 s	12 days	10^7 y.			
10^8	266 s	3 years	$3 \cdot 10^{10}$ y.			
10^{10}	9 hours	$3 \cdot 10^4$ y.				
10^{12}	46 days	$3 \cdot 10^8$ y.				

Graph Theory Overview

Graphs informally:

- A graph consists of nodes and edges.
- Each edge joins two nodes, it is directed or undirected.
- In a directed graph, the edge leaves one node and enters another one.
- In an undirected graph, an edge is a symmetric join of two nodes.

Directed Graph

Directed graph (digraph) is a triplet (V, E, Ψ):

- V is a finite set of **nodes**
- E is a finite set of **directed edges**
- Ψ is a mapping from the set of edges to the ordered pair of nodes, i.e. $\Psi : E \to \{(v, w) \in V \times V : v \neq w\}$

Undirected Graph

Undirected graph is a triplet (V, E, Ψ):

- V is a finite set of nodes.
- E is a finite set of **undirected edges**
- Ψ is a projection from the set of edges to the 2-element subset of V, i.e. $\Psi : E \rightarrow \{X \subseteq V : |X| = 2\}$

Edges

- Two edges e, e' are called **parallel edges** if $\Psi(e) = \Psi(e')$.
- A graph containing parallel edges is called a **multigraph** - we usually do not work with a multigraph.
- A graph that does not contain parallel edges is called **simple graph**.
- We usually denote simple graphs by pair $G = (V(G), E(G))$, where $V(G)$ is a set of nodes and $E(G)$ is set of (ordered) pairs of nodes that describes the edges (i.e. we identify an edge with its image $\Psi(e)$).
- Undirected edge $e = \{v, w\}$ or directed edge $e = (v, w)$ connects v and w. Nodes v and w are the endpoints of e or we can say they are incident with e. In case of a directed graph we say that e leaves v and enters w.

Comparison of Graphs

For directed graph G we can have an **underlaying undirected graph** G', with the same set of vertices and undirected edge $\{v, w\}$ for every directed edge (v, w) from G.
We call graph H a **subgraph** of graph G, if we can create it by omitting the nodes (zero or more of them) or edges (when an edge is in the subgraph, it's endpoints must be there as well). Special cases of subgraphs:

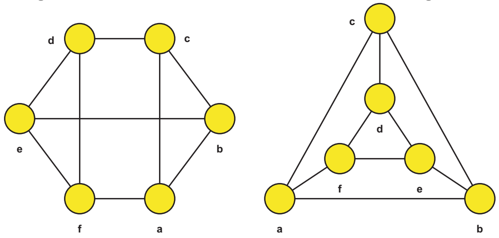
- Subgraph H of G is called **spanning** if we omit some or zero edges and $V(G) = V(H)$.
- Graph H is called **subgraph induced by set of vertices** $V(H) \subseteq V(G)$ if we omit some (or zero) nodes and incident edges, i.e. H contains all edges from G whose both endpoints are in $V(H)$.

Comparison of Graphs – Isomorphic Graphs

Two graphs G and H are called **isomorphic** if there are bijections: $\Phi_V : V(G) \rightarrow V(H)$ and $\Phi_E : E(G) \rightarrow E(H)$ such that:

- for directed graphs: $\Phi_E((v,w)) = (\Phi_V(v), \Phi_V(w))$ for all $(v,w) \in E(G)$
- for undirected graphs: $\Phi_E(\{v,w\}) = \{\Phi_V(v), \Phi_V(w)\}$ for all $\{v,w\} \in E(G)$

We usually don't deal with isomorphic graphs when talking about algorihms, but it is good to be aware of them during modelling.



Other Terms for Digraphs

For node v of digraph G we define:

- a set of **successors** of v is a set of nodes such that there is an edge from v to each of these nodes, i.e. $\{w \in V : (v, w) \in \Psi(E)\}$
- a set of **predecessors** v is a set of nodes such that there is an edge from each of these nodes to v, i.e. $\{w \in V : (w, v) \in \Psi(E)\}$
- $\Gamma(v)$, a set of **neighbors** of v, is set of nodes connected by an edge with v, i.e. a union of successors and predecessors
- $\delta^+(v)$, a set of edges leaving v
- $\delta^-(v)$, a set of edges entering v
- $\delta(v)$, a set of edges incident with v
- $|\delta^+(v)|$, **out-degree**
- $|\delta^-(v)|$, **in-degree**
- $|\delta(v)|$, **degree of node**

Notice: $\sum_{v \in V(G)} |\delta(v)| = 2\,|E(G)|$.
the number of nodes with an odd degree is even

Other Terms for Digraphs

For sets $X, Y \subseteq V(G)$ of directed graph G we define:

- $E^+(X,Y)$ a set of edges from X to Y, i.e. $E^+(X,Y) = \{(x,y) \in E(G) : x \in X \setminus Y, y \in Y \setminus X\}$
- $\delta^+(X)$, a set of edges leaving set X, i.e. $\delta^+(X) := E^+(X, V(G) \setminus X)$
- $\delta^-(X)$, a set of edges entering set X, i.e. $\delta^-(X) := \delta^+(V(G) \setminus X)$
- $\delta(X)$, a set of "border" edges of set X, i.e. $\delta(X) := \delta^+(X) \cup \delta^-(X)$
- $\Gamma(X)$, a set of neighbor nodes of set X, i.e. $\Gamma(X) := \{v \in V(G) \setminus X :$ "border" edge $e \in \delta(X)$ exists and it is incident with node $v\}$

Similar Terms for an Undirected Graph

These terms are used in undirected graphs:

- $\Gamma(v)$, a set of neighbors of node v
- $\delta(v)$, a set of edges incident with v
- $|\delta(v)|$, a degree of v
- $E(X, Y)$, a set of edges between X and Y, i.e. $E(X, Y) = \{\{x, y\} \in E(G) : x \in X \setminus Y, y \in Y \setminus X\}$
- $\delta(X)$, a set of "border" edges of set X, i.e. $\delta(X) := E(X, V(G) \setminus X)$
- $\Gamma(X)$, set of neighbours of set X, i.e. $\Gamma(X) := \{v \in V(G) \setminus X : E(X, v) \neq \emptyset\}$

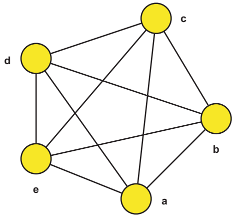
Mostly, we will use digraphs (from an undirected graph we can make a digraph by replacing every undirected edge by a pair of inverse directed edges).

Special Graphs

A complete digraph is (a simple) graph $G = (V, E)$, where E is a set of all possible pairs of different nodes of V.

A complete undirected graph is (a simple) graph, in which every pair of vertices is connected by a unique edge. We denote it K_n, where n is the number of nodes.

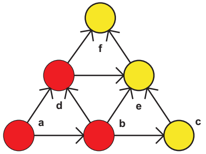
A graph is called **regular** if all its nodes have the same degree. If the degree of all nodes is k, the graph is called **k-regular**.

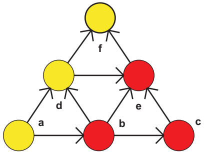


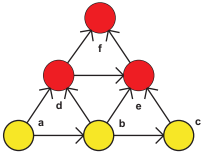
Special Graphs

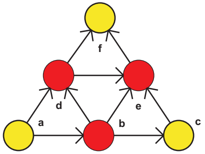
A complement of simple graph G is simple graph H such that $G + H$ is the complete graph. A pair of nodes in the complement is connected if it is not connected in G.

A clique is a subgraph that is complete. The number of nodes in the maximum (biggest) clique is called **the clique number**









Edge Progression, Walk and Path

- **Edge progression** is a sequence $v_1, e_1, v_2, e_2, \dots, v_k, e_k, v_{k+1}$ such that $e_i = (v_i, v_{i+1}) \in E(G)$ or $e_i = \{v_i, v_{i+1}\} \in E(G)$ for all $i = 1, \dots, k$.
- Edge progression is called **closed** if $v_1 = v_{k+1}$.
- Directed (undirected) **walk** is directed (undirected) edge progression, where no edge appears more than once, i.e. $e_i \neq e_j$ for all $1 \leq i < j \leq k$.
- Directed (undirected) **path** is directed (undirected) walk, where no node appears more than once, i.e. $v_i \neq v_j$ for all $1 \leq i < j \leq k+1$.
- The path can be also thought of as a graph $P = (\{v_1, v_2, \dots, v_{k+1}\}, \{e_1, e_2, \dots, e_k\})$ and we call it "path from v_1 to v_{k+1}" or a "v_1-v_{k+1} path".
- **The circuit (also called cycle)** is an undirected walk, where no node appears more than once except $v_1 = v_{k+1}$.
- **The cycle (also called circuit)** is a directed walk, where no node appears more than once except $v_1 = v_{k+1}$.

Tree

- An undirected graph is called **connected**, if every pair of nodes is connected by an undirected path.
- The maximal connected subgraphs of G are its **connected components**.
- Every node of the graph is included in exactly one connected component.
- The connected component containing node x can be found as a complete subgraph induced by the set of all nodes which can be reached from x via the undirected path.
- **A forest** is an undirected graph G without a circuit.
- **A tree** is an undirected graph G without a circuit that is connected.
- For every connected graph there exists a spanning that is the tree and it is called the **spanning tree**.

Undirected Graph - Spanning Tree

Let G be an undirected graph with n nodes, then the following statements are equivalent:

- (a) G is a tree.
- (b) G has $n - 1$ edges and no circuit.
- (c) G has $n - 1$ edges and is connected.
- (d) G is connected and while removing any edge it will not be connected anymore.
- (e) G is a minimal graph which has $\delta(X) \neq \oslash$ for all $\oslash \neq X \subset V(G)$
- (f) G is circuit-free and the addition of any edge creates a circuit.
- (g) G contains a unique path between any pair of vertices.

Undirected Graph - Spanning Tree

Proof:
(a)$\Rightarrow$(g): follows from the fact that the union of two distinct paths with the same endpoints contains a circuit.
(f)$\Rightarrow$(b)$\Rightarrow$(c): follows from the fact that for a forest with n nodes, m edges and p connected components $n = m + p$. (The proof is a trivial induction on m).
(g)$\Rightarrow$(e)$\Rightarrow$(d): see [1] page 17 Proposition 2.3.
(d)$\Rightarrow$(f): trivial.
(c)$\Rightarrow$(a): G is connected with $n-1$. As long as there are any circuits in G, we destroy them by deleting any edge of the circuit. Suppose we have deleted k circuits, the resulting graph G' is a tree (contains no circuit and is connected) and has $m = n - 1 - k$ edges. So
$n = m + p = n - 1 - k + 1$, implying $k = 0$.

Connectivity and Trees in Digraphs

- Digraph G is connected if the underlying undirected graph is connected.
- Digraph G is **strongly connected** if there is a path from x to y and from y to x for all x, y in G.
- **The strongly connected component** of G is every maximal strongly connected subgraph H of G.
- Node $x \in V(G)$ is a **root** of graph G, if there is a directed path path from x to every node of G.
- **An out-tree** or arborescence or branching is digraph G that contains a root. No edge enters the root, but exactly one edge enters every other node.
- Properties of trees in an undirected graph - see [1] page 18
- **A binary tree** is tree G in which each node has at most two child nodes.

Summary

A graph is:

- often used to formalize optimization problems
- very general
- easy to represent

Literature

B. H. Korte and Jens Vygen.
Combinatorial Optimization: Theory and Algorithms.
Springer, third edition, 2006.