ILP

Real estate investment, klasicky budovy s XORem a dalsi podminky vcetne 2 ze 3 musi platit.

https://support.dce.felk.cvut.cz/pub/hanzalek/KO/ILP_e.pdf 23/43

Budovy (foto) : http://imgur.com/a/8RNZO

Finanční portfolio

1|prec|wjCj - formulace, k čemu se dá použít relaxované LP řešení při řešení ILP

1|prec|wjCj - Scheduling on One Resource|the tasks are processed in an arbitrary order that satisfies the precedence relation|minimize weighted completion time

https://support.dce.felk.cvut.cz/pub/hanzalek/KO/sched_e.pdf 18-20

ILP formulation for $1\,|\mathrm{prec}|\sum w_j C_j$

min	$\sum_{j=1}^{n}\sum_{i=1}^{n} p_i \cdot x_{ij} \cdot w_j$		
subject to:			
	$x_{i,j} \geq e_{i,j}$	$i,j \in 1..n$	if T_i precedes T_j in G, then it precedes T_j in the schedule
	$x_{i,j} + x_{j,i} = 1$	$i,j \in 1..n, i \neq j$	either T_i precedes T_j, or vice versa
	$1 \leq x_{i,j} + x_{j,k} + x_{k,i} \leq 2$	$i,j,k \in 1..n$, $i \neq j \neq k$	no cycle exists in the digraph of x
	$x_{i,i} = 1$	$i \in 1..n$	
parameters:	$w_{i\in 1..n}, p_{i\in 1..n} \in \mathbb{R}_0^+$	$e_{i\in 1..n, j\in 1..n} \in \{0,1\}$	
variables:	$x_{i\in 1..n, j\in 1..n} \in \{0,1\}$		

Nějaká výroba 5 produktů - formulace

https://support.dce.felk.cvut.cz/pub/hanzalek/KO/ILP_e.pdf 28

Tričká (foto) : http://i.imgur.com/75rDNCn.jpg

Investice do n domů s příjmy z nájmů. Maximalizovat profit při omezení výše investice.

To je to real estate investment

Nakup reklam, zadana tabulka zisku hlasovacich hlasu na zaklade poctu vysilani reklamy, nakoupit muzeme maximalne 3 reklamy.

Scheduling with temp constraints

SCHEDULING

Rothkopf, solve P||Cmax by dynamic prog. p=(2,2,1,2), UB=7, R=2

https://support.dce.felk.cvut.cz/pub/hanzalek/KO/sched_e.pdf 41-42

Dynamic Programming for $P \,||\, C_{max}$ [Rothkopf]

Pseudopolynomial algorithm - the range of discreet values is limited by the upper bound. In some special cases there exists a polynomial algorithm for such a restricted problem.

- we add a binary variable $x_i(t_1, t_2, \ldots, t_R)$ where
  - $i = 1, 2 \ldots n$ is the task index
  - $v = 1, 2, \ldots R$ is the index of the resource
  - $t_v = 0, 1, 2, \ldots UB$ is the time variable associated to the resource v
  - UB is upper bound on C_{max}
- $x_i(t_1, t_2, \ldots, t_R) = 1$ iff tasks $T_1, T_2, \ldots, T_i$ can be assigned to the resource such that P_v is occupied during the time interval $\langle 0, t_v \rangle ; v = 1, 2, \ldots R$

Dynamic Programming for $P \,||\, C_{max}$ [Rothkopf]

Input: R, the number of parallel identical resources, n, the number of nonpreemptive tasks and their processing time $[p_1, p_2, ..., p_n]$.
Output: n-elements vectors s and z where s_i is the start time and z_i is the resource ID.

for $(t_1, t_2, \ldots, t_R) \in \{1, 2, \ldots UB\}^R$ **do** $x_0(t_1, t_2, \ldots, t_R) := 0$;
$x_0(0, 0, \ldots, 0) := 1$;
for $i := 1$ **to** n **do** `// for all tasks`
  for $(t_1, t_2, \ldots, t_R) \in \{0, 1, 2, \ldots UB\}^R$ **do** `// in the whole space`
    $x_i(t_1, t_2, \ldots, t_R) := \mathrm{OR}_{v=1}^{R} x_{i-1}(t_1, t_2, \ldots, t_v - p_i, \ldots t_R)$;
    `//` $x_i() = 1$ `iff there existed`
    `//` $x_{i-1}() = 1$ `''smaller'' by` p_i `in any direction`
  end
end
$C^*_{max} = \min_{x_n(t_1, t_2, \ldots, t_R)=1} \{\max_{v=1,2,\ldots R} \{t_v\}\}$;
Assign tasks $T_n, T_{n-1}, \ldots, T_1$ in the reverse direction;

Time complexity is $O(n \cdot UB^R)$. Example n=3, R=2, p=[2,1,2], C=5.

Formulate PS1|temp|Cmax as time-indexed ILP

https://support.dce.felk.cvut.cz/pub/hanzalek/KO/sched_e.pdf 53-54

lij = koľko má byť medzera medzi start time Ti a Tj

ILP formulation of $PS1 \,|\text{temp}|\, C_{max}$

Task can be represented in two ways:

- **Time-indexed** - ILP model is based on variable x_{it}, which is equal to 1 iff $s_i = t$. Otherwise, it is equal to zero. Processing times are positive integers.

Time-indexed Model for $PS1\,|\text{temp}|\,C_{max}$

$\min C_{max}$

$$\sum_{t=0}^{UB-1}(t\cdot x_{it})+l_{ij}\le\sum_{t=0}^{UB-1}(t\cdot x_{jt}) \quad \forall l_{ij}\neq-\infty \text{ a } i\neq j \text{ (prec. const.)}$$
$$\sum_{i=1}^{n}\left(\sum_{k=\max(0,t-p_i+1)}^{t}x_{ik}\right)\le 1 \quad \forall t\in\{0,\dots UB-1\} \text{ (resource)}$$
$$\sum_{t=0}^{UB-1}x_{it}=1 \quad \forall i\in\{1,\dots n\} \ (T_i \text{ is scheduled})$$
$$\sum_{t=0}^{UB-1}(t\cdot x_{it})+p_i\le C_{max} \quad \forall i\in\{1,\dots n\}$$

variables: $x_{it}\in\{0,1\}$, $C_{max}\in\{0,\dots UB\}$

UB - upper bound of C_{max} (e.g. $UB=\sum_{i=1}^{n}\max\left\{p_i,\max_{i,j\in\{1,\dots,n\}}l_{ij}\right\}$).
Start time of T_i is $s_i=\sum_{t=0}^{UB-1}(t\cdot x_{it})$.
Model contains $n\cdot UB+1$ variables and $|E|+UB+2n$ constraints.
Constant $|E|$ represents the number of temporal constraints (edges in G).

Rozvrhnout tasky pomocí Hornova algoritmu, nakreslit gantův diagram a spočítat Lmax (1|pmtn,rj|Lmax)
https://support.dce.felk.cvut.cz/pub/hanzalek/KO/sched_e.pdf 25 (foto)
http://i.imgur.com/283JBpL.jpg

Problem $1\,|\text{pmtn},r_j|\,L_{max}$ - Horn's Algorithm

Input: $\mathcal{T}$, set of n preemptive tasks. Processing times $(p_1,p_2,\dots,p_n)$, release dates $(r_1,r_2,\dots,r_n)$ and due-dates $(d_1,d_2,\dots,d_n)$.
Output: Start times of preempted parts of tasks.

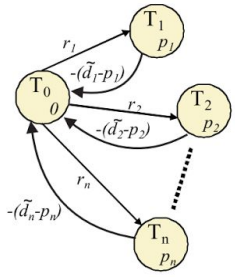
while $\mathcal{T}\neq\emptyset$ **do**
  $t_1:=\min_{T_j\in\mathcal{T}}\{r_j\}$;
  if all tasks are ready at time t_1 **then** $t_2=\infty$;
  else $t_2=\min_{T_j\in\mathcal{T}}\{r_j|r_j>t_1\}$;
  $\mathcal{T}'=\{T_j|T_j\in\mathcal{T},r_j=t_1\}$; // set of ready tasks
  choose $T_k\in\mathcal{T}'$ with minimal d_j; // EDD in $\mathcal{T}'$
  $\delta:=\min\{p_k,t_2-t_1\}$;
  schedule T_k or its part in interval $\langle t_1,t_1+\delta\rangle$;
  if $\delta=p_k$ **then** $\mathcal{T}:=\mathcal{T}\setminus\{T_k\}$;
  else $p_k:=p_k-\delta$; // preemption
  for $T_j\in\mathcal{T}'$ **do** $r_j:=t_1+\delta$;
end

Převod 1|rj,dj|Cmax na PS1|temp|Cmax
https://support.dce.felk.cvut.cz/pub/hanzalek/KO/sched_e.pdf 63

This polynomial reduction proves that $PS1\,|\text{temp}|\,C_{max}$ is NP-hard, since Bratley's problem is NP-hard.

Instance $1\left|r_j,\tilde{d}_j\right|C_{max}$
$r=[r_1,r_2,\dots,r_n]$
$p=[p_1,p_2,\dots,p_n]$
$d=[d_1,d_2,\dots,d_n]$

P-reducible



Úrovňový algoritmus + ganttův graf

Muntz&Coffman's Level Algorithm for $P\,|\text{pmtn, prec}|\,C_{max}$

Principle:

- tasks are picked from **the list ordered by the level** of tasks
- **the level of task** T_j - sum of p_i (including p_j) along the **longest path** from T_j to a terminal task (a task with no successor)
- when more tasks of the same level are assigned to less resources, each task gets **part of the resource capacity** β
- the algorithm moves forward to time τ when **one of the tasks ends** or the task with a lower level would be processed by a bigger capacity β than the tasks with a higher level

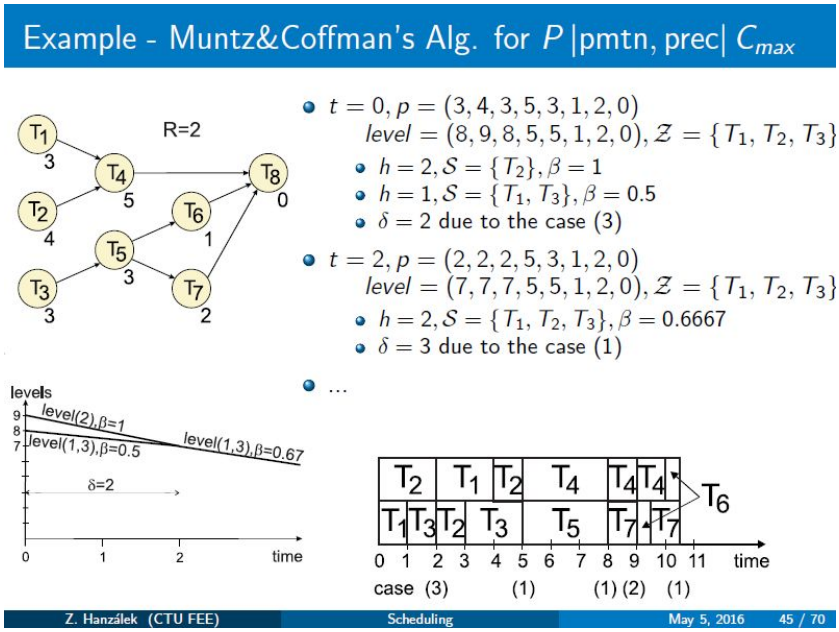
For $P2\,|\text{pmtn, prec}|\,C_{max}$ and $P\,|\text{pmtn, forest}|\,C_{max}$, the algorithm is **exact**.
For $P\,|\text{pmtn, prec}|\,C_{max}$ **approximation** alg. with factor $r_{MC} = 2 - \frac{2}{R}$.
Time complexity is $O(n^2)$.

Input: R, the number of parallel identical resources, n, the number of preemptive tasks and proc. times $[p_1, p_2, ..., p_n]$. Prec. graph G.
Output: n-elements vectors $s^1, ..., s^k, ...s^K$ and $z^1, ..., z^k, ...z^K$ where s_i^k is the start time of the k-th part of task T_i and z_i^k is corresponding resource ID.

Muntz&Coffman's Level Algorithm for $P\,|\text{pmtn, prec}|\,C_{max}$

```
compute the level of all tasks ; t:=0; h:=R; // h represents free res
while unfinished tasks exists do
    construct Z;                    // subset T of free tasks in time t
    while h > 0 and |Z| > 0 do // free resources and free tasks
        construct S;  // subset Z of tasks of the highest level
        if |S| > h then                  // more tasks than resources
            assign part of capacity β := h/|S| to tasks in S; h := 0;
        else
            assign one resource to each task in S; β := 1; h := h − |S|;
        end
        Z := Z \ S;
    end
    compute δ;                                  // see explanation below
    decrease level of tasks by (δ) · β;         // finished part of task
    t := t + δ;h := R;
end
Use McNaughton's alg. to re-schedule parts with more tasks on less res.;
```

$t + \delta$ is time when (1) EITHER one of the assigned tasks is finished (2) OR a current level of an assigned task becomes lower than a level of an unassigned ready task (3) OR a task executed at faster rate β starts to have current level below the current level of a task executed at slower rate



1. Máš precedenčný graf, pri každom tasku máš processing time - vyrátaš level každého tasku tak, že level = najdlhšia cesta z tasku do endového tasku
   t = čas,
   R = počet procesorov
   h = počet voľných procesorov, na začiatku R
2. Kým máš nespracované tasky
   a. Vytvor množinu Z (tasky, ktoré sú v čase t ready - podľa grafu napr. v čase 0 je ready T1, T2, T3)
   b. Kým máš nejaké tasky v Z a voľné procesory (h>0)
      - Vytvor množinu S - tasky zo Z s najvyšším levelom
      - Ak počet taskov v S > h (počet voľných zdrojov)
        - Každému tasku priradíš časť kapacity beta = h/|S| ; h=0
      - Inak beta = 1 pre každý task, h = h- |S|
      - Z = Z\S
   c. Keď si minul všetky tasky, priradil si k nim bety, ideš vyrátať t + d, 3 prípady:
      - ak nejaký task sa celý spracoval
      - Aktuálny level nejakého spracovaváného tasku je menší ako level ešte nespracovaného a ready
      - Aktuálny level tasku s väčším beta ako ostatných spracovávaných je menší

   (proste ako ide čas, stále to odčítavaš z toho levelu a aj processing timu a pozeráš, či sa neznížil viac ako pri taskoch s menším beta alebo ešte nezaradených)
   d. Level taskov sa zmenší podľa l = l - d * beta tasku
   e. t = t+d; h = R;

http://imgur.com/a/2mbON

Převod PSm 1|temp|Cmax na PS 1|temp|Cmax a určení zda je daná instance rozvrhnutelná.

TSP

Derive approx. factor of Christofides alg. (Nestaci napsat factor. Je potreba napsat, proc ma takovy factor.)
https://support.dce.felk.cvut.cz/pub/hanzalek/KO/TSP_e.pdf 25

Christofides' Algorithm is a $\frac{3}{2}$ Approximation

Time complexity is $O(n^3)$
It is a $\frac{3}{2}$ approximation algorithm for the **metric TSP**:

- 1. due to the triangle inequality the skipped nodes do not prolong the route, i.e $c(E(L)) \geq c(E(H))$
- 2. while deleting one edge in the circuit, we create the tree. Therefore, inequality $OPT(K_n, c) \geq c(E(T))$ holds
- 3. since the perfect matching M considers every second edge in the alternating path and being the minimum weight matching it chooses the smaller half, $\frac{OPT(K_n,c)}{2} \geq c(M)$ holds
- 4. due to the construction of L it holds $c(M) + c(E(T)) = c(E(L))$
- finally while inserting 2. and 3. in 4. and appending 1. we obtain $\frac{3}{2}OPT(K_n, c) \geq c(E(H))$

Formulace TSP s time windows pomocí ILP
https://support.dce.felk.cvut.cz/pub/hanzalek/KO/ILP_e.pdf 9

Example ILP3: Traveling Salesman Problem

Asymmetric Traveling Salesman Problem

- **Instance:** complete digraph K_n ($n \geq 3$), distance matrix $c : V \times V \to \mathbb{Q}^+$.
- **Goal:** find the shortest Hamiltonian cycle (i.e. a closed oriented walk going through all nodes).

$x_{i,j} = 1$ iff node i is in the cycle just before node j
The enter and leave constraints do not capture the TSP completely, since any disjoint cycle (i.e. consisting of several sub-tours) will satisfy them.
We use s_i, the "time" of entering node i, to **eliminate the sub-tours**.

$$
\begin{array}{lll}
\text{min} & \sum_{i\in 1..n}\sum_{j\in 1..n} c_{i,j} * x_{i,j} & \\
\text{subject to:} & & \\
 & \sum_{i\in 1..n} x_{i,j} = 1 \quad j \in 1..n & \text{enter once} \\
 & \sum_{j\in 1..n} x_{i,j} = 1 \quad i \in 1..n & \text{leave once} \\
s_i + c_{i,j} - (1 - x_{i,j}) * M \leq s_j & i \in 1..n, j \in 2..n & \text{cycle indivisibility} \\
\text{parameters:} & M \in \mathbb{Z}_0^+,\ n \in \mathbb{Z}_0^+,\ c_{i\in 1..n, j\in 1..n} \in \mathbb{Q}^+ & \\
\text{variables:} & x_{i\in 1..n, j\in 1..n} \in \{0,1\},\ s_{i\in 1..n} \in \mathbb{R}_0^+ &
\end{array}
$$

Pravděpodobná neexistence r-aproximačního algoritmu, převod na Hamilt. Kružnici
https://support.dce.felk.cvut.cz/pub/hanzalek/KO/TSP_e.pdf 18-20

Likely Nonexistence of Polynomial r-approximation Algorithm for General TSP

Theorem

If we believe P≠NP, than there is no polynomial r-approximation algorithm for **TSP** for $r \geq 1$.

Proof by contradiction:
Assume there exists a polynomial r-approximation algorithm $\mathcal{A}$ for **TSP**.
We further show that we can solve the **HC** problem while using such an "inaccurate" algorithm $\mathcal{A}$.
Since **HC** is NP-complete, P=NP.

In other words: if there exists a polynomial r-approximation algorithm $\mathcal{A}$ solving **TSP**, then the NP-complete **HC** problem can be solved in polynomial time by $\mathcal{A}$.

Likely Nonexistence of Polynomial r-approximation Algorithm for General TSP

Every **HC** instance can be polynomially reduced to a TSP instance "inaccurately" solved by r-approximation algorithm $\mathcal{A}$:

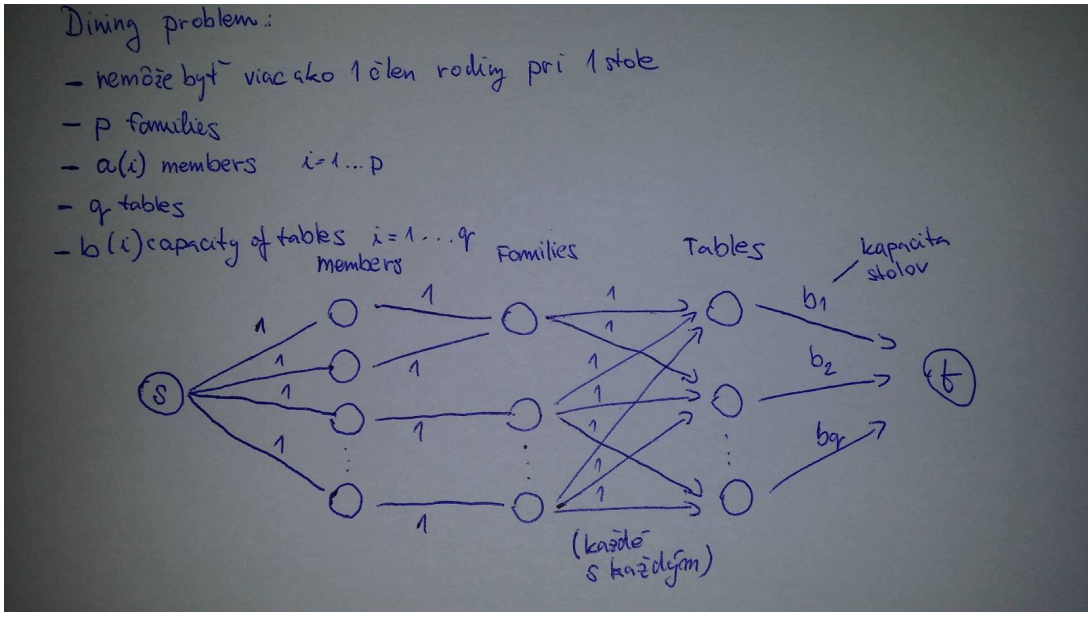
- Let G be an undirected graph in which we want to find the Hamiltonian circuit.
- Create a **TSP** instance such that every node from G is associated to one node (city) in the complete undirected graph K_n. Weight (distance) of $\{i,j\}$ in K_n equals:

$$c(\{i,j\}) = \begin{cases} 1 & \text{if } \{i,j\} \in E(G); \\ 2+(r-1)*n & \text{if } \{i,j\} \notin E(G). \end{cases}$$

- We use $\mathcal{A}$ to solve the instance.
  - if the result is in interval $\langle n, r*n \rangle$, then the Hamiltonian circuit exists,
  - otherwise the result is greater or equal to $(n-1)+2+(r-1)*n = r*n+1$ and G has no Hamiltonian circuit.

FLOWS

Dinning problem - max flow problem, Nekolik rodin je na veceri. Je potreba rozhodit cleny rodin tak, aby nesedeli dva clenove jedne rodiny u stejneho stolu. Je potreba urcit, kdy je reseni feasible.



ILP formulace multikomoditních toků.

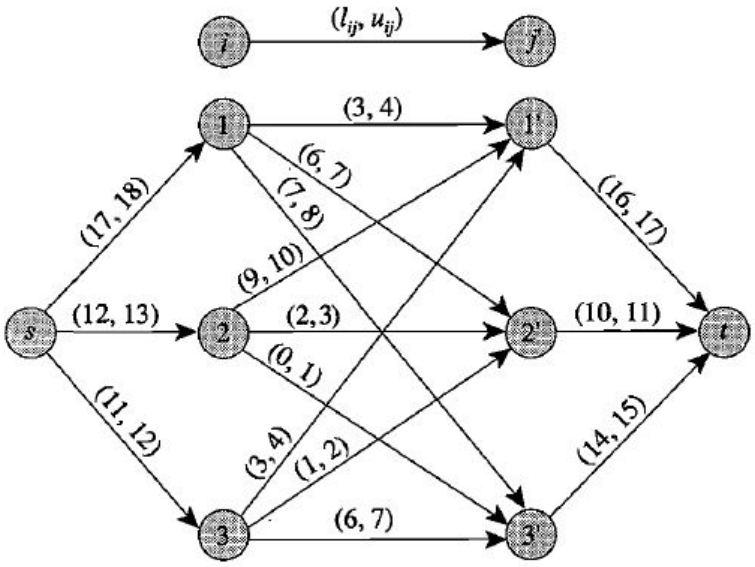
https://support.dce.felk.cvut.cz/pub/hanzalek/KO/Flows_e.pdf 40

https://moodle.fel.cvut.cz/pluginfile.php/42767/mod_resource/content/2/Warehousing%20seasonal%20products%20%28extended%29.pdf (foto) : http://imgur.com/a/FaWim

Zaokrouhlování čísel/řádků/sloupců v 3×3 matici

			Row sum
	3.1	6.8	7.3	17.2
	9.6	2.4	0.7	12.7
	3.6	1.2	6.5	11.3
Column sum	16.3	10.4	14.5	

Figure 6.2 Matrix rounding problem.



Nalezení počátečního přípustného toku pro Ford-Fulkersonův Algoritmus převést na rozhodovací problém přípustného toku v síti. (popsat algoritmus)

https://support.dce.felk.cvut.cz/pub/hanzalek/KO/Flows_e.pdf 24-25

Ak je l(e) všade nulový - nájdem nulový tok ktorý spĺňa KZ
Ak nie - transformácia na feasible flow decision problem

- Pridám hranu s ->t , u = Inf
- KZ aplikujem aj na s a t
- f(e) = f'(e) + l (e)

How to Find an Initial Feasible Flow for Ford-Fulkerson Algorithm?

If $\forall e \in E(G)$; $l(e) = 0$ - easy solution - we use zero flow which satisfies Kirchhoff's law.
If $\exists e \in E(G)$; $l(e) > 0$, we transform the feasible flow problem to the **feasible flow decision problem** as follows:
1) We transform the maximum flow problem (with non-zero lower bounds) to a circulation problem by **adding an arc from t to s of infinite capacity**. Consequently, the Kirchhoff's law applies to nodes s and t. Therefore, a feasible **circulation must satisfy**:

$$l(e) \leq f(e) \leq u(e) \qquad e \in E(G)$$
$$\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0 \qquad v \in V(G)$$

How to find an initial feasible flow for Ford-Fulkerson algorithm?

2) Substituting $f(e) = f(e)' + l(e)$, we obtain the transformed problem:

$$0 \le f(e)' \le u(e) - l(e) \qquad e \in E(G)$$

$$\sum_{e\in\delta^+(v)} f(e)' - \sum_{e\in\delta^-(v)} f(e)' = \underbrace{\sum_{e\in\delta^-(v)} l(e) - \sum_{e\in\delta^+(v)} l(e)}_{b(v)} \qquad v \in V(G)$$

3) This is a **feasible flow decision problem** because $\sum_{v\in V(G)} b(v) = 0$ (notice that $l(e)$ appears twice in summation, once with a positive and once with a negative sign).

4) While solving this decision problem (i.e. adding s', t' and solving the maximum flow problem with zero lower bounds) we obtain the initial feasible circulation/flow or decide that it does not exist.

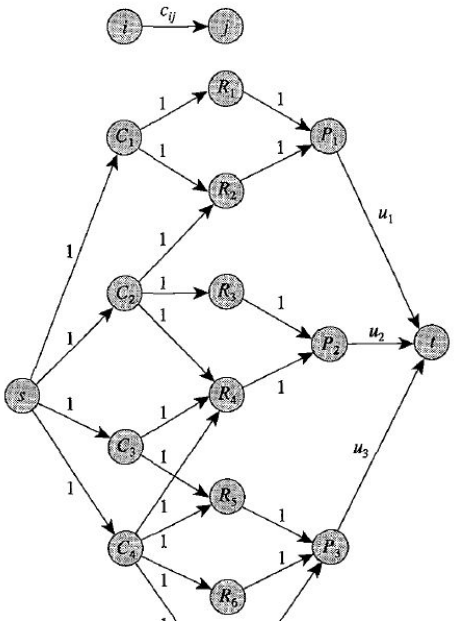
Conclusion: finding of the **initial flow with nonzero lower bounds** can be transformed to the **feasible flow decision problem** which can be transformed to the **maximum flow problem with zero lower bounds**.

Distinct Representatives, př. 6.2 na str. 170 v [AMO93] (formulovat jako úlohu maximálního toku)

Application 6.2 Problem of Representatives

A town has r residents $R_1, R_2, \ldots, R_r$; q clubs $C_1, C_2, \ldots, C_q$; and p political parties $P_1, P_2, \ldots, P_p$. Each resident is a member of at least one club and can belong to exactly one political party. Each club must nominate one of its members to represent it on the town's governing council so that the number of council members belonging to the political party P_k is at most u_k. Is it possible to find a council that satisfies this "balancing" property?

We illustrate this formulation with an example. We consider a problem with $r = 7$, $q = 4$, $p = 3$, and formulate it as a maximum flow problem in Figure 6.1. The nodes $R_1, R_2, \ldots, R_7$ represent the residents, the nodes $C_1, C_2, \ldots, C_4$ represent the clubs, and the nodes $P_1, P_2, \ldots, P_3$ represent the political parties.



Ford-Fulkerson algorithm, 3 iterace vcetne odecitani pres zpetnou hranu + urcit hrany minimum cut

SHORTEST PATHS

Investment oportunities, Mr. Dow Jones - prednasky SPT na slajdu 29/42

Mr. Dow Jones, 50 years old, wishes to place his Individual Retirement Account funds in various investment opportunities so that at the age of 65 years, when he withdraws the funds, he has accrued maximum possible amount of money. Assume that Mr. Jones knows the investment alternatives for the next 15 years - each opportunity has starting year k, maturity period p (in years) and appreciation a it offers for the maturity period. How would you formulate this investment problem as a shortest path problem, assuming that at any point in time, Mr. Jones invests all his funds in a single investment alternative.

Nie je tam riesenie

Formulace problému tak, aby šel vyřešit Dijkstrou. Máme řidiče autobusu a směny v době 9-17. K dispozici jsou různé časy směn (9-11, 9-13, 12-14, 12-16, 12-17, 14-16, 15-17 cca) a ke směnám ceny. Je to někde v knížce Network Flows - Ahuja, Magnanti, Orlin.

V knizke je to na strane 127 (144) neni tam riesenie

Najít nejdelší cesty Floydem

Varianta pre najdlhšiu cestu = ceny hrán vynásobíš -1 a klasicky pokračuješ

- Máš dve matice : l a p
  - l => dáš ceny hrán tam, kde hrany existujú, inak dáš všade Inf
  - pij := i (tam kde sú hrany, dáš node, z kade vychádza, pri i=i dáš i)
- Robíš k = 1..n iterácii - označíš si riadok a stĺpec podľa toho, v ktorej si iterácii
- Zoberieš si dve bunky - 1 z riadku, 1 zo stĺpca (lik, lkj) spočítaš ich a pozrieš sa do lij, či nová hodnota nie je menšia, ak hej, dáš to tam a v do pij dáš hodnotu z pkj (v ktorom stĺpci je zlepšenie -> j, a ktorý riadok máš označený -> k)

5 výroků a říct zda platí nebo ne. Pokud platí, dokázat, pokud ne, udělat protipříklad. Výroky typu „když zvýšíme cenu hrany v grafu o násobek k, délka nejkratší cesty se také zvýší k-krát“, „Dijkstrův algoritmus nalezne vždy nejkratší cestu s nejmenším počtem hran“)

Nejspolehlivejsi spojeni, vektor spolehlivosti p rozsah (0,1), vysledne spojeni je produkt vsech spolehlivosti po ceste a)prevod ulohy nejspolehlivejsiho spojeni, aby sel resit Dijkstrou b) Modifikace Dijkstry aby resil tuhle ulohu c) bude a) a b) fungovat, kdyz bude p v rozsahu (0,inf)

Dukaz spravnosti Dijkstry

KNAPSACK

a) Dyn. programovani b) existuje vice optimalnich reseni?

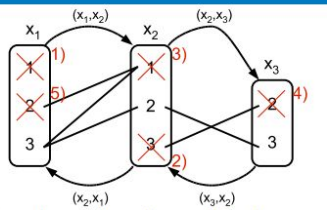
CONSTRAINT PROGR.

Hranová konzistence nějakého zadání (AC-3)

- Podla variables a constrainov si vytvoris hrany, aj opacne,napr. x1-x5 < 7 urobis hrany (x1,x5), (x5,x1)
- Vsetky hrany hodis do queue, vyberes prvu hranu (xk,xm) pozries sa na domain laveho nodu (xk) a prejdes vsetky prvky z domainu xk a vsetky constrainty kde je xk ci ti aspon nieco sedi, ak nie vymazes danu hodnotu z domainu xk (aj viac hodnot)
- Ak si aspon jednu vec vymazal a existuje nejaka hrana (xp,xk) tak ju hodis do queue, takto pokracujes pokym nemas prazdnu queue
- Vysledkom su domainy vsetkych D1 az Dn pre kazde x1 az xn + ten obrazok s hranami

Example: Application of REVISE

CSP with variables $X = \{x_1, x_2, x_3\}$, constraints $x_1 > x_2$, $x_2 \neq x_3$, $x_2 + x_3 > 4$, and domains $D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2, 3\}$, $D_3 = \{2, 3\}$.



revised arc	deleted	revised domain	(x_1, x_2)	(x_2, x_1)	(x_2, x_3)	(x_3, x_2)
(x_1, x_2)	$1^{1)}$	$D_1 = \{2, 3\}$	consist	nonconsist	nonconsist	consist
(x_2, x_1)	$3^{2)}$	$D_2 = \{1, 2\}$	consist	consist	nonconsist	nonconsist
(x_2, x_3)	$1^{3)}$	$D_2 = \{2\}$	nonconsist	consist	consist	nonconsist
(x_3, x_2)	$2^{4)}$	$D_3 = \{3\}$	nonconsist	consist	consist	consist

After revision, some of the arcs are still **nonconsistent**

- the reason is that some of the domains have been reduced
- continue in the revision until all the arc are consistent (without consistence check - see AC-3)

revised arc	deleted	revised domain	(x_1, x_2)	(x_2, x_1)	(x_2, x_3)	(x_3, x_2)
(x_1, x_2)	$2^{5)}$	$D_1 = \{3\}$	consist	consist	consist	consist

Arc Consistency - AC-3 Algorithm

Maintain a queue of arcs to be revised (the arc is put in the queue only if it's consistency could have been affected by the reduction of the domain).

procedure AC-3
Input: X, D, C and graph G.
Output: Binary variable *fail* indicating no solution in this part of the state space. The set of the revised domains D.

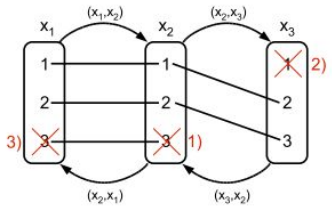
```
fail = 0;Q := E(G);                    // initialize Q by arcs of G
while Q ≠ ∅ do
    select and remove arc (x_k, x_m) from Q;
    (deleted, D_k)=REVISE(k, m, D_k, D_m, C);
    if deleted then
        if D_k = ∅ then fail = 1 and EXIT ;
        else Q := Q ∪ {(x_i, x_k) such that (x_i, x_k) ∈ E(G) and i ≠ m};
    end
end
```

The revision of (x_k, x_m) does not change the arc consistency of (x_m, x_k).

Example: Iteration of AC-3

CSP with variables $X = \{x_1, x_2, x_3\}$, constraints $x_1 = x_2$, $x_2 + 1 = x_3$ and domains $D_1 = \{1,2,3\}, D_2 = \{1,2,3\}, D_3 = \{1,2,3\}$.

Initialization: $Q = \{(x_1, x_2), (x_2, x_1), (x_2, x_3), (x_3, x_2)\}$
revise (x_1, x_2)
$D_1 = \{1,2,3\}, D_2 = \{1,2,3\}, D_3 = \{1,2,3\}$
$Q = \{(x_2, x_1), (x_2, x_3), (x_3, x_2)\}$
revise (x_2, x_1)
$D_1 = \{1,2,3\}, D_2 = \{1,2,3\}, D_3 = \{1,2,3\}$
$Q = \{(x_2, x_3), (x_3, x_2)\}$
revise (x_2, x_3)
$D_1 = \{1,2,3\}, D_2 = \{1,2\}^{1)}, D_3 = \{1,2,3\}$
$Q = \{(x_3, x_2), (\mathbf{x_1}, \mathbf{x_2})\}$
revise (x_3, x_2)
$D_1 = \{1,2,3\}, D_2 = \{1,2\}, D_3 = \{2,3\}^{2)}$
$Q = \{(x_1, x_2)\}$
revise (x_1, x_2)
$D_1 = \{1,2\}^{3)}, D_2 = \{1,2\}, D_3 = \{2,3\}$
$Q = \emptyset$



http://oi-wiki.cz/doku.php/courses/a4m35ko/zkouska24052011