PA167 Rozvrhování

16. května 2022

Hodnocení

Závěrečná písemná práce: 80 bodů

- minimální počet bodů za zkoušku: 40 bodů
- otázky z několika různých oblastí předmětu
- příklady: zadán problém, případně i metoda, cílem výpočet rozvrhu; srovnávací, algoritmus, pojmy
- cca 7 příkladů
- vzorové zadání písemné práce na webu předmětu

Vnitrosemestrální písemná práce: **28.3.2022** 20 bodů

- minimální počet bodů za vnitrosemestrální práci: 8 bodů

Bonusové body: až 1 bod za aktivní účast na přednášce

- odpovědi/zápojení do diskuse/otázky studentů

Hodnocení: A 90 a více, B 80-89, C 70-79, D 60-69, E 50-59

Základní informace

- Web předmětu: interaktivní osnova na IS MU
  - průsvitky + videa + dotazy k probírané látce
    průběžně zveřejňovány na webu předmětu
- Sbírka cca 240 vzorových příkladů
  - včetně řešení vybraných příkladů
  - poklad pro přípravu na zkoušku
- Rozvrhování a plánování v jiných přednáškách:
  - IV126 Umělá inteligence II
  - PA163 Omezující podmínky
  - IA158 Real Time Systems

Literatura

- Michael Pinedo: Planning and Scheduling in Manufacturing and Services, Springer, 2009.
- Philippe Baptiste, Claude Le Pape, Wim Nuijten: Constraint-based scheduling : applying constraint programming to scheduling problems. Kluwer Academic Publishers, 2001.
- Michael Pinedo: Scheduling Theory, Algorithms, and Systems Springer, 2016.

Elektronické zdroje

- Michael Pinedo, Planning and Scheduling in Manufacturing and Services a další knihy `https://wp.nyu.edu/michaelpinedo/books/`
- Sigurdur Olafsson, Iowa State University, USA
  `http://www.stern.nyu.edu/om/faculty/pinedo/book2/dowload.html`
- Erwin Hans, Johann Hurink, University of Twente, Nizozemí
  `http://www.stern.nyu.edu/om/faculty/pinedo/book2/dowload.html`
- Roman Barták, MFF UK, CR
  `http://kti.ms.mff.cuni.cz/~bartak/planovani/`

Předběžný přehled přednášky (I. část)

Úvod

- terminologie
- příklady reálných problémů
- Grahamova klasifikace rozvrhovacích problémů

Obecné řešící metody

- heuristiky
  - řídící pravidla *(dispatching rules)*
  - lokální prohledávání:
    simulované žíhání, tabu prohledávání, genetické algoritmy
- matematické programování: formulace
  - myšlenky řešení
  - lineární, celočíselné
- programování s omezujícími podmínkami
  - úvod k omezujícím podmínkám
  - rozvrhování s omezujícími podmínkami

Předběžný přehled přednášky (II. část)

- Plánování projektu: reprezentace projektu, kritická cesta, kompromis mezi časem a cenou, pracovní síla.
- Plánování úloh: řídící pravidla, metoda větví a mezí & paprskové prohledávání, matematické prohledávání.
- Rozvrhování montážních systémů:
  montážní linka s flexibilním časem, s fixním časem.
- Rezervace:
  intervalové rozvrhování, rezervační systémy s rezervou.
- Timetabling:
  rozvrhování s operátory, rozvrhování s pracovní silou.
- Univerzitní rozvrhování:
  teorie a praxe (rozvrhování na Masarykově univerzitě)

Úvod do rozvrhování

16. května 2022

Definice pojmu rozvrhování

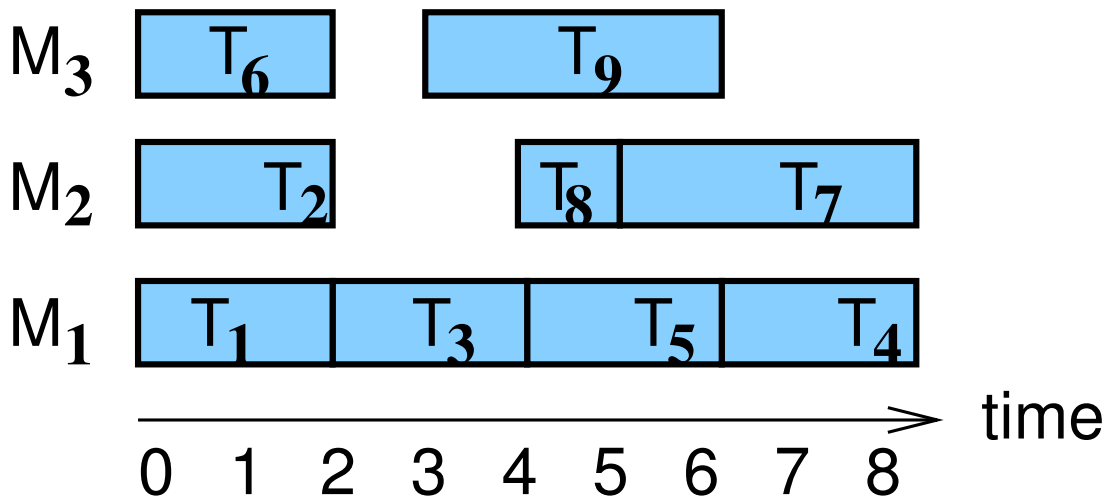
- Rozvrhování

  optimální alokace/přiřazení zdrojů množině úloh v čase

  - omezené množství zdrojů
  - maximalizace zisku za daných omezení

- Stroj $M_i, i = 1, \ldots, m$ úloha $T_j, j = 1, \ldots, n$

Strojově orientovaný Ganttův diagram



Rozvrh

Rozvrh:

- dán umístěním úloh do konkrétního času a na konkrétní zdroje, kde mají být úlohy prováděny

Úplný rozvrh:

- v rozvrhu jsou umístěny všechny úlohy ze zadání problému

Částečný rozvrh:

- některé úlohy ze zadání problému nejsou umístěny/přiřazeny

Konzistentní rozvrh:

- rozvrh, ve kterém jsou splněna všechna omezení kladená na zdroje a umístěné/přiřazené úlohy, např.
  - úloha je naplánována v čase, kdy je dostupná
  - na jednom stroji (s jednotkovou kapacitou) běží nejvýše jedna úloha

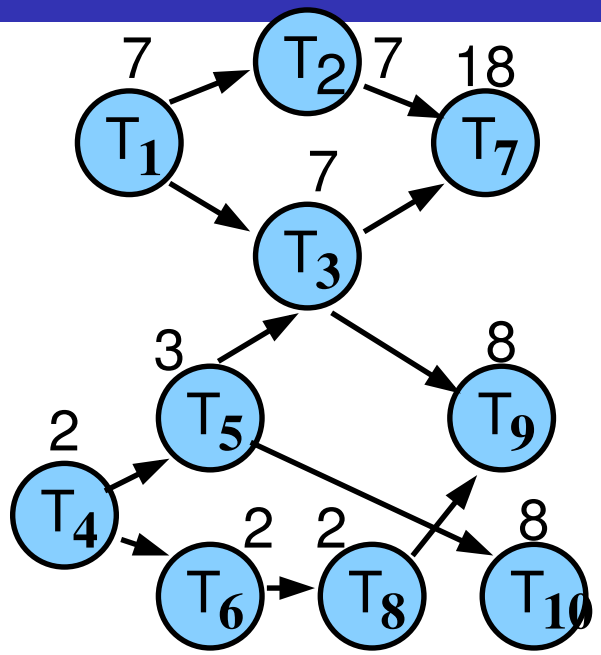
Konzistentní úplný rozvrh vs. konzistentní částečný rozvrh

Optimální rozvrh:

- umístění úloh na stroje je optimální vzhledem k zadanému optimalizačnímu kritériu, např.
  - min C_{max}: makespan (čas dokončení poslední úlohy) je minimální

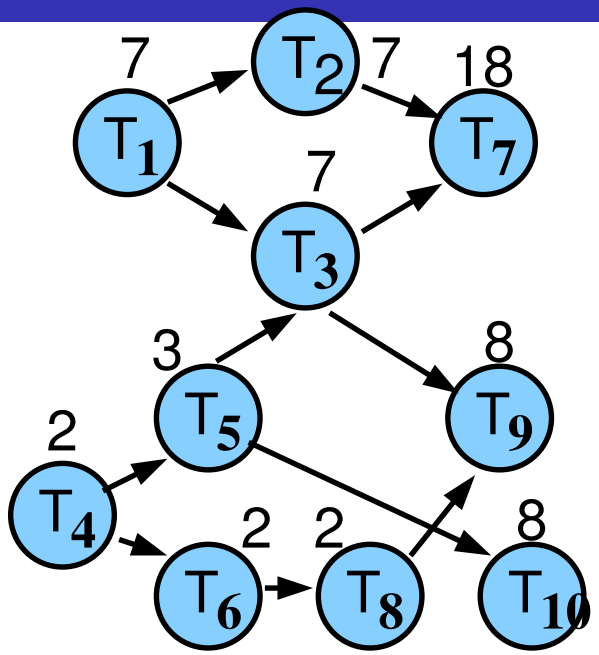
Příklad: montáž kola

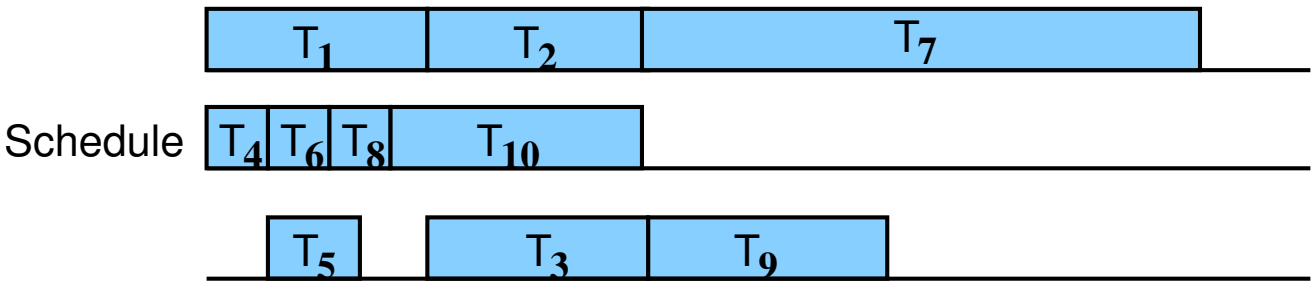
- 10 úloh s danou dobou trvání
- Precedenční podmínky
  - úlohu lze provést až po provedení zadané množiny úloh
- Nepreemtivní úlohy
  - úlohy nelze přerušit
- Optimalizační kritéria
  - minimalizace makespan
  - minimální počet pracovníků



Příklad: montáž kola

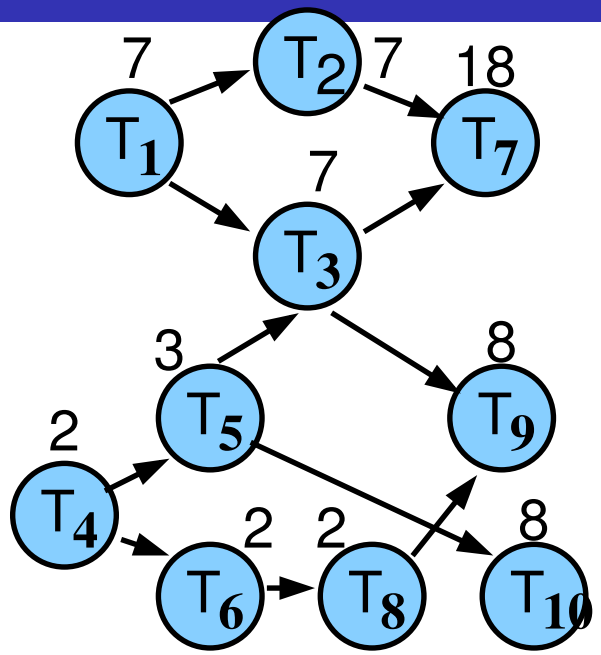
- 10 úloh s danou dobou trvání
- Precedenční podmínky
  - úlohu lze provést až po provedení zadané množiny úloh
- Nepreemtivní úlohy
  - úlohy nelze přerušit
- Optimalizační kritéria
  - minimalizace makespan
  - minimální počet pracovníků

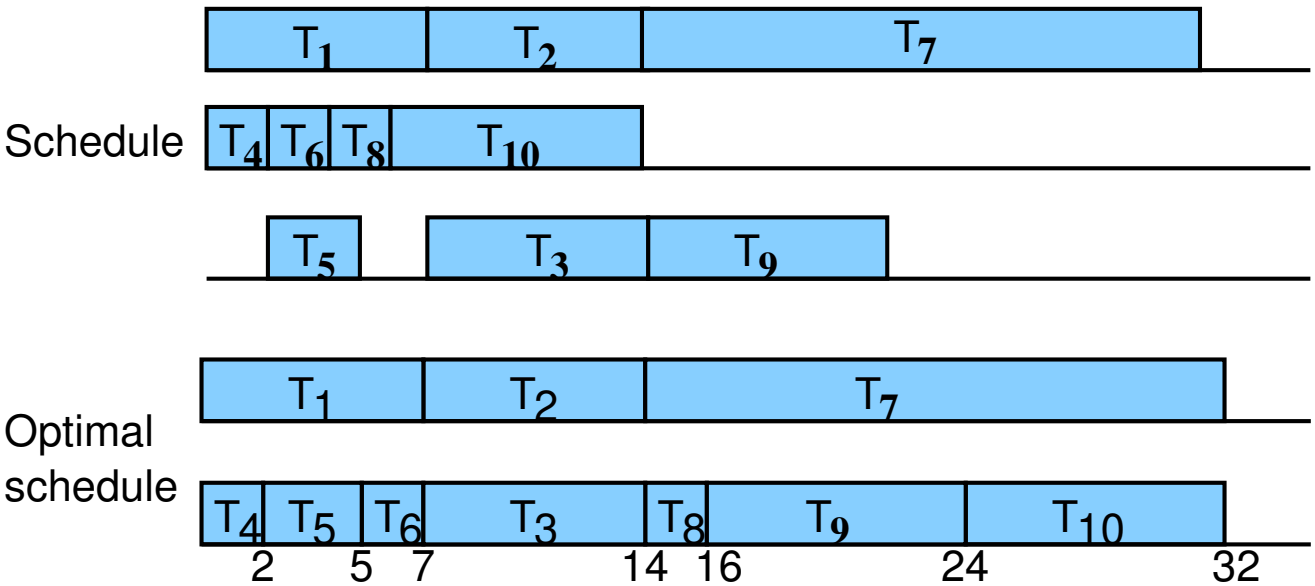




Příklad: montáž kola

- 10 úloh s danou dobou trvání
- Precedenční podmínky
  - úlohu lze provést až po provedení zadané množiny úloh
- Nepreemtivní úlohy
  - úlohy nelze přerušit
- Optimalizační kritéria
  - minimalizace makespan
  - minimální počet pracovníků





Reálný problém: rozvrhování sester v nemocnici

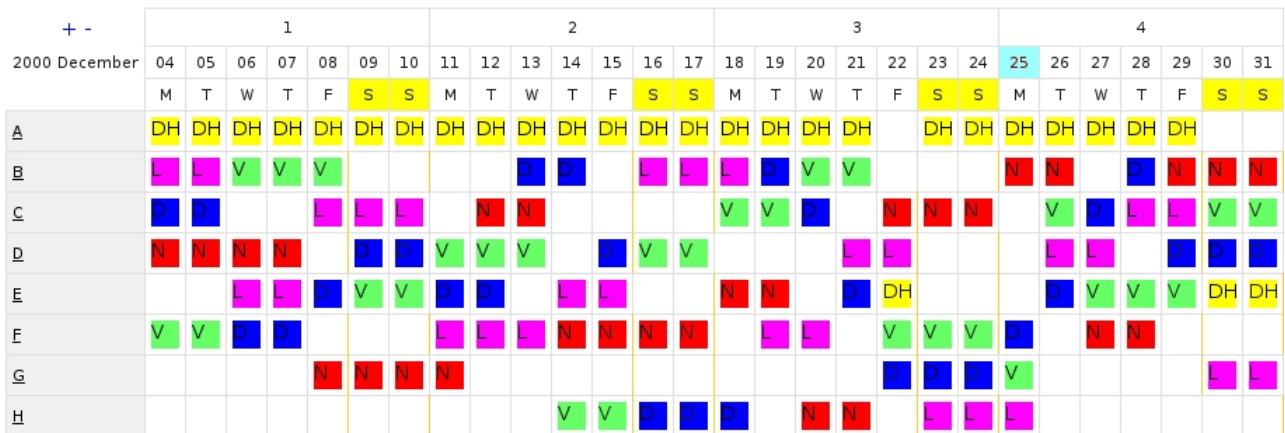
Jedná se o problém rozvrhování zaměstnanců

Požadavky na personál

- odlišný počet sester v pracovní dny a o víkendu
- menší nároky při obsazování nočních směn
- dodržení pravidel daných ze zákona
- preference zaměstnanců na pracovní dobu
- ...

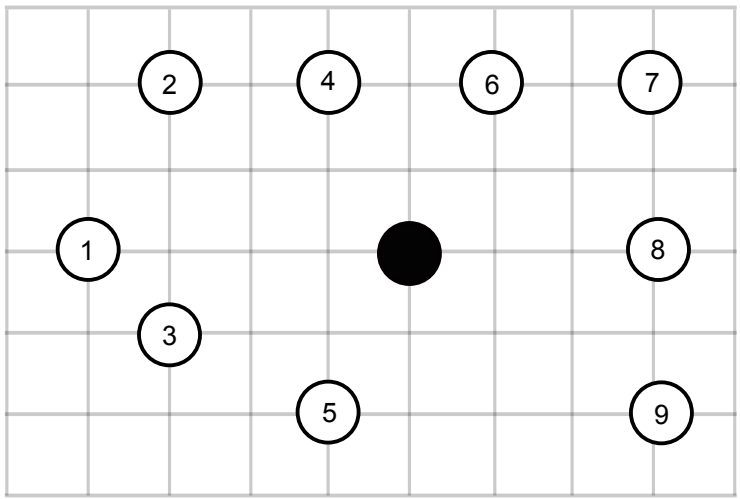
Cíl

- určit přiřazení sester na směny
- splnění požadavků
- minimalizace ceny



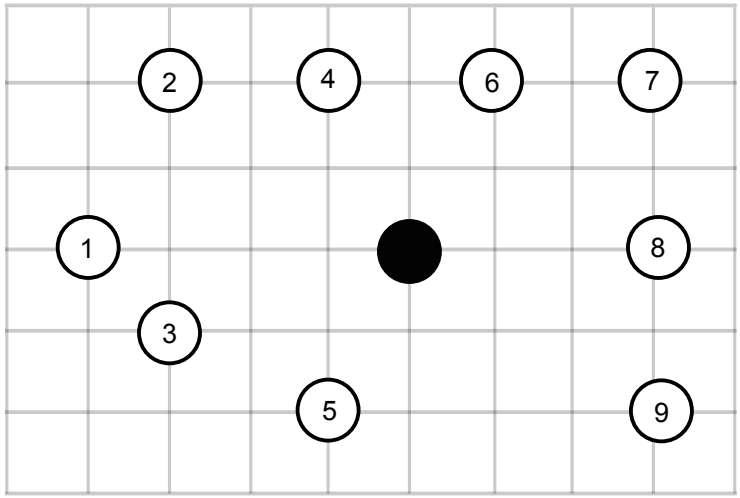
Plánování nákladní automobilové dopravy

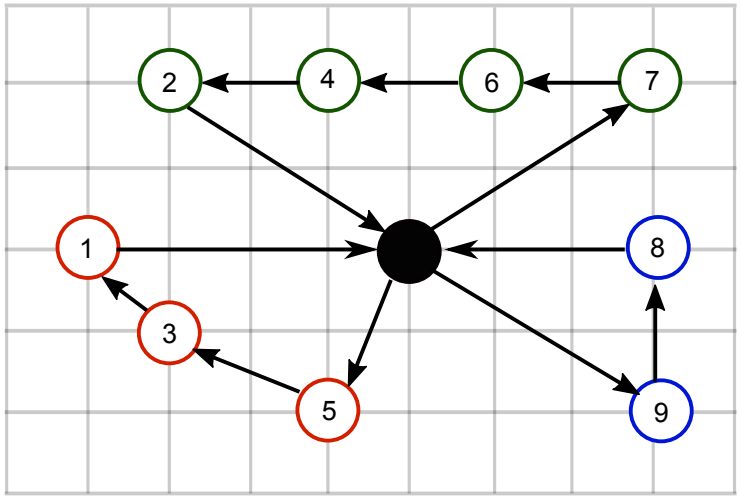
- Problém směrování vozidel (vehicle routing problem)
  - požadavky na doručení/vyzvednutí/doručení+vyzvednutí
    - lokace, časová okna, hmotnost, objem
  - vozidla, která musí tyto lokace obsloužit
    - kapacita/objem vozidel, jedno/více depot vozidel
    - stejná/různá vozidla
  - optimalizace: počtu vozidel, vzdálenosti, ceny
- Statický vs. dynamický problém
  - problém dán předem vs. reakce na změny problému při realizaci řešení
- Spolupráce FI s firmou Wereldo



Plánování nákladní automobilové dopravy

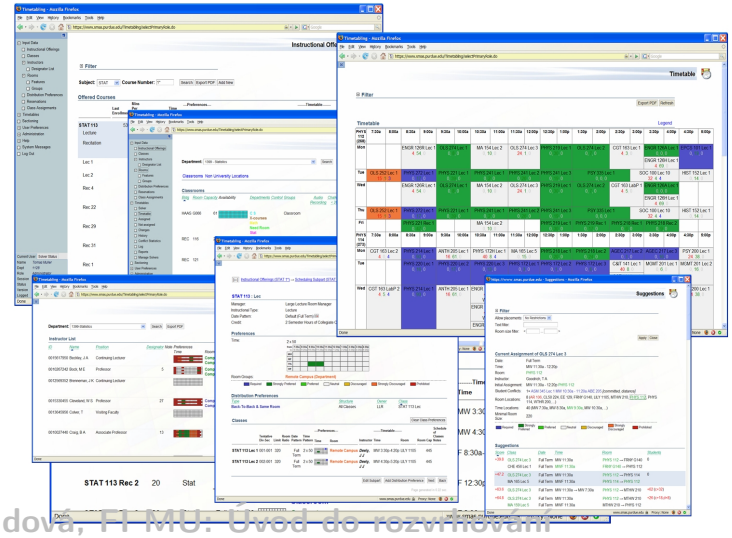
- Problém směrování vozidel (vehicle routing problem)
  - požadavky na doručení/vyzvednutí/doručení+vyzvednutí
    - lokace, časová okna, hmotnost, objem
  - vozidla, která musí tyto lokace obsloužit
    - kapacita/objem vozidel, jedno/více depot vozidel
    - stejná/různá vozidla
  - optimalizace: počtu vozidel, vzdálenosti, ceny
- Statický vs. dynamický problém
  - problém dán předem vs. reakce na změny problému při realizaci řešení
- Spolupráce FI s firmou Wereldo

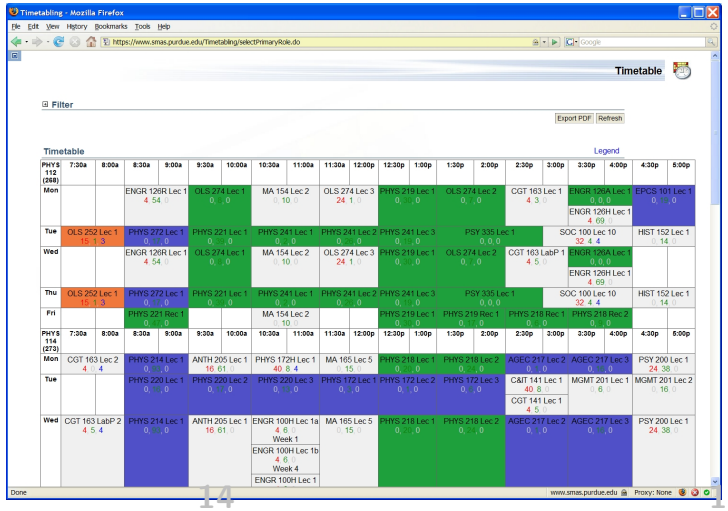




Univerzitní rozvrhování předmětů http://www.unitime.org

- Nalezení času a místnosti pro výuku předmětů na univerzitě
  - omezení kladena na umístění předmětů
    - optimalizace preferenčních požadavků na čas a místnosti
  - každý předmět má určeny své studenty (registrace, studijní obory)
    - minimalizace počtu překrývajících se předmětů pro všechny studenty
- Návrh, vývoj a používání systému UniTime
  - FI spolupracuje na návrhu a vývoji od 2001
  - primárně vyvinuto a používáno na Purdue University, USA
  - MU: používáno na 7 fakultách včetně FI
- International Timetabling Competition ITC 2019
  - reálné problémy z 10 různých univerzit po celém světě (UniTime data)





Terminologie: scheduling vs. timetabling

Scheduling ... rozvrhování/plánování

- alokace zdrojů za daných podmínek na objekty umístěných v časoprostoru tak, že je minimalizována celková cena daných zdrojů
- důraz je kladen na uspořádání objektů
  - precedenční podmínky
  - př. plánování výroby: stanovení pořadí operací, důležitost časových návazností operací
- *schedule* ... rozvrh
  - zahrnuje prostorové a časové informace

Terminologie: scheduling vs. timetabling

Scheduling ... rozvrhování/plánování

- alokace zdrojů za daných podmínek na objekty umístěných v časoprostoru tak, že je minimalizována celková cena daných zdrojů
- důraz je kladen na uspořádání objektů
  - precedenční podmínky
  - př. plánování výroby: stanovení pořadí operací, důležitost časových návazností operací
- *schedule* ... rozvrh
  - zahrnuje prostorové a časové informace

Timetabling ... rozvrhování

- alokace zdrojů za daných podmínek na objekty umístěných v časoprostoru tak, že jsou co nejlépe splněna zadaná kritéria
- důraz kladen na konkrétní časové umístění objektů
- často vymezen předem časový horizont (počet rozvrhovaných slotů)
  - př. školní rozvrhování: předmětům přiřazen čas a místo vyuky
- *timetable* ... rozvrh
  - ukazuje, kdy a kde se budou události konat

Terminologie: planning

Plánování (*planning*) někdy chápáno v dlouhodobějším horizontu

- krátkodobé podrobné rozvrhování + dlouhodobé obecné plánování
- např. plánování=vytvoření vhodné množiny úloh
  - tak aby bylo dosaženo zadaných cílů

  rozvrhování=přiřazení úloh v čase na zdroje
  - tak aby byla minimalizována cena
- nebo dlouhodobé plánování zdrojů
  - tak abychom pokryli budoucí potřeby

  vs. krátkodobé rozvrhování úloh na zdroje
  - tak aby byly splněny aktuální požadavky

Terminologie: AI planning

Plánování v umělé inteligenci (*AI planning*)

- vykonání posloupnosti akcí tak,
  abychom se dostali z počátečního stavu do koncového stavu
- př. plánování činností robota
  - počáteční stav v místnosti, cílový stav v místnosti,
    robot provede posloupnost akcí (např. přemístí předměty) tak, aby se dostal do cílového stavu
- lze chápat jako vytvoření vhodné množiny úloh jako u plánování
  (viz předchozí průsvitka)

Přednáška IV126 Umělá inteligence II

- zahrnut blok přednášek o plánování

Terminologie: sequencing, rostering

Sequencing ... seřazení

- za daných podmínek:

  konstrukce pořadí úloh, ve kterém budou prováděny
- *sequence* ... posloupnost
  - pořadí, ve kterém jsou úlohy prováděny
- př. pořadí automobilů na montážní linku

Rostering

- umístění zdrojů za daných podmínek do slotů s pomocí vzorů (*pattern*)
- *roster* ... rozpis
  - seznam jmen lidí, který určuje, které úlohy budou provádět a kdy
- př. rozpis sester v nemocnici, rozpis řidičů autobusů

Úlohy, stroje

- Stroje (zdroje, prostředky) $i = 1, \ldots, m$
- Úlohy (aktivity) $j = 1, \ldots, n$
- (i, j) operace nebo provádění úlohy j na stroji i
  - úloha se může skládat z několika operací
  - příklad: úloha 4 má tři operace s nenulovou dobou trvání (2,4),(3,4),(6,4), tj. je prováděna na strojích 2,3,6
- Statické parametry úlohy
  - doba trvání p_{ij}, p_j: doba provádění úlohy j na stroji i
  - termín dostupnosti j *(release date)* r_j:
    nejdřívější čas, ve kterém může být úloha j prováděna
  - termín dokončení *(due date)* d_j:
    čas, do kdy by měla být úloha j nejpozději dokončena (preference)
    vs. *deadline*: čas, do kdy musí být úloha j nejpozději dokončena (požadavek)
  - váha w_j: důležitost úlohy j relativně vzhledem k ostatním úloham v systému

Úlohy, stroje

- Stroje (zdroje, prostředky) $i = 1, \ldots, m$
- Úlohy (aktivity) $j = 1, \ldots, n$
- (i, j) operace nebo provádění úlohy j na stroji i
  - úloha se může skládat z několika operací
  - příklad: úloha 4 má tři operace s nenulovou dobou trvání (2,4),(3,4),(6,4), tj. je prováděna na strojích 2,3,6
- Statické parametry úlohy
  - doba trvání p_{ij}, p_j: doba provádění úlohy j na stroji i
  - termín dostupnosti j *(release date)* r_j:
    nejdřívější čas, ve kterém může být úloha j prováděna
  - termín dokončení *(due date)* d_j:
    čas, do kdy by měla být úloha j nejpozději dokončena (preference)
    vs. *deadline*: čas, do kdy musí být úloha j nejpozději dokončena (požadavek)
  - váha w_j: důležitost úlohy j relativně vzhledem k ostatním úloham v systému
- Dynamické parametry úlohy
  - čas startu úlohy *(start time)* S_{ij}, S_j: čas zahájení provádění úlohy j na stroji i
  - čas konce úlohy *(completion time)* C_{ij}, C_j: čas, kdy je dokončeno provádění úlohy j na stroji i

Grahamova klasifikace

Grahamova klasifikace $\alpha|\beta|\gamma$

používá se pro popis rozvrhovacích problémů

- α: charakteristiky stroje
  - popisuje způsob alokace úloh na stroje
- β: charakteristiky úloh
  - popisuje omezení aplikovaná na úlohy
- γ: optimalizační kritéria

Grahamova klasifikace

Grahamova klasifikace $\alpha|\beta|\gamma$

používá se pro popis rozvrhovacích problémů

- α: charakteristiky stroje
  - popisuje způsob alokace úloh na stroje
- β: charakteristiky úloh
  - popisuje omezení aplikovaná na úlohy
- γ: optimalizační kritéria

`http://www.informatik.uni-osnabrueck.de/knust/class/`

- složitost pro jednotlivé rozvrhovací problémy

Grahamova klasifikace

Grahamova klasifikace $\alpha|\beta|\gamma$

používá se pro popis rozvrhovacích problémů

- α: charakteristiky stroje
  - popisuje způsob alokace úloh na stroje
- β: charakteristiky úloh
  - popisuje omezení aplikovaná na úlohy
- γ: optimalizační kritéria

`http://www.informatik.uni-osnabrueck.de/knust/class/`

- složitost pro jednotlivé rozvrhovací problémy

Příklady:

- $P3|prec|C_{\max}$: montáž kola
- $Pm|r_j|\sum w_j C_j$: paralelní stroje

Vlastnosti stroje α

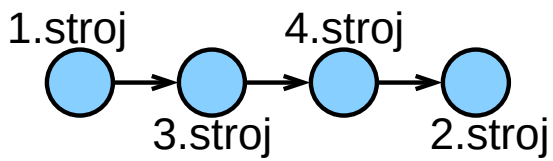
- Jeden stroj 1: $1|\ldots|\ldots$
- Identické paralelní stroje Pm
  - m identických strojů zapojených paralelně (se stejnou rychlostí)
  - úloha je dána jedinou operací
  - úloha může být prováděna na libovolném z m strojů
- Paralelní stroje s různou rychlostí Qm
  - doba trvání úlohy j na stroji i přímo závislá na jeho rychlosti v_i
  - $p_{ij} = p_j / v_i$
  - př. několik počítačů s různou rychlostí procesoru

Vlastnosti stroje α

- Jeden stroj 1: $1|\ldots|\ldots$
- Identické paralelní stroje Pm
  - m identických strojů zapojených paralelně (se stejnou rychlostí)
  - úloha je dána jedinou operací
  - úloha může být prováděna na libovolném z m strojů
- Paralelní stroje s různou rychlostí Qm
  - doba trvání úlohy j na stroji i přímo závislá na jeho rychlosti v_i
  - $p_{ij} = p_j/v_i$
  - př. několik počítačů s různou rychlostí procesoru
- Nezávislé paralelní stroje s různou rychlostí Rm
  - stroje mají různou rychlost pro různé úlohy
  - stroj i zpracovává úlohu j rychlostí v_{ij}
  - $p_{ij} = p_j/v_{ij}$
  - př. vektorový počítač počítá vektorové úlohy rychleji než klasické PC

Multi-operační *(shop)* problémy

- Multi-operační *(shop)* problémy
  - jedna úloha je prováděna postupně na několika strojích
    - úloha j se skládá z několika operací (i, j)
    - operace (i, j) úlohy j je prováděna na stroji i po dobu p_{ij}
    - příklad: úloha j se 4 operacemi $(1, j), (2, j), (3, j), (4, j)$

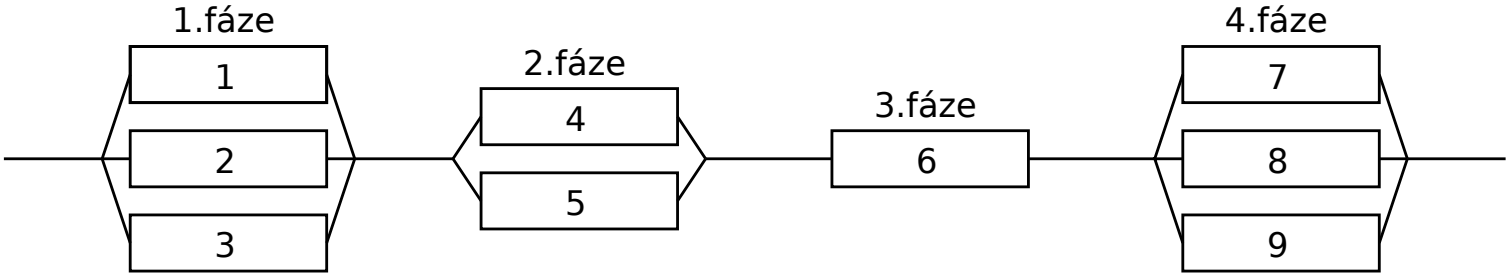


- Multi-operační problémy jsou klasické detailně studované problémy operačního výzkumu
- Reálné problémy ale často mnohem komplikovanější
  - využití znalostí o podproblémech nebo zjednodušených problémech a jejich řešicích metodách

Flow shop α

- *Flow shop Fm*
  - multi-operační problém s m stroji v sérii
  - každá úloha musí být prováděna na všech strojích
  - úloha musí být prováděna na všech strojích ve stejném pořadí
    - nejdříve se úloha provádí na 1. stroji, pak na 2., ...

- *Flow shop Fm*
  - multi-operační problém s m stroji v sérii
  - každá úloha musí být prováděna na všech strojích
  - úloha musí být prováděna na všech strojích ve stejném pořadí
    - nejdříve se úloha provádí na 1. stroji, pak na 2., ...
- *Flexible flow shop FFs*
  - zobecnění *flow shop* problému
  - s fází, každé fázi přísluší paralelní stroj
    - příklad: paralelní stroj 1.fáze: 1+2+3, paralelní stroj 2.fáze: 4+5, ...
  - tj. multi-operační problém s s paralelními stroji
  - úloha musí projít všemi fázemi ve stejném pořadí
    - nejprve se úloha provádí na paralelním stroji 1. fáze, pak na paralelním stroji 2. fáze, ...
  - na paralelním stroji příslušejícím dané fázi může být úloha prováděna na libovolném stroji



- *Job shop Jm*
  - multi-operační problém s m stroji
  - pořadí provádění operací pro každou úlohu je předem určeno
    - doba zpracování úlohy na některých strojích může být nulová
  - $(i,j) \rightarrow (k,j)$ určuje, že úloha j má být prováděna na stroji i dříve než na stroji k příklad: $(2,j) \rightarrow (1,j) \rightarrow (3,j) \rightarrow (4,j)$

- *Job shop Jm*
  - multi-operační problém s m stroji
  - pořadí provádění operací pro každou úlohu je předem určeno
    - doba zpracování úlohy na některých strojích může být nulová
  - $(i,j) \rightarrow (k,j)$ určuje, že úloha j má být prováděna na stroji i dříve než na stroji k příklad: $(2,j) \rightarrow (1,j) \rightarrow (3,j) \rightarrow (4,j)$
- *Open shop Om*
  - multi-operační problém s m stroji
  - doba zpracování úlohy na některých strojích může být nulová
  - rozvrhovač určí, v jakém pořadí je úloha prováděna na strojích

Omezení β

- Precedenční podmínky *prec*
  - lineární posloupnost, stromová struktura
  - pro úlohy a, b píšeme $a \rightarrow b$, což znamená $S_a + p_a \leq S_b$
  - příklad: montáž kola
- Přerušení úlohy *(preemptions)* *pmtn*
  - při příchodu úlohy s vyšší prioritou je současná úloha přerušena
- Vhodnost stroje M_j
  - podmnožina strojů M_j, na níž lze provádět úlohu j
  - přiřazení místností: postačující velikost učebny
  - hry: počítač s HW grafickou knihovnou
- Omezení na pracovní sílu W, W_l
  - do problému zavedeme další typ zdroje
  - stroje mohou potřebovat operátory a úlohy lze provádět jen tehdy, pokud jsou dostupní W operátorů
  - mohou existovat různé skupiny operátorů se specifickou kvalifikací W_l je počet operátorů ve skupině l

Omezení (pokračování) β

- Směrovací *(routing)* omezení
  - udávají, na kterých strojích musí být úloha prováděna
  - pořadí provádění úlohy v multi-operačních problémech
    - job shop problém: pořadí operací předem stanoveno
    - open shop problém: pořadí operací úlohy (*route for the job*) stanoveno až při rozvrhování
- Nastavovací *(setup)* doba a cena $s_{ijk}, c_{ijk}, s_{jk}, c_{jk}$
  - závislé na posloupnosti provádění
  - s_{ijk} čas nutný pro provádění úlohy k po úloze j na stroji i
  - c_{ijk} cena nutná pro provádění úlohy k po úloze j na stroji i
  - s_{jk}, c_{jk} čas/cena nezávislý na stroji
  - příklady
    - problém obchodního cestujícího $1|s_{jk}|C_{\max}$

Omezení (pokračování) β

- Výroba na objednávku a na sklad
  - výroba zboží na sklad, pokud je u něj záruka spotřeby
    nutno uvážit cenu za skladování
  - výroba zboží na objednávku vynucuje úvahu termínů dokončení
    vyprodukované množství závislé na zákazníkovi
- Skladovací prostor a doba čekání při výrobě
  - omezené množství prostoru při výrobě
  - horní hranice počtu úloh čekajících ve frontě na stroj

Omezení (pokračování) β

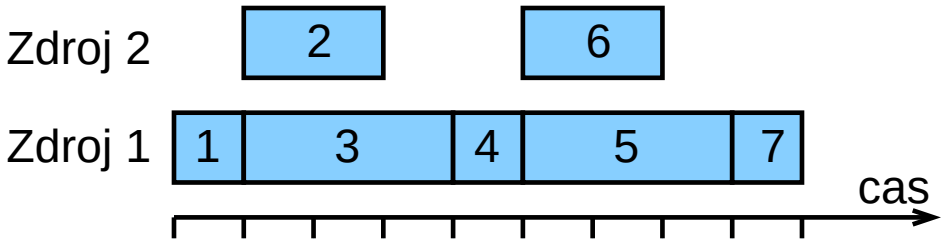
- Výroba na objednávku a na sklad
  - výroba zboží na sklad, pokud je u něj záruka spotřeby
    nutno uvážit cenu za skladování
  - výroba zboží na objednávku vynucuje úvahu termínů dokončení
    vyprodukované množství závislé na zákazníkovi
- Skladovací prostor a doba čekání při výrobě
  - omezené množství prostoru při výrobě
  - horní hranice počtu úloh čekajících ve frontě na stroj
  - blokování: úloha je zablokována na současném stroji, protože fronta na následujícím stroji je plná
- ...

Optimalizace: výkon a *makespan* γ

- Makespan $\mathbf{C_{max}}$: maximální čas konce úloh

$$\mathbf{C_{max}} = \max(C_1, \ldots, C_n)$$

- Příklad: $C_{max} = \max\{1, 3, 4, 5, 8, 7, 9\} = 9$

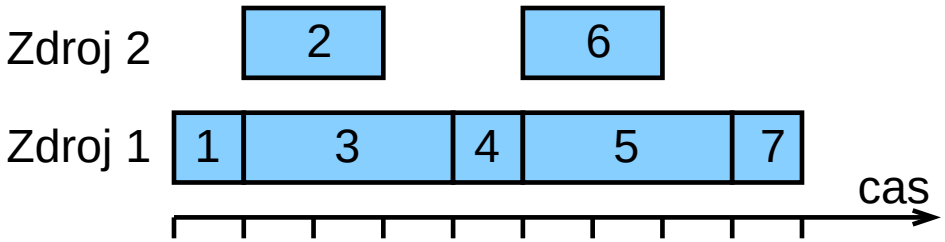


Optimalizace: výkon a *makespan* γ

- Makespan $\mathbf{C_{max}}$: maximální čas konce úloh

$$\mathbf{C_{max}} = \max(C_1, \ldots, C_n)$$

- Příklad: $C_{max} = \max\{1, 3, 4, 5, 8, 7, 9\} = 9$



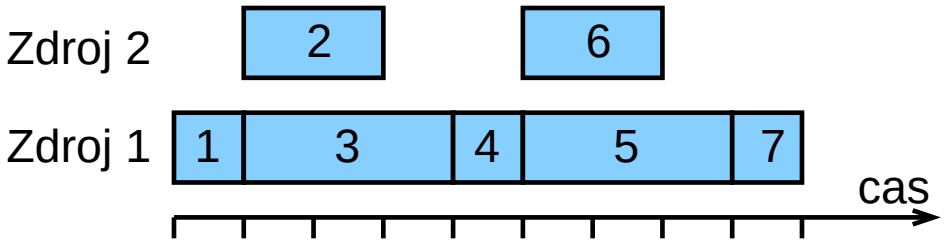
- Cíl: minimalizace makespan často
  - maximalizuje výkon (*throughput*)
  - zajišťuje rovnoměrné zatížení strojů (*load balancing*)
  - příklad:

Optimalizace: výkon a *makespan* γ

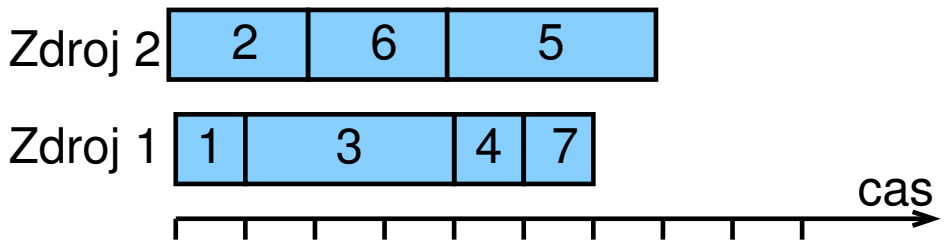
- Makespan $\mathbf{C_{max}}$: maximální čas konce úloh

$$\mathbf{C_{max}} = \max(C_1, \ldots, C_n)$$

- Příklad: $C_{max} = \max\{1, 3, 4, 5, 8, 7, 9\} = 9$



- Cíl: minimalizace makespan často
  - maximalizuje výkon (*throughput*)
  - zajišťuje rovnoměrné zatížení strojů (*load balancing*)
  - příklad: $C_{max} = \max\{1, 2, 4, 5, 7, 4, 6\} = 7$



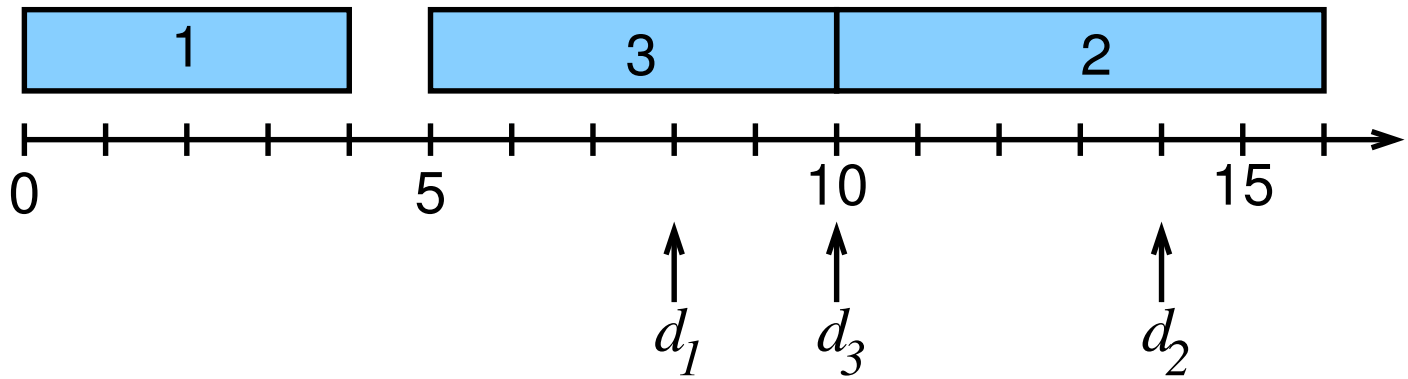
- Velmi často používané a základní kritérium

Optimalizace: zpoždění γ

- Zpoždění *(lateness)* úlohy j: $L_j = C_j - d_j$
- Maximální zpoždění $\mathbf{L_{max}}$

$$\mathbf{L_{max}} = \max(L_1, \ldots, L_n)$$

- Cíl: minimalizace maximálního zpoždění
- Příklad:



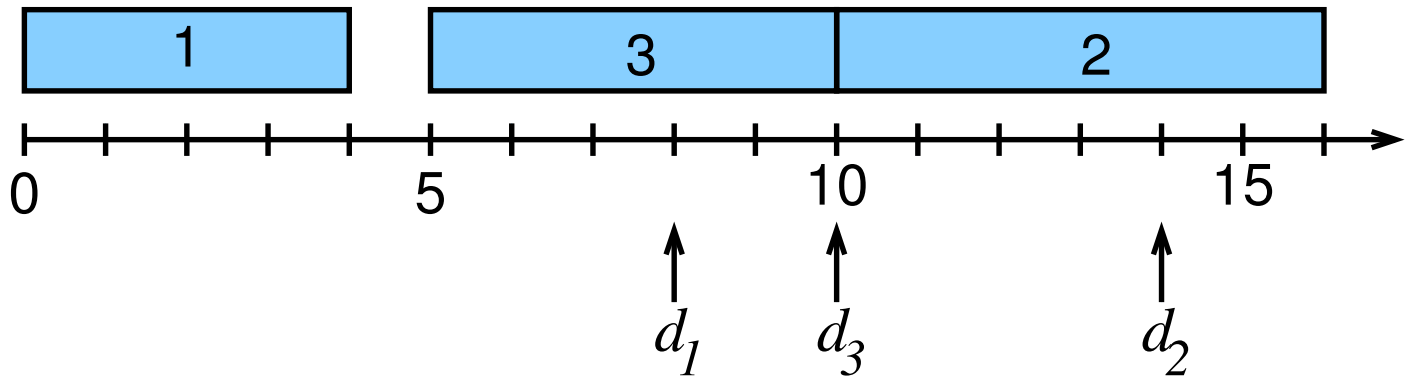
$$L_{max} = \max(L_1, L_2, L_3) =$$

Optimalizace: zpoždění γ

- Zpoždění *(lateness)* úlohy j: $L_j = C_j - d_j$
- Maximální zpoždění $\mathbf{L_{max}}$

$$\mathbf{L_{max}} = \max(L_1, \ldots, L_n)$$

- Cíl: minimalizace maximálního zpoždění
- Příklad:



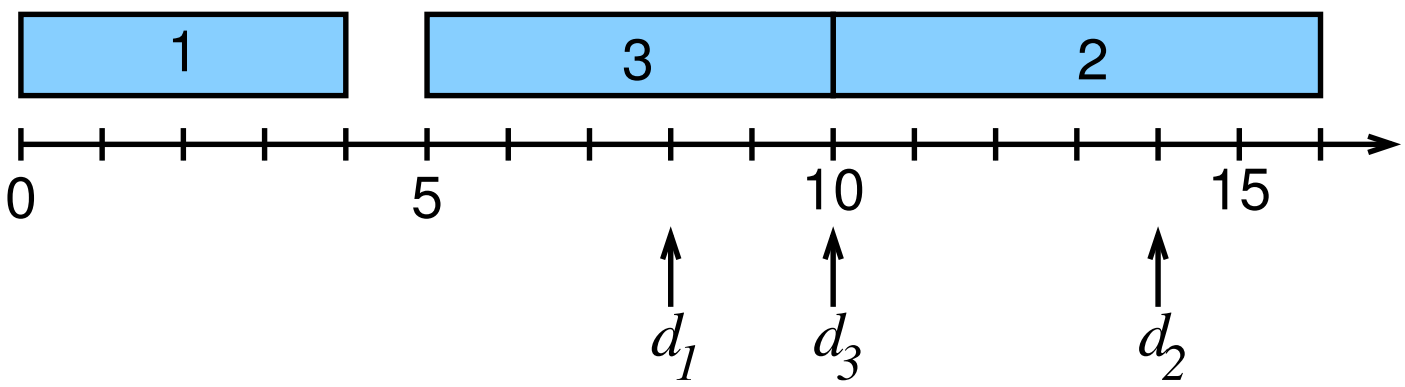
$$\begin{aligned} L_{max} &= \max(L_1, L_2, L_3) = \\ &= \max(C_1 - d_1, C_2 - d_2, C_3 - d_3) = \\ &= \max(4 - 8, 16 - 14, 10 - 10) = \\ &= \max(-4, 2, 0) = 2 \end{aligned}$$

Optimalizace: nezáporné zpoždění γ

- Nezáporné zpoždění *(tardiness)* úlohy j: $T_j = \max(C_j - d_j, 0)$
- Cíl: minimalizace celkového zpoždění

$$\sum_{j=1}^{n} T_j$$ celkové zpoždění

- Příklad:



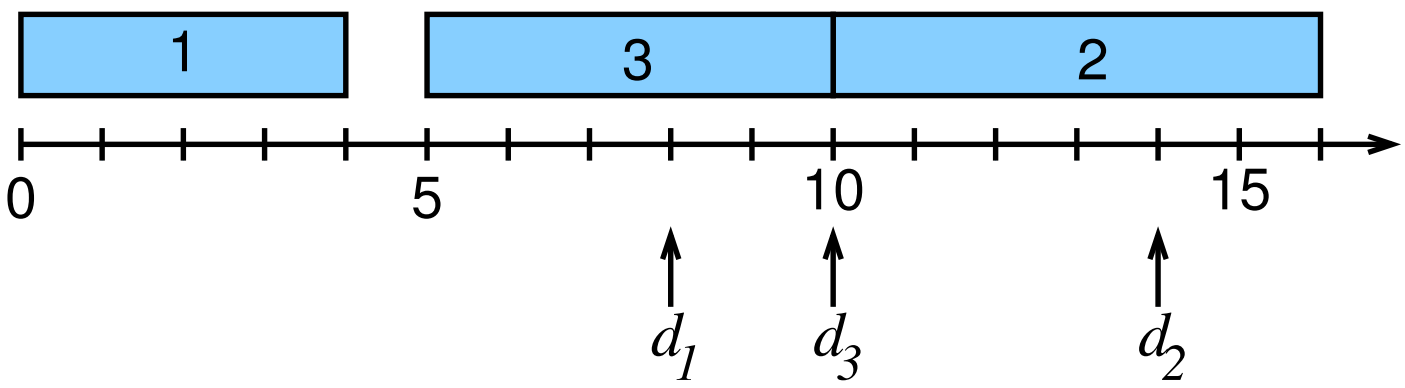
$T_1 + T_2 + T_3 =$

Optimalizace: nezáporné zpoždění γ

- Nezáporné zpoždění *(tardiness)* úlohy j: $T_j = \max(C_j - d_j, 0)$
- Cíl: minimalizace celkového zpoždění

$$\sum_{j=1}^{n} T_j$$ celkové zpoždění

- Příklad:



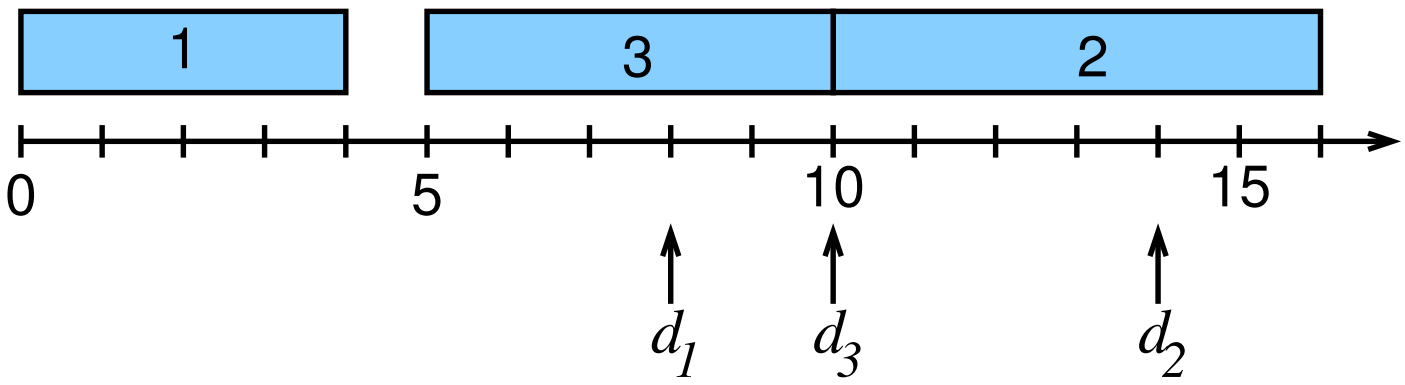
$$T_1 + T_2 + T_3 = \max(C_1 - d_1, 0) + \max(C_2 - d_2, 0) + \max(C_3 - d_3, 0) =$$
$$\max(4-8, 0) + \max(16-14, 0) + \max(10-10, 0) = 0+2+0 = 2$$

Optimalizace: nezáporné zpoždění γ

- Nezáporné zpoždění *(tardiness)* úlohy j: $T_j = \max(C_j - d_j, 0)$
- Cíl: minimalizace celkového zpoždění

$$\sum_{j=1}^{n} T_j$$ celkové zpoždění

- Příklad:

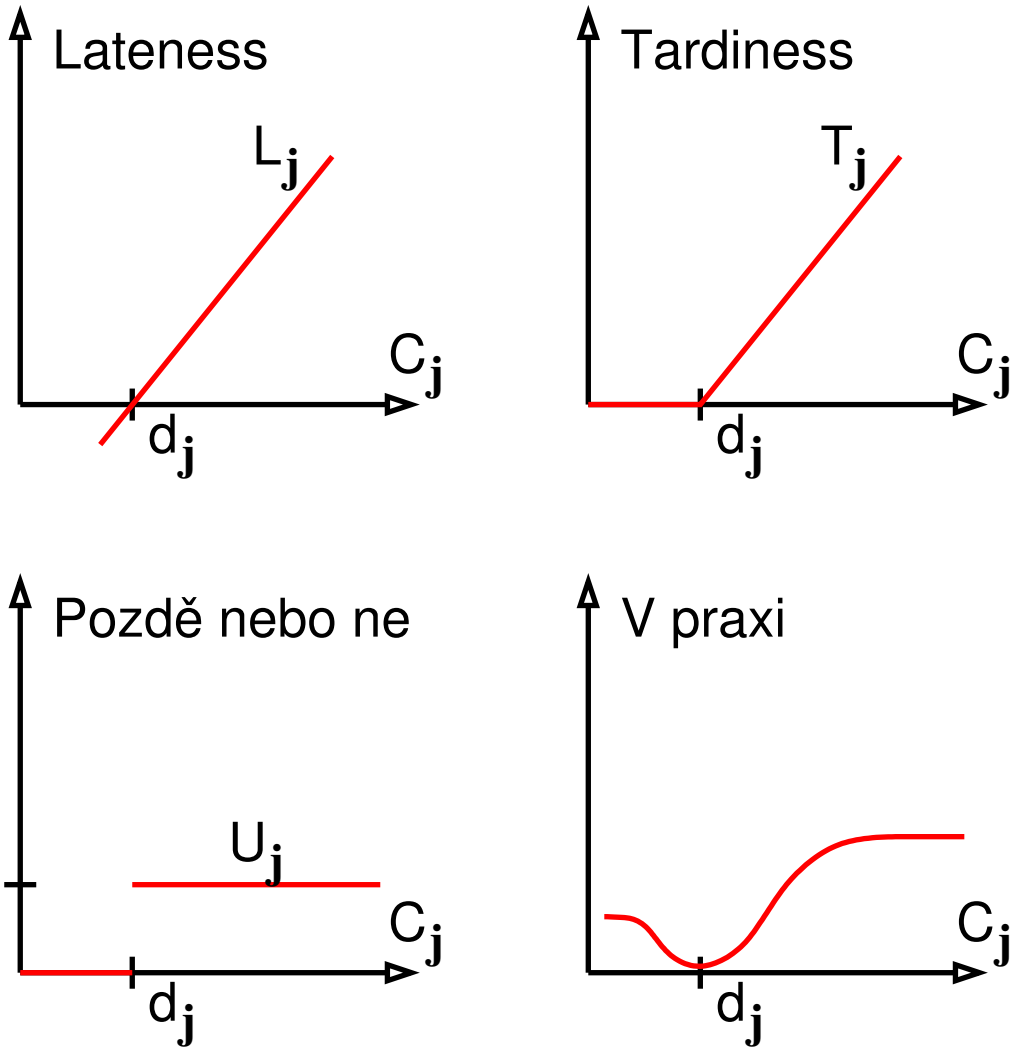


$$T_1 + T_2 + T_3 = \max(C_1 - d_1, 0) + \max(C_2 - d_2, 0) + \max(C_3 - d_3, 0) =$$
$$\max(4-8, 0) + \max(16-14, 0) + \max(10-10, 0) = 0+2+0 = 2$$

- Cíl: minimalizace celkového váženého zpoždění

$$\sum_{j=1}^{n} w_j T_j$$ celkové vážené zpoždění

Termín dokončení a grafy γ



Optimalizace: skladování při výrobě γ

- Cena za skladování vyrobeného zboží
- Cena za skladování při výrobě
  (Work-In-Process inventory cost)
  - příliš velké množství právě vyráběneho zboží může zaplnit linku
  - příliš dlouho odložené zboží může být znehodnoceno

Optimalizace: skladování při výrobě γ

- Cena za skladování vyrobeného zboží
- Cena za skladování při výrobě
  (Work-In-Process inventory cost)
  - příliš velké množství právě vyráběneho zboží může zaplnit linku
  - příliš dlouho odložené zboží může být znehodnoceno
- Délka skladování při výrobě svázána s časy konce úloh

  $\Rightarrow$ minimalizace součtu časů konců úloh

$$\sum_{j=1}^{n} C_j$$

Optimalizace: skladování při výrobě γ

- Cena za skladování vyrobeného zboží
- Cena za skladování při výrobě
  (Work-In-Process inventory cost)
  - příliš velké množství právě vyráběneho zboží může zaplnit linku
  - příliš dlouho odložené zboží může být znehodnoceno
- Délka skladování při výrobě svázána s časy konce úloh

  $\Rightarrow$ minimalizace součtu časů konců úloh

$$\sum_{j=1}^{n} C_j$$

  $\Rightarrow$ minimalizace váženého součtu časů konců úloh

$$\sum_{j=1}^{n} w_j C_j$$

  celková hodnota daná skladováním při výrobě

Další optimalizační kritéria γ

- Robustnost
  - robustnější rozvrh vyžaduje méně změn při změně problému (porucha stroje, dopravní špička)
- Cena za nastavení *(setup)*
  - cena za připravení letadla na odlet
    (čištění, zásobování, doplnění pohonných hmot)
- Cena za pracovní sílu
  - cena za přiřazení zaměstnanců na konkrétní směnu

V problému často řada optimalizačních kritérií

- multi-kriteriální rozvrhování
  - *Pareto* optimalizace
- žádoucí vztah mezi nimi nemusí být jasně definovaný
  - co je důležitější?
- ani samotná kritéria nemusí být jasně definována
  - jak daný požadavek reprezentovat kritériem?

Polynomiální a nedeterministicky polynomiální problémy

- Polynomiální problémy
  - existuje algoritmus polynomiální složitosti pro řešení problému
- NP a NP-úplné problémy
  - řešitelné nedeterministickým polynomiálním algoritmem
  - potenciální řešení lze ověřit v polynomiálním čase
  - v nejhorším případě exponenciální složitost (pokud neplatí P=NP)
  - NP-úplný problém

Polynomiální a nedeterministicky polynomiální problémy

- Polynomiální problémy
  - existuje algoritmus polynomiální složitosti pro řešení problému
- NP a NP-úplné problémy
  - řešitelné nedeterministickým polynomiálním algoritmem
  - potenciální řešení lze ověřit v polynomiálním čase
  - v nejhorším případě exponenciální složitost (pokud neplatí P=NP)
  - NP-úplný problém
    - libovolný problém v NP se na něj dá polynomiálně redukovat
- Příklady:
  - Polynomiální
    - $1||L_{\max}$ $P|pmtn|C_{\max}$ $1||\sum w_j C_j$
  - NP
    - $1|r_j|L_{\max}$ $P2||C_{\max}$ $P2||\sum w_j C_j$

Řídící pravidla

16. května 2022

Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2

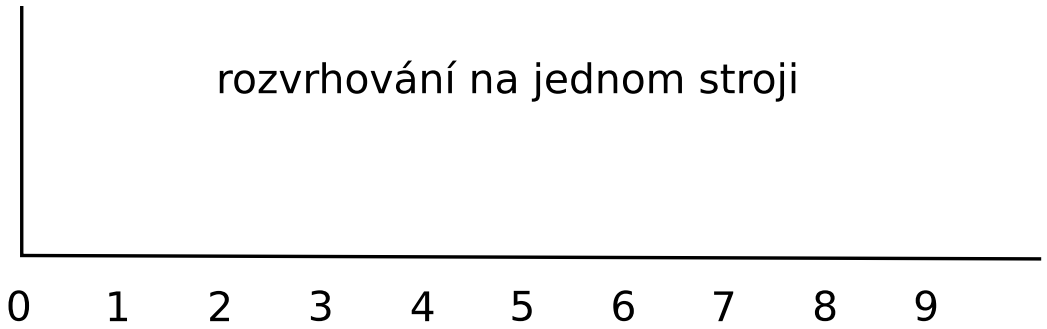
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



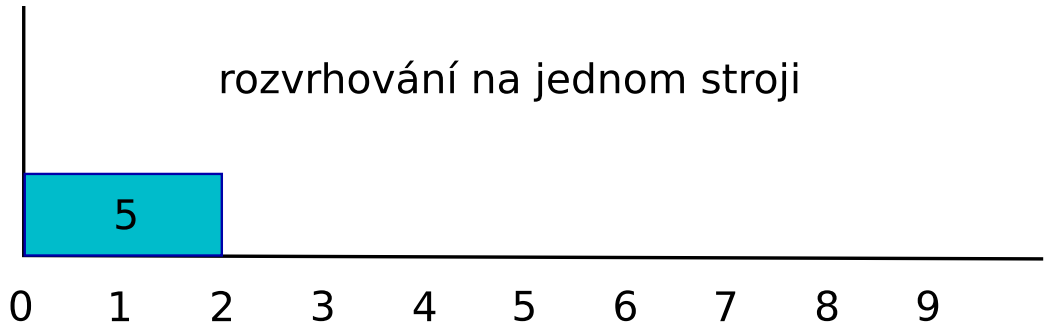
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



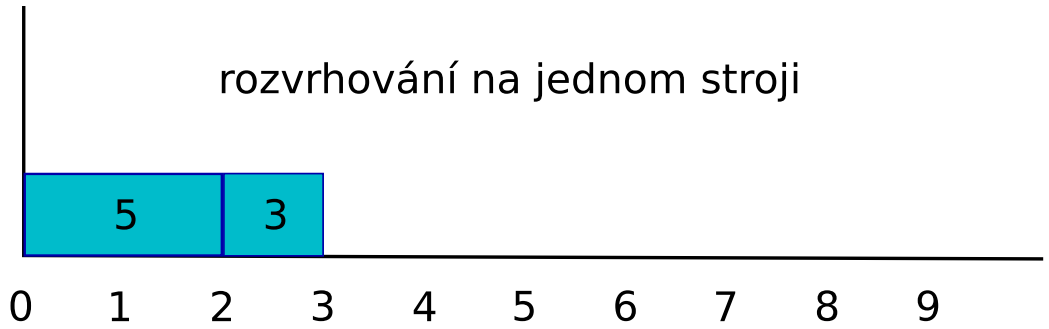
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



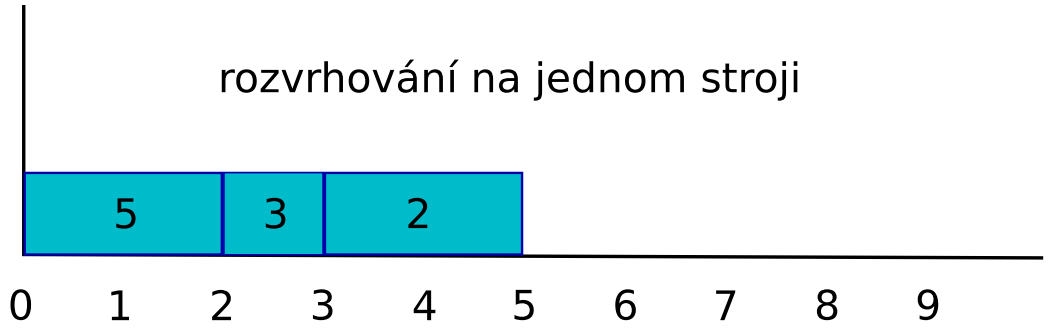
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



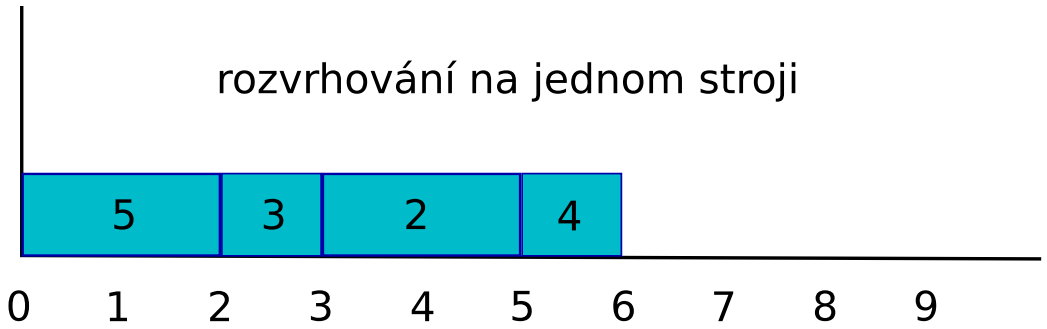
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



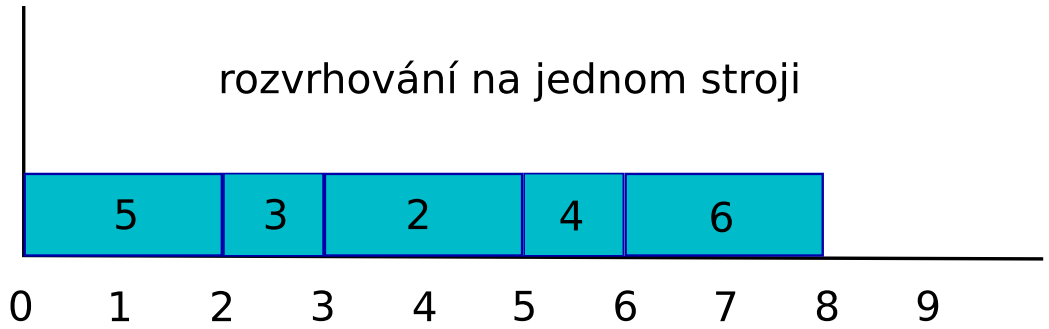
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



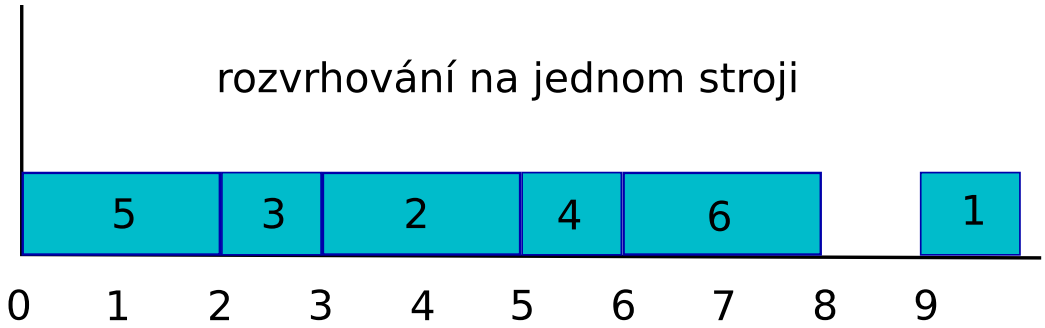
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



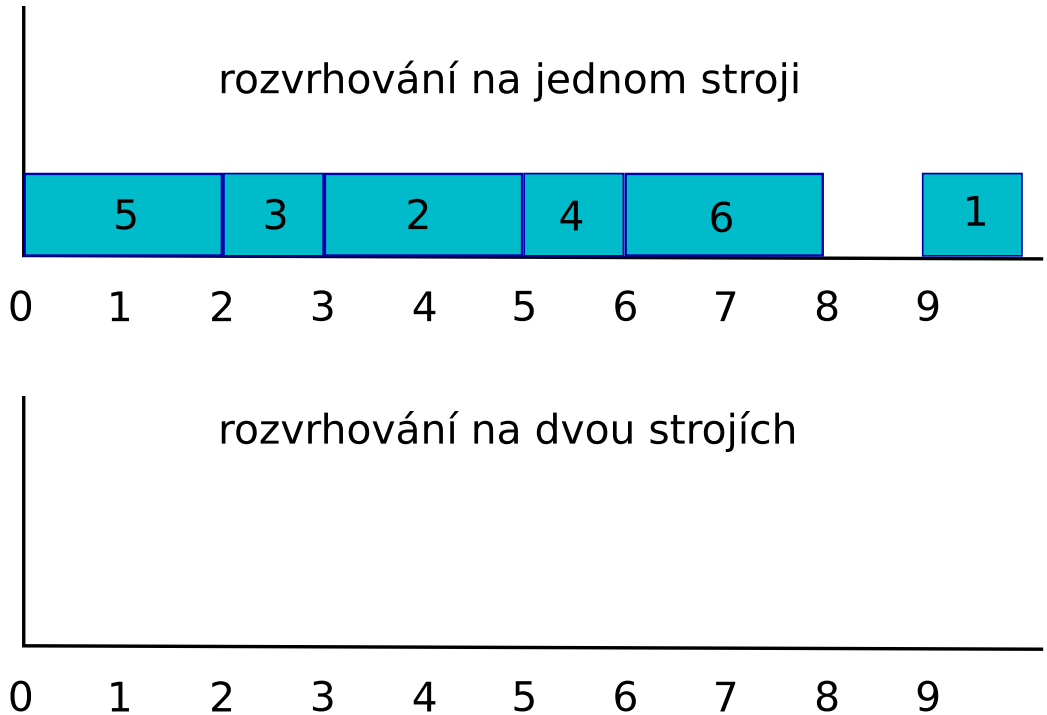
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



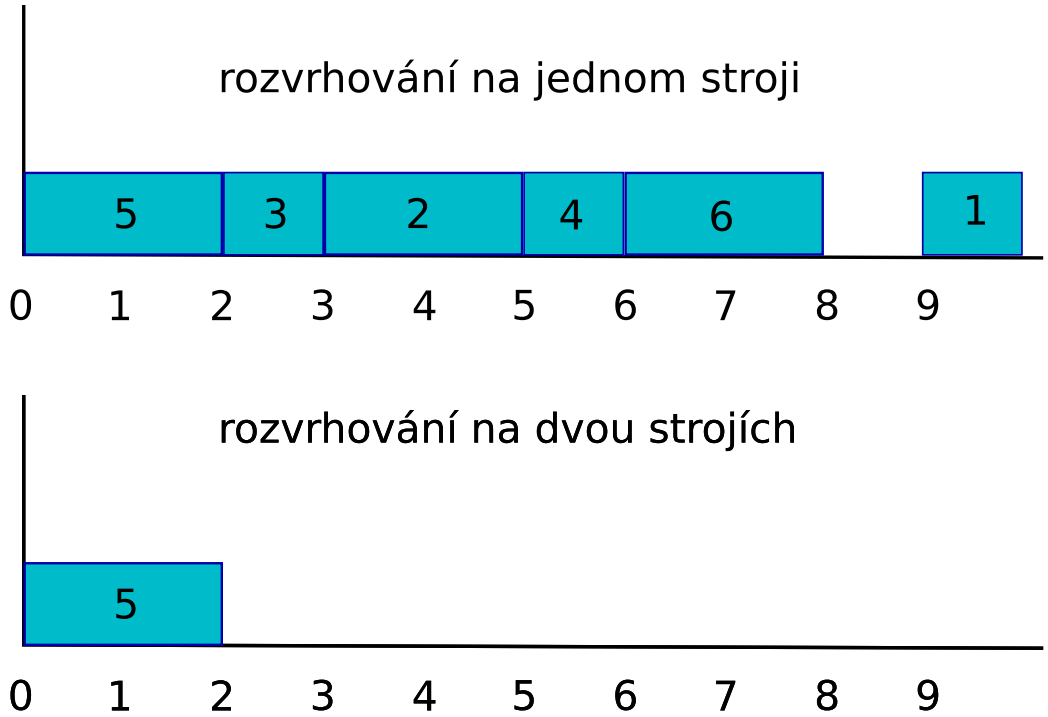
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



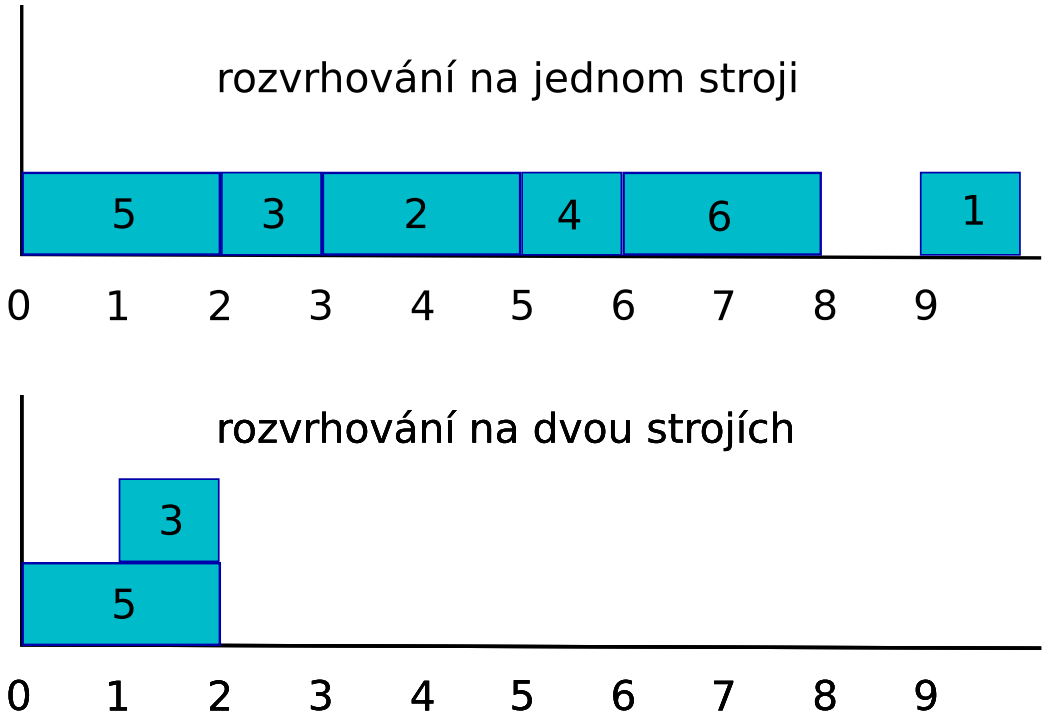
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



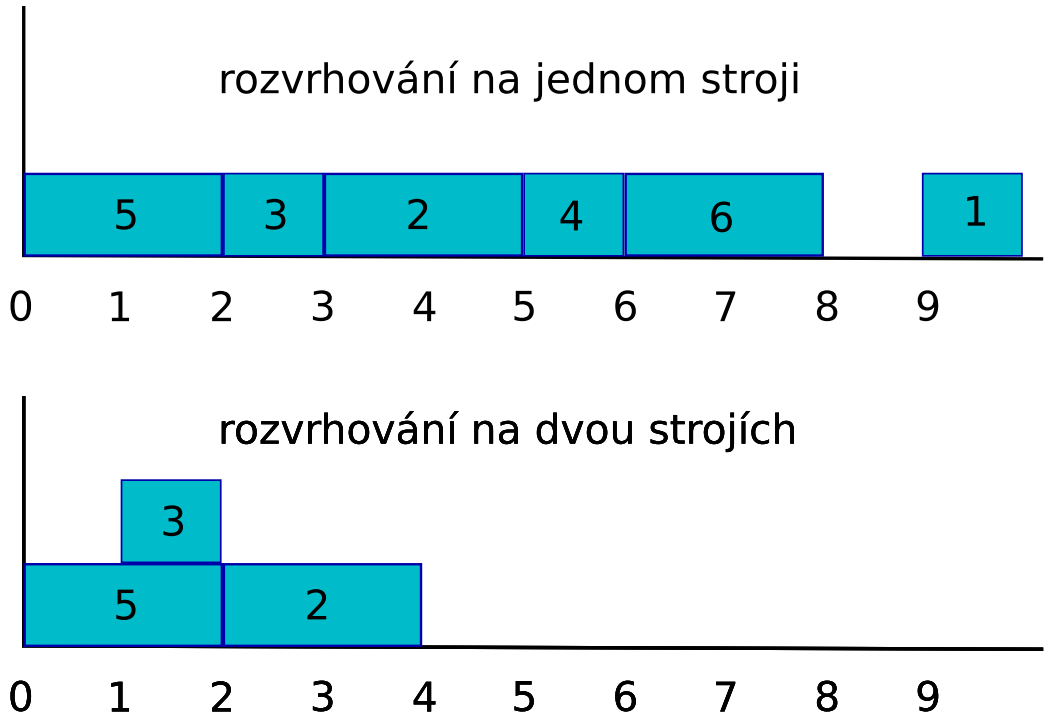
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



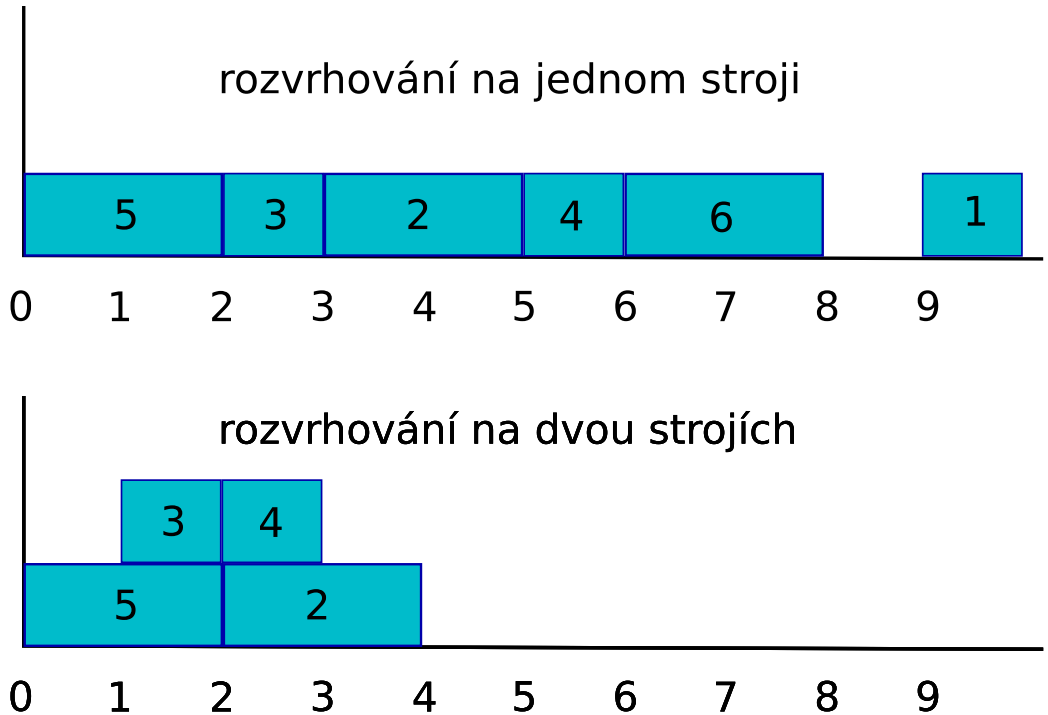
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



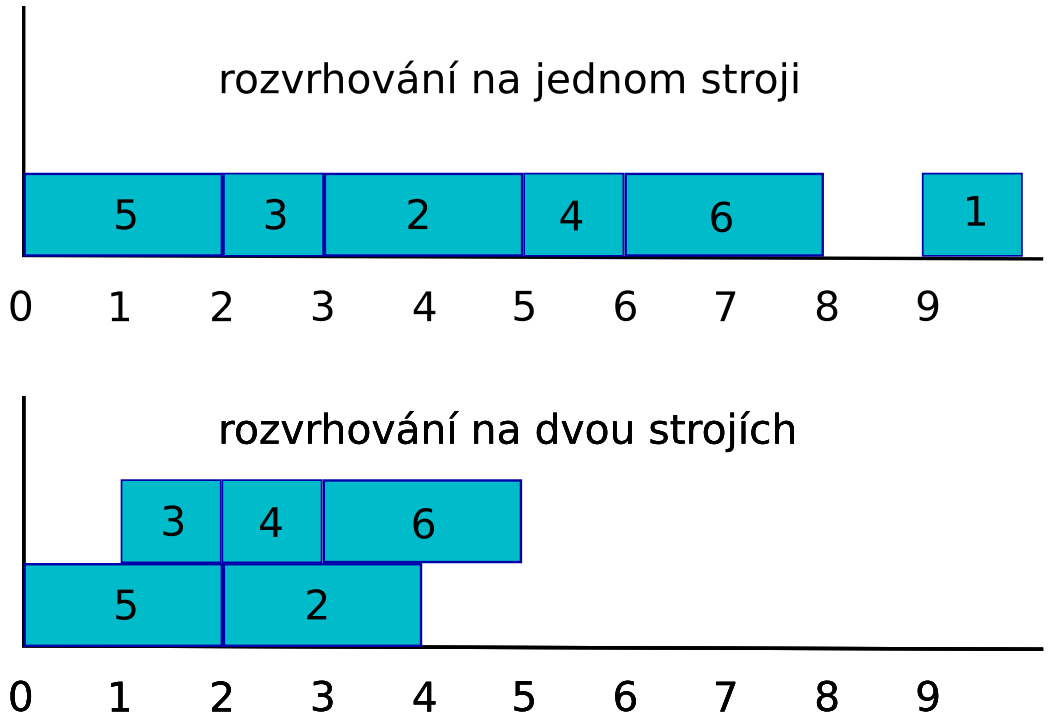
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



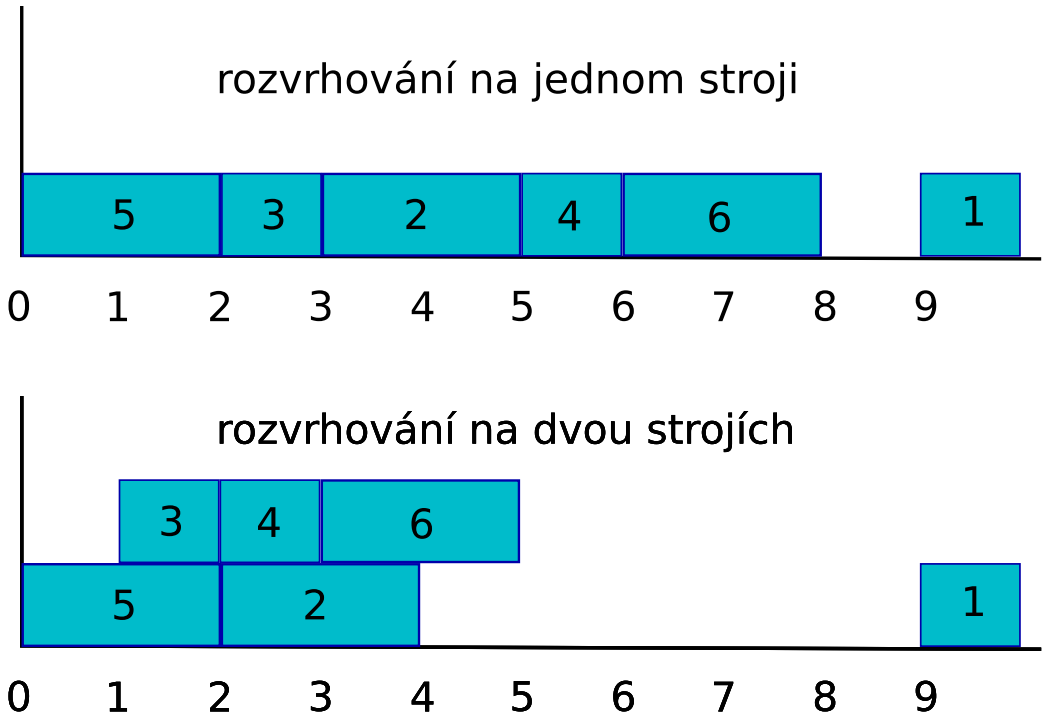
Řídící pravidla *(dispatching rules)*

Řídící pravidlo

- určuje pořadí (prioritu), ve kterém mají být úlohy prováděny
  - pokud má více úloh stejnou prioritu, úlohy jsou seřazeny náhodně (nebo např. dle čísla úlohy)
- jakmile se některý stroj uvolní, je vybrána nejprioritnější úloha

Příklad: použijte pro seřazení úloh nejdřívější termín dostupnosti r_j

úlohy	1	2	3	4	5	6
r_j	9	2	1	2	0	3
p_j	1	2	1	1	2	2



Pravidla s termíny dostupnosti r_j a dokončení d_j

- Nejdřívější termín dostupnosti *(Earliest Release Date first ERD)*
  - ekvivalentní nejdříve-příjde-nejdříve-obsloužen *(First-Come-First-Serve)*
  - minimalizuje odlišnosti v době čekání na stroji

Pravidla s termíny dostupnosti r_j a dokončení d_j

- Nejdřívější termín dostupnosti *(Earliest Release Date first ERD)*
  - ekvivalentní nejdříve-příjde-nejdříve-obsloužen *(First-Come-First-Serve)*
  - minimalizuje odlišnosti v době čekání na stroji

- Nejdřívější termín dokončení *(Earliest Due Date first EDD)*
  - směřuje k minimalizaci maximálního zpoždění mezi čekajícími úlohami
  - optimální pro $1||L_{max}$ (všechny úlohy dostupné na začátku)
  - pozor, i zde (zejména stejně jako u všech pravidel) musíme brát v úvahu termín dostupnosti, tj. úlohu lze plánovat teprve když je dostupná!!!
    - př. $r_2 = 3, d_2 = 5, \quad r_3 = 0, d_3 = 6$ – dříve plánujeme úlohu 3

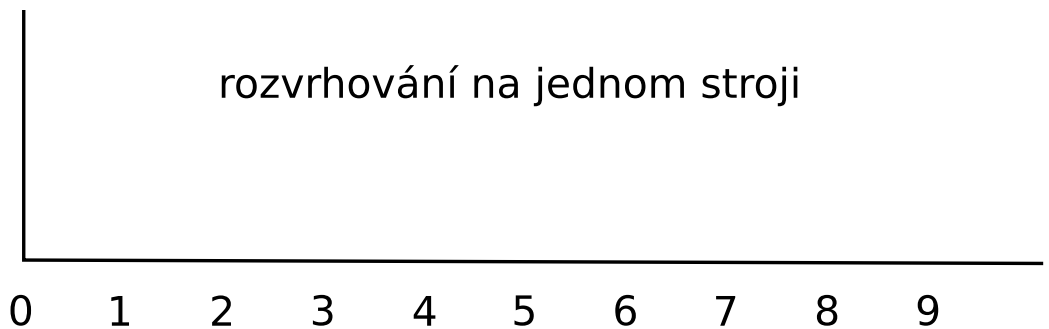
úlohy	1	2	3	4	5	6
r_j	9	3	0	2	0	6
p_j	1	2	1	1	2	2
d_j	10	5	6	9	2	8

Pravidla s termíny dostupnosti r_j a dokončení d_j

- Nejdřívější termín dostupnosti *(Earliest Release Date first ERD)*
  - ekvivalentní nejdříve-příjde-nejdříve-obsloužen *(First-Come-First-Serve)*
  - minimalizuje odlišnosti v době čekání na stroji

- Nejdřívější termín dokončení *(Earliest Due Date first EDD)*
  - směřuje k minimalizaci maximálního zpoždění mezi čekajícími úlohami
  - optimální pro $1||L_{max}$ (všechny úlohy dostupné na začátku)
  - pozor, i zde (zejména stejně jako u všech pravidel) musíme brát v úvahu termín dostupnosti, tj. úlohu lze plánovat teprve když je dostupná!!!
    - př. $r_2 = 3, d_2 = 5, \quad r_3 = 0, d_3 = 6$ – dříve plánujeme úlohu 3

úlohy	1	2	3	4	5	6
r_j	9	3	0	2	0	6
p_j	1	2	1	1	2	2
d_j	10	5	6	9	2	8

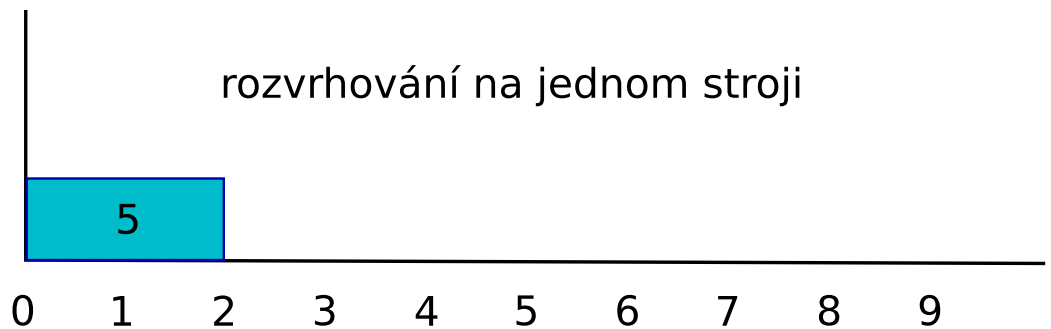


Pravidla s termíny dostupnosti r_j a dokončení d_j

- Nejdřívější termín dostupnosti *(Earliest Release Date first ERD)*
  - ekvivalentní nejdříve-příjde-nejdříve-obsloužen *(First-Come-First-Serve)*
  - minimalizuje odlišnosti v době čekání na stroji

- Nejdřívější termín dokončení *(Earliest Due Date first EDD)*
  - směřuje k minimalizaci maximálního zpoždění mezi čekajícími úlohami
  - optimální pro $1||L_{max}$ (všechny úlohy dostupné na začátku)
  - pozor, i zde (zejména stejně jako u všech pravidel) musíme brát v úvahu termín dostupnosti, tj. úlohu lze plánovat teprve když je dostupná!!!
    - př. $r_2 = 3, d_2 = 5, \quad r_3 = 0, d_3 = 6$ – dříve plánujeme úlohu 3

úlohy	1	2	3	4	5	6
r_j	9	3	0	2	0	6
p_j	1	2	1	1	2	2
d_j	10	5	6	9	2	8

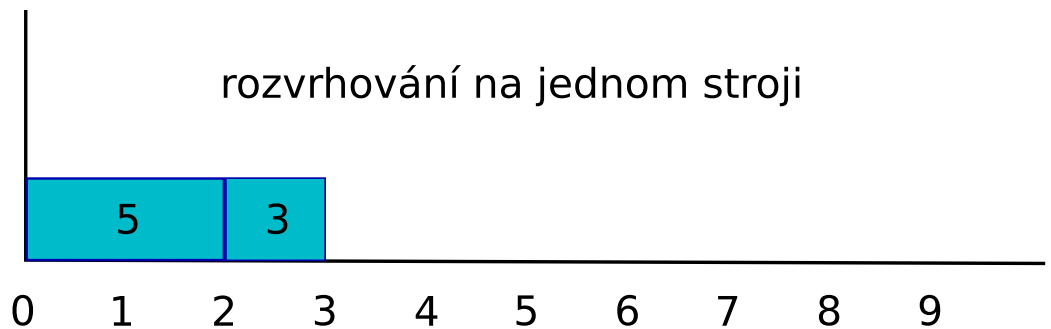


Pravidla s termíny dostupnosti r_j a dokončení d_j

- Nejdřívější termín dostupnosti *(Earliest Release Date first ERD)*
  - ekvivalentní nejdříve-příjde-nejdříve-obsloužen *(First-Come-First-Serve)*
  - minimalizuje odlišnosti v době čekání na stroji

- Nejdřívější termín dokončení *(Earliest Due Date first EDD)*
  - směřuje k minimalizaci maximálního zpoždění mezi čekajícími úlohami
  - optimální pro $1||L_{max}$ (všechny úlohy dostupné na začátku)
  - pozor, i zde (zejména stejně jako u všech pravidel) musíme brát v úvahu termín dostupnosti, tj. úlohu lze plánovat teprve když je dostupná!!!
    - př. $r_2 = 3, d_2 = 5, \quad r_3 = 0, d_3 = 6$ – dříve plánujeme úlohu 3

úlohy	1	2	3	4	5	6
r_j	9	3	0	2	0	6
p_j	1	2	1	1	2	2
d_j	10	5	6	9	2	8

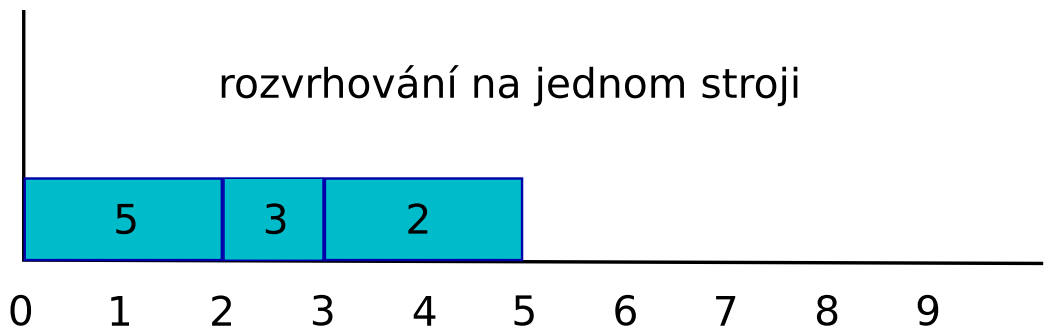


Pravidla s termíny dostupnosti r_j a dokončení d_j

- Nejdřívější termín dostupnosti *(Earliest Release Date first ERD)*
  - ekvivalentní nejdříve-příjde-nejdříve-obsloužen *(First-Come-First-Serve)*
  - minimalizuje odlišnosti v době čekání na stroji

- Nejdřívější termín dokončení *(Earliest Due Date first EDD)*
  - směřuje k minimalizaci maximálního zpoždění mezi čekajícími úlohami
  - optimální pro $1||L_{max}$ (všechny úlohy dostupné na začátku)
  - pozor, i zde (zejména stejně jako u všech pravidel) musíme brát v úvahu termín dostupnosti, tj. úlohu lze plánovat teprve když je dostupná!!!
    - př. $r_2 = 3, d_2 = 5, \quad r_3 = 0, d_3 = 6$ – dříve plánujeme úlohu 3

úlohy	1	2	3	4	5	6
r_j	9	3	0	2	0	6
p_j	1	2	1	1	2	2
d_j	10	5	6	9	2	8

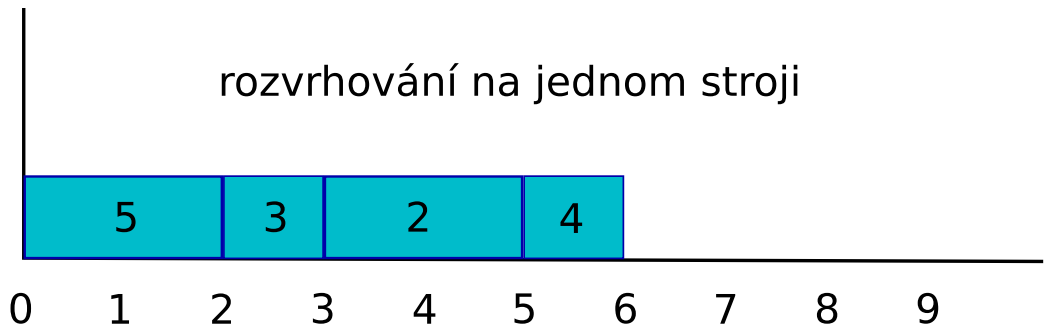


Pravidla s termíny dostupnosti r_j a dokončení d_j

- Nejdřívější termín dostupnosti *(Earliest Release Date first ERD)*
  - ekvivalentní nejdříve-příjde-nejdříve-obsloužen *(First-Come-First-Serve)*
  - minimalizuje odlišnosti v době čekání na stroji

- Nejdřívější termín dokončení *(Earliest Due Date first EDD)*
  - směřuje k minimalizaci maximálního zpoždění mezi čekajícími úlohami
  - optimální pro $1||L_{max}$ (všechny úlohy dostupné na začátku)
  - pozor, i zde (zejména stejně jako u všech pravidel) musíme brát v úvahu termín dostupnosti, tj. úlohu lze plánovat teprve když je dostupná!!!
    - př. $r_2 = 3, d_2 = 5, \quad r_3 = 0, d_3 = 6$ – dříve plánujeme úlohu 3

úlohy	1	2	3	4	5	6
r_j	9	3	0	2	0	6
p_j	1	2	1	1	2	2
d_j	10	5	6	9	2	8

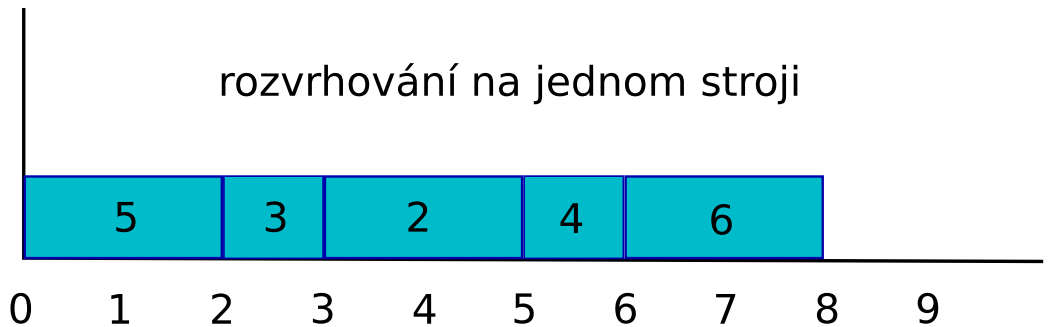


Pravidla s termíny dostupnosti r_j a dokončení d_j

- Nejdřívější termín dostupnosti *(Earliest Release Date first ERD)*
  - ekvivalentní nejdříve-příjde-nejdříve-obsloužen *(First-Come-First-Serve)*
  - minimalizuje odlišnosti v době čekání na stroji

- Nejdřívější termín dokončení *(Earliest Due Date first EDD)*
  - směřuje k minimalizaci maximálního zpoždění mezi čekajícími úlohami
  - optimální pro $1||L_{max}$ (všechny úlohy dostupné na začátku)
  - pozor, i zde (zejména stejně jako u všech pravidel) musíme brát v úvahu termín dostupnosti, tj. úlohu lze plánovat teprve když je dostupná!!!
    - př. $r_2 = 3, d_2 = 5, \quad r_3 = 0, d_3 = 6$ – dříve plánujeme úlohu 3

úlohy	1	2	3	4	5	6
r_j	9	3	0	2	0	6
p_j	1	2	1	1	2	2
d_j	10	5	6	9	2	8

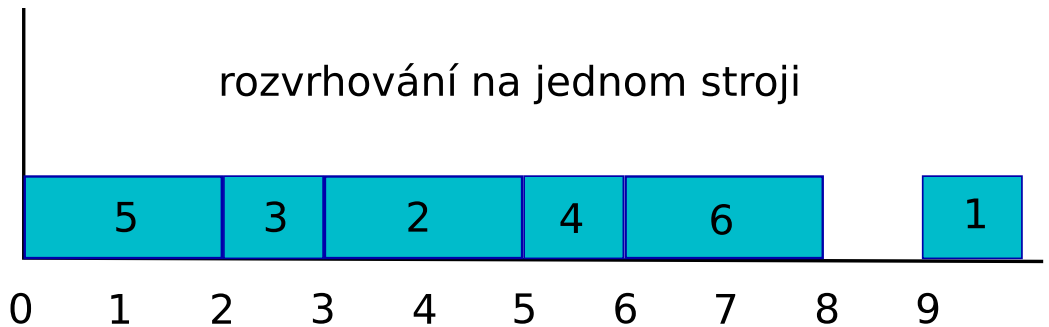


Pravidla s termíny dostupnosti r_j a dokončení d_j

- Nejdřívější termín dostupnosti *(Earliest Release Date first ERD)*
  - ekvivalentní nejdříve-příjde-nejdříve-obsloužen *(First-Come-First-Serve)*
  - minimalizuje odlišnosti v době čekání na stroji

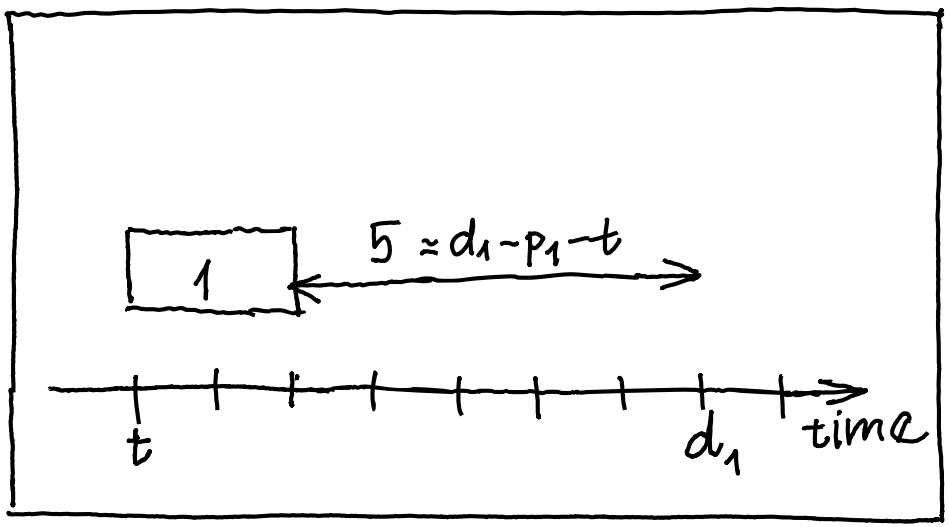
- Nejdřívější termín dokončení *(Earliest Due Date first EDD)*
  - směřuje k minimalizaci maximálního zpoždění mezi čekajícími úlohami
  - optimální pro $1||L_{max}$ (všechny úlohy dostupné na začátku)
  - pozor, i zde (zejména stejně jako u všech pravidel) musíme brát v úvahu termín dostupnosti, tj. úlohu lze plánovat teprve když je dostupná!!!
    - př. $r_2 = 3, d_2 = 5, \quad r_3 = 0, d_3 = 6$ – dříve plánujeme úlohu 3

úlohy	1	2	3	4	5	6
r_j	9	3	0	2	0	6
p_j	1	2	1	1	2	2
d_j	10	5	6	9	2	8



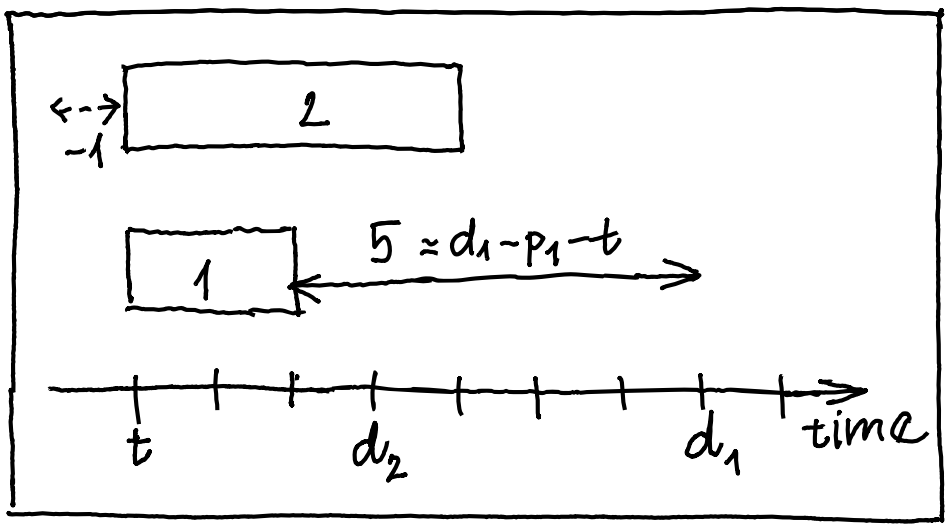
Pravidla s termíny dostupnosti: minimální rezerva

- Minimální rezerva *(Minimum Slack first MS)*
  - $\max(d_j - p_j - t, 0)$
    - d_j termín dokončení
    - p_j doba provádění
    - t aktuální čas
  - funkci max používáme, abychom neměli záporné hodnoty pro úlohy, které už to určitě nestihnou
  - minimalizace kriterií svázaných s termínem dokončení,
    tj. maximální zpoždění



Pravidla s termíny dostupnosti: minimální rezerva

- Minimální rezerva *(Minimum Slack first MS)*
  - $\max(d_j - p_j - t, 0)$
    - d_j termín dokončení
    - p_j doba provádění
    - t aktuální čas
  - funkci max používáme, abychom neměli záporné hodnoty pro úlohy, které už to určitě nestihnou
  - minimalizace kriterií svázaných s termínem dokončení,
    tj. maximální zpoždění



Statická vs. dynamická pravidla

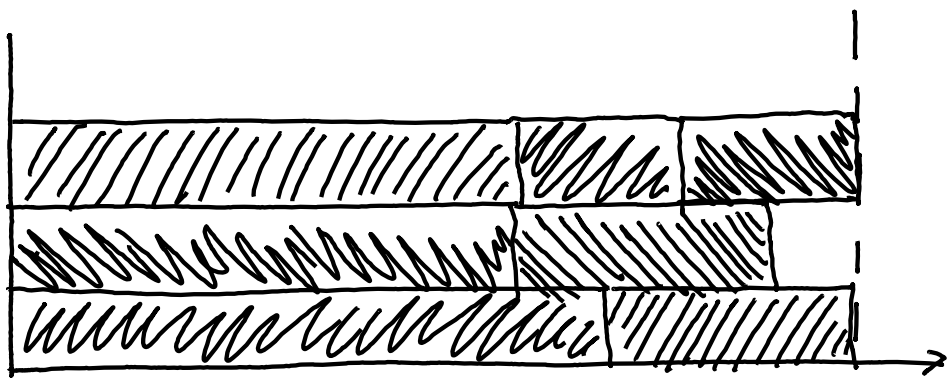
- Statická pravidla nejsou závislá na probíhajícím čase
  - pořadí se spočítá jako funkce závislá na úloze a/nebo stroji
  - pořadí nám definuje prioritní frontu úloh
  - př. nejdřívější termín dostupnosti
- Dynamická pravidla jsou závislá na čase
  - nutno zahrnout do výpočtu funkce i aktuální čas
  - uspořádání úloh závisí na čase $\Rightarrow$ v každém čase je nutné určit znovu úlohu s nejvyšší prioritou a tu zpracováváme
  - př. minimální rezerva

Pravidla s dobou trvání p_j

- Nejdelší doba trvání *(Longest Processing Time first LPT)*
  - směřuje k rovnoměrnému zatížení paralelních strojů,
    tj. k minimalizaci makespan

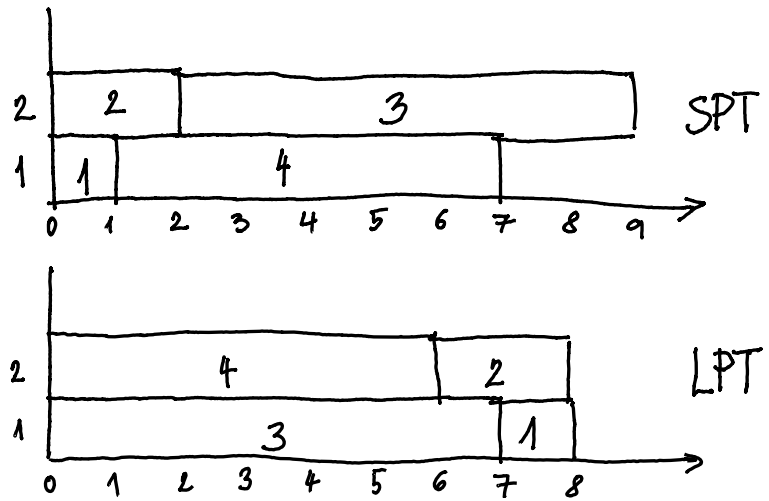
Pravidla s dobou trvání p_j

- Nejdelší doba trvání *(Longest Processing Time first LPT)*
  - směřuje k rovnoměrnému zatížení paralelních strojů,
    tj. k minimalizaci makespan
  - myšlenka: kratší úlohy lze později využít pro vyrovnání zátěže na konci; jakmile jsou úlohy přiřazeny na stroje, tak je lze přeuspořádat bez změny zatížení



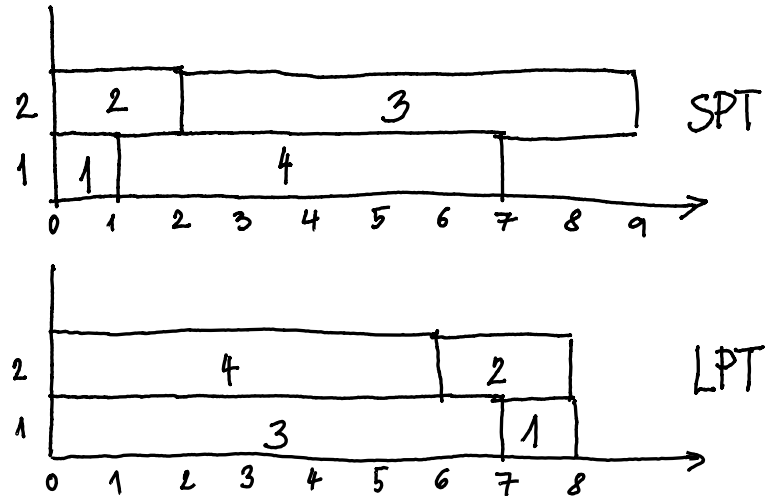
Pravidla s dobou trvání p_j

- Nejkratší doba trvání *(Shortest Processing Time first SPT)*
  - směřuje k minimalizaci součtu časů konců úloh, tj. WIP (Work In Process, cena za sklad při výrobě)



Pravidla s dobou trvání p_j

- Nejkratší doba trvání *(Shortest Processing Time first SPT)*
  - směřuje k minimalizaci součtu časů konců úloh, tj. WIP (Work In Process, cena za sklad při výrobě)



- Vážená nejkratší doba trvání *(Weighted Shortest Processing Time first WSPT)*
  - navíc w_j oproti SPT (řadím dle w_j/p_j)
  - minimalizace vážených součtu časů konců úloh, tj. WIP
  - optimální pro jeden stroj, kde jsou všechny úlohy dostupné na začátku ($r_j = 0$ pro každou úlohu j)

Další řídící pravidla

- Kritická cesta *(Critical Path CP)*
  - vhodné pro precedenční omezení
  - vybírá úlohu, která je první v nejdelším řetězci dob provádění v grafu úloh daném precedencemi
  - vede k minimalizaci makespan

Další řídící pravidla

- Kritická cesta *(Critical Path CP)*
  - vhodné pro precedenční omezení
  - vybírá úlohu, která je první v nejdelším řetězci dob provádění v grafu úloh daném precedencemi
  - vede k minimalizaci makespan
- Nejméně flexibilní úloha *(Least Flexible Job LFJ first)*
  - při zadání množiny vhodných strojů
  - vybírá se úloha, která může být prováděna na nejmenším počtu strojů (tj. nejméně alternativ)
  - vede k minimalizaci makespan

Další řídící pravidla

- Kritická cesta *(Critical Path CP)*
  - vhodné pro precedenční omezení
  - vybírá úlohu, která je první v nejdelším řetězci dob provádění v grafu úloh daném precedencemi
  - vede k minimalizaci makespan
- Nejméně flexibilní úloha *(Least Flexible Job LFJ first)*
  - při zadání množiny vhodných strojů
  - vybírá se úloha, která může být prováděna na nejmenším počtu strojů (tj. nejméně alternativ)
  - vede k minimalizaci makespan
- Náhodné pořadí *(Service in Random Order SIRO)*
  - náhodný výběr úloh

Řídící pravidla: diskuse

- Jednoduchá na implementaci
- Optimální ve speciálních případech
- Zaměřeny na jedno optimalizační kritérium
- Kombinování několika řídících pravidel: kompozitní řídící pravidla

Řídící pravidla: diskuse

- Jednoduchá na implementaci
- Optimální ve speciálních případech
- Zaměřeny na jedno optimalizační kritérium
- Kombinování několika řídících pravidel: kompozitní řídící pravidla

- Použití v praxi
  - pro řadu problémů příliš triviální
    - i tady lze např. použít jako generátor iniciálního řešení
    - nebo jako metodu pro řešení podproblémů
  - používá se pro složité problémy s vysokými nároky na propustnost
    - např. počet naplánovaných aktivit za vteřinu

    nebo pro problémy s vysokým stupněm dynamiky
    - např. plánování úloh na počítače
      (neznámá doba trvání, příchody nových úloh, výpadky strojů)

Lokální prohledávání

16. května 2022

Konstruktivní vs. lokální metody

Konstruktivní metody

- začneme s prázdným rozvrhem
- do rozvrhu přidáváme postupně jednotlivé úlohy tak, aby byl rozvrh stále konzistentní

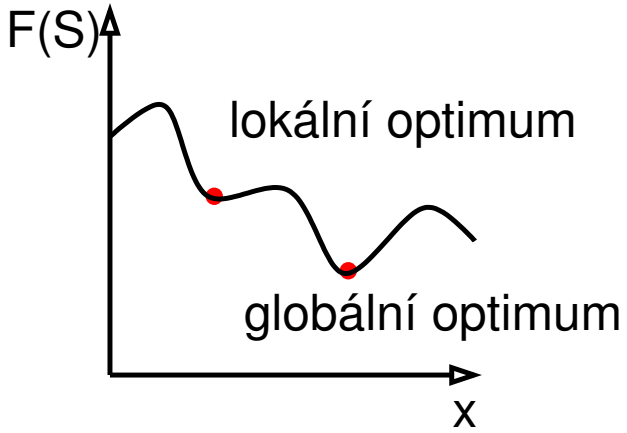
Lokální prohledávání

- začneme s úplným nekonzistentním rozvrhem
  - triviálně: s náhodně vygenerovaným
- snažíme se najít lepší „podobný" rozvrh lokálními změnami
- kvalitu rozvrhu posuzujeme optimalizačními kritérii
  - např. makespan
- optimalizační kritéria vyhodnocují také konzistenci rozvrhu
  - např. počet porušených precedenčních omezení

Hybridní přístupy

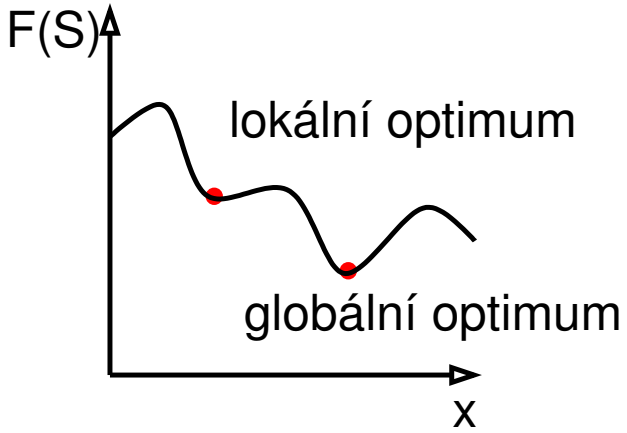
- kombinace obou metod

Algoritmus lokálního prohledávání



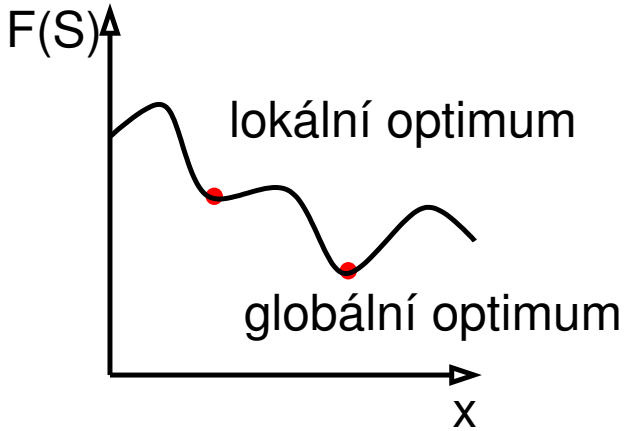
1. Inicializace
   - $k = 0$
   - výběr iniciálního rozvrhu S_0
   - zaznamenání dosud nejlepšího rozvrhu:
     $S_{best} = S_0$ a $best\_cost = F(S_0)$

Algoritmus lokálního prohledávání



1. Inicializace
   - $k = 0$
   - výběr iniciálního rozvrhu S_0
   - zaznamenání dosud nejlepšího rozvrhu:
     $S_{best} = S_0$ a $best\_cost = F(S_0)$
2. Výběr a aktualizace
   - výběr rozvrhu z okolí: $S_{k+1} \in N(S_k)$
   - pokud kriterium přijetí rozvrhu nesplňuje
     žádný prvek $N(S_k)$, pak algoritmus končí
   - jestliže $F(S_{k+1}) < best\_cost$ pak
     $S_{best} = S_{k+1}$ a $best\_cost = F(S_{k+1})$

Algoritmus lokálního prohledávání



1. Inicializace
   - $k = 0$
   - výběr iniciálního rozvrhu S_0
   - zaznamenání dosud nejlepšího rozvrhu:
     $S_{best} = S_0$ a $best\_cost = F(S_0)$
2. Výběr a aktualizace
   - výběr rozvrhu z okolí: $S_{k+1} \in N(S_k)$
   - pokud kriterium přijetí rozvrhu nesplňuje
     žádný prvek $N(S_k)$, pak algoritmus končí
   - jestliže $F(S_{k+1}) < best\_cost$ pak
     $S_{best} = S_{k+1}$ a $best\_cost = F(S_{k+1})$
3. Ukončení
   - jestliže platí podmínky ukončení pak algoritmus končí
   - jinak $k = k + 1$ a skok na krok 2.

Jeden stroj + nepreemptivní úlohy

Reprezentace rozvrhu

- permutace n úloh
- příklad se šesti úlohami: 1,4,2,6,3,5

Jeden stroj + nepreemptivní úlohy

Reprezentace rozvrhu

- permutace n úloh
- příklad se šesti úlohami: 1,4,2,6,3,5

Definice okolí

- párová výměna sousedních úloh
  - příklad: 1,4,2,6,3,5 se změní např. na 1,4,2,6,5,3
  - možných rozvrhů v okolí:

Jeden stroj + nepreemptivní úlohy

Reprezentace rozvrhu

- permutace n úloh
- příklad se šesti úlohami: 1,4,2,6,3,5

Definice okolí

- párová výměna sousedních úloh
  - příklad: 1,4,2,6,3,5 se změní např. na 1,4,2,6,5,3
  - možných rozvrhů v okolí: $n - 1$

Jeden stroj + nepreemptivní úlohy

Reprezentace rozvrhu

- permutace n úloh
- příklad se šesti úlohami: 1,4,2,6,3,5

Definice okolí

- párová výměna sousedních úloh
  - příklad: 1,4,2,6,3,5 se změní např. na 1,4,2,6,5,3
  - možných rozvrhů v okolí: $n - 1$
- nebo výběr libovolné úlohy v rozvrhu a umístění na libovolnou pozici
  - příklad: z 1,4,2,6,3,5 náhodně vybereme 4 a dáme ji jinam: 1,2,6,3,4,5
  - možných rozvrhů v okolí:

Jeden stroj + nepreemptivní úlohy

Reprezentace rozvrhu

- permutace n úloh
- příklad se šesti úlohami: 1,4,2,6,3,5

Definice okolí

- párová výměna sousedních úloh
  - příklad: 1,4,2,6,3,5 se změní např. na 1,4,2,6,5,3
  - možných rozvrhů v okolí: $n - 1$
- nebo výběr libovolné úlohy v rozvrhu a umístění na libovolnou pozici
  - příklad: z 1,4,2,6,3,5 náhodně vybereme 4 a dáme ji jinam: 1,2,6,3,4,5
  - možných rozvrhů v okolí: $\leq n(n-1)$

Iniciální řešení & podmínky ukončení

Iniciální řešení generujeme

- náhodně
- heuristicky např. konstruktivním algoritmem jako jsou řídící pravidla

Podmínka ukončení typicky

- zadaný počet iterací
- omezená doba běhu
- počet porovnání účelové funkce
- žádné zlepšení nezískáno po daném počtu iterací

Výběr rozvrhu z okolí

- Lokální změna
  - úprava rozvrhu výběrem rozvrhu z okolí
- Výběr rozvrhu z okolí, tj. prohledání okolí a výběr vhodného kandidáta
  - náhodný výběr
  - výběr nejslibnějšího kandidáta
    - vybíráme rozvrh s nejlepší hodnotou účelové funkce v okolí
    - snaha o zlepšení hodnoty účelové funkce

Kritérium výběru rozvrhu

Kritérium výběru rozvrhu =
kriterium přijetí/odmítnutí rozvrhu

- akceptovat vždy lepší rozvrh?
- někdy akceptovat i horší rozvrh?

Metody akceptování horšího rozvrhu

- pravděpodobnostní
  - náhodná procházka: s malou pravděpodobností (např. 0.01) akceptujeme i horší rozvrh
  - simulované žíhání
- deterministická
  - tabu prohledávání: udržujeme tabu seznam několika posledních stavů/změn, které jsou pro další výběr nepřípustné

Tabu prohledávání

Deterministické kritérium přijetí/odmítnutí rozvrhu

Udržován tabu seznam několika posledních změn v rozvrhu

- tabu seznam = seznam zakázaných změn
- každá nová změna je umístěna na vrchol tabu seznamu
  - př. uchovávané změny: výměna úloh j a k
- okolí omezeno na rozvrhy, které nepožadují změnu z tabu seznamu
  - zabraňuje cyklení
  - příklad triviálního cyklení:
    první krok: prohození úloh 3 a 4, druhý krok: prohození úloh 4 a 3
- pevná délka seznamu (typicky: 5-9)
  - nejstarší změny z tabu seznamu odstraněny
  - příliš malá délka:
  - příliš velká délka:

Aspirační kritérium

- určuje, kdy je možné akceptovat i změny v tabu seznamu
- př. změna z tabu seznamu povolena, pokud zlepšeno $F(S_{best})$

Tabu prohledávání

Deterministické kritérium přijetí/odmítnutí rozvrhu

Udržován tabu seznam několika posledních změn v rozvrhu

- tabu seznam = seznam zakázaných změn
- každá nová změna je umístěna na vrchol tabu seznamu
  - př. uchovávané změny: výměna úloh j a k
- okolí omezeno na rozvrhy, které nepožadují změnu z tabu seznamu
  - zabraňuje cyklení
  - příklad triviálního cyklení:
    první krok: prohození úloh 3 a 4, druhý krok: prohození úloh 4 a 3
- pevná délka seznamu (typicky: 5-9)
  - nejstarší změny z tabu seznamu odstraněny
  - příliš malá délka: nebezpečí cyklení
  - příliš velká délka: může omezit prohledávání příliš

Aspirační kritérium

- určuje, kdy je možné akceptovat i změny v tabu seznamu
- př. změna z tabu seznamu povolena, pokud zlepšeno $F(S_{best})$

Algoritmus tabu prohledávání

1.
   - $k = 1$
   - výběr iniciálního rozvrhu S_1 použitím heuristiky, $S_{best} = S_1$

Algoritmus tabu prohledávání

1.
   - $k = 1$
   - výběr iniciálního rozvrhu S_1 použitím heuristiky, $S_{best} = S_1$
2.
   - výběr $S_c \in N(S_k)$
   - jestliže je změna $S_k \to S_c$ zakázána
     protože je v tabu seznamu
     a není splněno aspirační kritérium
     pak běž na krok 2

Algoritmus tabu prohledávání

1.
   - $k = 1$
   - výběr iniciálního rozvrhu S_1 použitím heuristiky, $S_{best} = S_1$
2.
   - výběr $S_c \in N(S_k)$
   - jestliže je změna $S_k \to S_c$ zakázána
     protože je v tabu seznamu
     a není splněno aspirační kritérium
     pak běž na krok 2
3.
   - jestliže změna $S_k \to S_c$ není zakázána tabu seznamem
     nebo je splněno aspirační kritérium
     pak $S_{k+1} = S_c$
     ulož reversní změnu na vrchol tabu seznamu
     posuň další pozice v tabu seznamu o pozici níže
     smaž poslední položku z tabu seznamu
   - jestliže $F(S_c) < F(S_{best})$ pak $S_{best} = S_c$

Algoritmus tabu prohledávání

1.
   - $k = 1$
   - výběr iniciálního rozvrhu S_1 použitím heuristiky, $S_{best} = S_1$
2.
   - výběr $S_c \in N(S_k)$
   - jestliže je změna $S_k \to S_c$ zakázána
     protože je v tabu seznamu
     a není splněno aspirační kritérium
     pak běž na krok 2
3.
   - jestliže změna $S_k \to S_c$ není zakázána tabu seznamem
     nebo je splněno aspirační kritérium
     pak $S_{k+1} = S_c$
     ulož reversní změnu na vrchol tabu seznamu
     posuň další pozice v tabu seznamu o pozici níže
     smaž poslední položku z tabu seznamu
   - jestliže $F(S_c) < F(S_{best})$ pak $S_{best} = S_c$
4.
   - $k = k + 1$
   - jestliže platí podmínka ukončení pak konec
     jinak běž na krok 2.

Příklad: tabu seznam

- Uvažujte rozvrhovací problém s $1||\sum w_j T_j$
  - opakování: $T_j = \max(C_j - d_j, 0)$

úlohy	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

- Okolí: všechny rozvrhy získané párovou výměnou sousedních úloh
- Výběr rozvrhu z okolí: vybereme nejlepší rozvrh
- Tabu seznam: páry úloh (j, k), které byly přehozeny při posledních dvou změnách

Příklad: tabu seznam

- Uvažujte rozvrhovací problém s $1||\sum w_j T_j$
  - opakování: $T_j = \max(C_j - d_j, 0)$

úlohy	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

- Okolí: všechny rozvrhy získané párovou výměnou sousedních úloh
- Výběr rozvrhu z okolí: vybereme nejlepší rozvrh
- Tabu seznam: páry úloh (j, k), které byly přehozeny při posledních dvou změnách

$S_1 = (2, 1, 4, 3)$
$F(S_1) = \sum w_j T_j =$

Příklad: tabu seznam

- Uvažujte rozvrhovací problém s $1||\sum w_j T_j$
  - opakování: $T_j = \max(C_j - d_j, 0)$

úlohy	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

- Okolí: všechny rozvrhy získané párovou výměnou sousedních úloh
- Výběr rozvrhu z okolí: vybereme nejlepší rozvrh
- Tabu seznam: páry úloh (j, k), které byly přehozeny při posledních dvou změnách

$S_1 = (2, 1, 4, 3)$
$F(S_1) = \sum w_j T_j = 12 \cdot 8 + 14 \cdot 16 + 12 \cdot 12 + 1 \cdot 36 = 500 = F(S_{best})$
$F(1, 2, 4, 3) = 480$
$F(2, \underline{4}, \underline{1}, 3) = 436 = F(S_{best})$
$F(2, 1, 3, 4) = 652$
Tabu seznam: $\langle(1, 4)\rangle$

Příklad: tabu seznam (pokračování)

$S_2 = (2, 4, 1, 3), F(S_2) = 436$
$F(\underline{4}, \underline{2}, 1, 3) = 460$
$F(2, 1, 4, 3)(= 500)$ tabu!
$F(2, 4, 3, 1) = 608$
Tabu seznam: $\langle (2, 4), (1, 4) \rangle$

Příklad: tabu seznam (pokračování)

$S_2 = (2, 4, 1, 3), F(S_2) = 436$
$F(\underline{4}, \underline{2}, 1, 3) = 460$
$F(2, 1, 4, 3)(= 500)$ tabu!
$F(2, 4, 3, 1) = 608$
Tabu seznam: $\langle (2, 4), (1, 4) \rangle$

$S_3 = (4, 2, 1, 3), F(S_3) = 460$
$F(2, 4, 1, 3)(= 436)$ tabu!
$F(4, \underline{1}, \underline{2}, 3) = 440$
$F(4, 2, 3, 1) = 632$
Tabu seznam: $\langle (2, 1), (2, 4) \rangle$

Příklad: tabu seznam (pokračování)

$S_2 = (2, 4, 1, 3), F(S_2) = 436$
$F(\underline{4}, \underline{2}, 1, 3) = 460$
$F(2, 1, 4, 3)(= 500)$ tabu!
$F(2, 4, 3, 1) = 608$
Tabu seznam: $\langle (2, 4), (1, 4) \rangle$

$S_3 = (4, 2, 1, 3), F(S_3) = 460$
$F(2, 4, 1, 3)(= 436)$ tabu!
$F(4, \underline{1}, \underline{2}, 3) = 440$
$F(4, 2, 3, 1) = 632$
Tabu seznam: $\langle (2, 1), (2, 4) \rangle$

$F_4 = (4, 1, 2, 3), F(S_4) = 440$
$F(\underline{1}, \underline{4}, 2, 3) = 408 = F(S_{best})$
$F(4, 2, 1, 3)(= 460)$ tabu!
$F(4, 1, 3, 2) = 586$
Tabu seznam: $\langle (4, 1), (2, 1) \rangle$
$F(S_{best}) = 408$

Cvičení: tabu seznam

- Uvažujte rozvrhovací problém s $1||\sum w_j T_j$

úlohy	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

- Aplikujte tabu prohledávání pro iniciální řešení $(2, 1, 4, 3)$
- Okolí: všechny rozvrhy získané párovou výměnou sousedních úloh
- Výběr rozvrhu z okolí: vybereme nejlepší rozvrh
- Proveďte čtyři iterace
- Tabu seznam: páry úloh (j, k), které byly přehozeny při
  1. jedné poslední změně — výsledek: $F(S_{best}) = 408$
  2. třech posledních změnách — výsledek: $F(S_{best}) = 436$

Simulované žíhání: principy

- Myšlenka: simulace procesu ochlazování kovů
  - na začátku při vyšší teplotě atomy více kmitají a pravděpodobnost změny krystalické mřížky je vyšší
  - postupným ochlazováním se atomy usazují do „nejlepší polohy" s nejmenší energií a pravděpodobnost změny je menší

  $\Rightarrow$ na začátku je tedy pravděpodobnost toho, že akceptujeme zhoršování řešení, vyšší
- Aplikace u lokálního prohledávání
  - lepší nebo stejné řešení akceptováno
  - algoritmus umožňuje akceptovat horší řešení, abychom unikli z lokálního minima
  - pravděpodobnost přijetí horšího řešení se postupně snižuje

Simulované žíhání: realizace

- Parametr chlazení t
  - t je iniciálně vysoké: hodně změn je akceptováno
  - t postupně klesá: horší změny jsou skoro vždy odmítnuty
- Rozdíl mezi kvalitou nového a existujícího řešení
  - $\Delta c = F(S_{new}) - F(S_{old})$
- Metropolisovo kritérium
  - předpoklad: minimalizace F
  - lepší nebo stejné řešení akceptováno: $\Delta c \leq 0$, tj. $F(S_{new}) \leq F(S_{old})$
  - pravděpodobnostní přijetí/odmítnutí rozvrhu:
    horší řešení ($\Delta c > 0$) akceptováno pokud

$$U < e^{-\Delta c/t}$$

  - U náhodné číslo z intervalu $(0, 1)$

Metropolisovo kritérium

Horší řešení ($F(S_{new}) - F(S_{old}) = \Delta c > 0$) akceptováno pokud

$$U < e^{-\Delta c/t}$$

- U náhodné číslo z intervalu $(0, 1)$
- $\Delta c > 0$, tj. $-\Delta c < 0$
- pokud klesá t nebo klesá $-\Delta c$, tak klesá i $e^{-\Delta c/t}$
- pomůcka: porovnej $e^{-10/100}$ vs. $e^{-100/100}$

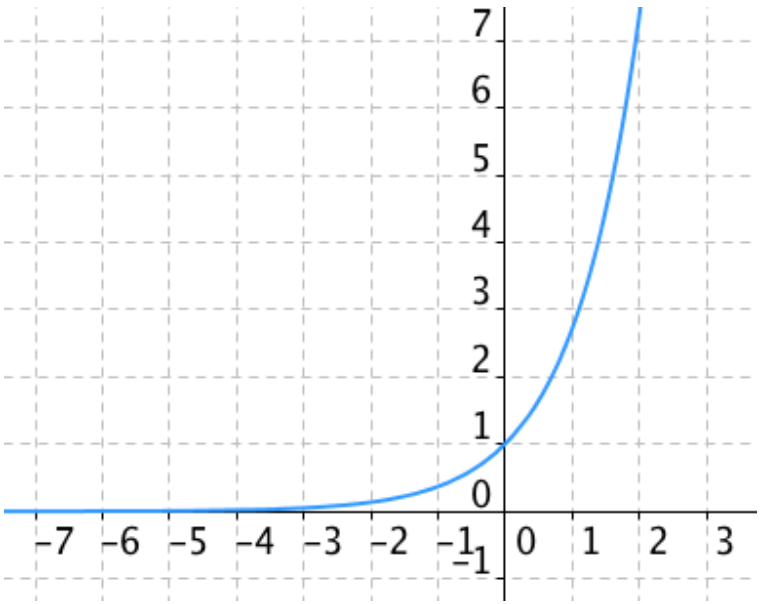
Metropolisovo kritérium

Horší řešení ($F(S_{new}) - F(S_{old}) = \Delta c > 0$) akceptováno pokud

$$U < e^{-\Delta c/t}$$

- U náhodné číslo z intervalu $(0, 1)$
- $\Delta c > 0$, tj. $-\Delta c < 0$
- pokud klesá t nebo klesá $-\Delta c$, tak klesá i $e^{-\Delta c/t}$
- pomůcka: porovnej $e^{-10/100}$ vs. $e^{-100/100}$

Graf e^x



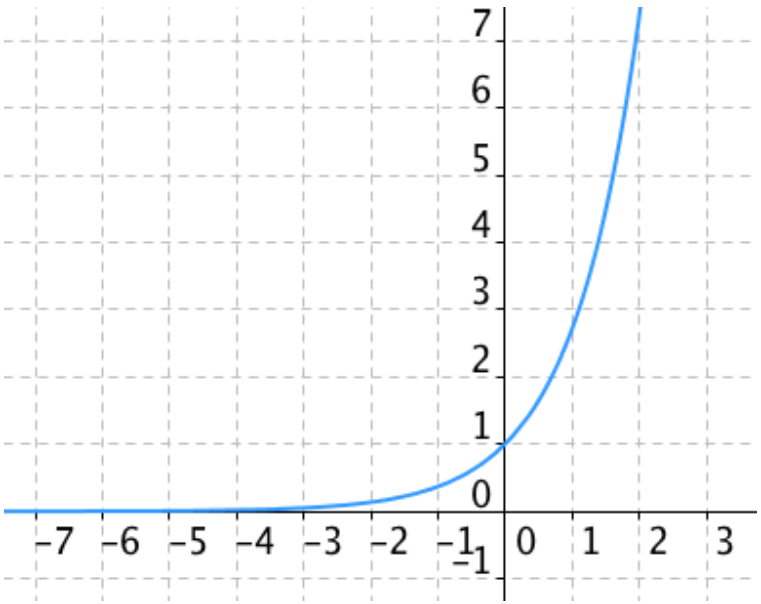
Metropolisovo kritérium

Horší řešení ($F(S_{new}) - F(S_{old}) = \Delta c > 0$) akceptováno pokud

$$U < e^{-\Delta c/t}$$

- U náhodné číslo z intervalu $(0, 1)$
- $\Delta c > 0$, tj. $-\Delta c < 0$
- pokud klesá t nebo klesá $-\Delta c$, tak klesá i $e^{-\Delta c/t}$
- pomůcka: porovnej $e^{-10/100}$ vs. $e^{-100/100}$ a $e^{-10/100}$ vs. $e^{-10/1}$

Graf e^x



Algoritmus simulovaného žíhání

1.
   - $k = 1$
   - výběr iniciálního rozvrhu S_1 použitím heuristiky, $S_{best} = S_1$
   - nastavení iniciální teploty $t_0 > 0$
   - výběr funkce redukce teploty $\alpha(t)$

Algoritmus simulovaného žíhání

1.
   - $k = 1$
   - výběr iniciálního rozvrhu S_1 použitím heuristiky, $S_{best} = S_1$
   - nastavení iniciální teploty $t_0 > 0$
   - výběr funkce redukce teploty $\alpha(t)$
2.
   - výběr $S_{new} \in N(S_k)$
   - jestliže $F(S_{new}) < F(S_{best})$ pak $S_{best} = S_{new}$, $S_{k+1} = S_{new}$
     jinak
     jestliže $F(S_{new}) \leq F(S_k)$ pak $S_{k+1} = S_{new}$
     jinak
     generuj náhodné číslo U_k
     jestliže $U_k < e^{\frac{F(S_k) - F(S_{new})}{t}}$ pak $S_{k+1} = S_{new}$
     jinak $S_{k+1} = S_k$

Algoritmus simulovaného žíhání

1.
   - $k = 1$
   - výběr iniciálního rozvrhu S_1 použitím heuristiky, $S_{best} = S_1$
   - nastavení iniciální teploty $t_0 > 0$
   - výběr funkce redukce teploty $\alpha(t)$
2.
   - výběr $S_{new} \in N(S_k)$
   - jestliže $F(S_{new}) < F(S_{best})$ pak $S_{best} = S_{new}$, $S_{k+1} = S_{new}$
     jinak
     jestliže $F(S_{new}) \leq F(S_k)$ pak $S_{k+1} = S_{new}$
     jinak
     generuj náhodné číslo U_k
     jestliže $U_k < e^{\frac{F(S_k) - F(S_{new})}{t}}$ pak $S_{k+1} = S_{new}$
     jinak $S_{k+1} = S_k$
3.
   - $t_k = \alpha(t_k)$
   - $k = k + 1$
   - jestliže platí podmínka ukončení pak konec
     jinak běž na krok 2.

Příklad: simulované žíhání

- Opakování: $T_j = \max(C_j - d_j, 0)$
- Uvažujte rozvrhovací problém s $1||\sum w_j T_j$

úlohy	1	2	3	4
p_j	9	9	12	3
d_j	10	8	5	28
w_j	14	12	1	12

- Použijte simulované žíhání s iniciálním rozvrhem $(3, 1, 4, 2)$
- Okolí: všechny rozvrhy, které dostaneme sousedními párovými výměnami
- Výběr rozvrhu z okolí náhodně
- Zvolte $\alpha(t) = 0.9 \times t$
- $t_0 = 0.9$
- Použijte následující náhodná čísla: $0.17, 0.91, \ldots$

Příklad: simulované žíhání (řešení)

$S_{best} = S_1 = (3, 1, 4, 2)$
$F(S_1) = \sum w_j T_j =$

Příklad: simulované žíhání (řešení)

$S_{best} = S_1 = (3, 1, 4, 2)$
$F(S_1) = \sum w_j T_j = 1 \cdot 7 + 14 \cdot 11 + 12 \cdot 0 + 12 \cdot 25 = 461 = F(S_{best})$
$t_0 = 0.9$

Příklad: simulované žíhání (řešení)

$S_{best} = S_1 = (3, 1, 4, 2)$
$F(S_1) = \sum w_j T_j = 1 \cdot 7 + 14 \cdot 11 + 12 \cdot 0 + 12 \cdot 25 = 461 = F(S_{best})$
$t_0 = 0.9$

$S_{new} = (1, 3, 4, 2)$
$F(S_{new}) = 316 < F(S_{best})$
$S_{best} = (1, 3, 4, 2)$
$F(S_{best}) = 316$
$S_2 = (1, 3, 4, 2)$
$t = 0.9 \times 0.9 = 0.81$

Příklad: simulované žíhání (řešení)

$S_{best} = S_1 = (3, 1, 4, 2)$
$F(S_1) = \sum w_j T_j = 1 \cdot 7 + 14 \cdot 11 + 12 \cdot 0 + 12 \cdot 25 = 461 = F(S_{best})$
$t_0 = 0.9$

$S_{new} = (1, 3, 4, 2)$
$F(S_{new}) = 316 < F(S_{best})$
$S_{best} = (1, 3, 4, 2)$
$F(S_{best}) = 316$
$S_2 = (1, 3, 4, 2)$
$t = 0.9 \times 0.9 = 0.81$

$S_{new} = (1, 3, 2, 4)$
$F(S_{new}) = 340 > F(S_{best}) = 316$
$F(S_{new}) > F(S_2) = 316$
$U_1 = 0.17 > e^{-(340-316)/0.81} = 1.35 \times 10^{-13}$
$S_3 = S_2 = (1, 3, 4, 2)$
$t = 0.729$

Příklad: simulované žíhání (řešení)

$S_{best} = S_1 = (3, 1, 4, 2)$
$F(S_1) = \sum w_j T_j = 1 \cdot 7 + 14 \cdot 11 + 12 \cdot 0 + 12 \cdot 25 = 461 = F(S_{best})$
$t_0 = 0.9$

$S_{new} = (1, 3, 4, 2)$
$F(S_{new}) = 316 < F(S_{best})$
$S_{best} = (1, 3, 4, 2)$
$F(S_{best}) = 316$
$S_2 = (1, 3, 4, 2)$
$t = 0.9 \times 0.9 = 0.81$

$S_{new} = (1, 3, 2, 4)$
$F(S_{new}) = 340 > F(S_{best}) = 316$
$F(S_{new}) > F(S_2) = 316$
$U_1 = 0.17 > e^{-(340-316)/0.81} = 1.35 \times 10^{-13}$
$S_3 = S_2 = (1, 3, 4, 2)$
$t = 0.729$

$S_{new} = (1, 4, 3, 2)$
$F(S_{new}) = 319 > F(S_{best}) = 316$
$F(S_{new}) > F(S_3) = 316$
$U_3 = 0.91 > e^{-(319-316)/0.729} = 0.016$
$S_4 = S_3 = (1, 3, 4, 2)$
$t = 0.6561$
...

Praktické úvahy

- Iniciální teplota
  - musí být „vysoká”
  - kritérium přijetí rozvrhu: 40%–60% vykazuje dobré výsledky v mnoha situacích

Praktické úvahy

- Iniciální teplota
  - musí být „vysoká"
  - kritérium přijetí rozvrhu: 40%–60% vykazuje dobré výsledky v mnoha situacích
- Parametr chlazení
  - několik změn při dané teplotě
  - jedna změna při každé teplotě

  $t = \alpha \times t$

  $t = \frac{t}{1+\beta t}$

  α typicky v intervalu [0.9,0.99]

  β typicky blízké k 0

Praktické úvahy

- Iniciální teplota
  - musí být „vysoká"
  - kritérium přijetí rozvrhu: 40%–60% vykazuje dobré výsledky v mnoha situacích
- Parametr chlazení
  - několik změn při dané teplotě
  - jedna změna při každé teplotě

  $t = \alpha \times t$

  $t = \frac{t}{1+\beta t}$

  α typicky v intervalu [0.9,0.99]

  β typicky blízké k 0

Genetické algoritmy

Srovnání

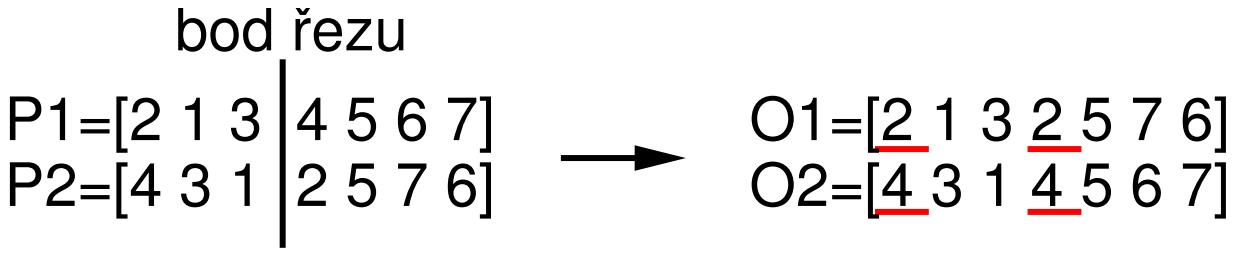
- simulované žíhání, tabu prohledávání
  - jedno řešení je přenášeno z jedné iterace do druhé
- genetické algoritmy
  - udržována populace (několika přenášených) řešení

Genetické algoritmy

- rozvrhy jsou jednotlivci (chromozomy), kteří tvoří populaci
- rozhodovací proměnná (např. čas jedné úlohy) je gen
- hodnota rozhodovací proměnné (např. konkrétní čas) je alela
- každý jednotlivec je vyhodnocen kritériem vhodnosti *(fitness)*
- někteří jednotlivci mutují
- jednotlivci jsou vybráni k reprodukci křížením a mají děti tvořící následující populaci
- nejvhodnější jedinci přežijí do další populace

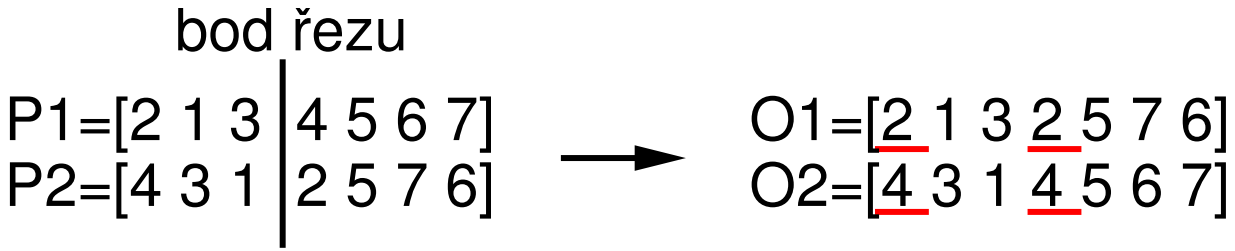
Reprodukce

- Křížení *(crossover)*: kombinování posloupnosti operací na jednom stroji v jednom rodičovském rozvrhu s posloupností operací na jiném stroji u jiného rodiče
- Operátor jednoduchého křížení není užitečný!



Reprodukce

- Křížení *(crossover)*: kombinování posloupnosti operací na jednom stroji v jednom rodičovském rozvrhu s posloupností operací na jiném stroji u jiného rodiče
- Operátor jednoduchého křížení není užitečný!

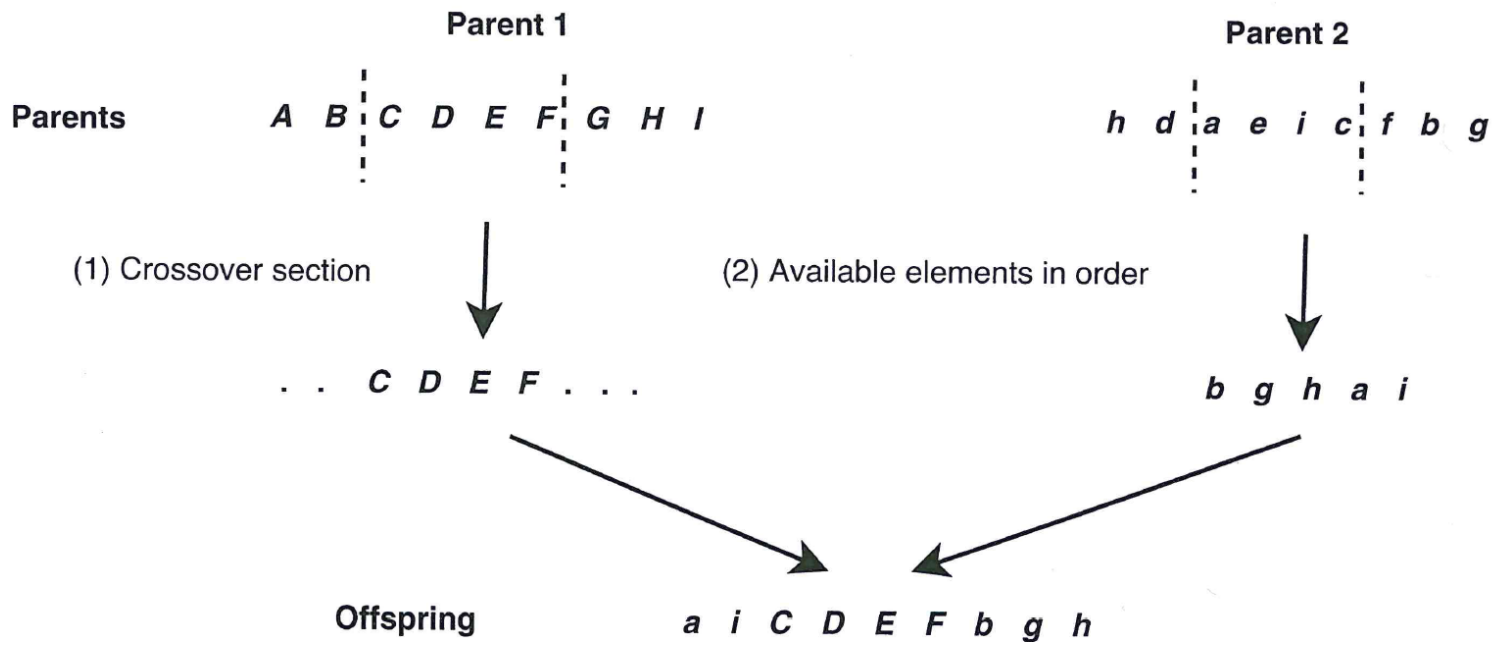


- Každá alela (hodnota) se musí výskytnout v jedinci právě jednou
  - používány různé formy mapování

Křížení dané pořadím

Křížení dané pořadím (Order crossover, OX)

1. Vybrány náhodně dva body křížení
2. Z rodiče 1 hodnoty mezi nimi zkopírovány na stejné pozice v potomkovi
3. Z rodiče 2 začneme od druhého bodu křížení vybírat prvky, které již nebyly vybrány z rodiče 1, a dávame je do potomka od 2. bodu křížení



Mutace

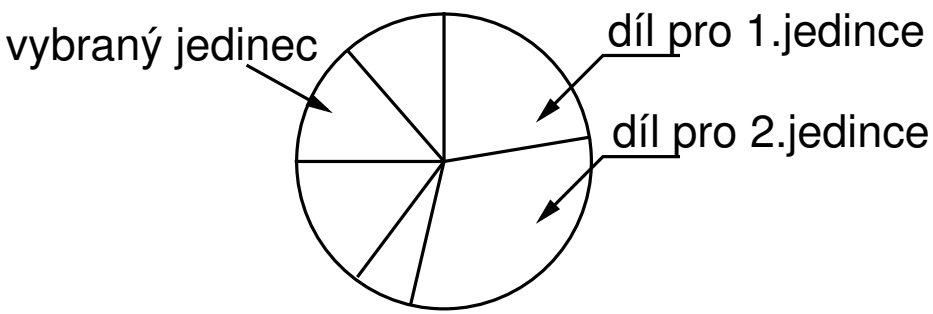
- Mutace umožňuje genetickému algoritmu prohledávat prostor nedosažitelný operátorem křížení
- Párová výměna sousedů v posloupnosti

$$[1, 2, \ldots, n] \rightarrow [2, 1, \ldots, n]$$

- Mutace výměnou: změna dvou náhodně vybraných prvků permutace
- Mutace posunem: přesun náhodně vybraného prvku o náhodný počet míst doleva nebo doprava
- Mutace promícháním podseznamu: výběr dvou bodů v řetězci náhodně a náhodná permutace prvků mezi těmito dvěma pozicemi

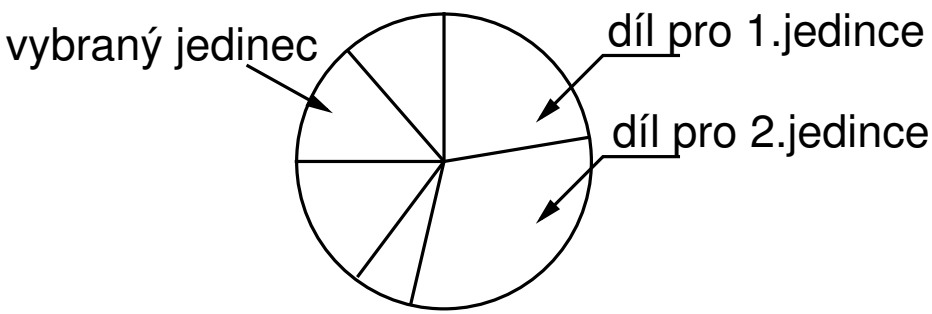
Výběr

- Ruletové kolo
  - šance na reprodukci jsou proporcionálně závislé na vhodnosti jedince
  - velikost každého dílu ruletového kola záleží na vhodnosti konkrétního jedince
  - př. kritérium vhodnosti pro 6 jedinců: 5,7,1,3,3,3
    první díl má velikost 5, druhý 7, třetí 1, čtvrtý 3, páty 3, šestý 3



Výběr

- Ruletové kolo
  - šance na reprodukci jsou proporcionálně závislé na vhodnosti jedince
  - velikost každého dílu ruletového kola záleží na vhodnosti konkrétního jedince
  - př. kritérium vhodnosti pro 6 jedinců: 5,7,1,3,3,3
    první díl má velikost 5, druhý 7, třetí 1, čtvrtý 3, páty 3, šestý 3



- Kroky pro ruletové kolo
  1. sečti vhodnost všech jednotlivců populace: TF
     - př. $TF = 5 + 7 + 1 + 3 + 3 + 3 = 22$
  2. generuj náhodné číslo m mezi 0 a 22
  3. vrať prvního jednotlivce populace do jehož dílu spadá m
     - čím větší vhodnost jedince, tím větší pravděpodobnost, že se na něj strefíme

Výběr (pokračování)

Turnajový výběr

1. výběr jednoho jedince
   - náhodně vyber skupinu t jedinců z populace
   - vyber nejlepšího jedince
2. pokud potřebujeme vybrat k jedinců, pak postup opakujeme k-krát

Jak garantovat, že nejlepší člen/členové populace přežijí?

Elitářský model:

- nejlepší člen populace je vybrán jako člen následující populace

  nebo
- nejlepší potomci a rodiče jsou vybíráni do následující populace

Základní implementace genetického algoritmu

1.
   - $k = 1$
   - vyber N iniciálních rozvrhů $S_{1,1}, \ldots, S_{1,N}$ použitím heuristiky
   - vyhodnoť jednotlivce populace

Základní implementace genetického algoritmu

1.
   - $k = 1$
   - vyber N iniciálních rozvrhů $S_{1,1}, \ldots, S_{1,N}$ použitím heuristiky
   - vyhodnoť jednotlivce populace
2.
   - vytvoř nové jednotlivce spojením jednotlivců současné populace pomocí křížení a mutace
   - smaž členy existující populace, aby udělali místo novým jednotlivcům
   - vyhodnoť nové jednotlivce a přidej je do populace $S_{k+1,1}, \ldots, S_{k+1,N}$

Základní implementace genetického algoritmu

1.
   - $k = 1$
   - vyber N iniciálních rozvrhů $S_{1,1}, \ldots, S_{1,N}$ použitím heuristiky
   - vyhodnoť jednotlivce populace
2.
   - vytvoř nové jednotlivce spojením jednotlivců současné populace pomocí křížení a mutace
   - smaž členy existující populace, aby udělali místo novým jednotlivcům
   - vyhodnoť nové jednotlivce a přidej je do populace $S_{k+1,1}, \ldots, S_{k+1,N}$
3.
   - $k = k + 1$
   - jestliže platí podmínka ukončení pak vrať nejlepšího jedince jako řešení a skonči jinak běž na krok 2.

Příklad: genetické algoritmy

- Uvažujte rozvrhovací problém s $1||\sum T_j$

úlohy	1	2	3	4	5
p_j	4	3	7	2	2
d_j	5	6	8	8	17

- Velikost populace: 3
- Výběr
  - v každé populaci se reprodukuje nejvhodnější jedinec
    - triviální verze pro snadný výpočet!
  - pro reprodukci je použita párová výměna sousedů
  - počet možných potomků:

Příklad: genetické algoritmy

- Uvažujte rozvrhovací problém s $1||\sum T_j$

úlohy	1	2	3	4	5
p_j	4	3	7	2	2
d_j	5	6	8	8	17

- Velikost populace: 3
- Výběr
  - v každé populaci se reprodukuje nejvhodnější jedinec
    - triviální verze pro snadný výpočet!
  - pro reprodukci je použita párová výměna sousedů
  - počet možných potomků: 4
    pro reprodukci náhodně vybrán jeden z nich
  - potomek nahradí nejhoršího rodiče
- Iniciální populace: náhodné posloupnosti permutací

Příklad: genetické algoritmy (řešení)

Populace 1	Jedinec	25314	14352	12345
	Cena	25	17	16

Vybraný jedinec:

Příklad: genetické algoritmy (řešení)

Populace 1	Jedinec	25314	14352	12345
	Cena	25	17	16

Vybraný jedinec: 12345 s potomkem

Příklad: genetické algoritmy (řešení)

Populace 1	Jedinec	25314	14352	12345
	Cena	25	17	16

Vybraný jedinec: 12345 s potomkem 13245, cena 20

Příklad: genetické algoritmy (řešení)

Populace 1	Jedinec	25314	14352	12345
	Cena	25	17	16

Vybraný jedinec: 12345 s potomkem 13245, cena 20

Populace 2	Jedinec	13245	14352	12345
	Cena	20	17	16

Vybraný jedinec:

Příklad: genetické algoritmy (řešení)

Populace 1	Jedinec	25314	14352	12345
	Cena	25	17	16

Vybraný jedinec: 12345 s potomkem 13245, cena 20

Populace 2	Jedinec	13245	14352	12345
	Cena	20	17	16

Vybraný jedinec: 12345 s potomkem 12354, cena 17

Příklad: genetické algoritmy (řešení)

Populace 1	Jedinec	25314	14352	12345
	Cena	25	17	16

Vybraný jedinec: 12345 s potomkem 13245, cena 20

Populace 2	Jedinec	13245	14352	12345
	Cena	20	17	16

Vybraný jedinec: 12345 s potomkem 12354, cena 17

Populace 3	Jedinec	12354	14352	12345
	Cena	17	17	16

Vybraný jedinec:

Příklad: genetické algoritmy (řešení)

Populace 1	Jedinec	25314	14352	12345
	Cena	25	17	16

Vybraný jedinec: 12345 s potomkem 13245, cena 20

Populace 2	Jedinec	13245	14352	12345
	Cena	20	17	16

Vybraný jedinec: 12345 s potomkem 12354, cena 17

Populace 3	Jedinec	12354	14352	12345
	Cena	17	17	16

Vybraný jedinec: 12345 s potomkem 12435, cena 11

Příklad: genetické algoritmy (řešení)

Populace 1	Jedinec	25314	14352	12345
	Cena	25	17	16

Vybraný jedinec: 12345 s potomkem 13245, cena 20

Populace 2	Jedinec	13245	14352	12345
	Cena	20	17	16

Vybraný jedinec: 12345 s potomkem 12354, cena 17

Populace 3	Jedinec	12354	14352	12345
	Cena	17	17	16

Vybraný jedinec: 12345 s potomkem 12435, cena 11

Populace 4	Jedinec	14352	12345	12435
	Cena	17	16	11

Vybraný jedinec:

Příklad: genetické algoritmy (řešení)

Populace 1	Jedinec	25314	14352	12345
	Cena	25	17	16

Vybraný jedinec: 12345 s potomkem 13245, cena 20

Populace 2	Jedinec	13245	14352	12345
	Cena	20	17	16

Vybraný jedinec: 12345 s potomkem 12354, cena 17

Populace 3	Jedinec	12354	14352	12345
	Cena	17	17	16

Vybraný jedinec: 12345 s potomkem 12435, cena 11

Populace 4	Jedinec	14352	12345	12435
	Cena	17	16	11

Vybraný jedinec: 12435 – optimální řešení

Příklad: genetické algoritmy (řešení)

Populace 1	Jedinec	25314	14352	12345
	Cena	25	17	16

Vybraný jedinec: 12345 s potomkem 13245, cena 20

Populace 2	Jedinec	13245	14352	12345
	Cena	20	17	16

Vybraný jedinec: 12345 s potomkem 12354, cena 17

Populace 3	Jedinec	12354	14352	12345
	Cena	17	17	16

Vybraný jedinec: 12345 s potomkem 12435, cena 11

Populace 4	Jedinec	14352	12345	12435
	Cena	17	16	11

Vybraný jedinec: 12435 – optimální řešení

Nevýhoda takto jednoduchého nastavení algoritmu:
výběr nejlepšího jedince způsobuje opakovanou reprodukci nejlepšího jedince, dokud není nahrazen lepším potomkem

Praktické úvahy

- Velikost populace
  - malá populace
  - velká populace

Praktické úvahy

- Velikost populace
  - malá populace riskuje příliš malé pokrytí prohledávacího prostoru
  - velká populace

Praktické úvahy

- Velikost populace
  - malá populace riskuje příliš malé pokrytí prohledávacího prostoru
  - velká populace má velké výpočetní nároky

Praktické úvahy

- Velikost populace
  - malá populace riskuje příliš malé pokrytí prohledávacího prostoru
  - velká populace má velké výpočetní nároky
  - dle empirických výsledků
    - velikost populace kolem 30 často adekvátní
    - velikost populace 20-100 je běžná
- Mutace: obvykle použita s pravděpodobností
  - např. mutace jednoho genu jedince

Praktické úvahy

- Velikost populace
  - malá populace riskuje příliš malé pokrytí prohledávacího prostoru
  - velká populace má velké výpočetní nároky
  - dle empirických výsledků
    - velikost populace kolem 30 často adekvátní
    - velikost populace 20-100 je běžná
- Mutace: obvykle použita s velmi malou pravděpodobností
  - např. mutace jednoho genu jedince

Matematické programování

16. května 2022

Matematické programování

- Řada rozvrhovacích problémů může být formulována jako matematické programy
- Lineární programování, nelineární programování, celočíselné programování
- Předměty
  - PřF:M0160 Optimalizace
  - ESF:BKM_OPRO Optimalizace a rozhodování

Lineární programování

- Lineární program (LP)

  Minimalizace $c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$

  za předpokladu

  $$a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n \geq b_1$$
  $$a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n \geq b_2$$
  $$\ldots$$
  $$a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n \geq b_m$$
  $$x_j \geq 0 \text{ pro } j = 1, \ldots, n$$
  $$x_j \in \mathbb{R} \text{ pro } j = 1, \ldots, n$$

  V maticovém zápisu: minimalizace $\overline{c}\,\overline{x}$

  za předpokladu $\mathbb{A}\overline{x} \geq \overline{b}$

  $\overline{x} \geq 0$

Lineární programování: příklad

- Výrobce vyrábí 3 výrobky A, B, C, na které spotřebuje materiál M a pracovní dobu P
- Maximalizujeme denní "Zisk"za předpokladu, že platí:
  - zisk z 1 kusu A = 500 Kč, spotřebujeme 4M a 2P.
  - zisk z 1 kusu B = 800 Kč, spotřebujeme 1M a 5P.
  - zisk z 1 kusu C = 300 Kč, spotřebujeme 2M a 1P.
  - přitom platí denní omezení materiálu M$\leq$30 kusů a pracovních hodin P$\leq$48 hodin (např. 6 lidí pracujících 8 hodin denně)
- Jak lze použít obecný LP vzorec?

Lineární programování: příklad

- Výrobce vyrábí 3 výrobky A, B, C, na které spotřebuje materiál M a pracovní dobu P
- Maximalizujeme denní "Zisk"za předpokladu, že platí:
  - zisk z 1 kusu A = 500 Kč, spotřebujeme 4M a 2P.
  - zisk z 1 kusu B = 800 Kč, spotřebujeme 1M a 5P.
  - zisk z 1 kusu C = 300 Kč, spotřebujeme 2M a 1P.
  - přitom platí denní omezení materiálu M$\leq$30 kusů a pracovních hodin P$\leq$48 hodin (např. 6 lidí pracujících 8 hodin denně)
- Jak lze použít obecný LP vzorec?
  - max(Zisk) = 500*A + 800*B + 300*C ($c_1 = 500, x_1 =$ A,...)

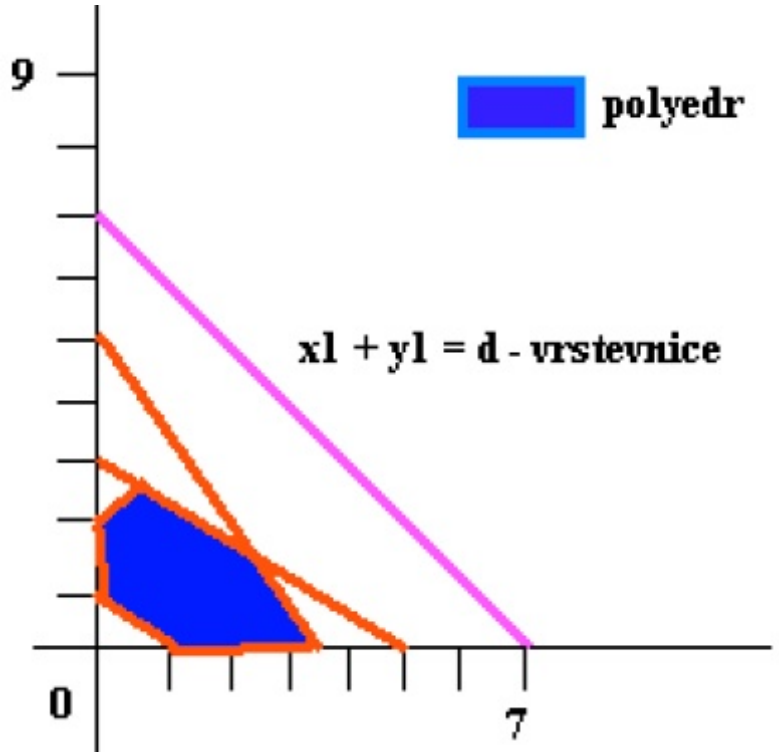
Lineární programování: příklad

- Výrobce vyrábí 3 výrobky A, B, C, na které spotřebuje materiál M a pracovní dobu P
- Maximalizujeme denní "Zisk"za předpokladu, že platí:
  - zisk z 1 kusu A = 500 Kč, spotřebujeme 4M a 2P.
  - zisk z 1 kusu B = 800 Kč, spotřebujeme 1M a 5P.
  - zisk z 1 kusu C = 300 Kč, spotřebujeme 2M a 1P.
  - přitom platí denní omezení materiálu M$\leq$30 kusů a pracovních hodin P$\leq$48 hodin (např. 6 lidí pracujících 8 hodin denně)
- Jak lze použít obecný LP vzorec?
  - max(Zisk) = 500*A + 800*B + 300*C ($c_1 = 500, x_1 =$ A,...)
  - 4*A + 1*B + 2*C $\leq$ 30 (omezení materiálu)
  - 2*A + 5*B + 1*C $\leq$ 48 (omezení prac. hodin)
- Jak bude vypadat výsledek? V jakých proměnných budeme mít uloženou odpověď na naši otázku? Co budou vyjadřovat?
- Bude to použitelné v praxi?
- Max vs. min není problém, neboť platí:
  $\max f(x) = -min(-f(x))$ pro $x \in \mathbb{R}^n$

Lineární programování: metody řešení

- Pro malé 2D, 3D problémy si často vystačíme s grafickým řešením
  - cíl je rovnice s parametrem = vrstevnice v ploše
  - omezení jsou poloroviny
  - oblast řešení je průnik polorovin (polyedr, tj. mnohostěn)
  - řešení je průnik rovnice s parametrem a vzniklého polyedru

- Simplexová metoda
  - efektivní nalezení řešení
  - většinou polynomiální čas
  - použití v praxi pro řešení rozsáhlých problémů
- Elipsoidová metoda
- Metoda vnitřních bodů



Celočíselné programování

- Celočíselné programování (integer programming IP)
  - lineární programování + všechny proměnné celočíselné
- Mixed-integer programming (MIP)
  - použity celočíselné i reálné obory hodnot proměnných
- Mnohem obtížnější než lineární programování
- Pro rozvrhování mnohem užitečnější

Celočíselné programování a metody řešení

- Pracuje se s LP relaxací celočíselného programu,
  tj. z celočíselného programu jsou odstraněna omezení požadující řešení z oboru celých čísel a řešíme lineární programy a řešíme lineární programy
- Metoda řezné roviny (cutting plane)
  - generována přídavná lineární omezení (řezné roviny),
    která musí být splněna v celočíselném řešení
  - přídavná omezení zužují množinu přípustných řešení
    při zachování celočíselných řešení
  - řešení LP relaxace celočíselného programu s přídavnými omezeními
  - pokud nenalezeno řešení celočíselné, přidáváme dalsí řezné roviny a opakujeme postup

Celočíselné programování a metody řešení

- Pracuje se s LP relaxací celočíselného programu,
  tj. z celočíselného programu jsou odstraněna omezení požadující řešení z oboru celých čísel a řešíme lineární programy a řešíme lineární programy
- Metoda řezné roviny (cutting plane)
  - generována přídavná lineární omezení (řezné roviny),
    která musí být splněna v celočíselném řešení
  - přídavná omezení zužují množinu přípustných řešení
    při zachování celočíselných řešení
  - řešení LP relaxace celočíselného programu s přídavnými omezeními
  - pokud nenalezeno řešení celočíselné, přidáváme dalsí řezné roviny a opakujeme postup
- Metoda větví a mezí (branch and bound)
  - větvení na rozhodovacích proměnných
    ($x_j = r$ v LP řešení pak přidáme $x_j \leq \lfloor r \rfloor$, $x_j \geq \lceil r \rceil$)
  - lineární programy s přidanými omezeními poskytují hranice (meze)
- Hybridní metody
  - kombinace různých metod
  - např. kombinace metody větví a mezí a řezných rovin (branch and cut)

Ukázka problému: rozvrhování směn

- Směna: množina period, kdy zaměstnanec pracuje
  - často je směna chápána jako množina po sobě jdoucích period
  - př. denní směna 6:00-18:00, noční směna 18:00-6:00
- Problém rozvrhování směn
  - cyklus je dán předem
    - např. 1 den je cyklus, časová jednotka (perioda) je hodina
  - je dáno několik vzorků směn s odlišnou cenou
  - cíl je minimalizovat celkovou cenu

Ukázka problému: rozvrhování směn

- Směna: množina period, kdy zaměstnanec pracuje
  - často je směna chápána jako množina po sobě jdoucích period
  - př. denní směna 6:00-18:00, noční směna 18:00-6:00
- Problém rozvrhování směn
  - cyklus je dán předem
    - např. 1 den je cyklus, časová jednotka (perioda) je hodina
  - je dáno několik vzorků směn s odlišnou cenou
  - cíl je minimalizovat celkovou cenu

- Problém si popíšeme jako matematický celočíselný program

$$\text{minimalizace} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{za předpokladu:} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad \forall i : 1 \leq i \leq m$$

$$x_j \geq 0 \quad \forall j : 1 \leq j \leq n$$

$$x_j \in \mathbb{Z} \quad \forall j : 1 \leq j \leq n$$

Formulace problému

- m period (hodin)
  - př. od 10:00 do 21:00
- V periodě $i, i = 1, \ldots, m$ je potřeba b_i zaměstnanců
  - př. 10:00: 3, 11:00: 4, 12:00 6, ...
- n vzorků směn
  - př. 10:00 – 18:00, 13:00 – 21:00, 18:00 – 21:00, ...
- Každý zaměstnanec přiřazen právě jednomu vzorku

Formulace problému

- m period (hodin)
  - př. od 10:00 do 21:00
- V periodě $i, i = 1, \ldots, m$ je potřeba b_i zaměstnanců
  - př. 10:00: 3, 11:00: 4, 12:00 6, ...
- n vzorků směn
  - př. 10:00 – 18:00, 13:00 – 21:00, 18:00 – 21:00, ...
- Každý zaměstnanec přiřazen právě jednomu vzorku
- Vzorek směny j je vektor $(a_{1j}, a_{2j}, \ldots, a_{mj})$
  - $a_{ij} = 1$: i je pracovní perioda
  - $a_{ij} = 0$: jinak
  - př. směna 18:00 – 21:00 odpovídá (0,0,0,0,0,0,0,0,1,1,1) pro 10:00 – 21:00

Formulace problému

- m period (hodin)
  - př. od 10:00 do 21:00
- V periodě $i, i = 1, \ldots, m$ je potřeba b_i zaměstnanců
  - př. 10:00: 3, 11:00: 4, 12:00 6, ...
- n vzorků směn
  - př. 10:00 – 18:00, 13:00 – 21:00, 18:00 – 21:00, ...
- Každý zaměstnanec přiřazen právě jednomu vzorku
- Vzorek směny j je vektor $(a_{1j}, a_{2j}, \ldots, a_{mj})$
  - $a_{ij} = 1$: i je pracovní perioda
  - $a_{ij} = 0$: jinak
  - př. směna 18:00 – 21:00 odpovídá (0,0,0,0,0,0,0,0,1,1,1) pro 10:00 – 21:00
- Cena přiřazení zaměstnance na směnu j: c_j
- x_j: proměnná reprezentující počet zaměstnanců přiřazených ke směně j
- Cíl: minimalizace celkové ceny přiřazených zaměstnanců

Formulace problému a řešení

- m period (hodin)
  - př. od 10:00 do 21:00
- V periodě $i, i = 1, \ldots, m$ je potřeba b_i zaměstnanců
  - př. 10:00: 3, 11:00: 4, 12:00 6, ...
- n vzorků směn
  - př. 10:00 – 18:00, 13:00 – 21:00, 18:00 – 21:00, ...
- Každý zaměstnanec přiřazen právě jednomu vzorku
- Vzorek směny j je vektor $(a_{1j}, a_{2j}, \ldots, a_{mj})$
  - $a_{ij} = 1$: i je pracovní perioda
  - $a_{ij} = 0$: jinak
  - př. směna 18:00 – 21:00 odpovídá (0,0,0,0,0,0,0,0,1,1,1) pro 10:00 – 21:00
- Cena přiřazení zaměstnance na směnu j: c_j
- x_j: proměnná reprezentující počet zaměstnanců přiřazených ke směně j
- Cíl: minimalizace celkové ceny přiřazených zaměstnanců

- Problém je NP-těžký, nicméně
  - $\mathbb{A}$ má speciální tvar, kde směna je dána jako kontinuální posloupnost 1
  - platí: řešení tohoto lineárního programu je vždy celočíselné

Příklad rozvrhování směn v obchodě

- Obchod otevřen
  od 10:00 do 21:00
- 5 vzorků (typů) směny

Vzorek	Doba	Hodin	Cena
1	10-18	8	50
2	13-21	8	60
3	12-18	6	30
4	10-13	3	15
5	18-21	3	16

- Požadavky na počet zaměstnanců v obchodě

Hodina	10	11	12	13	14	15	16	17	18	19	20
Počet zaměstnanců	3	4	6	4	7	8	7	6	4	7	8

Příklad rozvrhování směn v obchodě

- Obchod otevřen od 10:00 do 21:00
- 5 vzorků (typů) směny

Vzorek	Doba	Hodin	Cena
1	10-18	8	50
2	13-21	8	60
3	12-18	6	30
4	10-13	3	15
5	18-21	3	16

- Požadavky na počet zaměstnanců v obchodě

Hodina	10	11	12	13	14	15	16	17	18	19	20
Počet zaměstnanců	3	4	6	4	7	8	7	6	4	7	8

- Lineární relaxace

$$\overline{c} = (50, 60, 30, 15, 16) \qquad \mathbb{A} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad \overline{b} = \begin{bmatrix} 3 \\ 4 \\ 6 \\ 4 \\ 7 \\ 8 \\ 7 \\ 6 \\ 4 \\ 7 \\ 8 \end{bmatrix}$$

Příklad rozvrhování směn v obchodě

- Obchod otevřen od 10:00 do 21:00
- 5 vzorků (typů) směny

Vzorek	Doba	Hodin	Cena
1	10-18	8	50
2	13-21	8	60
3	12-18	6	30
4	10-13	3	15
5	18-21	3	16

- Požadavky na počet zaměstnanců v obchodě

Hodina	10	11	12	13	14	15	16	17	18	19	20
Počet zaměstnanců	3	4	6	4	7	8	7	6	4	7	8

- Lineární relaxace

$$\overline{c} = (50, 60, 30, 15, 16) \qquad \mathbb{A} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad \overline{b} = \begin{bmatrix} 3 \\ 4 \\ 6 \\ 4 \\ 7 \\ 8 \\ 7 \\ 6 \\ 4 \\ 4 \\ 7 \\ 8 \end{bmatrix}$$

- Řešení $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 8, 4, 8)$

Omezující podmínky (pokračování)

16. května 2022

Prohledávání/přiřazování

Konzistenční techniky jsou (obvykle) neúplné

$\Rightarrow$ potřeba prohledávací algoritmus, který vyřeší "zbytek"

- př. X in 1..2, Y in 1..2, Z in 1..2, X$\neq$Y, Y$\neq$Z, X$\neq$Z

Prohledávání/přiřazování

Konzistenční techniky jsou (obvykle) neúplné

$\Rightarrow$ potřeba prohledávací algoritmus, který vyřeší "zbytek"

- př. X in 1..2, Y in 1..2, Z in 1..2, X$\neq$Y, Y$\neq$Z, X$\neq$Z
  AC neodstraní žádnou hodnotu ale problem je nekonzistentní

Prohledávání/přiřazování

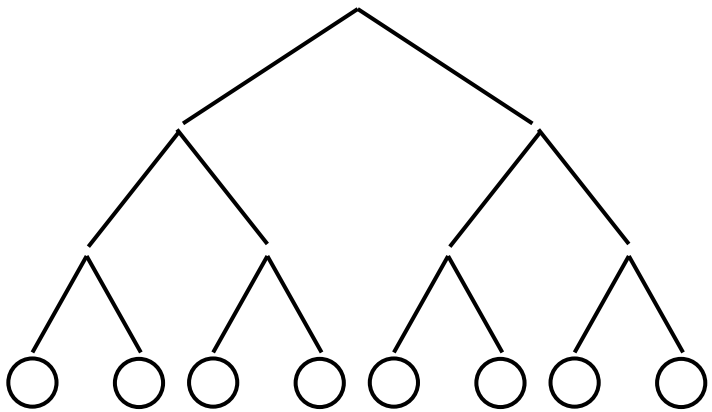
Konzistenční techniky jsou (obvykle) neúplné

$\Rightarrow$ potřeba prohledávací algoritmus, který vyřeší "zbytek"

- př. X in 1..2, Y in 1..2, Z in 1..2, X$\neq$Y, Y$\neq$Z, X$\neq$Z
  AC neodstraní žádnou hodnotu ale problem je nekonzistentní

Přiřazovaní *(labeling)*

- prohledávání do hloubky (DFS/BT)
  - přiřaď hodnotu do proměnné
  - propaguj = uděléj
    problém lokálně konzistentní
  - vrať se v případě neúspěchu



Prohledávání/přiřazování

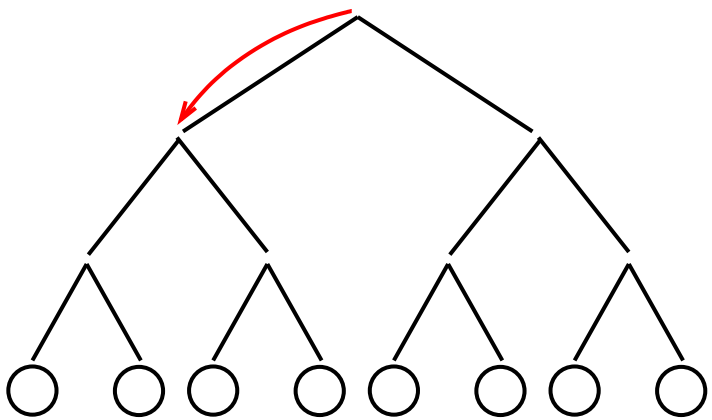
Konzistenční techniky jsou (obvykle) neúplné

$\Rightarrow$ potřeba prohledávací algoritmus, který vyřeší "zbytek"

- př. X in 1..2, Y in 1..2, Z in 1..2, X$\neq$Y, Y$\neq$Z, X$\neq$Z
  AC neodstraní žádnou hodnotu ale problem je nekonzistentní

Přiřazovaní *(labeling)*

- prohledávání do hloubky (DFS/BT)
  - přiřaď hodnotu do proměnné
  - propaguj = uděléj
    problém lokálně konzistentní
  - vrať se v případě neúspěchu



Prohledávání/přiřazování

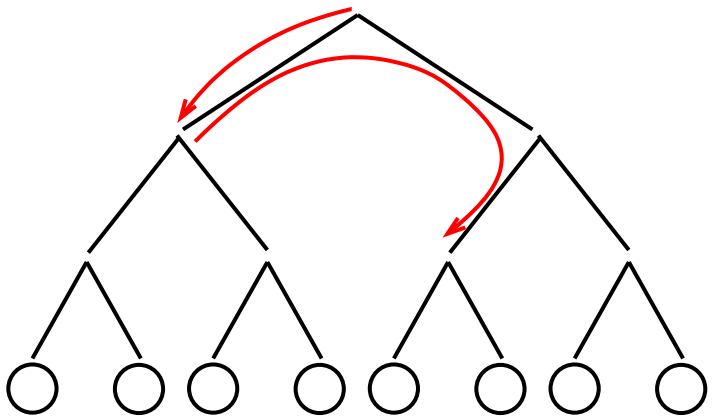
Konzistenční techniky jsou (obvykle) neúplné

$\Rightarrow$ potřeba prohledávací algoritmus, který vyřeší "zbytek"

- př. X in 1..2, Y in 1..2, Z in 1..2, X$\neq$Y, Y$\neq$Z, X$\neq$Z
  AC neodstraní žádnou hodnotu ale problem je nekonzistentní

Přiřazovaní *(labeling)*

- prohledávání do hloubky (DFS/BT)
  - přiřaď hodnotu do proměnné
  - propaguj = uděléj
    problém lokálně konzistentní
  - vrať se v případě neúspěchu



Prohledávání/přiřazování

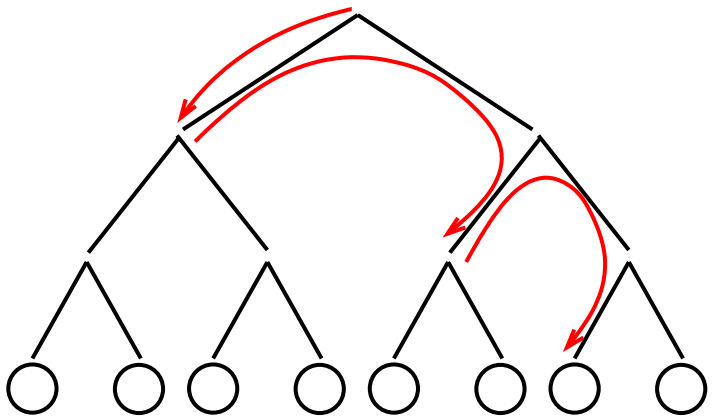
Konzistenční techniky jsou (obvykle) neúplné

$\Rightarrow$ potřeba prohledávací algoritmus, který vyřeší "zbytek"

- př. X in 1..2, Y in 1..2, Z in 1..2, X$\neq$Y, Y$\neq$Z, X$\neq$Z
  AC neodstraní žádnou hodnotu ale problem je nekonzistentní

Přiřazovaní *(labeling)*

- prohledávání do hloubky (DFS/BT)
  - přiřaď hodnotu do proměnné
  - propaguj = uděléj
    problém lokálně konzistentní
  - vrať se v případě neúspěchu



Prohledávání/přiřazování

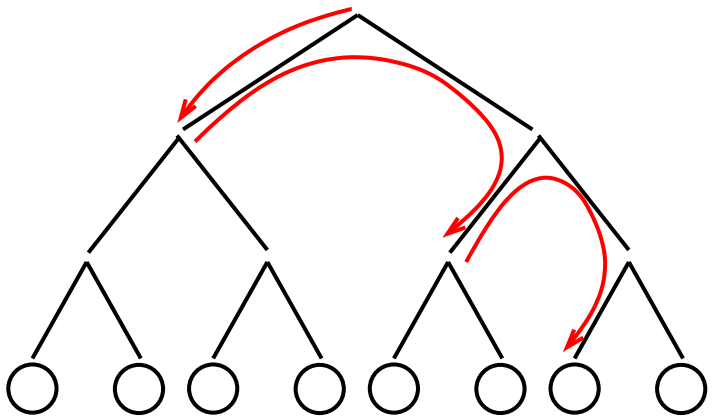
Konzistenční techniky jsou (obvykle) neúplné

$\Rightarrow$ potřeba prohledávací algoritmus, který vyřeší "zbytek"

- př. X in 1..2, Y in 1..2, Z in 1..2, X$\neq$Y, Y$\neq$Z, X$\neq$Z
  AC neodstraní žádnou hodnotu ale problem je nekonzistentní

Přiřazovaní *(labeling)*

- prohledávání do hloubky (DFS/BT)
  - přiřaď hodnotu do proměnné
  - propaguj = uděléj
    problém lokálně konzistentní
  - vrať se v případě neúspěchu



- X in 1..5 $\equiv$ X=1 $\vee$ X=2 $\vee$ X=3 $\vee$ X=4 $\vee$ X=5

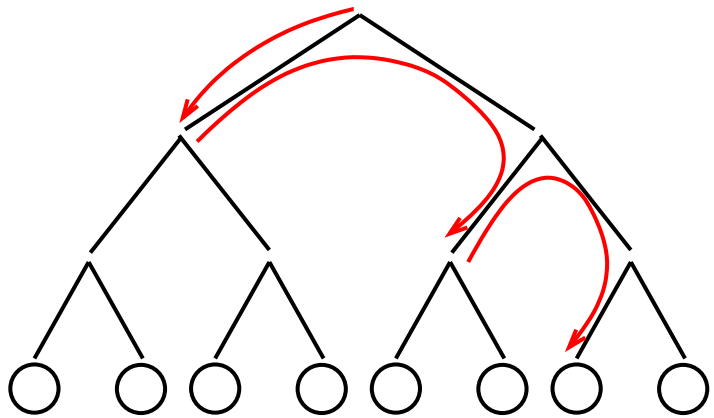
Prohledávání/přiřazování

Konzistenční techniky jsou (obvykle) neúplné
$\Rightarrow$ potřeba prohledávací algoritmus, který vyřeší "zbytek"

- př. X in 1..2, Y in 1..2, Z in 1..2, X$\neq$Y, Y$\neq$Z, X$\neq$Z
  AC neodstraní žádnou hodnotu ale problem je nekonzistentní

Přiřazovaní *(labeling)*

- prohledávání do hloubky (DFS/BT)
  - přiřaď hodnotu do proměnné
  - propaguj = uděléj
    problém lokálně konzistentní
  - vrať se v případě neúspěchu



- X in 1..5 $\equiv$ X=1 $\vee$ X=2 $\vee$ X=3 $\vee$ X=4 $\vee$ X=5

Obecně: prohledávací algoritmus řeší zbylé disjunkce

- X=1 $\vee$ X$\neq$1 standardní přiřazování
- X<3 $\vee$ X$\geq$3 dělení domén
- X<Y $\vee$ X$\geq$Y uspořádání proměnných

Kostra ohodnocování

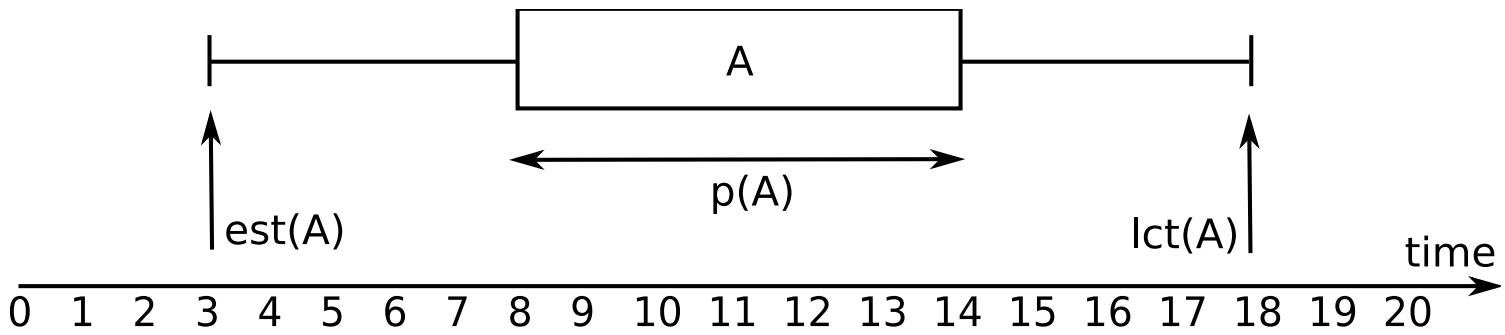
- Prohledávání do hloubky je kombinováno s AC, které omezuje prohledávaný prostor
- Technika pohledu dopředu (MAC)
- procedure labeling(V,D,C)
  if (všechny proměnné z V přiřazeny) then return V
  vyber dosud nepřiřazenou proměnnou x z V
  for (každou hodnotu v z D_x) do
  (TestOk,D') := consistent(V,D,C $\cup$ {x=v})
  if TestOk=true then R := labeling(V,D',C)
  if R$\neq$fail then return R
  return fail
  end labeling
- Před labeling je spuštěn iniciální běh konzistenčních algoritmů, aby byla zajištěna iniciální konzistence

Rozvrhování jako CSP: čas

Aktivita A: entita vyžadující prostor (zdroje) a čas

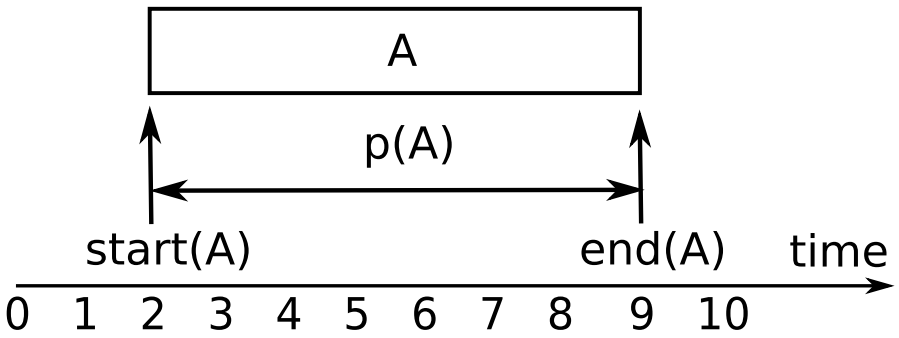
Proměnné a jejich domény pro časové přiřazení aktivity

- start(A): startovní čas aktivity
  - aktivita nemůže začít před svým termínem dostupnosti
  - est(A) = min(start(A)), earliest start time/nejdřívější startovní čas
- end(A): čas skončení aktivity
  - aktivita musí skončit před svým deadline
  - lct(A) = max(end(A)), latest completion time/nejpozdější koncový čas
- p(A): doba provádění aktivity
  - start(A) = {est(A), . . . , (lct(A)-p(A))}
  - end(A) = {(est(A)+p(A)), . . . , lct(A)}



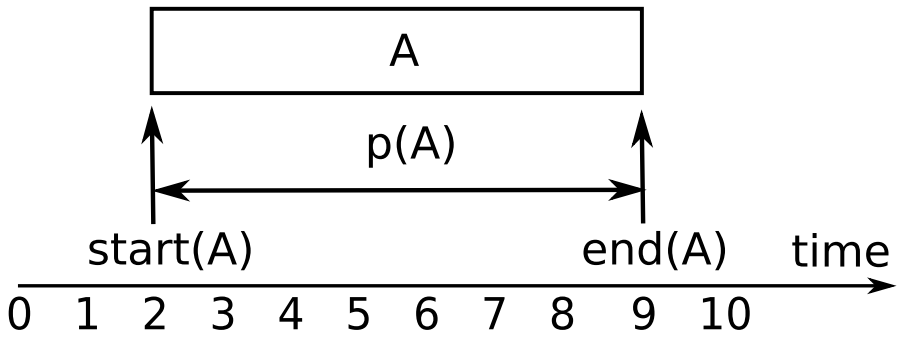
Rozvrhování jako CSP: základní omezení I.

- Nepřerušitelná aktivita: žádné přerušení během jejího zpracování
  - start(A) + p(A) = end(A)

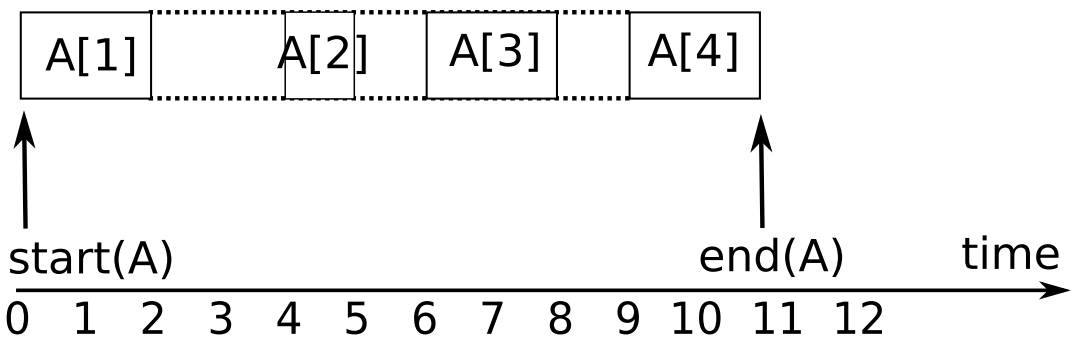


Rozvrhování jako CSP: základní omezení I.

- Nepřerušitelná aktivita: žádné přerušení během jejího zpracování
  - start(A) + p(A) = end(A)



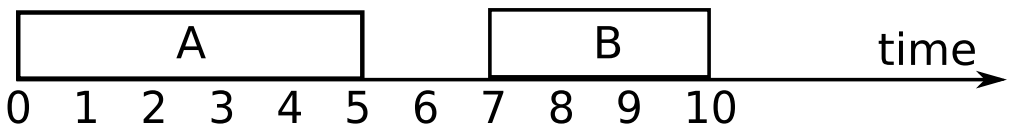
- Přerušitelná aktivita: může být přerušena během svého zpracování
  - start(A) + p(A) $\leq$ end(A)



$$p(A) = p(A[1]) + p(A[2]) + p(A[3]) + p(A[4])$$

Rozvrhování jako CSP: základní omezení II.

- Seřazení A<<B aktivity A,B
  (také: precedenční omezení mezi aktivitami A,B)
  - end(A) $\leq$ start(B)



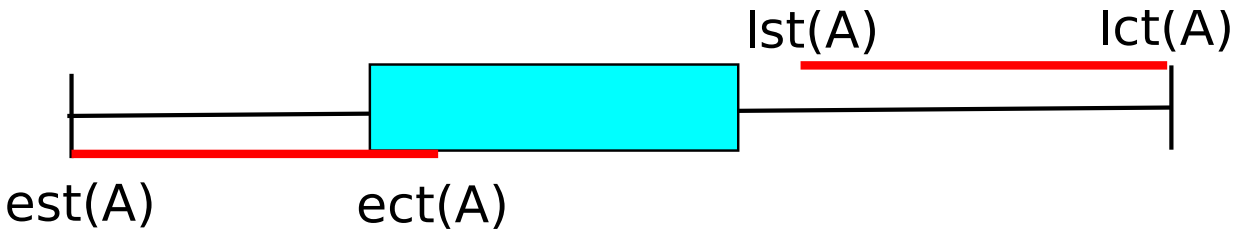
- Disjunktivní omezení: nepřekrývání aktivit A a B
  - nepřerušitelné aktivity
  - A<<B or B<<A
  - end(A) $\leq$ start(B) or end(B) $\leq$ start(A)
  - viz dále unární zdroje

Rozvrhování jako CSP: zdroje

Doménové proměnné pro zdroje:

- cap(A): požadovaná kapacita zdroje aktivitou A
  - unární/disjunktivní zdroje
    - v daném čase může být zpracována maximálně jedna aktivita
  - kumulativní zdroje
    - několik aktivit se může zpracovávat paralelně, ovšem za předpokladu, že není překročena kapacita zdroje
  - produkovatelné/spotřebovatelné zdroje
    - aktivita konzumuje (snižuje) nebo produkuje (navyšuje) aktuální množství zdroje
    - na zdroji musí zbýt nějaká minimální volná kapacita a maximální kapacita zdroje nemůže být překročena
    - příklad: reservoár
- resource(A): alternativní zdroje pro A
  - pro zpracování A vybereme jeden z alternativních zdrojů
  - jednotlivé hodnoty z domény resource(A) odkazují na konkrétní zdroje

Značení

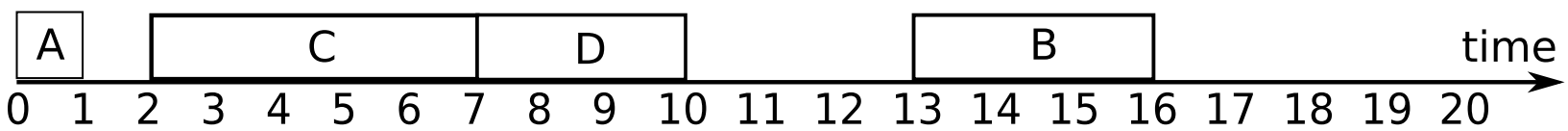


- est(A) nejdřívější startovní čas aktivity A (earliest start time)
- ect(A) nejdřívější koncový čas aktivity A (earliest completion time)
- lst(A) nejpozdější startovní čas aktivity A (latest start time)
- lct(A) nejpozdější koncový čas aktivity A (latest completion time)

- Ω je množina aktivit
- $p(\Omega) = \sum_{A \in \Omega} p(A)$
- $est(\Omega) = \min\{est(A) \mid A \in \Omega\}$
- $lct(\Omega) = \max\{lct(A) \mid A \in \Omega\}$

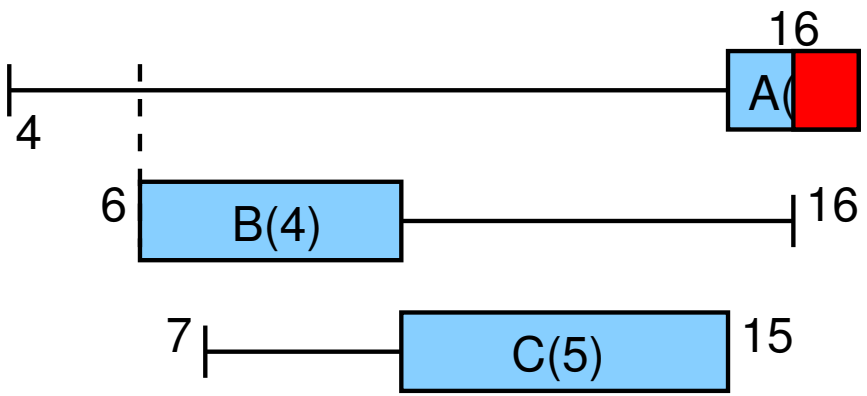
Unární zdroje

- Aktivivity se nemohou překrývat
  - v daném čase běží maximálně jedna aktivita, proto se těmto zdrojům říká unární
  - Grahamova klasifikace: jeden stroj
- Předpokládáme, že aktivity jsou nepřerušitelné
  - nepřerušitelná aktivita zabírá zdroj od svého startu až do ukončení
- Jednoduchý model s disjunktními omezeními
  - $A << B \vee B << A$
    $end(A) \leq start(B) \vee end(B) \leq start(A)$
  - těmto zdrojům se proto někdy říká disjunktní



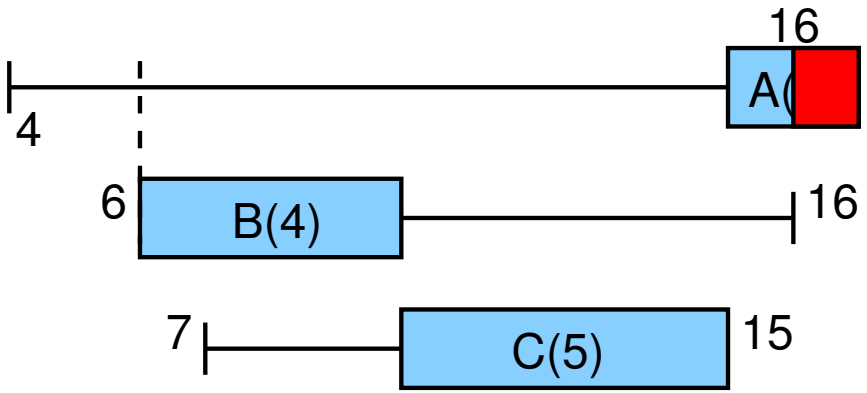
Hledání hran *(edge finding)*

- Baptiste & Le Pape (1996)
- Co se stane, pokud nebude aktivita A zpracována jako první?

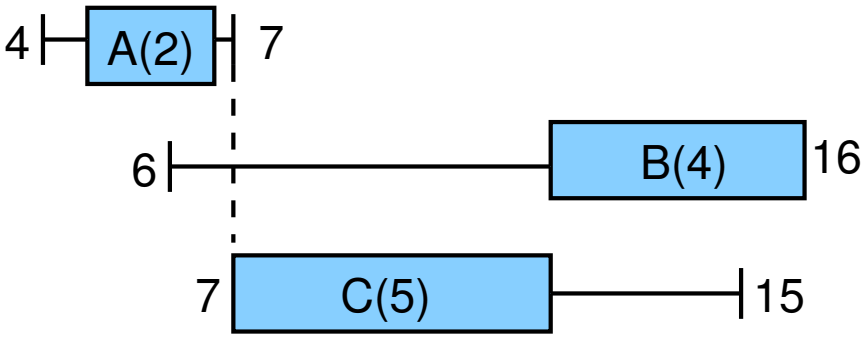


Hledání hran *(edge finding)*

- Baptiste & Le Pape (1996)
- Co se stane, pokud nebude aktivita A zpracována jako první?

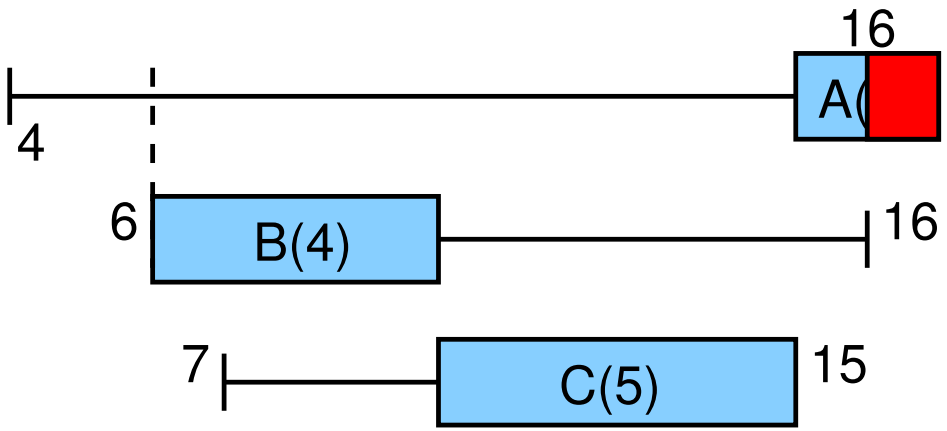


- Pro A,B,C není dost času, a tedy aktivita A musí být první



Hledání hran: příklad s odvozovacími pravidly

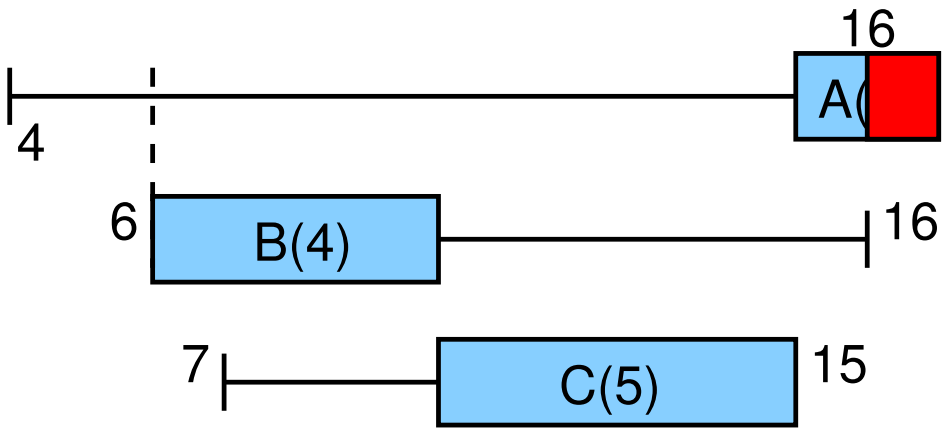
- $lct(\Omega \cup \{A\}) - est(\Omega) < p(\Omega \cup \{A\}) \Rightarrow A << \Omega$



$\Omega = \{B, C\}$

Hledání hran: příklad s odvozovacími pravidly

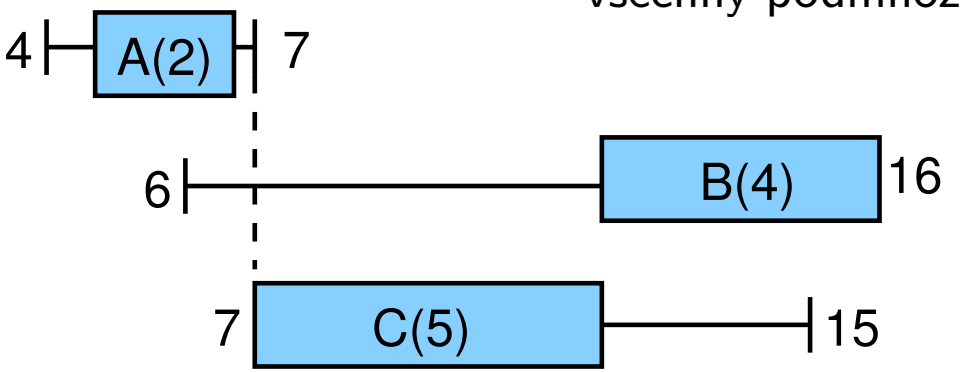
- $lct(\Omega \cup \{A\}) - est(\Omega) < p(\Omega \cup \{A\}) \Rightarrow A << \Omega$



$\Omega = \{B, C\}$

- $A << \Omega \Rightarrow end(A) \leq \min\{lct(\Omega') - p(\Omega') \mid \Omega' \subseteq \Omega\}$

všechny podmnožiny Ω musí stihnout své zpracování



Hledání hran: všechna odvozovací pravidla

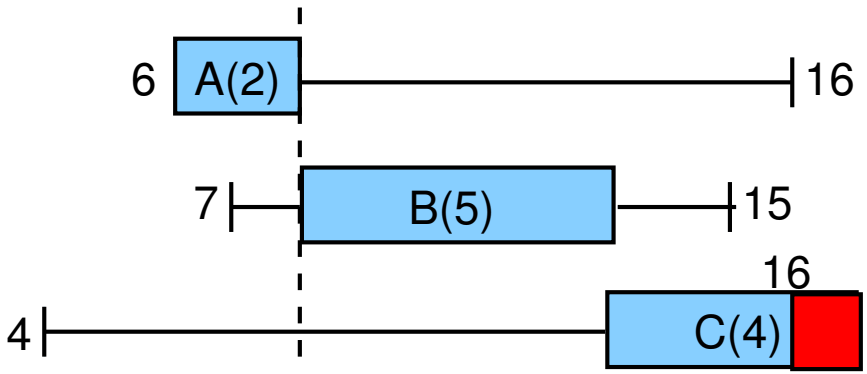
- Zopakujme tedy odvozovací pravidla: pro $end(A)$
  (co se stane, pokud nebude aktivita A zpracována jako první?)
  - $lct(\Omega \cup \{A\}) - est(\Omega) < p(\Omega \cup \{A\})$
    $\Rightarrow A << \Omega$
  - $A << \Omega \Rightarrow$
    $end(A) \leq \min\{lct(\Omega') - p(\Omega') \mid \Omega' \subseteq \Omega\}$
- Symetrická odvozovací pravidla: pro $start(A)$
  (co se stane, pokud nebude aktivita A zpracována jako poslední?)
  - $lct(\Omega) - est(\Omega \cup \{A\}) < p(\Omega \cup \{A\})$
    $\Rightarrow \Omega << A$
  - $\Omega << A \Rightarrow$
    $start(A) \geq \max\{est(\Omega') + p(\Omega') \mid \Omega' \subseteq \Omega\}$

Hledání hran: všechna odvozovací pravidla

- Zopakujme tedy odvozovací pravidla: pro $end(A)$
  (co se stane, pokud nebude aktivita A zpracována jako první?)
  - $lct(\Omega \cup \{A\}) - est(\Omega) < p(\Omega \cup \{A\})$
    $\Rightarrow A << \Omega$
  - $A << \Omega \Rightarrow$
    $end(A) \leq \min\{lct(\Omega') - p(\Omega') \mid \Omega' \subseteq \Omega\}$
- Symetrická odvozovací pravidla: pro $start(A)$
  (co se stane, pokud nebude aktivita A zpracována jako poslední?)
  - $lct(\Omega) - est(\Omega \cup \{A\}) < p(\Omega \cup \{A\})$
    $\Rightarrow \Omega << A$
  - $\Omega << A \Rightarrow$
    $start(A) \geq \max\{est(\Omega') + p(\Omega') \mid \Omega' \subseteq \Omega\}$
- V praxi:
  - celkem existuje $n \cdot 2^n$ párů A,Ω (příliš mnoho!)
  - Carlier & Pinson 1994, Vilím & Barták & Čepek 2004
    algoritmus s časovou složitostí $O(n \log n)$
    (je nutné kontrolovat pouze některé páry A,Ω)

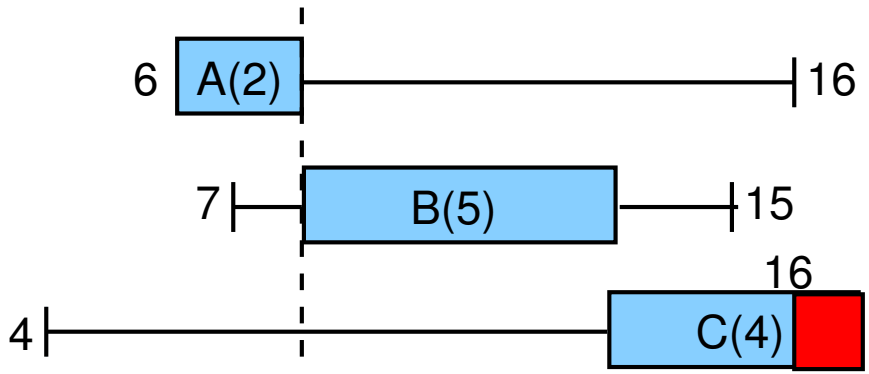
Ne-první/ne-poslední *(not-first/not-last)*

- Torres & Lopez 2000
- Co se stane, pokud aktivita A bude zpracována jako první?

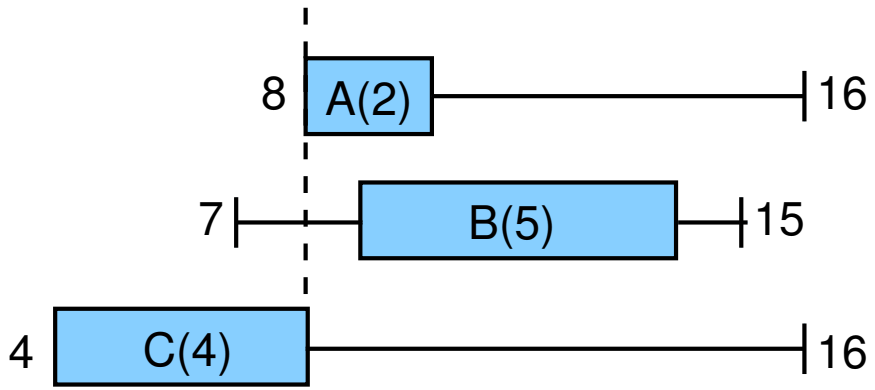


Ne-první/ne-poslední *(not-first/not-last)*

- Torres & Lopez 2000
- Co se stane, pokud aktivita A bude zpracována jako první?

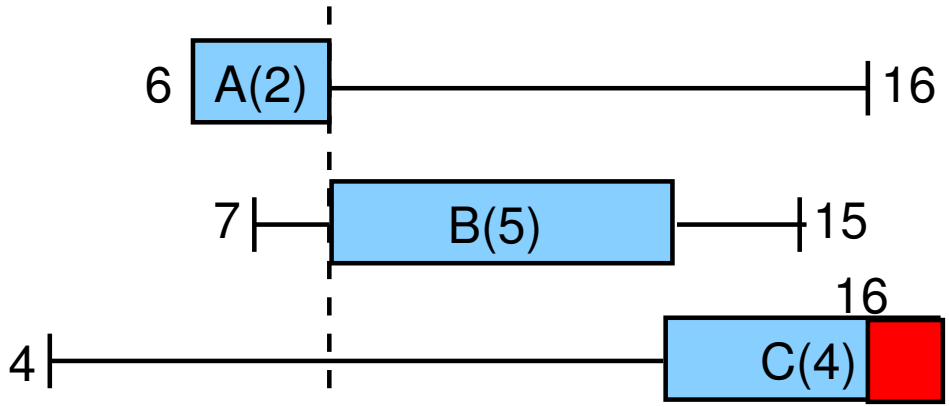


- Pro B a C není dost času, a tedy aktivita A nemůže být první



Ne-první/ne-poslední: př. s odvozovacími pravidly

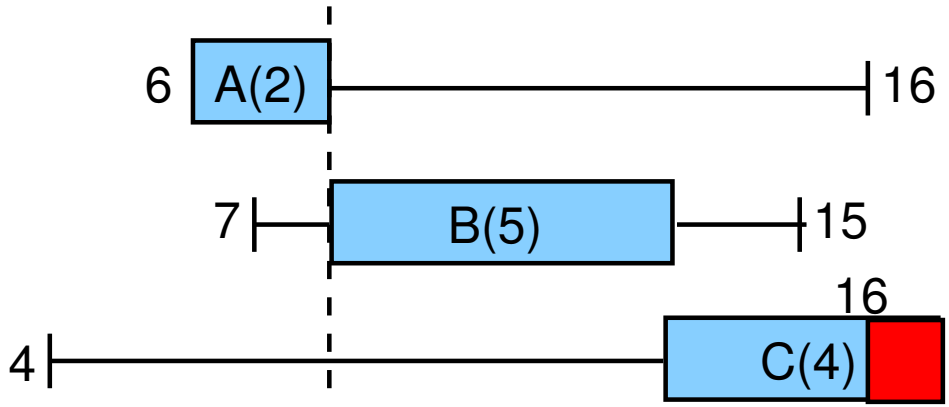
- $p(\Omega \cup \{A\}) > lct(\Omega) - est(A) \Rightarrow \neg A << \Omega$



$\Omega = \{B, C\}$

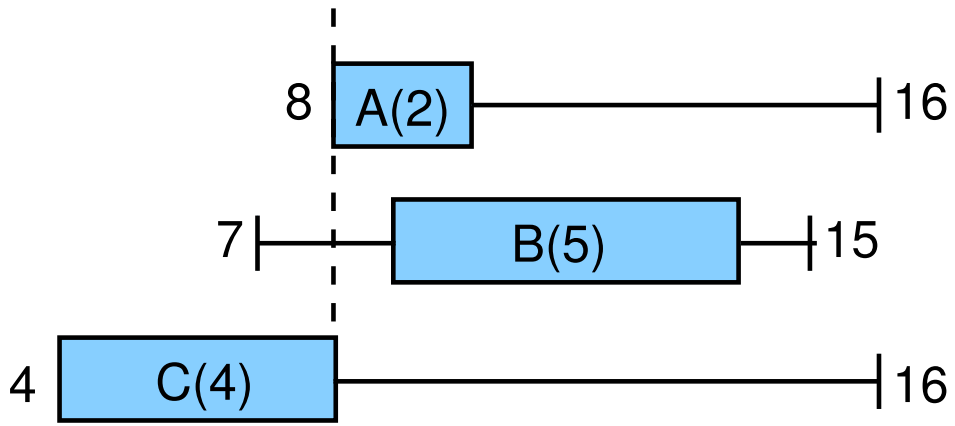
Ne-první/ne-poslední: př. s odvozovacími pravidly

- $p(\Omega \cup \{A\}) > lct(\Omega) - est(A) \Rightarrow \neg A << \Omega$



$\Omega = \{B, C\}$

- $\neg A << \Omega \Rightarrow start(A) \geq \min\{ect(B)|B \in \Omega\}$



Odvozovací pravidla pro ne-první/ne-poslední

- Zopakujme Ne-první pravidla:

  (co se stane, pokud aktivita A bude zpracována jako první?)

  - $lct(\Omega) - est(A) < p(\Omega \cup \{A\})$
    $\Rightarrow \neg A << \Omega$
  - $\neg A << \Omega$
    $\Rightarrow start(A) \geq \min\{ect(B) | B \in \Omega\}$

- Ne-poslední (symetrická) pravidla:

  (co se stane, pokud aktivita A bude zpracována jako poslední?)

  - $lct(A) - est(\Omega) < p(\Omega \cup \{A\})$
    $\Rightarrow \neg \Omega << A$
  - $\neg \Omega << A \Rightarrow$
    $end(A) \leq \max\{lst(B) | B \in \Omega\}$

Odvozovací pravidla pro ne-první/ne-poslední

- Zopakujme Ne-první pravidla:

  (co se stane, pokud aktivita A bude zpracována jako první?)

  - $lct(\Omega) - est(A) < p(\Omega \cup \{A\})$
    $\Rightarrow \neg A << \Omega$
  - $\neg A << \Omega$
    $\Rightarrow start(A) \geq \min\{ect(B) | B \in \Omega\}$

- Ne-poslední (symetrická) pravidla:

  (co se stane, pokud aktivita A bude zpracována jako poslední?)

  - $lct(A) - est(\Omega) < p(\Omega \cup \{A\})$
    $\Rightarrow \neg \Omega << A$
  - $\neg \Omega << A \Rightarrow$
    $end(A) \leq \max\{lst(B) | B \in \Omega\}$

- V praxi:
  - Vilím 2004: algoritmus s časovou složitostí $O(n \log n)$

Alternativní zdroje

Jak modelovat alternativní zdroje pro danou aktivitu?

Pro každý zdroj uděláme duplikát aktivity

- duplikát se účastní příslušných zdrojových podmínek,
  ale neomezuje další aktivity na daném zdroji
  - „neúspěch" u duplikátu znamená odstranění zdroje z domény proměnné resource(A) příslušné aktivity
  - odstranění zdroje z domény proměnné resource(A) znamená „smazání" odpovídajícího duplikátu

Alternativní zdroje

Jak modelovat alternativní zdroje pro danou aktivitu?

Pro každý zdroj uděláme duplikát aktivity

- duplikát se účastní příslušných zdrojových podmínek,
  ale neomezuje další aktivity na daném zdroji
  - „neúspěch" u duplikátu znamená odstranění zdroje z domény proměnné resource(A) příslušné aktivity
  - odstranění zdroje z domény proměnné resource(A) znamená „smazání" odpovídajícího duplikátu
- původní aktivita se účastní precedenčních podmínek
  (např. v rámci multi-operační úlohy)
- omezení časů u duplikátu se propaguje do originálu aktivity a naopak

Alternativní zdroje: odvozovací pravidla

Nechť A_u reprezentuje duplikát aktivity A na zdroji u∈resource(A), pak probíhají následující propagace:

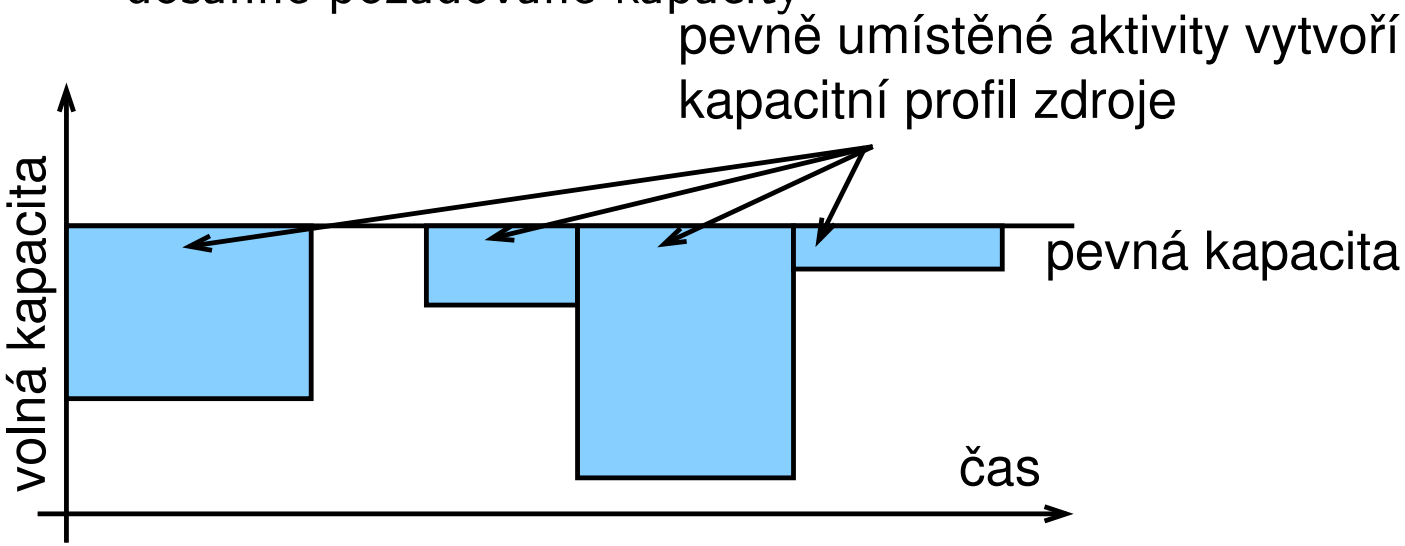
- u∈resource(A) $\Rightarrow$ start(A) $\leq$ start(A_u)
- u∈resource(A) $\Rightarrow$ end(A_u) $\leq$ end(A)

- start(A) $\geq$ min{start(A_u): u $\in$ resource(A)}
- end(A) $\leq$ max{end(A_u): u $\in$ resource(A)}

- neúspěch pro A_u $\Rightarrow$ resource(A)\{u}

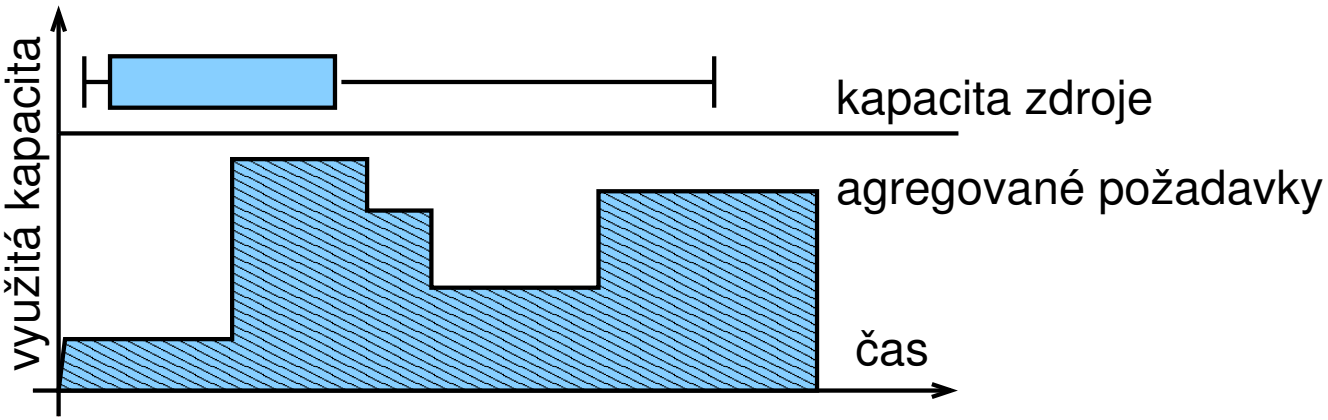
Kumulativní zdroje

- Každá aktivita využívá jistou kapacitu zdroje cap(A)
- Aktivity mohou být zpracovány paralelně, pokud není překročena kapacita zdroje
- Kapacita zdroje může být v čase proměnná
  - takové zdroje lze modelovat pomocí v čase neměnné kapacity, od které se odečte kapacita pevně umístěných aktivit, čímž se v každém čase dosáhne požadované kapacity



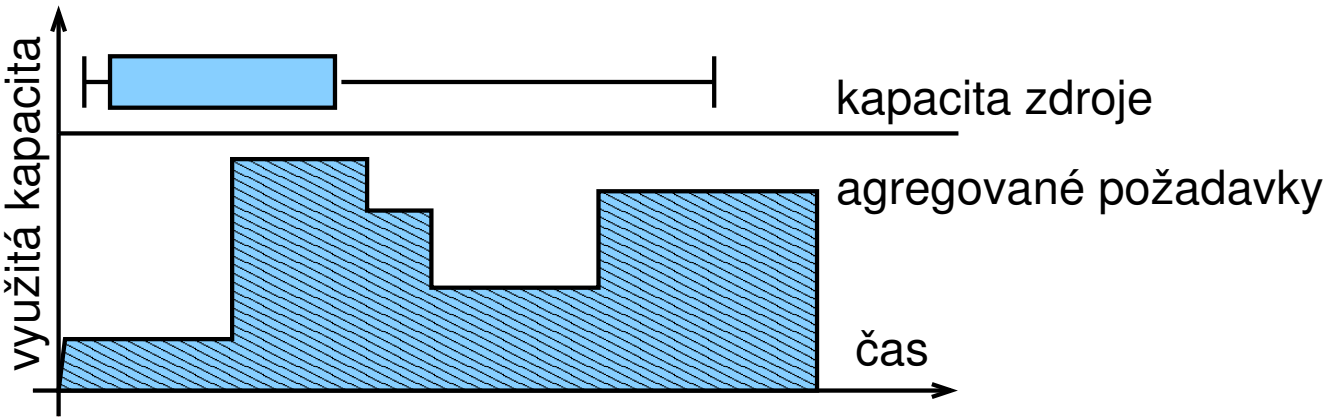
Agregované požadavky

- Baptiste et al. 2001
- Kdy je dostatečná kapacita pro zpracování aktivity?

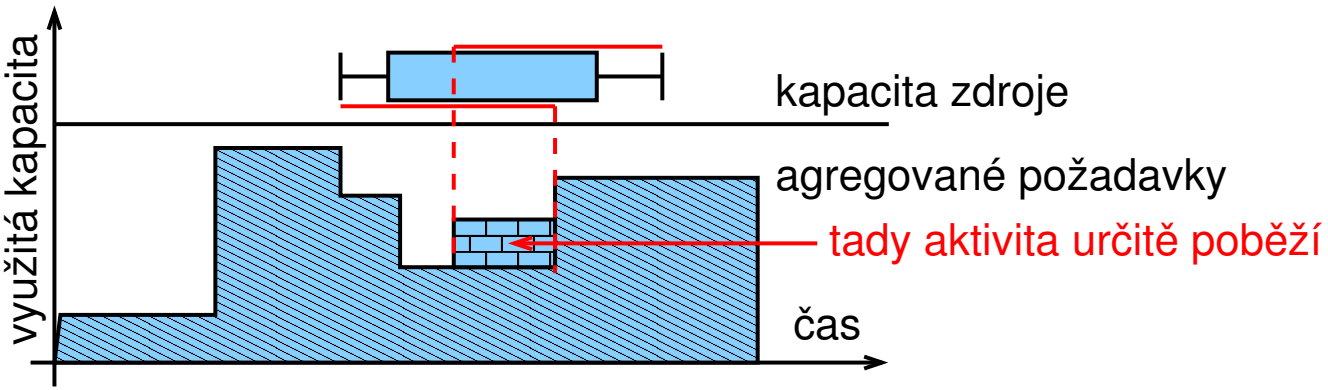


Agregované požadavky

- Baptiste et al. 2001
- Kdy je dostatečná kapacita pro zpracování aktivity?



- Jak se konstruuje graf agregovaných požadavků?



Podmínka tabulky *(timetable constraint)*

- Uvažujeme diskrétní čas
- Jak zajistit, že v žádném čase není překročena maximální kapacita?

$$\forall t \sum_{start(A_i) \leq t \leq end(A_i)} cap(A_i) \leq MaxCapacity$$

Podmínka tabulky *(timetable constraint)*

- Uvažujeme diskrétní čas
- Jak zajistit, že v žádném čase není překročena maximální kapacita?

$$\forall t \sum_{start(A_i) \leq t \leq end(A_i)} cap(A_i) \leq MaxCapacity$$

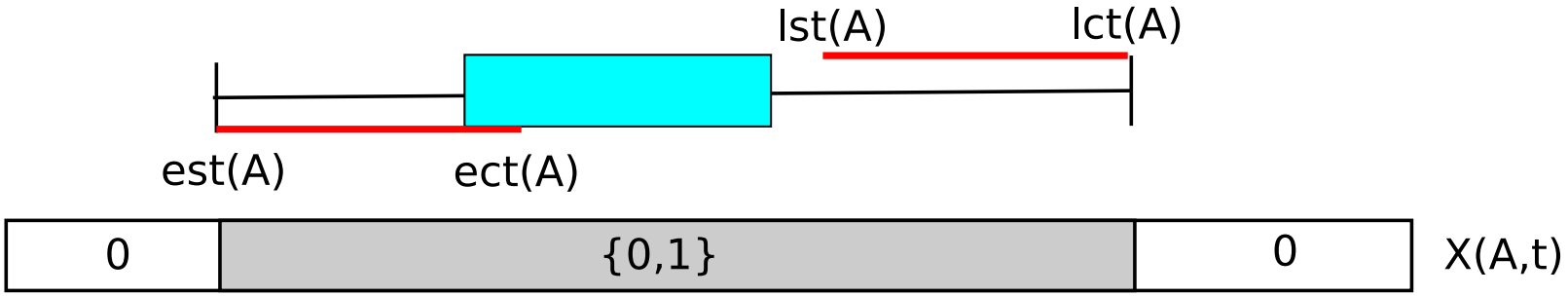
- Tabulka *(timetable)* pro aktivitu A je množina boolovských proměnných $X(A, t)$ udávajících, zda A běží v čase t

$$\forall t \sum_{A_i} X(A_i, t) cap(A_i) \leq MaxCapacity \qquad (*)$$

$$\forall t, i \; start(A_i) \leq t \leq end(A_i) \Leftrightarrow X(A_i, t)$$

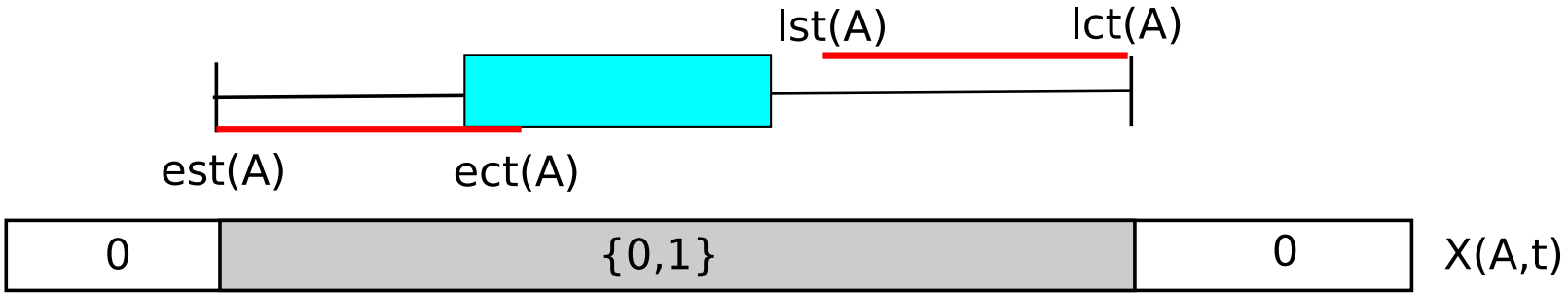
Podmínka tabulky: př. s odvozovacími pravidly

Počáteční stav

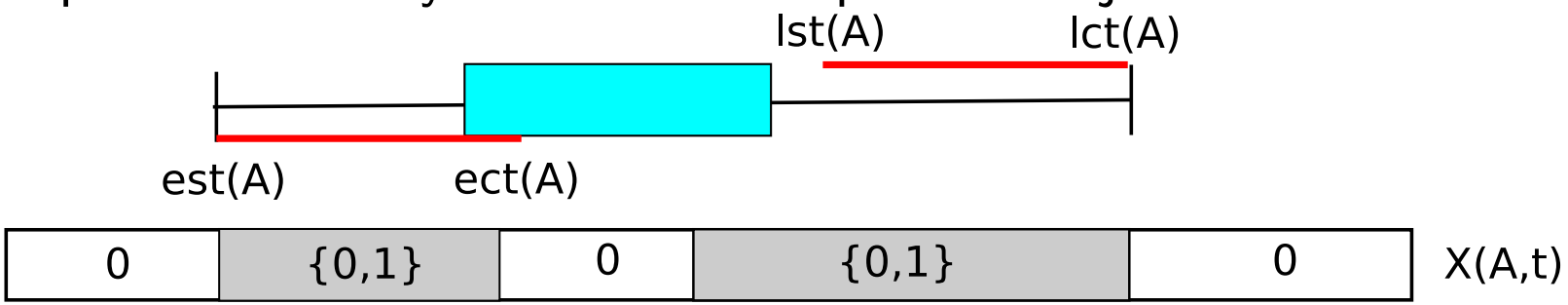


Podmínka tabulky: př. s odvozovacími pravidly

Počáteční stav

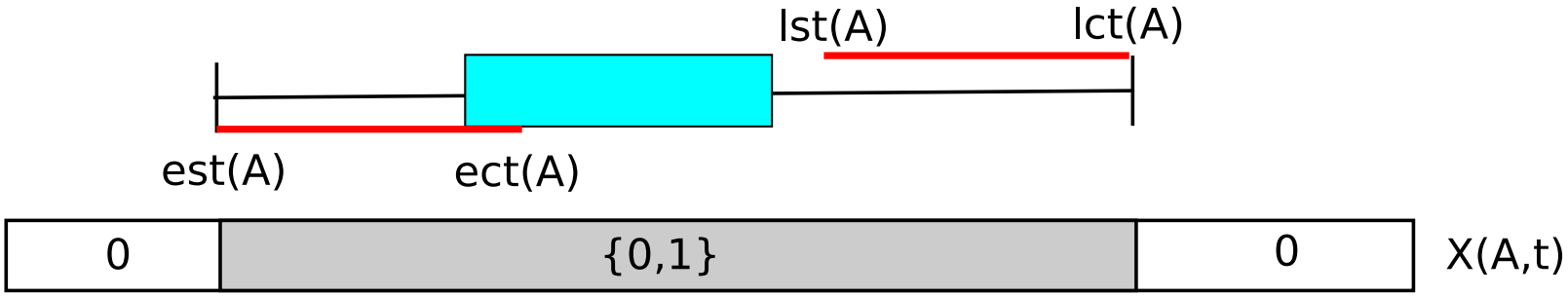


Některé pozice zakázány vzhledem ke kapacitě zdroje

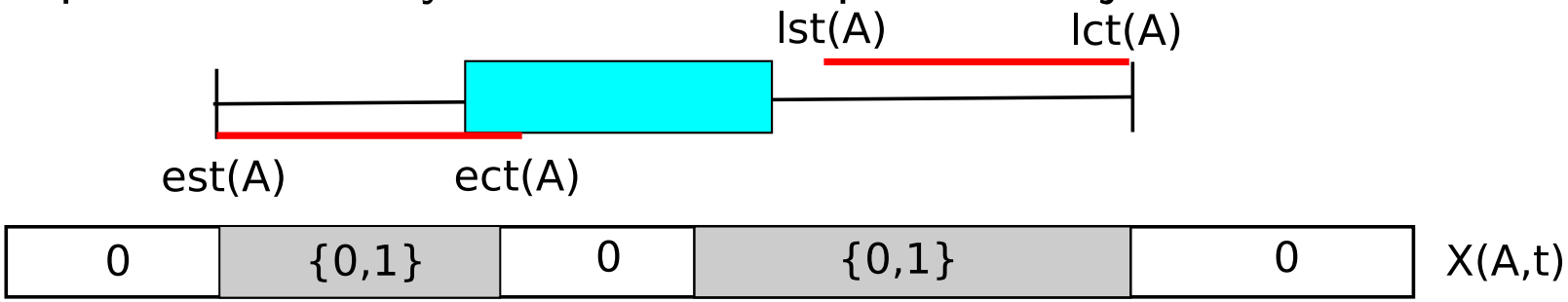


Podmínka tabulky: př. s odvozovacími pravidly

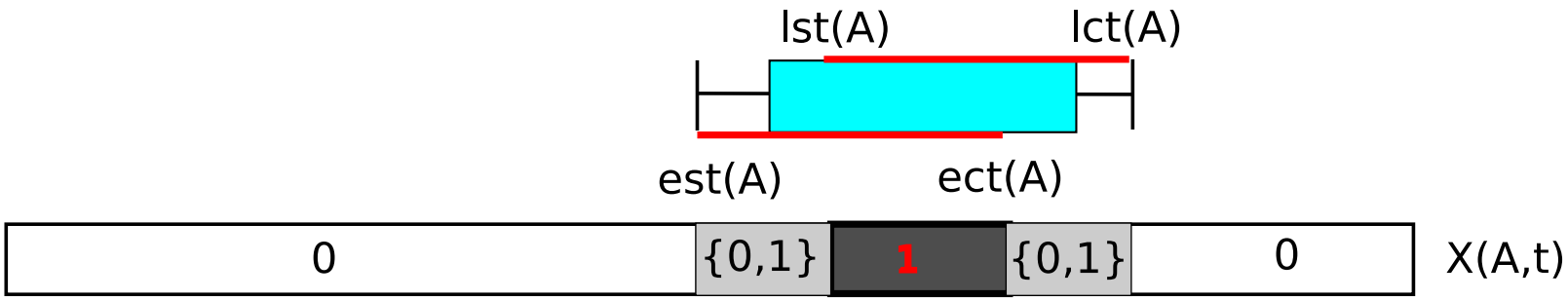
Počáteční stav



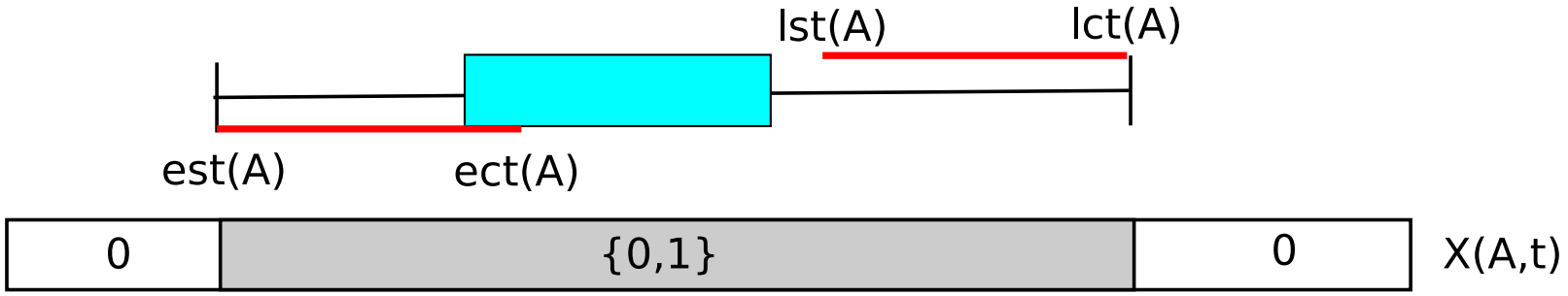
Některé pozice zakázány vzhledem ke kapacitě zdroje



Nový stav



Podmínka tabulky: odvozovací pravidla

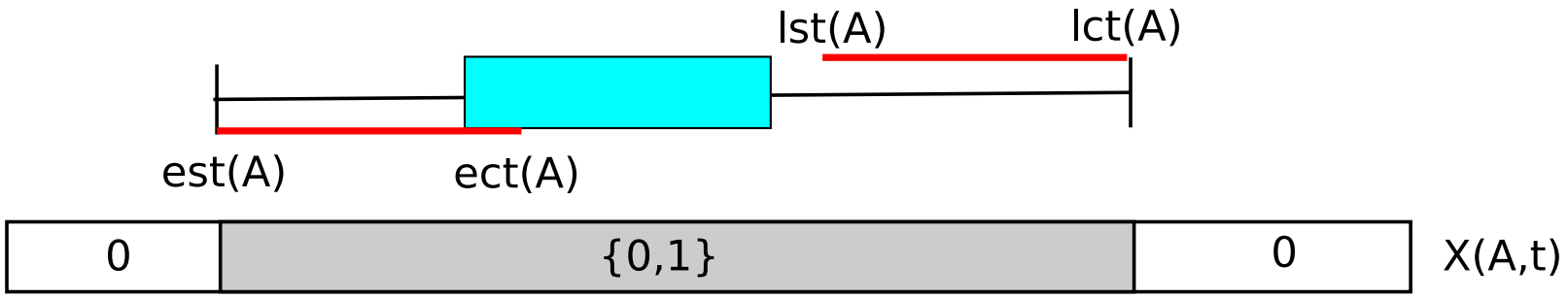


Jak realizovat filtraci přes omezení

$$\forall t, i \; start(A_i) \leq t < end(A_i) \Leftrightarrow X(A_i, t) \; ?$$

Problém: t je zároveň index a také proměnná

Podmínka tabulky: odvozovací pravidla



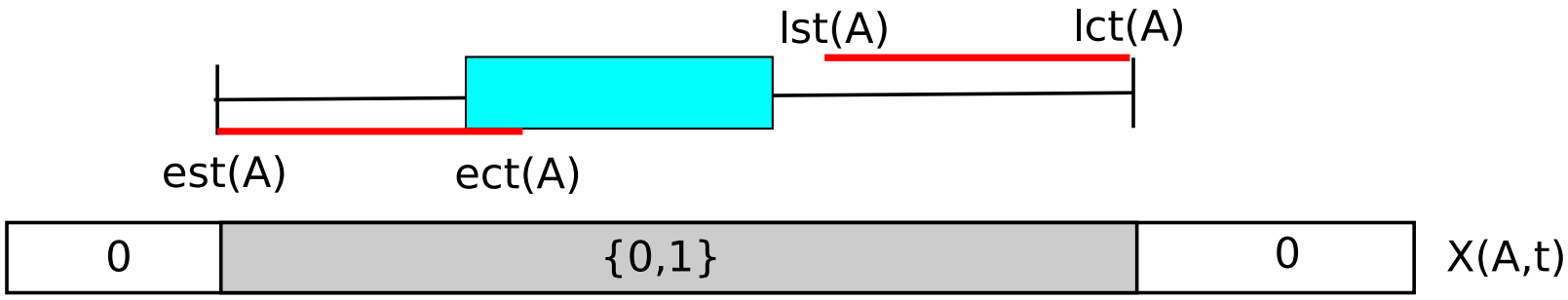
Jak realizovat filtraci přes omezení

$$\forall t, i \; start(A_i) \leq t < end(A_i) \Leftrightarrow X(A_i, t) \; ?$$

Problém: t je zároveň index a také proměnná

- start(A) $\geq$ min{t | 1 $\in$ X(A,t)}
- end(A) $\leq$ 1+max{t | 1 $\in$ X(A,t)}

Podmínka tabulky: odvozovací pravidla



Jak realizovat filtraci přes omezení

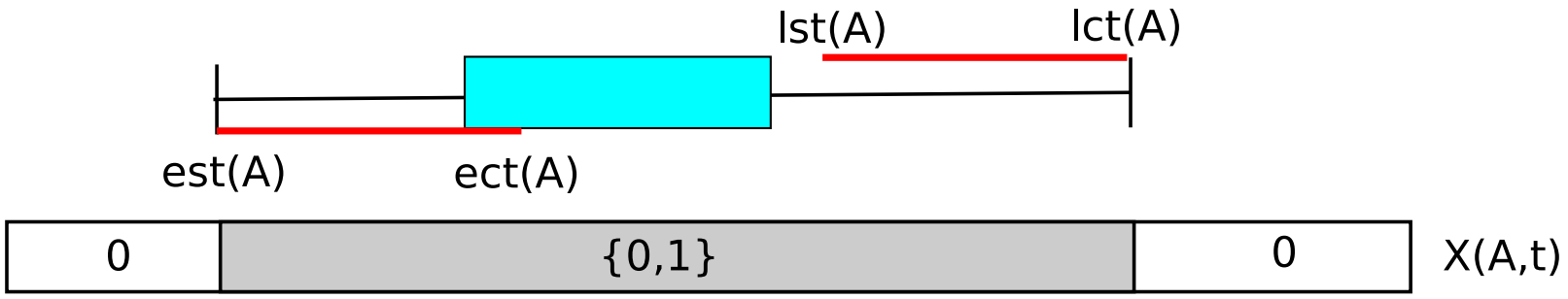
$$\forall t, i \; start(A_i) \leq t < end(A_i) \Leftrightarrow X(A_i, t) \; ?$$

Problém: t je zároveň index a také proměnná

- start(A) $\geq$ min{t | 1 $\in$ X(A,t)}
- end(A) $\leq$ 1+max{t | 1 $\in$ X(A,t)}

- X(A,t)=0 $\wedge$ t < ect(A) $\Rightarrow$ start(A)>t
- X(A,t)=0 $\wedge$ lst(A)$\leq$t $\Rightarrow$ end(A)$\leq$t

Podmínka tabulky: odvozovací pravidla



Jak realizovat filtraci přes omezení

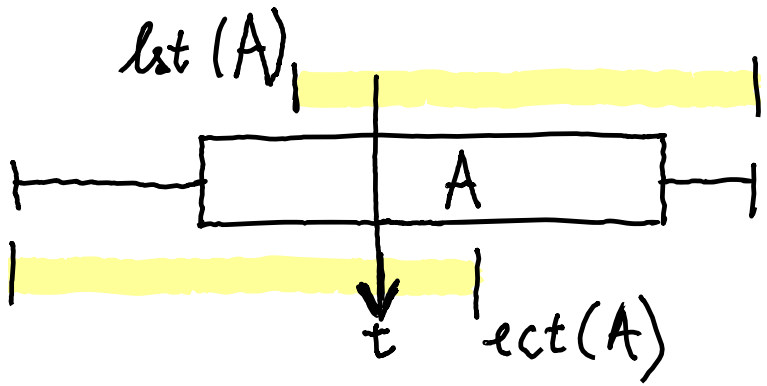
$$\forall t, i \; start(A_i) \leq t < end(A_i) \Leftrightarrow X(A_i, t) \; ?$$

Problém: t je zároveň index a také proměnná

- start(A) $\geq$ min{t | 1 $\in$ X(A,t)}
- end(A) $\leq$ 1+max{t | 1 $\in$ X(A,t)}

- X(A,t)=0 $\wedge$ t < ect(A) $\Rightarrow$ start(A)>t
- X(A,t)=0 $\wedge$ lst(A)$\leq$t $\Rightarrow$ end(A)$\leq$t

- lst(A)$\leq$t $\wedge$ t < ect(A) $\Rightarrow$ X(A,t)=1



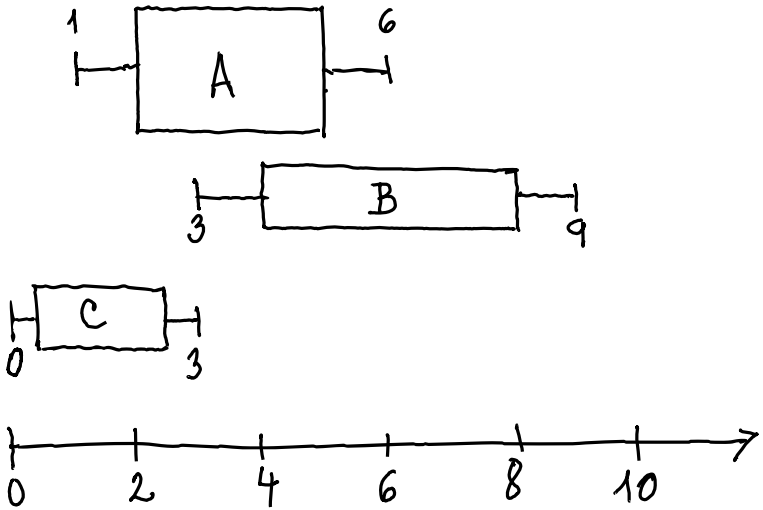
Cvičení: podmínka tabulky

Máme zadány zdroj s kapacitou 2 a aktivity

j	cap(j)	est(j)	lct(j)	p(j)
A	2	1	6	3
B	1	3	9	4
C	1	0	3	2

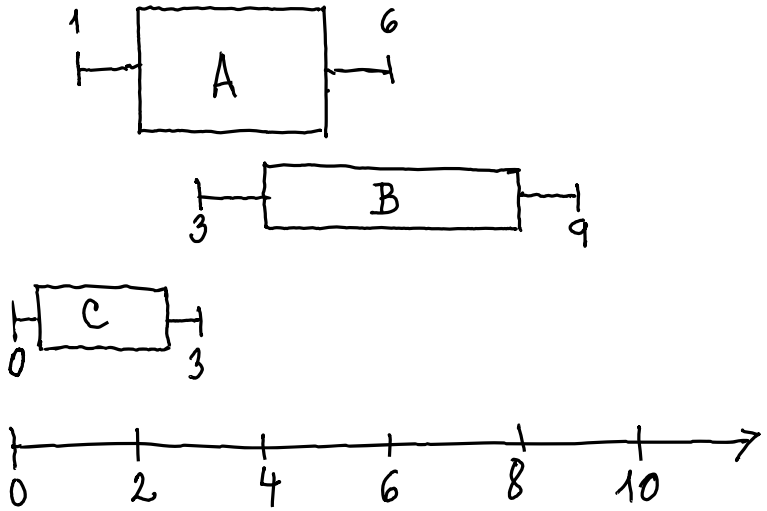
1. Jak jsou inicializovány proměnné X(j,t)?
2. Jak se jejich hodnoty mění při použití odvozovacích pravidel podmínky tabulky?
3. Jak by mohly vypadat výsledné rozvrhy po aplikaci pravidel?

Cvičení: podmínka tabulky



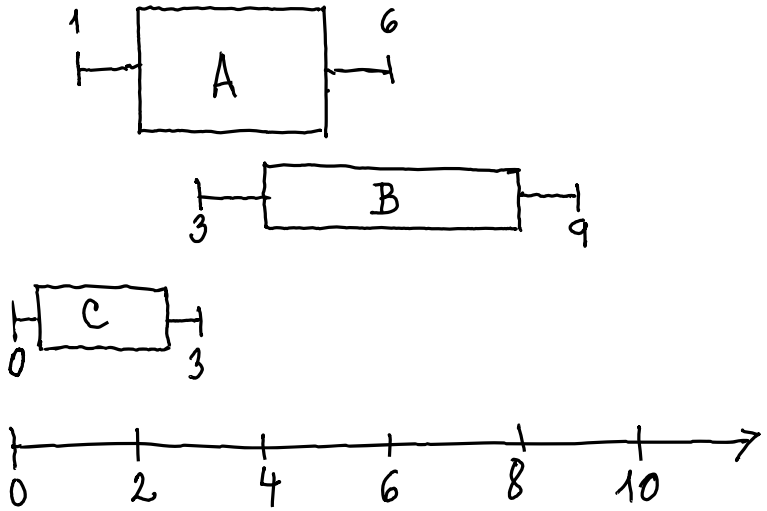
1. Jak jsou inicializovány proměnné X(j,t)?

Cvičení: podmínka tabulky



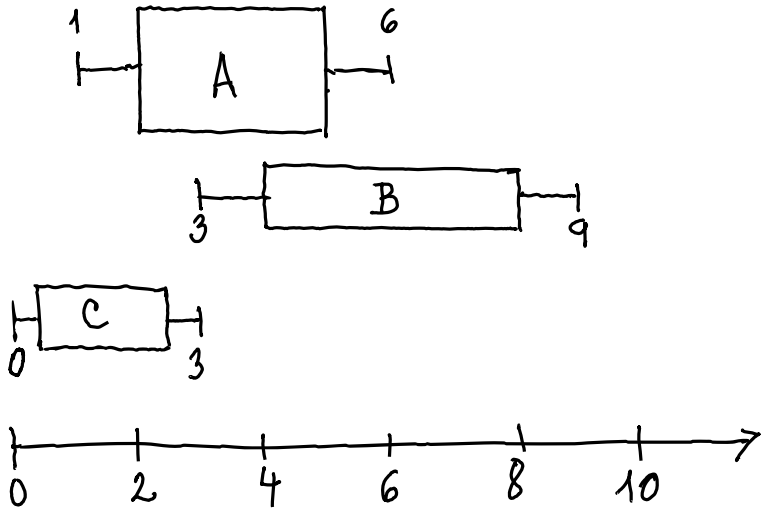
1. Jak jsou inicializovány proměnné X(j,t)?
   - $X(A,1)$ až $X(A,5)$ jsou $\{0,1\}$, $X(B,3)$ až $X(B,8)$ jsou $\{0,1\}$, $X(C,0)$ až $X(C,2)$ jsou $\{0,1\}$, ostatní proměnné nulové
2. Jak se jejich hodnoty mění při použití odvozovacích pravidel podmínky tabulky?

Cvičení: podmínka tabulky



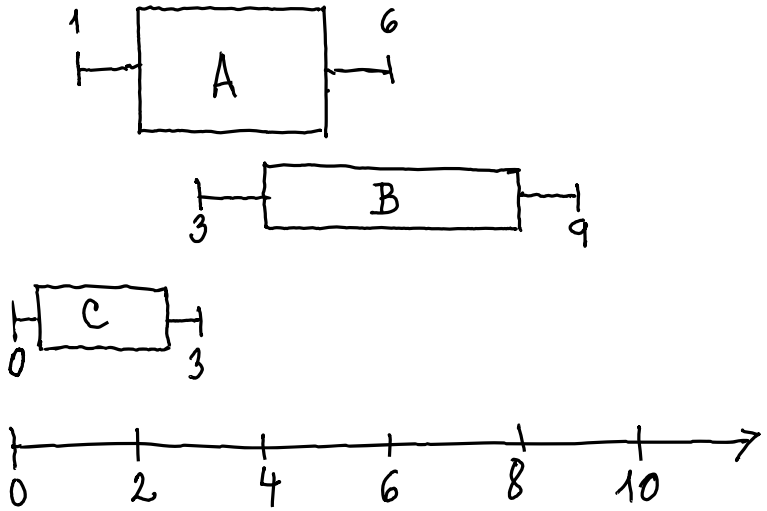
1. Jak jsou inicializovány proměnné X(j,t)?
   - $X(A,1)$ až $X(A,5)$ jsou $\{0,1\}$, $X(B,3)$ až $X(B,8)$ jsou $\{0,1\}$, $X(C,0)$ až $X(C,2)$ jsou $\{0,1\}$, ostatní proměnné nulové
2. Jak se jejich hodnoty mění při použití odvozovacích pravidel podmínky tabulky?
   1. dle $(*)$ B může začít nejdříve v čase 4 kvůli A, tj. $X(B,3) = 0$ a A musí být před B, tj. A nejpozději skončí v čase 5 a $X(A,5) = 0$

Cvičení: podmínka tabulky



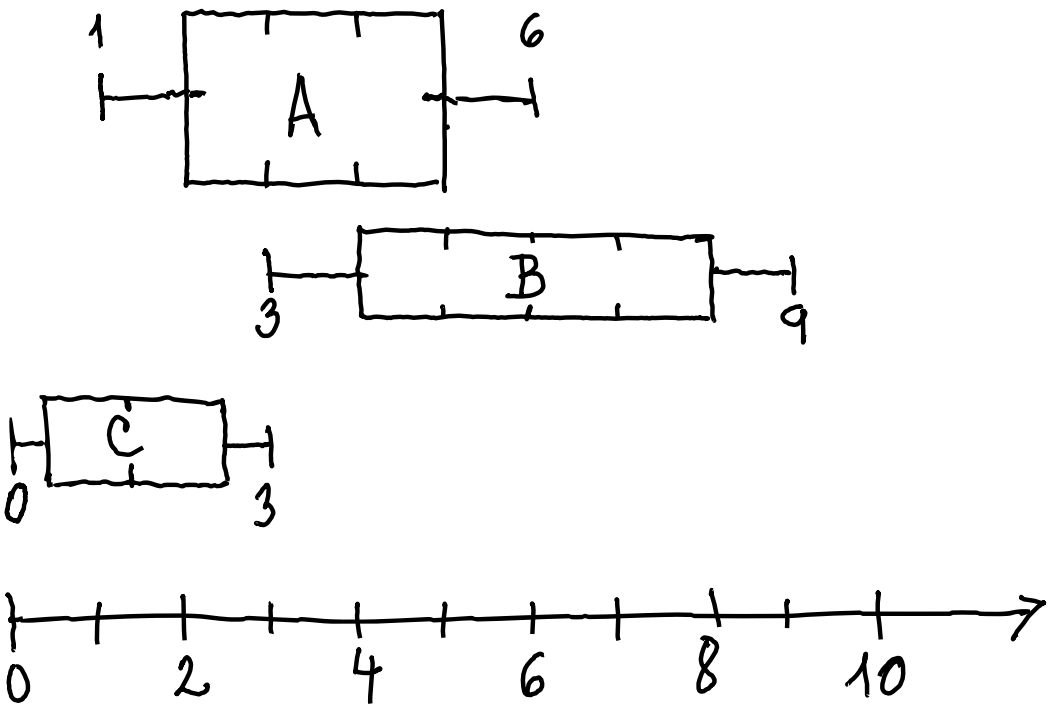
1. Jak jsou inicializovány proměnné X(j,t)?
   - $X(A,1)$ až $X(A,5)$ jsou $\{0,1\}$, $X(B,3)$ až $X(B,8)$ jsou $\{0,1\}$, $X(C,0)$ až $X(C,2)$ jsou $\{0,1\}$, ostatní proměnné nulové
2. Jak se jejich hodnoty mění při použití odvozovacích pravidel podmínky tabulky?
   1. dle $(*)$ B může začít nejdříve v čase 4 kvůli A, tj. $X(B,3) = 0$ a A musí být před B, tj. A nejpozději skončí v čase 5 a $X(A,5) = 0$
   2. dále z $(*)$ $X(C,2) = 0$, C začne v čase 0 $X(A,1) = 0$ a A začne v čase 2

Cvičení: podmínka tabulky

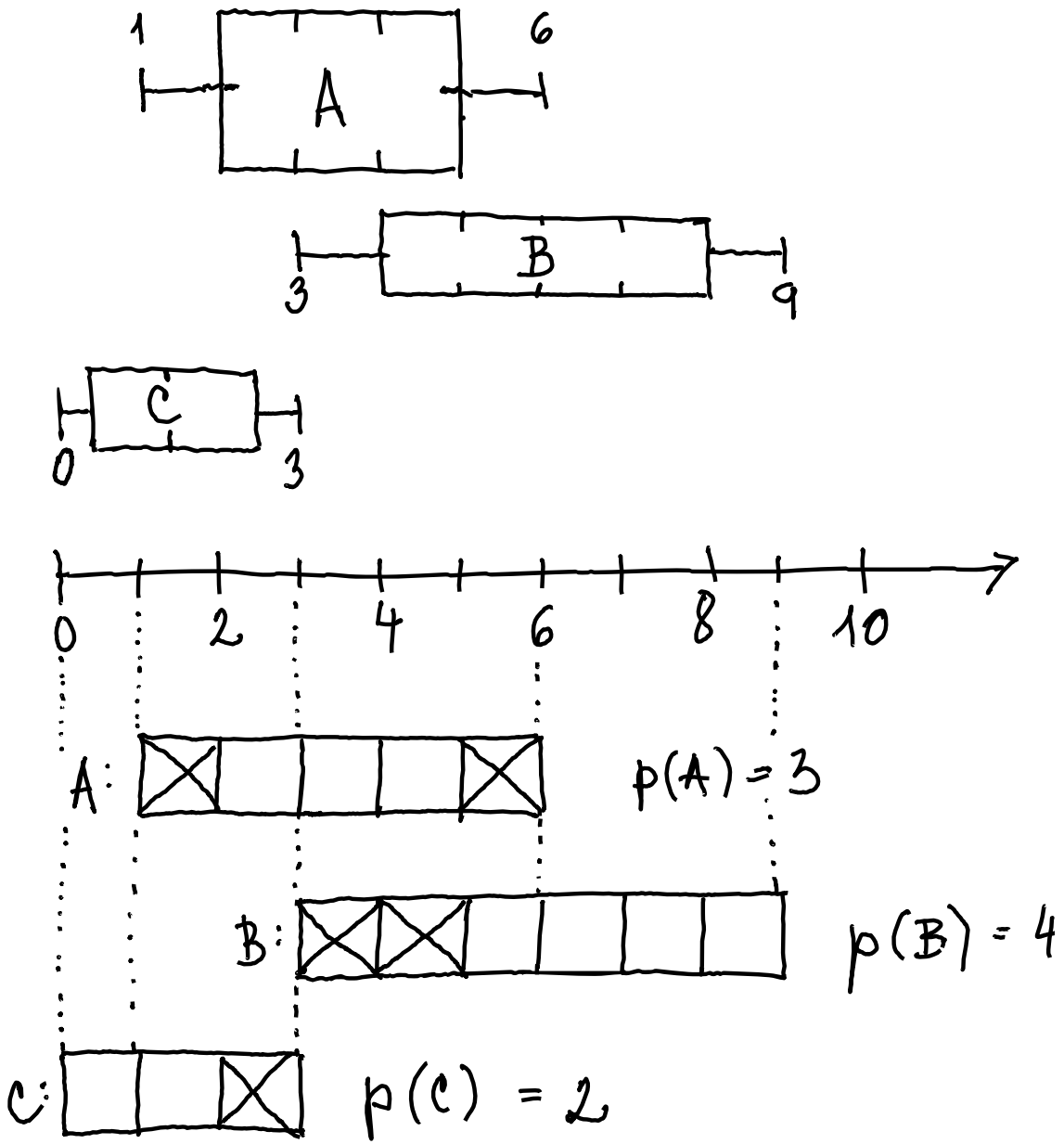


1. Jak jsou inicializovány proměnné X(j,t)?
   - $X(A,1)$ až $X(A,5)$ jsou $\{0,1\}$, $X(B,3)$ až $X(B,8)$ jsou $\{0,1\}$, $X(C,0)$ až $X(C,2)$ jsou $\{0,1\}$, ostatní proměnné nulové
2. Jak se jejich hodnoty mění při použití odvozovacích pravidel podmínky tabulky?
   1. dle $(*)$ B může začít nejdříve v čase 4 kvůli A, tj. $X(B,3)=0$ a A musí být před B, tj. A nejpozději skončí v čase 5 a $X(A,5)=0$
   2. dále z $(*)$ $X(C,2)=0$, C začne v čase 0
      $X(A,1)=0$ a A začne v čase 2 a také
      B musí začít až v čase 5 a $X(B,4)=0$
      a máme jediné řešení

Cvičení: podmínka tabulky



Cvičení: podmínka tabulky



Unární a kumulativní zdroje: shrnutí a možná rozšíření

Disjunktivní omezení

- známe: unarní zdroje, nepřerušitelné aktivity
- rozšíření: přerušitelné aktivity, kumulativní zdroje

Hledání hran

- známe: unarní zdroje, nepřerušitelné aktivity
- rozšíření: přerušitelné aktivity, kumulativní zdroje

Ne-první/ne-poslední

- známe: unarní zdroje, nepřerušitelné aktivity
- rozšíření: kumulativní zdroje

Podmínka tabulky

- známe: kumulativní zdroje, nepřerušitelné aktivity
- rozšíření: přerušitelné aktivity

Globální podmínky

Pro reprezentaci zdrojů využívány v programovacích jazycích
tzv. globální podmínky

- definované pro libovolný konečný počet proměnných
- komplexní podmínky s vlastním propagačním algoritmem

Základní globální podmínky (pro rozvrhování)

- příklady z IBM ILOG OPL (Optimization Programming Language)
- všechny proměnné různé
  - `allDifferent`
- disjunktivní zdroj
  - `dvar interval`, `dvar sequence`
  - `noOverlap`
- kumulativní zdroj
  - `cumuFunction`, `pulse`

Všechny proměnné různé

Proměnné v poli `Array` jsou různé

- reprezentace unárního zdroje s jednotkovou dobou trvání všech aktivit
- `dvar int Array[Interval];`
- globální podmínka: `allDifferent(Array)`

Příklad: učitelé musí učit v různé hodiny

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

Všechny proměnné různé

Proměnné v poli `Array` jsou různé

- reprezentace unárního zdroje s jednotkovou dobou trvání všech aktivit
- `dvar int Array[Interval];`
- globální podmínka: `allDifferent(Array)`

Příklad: učitelé musí učit v různé hodiny

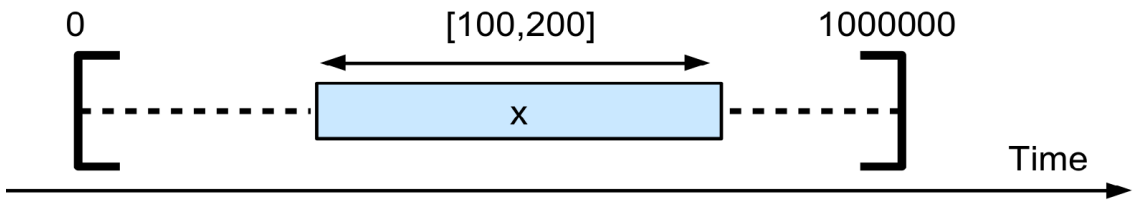
- `Jan = 6, Ota = 2, Anna = 5,`
  `Marie = 1, Petr` $\in \{3,4\}$`, Eva` $\in \{3,4\}$

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

Intervalové proměnné

Intervalová proměnná: pro modelování časového intervalu (úlohy, aktivity)

- hodnotou intervalové proměnné je celočíselný interval $[start, end)$
- příklad: `dvar interval x in 0..1000000 size 100..200;`



Disjunktní rozvrhování

Sekvenční proměnná `p`

- definována na množině intervalových proměnných `x`

```
dvar interval x[i in 1..n] ...;
dvar sequence p in x;
```

- hodnota intervalové proměnné `p` je permutace přítomných intervalů
  - pozor, permutace `t` ještě neimplikuje žádné uspořádání v čase

Disjunktní rozvrhování

Sekvenční proměnná `p`

- definována na množině intervalových proměnných `x`

```
dvar interval x[i in 1..n] ...;
dvar sequence p in x;
```

- hodnota intervalové proměnné `p` je permutace přítomných intervalů
  - pozor, permutace `t` ještě neimplikuje žádné uspořádání v čase

Omezení `noOverlap(p)`

- vyjadřuje, že sekvenční proměnná `p` reprezentuje řetězec nepřekrývajících se intervalových proměnných
- pro vyjádření rozvrhování na unárním/disjunktivním zdroji, kde se intervaly/úlohy nepřekrývají

Precedence

Mezi intervalovými proměnnými můžeme definovat precedenční podmínky:

```
dvar interval i;
dvar interval j;

endBeforeStart(i, j);
startBeforeStart(i, j);
startAtStart(i, j);
...
```

Precedence, účelová funkce

Mezi intervalovými proměnnými můžeme definovat precedenční podmínky:

```
dvar interval i;
dvar interval j;


endBeforeStart(i, j);
startBeforeStart(i, j);
startAtStart(i, j);
...
```

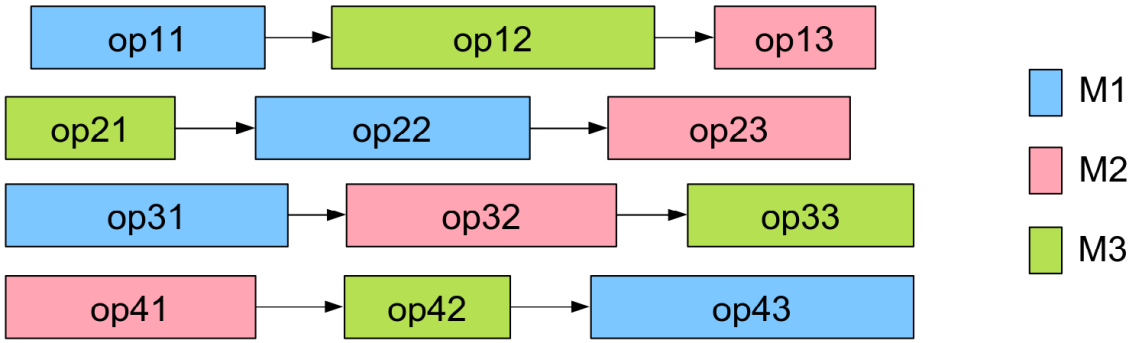
Pro vytváření účelových funkcí nebo definici omezení

```
startOf(x)
endOf(x)
sizeOf(x, V)
```

Příklad: minimalizace makespanu

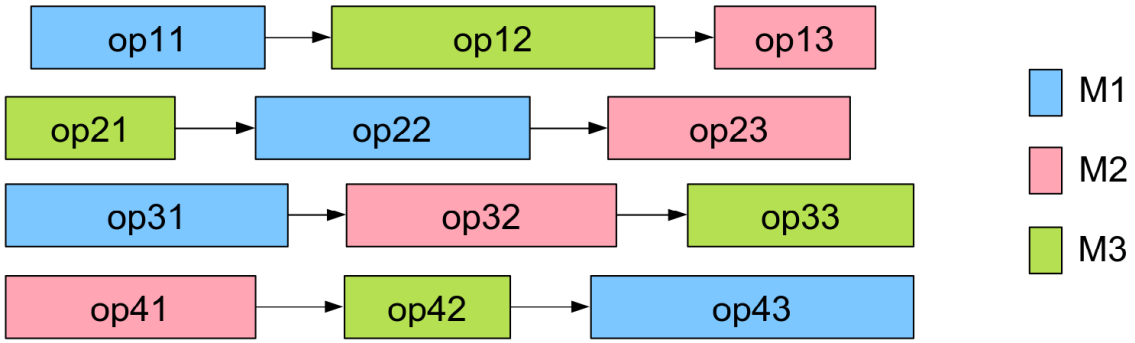
```
minimize max(i in 1..n) endOf(x[i])
```

Příklad: rozvrhování problému job-shop




```
tuple Operation {
  int mch; // machine
  int pt;  // processing time
};
Operation Ops[j in Jobs][p in Pos] = ...;
```

Příklad: rozvrhování problému job-shop




```
tuple Operation {
  int mch; // machine
  int pt;  // processing time
};
Operation Ops[j in Jobs][p in Pos] = ...;
```

```
1  dvar interval op[j in Jobs][p in Pos] size Ops[j][p].pt;
2  dvar sequence mchs[m in Mchs] in
3    all(j in Jobs, p in Pos: Ops[j][p].mch == m) op[j][p];
4
5  minimize max(j in Jobs) endOf(op[j][nbPos]);
6  subject to {
7    forall(m in Mchs)
8      noOverlap(mchs[m]);
9    forall(j in Jobs, p in 2..nbPos)
10     endBeforeStart(op[j][p-1],op[j][p]);
11 }
```

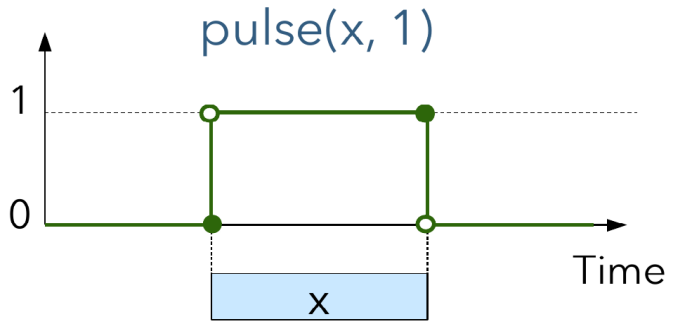
`op[j][p]` odkazuje operaci úlohy `j`, která je zpracovávána v rámci úlohy jako p-tá

Kumulativní zdroj pomocí kumulativní funkce

Hodnota výrazu kumulativní funkce reprezentuje vývoj kvantity v čase, která může být inkrementálně změněna (snížena nebo navýšena) intervalovými proměnnými.

Intervalové proměnné `x[i]` přispívají do kumul. funkce po dobu svého provádění

```
int capacity[1..5] = [1,3,2,4,1];
cumulFunction y = sum(i in 1..5) pulse(x[i],capacity[i]);
```



Omezení na výrazech kumulativní funkce: pro omezení kapacity zdroje

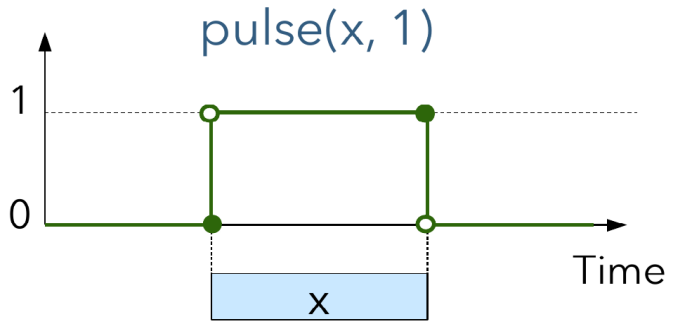
```
int h = ...
cumulFunction f= ...
f<=h
```

Kumulativní zdroj pomocí kumulativní funkce

Hodnota výrazu kumulativní funkce reprezentuje vývoj kvantity v čase, která může být inkrementálně změněna (snížena nebo navýšena) intervalovými proměnnými.

Intervalové proměnné `x[i]` přispívají do kumul. funkce po dobu svého provádění

```
int capacity[1..5] = [1,3,2,4,1];
cumulFunction y = sum(i in 1..5) pulse(x[i],capacity[i]);
```



Omezení na výrazech kumulativní funkce: pro omezení kapacity zdroje

```
int h = ...
cumulFunction f= ...
f<=h
```

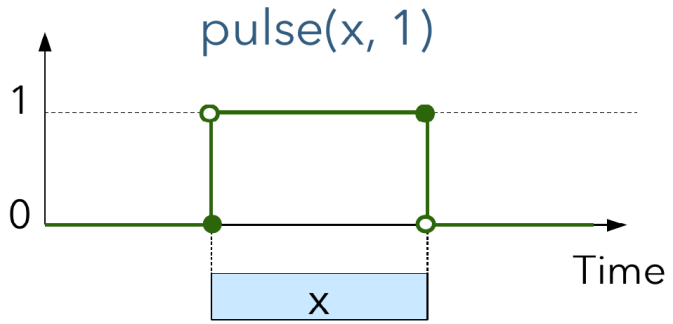
Příklad: job-shop a omezení celkového počtu strojů

Kumulativní zdroj pomocí kumulativní funkce

Hodnota výrazu kumulativní funkce reprezentuje vývoj kvantity v čase, která může být inkrementálně změněna (snížena nebo navýšena) intervalovými proměnnými.

Intervalové proměnné `x[i]` přispívají do kumul. funkce po dobu svého provádění

```
int capacity[1..5] = [1,3,2,4,1];
cumulFunction y = sum(i in 1..5) pulse(x[i],capacity[i]);
```



Omezení na výrazech kumulativní funkce: pro omezení kapacity zdroje

```
int h = ...
cumulFunction f= ...
f<=h
```

Příklad: job-shop a omezení celkového počtu strojů

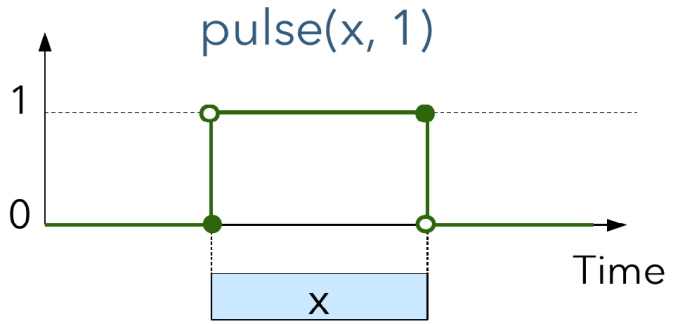
```
cumulFunction allMachines = sum(j in Jobs, p in Pos) pulse(op[j][p],1);
```

Kumulativní zdroj pomocí kumulativní funkce

Hodnota výrazu kumulativní funkce reprezentuje vývoj kvantity v čase, která může být inkrementálně změněna (snížena nebo navýšena) intervalovými proměnnými.

Intervalové proměnné `x[i]` přispívají do kumul. funkce po dobu svého provádění

```
int capacity[1..5] = [1,3,2,4,1];
cumulFunction y = sum(i in 1..5) pulse(x[i],capacity[i]);
```



Omezení na výrazech kumulativní funkce: pro omezení kapacity zdroje

```
int h = ...
cumulFunction f= ...
f<=h
```

Příklad: job-shop a omezení celkového počtu strojů

```
cumulFunction allMachines = sum(j in Jobs, p in Pos) pulse(op[j][p],1);
allMachines <= m;
```

Prohledávání/přiřazování (opakování)

Konzistenční techniky jsou (obvykle) neúplné

$\Rightarrow$ potřeba prohledávací algoritmus, který vyřeší "zbytek"

Přiřazovaní *(labeling)*

- prohledávání do hloubky (DFS/BT)
  - přiřaď hodnotu do proměnné
  - propaguj = uděl̆ej
    problém lokálně konzistentní
  - vrať se v případě neúspěchu

Způsoby větvení

Jaká proměnná má být ohodnocena první?

- princip prvotního neúspěchu *(first-fail)*
  - preferuj proměnnou, jejíž přiřazení je nejobtížnější
  - např. proměnné s nejmenší doménou:
    doména se snadněji vyprázdní
  - nebo proměnné s nejvíce podmínkami: pro proměnné s více podmínkami je obecně obtížnější nalézt hodnotu proměnné
- definuje tvar prohledávacího stromu
  - výběr proměnné s malou velikostí domény: malé větvení na této úrovni
  - výběr proměnné s velkou velikostí domény: velké větvení na této úrovni

Způsoby větvení

Jaká proměnná má být ohodnocena první?

- princip prvotního neúspěchu *(first-fail)*
  - preferuj proměnnou, jejíž přiřazení je nejobtížnější
  - např. proměnné s nejmenší doménou:
    doména se snadněji vyprázdní
  - nebo proměnné s nejvíce podmínkami: pro proměnné s více podmínkami
    je obecně obtížnější nalézt hodnotu proměnné
- definuje tvar prohledávacího stromu
  - výběr proměnné s malou velikostí domény: malé větvení na této úrovni
  - výběr proměnné s velkou velikostí domény: velké větvení na této úrovni

Jaká hodnota má být vyzkoušena první?

- princip prvotního úspěchu *(succeed-first)*
  - preferuj hodnoty, které nejspíše patří do řešení
  - např. hodnoty s nejvíce podporami v okolních proměnných
  - tato heuristika je obvykle problémově závislá
- definuje pořadí procházení větví

Schémata větvení

Větvení = řešení disjunkcí

Tradiční rozvrhovací přístupy

- kritická rozhodnutí se dělají první
  - vyřeš kritická místa *(bottlenecks)*, ...
  - definuje tvar prohledávacího stromu
  - podobně jako princip prvního neúspěchu *(first-fail)*

Schémata větvení

Větvení = řešení disjunkcí

Tradiční rozvrhovací přístupy

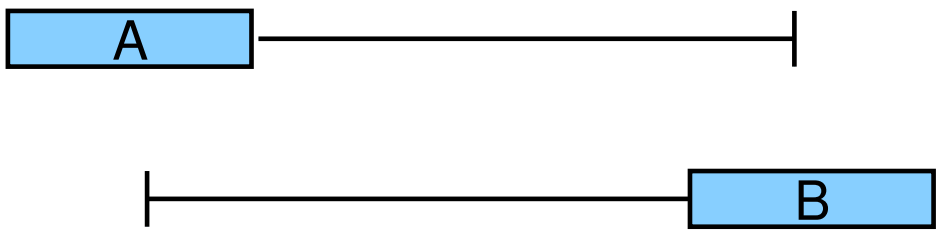
- kritická rozhodnutí se dělají první
  - vyřeš kritická místa *(bottlenecks)*, ...
  - definuje tvar prohledávacího stromu
  - podobně jako princip prvního neúspěchu *(first-fail)*
- preferuj alternativy s větší flexibilitou
  - definuje pořadí větví pro prozkoumání
  - podobně jako princip prvního úspěchu *(succeed-first)*

Jak popsat, co je kritické a co je flexibilní?

Rezerva *(slack)* Smith&Cheng 1993

- Rezerva *(slack)* je formální popis flexibility
- Rezerva pro dané pořadí dvou aktivit

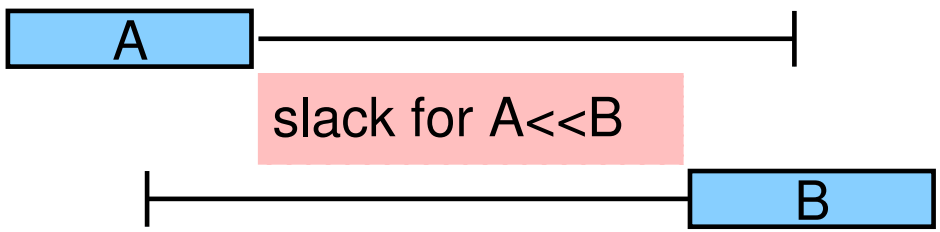
  „volný čas pro posunování aktivit"



Rezerva *(slack)* Smith&Cheng 1993

- Rezerva *(slack)* je formální popis flexibility
- Rezerva pro dané pořadí dvou aktivit

  „volný čas pro posunování aktivit"

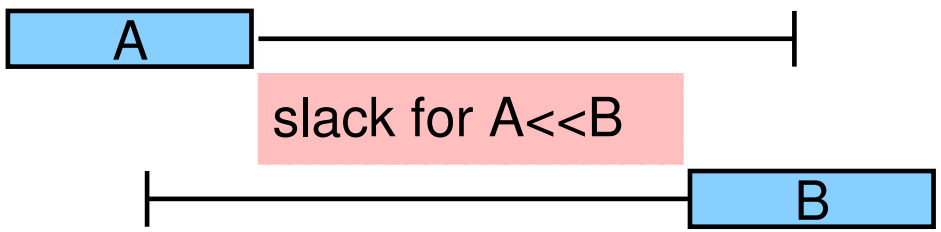


$$slack(A << B) = \max(end(B)) - min(start(A)) - p(A) - p(B))$$

Rezerva *(slack)* Smith&Cheng 1993

- Rezerva *(slack)* je formální popis flexibility
- Rezerva pro dané pořadí dvou aktivit

  „volný čas pro posunování aktivit"



$slack(A << B) = \max(end(B)) - min(start(A)) - p(A) - p(B))$

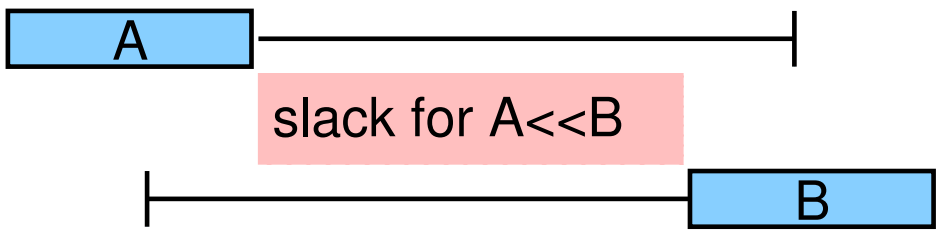
- Rezerva pro dvě aktivity (bez určení pořadí)

  $slack(\{A, B\}) = \max(slack(A << B), slack(B << A))$

Rezerva *(slack)* Smith&Cheng 1993

- Rezerva *(slack)* je formální popis flexibility
- Rezerva pro dané pořadí dvou aktivit

  „volný čas pro posunování aktivit"



$slack(A << B) = \max(end(B)) - min(start(A)) - p(A) - p(B))$

- Rezerva pro dvě aktivity (bez určení pořadí)

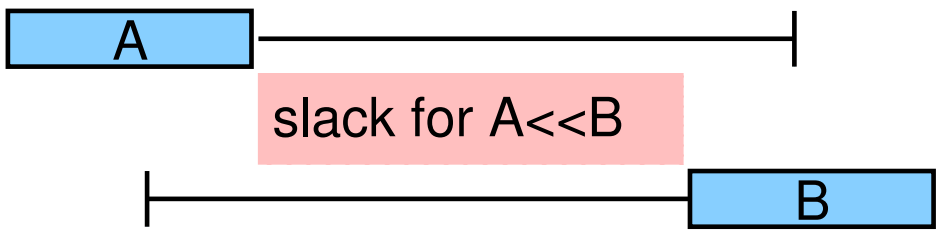
  $slack(\{A, B\}) = \max(slack(A << B), slack(B << A))$

- Rezerva pro skupinu aktivit

Rezerva *(slack)* Smith&Cheng 1993

- Rezerva *(slack)* je formální popis flexibility
- Rezerva pro dané pořadí dvou aktivit

  „volný čas pro posunování aktivit"



$slack(A << B) = \max(end(B)) - min(start(A)) - p(A) - p(B))$

- Rezerva pro dvě aktivity (bez určení pořadí)

  $slack(\{A, B\}) = \max(slack(A << B), slack(B << A))$

- Rezerva pro skupinu aktivit

  $slack(\Omega) = \max(end(\Omega)) - min(start(\Omega)) - p(\Omega)$

Větvení uspořadáním dvojic aktivit

$A << B \quad \vee \quad \neg A << B$

Jaké aktivity mají být uspořádány první?

Větvení uspořadáním dvojic aktivit

$$A << B \quad \vee \quad \neg A << B$$

Jaké aktivity mají být uspořádány první?

- nejkritičtější pár (first-fail)
- pár s minimální rezervou $slack(\{A, B\})$

Větvení uspořadáním dvojic aktivit

$A << B \quad \vee \quad \neg A << B$

Jaké aktivity mají být uspořádány první?

- nejkritičtější pár (first-fail)
- pár s minimální rezervou $slack(\{A, B\})$

Jaké pořadí aktivit má být zvoleno?

Větvení uspořadáním dvojic aktivit

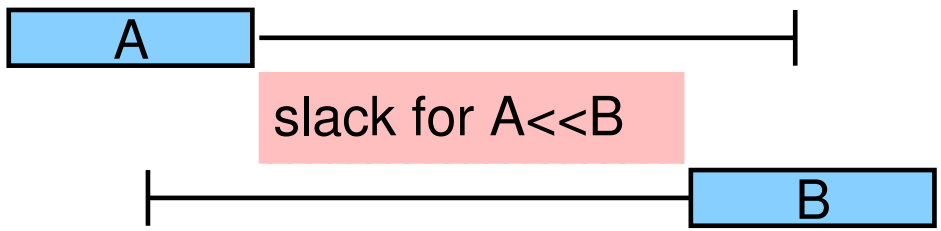
$A << B \quad \vee \quad \neg A << B$

Jaké aktivity mají být uspořádány první?

- nejkritičtější pár (first-fail)
- pár s minimální rezervou $slack(\{A, B\})$

Jaké pořadí aktivit má být zvoleno?

- nejflexibilnější pořadí (succeed-first)
- pořadí maximalizující $slack(A??B)$
- Příklad: vybereme pořadí $A << B$



Bodů volby při n aktivitách:

Větvení uspořadáním dvojic aktivit

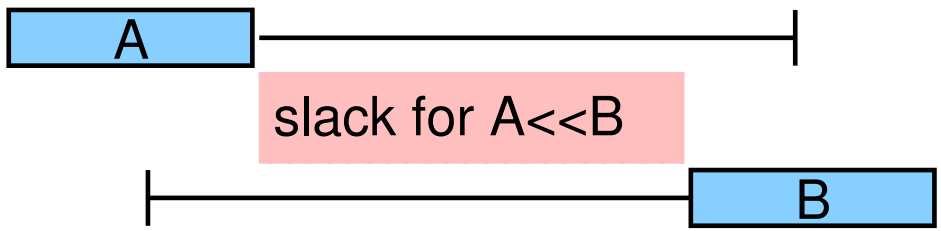
$A << B \quad \vee \quad \neg A << B$

Jaké aktivity mají být uspořádány první?

- nejkritičtější pár (first-fail)
- pár s minimální rezervou $slack(\{A, B\})$

Jaké pořadí aktivit má být zvoleno?

- nejflexibilnější pořadí (succeed-first)
- pořadí maximalizující $slack(A??B)$
- Příklad: vybereme pořadí $A << B$



Bodů volby při n aktivitách: $O(n^2)$

Větvení první/poslední

Baptiste at al. 1995

$$(A << \Omega \vee \neg A << \Omega) \quad \vee \quad (\Omega << A \vee \neg \Omega << A)$$

Máme hledat první nebo poslední aktivitu?

- díváme se na množinu možných kandidátů na první aktivitu a na množinu možných kandidátu na poslední aktivitu
- výbereme menší z těchto dvou množin (first-fail)
  - menší počet kandidátů znamená, že

Větvení první/poslední

Baptiste at al. 1995

$$(A << \Omega \vee \neg A << \Omega) \quad \vee \quad (\Omega << A \vee \neg \Omega << A)$$

Máme hledat první nebo poslední aktivitu?

- díváme se na množinu možných kandidátů na první aktivitu a na množinu možných kandidátu na poslední aktivitu
- výbereme menší z těchto dvou množin (first-fail)
  - menší počet kandidátů znamená, že je těžší najít vhodného kandidáta

Větvení první/poslední — Baptiste at al. 1995

$$(A << \Omega \vee \neg A << \Omega) \quad \vee \quad (\Omega << A \vee \neg \Omega << A)$$

Máme hledat první nebo poslední aktivitu?

- díváme se na množinu možných kandidátů na první aktivitu a na množinu možných kandidátu na poslední aktivitu
- výbereme menší z těchto dvou množin (first-fail)
  - menší počet kandidátů znamená, že je těžší najít vhodného kandidáta

Jaká aktivita má být vybrána?

- pokud se hledá první aktivita, potom preferuj aktivitu, která

Větvení první/poslední Baptiste at al. 1995

$$(A << \Omega \vee \neg A << \Omega) \quad \vee \quad (\Omega << A \vee \neg \Omega << A)$$

Máme hledat první nebo poslední aktivitu?

- díváme se na množinu možných kandidátů na první aktivitu a na množinu možných kandidátu na poslední aktivitu
- výbereme menší z těchto dvou množin (first-fail)
  - menší počet kandidátů znamená, že je těžší najít vhodného kandidáta

Jaká aktivita má být vybrána?

- pokud se hledá první aktivita,
  potom preferuj aktivitu, která má nejmenší $min(start(A))$

Větvení první/poslední

Baptiste at al. 1995

$$(A << \Omega \vee \neg A << \Omega) \quad \vee \quad (\Omega << A \vee \neg \Omega << A)$$

Máme hledat první nebo poslední aktivitu?

- díváme se na množinu možných kandidátů na první aktivitu a na množinu možných kandidátu na poslední aktivitu
- výbereme menší z těchto dvou množin (first-fail)
  - menší počet kandidátů znamená, že je těžší najít vhodného kandidáta

Jaká aktivita má být vybrána?

- pokud se hledá první aktivita,
  potom preferuj aktivitu, která má nejmenší $min(start(A))$
- pokud se hledá poslední aktivita,
  potom preferuj aktivitu, která

Větvení první/poslední — Baptiste at al. 1995

$$(A << \Omega \vee \neg A << \Omega) \quad \vee \quad (\Omega << A \vee \neg \Omega << A)$$

Máme hledat první nebo poslední aktivitu?

- díváme se na množinu možných kandidátů na první aktivitu a na množinu možných kandidátu na poslední aktivitu
- výbereme menší z těchto dvou množin (first-fail)
  - menší počet kandidátů znamená, že je těžší najít vhodného kandidáta

Jaká aktivita má být vybrána?

- pokud se hledá první aktivita,
  potom preferuj aktivitu, která má nejmenší $min(start(A))$
- pokud se hledá poslední aktivita,
  potom preferuj aktivitu, která má největší $max(end(A))$

Bodů volby:

Větvení první/poslední — Baptiste at al. 1995

$$(A << \Omega \vee \neg A << \Omega) \quad \vee \quad (\Omega << A \vee \neg \Omega << A)$$

Máme hledat první nebo poslední aktivitu?

- díváme se na množinu možných kandidátů na první aktivitu a na množinu možných kandidátu na poslední aktivitu
- výbereme menší z těchto dvou množin (first-fail)
  - menší počet kandidátů znamená, že je těžší najít vhodného kandidáta

Jaká aktivita má být vybrána?

- pokud se hledá první aktivita, potom preferuj aktivitu, která má nejmenší $min(start(A))$
- pokud se hledá poslední aktivita, potom preferuj aktivitu, která má největší $max(end(A))$

Bodů volby: $O(n)$

Zdrojová rezerva

Zdrojová rezerva je definovaná jako

rezerva množiny aktivit zpracovávaných daným zdrojem

Jak používat zdrojovou rezervu?

- pokud volíme zdroj, na kterém budou aktivity uspořádány jako první
  - vyber zdroj s

Zdrojová rezerva

Zdrojová rezerva je definovaná jako

rezerva množiny aktivit zpracovávaných daným zdrojem

Jak používat zdrojovou rezervu?

- pokud volíme zdroj, na kterém budou aktivity uspořádány jako první
  - vyber zdroj s minimální rezervou (kritické místo)

Zdrojová rezerva

Zdrojová rezerva je definovaná jako

rezerva množiny aktivit zpracovávaných daným zdrojem

Jak používat zdrojovou rezervu?

- pokud volíme zdroj, na kterém budou aktivity uspořádány jako první
  - vyber zdroj s minimální rezervou (kritické místo)
- pokud volíme zdroj, na který alokovat danou aktivitu
  - vyber zdroj s

Zdrojová rezerva

Zdrojová rezerva je definovaná jako

rezerva množiny aktivit zpracovávaných daným zdrojem

Jak používat zdrojovou rezervu?

- pokud volíme zdroj, na kterém budou aktivity uspořádány jako první
  - vyber zdroj s minimální rezervou (kritické místo)
- pokud volíme zdroj, na který alokovat danou aktivitu
  - vyber zdroj s maximální rezervou (flexibilita)

Omezující podmínky: shrnutí

Problém splňování podmínek

- popis problému pomocí doménových proměnných a omezení
- konzistence a propagace
- prohledávání

Rozvrhování jako problém splňování podmínek

- doménové proměnné pro čas a zdroje
- propagační algoritmy pro
  - unární zdroje
  - kumulativní zdroje
  - produkovatelné a spotřebovatelné zdroje
  - alternativní zdroje
- globálních podmínky pro zdroje
- prohledávání a rozvrhovací strategie
  - pojem rezervy
  - přístupy k větvení

Plánování projektu

16. května 2022

Problémy plánování projektu

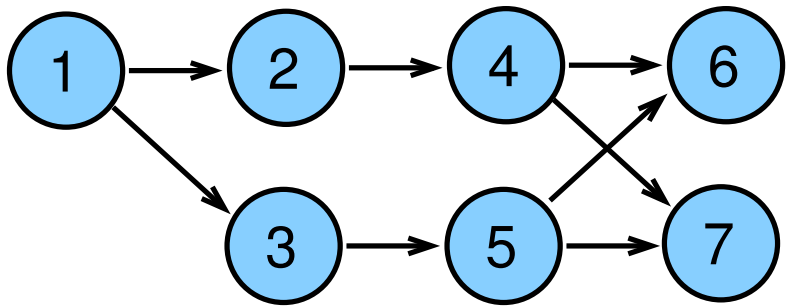
- Zprostředkování, instalace a testování rozsáhlého počítačového systému
  - projekt zahrnuje
    - evaluace a výběr hardware, vývoj software, nábor a školení lidí, testování a ladění systému, ...
  - precedenční vztahy
    - některé úlohy mohou být prováděny paralelně
    - úloha musí být realizována až po dokončení jiných úloh
  - cíl: minimalizovat čas na realizaci celého projektu

Problémy plánování projektu

- Zprostředkování, instalace a testování rozsáhlého počítačového systému
  - projekt zahrnuje
    - evaluace a výběr hardware, vývoj software, nábor a školení lidí, testování a ladění systému, ...
  - precedenční vztahy
    - některé úlohy mohou být prováděny paralelně
    - úloha musí být realizována až po dokončení jiných úloh
  - cíl: minimalizovat čas na realizaci celého projektu
- Příklady dalších problémů
  - stavba nemovitostí, konstrukce center elektráren, vojenský průmysl

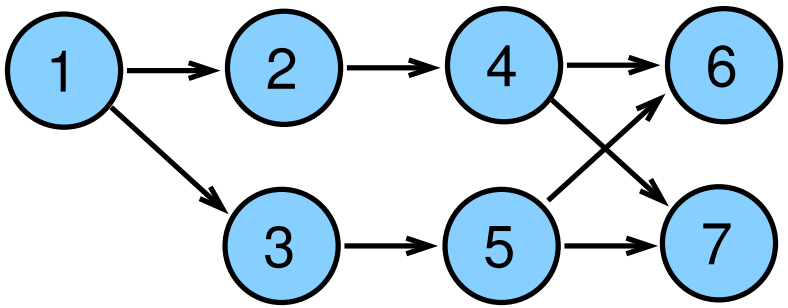
Plánování projektu

- Základní problém plánování projektu
  - precedenční podmínky
  - paralelní stroj s neomezeným počtem strojů
  - minimalizace maximálního času konce úloh *(makespan)*
  - relativně jednoduché na řešení



Plánování projektu

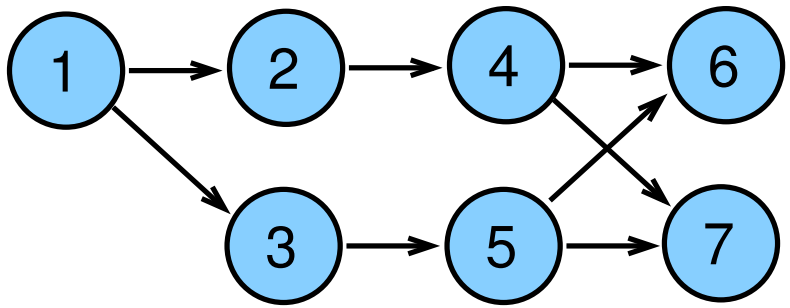
- Základní problém plánování projektu
  - precedenční podmínky
  - paralelní stroj s neomezeným počtem strojů
  - minimalizace maximálního času konce úloh *(makespan)*
  - relativně jednoduché na řešení



- Rozšíření
  - variabilní doba trvání (spojena s cenou provádění)
    - optimalizace: kompromis mezi cenou na ukončení projektu a cenou za zkrácení délky úloh

Plánování projektu

- Základní problém plánování projektu
  - precedenční podmínky
  - paralelní stroj s neomezeným počtem strojů
  - minimalizace maximálního času konce úloh *(makespan)*
  - relativně jednoduché na řešení



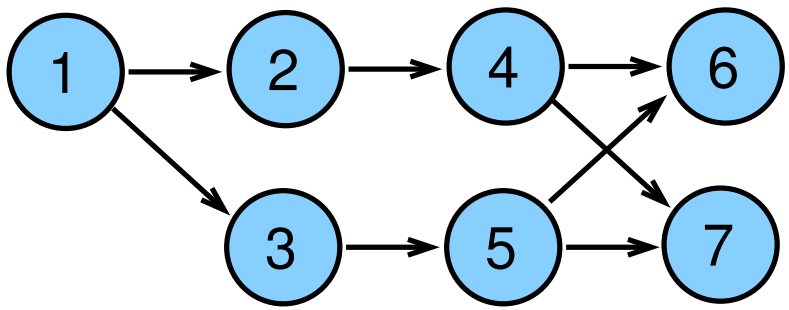
- Rozšíření
  - variabilní doba trvání (spojena s cenou provádění)
    - optimalizace: kompromis mezi cenou na ukončení projektu a cenou za zkrácení délky úloh
  - pracovní síla (skupiny operátorů s odlišnou specializací)
    - při sdílení omezeného množství operátorů nutno uvažovat disjunktivní hrany

    $\Rightarrow$ komplexní problém, jehož řešení je velmi obtížné

Reprezentace projektu: úloha jako uzel

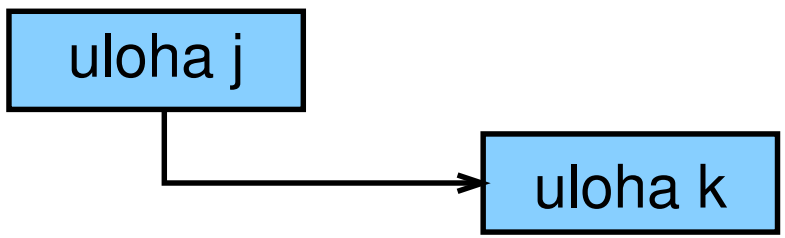
- Reprezentace: úloha jako uzel
  - hrany reprezentují precedenční podmínky
  - síť neobsahuje žádné orientované cykly

Úloha	Doba trvání	Předchůdci
1	2	–
2	3	1
3	1	1
4	4	2
5	2	3
6	1	4,5
7	3	4,5



Výhodnější reprezentace

- Běžně používana reprezentace úloha jako hrana
- Výhodnější ale úloha jako uzel
  - nejsou nutné redundantní hrany (pomocné úlohy) pro korektní vyjádření precedencí
  - úloha jako uzel lze převést na úloha jako obdelník
    - horizontální strany obdelníku použity jako časové osy odpovídající době provádění úlohy



Popis problému

- Popis problému
  - m paralelně zapojených strojů
  - n úloh s precedenčními omezeními
  - doba provádění p_j
  - objektivní funkce: minimalizace maximálního času konce úloh *(makespan)*
- $P_\infty|prec|C_{max}$ (a $m \geq n$) polynomiální složitost, metoda kritické cesty
  $P_m|prec|C_{max}$ $2 \leq m < n$ NP úplný problém

Popis problému

- Popis problému
  - m paralelně zapojených strojů
  - n úloh s precedenčními omezeními
  - doba provádění p_j
  - objektivní funkce: minimalizace maximálního času konce úloh *(makespan)*
- $P_\infty|prec|C_{max}$ (a $m \geq n$) polynomiální složitost, metoda kritické cesty
  $P_m|prec|C_{max}$ $2 \leq m < n$ NP úplný problém
- Značení
  - S'_j nejdřívější startovní čas úlohy j
    $C'_j = S'_j + p_j$ nejdřívější koncový úlohy j
  - S''_j nejpozdější startovní čas úlohy j
    C''_j nejpozdější koncový čas úlohy j
  - $Prec_j$ (přímí) předchůdci úlohy j
    $\forall k \in Prec_j$ všechny úlohy k, které předcházejí úlohu j
    $\forall j : k \in Prec_j$ všechny úlohy j, které následují úlohu k

Metoda kritické cesty

- Popis algoritmu pro nalezení kritické cesty
  - dopředná procedura
    - start v čase 0
    - výpočet nejdřívějšího startovního času každé úlohy
    - čas dokončení poslední úlohy je *makespan*

Metoda kritické cesty

- Popis algoritmu pro nalezení kritické cesty
  - dopředná procedura
    - start v čase 0
    - výpočet nejdřívějšího startovního času každé úlohy
    - čas dokončení poslední úlohy je *makespan*
  - zpětná procedura
    - start v čase rovném *makespan*
    - výpočet nejpozdějšího startovního času, aby byl realizován tento *makespan*

Metoda kritické cesty

- Popis algoritmu pro nalezení kritické cesty
  - dopředná procedura
    - start v čase 0
    - výpočet nejdřívějšího startovního času každé úlohy
    - čas dokončení poslední úlohy je *makespan*
  - zpětná procedura
    - start v čase rovném *makespan*
    - výpočet nejpozdějšího startovního času, aby byl realizován tento *makespan*
- Úloha s rezervou *(slack job)*
  - její startovní čas může být odložen aniž je navýšen *makespan*
  - úloha, jejichž nejdřívější startovní čas je menší než nejpozdější startovní čas

Metoda kritické cesty

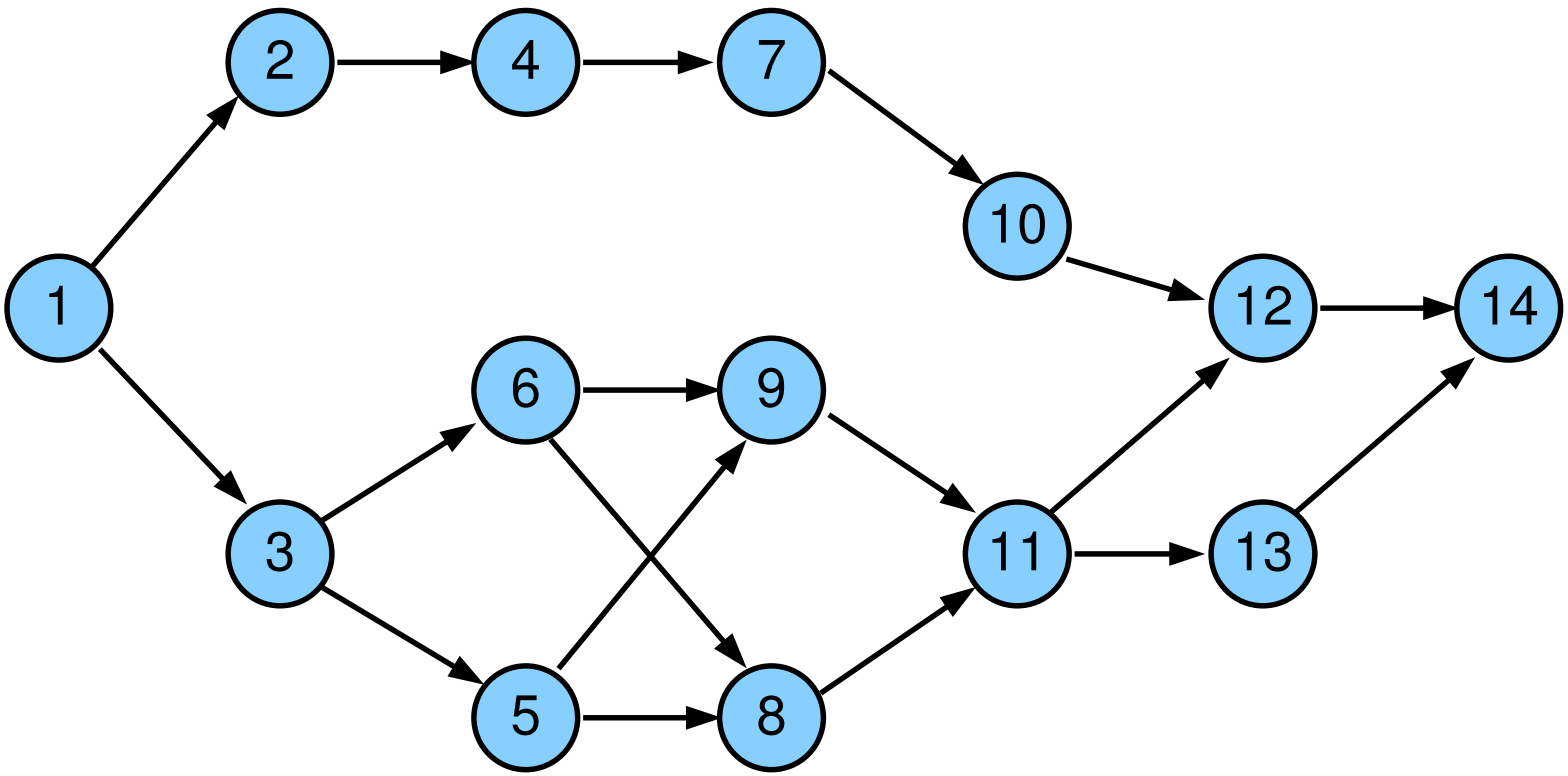
- Popis algoritmu pro nalezení kritické cesty
  - dopředná procedura
    - start v čase 0
    - výpočet nejdřívějšího startovního času každé úlohy
    - čas dokončení poslední úlohy je *makespan*
  - zpětná procedura
    - start v čase rovném *makespan*
    - výpočet nejpozdějšího startovního času, aby byl realizován tento *makespan*
- Úloha s rezervou *(slack job)*
  - její startovní čas může být odložen aniž je navýšen *makespan*
  - úloha, jejichž nejdřívější startovní čas je menší než nejpozdější startovní čas
- Kritická úloha
  - úloha, která nesmí být odložena
  - úlohy, jejichž nejdřívější startovní čas je roven nejpozdějšímu startovnímu čas

Metoda kritické cesty

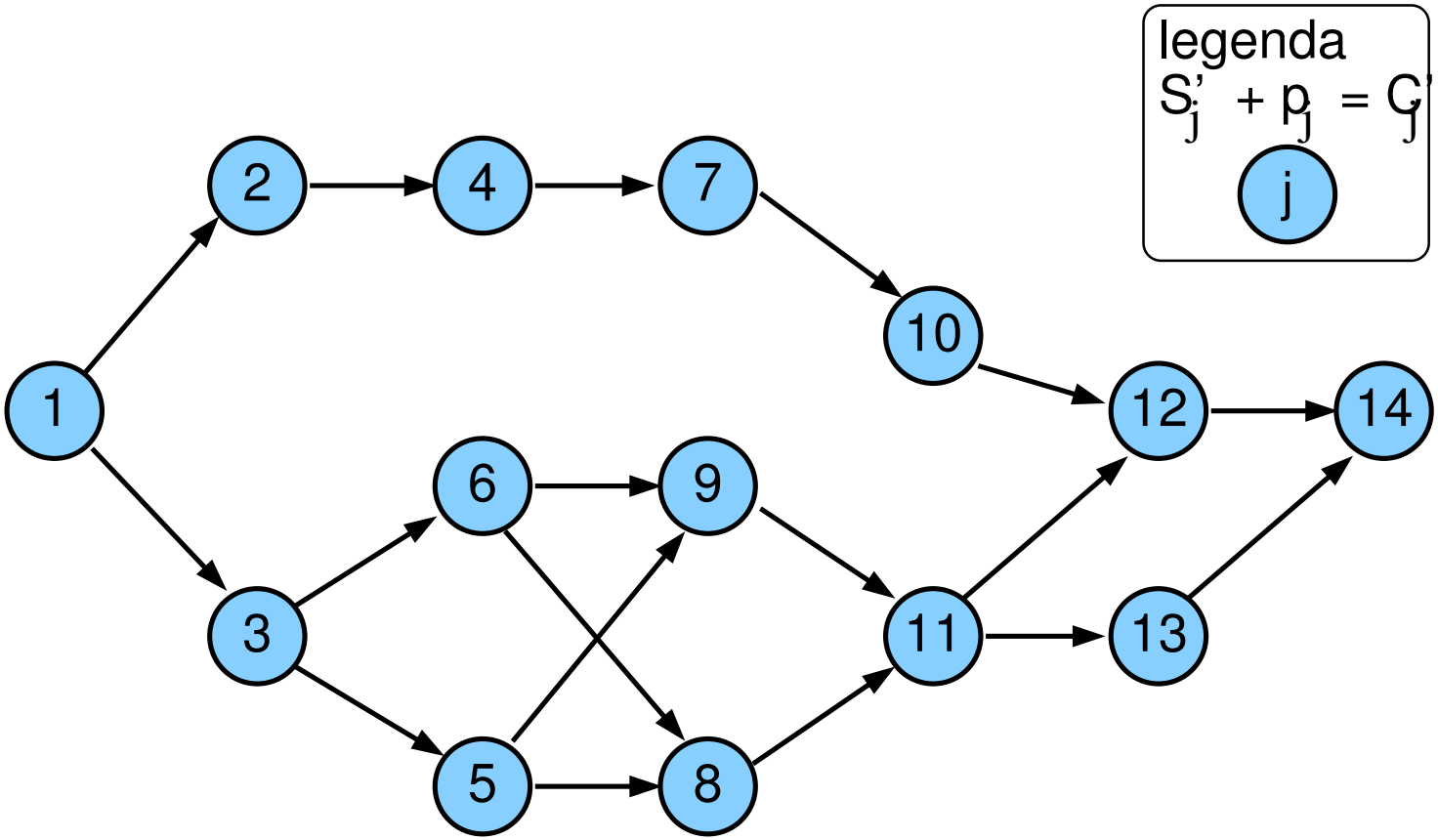
- Popis algoritmu pro nalezení kritické cesty
  - dopředná procedura
    - start v čase 0
    - výpočet nejdřívějšího startovního času každé úlohy
    - čas dokončení poslední úlohy je *makespan*
  - zpětná procedura
    - start v čase rovném *makespan*
    - výpočet nejpozdějšího startovního času, aby byl realizován tento *makespan*
- Úloha s rezervou *(slack job)*
  - její startovní čas může být odložen aniž je navýšen *makespan*
  - úloha, jejichž nejdřívější startovní čas je menší než nejpozdější startovní čas
- Kritická úloha
  - úloha, která nesmí být odložena
  - úlohy, jejichž nejdřívější startovní čas je roven nejpozdějšímu startovnímu čas
- Kritická cesta
  - řetěz úloh začínající v čase 0 a končící v čase C_{max}
  - v grafu může existovat více kritických cest
    kritické cesty se mohou částečně překrývat
  - graf kritických cest G_{CP}: podgraf daný množinou kritických úloh a kritických cest

Kritická cesta: zadání příkladu

j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
$\mathbf{p_j}$	5	6	9	12	7	12	10	6	10	9	7	8	7	5

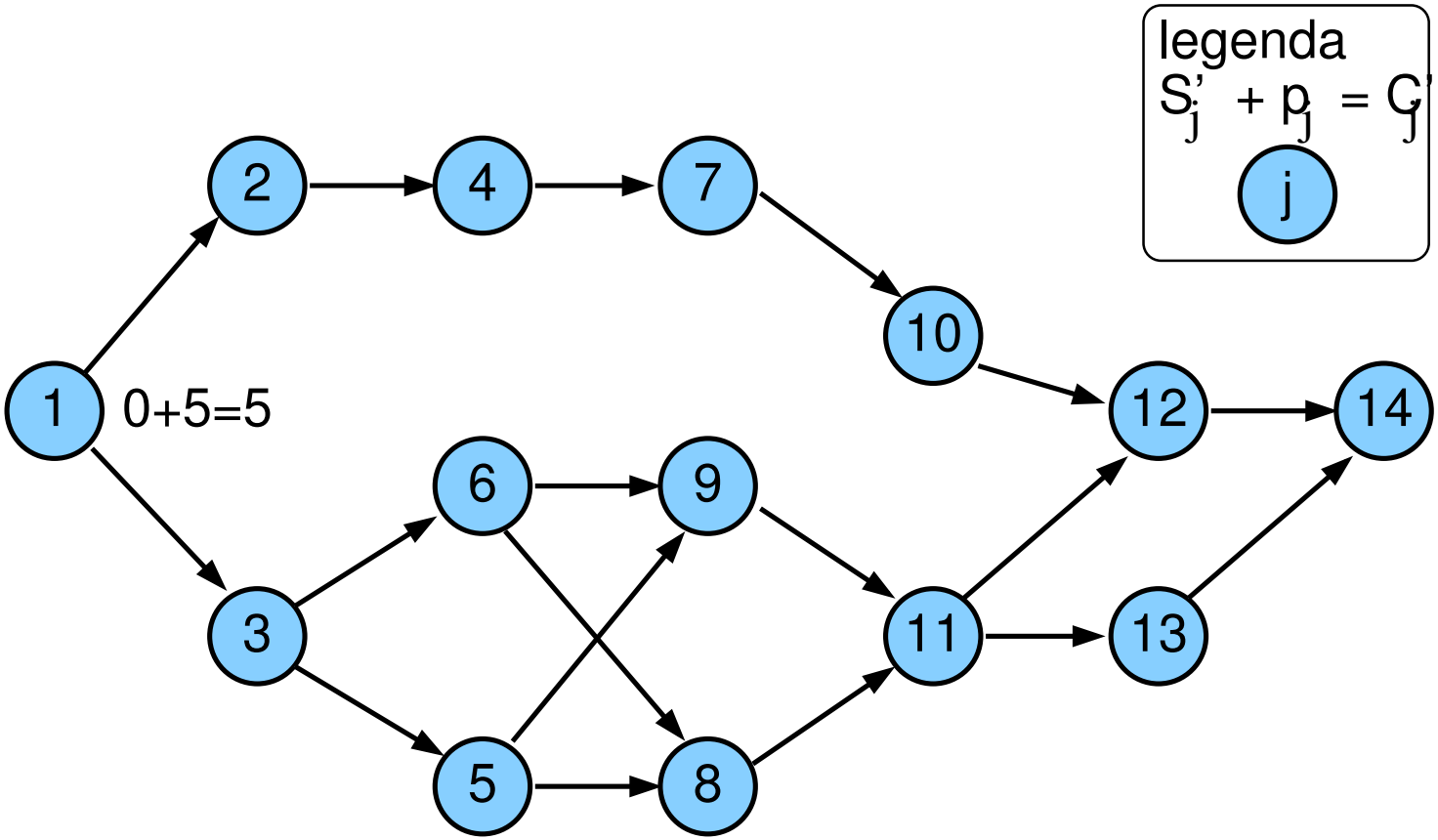


Příklad: dopředná procedura



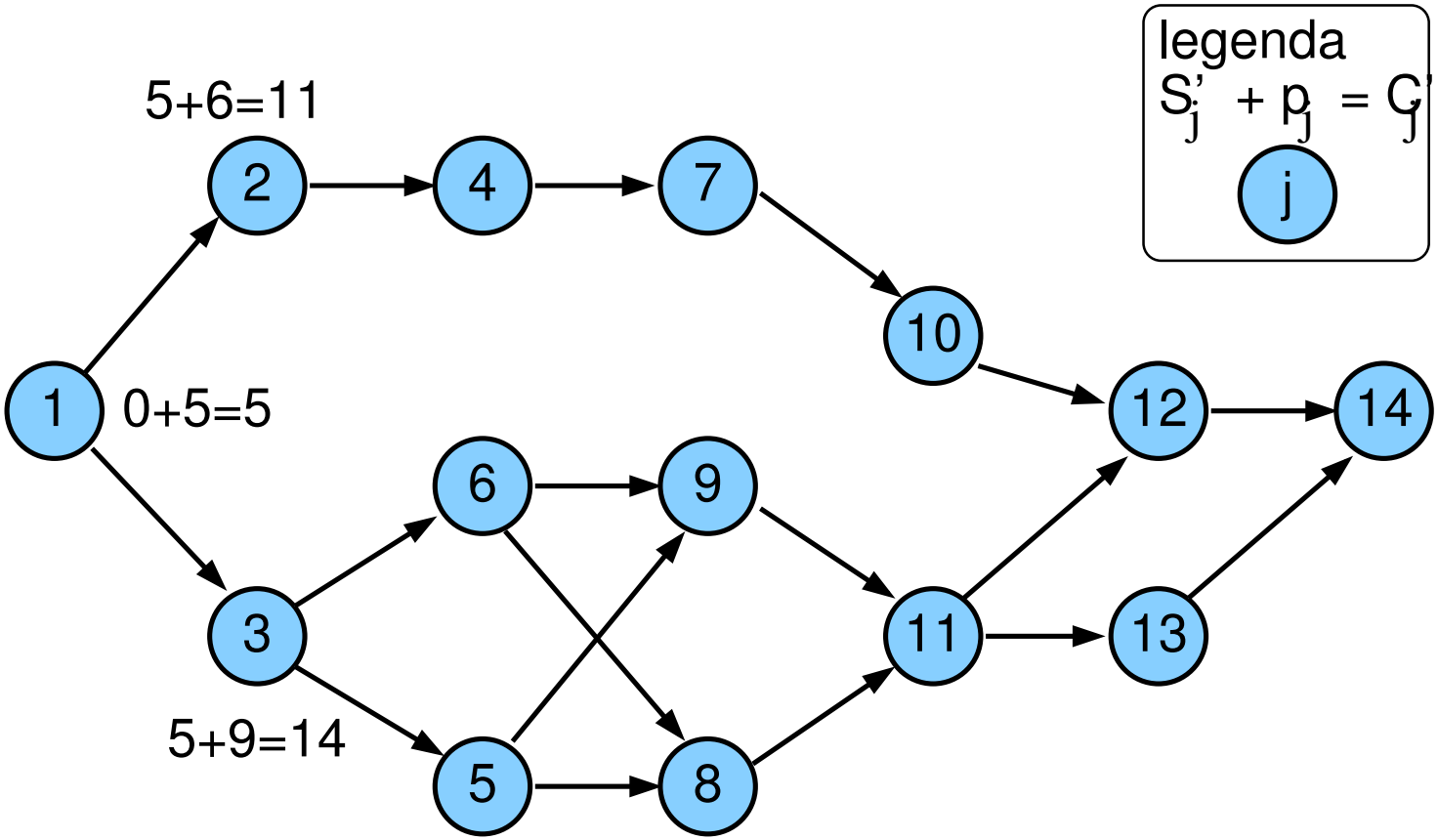
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: dopředná procedura



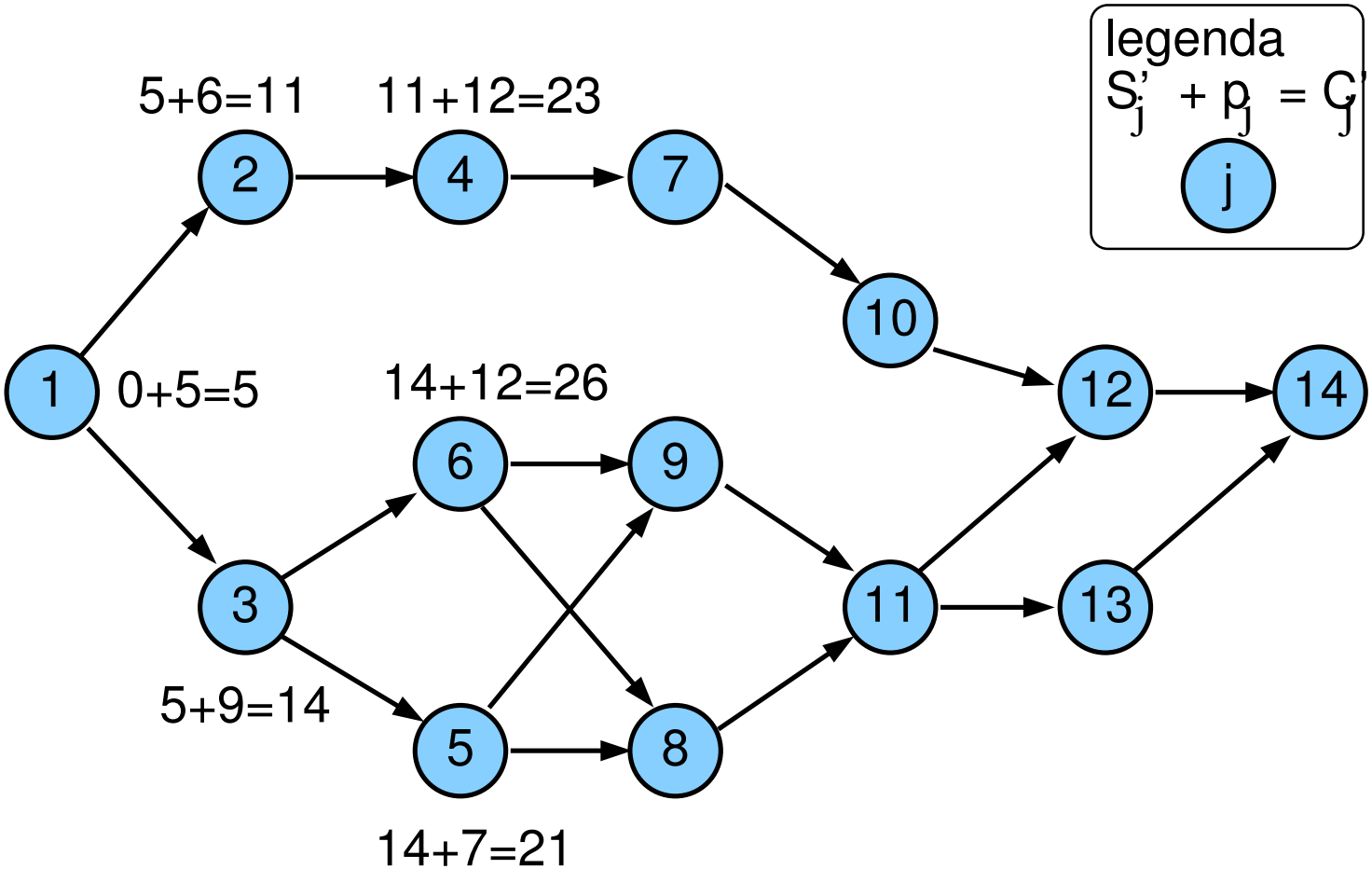
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: dopředná procedura



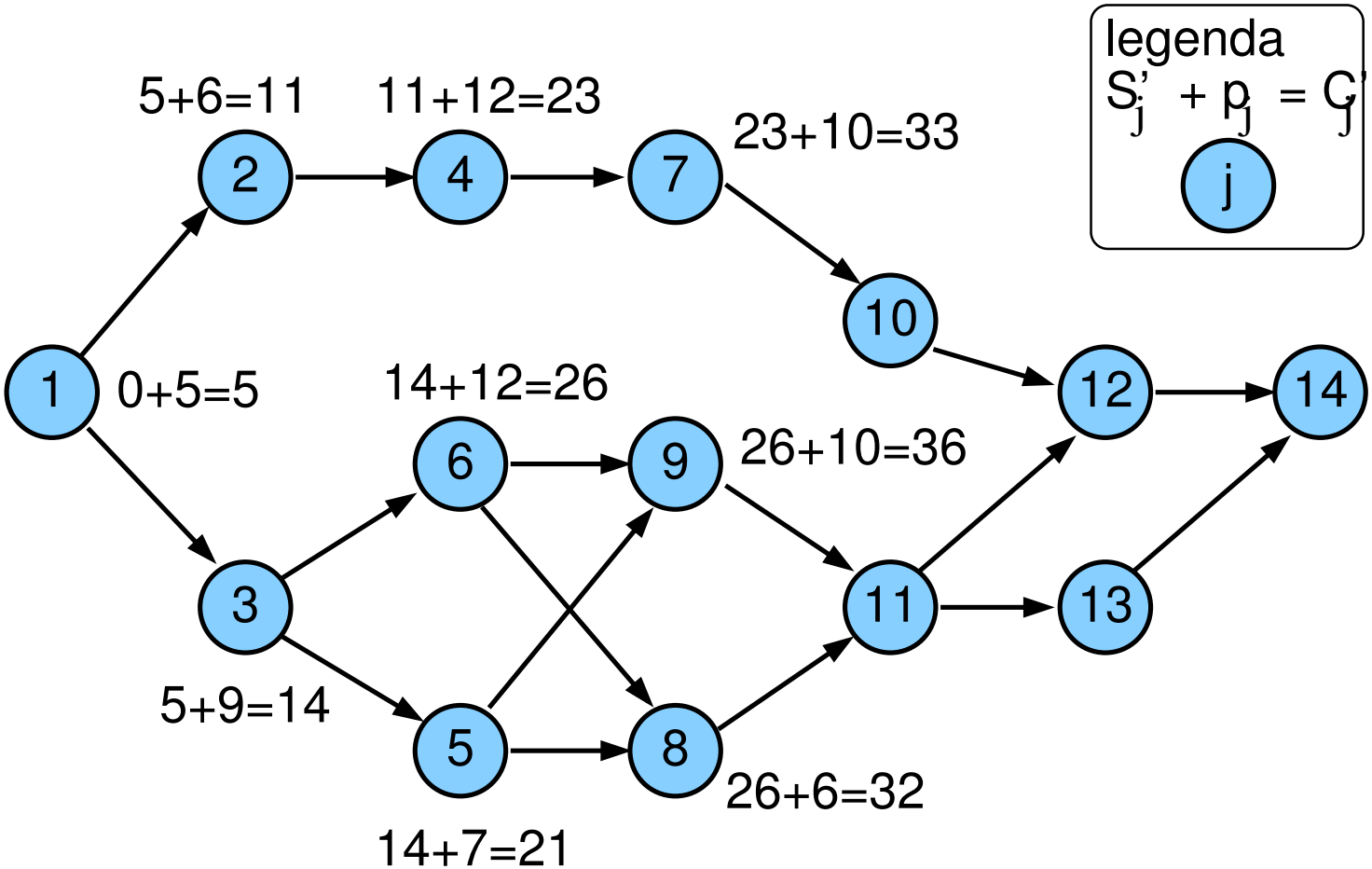
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: dopředná procedura



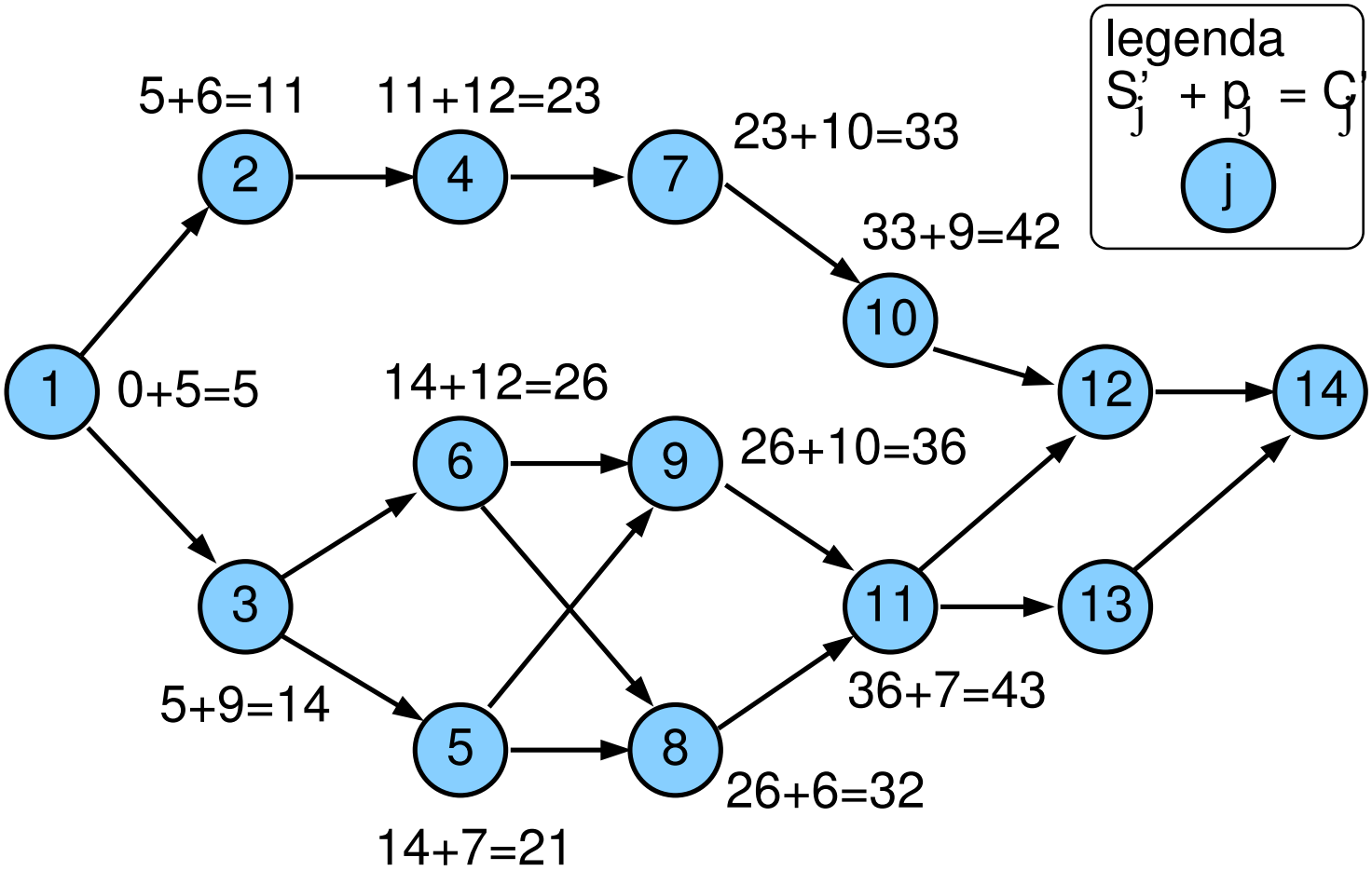
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: dopředná procedura



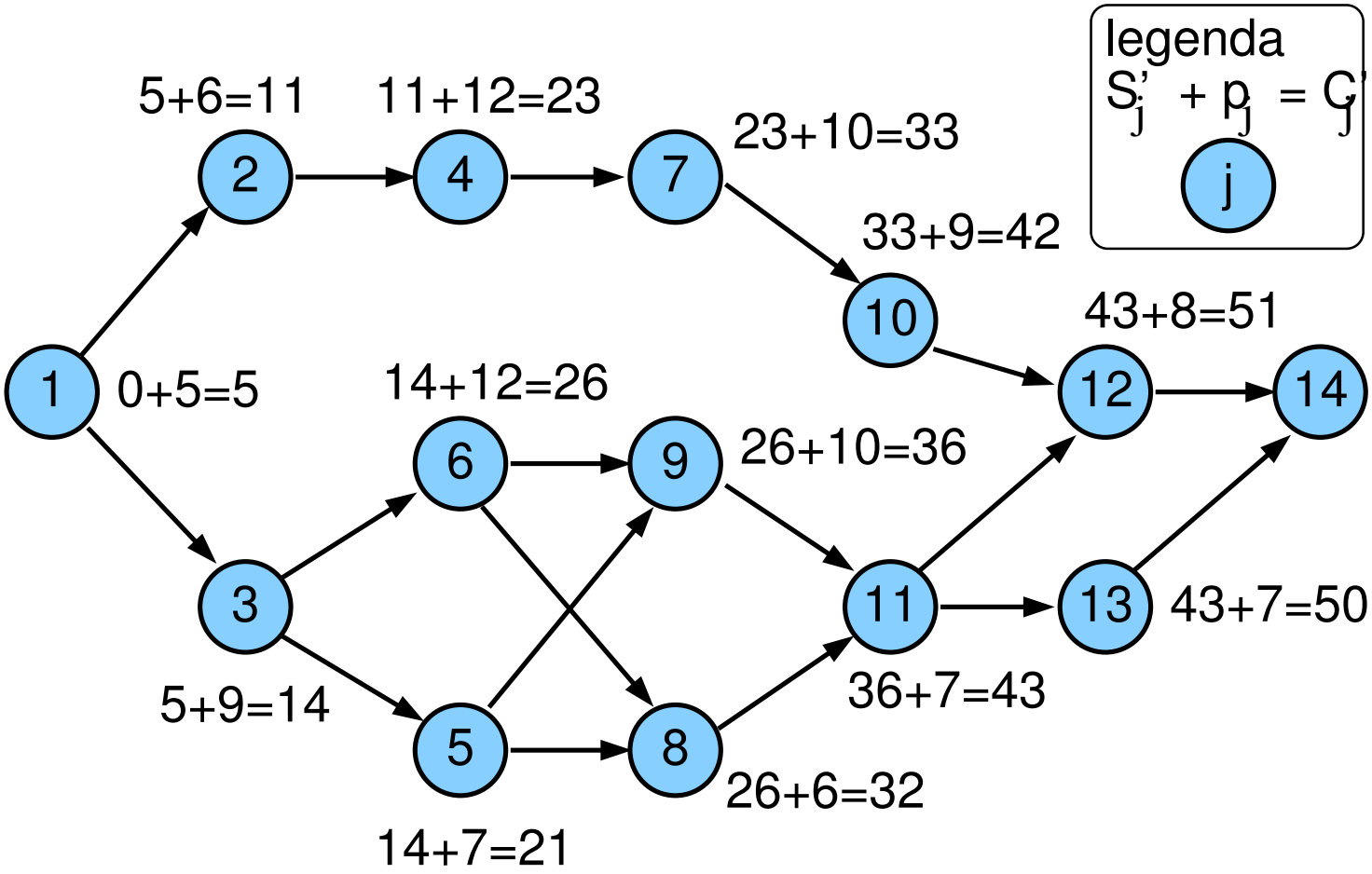
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: dopředná procedura



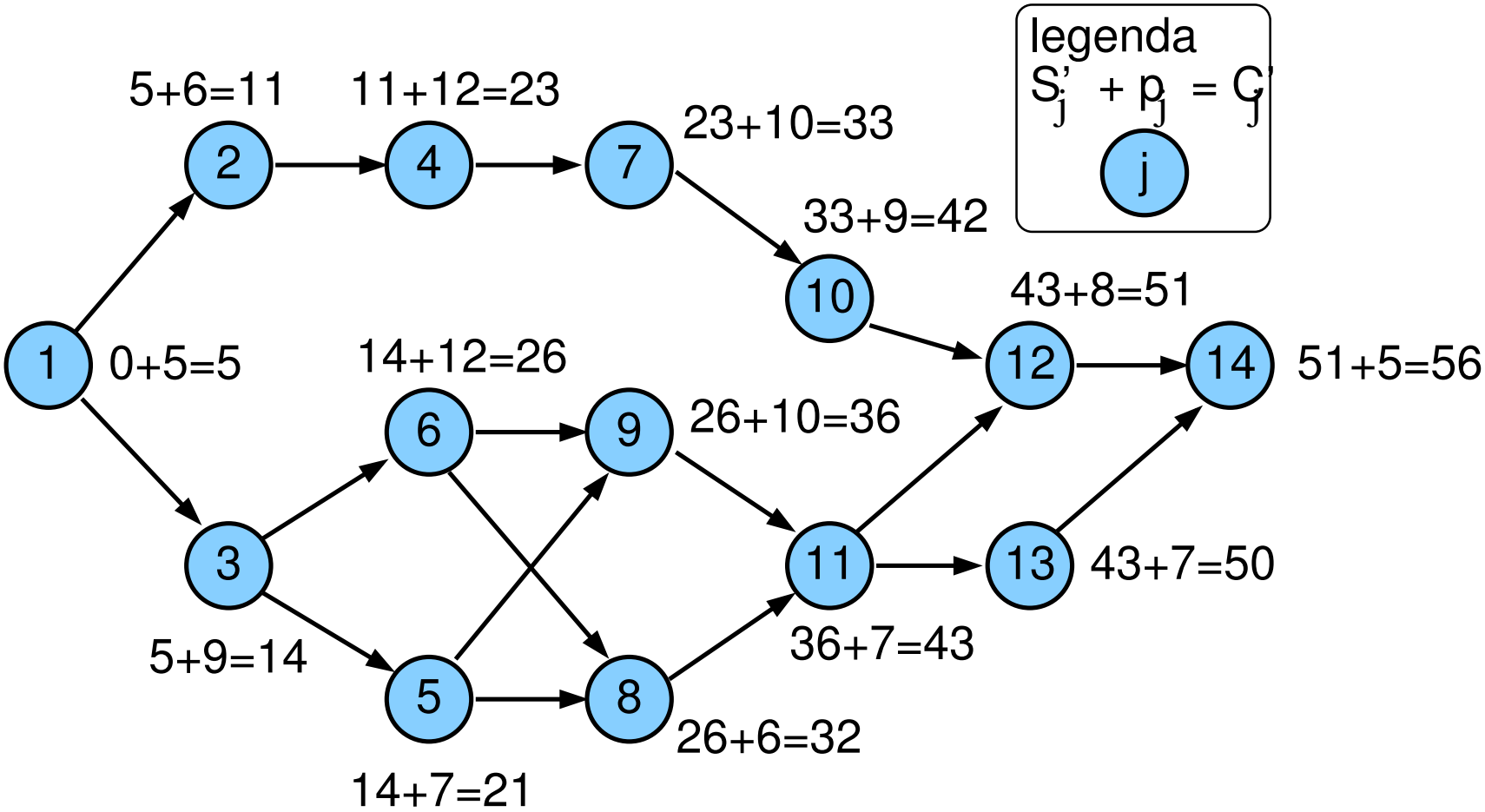
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: dopředná procedura



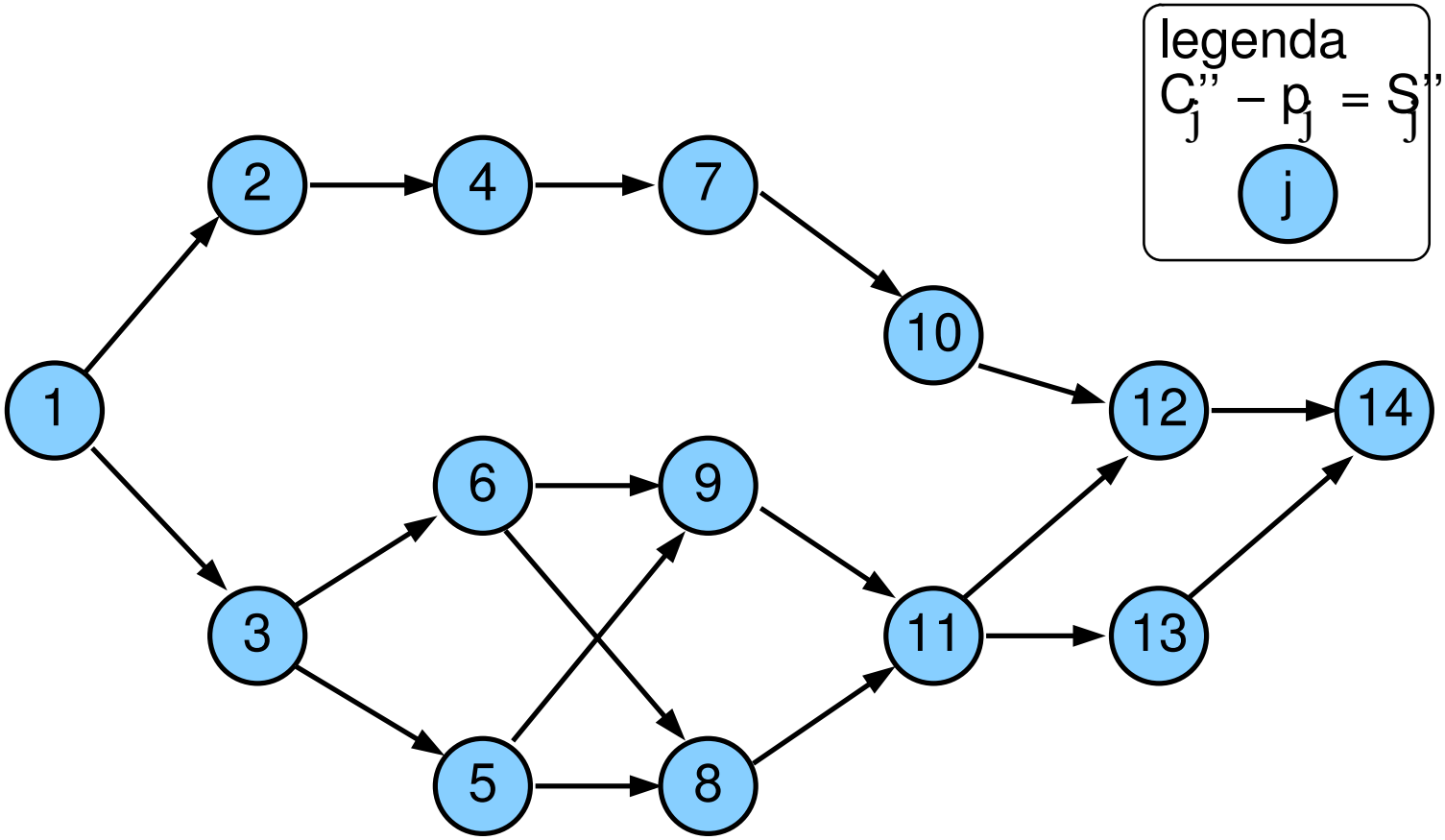
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: dopředná procedura



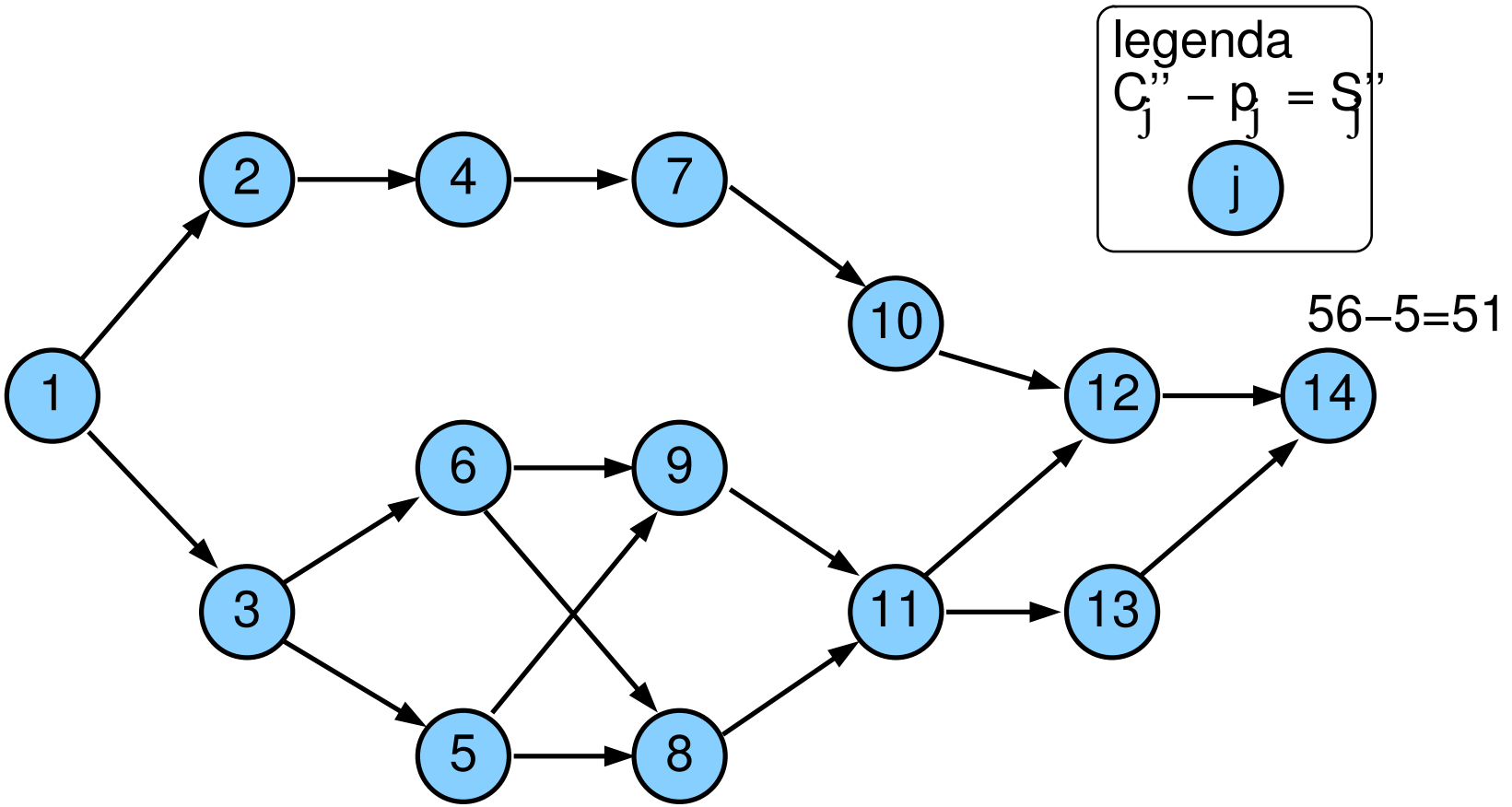
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: zpětná procedura



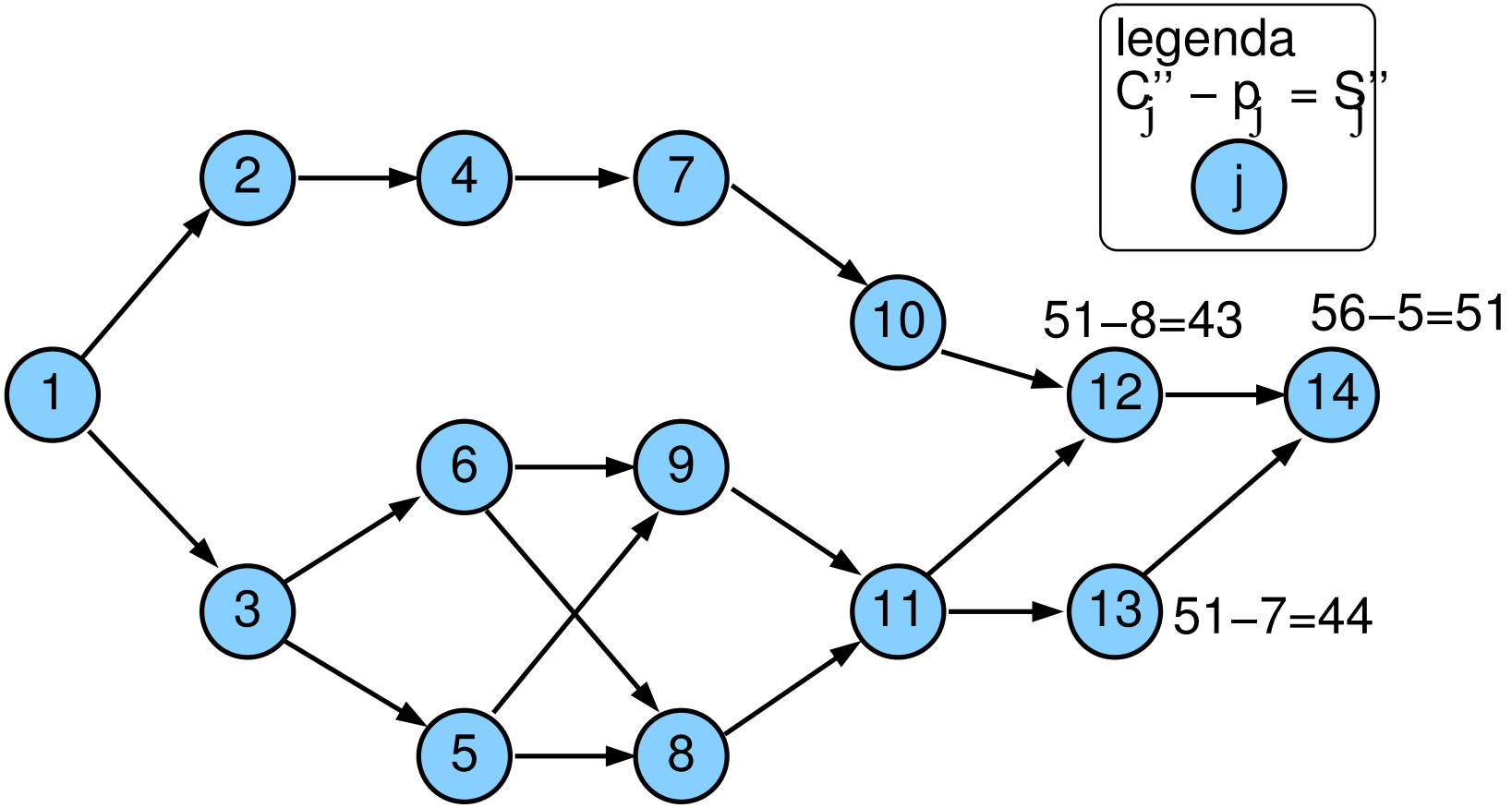
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: zpětná procedura



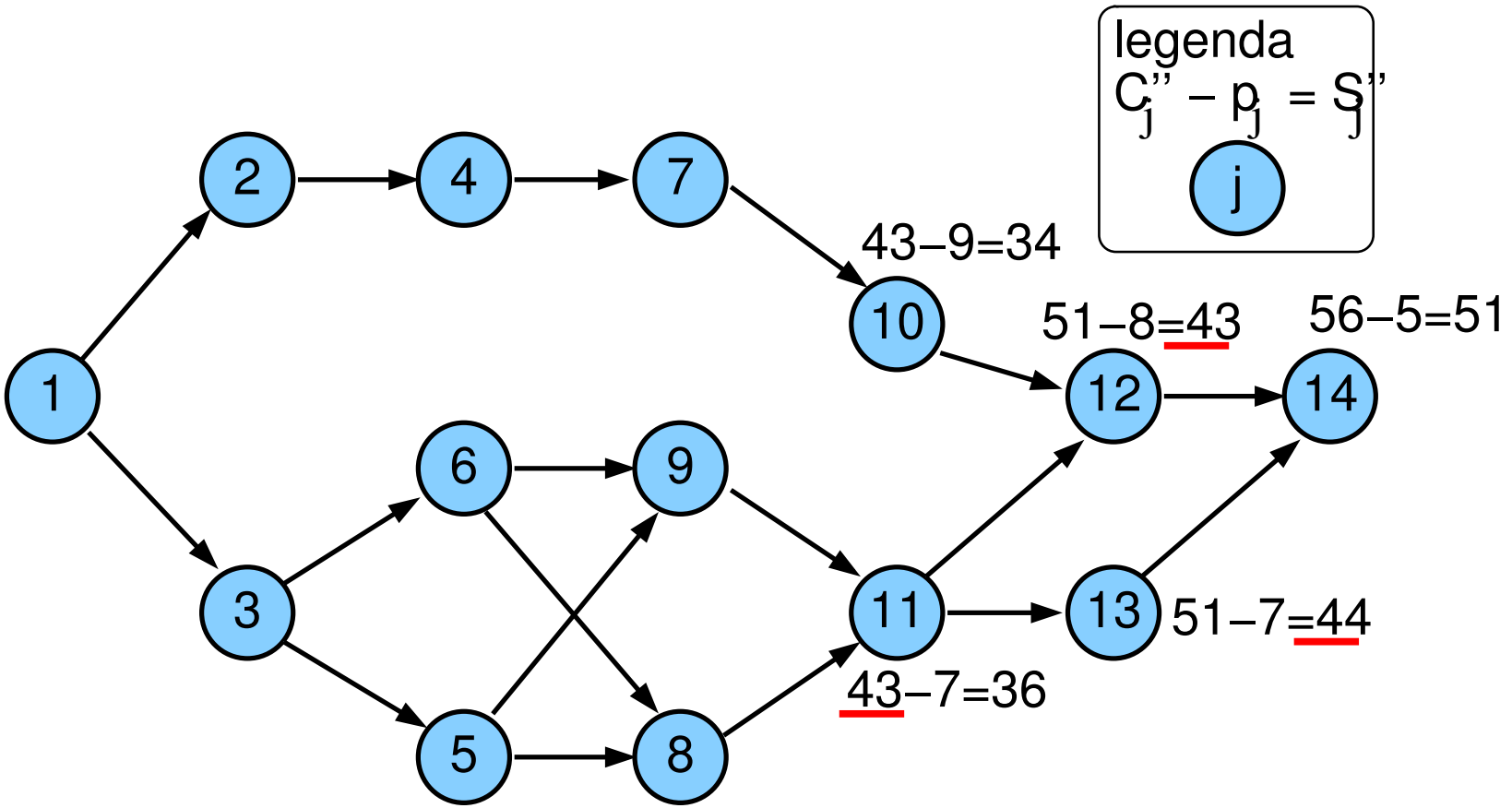
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: zpětná procedura



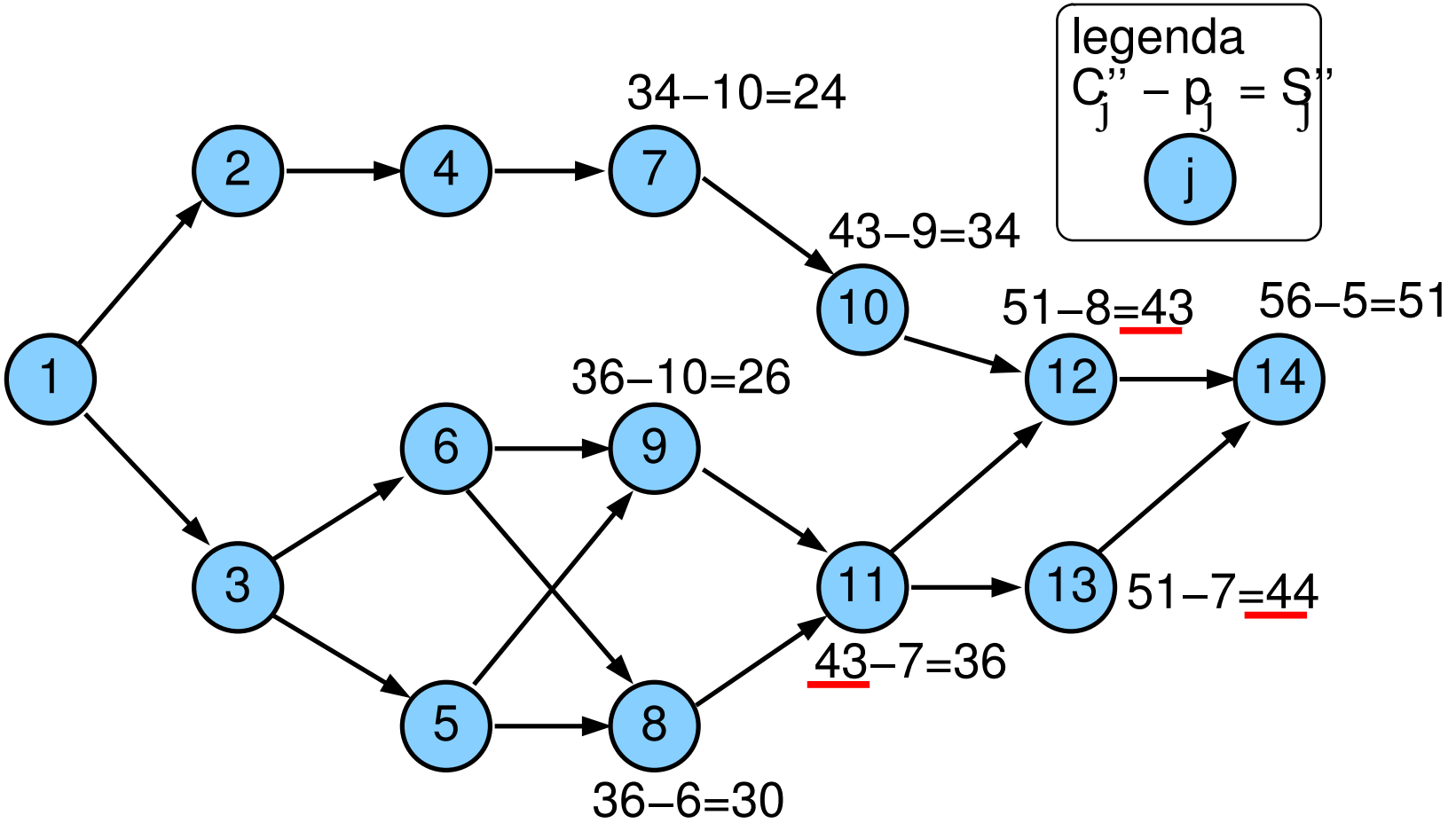
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: zpětná procedura



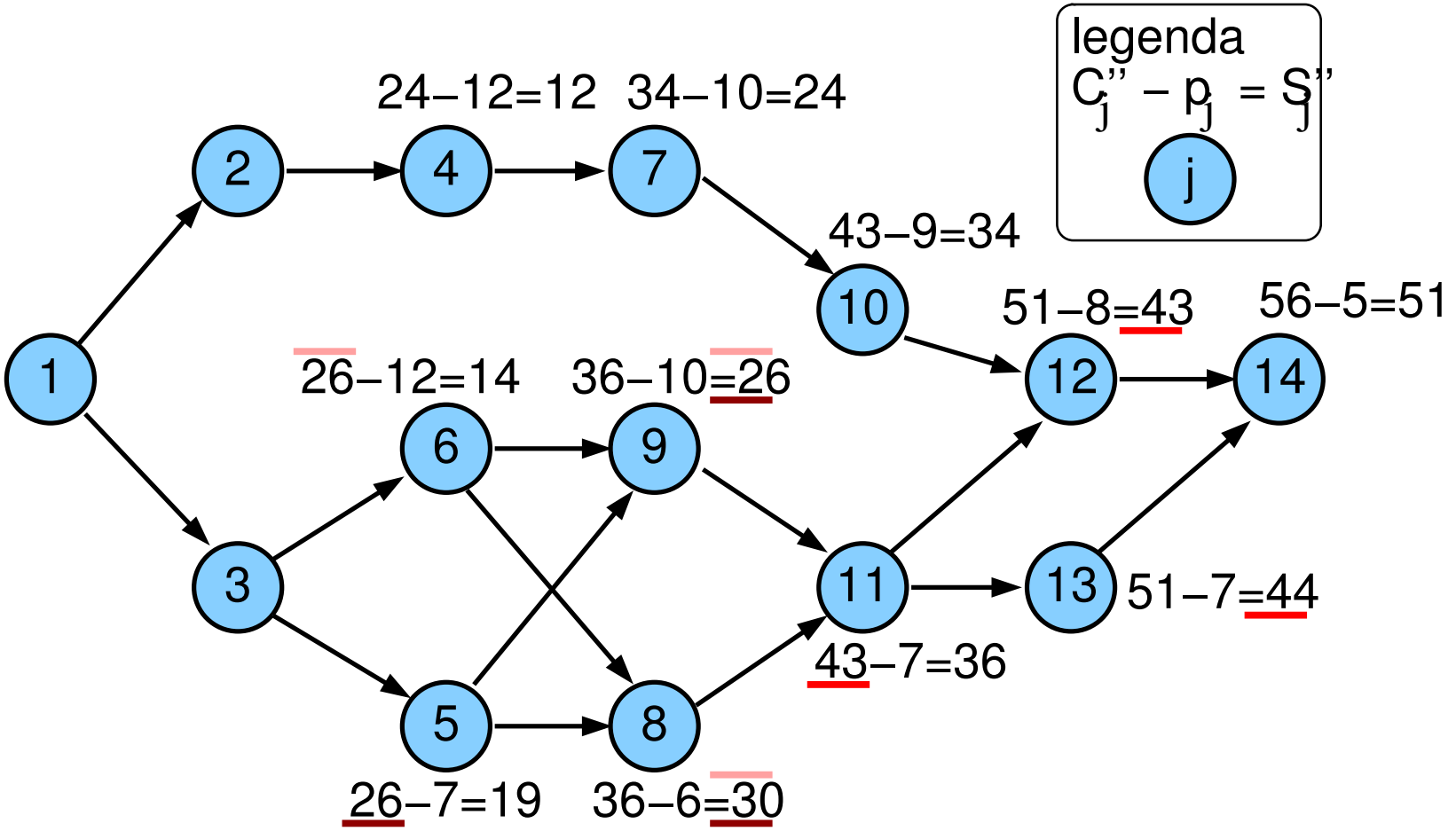
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: zpětná procedura



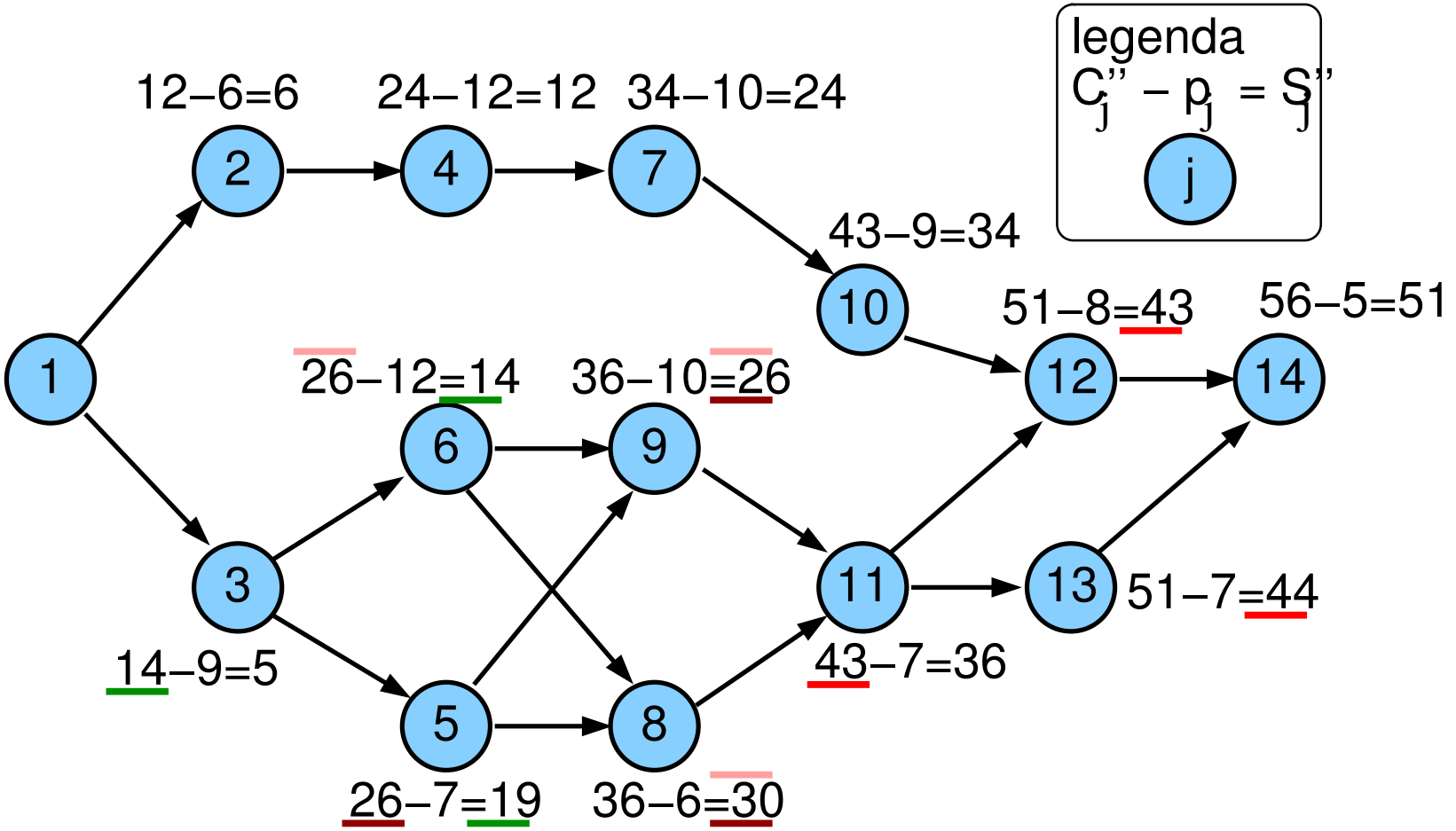
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: zpětná procedura



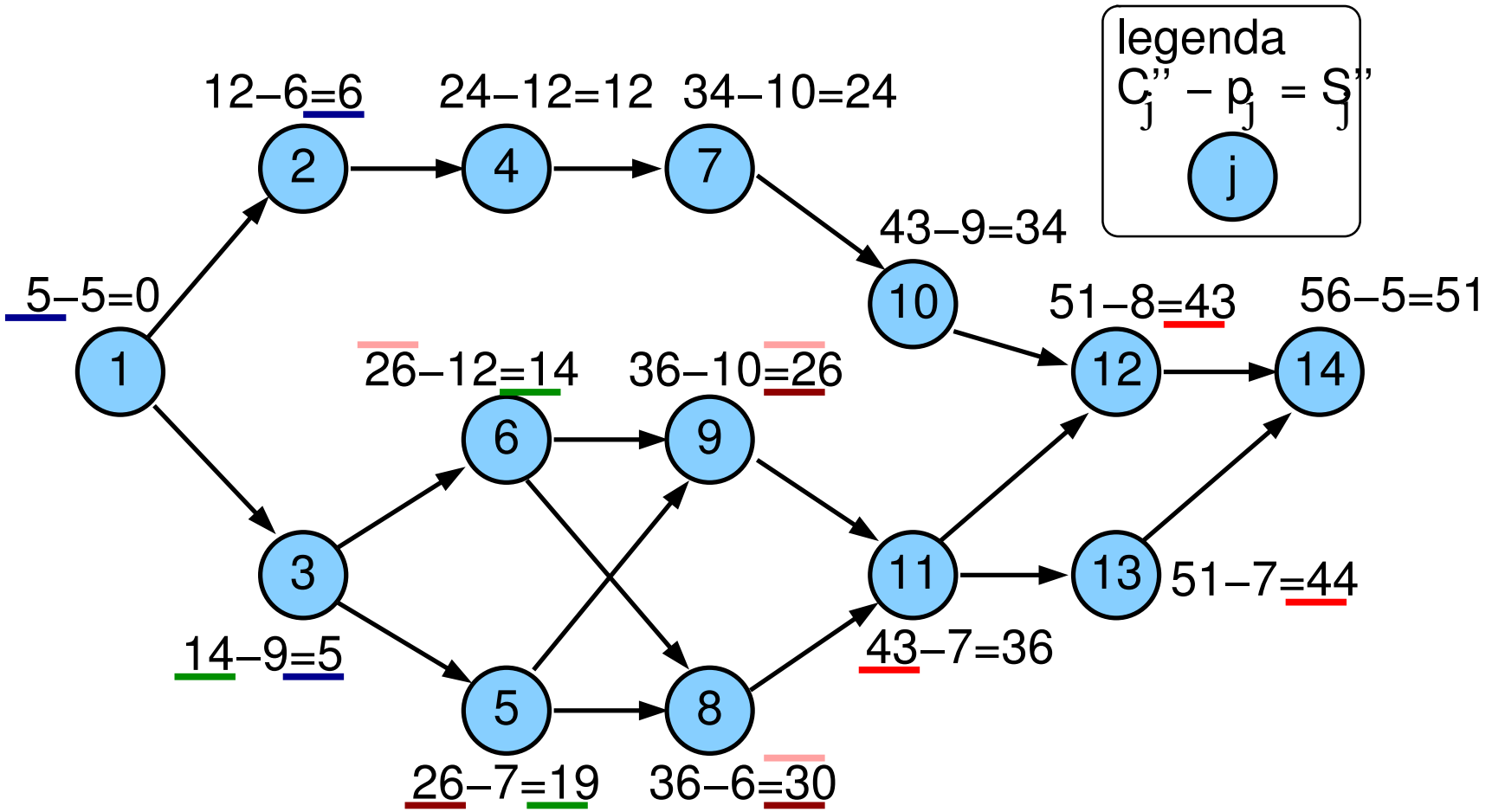
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: zpětná procedura



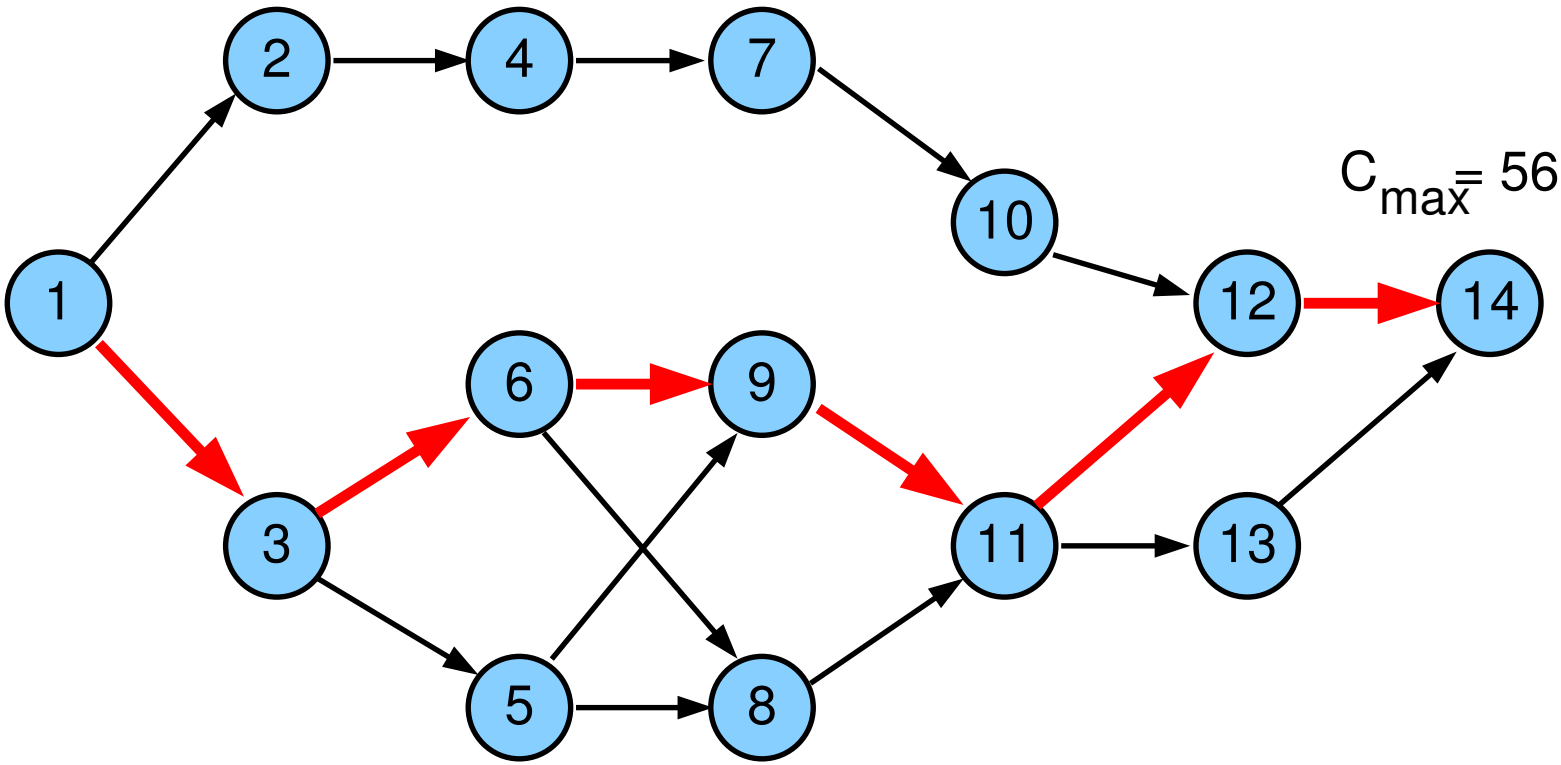
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Příklad: zpětná procedura



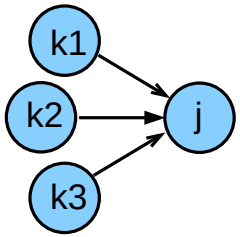
j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j	5	6	9	12	7	12	10	6	10	9	7	8	7	5

Kritická cesta



Algoritmus pro nalezení kritické cesty

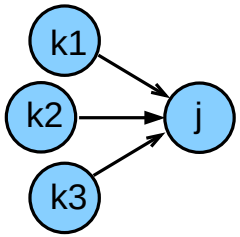
1. Dopředná procedura
   1. $t = 0$
   2. pro všechny úlohy j bez předchůdce: $S'_j = 0, C'_j = p_j$
   3. vypočítej postupně pro všechny zbývající úlohy j:



$$S'_j =$$

Algoritmus pro nalezení kritické cesty

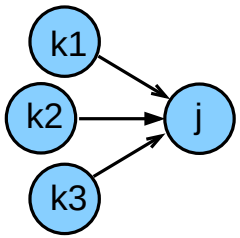
1. Dopředná procedura
   1. $t = 0$
   2. pro všechny úlohy j bez předchůdce: $S'_j = 0, C'_j = p_j$
   3. vypočítej postupně pro všechny zbývající úlohy j:



$$S'_j = \max_{\forall k \in Prec_j} C'_k, \quad C'_j = S'_j + p_j$$

Algoritmus pro nalezení kritické cesty

1. Dopředná procedura
   1. $t = 0$
   2. pro všechny úlohy j bez předchůdce: $S'_j = 0, C'_j = p_j$
   3. vypočítej postupně pro všechny zbývající úlohy j:

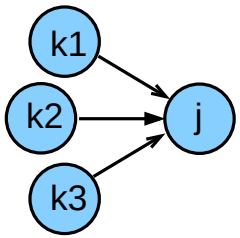


$$S'_j = \max_{\forall k \in Prec_j} C'_k, \quad C'_j = S'_j + p_j$$

   4. optimální *makespan* je $C_{max} = \max(C'_1, \ldots, C'_n)$

Algoritmus pro nalezení kritické cesty

1. Dopředná procedura
   1. $t = 0$
   2. pro všechny úlohy j bez předchůdce: $S'_j = 0, C'_j = p_j$
   3. vypočítej postupně pro všechny zbývající úlohy j:

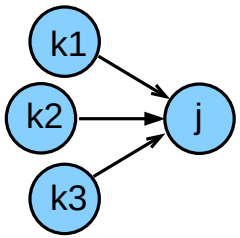


$$S'_j = \max_{\forall k \in Prec_j} C'_k, \quad C'_j = S'_j + p_j$$

   4. optimální *makespan* je $C_{max} = \max(C'_1, \ldots, C'_n)$
2. Zpětná procedura
   - $t = C_{max}$
   - pro všechny úlohy j bez následníka: $C''_j = C_{max}, S''_j = C_{max} - p_j$

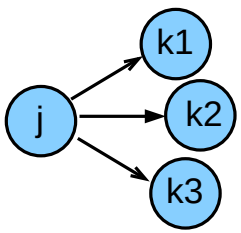
Algoritmus pro nalezení kritické cesty

1. Dopředná procedura
   1. $t = 0$
   2. pro všechny úlohy j bez předchůdce: $S'_j = 0, C'_j = p_j$
   3. vypočítej postupně pro všechny zbývající úlohy j:



$$S'_j = \max_{\forall k \in Prec_j} C'_k, \quad C'_j = S'_j + p_j$$

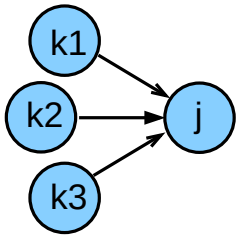
   4. optimální *makespan* je $C_{max} = \max(C'_1, \ldots, C'_n)$
2. Zpětná procedura
   - $t = C_{max}$
   - pro všechny úlohy j bez následníka: $C''_j = C_{max}, S''_j = C_{max} - p_j$
   - vypočítej postupně pro všechny zbývající úlohy j:



$$C''_j =$$

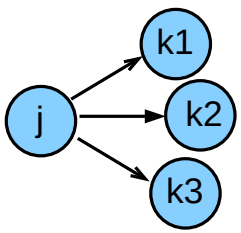
Algoritmus pro nalezení kritické cesty

1. Dopředná procedura
   1. $t = 0$
   2. pro všechny úlohy j bez předchůdce: $S'_j = 0, C'_j = p_j$
   3. vypočítej postupně pro všechny zbývající úlohy j:



$$S'_j = \max_{\forall k \in Prec_j} C'_k, \quad C'_j = S'_j + p_j$$

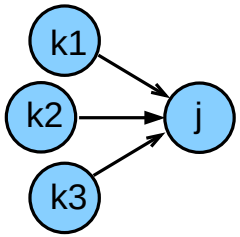
   4. optimální *makespan* je $C_{max} = \max(C'_1, \ldots, C'_n)$
2. Zpětná procedura
   - $t = C_{max}$
   - pro všechny úlohy j bez následníka: $C''_j = C_{max}, S''_j = C_{max} - p_j$
   - vypočítej postupně pro všechny zbývající úlohy j:



$$C''_j = \min_{\forall k: j \in Prec_k} S''_k, \quad S''_j = C''_j - p_j$$

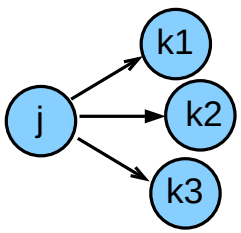
Algoritmus pro nalezení kritické cesty

1. Dopředná procedura
   1. $t = 0$
   2. pro všechny úlohy j bez předchůdce: $S'_j = 0, C'_j = p_j$
   3. vypočítej postupně pro všechny zbývající úlohy j:



$$S'_j = \max_{\forall k \in Prec_j} C'_k, \quad C'_j = S'_j + p_j$$

   4. optimální *makespan* je $C_{max} = \max(C'_1, \ldots, C'_n)$
2. Zpětná procedura
   - $t = C_{max}$
   - pro všechny úlohy j bez následníka: $C''_j = C_{max}, S''_j = C_{max} - p_j$
   - vypočítej postupně pro všechny zbývající úlohy j:

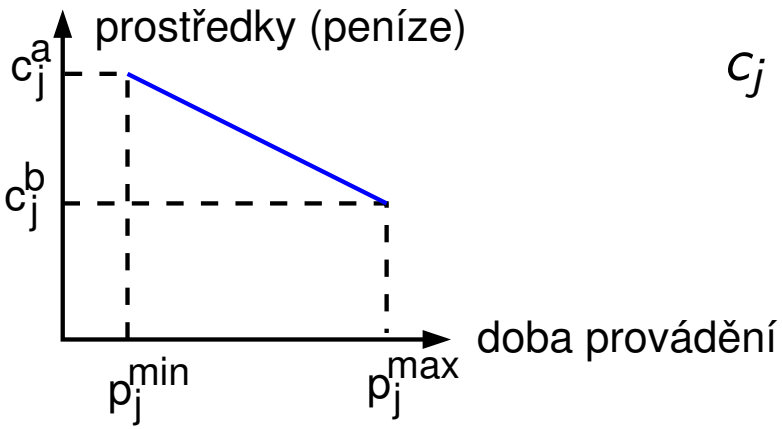


$$C''_j = \min_{\forall k: j \in Prec_k} S''_k, \quad S''_j = C''_j - p_j$$

   - ověř, že $0 = \min(S''_1, \ldots, S''_n)$

Kompromis mezi časem a cenou

- Lze uvažovat variabilní dobu trvání úloh
  - za předpokladu vyšší ceny lze zkrátit dobu provádění
- Lineární cena
- Doba trvání $p_j^{min} \leq p_j \leq p_j^{max}$
- Marginální cena: cena za zkrácení doby trvání úlohy o 1 časovou jednotku

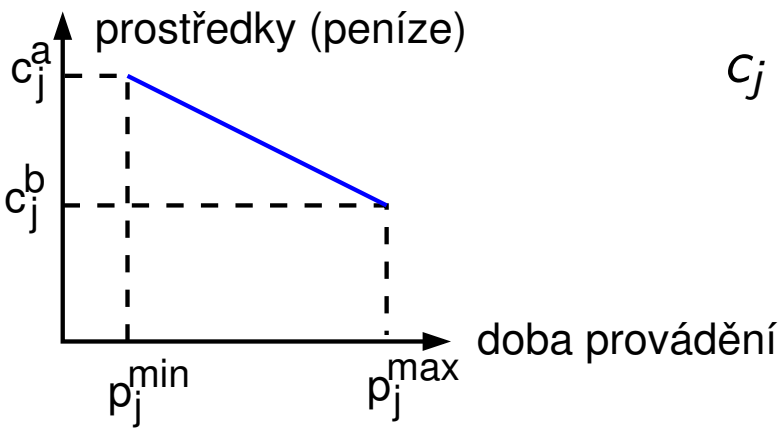


$$c_j = \frac{c_j^a - c_j^b}{p_j^{max} - p_j^{min}}$$

- př. $p_j^{min} = 10,\ p_j^{max} = 15,\ c_j^b = 10,\ c_j^a = 20, \quad c_j =$

Kompromis mezi časem a cenou

- Lze uvažovat variabilní dobu trvání úloh
  - za předpokladu vyšší ceny lze zkrátit dobu provádění
- Lineární cena
- Doba trvání $p_j^{min} \leq p_j \leq p_j^{max}$
- Marginální cena: cena za zkrácení doby trvání úlohy o 1 časovou jednotku

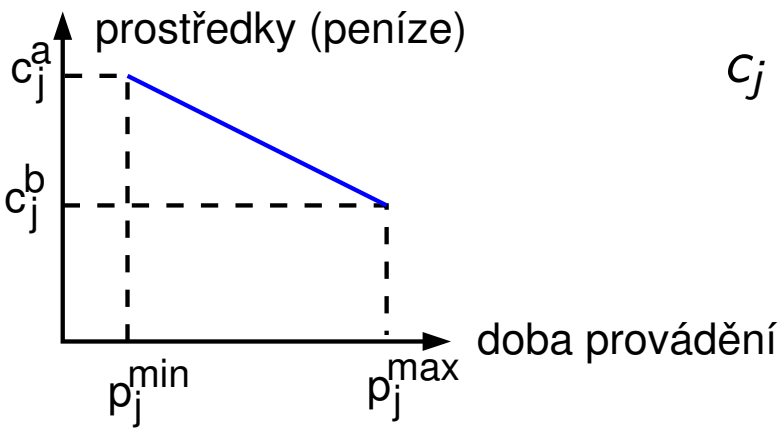


$$c_j = \frac{c_j^a - c_j^b}{p_j^{max} - p_j^{min}}$$

  - př. $p_j^{min} = 10,\ p_j^{max} = 15,\ c_j^b = 10,\ c_j^a = 20, \quad c_j = 2$

Kompromis mezi časem a cenou

- Lze uvažovat variabilní dobu trvání úloh
  - za předpokladu vyšší ceny lze zkrátit dobu provádění
- Lineární cena
- Doba trvání $p_j^{min} \leq p_j \leq p_j^{max}$
- Marginální cena: cena za zkrácení doby trvání úlohy o 1 časovou jednotku



$$c_j = \frac{c_j^a - c_j^b}{p_j^{max} - p_j^{min}}$$

  - př. $p_j^{min} = 10,\ p_j^{max} = 15,\ c_j^b = 10,\ c_j^a = 20, \quad c_j = 2$

$\Rightarrow$ cena provádění úlohy j po dobu p_j: $c_j^b + c_j(p_j^{max} - p_j)$

Cena za provádění projektu

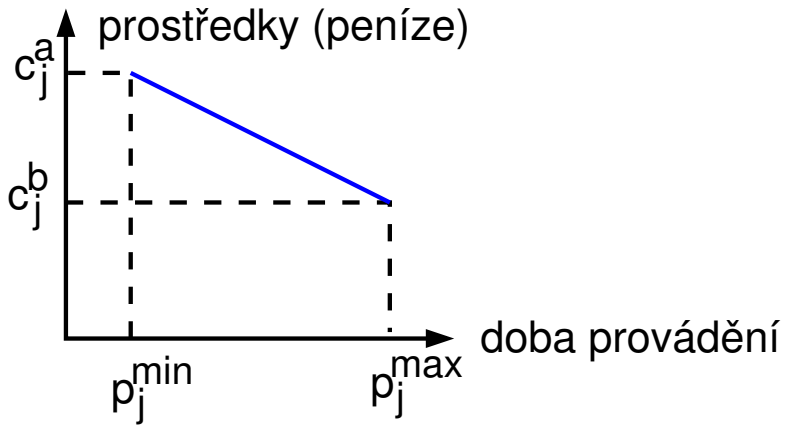
- Fixní režijní náklady c_0 celkem: $C_{max} c_0$

  na časovou jednotku doby provádění projektu
- Cena $F(p_j)$ za provádění projektu
  - při době provádění úloh p_j

  určena jako součet
  - ceny za provádění všech úloh
  - fixních režijních nákladů

$$F(p_j) = C_{max} c_0 + \sum_j \left(c_j^b + c_j (p_j^{max} - p_j) \right)$$

- Cíl: $\forall j$ nalézt p_j a S_j tak, aby byla $F(p_j)$ minimální



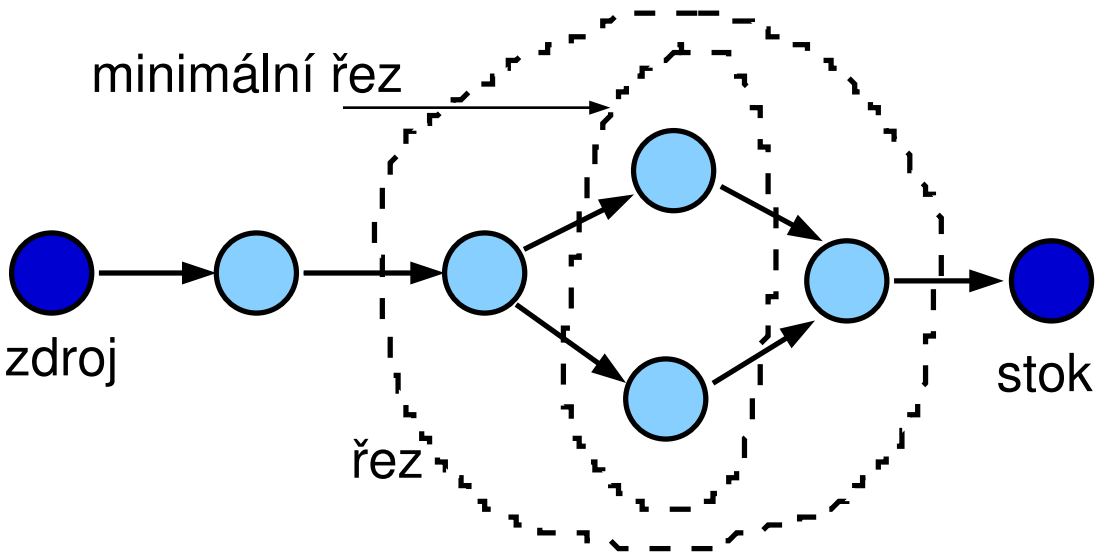
Variabilní doba trvání: metody řešení

- Objektivní funkce: minimální cena projektu
- Kompromisní heuristika mezi časem a cenou
  - dobrá kvalita rozvrhu
  - použitené i pro nelineární cenu
- Formulace lineárního programování
  - optimální rozvrh
  - nelineární verze obtížně řešitelné

Opakování: Řez, minimální řez

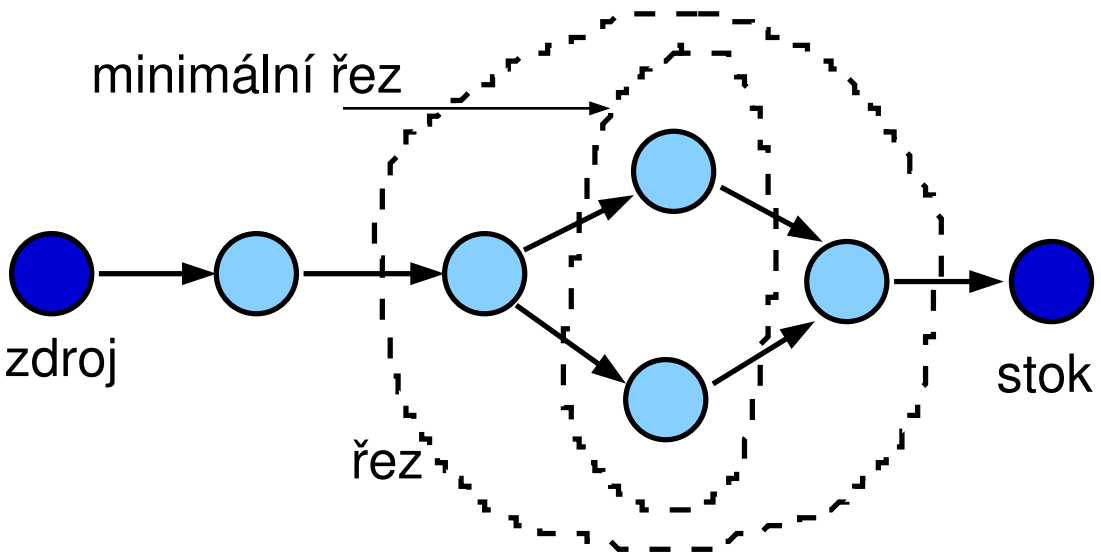
- Orientovaný graf $G = (V, E)$
- Počáteční uzel: zdroj $s \in V$
- Koncový uzel: stok $t \in V$
- Řez: ... také mluvíme o vrcholovém řezu
  množina uzlů V', jejíž smazáním z grafu se rozpojí zdroj a stok
  - E' množina hran incidentních s V'

  tj. v G'=(V-V',E-E') neexistuje orientovaná cesta z s to t
- Minimální řez: řez U takový, že neexistuje řez $W \subset U$
  - tj. vrácení libovolného uzlu z U do grafu znovu spojí zdroj a stok



Řez, minimální řez II.

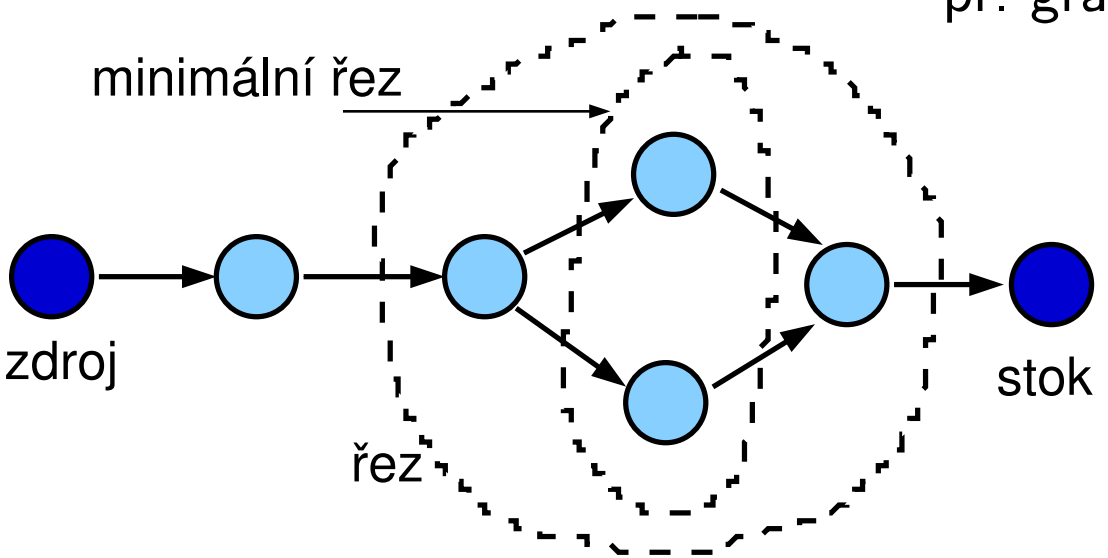
- Uvažujme orientovaný graf $G_0 = (V_0, E_0)$
- Do grafu přidáme zdroj:
  - nový vrchol s a
  - hrany S vedoucí z s do všech vrcholů G_0 bez předchůdců
- Do grafu přidáme stok:
  - nový vrchol t a
  - hrany T vedoucí ze všech vrcholů G_0 bez následníků do t
- Nový graf $G = (V, E)$: $V = V_0 \cup \{s, t\}, \quad E = E_0 \cup S \cup T$
- Budeme hledat řezy a minimální řezy z s do t v G



Řez, minimální řez II.

- Uvažujme orientovaný graf $G_0 = (V_0, E_0)$
- Do grafu přidáme zdroj:
  - nový vrchol s a
  - hrany S vedoucí z s do všech vrcholů G_0 bez předchůdců
- Do grafu přidáme stok:
  - nový vrchol t a
  - hrany T vedoucí ze všech vrcholů G_0 bez následníků do t
- Nový graf $G = (V, E)$: $V = V_0 \cup \{s, t\}, \quad E = E_0 \cup S \cup T$
- Budeme hledat řezy a minimální řezy z s do t v G

př. graf má 4 minimální řezy

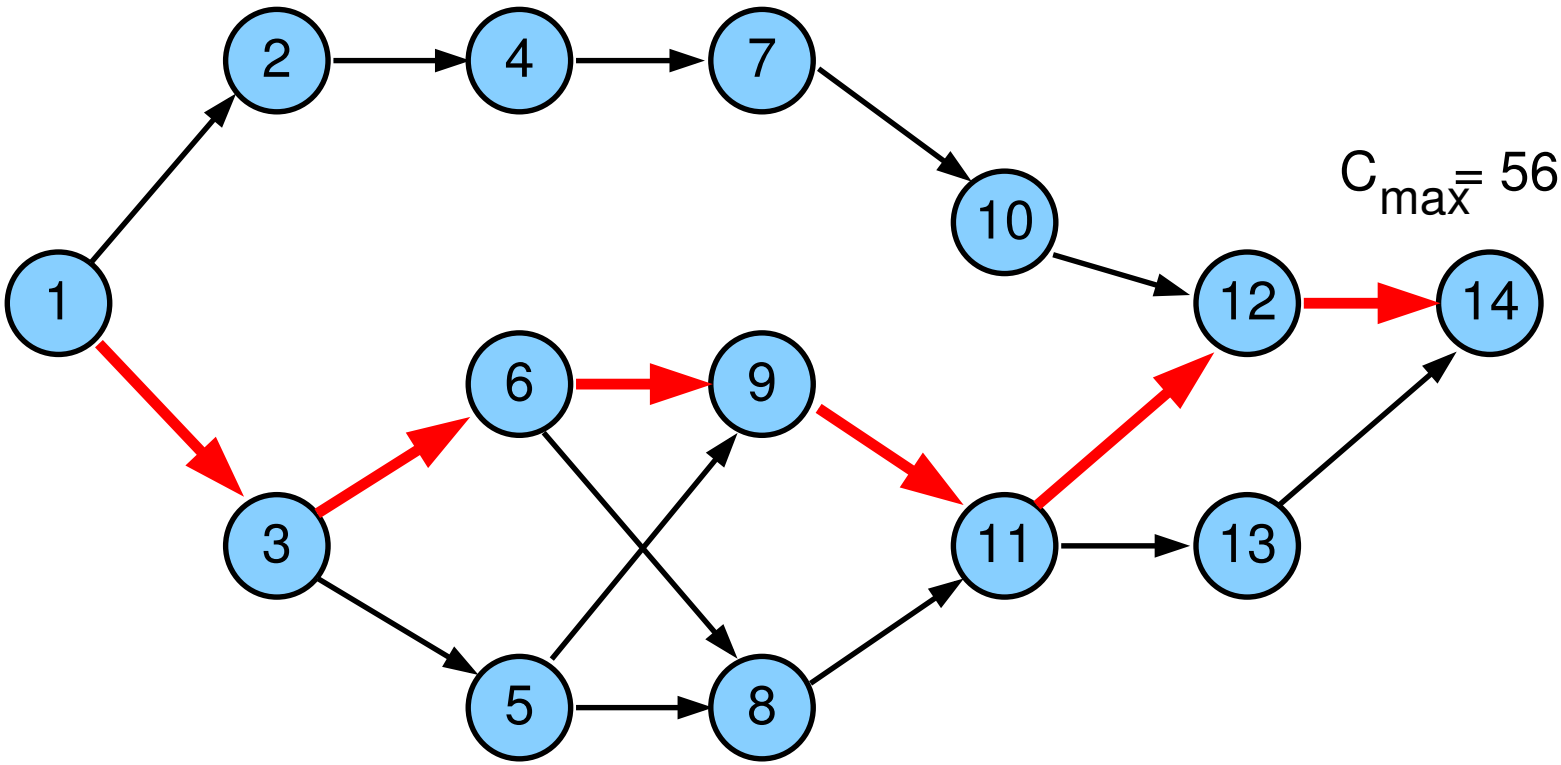


Kompromisní heuristika: příklad

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p_j^{max}	5	6	9	12	7	12	10	6	10	9	7	8	7	5
p_j^{min}	3	5	7	9	5	9	8	3	7	5	4	5	5	2
c_j^b	20	25	20	15	30	40	35	25	30	20	25	35	20	10
c_j	7	2	4	3	4	3	4	4	4	5	2	2	4	8

fixní režijní náklady na časovou jednotku c_0=6

Příklad (pokračování): maximální doba provádění



Kompromisní heuristika: příklad

c_0=6

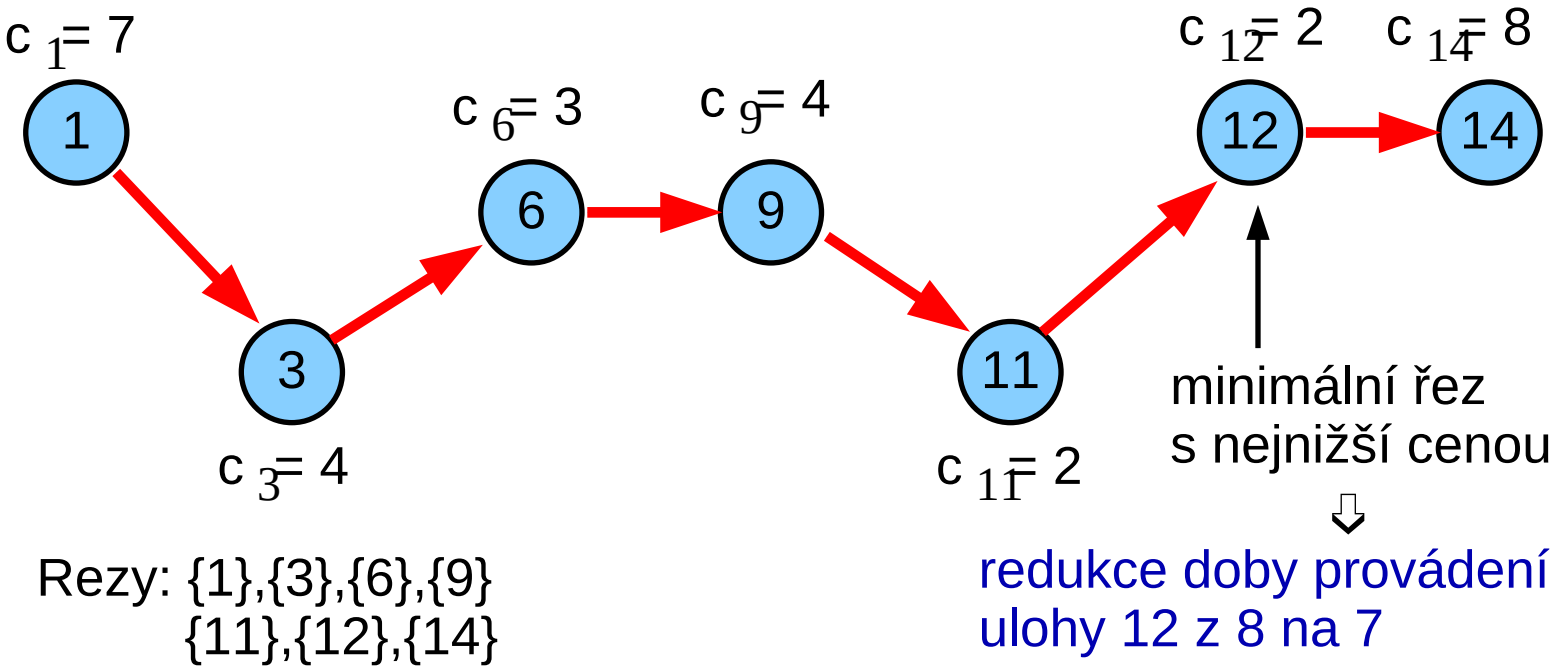
j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
$\mathbf{p_j^{max}}$	5	6	9	12	7	12	10	6	10	9	7	8	7	5
$\mathbf{p_j^{min}}$	3	5	7	9	5	9	8	3	7	5	4	5	5	2
$\mathbf{c_j^b}$	20	25	20	15	30	40	35	25	30	20	25	35	20	10
$\mathbf{c_j}$	7	2	4	3	4	3	4	4	4	5	2	2	4	8

Náklady na provedení projektu při maximální době trvání úloh

$$
\begin{aligned}
F(p_j^{max}) &= C_{\max} c_0 + \sum_j \left(c_j^b + c_j (p_j^{max} - p_j^{max}) \right) = \\
&= C_{\max} c_0 + \sum_j c_j^b = \\
&= 56 \times 6 + 20 + 25 + 20 + 15 + 30 + 40 + 35 + 25 + \\
&\quad + 30 + 20 + 25 + 35 + 20 + 10 = \\
&= 336 + 350 = 686
\end{aligned}
$$

Podgraf s kritickou cestou (G_{CP})

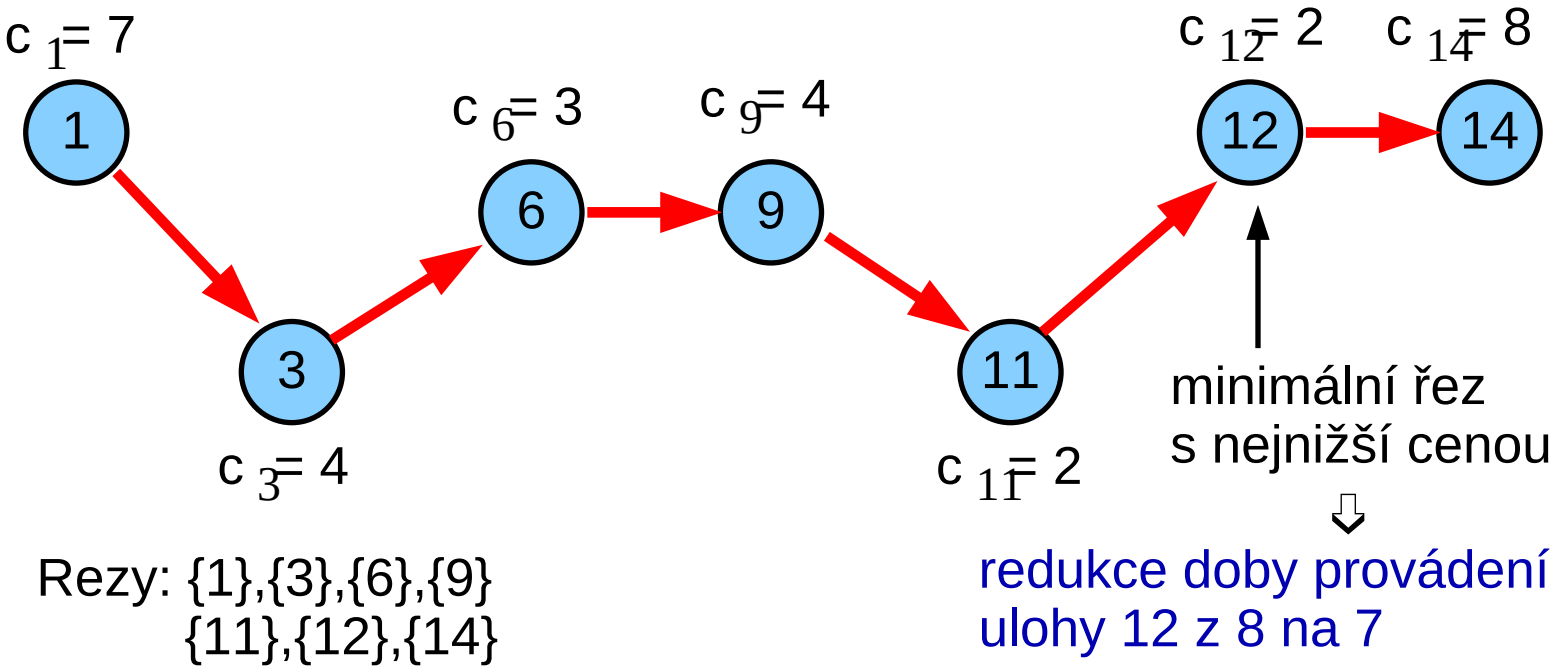
- Kandidáti na redukci: uzel 11 a uzel 12, vybereme uzel 12



- Fixní režijní náklady se redukují z 56×6 na

Podgraf s kritickou cestou (G_{CP})

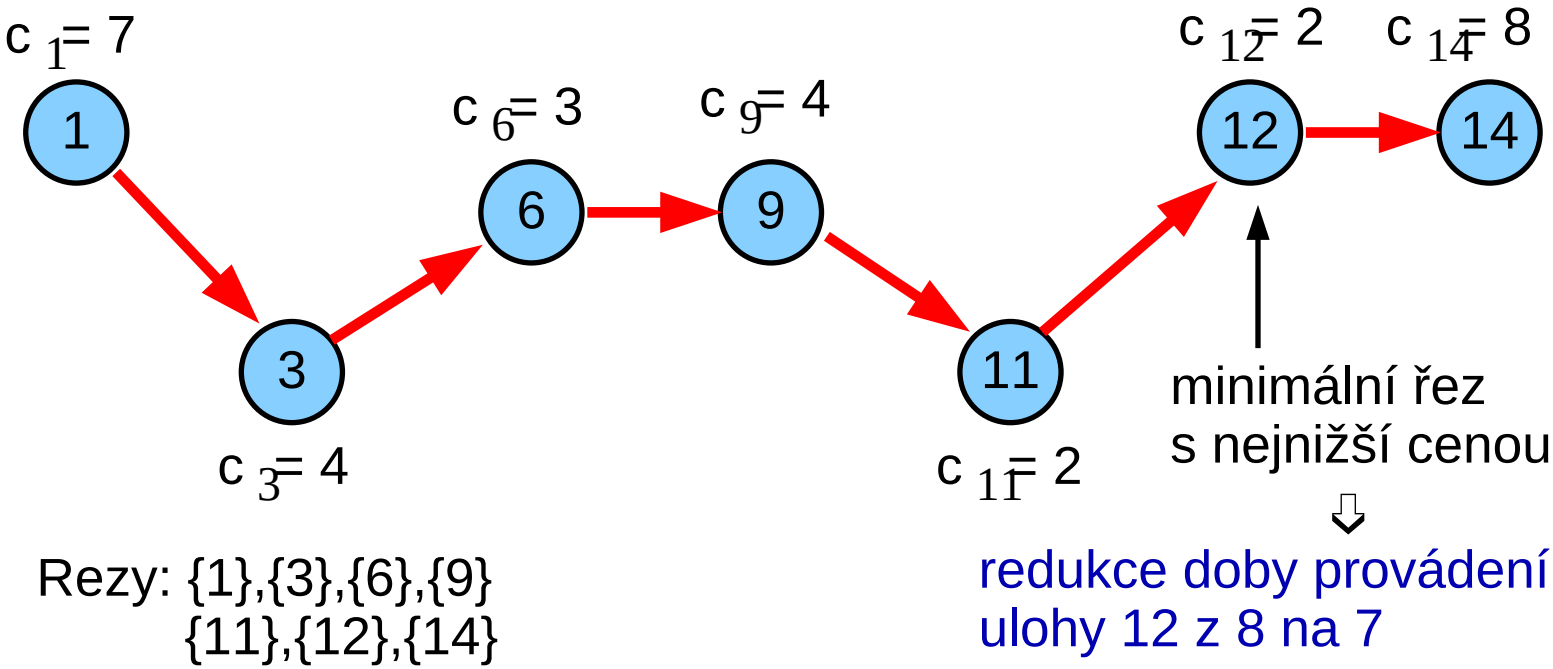
- Kandidáti na redukci: uzel 11 a uzel 12, vybereme uzel 12



- Fixní režijní náklady se redukují z 56×6 na $55 \times 6 = 330$

Podgraf s kritickou cestou (G_{CP})

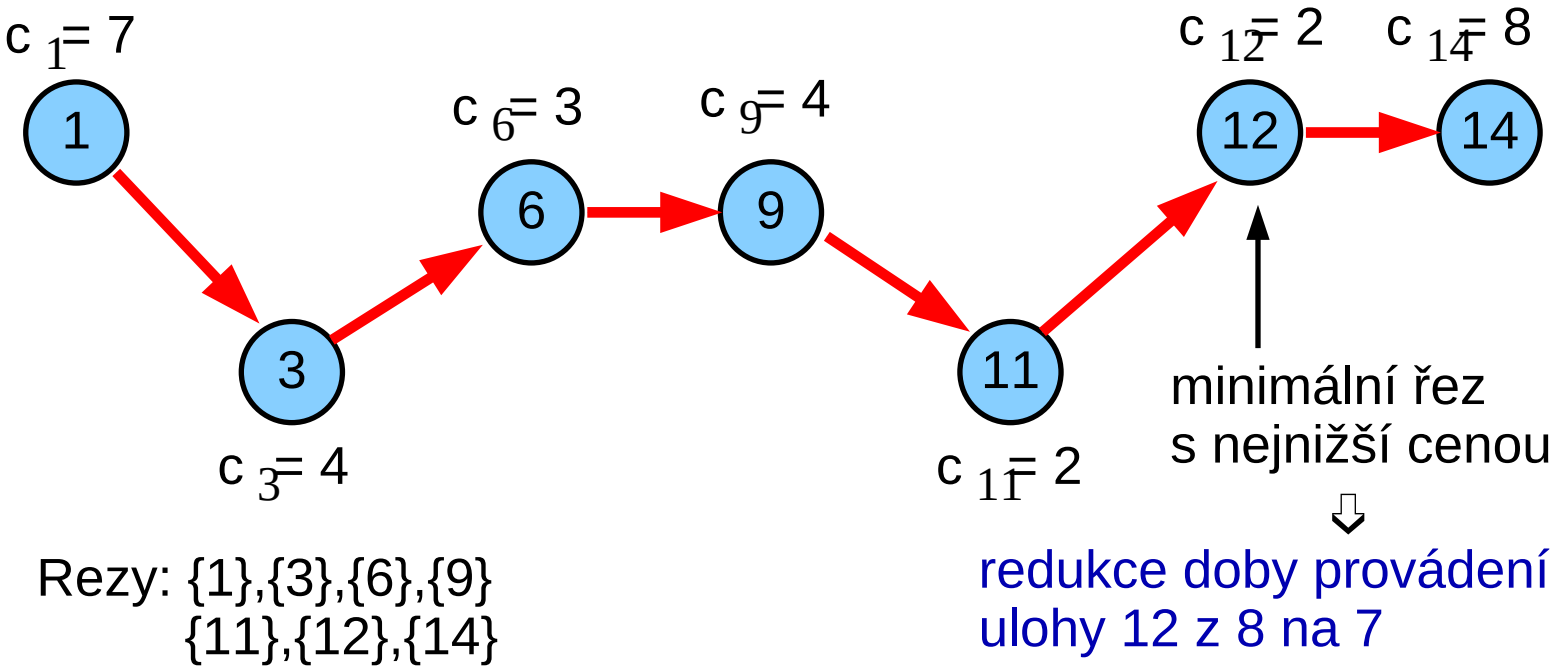
- Kandidáti na redukci: uzel 11 a uzel 12, vybereme uzel 12



- Fixní režijní náklady se redukují z 56×6 na $55 \times 6 = 330$
- Cena za provádění úloh

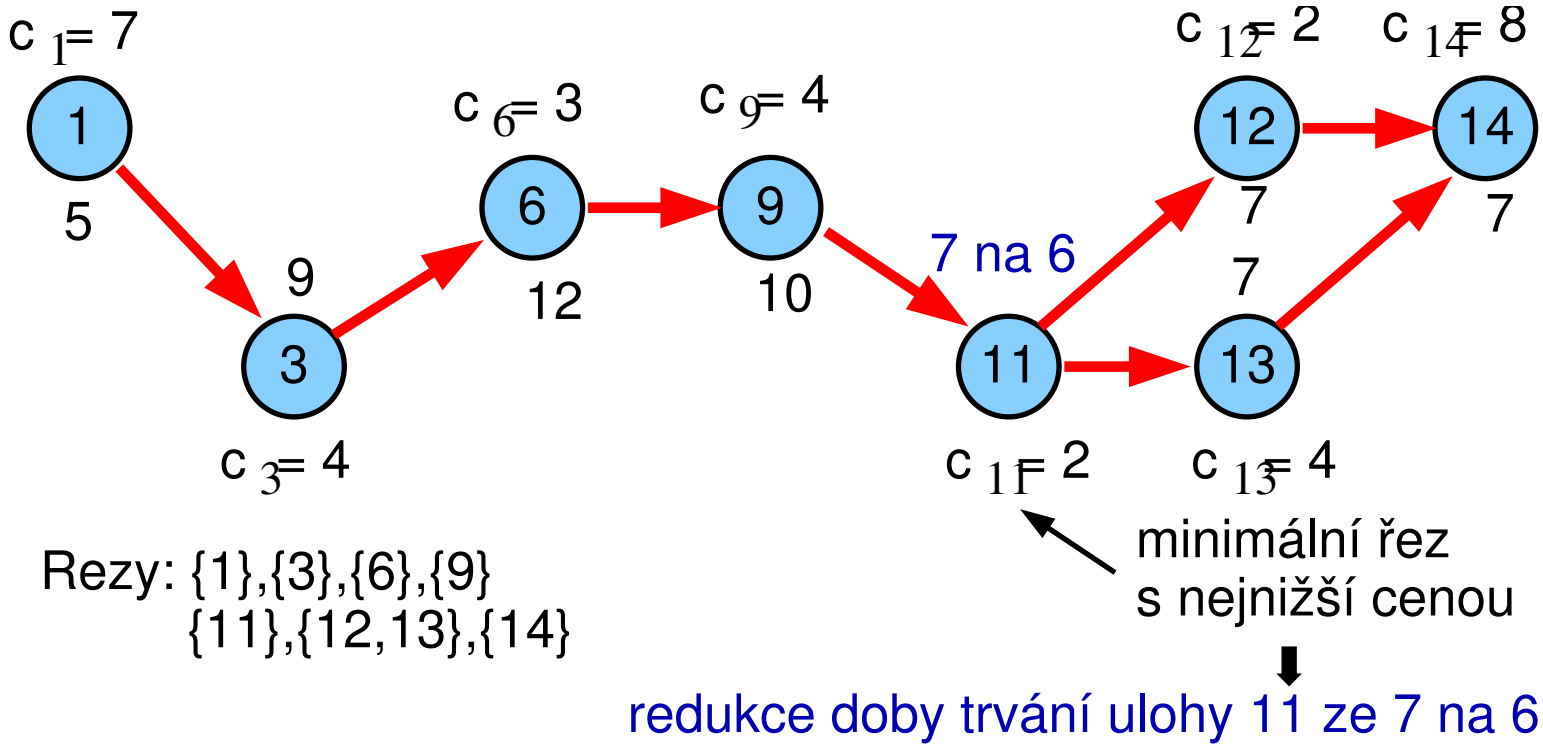
Podgraf s kritickou cestou (G_{CP})

- Kandidáti na redukci: uzel 11 a uzel 12, vybereme uzel 12



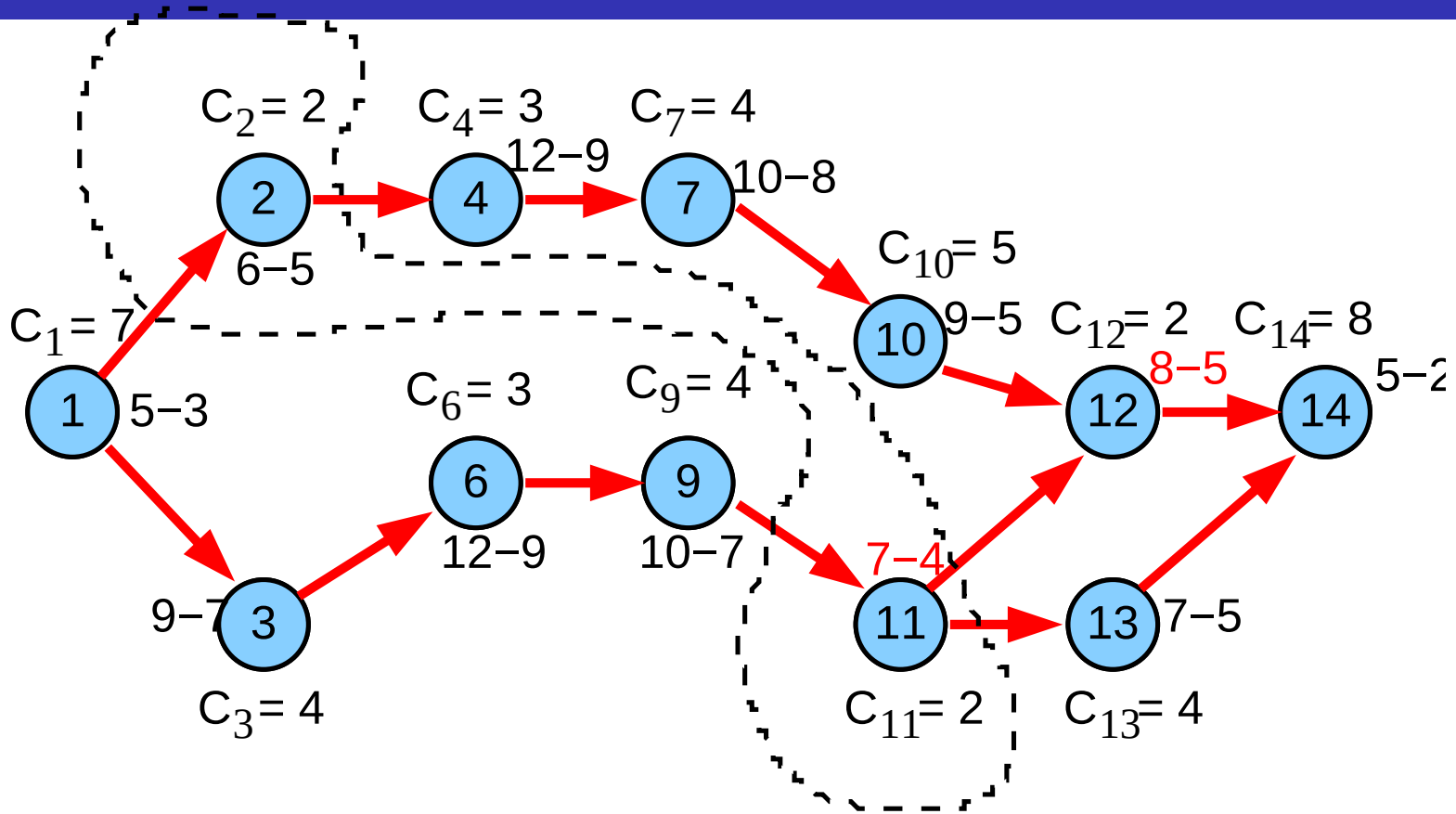
- Fixní režijní náklady se redukují z 56×6 na $55 \times 6 = 330$
- Cena za provádění úloh naroste o $c_{12} = 2$, tj. $350 + 2 = 352$
- Celková cena klesla z 686 na $330 + 352 = 682$

Podgraf s kritickou cestou (G_{CP})



- Fixní režijní náklady se redukují z 55×6 na $54 \times 6 = 324$
- Cena za provádění úloh naroste o $c_{11} = 2$, tj. $352 + 2 = 354$
- Celková cena klesla z 682 na $324 + 354 = 678$

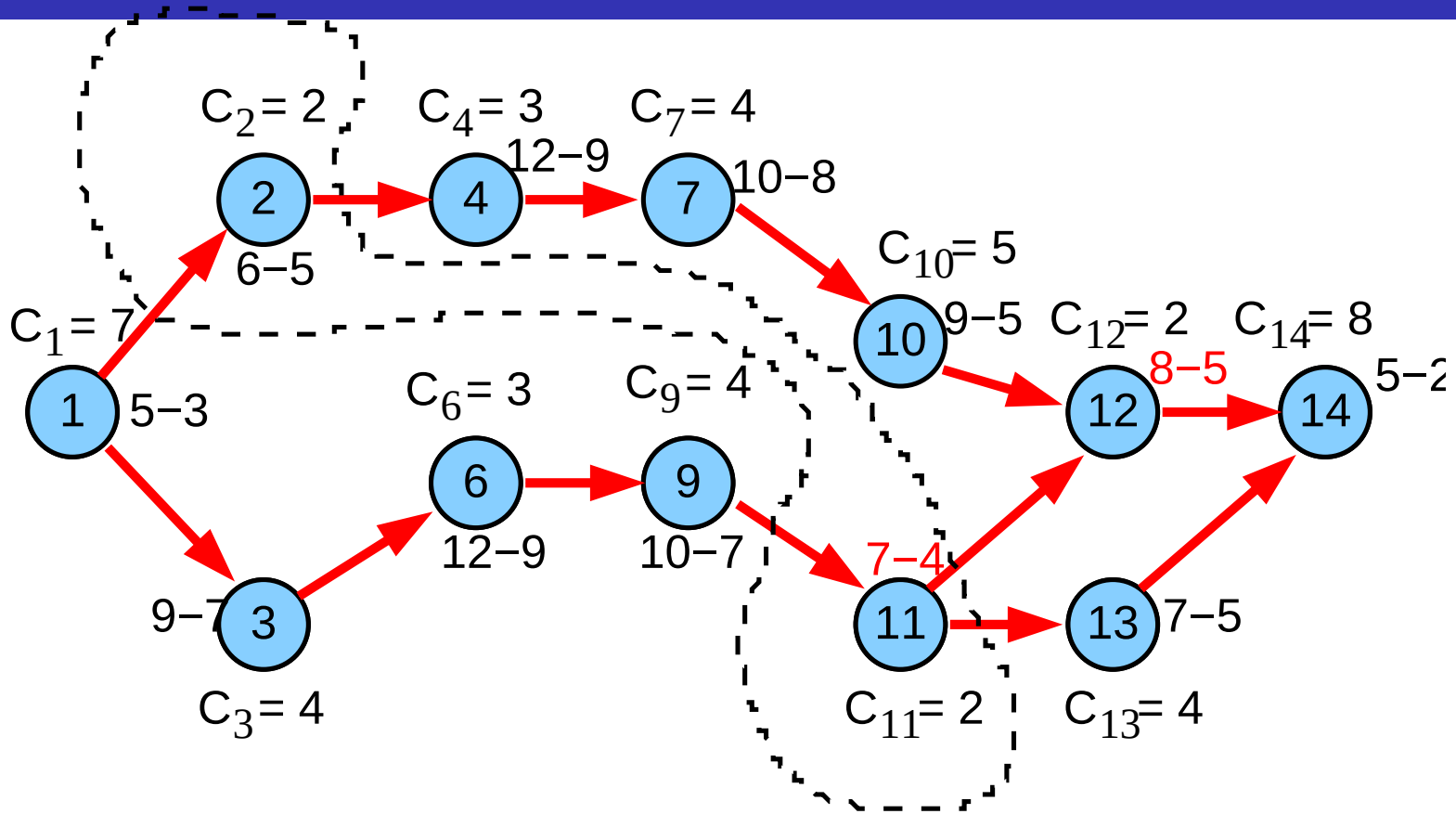
Podgraf s kritickou cestou (G_{CP})



další redukce: pro uzel 2 na 5 a pro uzel 11 na 5, ...

- Fixní režijní náklady se redukují z 54×6 na $53 \times 6 = 318$
- Cena za provádění úloh naroste o

Podgraf s kritickou cestou (G_{CP})



další redukce: pro uzel 2 na 5 a pro uzel 11 na 5, ...

- Fixní režijní náklady se redukují z 54×6 na $53 \times 6 = 318$
- Cena za provádění úloh naroste o $c_2 + c_{11} = 2 + 2$, tj. $354 + 4 = 358$
- Celková cena klesla z 678 na $318 + 358 = 676$

Algoritmus kompromisní heuristiky

1.
   - Nastav doby provádění na jejich maximum: $p_j = p_j^{max}$
   - Urči všechny kritické cesty s těmito dobami provádění
   - Zkonstruuj graf G_{CP} kritických cest

Algoritmus kompromisní heuristiky

1.
   - Nastav doby provádění na jejich maximum: $p_j = p_j^{max}$
   - Urči všechny kritické cesty s těmito dobami provádění
   - Zkonstruuj graf G_{CP} kritických cest
2.
   - Urči všechny minimální řezy v G_{CP}
     - pozn. pokud zkrátíme dobu provádění všech úloh v minimálním řezu, podaří se nám i zkrátit dobu provádění projektu!
   - Najdi řezy, jejichž doba provádění je větší než jejich minimum: $p_j > p_j^{min} \quad \forall j \in G_{CP}$
   - Pokud takový řez neexistuje STOP, jinak běž na krok 3

Algoritmus kompromisní heuristiky

1. Nastav doby provádění na jejich maximum: $p_j = p_j^{max}$
   - Urči všechny kritické cesty s těmito dobami provádění
   - Zkonstruuj graf G_{CP} kritických cest
2. Urči všechny minimální řezy v G_{CP}
   - pozn. pokud zkrátíme dobu provádění všech úloh v minimálním řezu, podaří se nám i zkrátit dobu provádění projektu!
   - Najdi řezy, jejichž doba provádění je větší než jejich minimum: $p_j > p_j^{min} \quad \forall j \in G_{CP}$
   - Pokud takový řez neexistuje STOP, jinak běž na krok 3
3. Pro každý minimální řez: spočítej cenu redukující všechny doby provádění o 1 časovou jednotku
   - Vyber minimální řez s nejnižší cenou
     - pozn. abychom za snížení zaplatili co nejnižší cenu
   - Jestliže je cena řezu menší než fixní režijní náklady c_0 za časovou jednotku běž na krok 4, jinak STOP

Algoritmus kompromisní heuristiky

1. Nastav doby provádění na jejich maximum: $p_j = p_j^{max}$
   - Urči všechny kritické cesty s těmito dobami provádění
   - Zkonstruuj graf G_{CP} kritických cest
2. Urči všechny minimální řezy v G_{CP}
   - pozn. pokud zkrátíme dobu provádění všech úloh v minimálním řezu, podaří se nám i zkrátit dobu provádění projektu!
   - Najdi řezy, jejichž doba provádění je větší než jejich minimum: $p_j > p_j^{min} \quad \forall j \in G_{CP}$
   - Pokud takový řez neexistuje STOP, jinak běž na krok 3
3. Pro každý minimální řez: spočítej cenu redukující všechny doby provádění o 1 časovou jednotku
   - Vyber minimální řez s nejnižší cenou
     - pozn. abychom za snížení zaplatili co nejnižší cenu
   - Jestliže je cena řezu menší než fixní režijní náklady c_0 za časovou jednotku běž na krok 4, jinak STOP
4. Redukuj všechny doby provádění v minimálním řezu o 1 časovou jednotku
   - Urči novou množinu kritických cest
   - Reviduj graf G_{CP} a běž na krok 2

Lineární program

- Heuristika negarantuje nalezení optima
- Celková cena je lineární

$$c_0 C_{max} + \sum_{j=1}^{n} \left(c_j^b + c_j (p_j^{max} - p_j) \right)$$

všimněte si: stejná účelová funkce jako cena za provádění projektu

- Lineární program:

  minimalizace: $c_0 C_{max} - \sum_{j=1}^{n} c_j p_j$

  c_j^b a $c_j\ p_j^{max}$ se nemění

Lineární program

- Heuristika negarantuje nalezení optima
- Celková cena je lineární

$$c_0 C_{max} + \sum_{j=1}^{n} \left(c_j^b + c_j (p_j^{max} - p_j) \right)$$

všimněte si: stejná účelová funkce jako cena za provádění projektu

- Lineární program:

  c_j^b a $c_j\, p_j^{max}$ se nemění

  minimalizace: $c_0 C_{max} - \sum_{j=1}^{n} c_j p_j$

  za předpokladu:

$$\begin{aligned} x_k - p_j - x_j &\geq 0 && \forall j \in Prec_k \\ p_j &\leq p_j^{max} && \forall j \\ p_j &\geq p_j^{min} && \forall j \\ x_j &\geq 0 && \forall j \\ C_{max} - x_j - p_j &\geq 0 && \forall j \end{aligned}$$

  kde x_j je nejdřívější možný startovní čas úlohy j

Přidání pracovní síly

- Pracovní síla = operátor = zdroj
- Problém popsán v literatuře jako
  problém plánování projektu s omezenými zdroji
  resource-constrained project scheduling problem (RCPSP)
  - n úloh
  - N zdrojů
  - R_i kapacita zdroje i
  - p_j doba provádění úlohy j
  - R_{ij} požadavek úlohy j na zdroj i
  - $Prec_j$ (přímí) předchůdci úlohy j

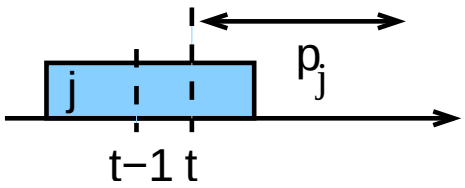
Formulace celočíselného programování

- Pomocná úloha $n+1$ jako stok, $p_{n+1} = 0$
- $x_{jt} = 1$ úloha j je dokončena v čase t
  $x_{jt} = 0$ jinak

Formulace celočíselného programování

- Pomocná úloha $n+1$ jako stok, $p_{n+1} = 0$
- $x_{jt} = 1$ úloha j je dokončena v čase t
  $x_{jt} = 0$ jinak
- Kapacita zdroje i, který potřebuje úloha j

  v intervalu $[t-1, t]$: $R_{ij} \sum_{u=t}^{t+p_j-1} x_{ju}$



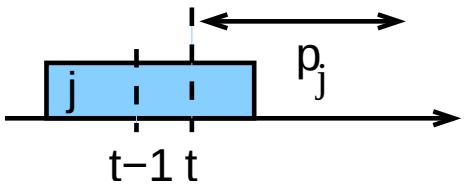
$\sum_{u=t}^{t+p_j-1} x_{ju}$ ... počítá, zda úloha j běží v čase $[t-1, t]$

př. úloha s $S_j = 2, p_j = 2$ a $t = 2, 3, 4, 5$

Formulace celočíselného programování

- Pomocná úloha $n+1$ jako stok, $p_{n+1} = 0$
- $x_{jt} = 1$ úloha j je dokončena v čase t
  $x_{jt} = 0$ jinak
- Kapacita zdroje i, který potřebuje úloha j

  v intervalu $[t-1, t]$: $R_{ij} \sum_{u=t}^{t+p_j-1} x_{ju}$



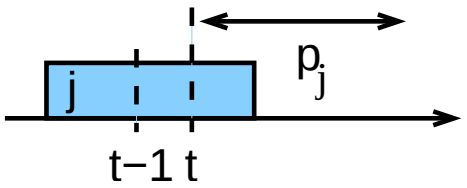
$\sum_{u=t}^{t+p_j-1} x_{ju}$ ... počítá, zda úloha j běží v čase $[t-1, t]$

př. úloha s $S_j = 2, p_j = 2$ a $t = 2, 3, 4, 5$ ($x_{j4} = 1$,

Formulace celočíselného programování

- Pomocná úloha $n+1$ jako stok, $p_{n+1} = 0$
- $x_{jt} = 1$ úloha j je dokončena v čase t
  $x_{jt} = 0$ jinak
- Kapacita zdroje i, který potřebuje úloha j

  v intervalu $[t-1, t]$: $R_{ij} \sum_{u=t}^{t+p_j-1} x_{ju}$



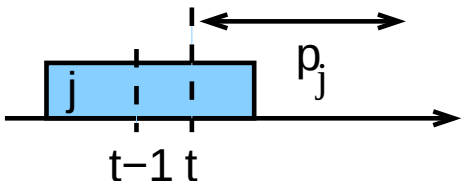
$\sum_{u=t}^{t+p_j-1} x_{ju}$ ... počítá, zda úloha j běží v čase $[t-1, t]$

př. úloha s $S_j = 2, p_j = 2$ a $t = 2, 3, 4, 5$ ($x_{j4} = 1$, pro $t = 3, 4$ úloha započítána)

Formulace celočíselného programování

- Pomocná úloha $n+1$ jako stok, $p_{n+1} = 0$
- $x_{jt} = 1$ úloha j je dokončena v čase t
  $x_{jt} = 0$ jinak
- Kapacita zdroje i, který potřebuje úloha j

  v intervalu $[t-1, t]$: $R_{ij} \sum_{u=t}^{t+p_j-1} x_{ju}$



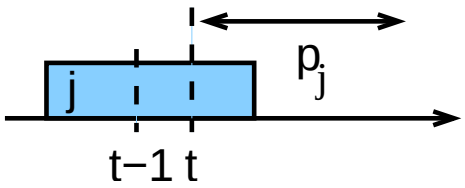
$\sum_{u=t}^{t+p_j-1} x_{ju}$ ... počítá, zda úloha j běží v čase $[t-1, t]$

př. úloha s $S_j = 2, p_j = 2$ a $t = 2, 3, 4, 5$ ($x_{j4} = 1$, pro $t = 3, 4$ úloha započítána)

- H jako horní hranice *makespan*: $H = \sum_{j=1}^{n} p_j$

Formulace celočíselného programování

- Pomocná úloha $n+1$ jako stok, $p_{n+1} = 0$
- $x_{jt} = 1$ úloha j je dokončena v čase t
  $x_{jt} = 0$ jinak
- Kapacita zdroje i, který potřebuje úloha j
  v intervalu $[t-1, t]$: $R_{ij} \sum_{u=t}^{t+p_j-1} x_{ju}$



$\sum_{u=t}^{t+p_j-1} x_{ju}$ ... počítá, zda úloha j běží v čase $[t-1, t]$

př. úloha s $S_j = 2, p_j = 2$ a $t = 2, 3, 4, 5$ ($x_{j4} = 1$, pro $t = 3, 4$ úloha započítána)

- H jako horní hranice *makespan*: $H = \sum_{j=1}^{n} p_j$
- Koncový čas úlohy j: $C_j = \sum_{t=1}^{H} t\, x_{jt}$
- *Makespan*: $C_{max} = \sum_{t=1}^{H} t\, x_{n+1,t}$ koncový čas stoku

Celočíselný program

Minimalizace $\sum_{t=1}^{H} t \, x_{n+1,t}$ minimalizace *makespan*

Celočíselný program

Minimalizace $\sum_{t=1}^{H} t\, x_{n+1,t}$ minimalizace *makespan*

za předpokladu

jestliže j je předchůdce k, pak $C_j \leq S_k$, tj. $C_j \leq C_k - p_k$, tj. $C_j + p_k - C_k \leq 0$

$$\sum_{t=1}^{H} t\, x_{jt} + p_k - \sum_{t=1}^{H} t\, x_{kt} \leq 0 \qquad \forall j \in Prec_k$$

Celočíselný program

Minimalizace $\sum_{t=1}^{H} t\, x_{n+1,t}$ minimalizace *makespan*

za předpokladu

jestliže j je předchůdce k, pak $C_j \leq S_k$, tj. $C_j \leq C_k - p_k$, tj. $C_j + p_k - C_k \leq 0$

$$\sum_{t=1}^{H} t\, x_{jt} + p_k - \sum_{t=1}^{H} t\, x_{kt} \leq 0 \qquad \forall j \in Prec_k$$

pro každý čas t: požadavek na zdroj i nepřeroste kapacitu i

$$\sum_{j=1}^{n} \left(R_{ij} \sum_{u=t}^{t+p_j-1} x_{ju} \right) \leq R_i \qquad \forall i\, \forall t$$

Celočíselný program

Minimalizace $\sum_{t=1}^{H} t\, x_{n+1,t}$ minimalizace *makespan*

za předpokladu

jestliže j je předchůdce k, pak $C_j \leq S_k$, tj. $C_j \leq C_k - p_k$, tj. $C_j + p_k - C_k \leq 0$

$$\sum_{t=1}^{H} t\, x_{jt} + p_k - \sum_{t=1}^{H} t\, x_{kt} \leq 0 \qquad \forall j \in Prec_k$$

pro každý čas t: požadavek na zdroj i nepřeroste kapacitu i

$$\sum_{j=1}^{n} \left(R_{ij} \sum_{u=t}^{t+p_j-1} x_{ju} \right) \leq R_i \qquad \forall i\, \forall t$$

každá úloha j skončí právě jednou

$$\sum_{t=1}^{H} x_{jt} = 1 \qquad \forall j$$

Diskuse

- Řešení celočíselného programu obtížné
- Při velkém počtu úloh a dlouhém časovém horizontu
  - použití heuristik
- Lze použít programování s omezujícími podmínkami
  - kumulativní zdroje
  - precedenční podmínky
- Probírané speciální případy problému
  - *job shop + makespan*
  - *timetabling*

Plánování projektu: shrnutí

- Základní problém s neomezenými zdroji
  - metoda kritické cesty
- Neomezené zdroje + variabilní doba trvání (lineární)
  - kompromisní heuristika mezi časem a cenou
  - lineární programování
- Problém plánování projektu s omezenými zdroji
  - celočíselné programování
  - heuristiky
  - programování s omezujícími podmínkami

Plánování úloh (na jednom stroji)

16. května 2022

Jeden stroj a paralelní stroj

- Dekompoziční problémy pro složité *(flexible) job shop* problémy používají
  - jeden stroj
  - paralelní stroj

  jako podproblémy při řešení
- Metody řešení:
  - řídící pravidla
  - metoda větví & mezí
  - paprskové prohledávání

Řídící pravidla pro jeden stroj: přehled

- C_{max} a $r_j = 0, d_j = \infty$ (snadné: nezávisí na rozvrhu)
- $\sum w_j C_j$ a $r_j = 0, d_j = \infty$
  - vážená nejkratší doba provádění (WSPT) je optimální
    - rozvrhuje úlohy v klesajícím pořadí podle w_j/p_j

Řídící pravidla pro jeden stroj: přehled

- C_{max} a $r_j = 0, d_j = \infty$ (snadné: nezávisí na rozvrhu)
- $\sum w_j C_j$ a $r_j = 0, d_j = \infty$
  - vážená nejkratší doba provádění (WSPT) je optimální
    - rozvrhuje úlohy v klesajícím pořadí podle w_j/p_j
- L_{max} a $r_j = 0$
  - nejdřívější termín dokončení (EDD) je optimální
    - rozvrhuje úlohy ve vzrůstající velikosti d_j
  - minimální rezerva (MS)
    - pravidlo příbuzné EDD ale dynamické
    - $\max(d_j - p_j - t, 0)$, kde je t aktuální čas

Řídící pravidla pro jeden stroj: přehled

- C_{max} a $r_j = 0, d_j = \infty$ (snadné: nezávisí na rozvrhu)
- $\sum w_j C_j$ a $r_j = 0, d_j = \infty$
  - vážená nejkratší doba provádění (WSPT) je optimální
    - rozvrhuje úlohy v klesajícím pořadí podle w_j/p_j
- L_{max} a $r_j = 0$
  - nejdřívější termín dokončení (EDD) je optimální
    - rozvrhuje úlohy ve vzrůstající velikosti d_j
  - minimální rezerva (MS)
    - pravidlo příbuzné EDD ale dynamické
    - $\max(d_j - p_j - t, 0)$, kde je t aktuální čas
- L_{max} a rozdílné r_j
  - NP-těžký problém
  - základní podproblém v rámci známé heuristiky posouvání kritického místa
  - řešen pomocí metody větví a mezí nebo dynamického programování

Řídící pravidla pro jeden stroj: přehled

- C_{max} a $r_j = 0, d_j = \infty$ (snadné: nezávisí na rozvrhu)
- $\sum w_j C_j$ a $r_j = 0, d_j = \infty$
  - vážená nejkratší doba provádění (WSPT) je optimální
    - rozvrhuje úlohy v klesajícím pořadí podle w_j/p_j
- L_{max} a $r_j = 0$
  - nejdřívější termín dokončení (EDD) je optimální
    - rozvrhuje úlohy ve vzrůstající velikosti d_j
  - minimální rezerva (MS)
    - pravidlo příbuzné EDD ale dynamické
    - $\max(d_j - p_j - t, 0)$, kde je t aktuální čas
- L_{max} a rozdílné r_j
  - NP-těžký problém
  - základní podproblém v rámci známé heuristiky posouvání kritického místa
  - řešen pomocí metody větví a mezí nebo dynamického programování
- $\sum T_j$, $\sum w_j T_j$
  - mnohem obtížnější na optimalizaci
  - kompozitní řídící pravidlo *apparent tardiness cost* (ATC) využívající kombinaci WSPT a MS

Řídící pravidla a paralelní stroj: přehled

- C_{max}: důležité kritérium při balancování zátěže strojů
  - NP-těžký
  - nejdelší doba provádění (LPT)
    - kratší úlohy odloženy, protože je snadnější je narozvrhovat
    - nalezne řešení s garancí rozsahu 33% optima

Řídící pravidla a paralelní stroj: přehled

- C_{max}: důležité kritérium při balancování zátěže strojů
  - NP-těžký
  - nejdelší doba provádění (LPT)
    - kratší úlohy odloženy, protože je snadnější je narozvrhovat
    - nalezne řešení s garancí rozsahu 33% optima
- $\sum C_j$ a $r_j = 0$
  - nepreemptivní nejkratší doba provádění (SPT)
    - nepreemptivní SPT minimalizuje $\sum C_j$
    - nepreemptivní SPT zůstává optimální i při povolených přerušeních

Řídící pravidla a paralelní stroj: přehled

- C_{max}: důležité kritérium při balancování zátěže strojů
  - NP-těžký
  - nejdelší doba provádění (LPT)
    - kratší úlohy odloženy, protože je snadnější je narozvrhovat
    - nalezne řešení s garancí rozsahu 33% optima
- $\sum C_j$ a $r_j = 0$
  - nepreemptivní nejkratší doba provádění (SPT)
    - nepreemptivní SPT minimalizuje $\sum C_j$
    - nepreemptivní SPT zůstává optimální i při povolených přerušeních
- $\sum w_j C_j$ a $r_j = 0$
  - NP-těžký
  - WSPT grarantuje řešení v rámci 22% optima

Řídící pravidla a paralelní stroj: přehled

- C_{max}: důležité kritérium při balancování zátěže strojů
  - NP-těžký
  - nejdelší doba provádění (LPT)
    - kratší úlohy odloženy, protože je snadnější je narozvrhovat
    - nalezne řešení s garancí rozsahu 33% optima
- $\sum C_j$ a $r_j = 0$
  - nepreemptivní nejkratší doba provádění (SPT)
    - nepreemptivní SPT minimalizuje $\sum C_j$
    - nepreemptivní SPT zůstává optimální i při povolených přerušeních
- $\sum w_j C_j$ a $r_j = 0$
  - NP-těžký
  - WSPT grarantuje řešení v rámci 22% optima
- $\sum w_j T_j$
  - ještě obtížnější problém
  - lze použít ATC (apparent tardiness cost), ale kvalita řešení nemusí být dobrá

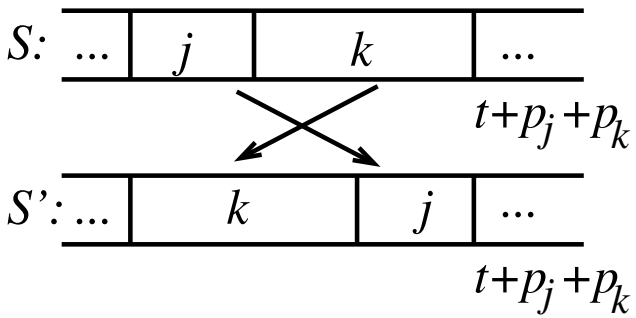
Vážený součet koncových časů pro jeden stroj

- Řídící pravidlo vážená nejkratší doba trvání
  (weighted shortest processing time, WSPT)
  je optimální pro $1||\sum w_j C_j$.
  - WSPT rozvrhuje úlohy v klesajícím pořadí podle w_j/p_j
- Důkaz:
  - předpokládejme, že to není pravda a že S je optimální
  - pak existují dvě sousední úlohy, např. úloha j následovaná úlohou k takové, že $\frac{w_j}{p_j} < \frac{w_k}{p_k}$, tj. $\mathbf{p_k w_j < p_j w_k}$

S' :

Vážený součet koncových časů pro jeden stroj

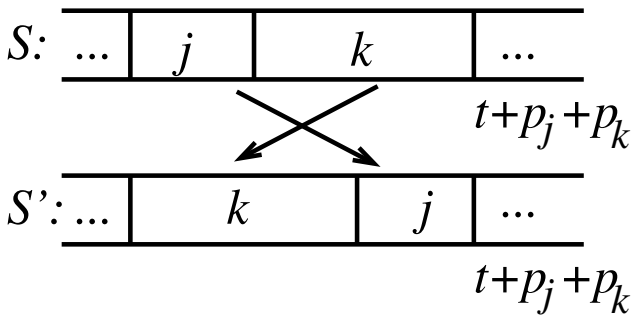
- Řídící pravidlo vážená nejkratší doba trvání
  (weighted shortest processing time, WSPT)
  je optimální pro $1||\sum w_j C_j$.
  - WSPT rozvrhuje úlohy v klesajícím pořadí podle w_j/p_j
- Důkaz:
  - předpokládejme, že to není pravda a že S je optimální
  - pak existují dvě sousední úlohy, např. úloha j následovaná úlohou k takové, že $\frac{w_j}{p_j} < \frac{w_k}{p_k}$, tj. $\mathbf{p_k w_j < p_j w_k}$
  - po provedení párové výměny máme rozvrh S'



  - Příspěvky do účelové funkce pro j a k:
    S :
    S' :

Vážený součet koncových časů pro jeden stroj

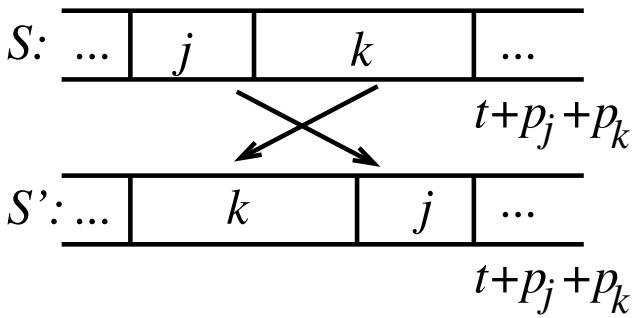
- Řídící pravidlo vážená nejkratší doba trvání
  (weighted shortest processing time, WSPT)
  je optimální pro $1||\sum w_j C_j$.
  - WSPT rozvrhuje úlohy v klesajícím pořadí podle w_j/p_j
- Důkaz:
  - předpokládejme, že to není pravda a že S je optimální
  - pak existují dvě sousední úlohy, např. úloha j následovaná úlohou k takové, že $\frac{w_j}{p_j} < \frac{w_k}{p_k}$, tj. $\mathbf{p_k w_j < p_j w_k}$
  - po provedení párové výměny máme rozvrh S'



  - Příspěvky do účelové funkce pro j a k:
    $S : (t + p_j)w_j + (t + p_j + p_k)w_k = tw_j + p_j w_j + tw_k + \mathbf{p_j w_k} + p_k w_k$
    S' :

Vážený součet koncových časů pro jeden stroj

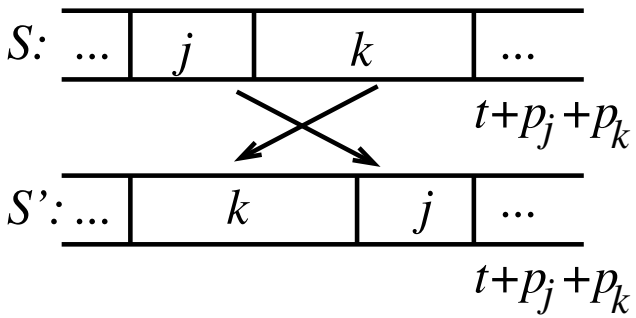
- Řídící pravidlo vážená nejkratší doba trvání
  (weighted shortest processing time, WSPT)
  je optimální pro $1||\sum w_j C_j$.
  - WSPT rozvrhuje úlohy v klesajícím pořadí podle w_j/p_j
- Důkaz:
  - předpokládejme, že to není pravda a že S je optimální
  - pak existují dvě sousední úlohy, např. úloha j následovaná úlohou k takové, že $\frac{w_j}{p_j} < \frac{w_k}{p_k}$, tj. $\mathbf{p_k w_j < p_j w_k}$
  - po provedení párové výměny máme rozvrh S'



  - Příspěvky do účelové funkce pro j a k:
    $S : (t + p_j)w_j + (t + p_j + p_k)w_k = tw_j + p_j w_j + tw_k + \mathbf{p_j w_k} + p_k w_k$
    $S' : (t + p_k)w_k + (t + p_k + p_j)w_j = tw_k + p_k w_k + tw_j + \mathbf{p_k w_j} + p_j w_j$

Vážený součet koncových časů pro jeden stroj

- Řídící pravidlo vážená nejkratší doba trvání
  (weighted shortest processing time, WSPT)
  je optimální pro $1||\sum w_j C_j$.
  - WSPT rozvrhuje úlohy v klesajícím pořadí podle w_j/p_j
- Důkaz:
  - předpokládejme, že to není pravda a že S je optimální
  - pak existují dvě sousední úlohy, např. úloha j následovaná úlohou k takové, že $\frac{w_j}{p_j} < \frac{w_k}{p_k}$, tj. $\mathbf{p_k w_j < p_j w_k}$
  - po provedení párové výměny máme rozvrh S'



  - Příspěvky do účelové funkce pro j a k:
    $S : (t + p_j)w_j + (t + p_j + p_k)w_k = tw_j + p_j w_j + tw_k + \mathbf{p_j w_k} + p_k w_k$
    $S' : (t + p_k)w_k + (t + p_k + p_j)w_j = tw_k + p_k w_k + tw_j + \mathbf{p_k w_j} + p_j w_j$
  - $\sum w_j C_j$ pro $S' < \sum w_j C_j$ pro S spor s optimalitou S

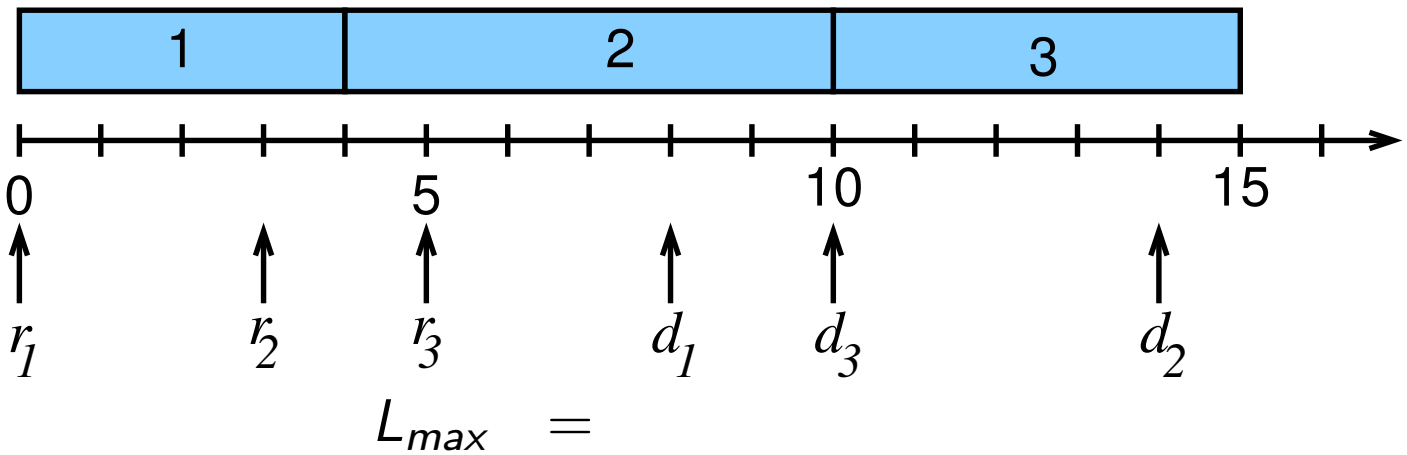
Zpoždění pro jeden stroj

- EDD optimální pro $1||L_{max}$
- Rozdílné termíny dostupnosti r_j, tj. problém $1|r_j|L_{max}$: NP-úplný problém
- Proč je tento problém tak obtížný?

$1|r_j|L_{max}$

Úlohy	1	2	3
p_j	4	6	5
r_j	0	3	5
d_j	8	14	10

Rozvrh pomocí EDD:

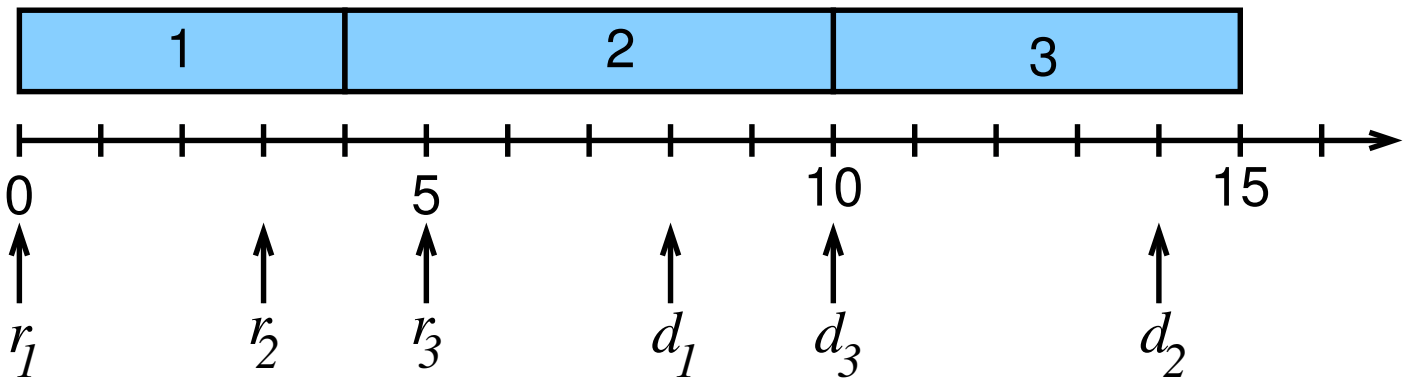


$L_{max} =$

$1|r_j|L_{max}$

Úlohy	1	2	3
p_j	4	6	5
r_j	0	3	5
d_j	8	14	10

Rozvrh pomocí EDD:

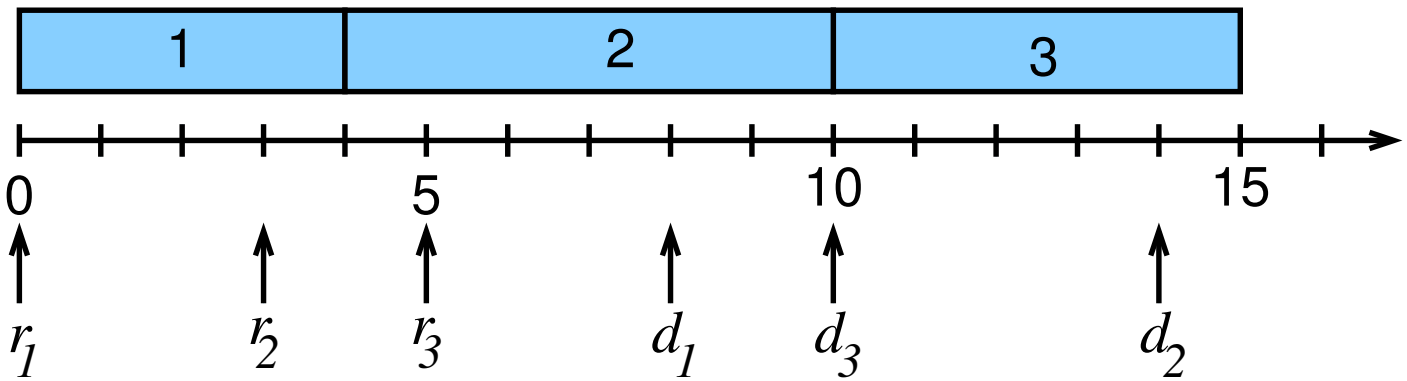


$$L_{max} = \max(L_1, L_2, L_3) =$$

$1|r_j|L_{max}$

Úlohy	1	2	3
p_j	4	6	5
r_j	0	3	5
d_j	8	14	10

Rozvrh pomocí EDD:

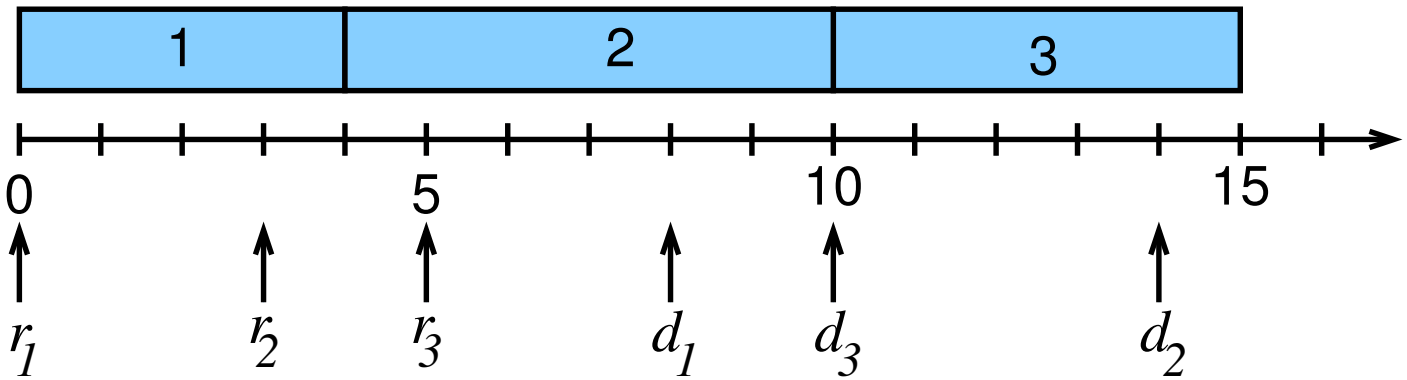


$$L_{max} = \max(L_1, L_2, L_3) =$$
$$= \max(C_1 - d_1, C_2 - d_2, C_3 - d_3) =$$

$1|r_j|L_{max}$

Úlohy	1	2	3
p_j	4	6	5
r_j	0	3	5
d_j	8	14	10

Rozvrh pomocí EDD:

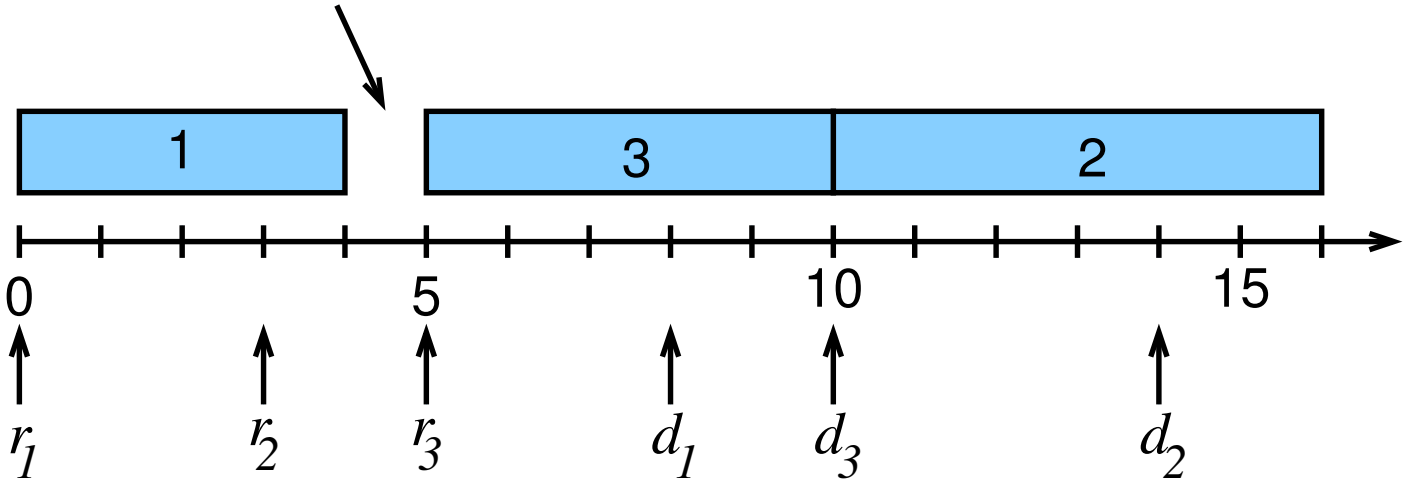


$$\begin{aligned} L_{max} &= \max(L_1, L_2, L_3) = \\ &= \max(C_1 - d_1, C_2 - d_2, C_3 - d_3) = \\ &= \max(4 - 8, 10 - 14, 15 - 10) = \\ &= \max(-4, -4, 5) = 5 \end{aligned}$$

Existuje lepší řešení?

Rozvrh se zdržením pro $1|r_j|L_{max}$

Pozdržíme provádění úloh:



$$\begin{aligned} L_{max} &= \max(L_1, L_2, L_3) = \\ &= \max(C_1 - d_1, C_2 - d_2, C_3 - d_3) = \\ &= \max(4 - 8, 16 - 14, 10 - 10) = \\ &= \max(-4, 2, 0) = 2 \end{aligned}$$

Problém je obtížný, protože

optimální rozvrh není nutně bez zdržení

$1|r_j, prmp|L_{max}$

- Preemptivní úlohy: je možné přerušit jejich provádění
- Preemptivní verze řídících pravidel:
  - nečekáme až na dokončení prováděné úlohy
    pro výběr další úlohy k provádění
  - v každém časovém okamžiku je nutné zvážit, zda není k dispozici
    jiná prioritnější úloha (např. vzhledem k jejímu r_j)
  - pokud existuje prioritnější úloha,
    přerušíme aktuální úlohu a spustíme prioritnější úlohu
- Cvičení: aplikujte preemptivní EDD na předchozí příklad

$1|r_j, prmp|L_{max}$

- Preemptivní úlohy: je možné přerušit jejich provádění
- Preemptivní verze řídících pravidel:
  - nečekáme až na dokončení prováděné úlohy pro výběr další úlohy k provádění
  - v každém časovém okamžiku je nutné zvážit, zda není k dispozici jiná prioritnější úloha (např. vzhledem k jejímu r_j)
  - pokud existuje prioritnější úloha, přerušíme aktuální úlohu a spustíme prioritnější úlohu
- Cvičení: aplikujte preemptivní EDD na předchozí příklad

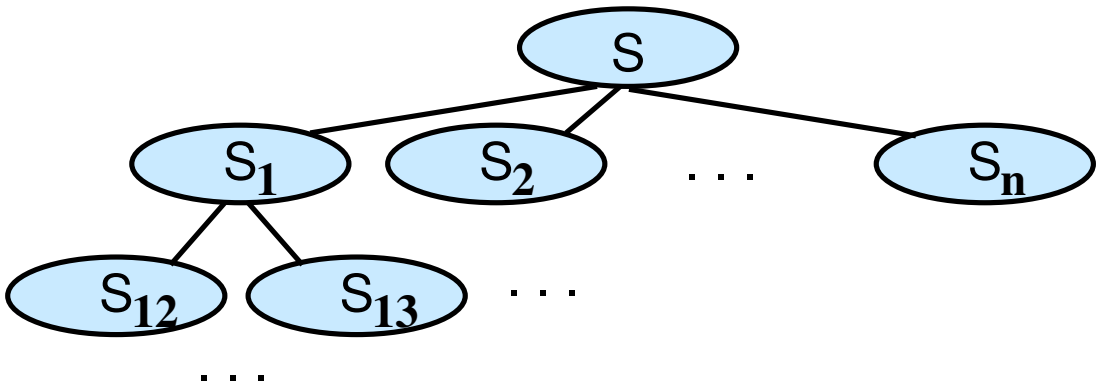
- Preemptivní EDD pravidlo je optimální pro preemptivní verzi problému

$$1|r_j, prmp|L_{max}$$

- Preemptivní EDD je optimální pro předchozí příklad

Metoda větví a mezí

- Prohledávací prostor se rychle zvětšuje se zvětšujícím počtem proměnných
- Metoda větví a mezí *(Branch & Bound search, BB)*
  - založena na myšlence postupného dělení prohledávacího prostoru



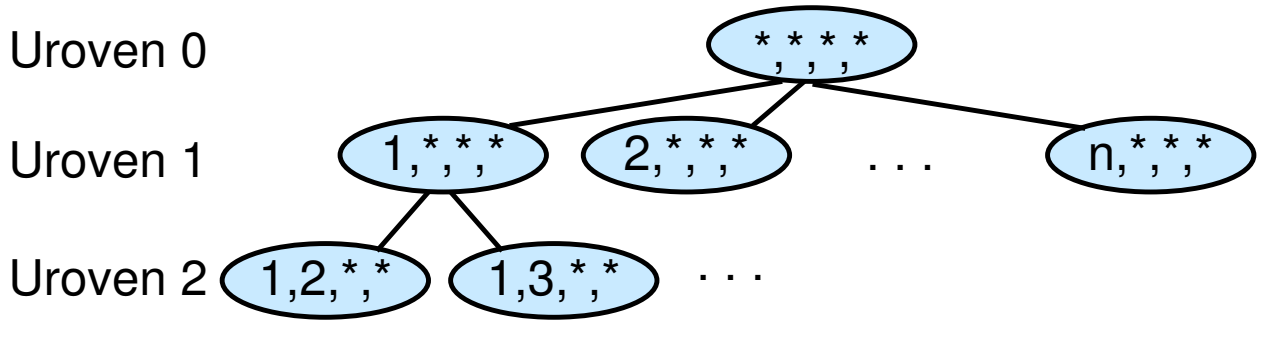
$$S = S_1 \cup S_2 \cup \cdots \cup S_n \qquad S_1 \cap S_2 \cap \cdots \cap S_n = \emptyset$$

  - potřebujeme spočítat hranici/mez na cenu řešení
  - části stavového prostoru, které dávají řešení horší než tato mez nemusíme prohledávat

Výpočet dolní hranice řešení

- Typický způsob, jak zjistit hranice je
  relaxovat (zjednodušit) původní problém
  (např. odstraněním některých požadavků)
  na snadno řešitelný problém
  - jestliže neexistuje řešení pro relaxovaný problém,
    pak neexistuje řešení pro původní problém
    (větev lze smazat)
  - jestliže je optimální řešení relaxovaného problému zároveň řešením původního problému,
    pak je pro něj také optimální
  - jestliže optimální řešení není řešením původního problému, pak dává hranici na jeho řešení
    (původní problém nebude mít zcela jistě lepší řešení)

Pravidla větvení pro $1|r_j|L_{max}$

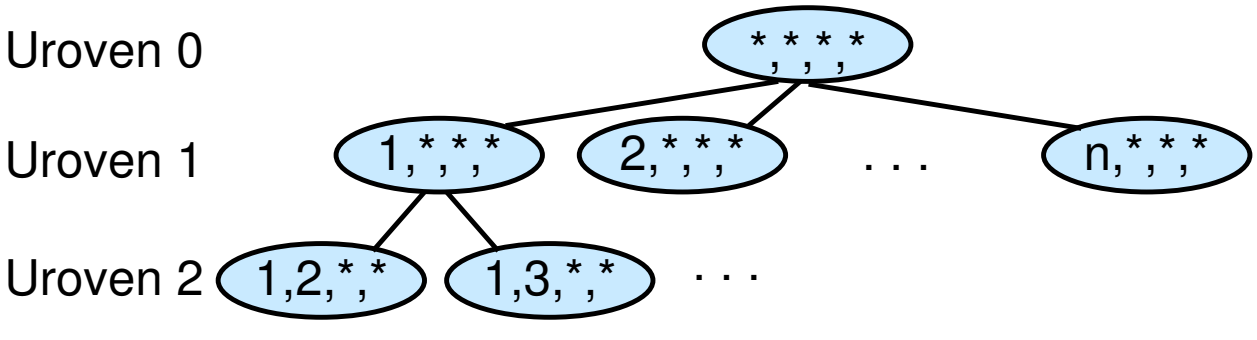


- Rozvrh je konstruován od času $t = 0$
- Úroveň 1:

  n větví, ve kterých je každá z n úloh rozvrhována první
- Úroveň $k - 1$:

  úlohy $j_1, \ldots, j_{k-1}$ jsou rozvrhovány v pořadí $1, \ldots, k - 1$

Pravidla větvení pro $1|r_j|L_{max}$

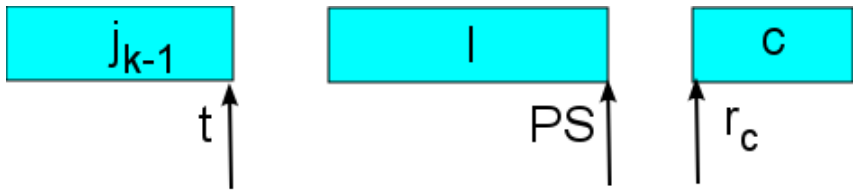


- Rozvrh je konstruován od času $t = 0$
- Úroveň 1:

  n větví, ve kterých je každá z n úloh rozvrhována první
- Úroveň $k - 1$:

  úlohy $j_1, \ldots, j_{k-1}$ jsou rozvrhovány v pořadí $1, \ldots, k - 1$
- Úlohu c uvažujeme na úrovni k (a odpovídající větvení) pokud:

$r_c < \min_{l \in J} (\max(t, r_l) + p_l) = PS$



  - J množina dosud nerozvržených úloh
  - t čas, kdy je skončena j_{k-1} a lze rozvrhovat další úlohu
  - pokud je $r_c \geq PS$, pak je třeba rozvrhovat úlohu minimalizující PS na pozici k a úlohu c na pozici $k + 1$ (nebo i později)
    (toto uspořádání úloh stejně vůbec neovlivní čas dokončení c)

Dolní hranice pro $1|r_j|L_{max}$

- Relaxace problému $1|r_j|L_{max}$ je problém $1|r_j, prmp|L_{max}$
  - neumožnění přerušení je omezení pouze v původním problému $1|r_j|L_{max}$
- Preemptivní EDD pravidlo je optimální pro $1|r_j, prmp|L_{max}$

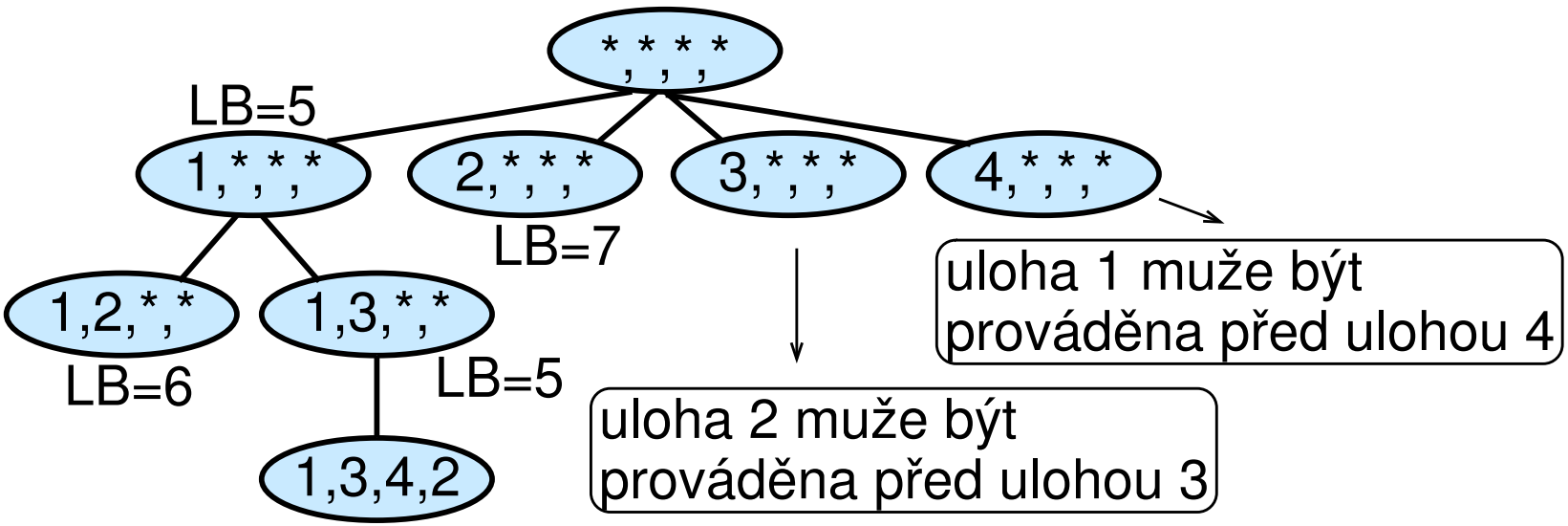
$\Rightarrow$ Preemptivní rozvrh (rozvrh bez zdržení) určitě nebude mít L_{max} větší než ne-preemptivní rozvrh (rozvrh potenciálně se zdržením)

$\Rightarrow$ Dolní hranice na úrovni $k-1$ může být založena na rozvrhu zbývajících úloh podle preemptivního EDD pravidla

$+$ Jestliže preemptivní EDD dává nepreemptivní rozvrh, pak můžeme všechny uzly s větší dolní hranicí zrušit

Příklad: BB pro $1|r_j|L_{max}$

Úlohy	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10

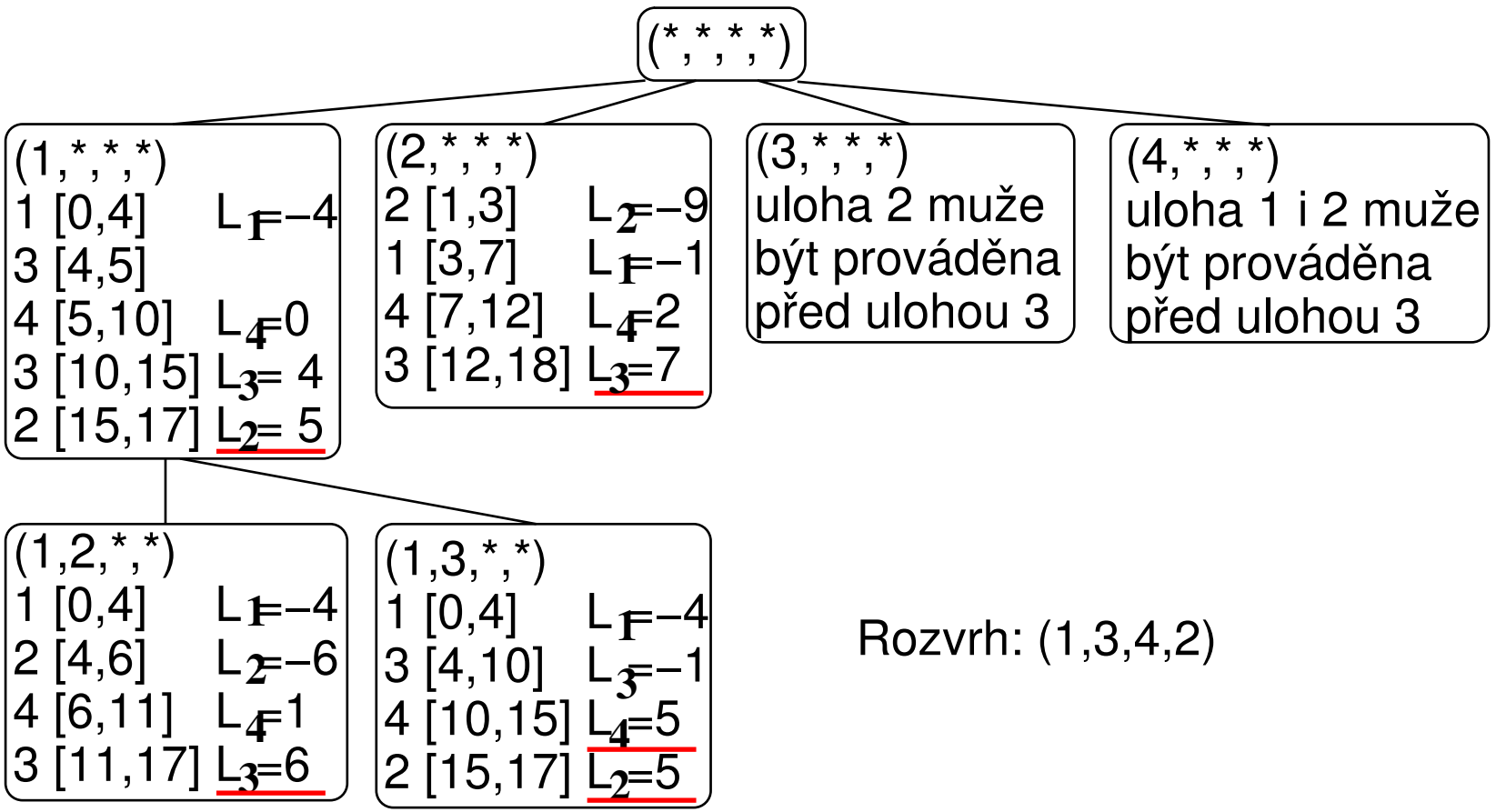


1,4,*,* nemá smysl prozkoumávat,

protože už v 1,3,*,* máme šanci na řešení s minimání LB=5

Příklad: dolní hranice BB pro $1|r_j|L_{max}$

Úlohy	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10



Rozvrh: (1,3,4,2)

Paprskové prohledávání

- Metoda větví a mezí
  - uvažuje každý uzel
    - pro n úloh:
      na první úrovni n uzlů, na druhé úrovni $n(n-1)$ uzlů,
      na třetí úrovni $n(n-1)(n-2)$ uzlů, ...
    - $\Rightarrow$ $n!$ uzlů na n-té úrovni
  - garantuje optimum
  - obvykle příliš pomalá

Paprskové prohledávání

- Metoda větví a mezí
  - uvažuje každý uzel
    - pro n úloh:
      na první úrovni n uzlů, na druhé úrovni $n(n-1)$ uzlů,
      na třetí úrovni $n(n-1)(n-2)$ uzlů, ...
    - $\Rightarrow$ $n!$ uzlů na n-té úrovni
  - garantuje optimum
  - obvykle příliš pomalá
- Paprskové prohledávání *(Beam Search, BS)*
  - uvažuje pouze slibné uzly
  - šířka paprsku *(beam width)*
    - udává, u kolika uzlů budeme na každé úrovni pokračovat v prohledávání
  - negarantuje optimální řešení
  - mnohem rychlejší

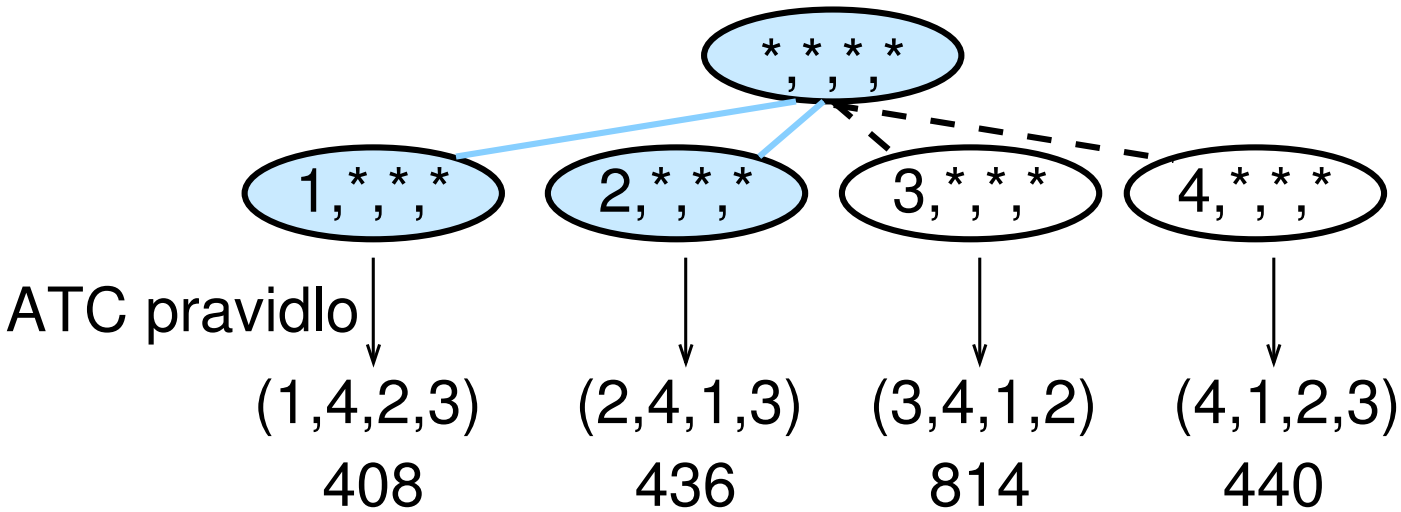
Větvení

Kriterium: $1||\sum_j w_j T_j$

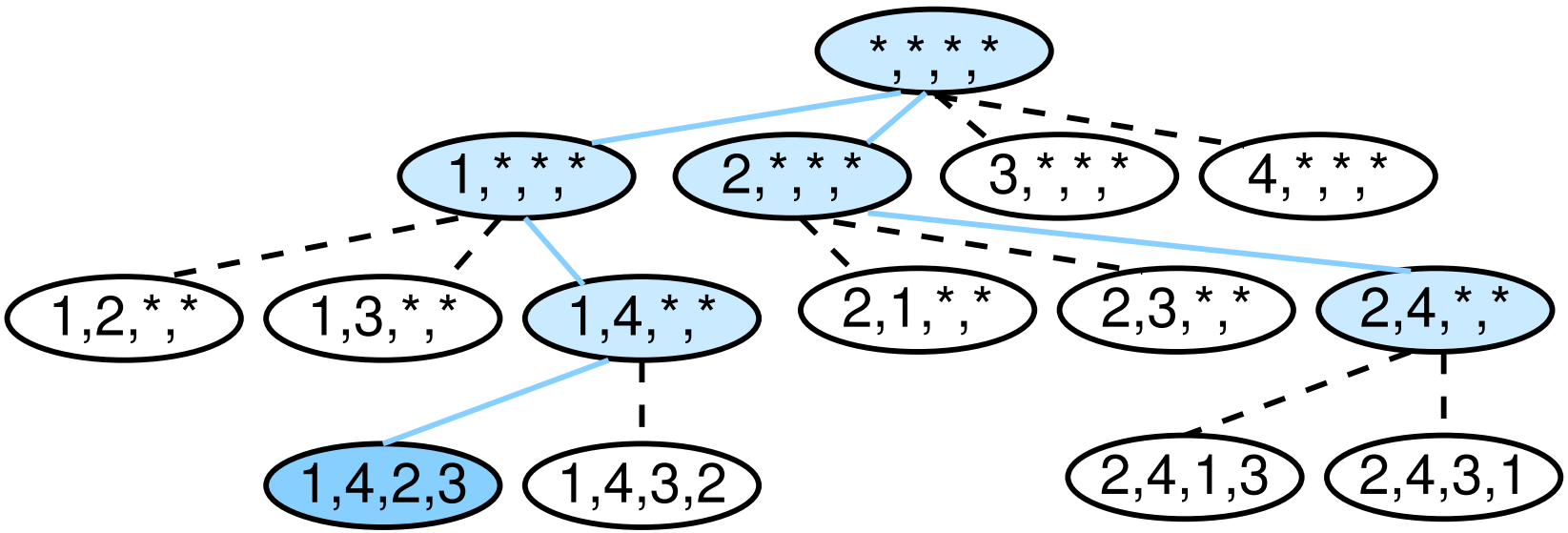
Problém:

Úlohy	1	2	3	4
p_j	10	10	13	4
r_j	4	2	1	12
d_j	14	12	1	12

Šířka paprsku 2:



Paprskové prohledávání



Diskuse

- Kompromisy při implementaci:
  - podrobná evaluace uzlů (kvůli přesnosti)
  - odhad evaluace uzlu (kvůli rychlosti)
- Dvoufázová procedura
  - odhad evaluace
    - je prováděn pro všechny uzly na úrovni k
  - podrobná evaluace
    - šířka filtru $>$ šířka paprsku
    - prováděna pro počet uzlů odpovídající šířce filtru

Celočíselný program

$$\text{minimalizace} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{za předpokladu:} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad \forall i : 1 \leq i \leq m$$

$$x_j \geq 0 \quad \forall j : 1 \leq j \leq n$$

$$x_j \in \mathbb{Z} \quad \forall j : 1 \leq j \leq n$$

Model problému 1| | $\sum_{j=1}^{n} w_j C_j$

- Proměnné $x_{jt} = 1$ jestliže úloha j začne v čase t
  $= 0$ jinak
- Formulace celočíselného programu

  Minimalizace

$$\sum_{j=1}^{n} \sum_{t=0}^{C_{max}-1} w_j(t + p_j)x_{jt} \qquad \text{tj.} \sum_{j=1}^{n} w_j C_j$$

Model problému 1| | $\sum_{j=1}^{n} w_j C_j$

- Proměnné $x_{jt} = 1$ jestliže úloha j začne v čase t
  $= 0$ jinak
- Formulace celočíselného programu

  Minimalizace

$$\sum_{j=1}^{n} \sum_{t=0}^{C_{max}-1} w_j(t + p_j)x_{jt} \qquad \text{tj.} \sum_{j=1}^{n} w_j C_j$$

  za předpokladu

$$x_{jt} \in 0,1 \quad \forall j,t$$

každá úloha právě jednou začne:

$$\sum_{t=0}^{C_{max}-1} x_{jt} = 1 \quad \forall j$$

v každém čase běží právě jedna úloha:

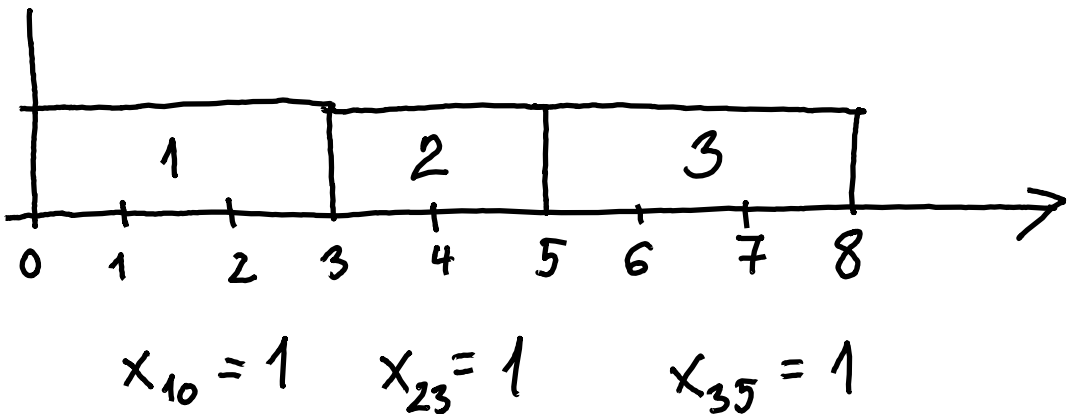
$$\sum_{j=1}^{n} \sum_{s=\max(t-p_j,0)}^{t-1} x_{js} = 1 \quad \forall t$$

(pro každé t běží v intervalu $\langle t-1, t)$ právě jedna úloha)

V každém čase jedna úloha

- Proměnné $x_{jt} = 1$ jestliže úloha j začne v čase t
- V každém čase, tj. v každém intervalu $\langle t-1, t\rangle$, běží právě jedna úloha:

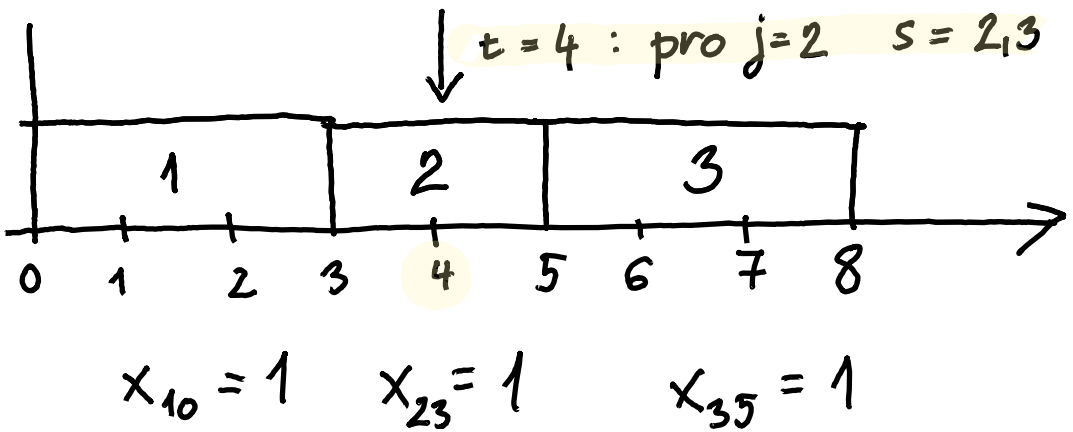
$$\sum_{j=1}^{n} \sum_{s=\max(t-p_j,0)}^{t-1} x_{js} = 1 \qquad \forall t$$



V každém čase jedna úloha

- Proměnné $x_{jt} = 1$ jestliže úloha j začne v čase t
- V každém čase, tj. v každém intervalu $\langle t-1, t \rangle$, běží právě jedna úloha:

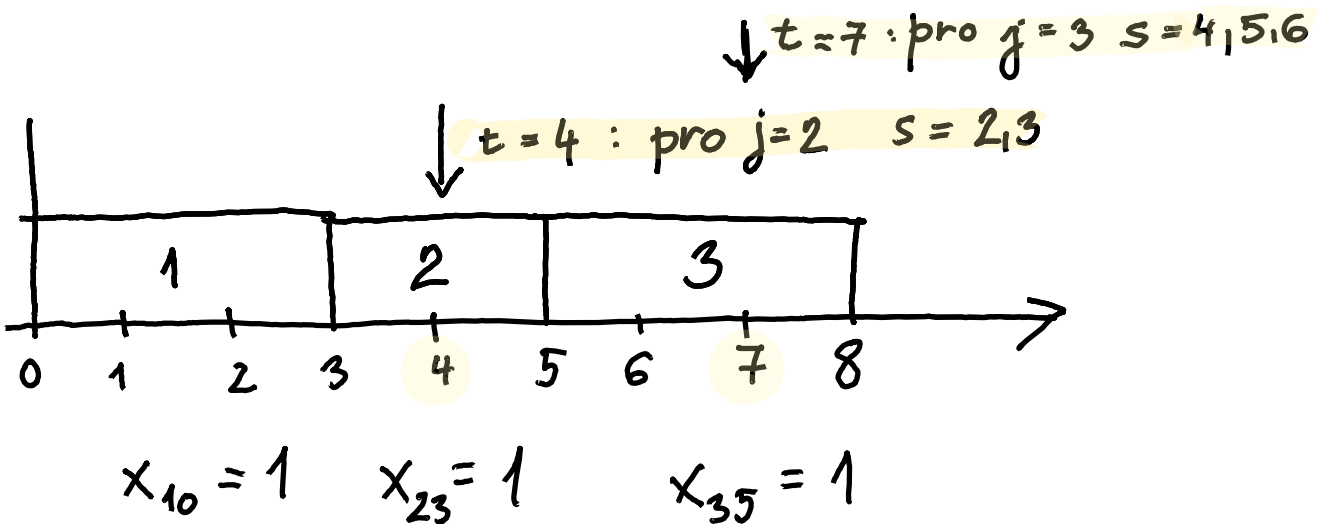
$$\sum_{j=1}^{n} \sum_{s=\max(t-p_j,0)}^{t-1} x_{js} = 1 \qquad \forall t$$



V každém čase jedna úloha

- Proměnné $x_{jt} = 1$ jestliže úloha j začne v čase t
- V každém čase, tj. v každém intervalu $\langle t-1, t \rangle$, běží právě jedna úloha:

$$\sum_{j=1}^{n} \sum_{s=\max(t-p_j,0)}^{t-1} x_{js} = 1 \qquad \forall t$$



Plánování job-shopu

16. května 2022

Multi-operační rozvrhování: job shop s minimalizací makespan

- n úloh
- m strojů
- operace (i, j): provádění úlohy j na stroji i
- Pořadí operací úlohy je stanoveno:
  - $(i, j) \rightarrow (k, j)$ specifikuje, že úloha j má být prováděna na stroji i dříve než na stroji k
- p_{ij}: trvání operace (i, j)
- Cíl: rozvrhovat úlohy na strojích
  - bez překrytí na strojích
  - bez překrytí v rámci úlohy
  - minimalizace *makespan* C_{max}

Příklad: *job shop* problém

Data:

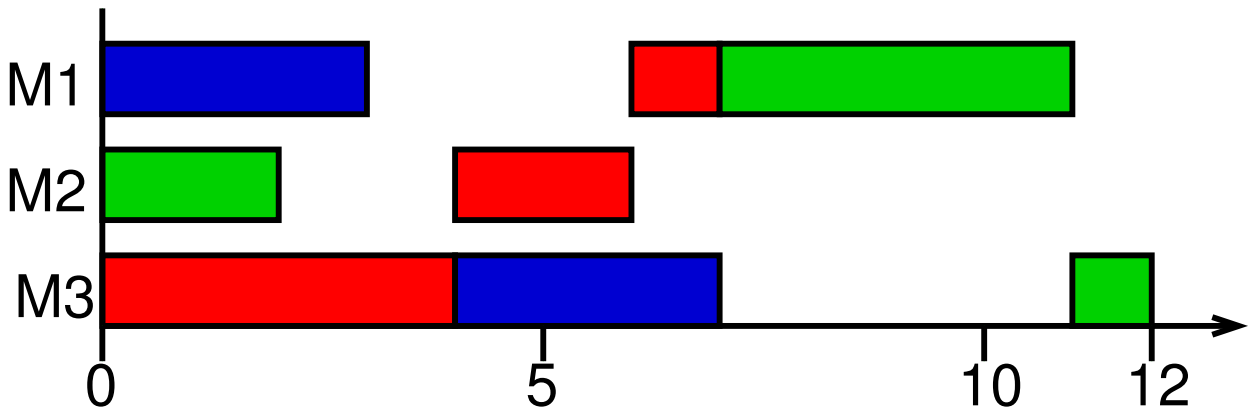
- stroje: $M1, M2, M3$
- úlohy: $J1 : (3,1) \rightarrow (2,1) \rightarrow (1,1)$
  $J2 : (1,2) \rightarrow (3,2)$
  $J3 : (2,3) \rightarrow (1,3) \rightarrow (3,3)$
- doby trvání: $p31 = 4, p21 = 2, p11 = 1$
  $p12 = 3, p32 = 3$
  $p23 = 2, p13 = 4, p33 = 1$

Příklad: *job shop* problém

Data:

- stroje: $M1, M2, M3$
- úlohy: $J1 : (3,1) \rightarrow (2,1) \rightarrow (1,1)$
  $J2 : (1,2) \rightarrow (3,2)$
  $J3 : (2,3) \rightarrow (1,3) \rightarrow (3,3)$
- doby trvání: $p31 = 4, p21 = 2, p11 = 1$
  $p12 = 3, p32 = 3$
  $p23 = 2, p13 = 4, p33 = 1$

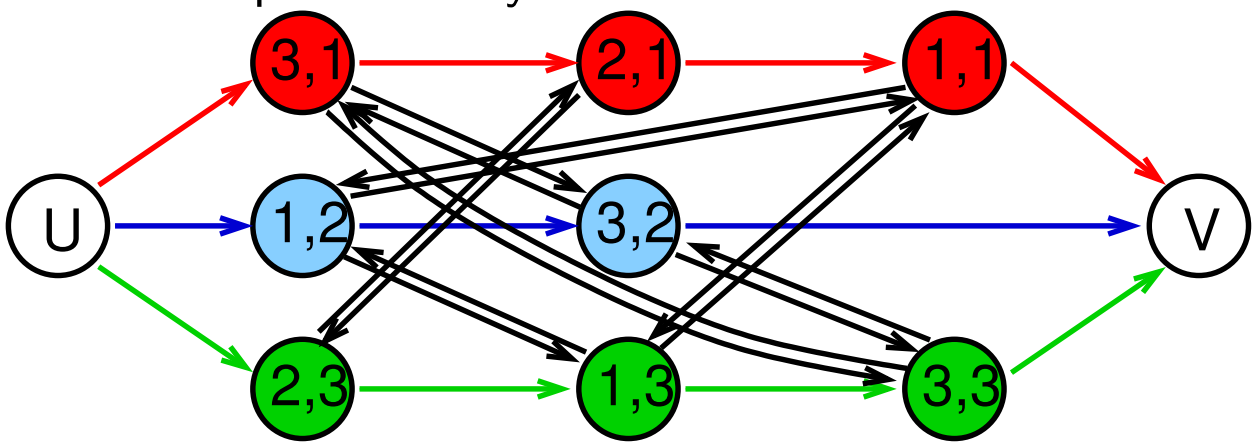
Řešení:



Disjunktivní grafová reprezentace

Graf $G = (N, A \cup B \cup P)$

- uzly odpovídají operacím $N = \{(i,j)|(i,j) \text{ je operace}\} \cup \{U, V\}$
- dva pomocné uzly U a V reprezentující zdroj a stok
- konjunktivní hrany A reprezentují pořadí operací úlohy
  - $(i,j) \to (k,j) \in A \iff$ operace (i,j) předchází (k,j)
- disjunktivní hrany B reprezentují konflikty na strojích
  - dvě operace (i,j) a (i,l) jsou spojeny dvěma opačně orientovanými hranami
- pomocné hrany P
  - hrany z U ke všem prvním operacím úlohy
  - hrany ze všech posledních operací úlohy do V



Výběr hran

Pojmy:

- Podmnožina $D \subset B$ je nazývána výběr, jestliže obsahuje z každého páru disjuktivních hran právě jednu
- Výběr D je splnitelný, jestliže výsledný orientovaný graf $G(D) = (N, A \cup D \cup P)$ je acyklický
  - jedná se o graf s pomocnými, konjunktivními hranami a vybranými disjunktními hranami

Poznámky:

- splnitelný výběr určuje posloupnost, ve které jsou operace prováděny na strojích
- vztah splnitelného výběru a (konzistentního) rozvrhu?

Výběr hran

Pojmy:

- Podmnožina $D \subset B$ je nazývána výběr, jestliže obsahuje z každého páru disjuktivních hran právě jednu
- Výběr D je splnitelný, jestliže výsledný orientovaný graf $G(D) = (N, A \cup D \cup P)$ je acyklický
  - jedná se o graf s pomocnými, konjunktivními hranami a vybranými disjunktními hranami

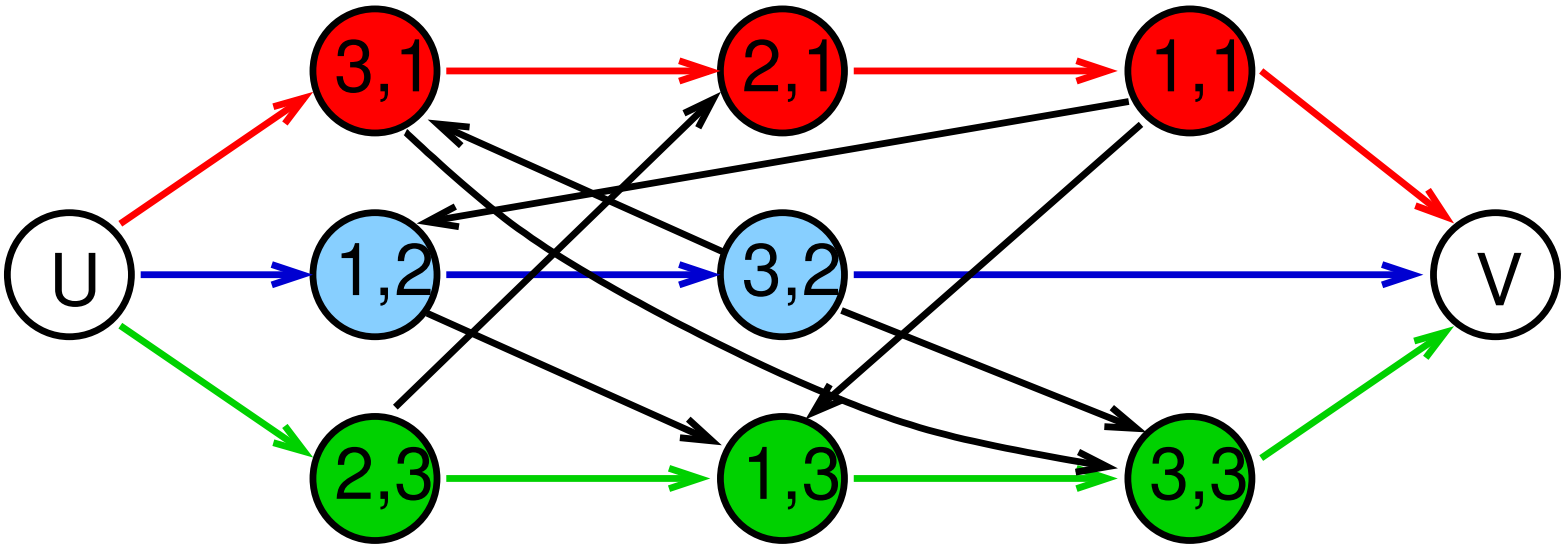
Poznámky:

- splnitelný výběr určuje posloupnost, ve které jsou operace prováděny na strojích
- vztah splnitelného výběru a (konzistentního) rozvrhu?

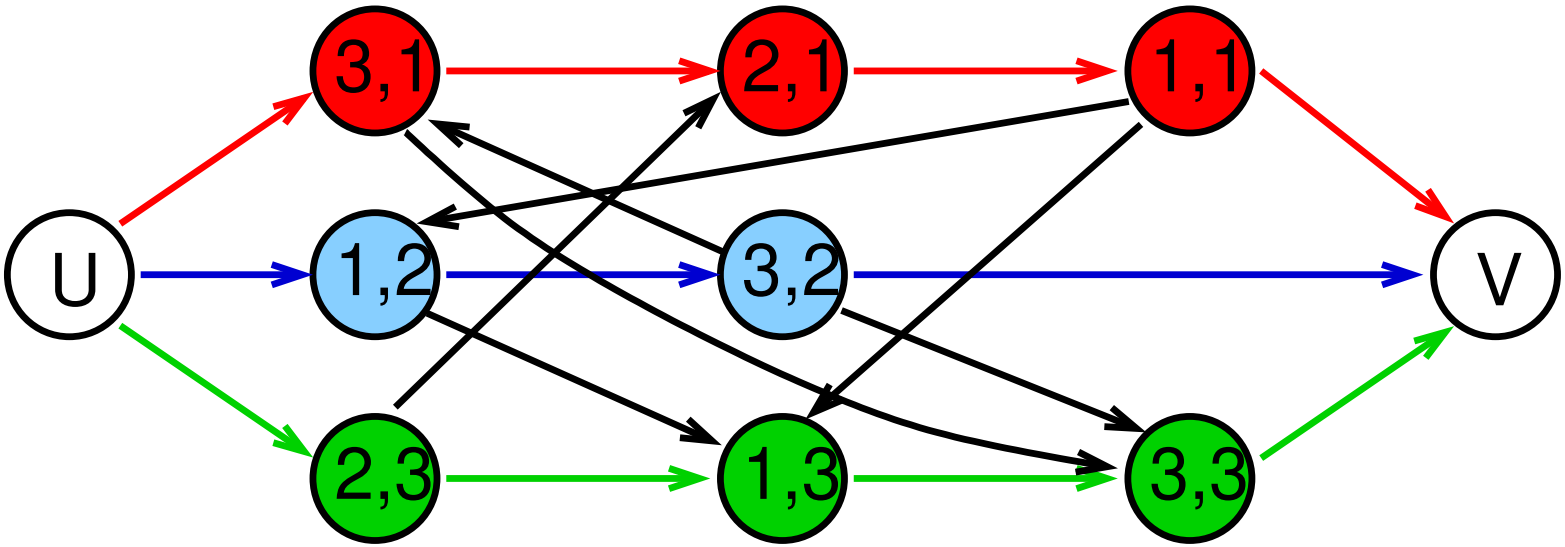
  každý (konzistentní) rozvrh jednoznačně určuje splnitelný výběr

  každý splnitelný výběr jednoznačně určuje (konzistentní) rozvrh

Příklad: nesplnitelný výběr



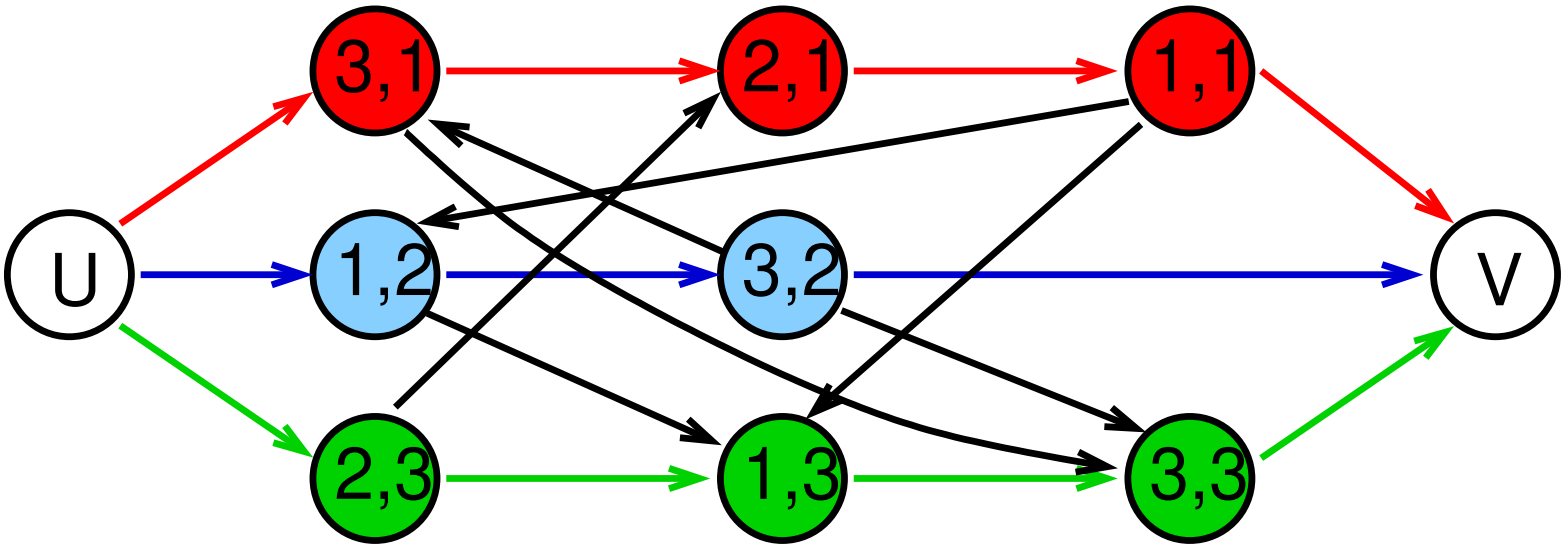
Příklad: nesplnitelný výběr



V grafu existuje v důsledku nevhodného výběru hran cyklus:

- $(1, 2) \to (3, 2)$

Příklad: nesplnitelný výběr



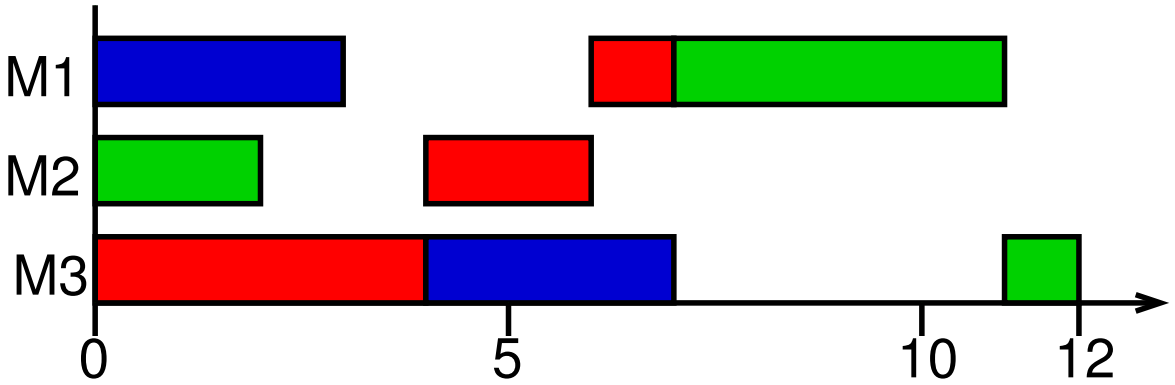
V grafu existuje v důsledku nevhodného výběru hran cyklus:

- $(1, 2) \to (3, 2)$
- $(3, 2) \to (3, 1) \to (2, 1) \to (1, 1) \to (1, 2)$

$\Rightarrow$ nelze splnit (k tomuto výběru neexistuje rozvrh)

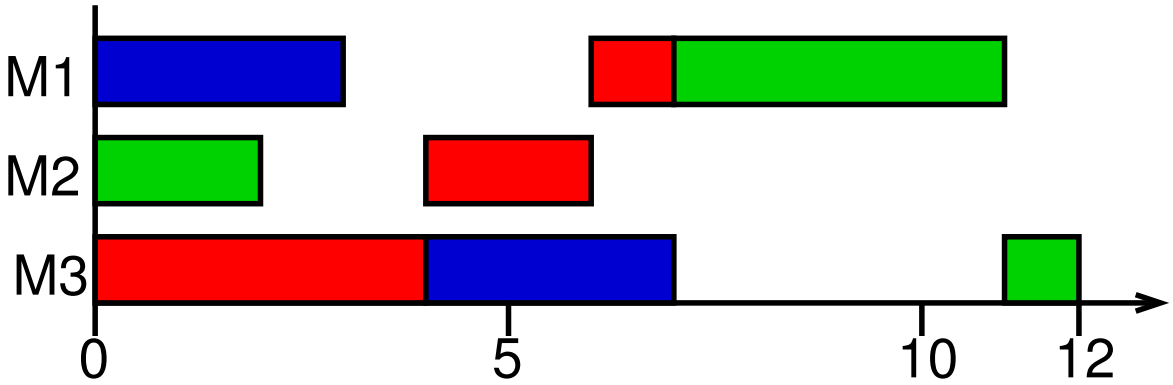
Příklad: výběr pro daný rozvrh

Nalezněte (splnitelný) výběr hran pro daný rozvrh:



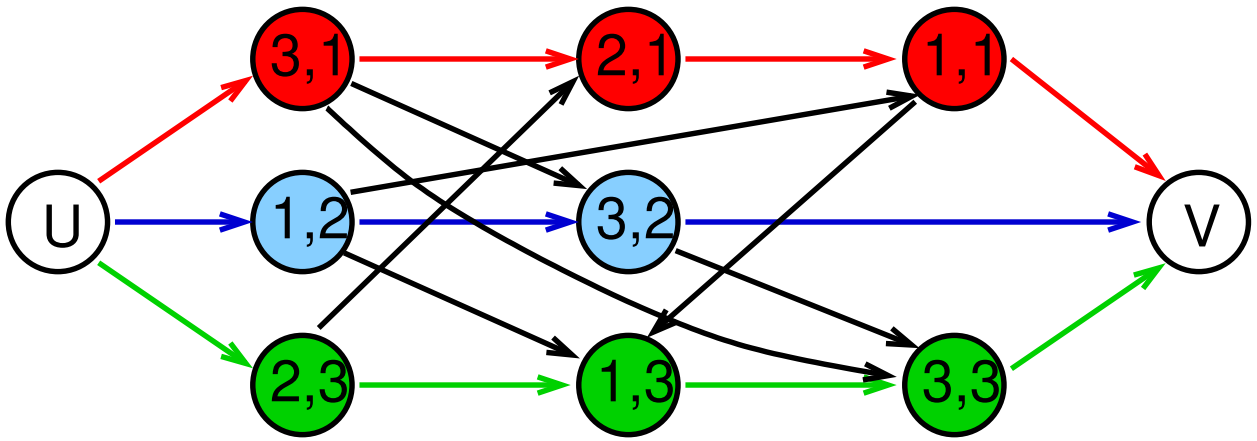
Příklad: výběr pro daný rozvrh

Nalezněte (splnitelný) výběr hran pro daný rozvrh:



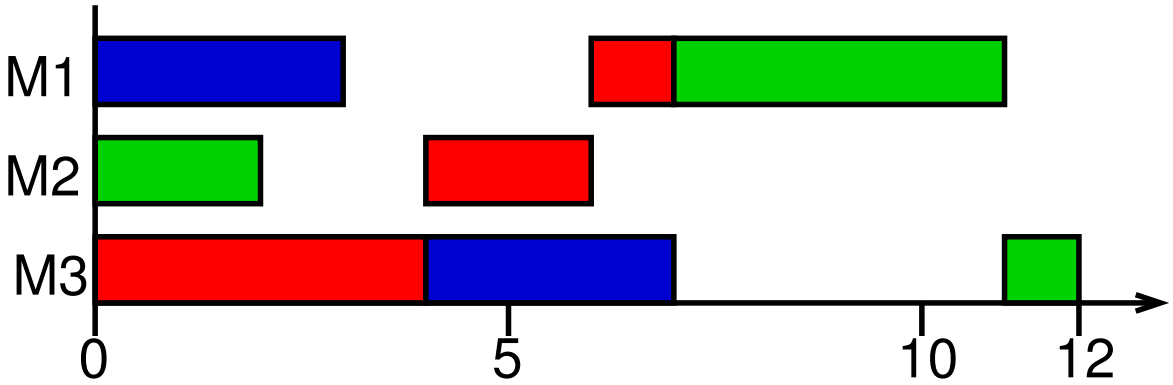
Konstrukce odpovídajícího výběru:

vybereme (jeden stroj po druhém) disjunktivní hrany,
které odpovídají uspořádání operací na stroji v daném rozvrhu:



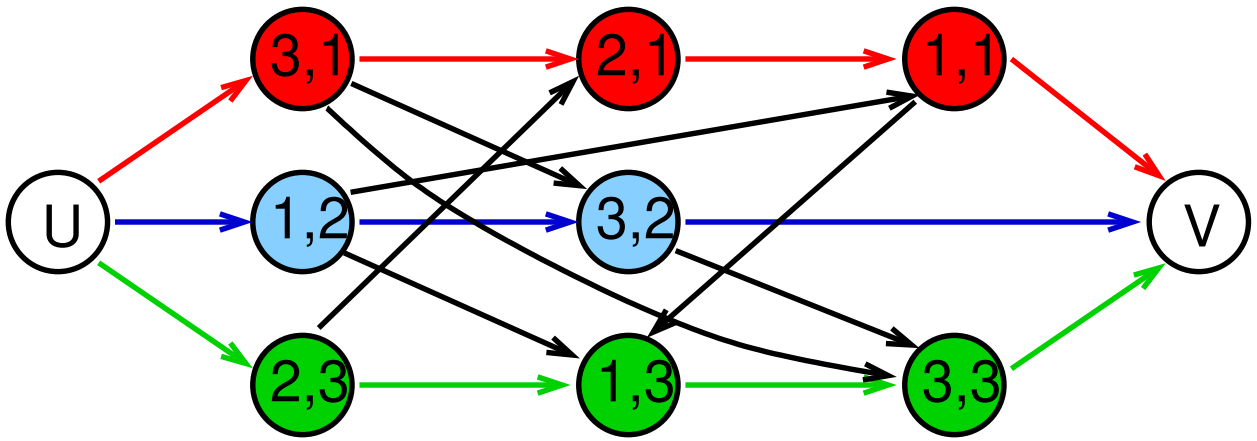
Příklad: výběr pro daný rozvrh

Nalezněte (splnitelný) výběr hran pro daný rozvrh:



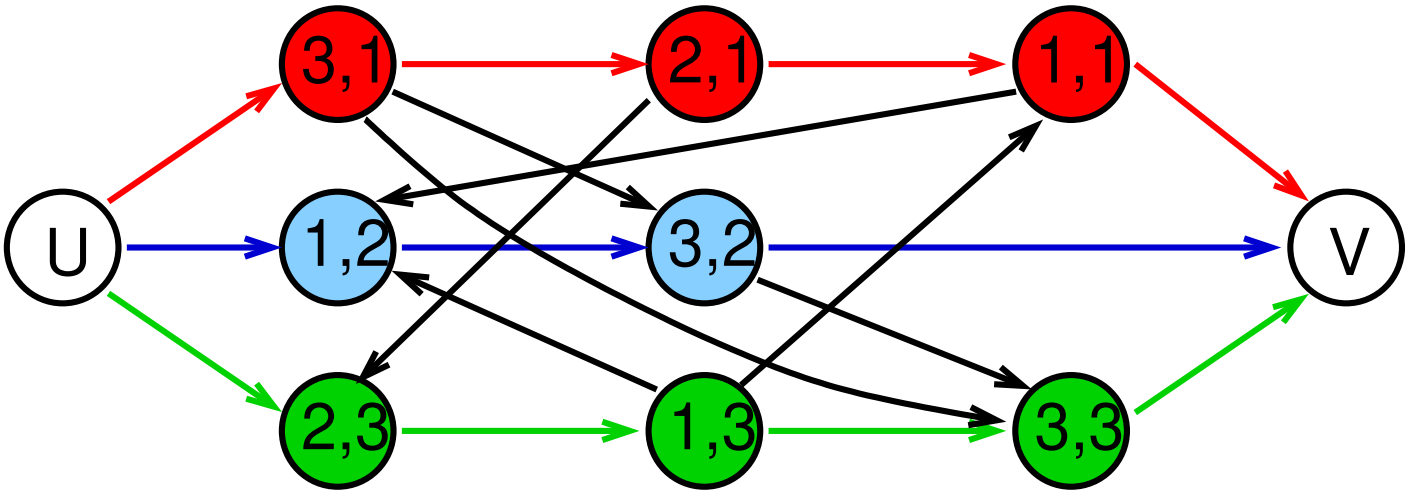
Konstrukce odpovídajícího výběru:

vybereme (jeden stroj po druhém) disjunktivní hrany,
které odpovídají uspořádání operací na stroji v daném rozvrhu:

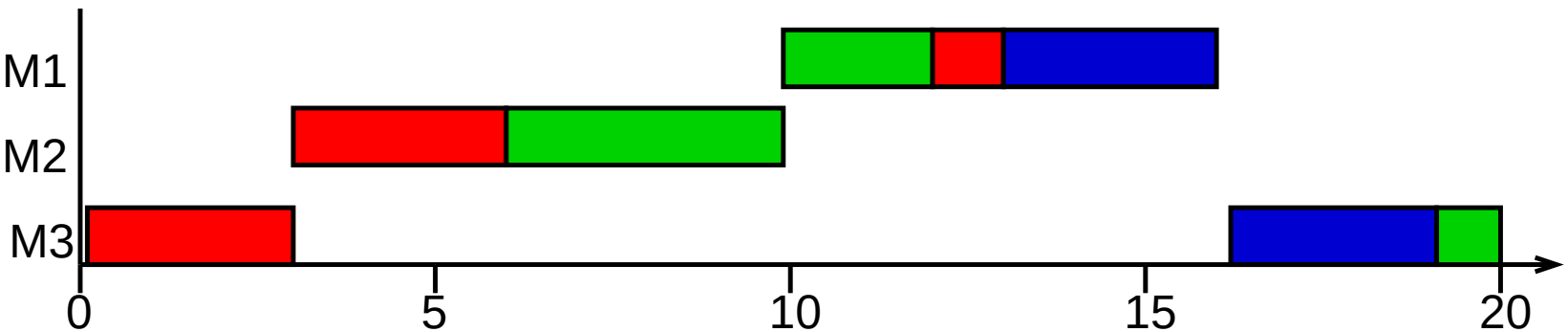


Příklad: rozvrh pro daný splnitelný výběr

Jakým způsobem nalézt rozvrh pro daný splnitelný výběr?



Tedy: jakým způsobem lze nalézt tento odpovídající rozvrh:



Výpočet rozvrhu pro výběr

Metoda: výpočet nejdelších cest z U do dalších uzlů v $G(D)$

obdoba dopředného zpracování z metody kritické cesty pro plánování projektu

Technický popis:

- uzly (i, j) mají ohodnocení p_{ij}, uzel U má váhu 0
- délka cesty $i_1, i_2, \ldots, i_r$ je součet ohodnocení uzlů $i_1, i_2, \ldots, i_{r-1}$
- spočítej délku l_{ij} nejdelší cesty z U do (i, j) a V

Výpočet rozvrhu pro výběr

Metoda: výpočet nejdelších cest z U do dalších uzlů v $G(D)$

obdoba dopředného zpracování z metody kritické cesty pro plánování projektu

Technický popis:

- uzly (i, j) mají ohodnocení p_{ij}, uzel U má váhu 0
- délka cesty $i_1, i_2, \ldots, i_r$ je součet ohodnocení uzlů $i_1, i_2, \ldots, i_{r-1}$
- spočítej délku l_{ij} nejdelší cesty z U do (i, j) a V
  1. $l_U = 0$ a pro každý uzel (i, j) s jediným předchůdcem U: $l_{ij} = 0$
  2. vypočítej postupně pro všechny zbývající uzly (i, j) (a pro uzel V):

Výpočet rozvrhu pro výběr

Metoda: výpočet nejdelších cest z U do dalších uzlů v $G(D)$
obdoba dopředného zpracování z metody kritické cesty pro plánování projektu

Technický popis:

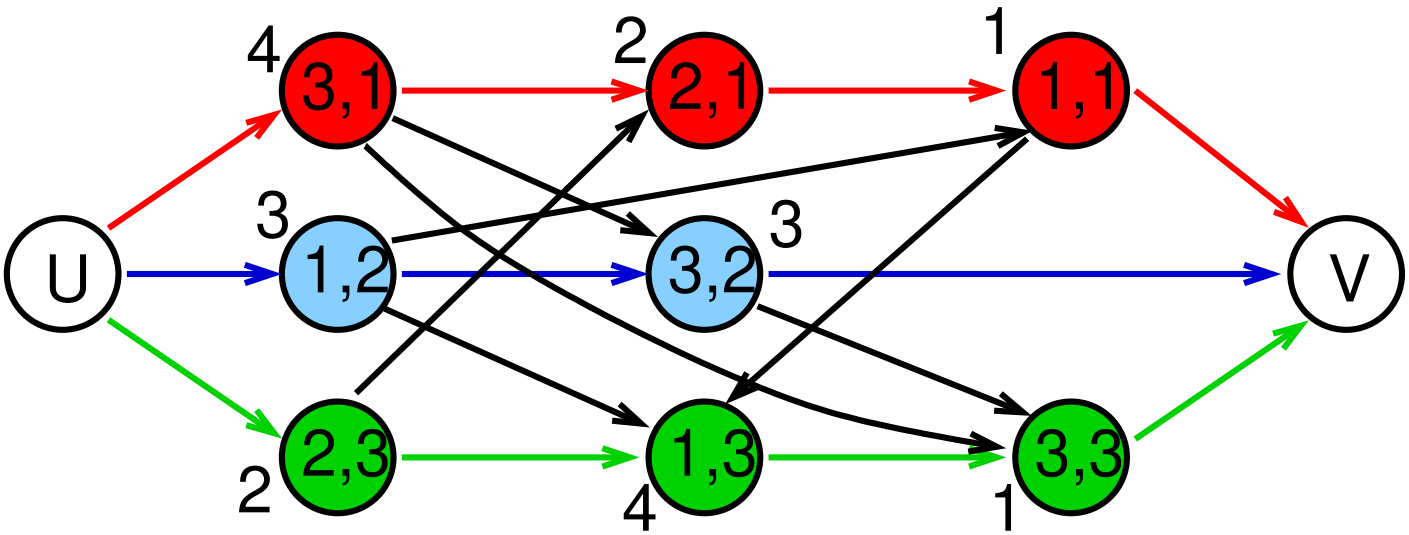
- uzly (i, j) mají ohodnocení p_{ij}, uzel U má váhu 0
- délka cesty $i_1, i_2, \ldots, i_r$ je součet ohodnocení uzlů $i_1, i_2, \ldots, i_{r-1}$
- spočítej délku l_{ij} nejdelší cesty z U do (i, j) a V
  1. $l_U = 0$ a pro každý uzel (i, j) s jediným předchůdcem U: $l_{ij} = 0$
  2. vypočítej postupně pro všechny zbývající uzly (i, j) (a pro uzel V):

$$l_{ij} = \max_{\forall (k,l):(k,l)\to(i,j)} (l_{kl} + p_{kl})$$

tj. projdeme všechny předchůdce (k, l) uzlu (i, j)
délka cesty do (i, j) přes (k, l) je $l_{kl} + p_{kl}$

- zahaj operaci (i, j) v čase l_{ij}
- délka nejdelší cesty z U do V je rovna *makespan*
  - tato cesta je kritická cesta

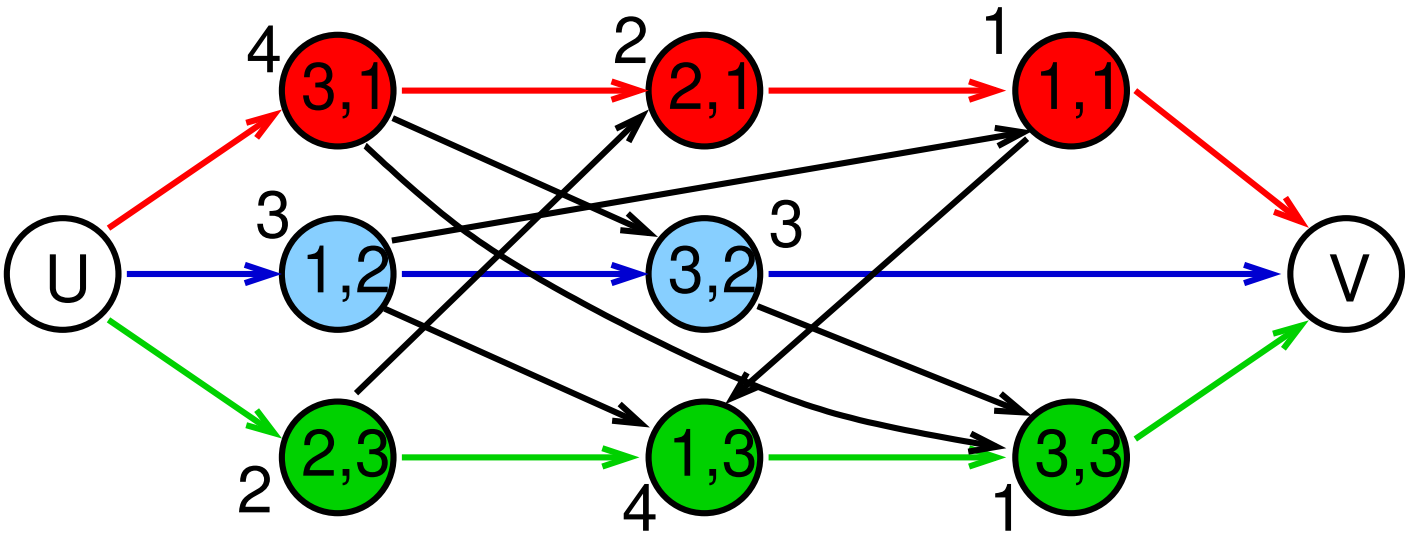
Výpočet rozvrhu pro výběr



Výpočet l_{ij}:

uzel	(3,1)	(1,2)	(2,3)	(2,1)	(3,2)	(1,1)	(1,3)	(3,3)	V
délka	0	0	0						

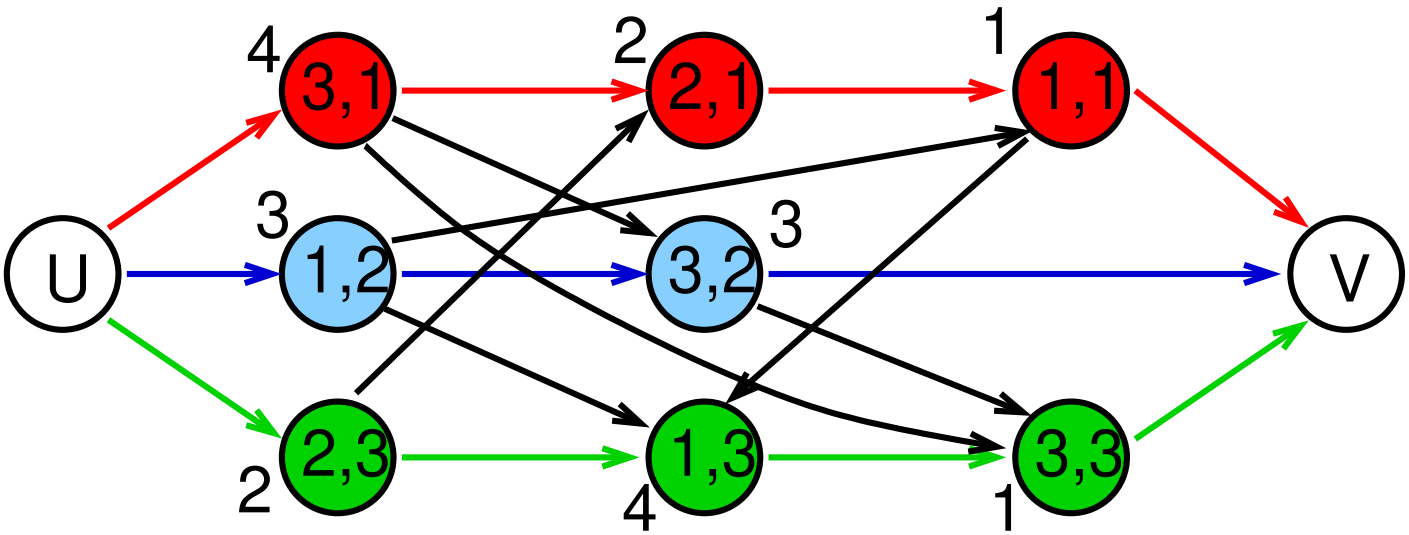
Výpočet rozvrhu pro výběr



Výpočet l_{ij}:

uzel	(3,1)	(1,2)	(2,3)	(2,1)	(3,2)	(1,1)	(1,3)	(3,3)	V
délka	0	0	0	4	4				

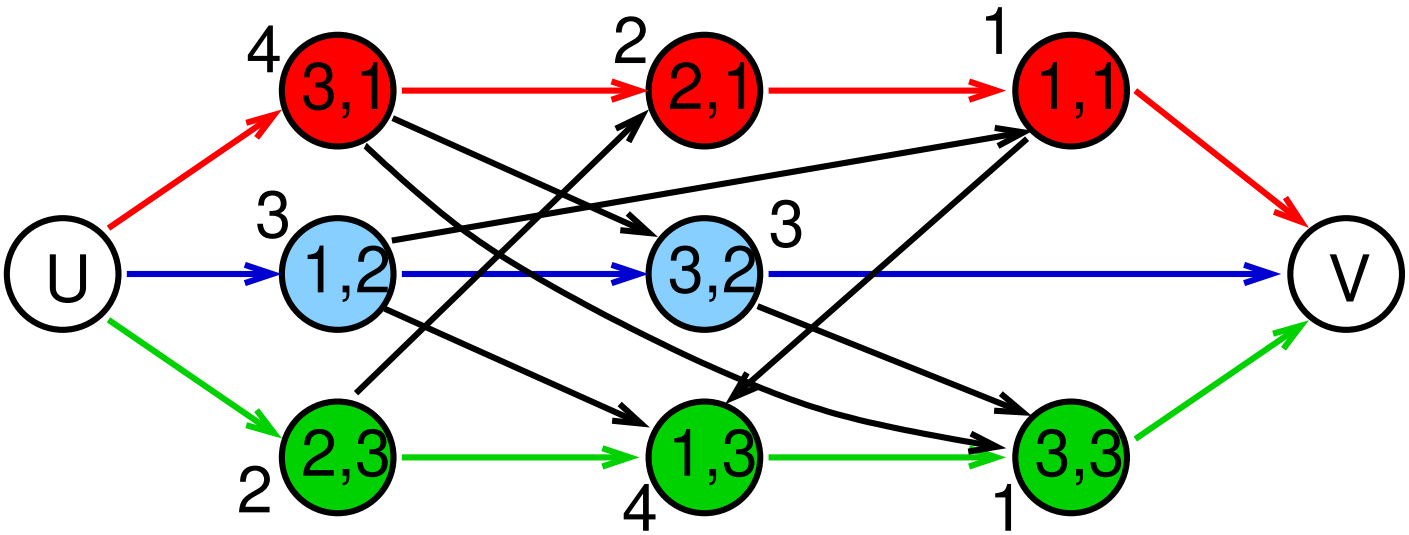
Výpočet rozvrhu pro výběr



Výpočet l_{ij}:

uzel	(3,1)	(1,2)	(2,3)	(2,1)	(3,2)	(1,1)	(1,3)	(3,3)	V
délka	0	0	0	4	4	6			

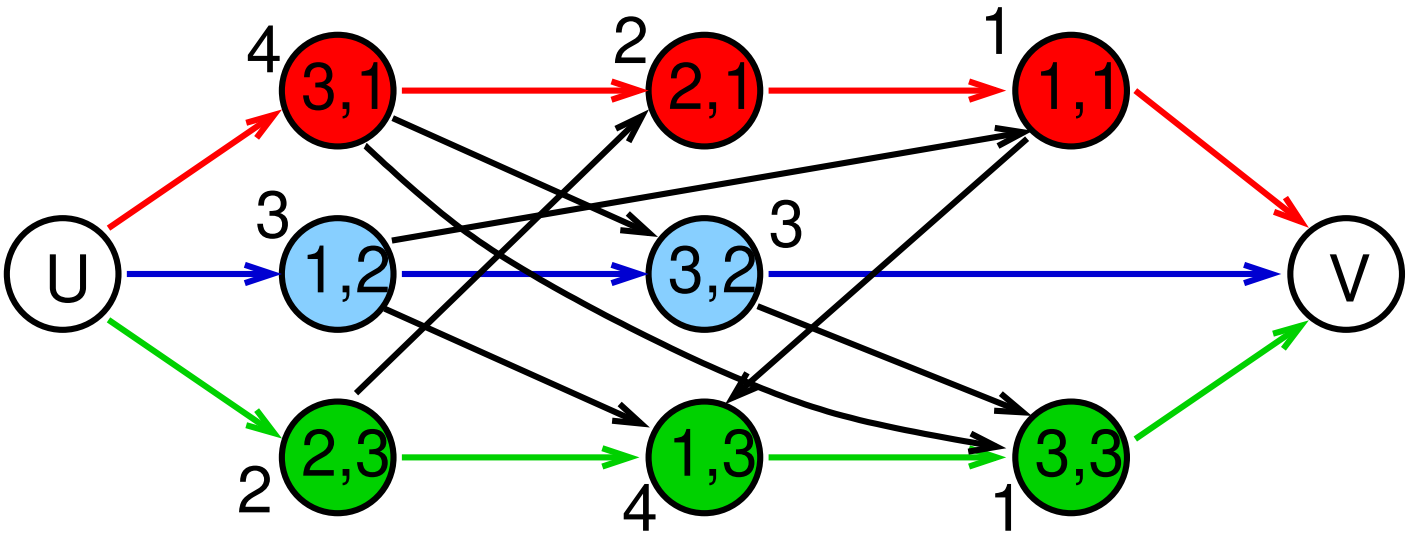
Výpočet rozvrhu pro výběr



Výpočet l_{ij}:

uzel	(3,1)	(1,2)	(2,3)	(2,1)	(3,2)	(1,1)	(1,3)	(3,3)	V
délka	0	0	0	4	4	6	7		

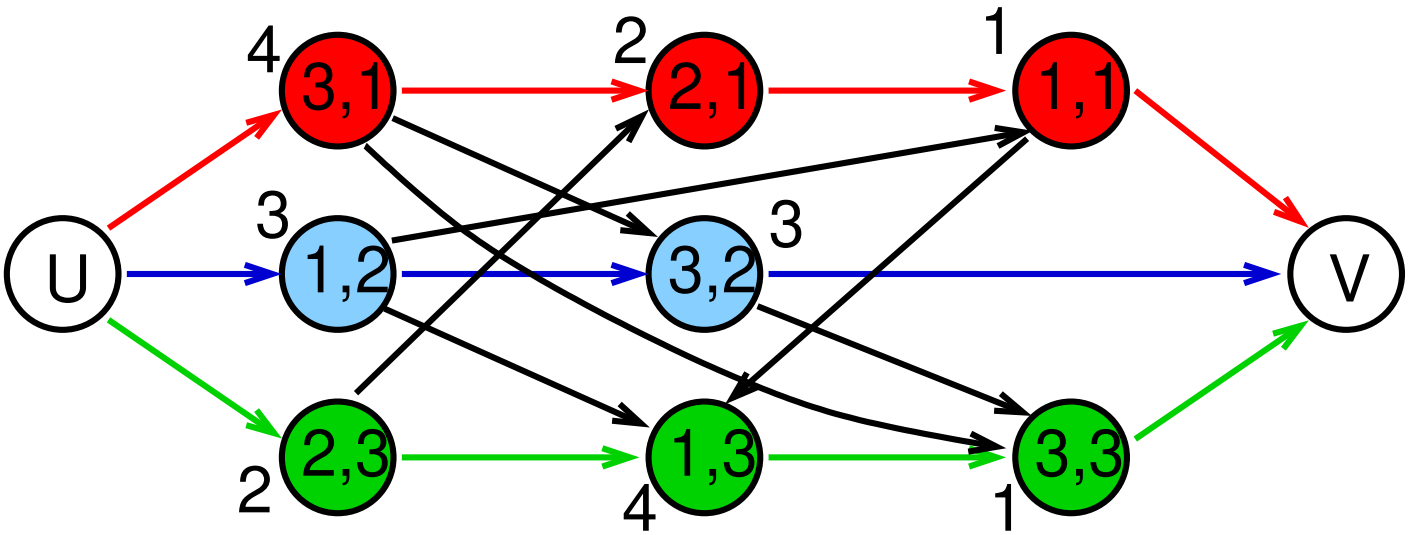
Výpočet rozvrhu pro výběr



Výpočet l_{ij}:

uzel	(3,1)	(1,2)	(2,3)	(2,1)	(3,2)	(1,1)	(1,3)	(3,3)	V
délka	0	0	0	4	4	6	7	11	

Výpočet rozvrhu pro výběr



Výpočet l_{ij}:

uzel	(3,1)	(1,2)	(2,3)	(2,1)	(3,2)	(1,1)	(1,3)	(3,3)	V
délka	0	0	0	4	4	6	7	11	12

Disjunktivní programování

- Formulace disjunktivního programování
  - nejčastěji používaná formulace matematického programování pro *job shop* bez recirkulace
    (bez recirkulace: úlohy prováděny na stroji nejvýše jednou)
  - vychází z disjunktivní grafové reprezentace

- Disjunktivní programování
  - jedná se o matematické programování
  - omezení rozděleny do množin konjunktivních omezení
    - všechna tato omezení musí být splněna
    - standardní matematické programování: všechna omezení konjuktivní

    a jedné nebo více množin disjunktivních omezení
    - z každé množiny disjunktivních omezení musí být alespoň jedno omezení splněno

Disjunktivní programování a *job shop*

y_{ij} značí startovní čas operace (i, j) úlohy j na stroji i

Minimalizace C_{max}

za předpokladu

$$\forall (i, j) \rightarrow (k, j) \in A$$

Disjunktivní programování a *job shop*

y_{ij} značí startovní čas operace (i, j) úlohy j na stroji i

Minimalizace C_{max}

za předpokladu

$$y_{kj} - y_{ij} \geq p_{ij} \qquad \forall (i,j) \to (k,j) \in A$$

Disjunktivní programování a *job shop*

y_{ij} značí startovní čas operace (i, j) úlohy j na stroji i

Minimalizace C_{max}

za předpokladu

$$y_{kj} - y_{ij} \geq p_{ij} \qquad \forall (i,j) \to (k,j) \in A$$

$$\forall (i,j) \in N$$

Disjunktivní programování a *job shop*

y_{ij} značí startovní čas operace (i, j) úlohy j na stroji i

Minimalizace C_{max}

za předpokladu

$$y_{kj} - y_{ij} \geq p_{ij} \qquad \forall (i,j) \to (k,j) \in A$$

$$C_{max} - y_{ij} \geq p_{ij} \qquad \forall (i,j) \in N$$

Disjunktivní programování a *job shop*

y_{ij} značí startovní čas operace (i, j) úlohy j na stroji i

Minimalizace C_{max}

za předpokladu

$$y_{kj} - y_{ij} \geq p_{ij} \qquad \forall (i,j) \to (k,j) \in A$$

$$C_{max} - y_{ij} \geq p_{ij} \qquad \forall (i,j) \in N$$

$$\forall (i,l),(i,j) : \; i = 1 \ldots m$$

Disjunktivní programování a *job shop*

y_{ij} značí startovní čas operace (i, j) úlohy j na stroji i

Minimalizace C_{max}

za předpokladu

$$
\begin{aligned}
y_{kj} - y_{ij} &\geq p_{ij} && \forall (i,j) \to (k,j) \in A \\
C_{max} - y_{ij} &\geq p_{ij} && \forall (i,j) \in N \\
y_{ij} - y_{il} \geq p_{il} \text{ nebo } y_{il} - y_{ij} &\geq p_{ij} && \forall (i,l),(i,j): \\
& && i = 1 \ldots m
\end{aligned}
$$

Disjunktivní programování a *job shop*

y_{ij} značí startovní čas operace (i, j) úlohy j na stroji i

Minimalizace C_{max}

za předpokladu

$$
\begin{aligned}
y_{kj} - y_{ij} &\geq p_{ij} && && \forall (i,j) \to (k,j) \in A \\
C_{max} - y_{ij} &\geq p_{ij} && && \forall (i,j) \in N \\
y_{ij} - y_{il} &\geq p_{il} && \text{nebo } y_{il} - y_{ij} \geq p_{ij} && \forall (i,l),(i,j): \\
& && && \quad i = 1 \ldots m \\
y_{ij} &\geq 0 && && \forall (i,j) \in N
\end{aligned}
$$

Disjunktivní programování a *job shop*

y_{ij} značí startovní čas operace (i,j) úlohy j na stroji i

Minimalizace C_{max}

za předpokladu

$$
\begin{aligned}
y_{kj} - y_{ij} &\geq p_{ij} && && \forall (i,j) \to (k,j) \in A \\
C_{max} - y_{ij} &\geq p_{ij} && && \forall (i,j) \in N \\
y_{ij} - y_{il} &\geq p_{il} && \text{nebo } y_{il} - y_{ij} \geq p_{ij} && \forall (i,l),(i,j): \\
& && && \quad i = 1 \ldots m \\
y_{ij} &\geq 0 && && \forall (i,j) \in N
\end{aligned}
$$

- Tato formulace ale nezajišťuje existenci standardní řešící procedury
- Problém velmi obtížný
- Ukážeme další přístupy: enumerační procedury (BB),
  heuristiky (posunování kritického místa)

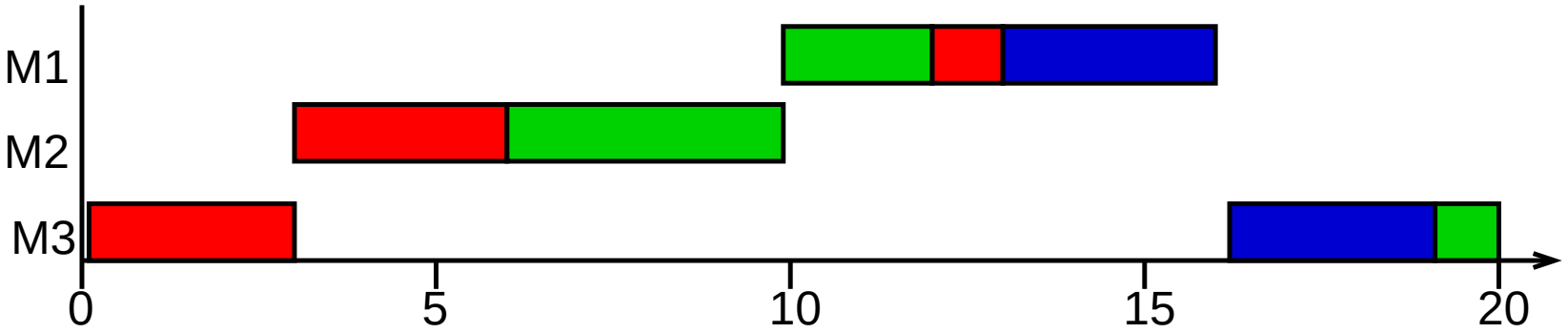
Typy rozvrhů

- Rozvrh je bez zdržení *(nondelay)*, pokud žádný stroj nečeká, když existuje dostupná operace

- Rozvrh je aktivní, pokud nemůže být žádná operace rozvrhována dříve změnou posloupnosti úloh na stroji bez zdržení jiné operace
  - redukce *makespan* aktivního rozvrhu je možná pouze zvýšením startovního času alespoň jedné operace
  - optimální rozvrh je aktivní rozvrh

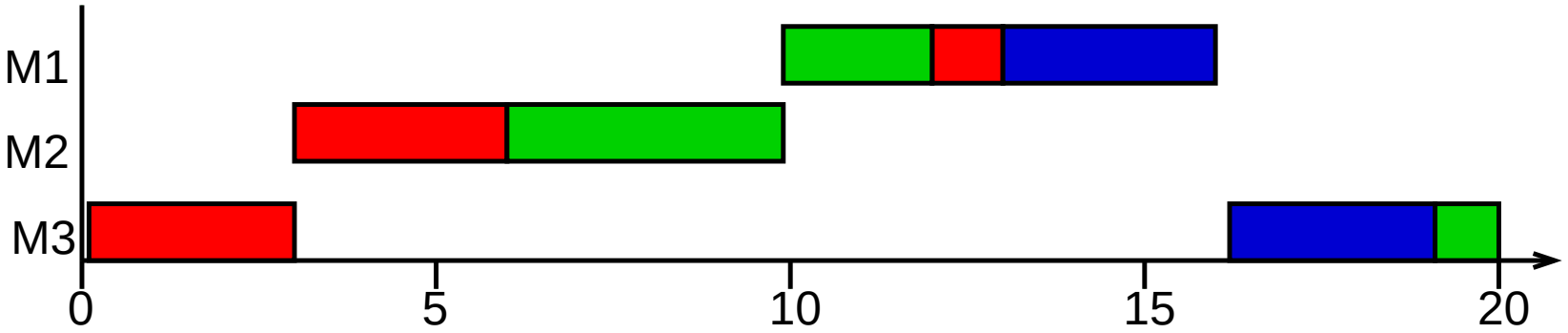
Příklad: aktivní vs. neaktivní rozvrh

Neaktivní rozvrh:

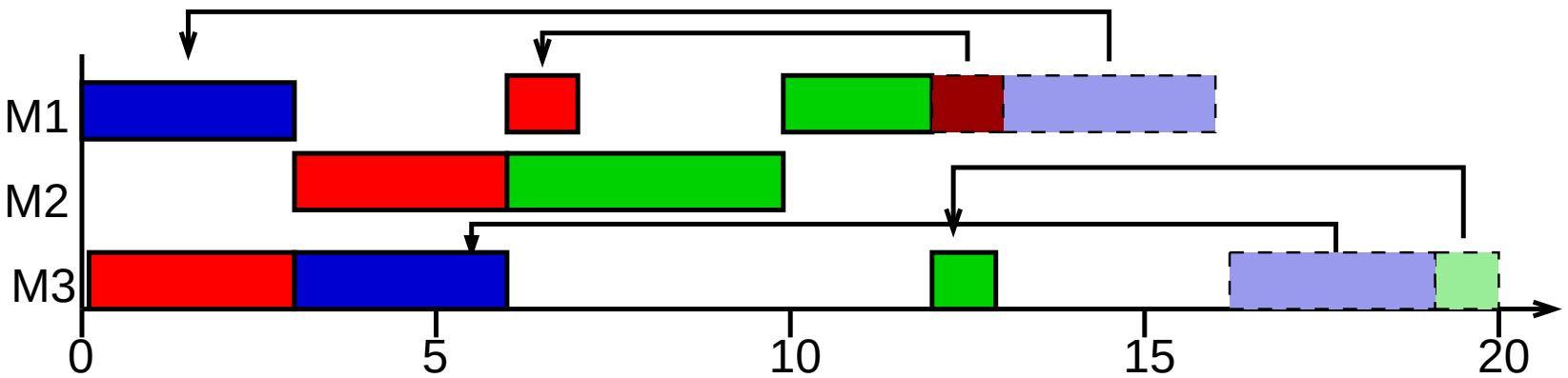


Příklad: aktivní vs. neaktivní rozvrh

Neaktivní rozvrh:



Aktivní rozvrh:



Aplikace metody větví a mezí na *job shop*

- Víme: optimální rozvrh je aktivní rozvrh
- Metoda řešení
  - generování množiny aktivních rozvrhů
  - výběr nejlepšího rozvrhu
- Zlepšení
  - použití metody větví a mezí při generování
- Důsledek
  - potřebujeme algoritmus pro generování všech aktivních rozvrhů

Aplikace metody větví a mezí na *job shop*

- Víme: optimální rozvrh je aktivní rozvrh
- Metoda řešení
  - generování množiny aktivních rozvrhů
  - výběr nejlepšího rozvrhu
- Zlepšení
  - použití metody větví a mezí při generování
- Důsledek
  - potřebujeme algoritmus pro generování všech aktivních rozvrhů
- Značení
  - Ω: množina všech operací, jejichž předchůdci už jsou narozvrhováni
  - r_{ij}: nejdřívější startovní čas operace $(i, j) \in \Omega$
    - může být spočítano pomocí výpočtu nejdelší cesty
  - Ω': podmnožina Ω

Generování množiny všech aktivních rozvrhů

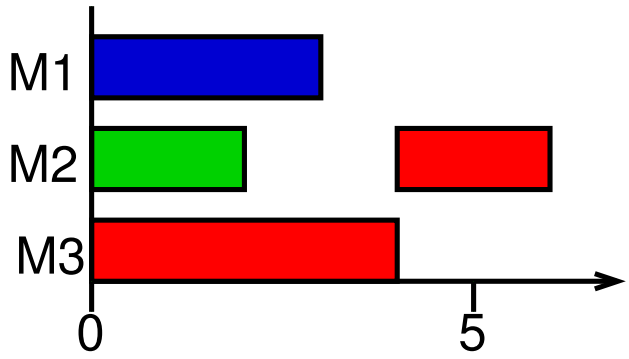
1. Inicializace
   - $\Omega := \{\text{první operace každé úlohy}\}$
   - $r_{ij} := 0$ pro všechna $(i,j) \in \Omega$
2. Výběr stroje
   - spočítej pro současný částečný rozvrh

$$t(\Omega) := \min_{(i,j)\in\Omega} \{r_{ij} + p_{ij}\}$$

     tj. kdy nejdřívě může nějaká úloha z Ω skončit
   - $i^* :=$ stroj, na němž bylo dosaženo minima
3. Větvení
   - $\Omega' := \{(i^*,j) | r_{i^*j} < t(\Omega)\}$
   - pro všechna $(i^*,j) \in \Omega'$
     přidej do rozvrhu (i^*,j) jako další operaci na stroji i^*
     smaž (i^*,j) z Ω
     přidej následníky úlohy (i^*,j) do Ω
     běž na krok 2

Příklad: generování aktivních rozvrhů

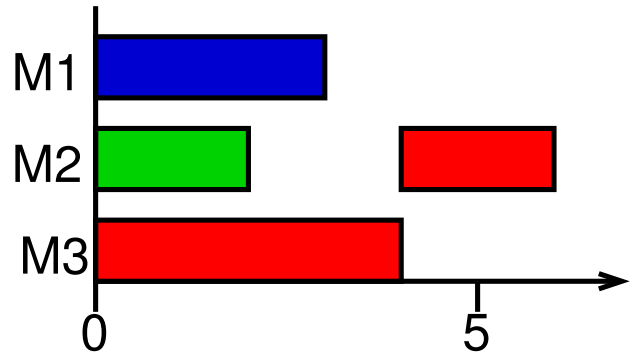
- Úlohy: $J1 : (3,1) \rightarrow (2,1) \rightarrow (1,1)$
  $J2 : (1,2) \rightarrow (3,2)$
  $J3 : (2,3) \rightarrow (1,3) \rightarrow (3,3)$
- Částečný rozvrh:
  - neřešíme od začátku,
    začneme řešit už s tímto rozvrhem,
    aby byl postup demonstrativní



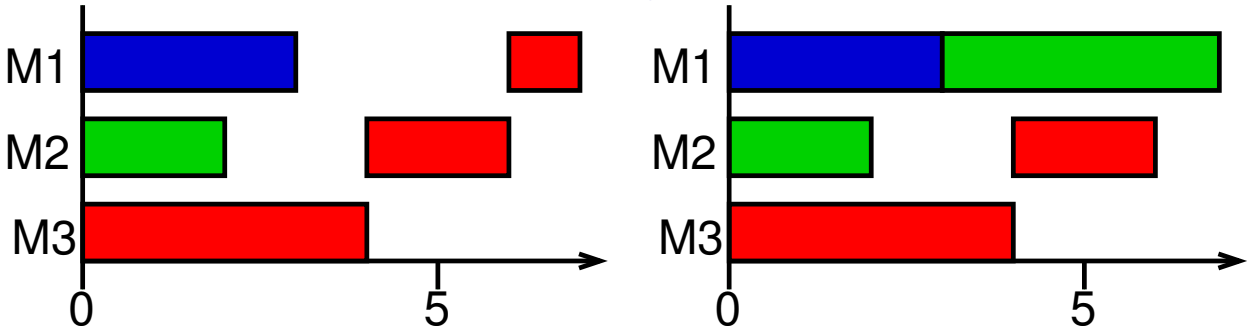
- $\Omega = \{(1,1),(3,2),(1,3)\}$
  - $p11 = 1, p32 = 3, p13 = 4 \quad r11 = 6, r32 = 4, r13 = 3$
- $t(\Omega) = \min(6+1, 4+3, 3+4) = 7$ např. $i^* = M1$

Příklad: generování aktivních rozvrhů

- Úlohy: $J1 : (3,1) \to (2,1) \to (1,1)$
  $J2 : (1,2) \to (3,2)$
  $J3 : (2,3) \to (1,3) \to (3,3)$
- Částečný rozvrh:
  - neřešíme od začátku,
    začneme řešit už s tímto rozvrhem,
    aby byl postup demonstrativní



- $\Omega = \{(1,1),(3,2),(1,3)\}$
  - $p11 = 1, p32 = 3, p13 = 4 \quad r11 = 6, r32 = 4, r13 = 3$
- $t(\Omega) = \min(6+1, 4+3, 3+4) = 7$ např. $i^* = M1$
- $\Omega' = \{(1,1),(1,3)\}$
- Rozšířené částečné rozvrhy:



Disjunkce vybrané při větvení

- Větvení algoritmu volí disjunkce
- Předpokládejme větvení $\Omega' = \{(i^*, j), (i^*, l)\}$

  $\rightarrow$ výběr (i^*, j) při větvení

  přidání disjunkce $(i^*, j) \rightarrow (i^*, k)$ pro všechny dosud nenarozvrhované operace (i^*, k)

  $\rightarrow$ výběr (i^*, l) při větvení

  přidání disjunkce $(i^*, l) \rightarrow (i^*, k)$ pro všechny dosud nenarozvrhované operace (i^*, k)

- Důsledek:
  - každý uzel stromu je charakterizován množinou D' vybraných disjukcí

Výpočet dolní hranice

Předpokládejme uzel V s vybranými disjukcemi D'

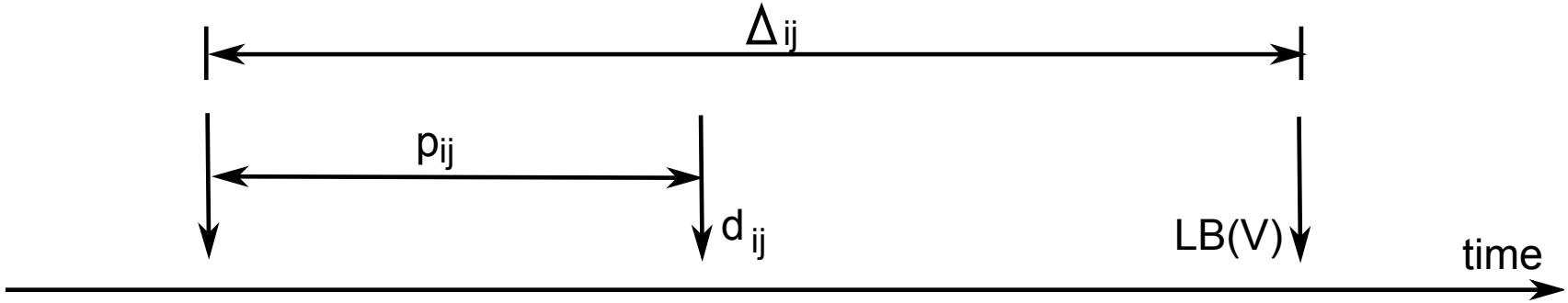
- Jednoduchá dolní hranice
  - spočítej kritickou cestu v $G(D')$
    dolní hranice $LB(V)$
- Vylepšená dolní hranice
  - uvažuj stroj i
  - povolíme překrývání operací na všech strojích kromě i
  - vyřeš problém na stroji i

Podproblém: $1|r_j|L_{max}$

- Vypočítej nejdřívější startovní čas r_{ij} všech operací (i, j) na stroji i
  - nejdelší cesta ze zdroje v $G(D')$
- Vypočítej minimální množství času Δ_{ij} mezi:

  startem operace (i, j) (tj. r_{ij}) a

  koncem rozvrhu (nejdelší cesta v $G(D')$ z uzlu do stoku)
- Získáme termíny dokončení $d_{ij} = LB(V) - \Delta_{ij} + p_{ij}$



- Vyřeš výsledný problém: $1|r_j|L_{max}$
  - viz dříve

Vylepšená dolní hranice

- Vyřeš $1|r_j|L_{max}$ pro všechny stroje
- Výsledek: $L_1, \ldots, L_m$

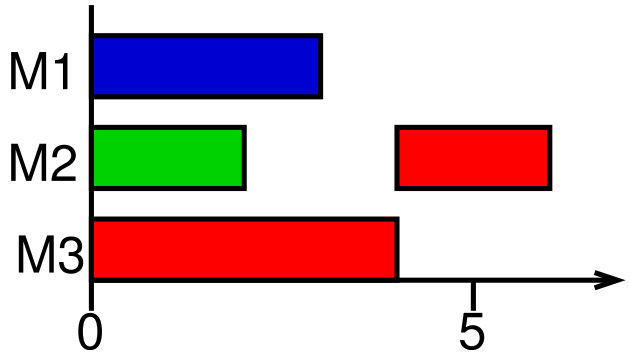
$$LB^{new}(V) = LB(V) + \max_{i=1\ldots m} L_i$$

tj. výsledné řešení nemůže mít lepší kvalitu než nejlepší možné řešení pro každý stroj zvlášť, a proto zahrneme do dolní hranice nejhorší (největší) spočítané zpoždění

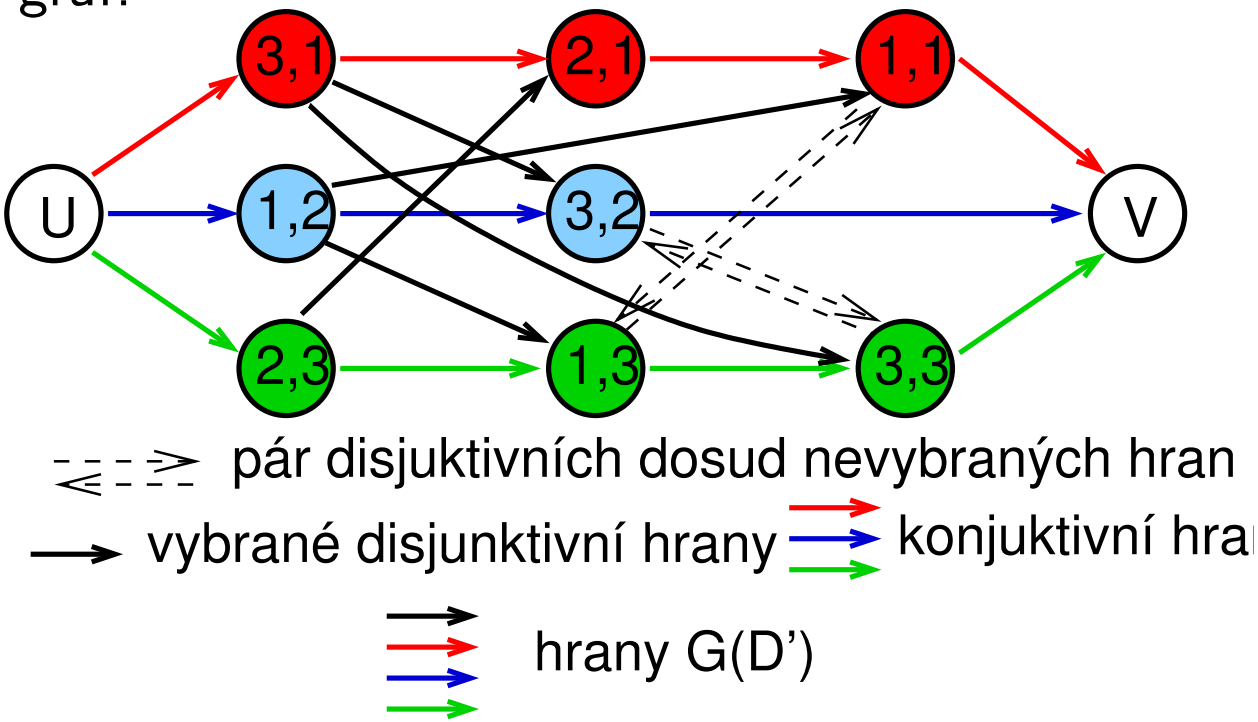
- Poznámky:
  - $1|r_j|L_{max}$ je NP-úplný problém
  - experimentální výsledky přesto ukazují, že se vyplatí řešit m NP-úplných problémů pro každý uzel stromu
  - 20×20 instance jsou už obtížně řešitelné metodou větví a mezí

Příklad: dolní hranice

Částečný rozvrh:

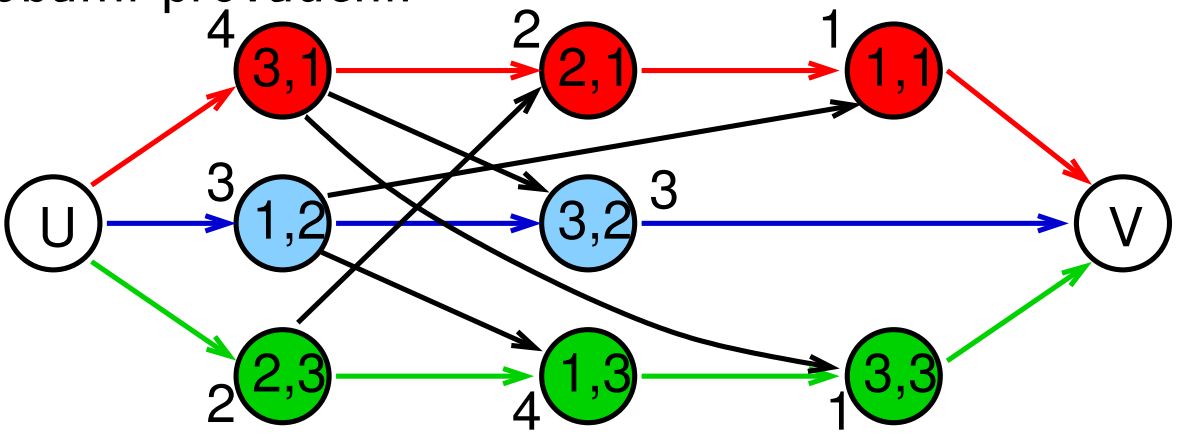


Odpovídající graf:



Příklad: dolní hranice

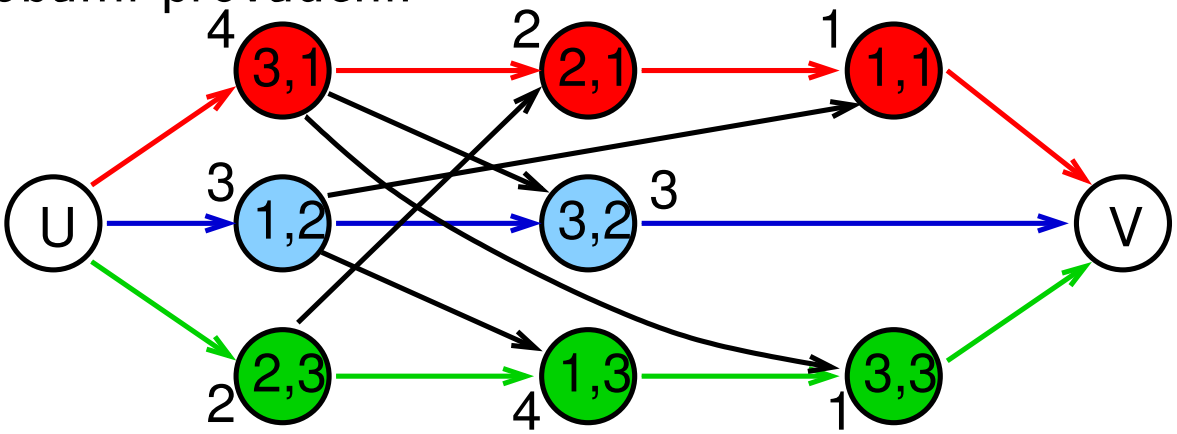
Graf $G(D')$ s dobami provádění:



$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



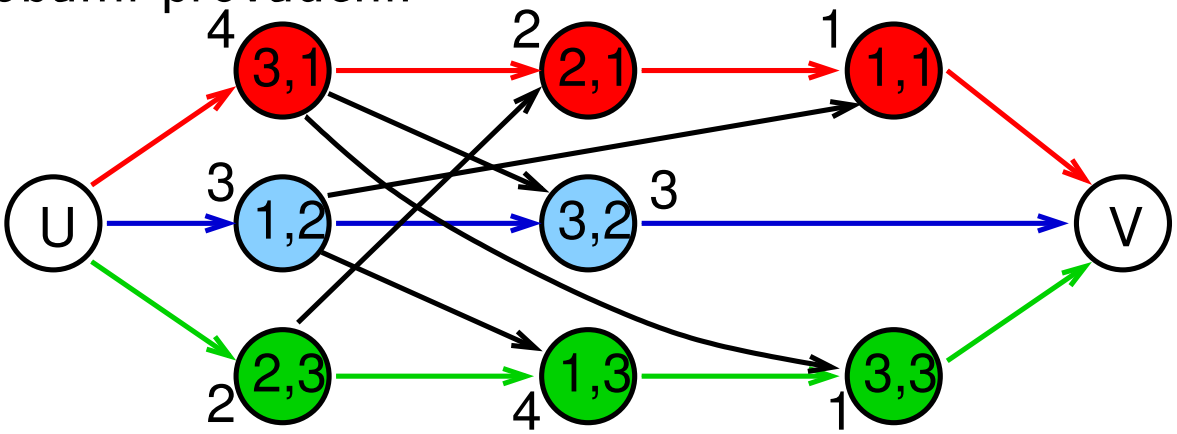
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

	modrá	zelená	červená
r_{12}			

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



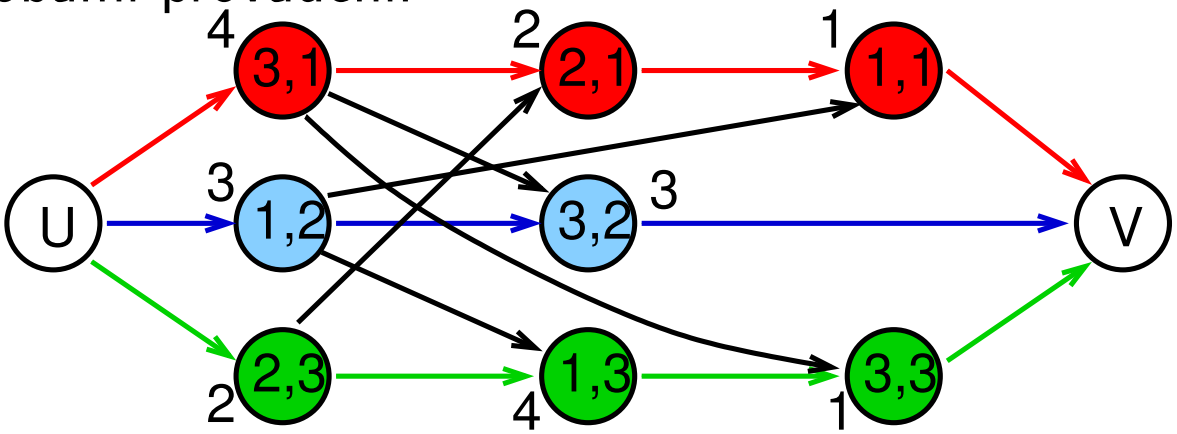
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$		

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



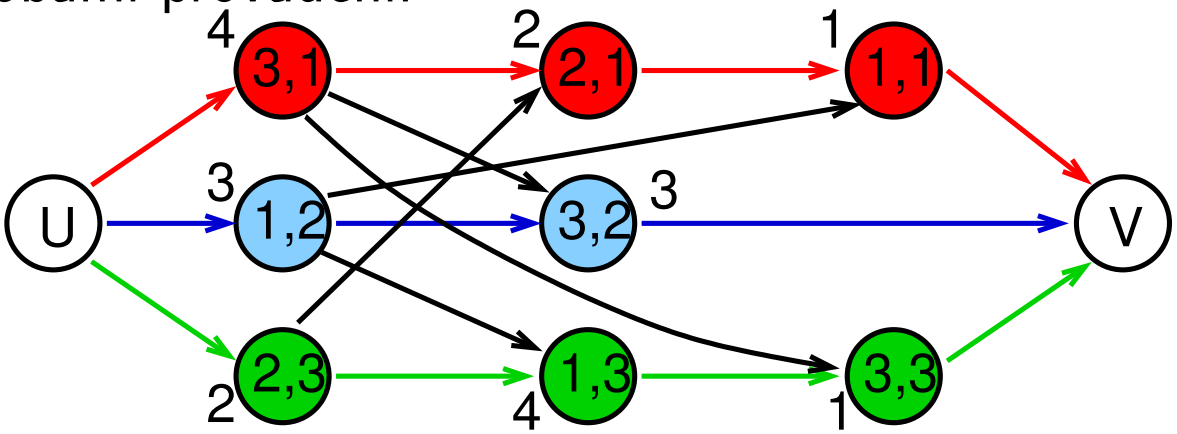
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} =$	

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



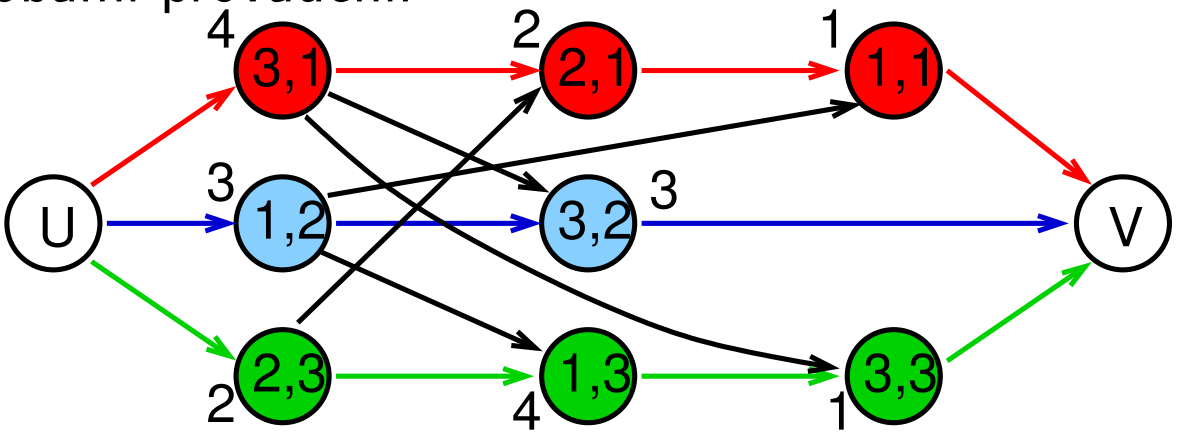
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



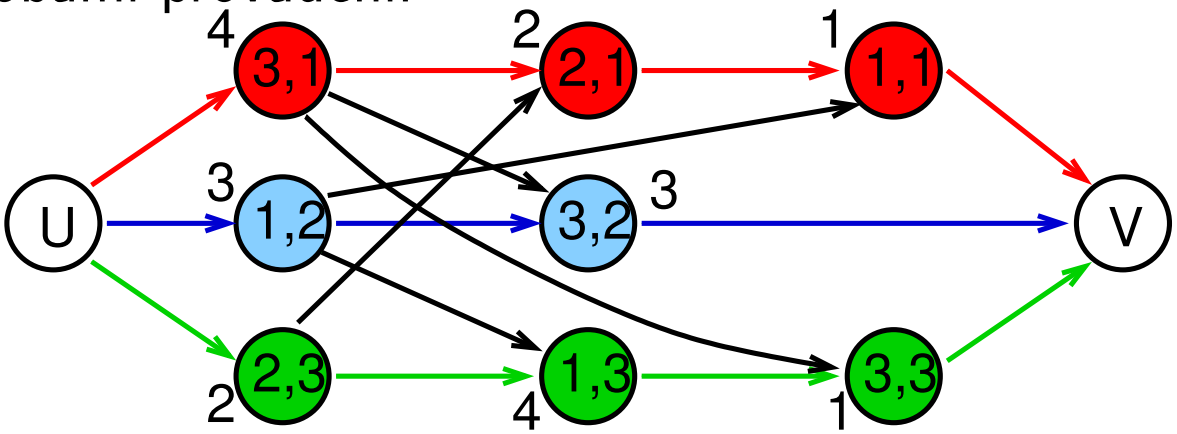
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	r_{11}

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



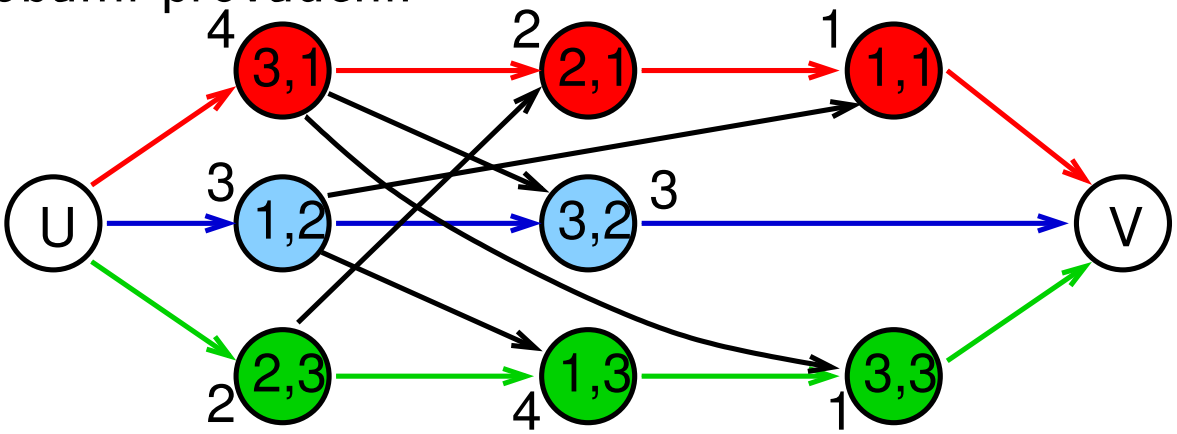
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



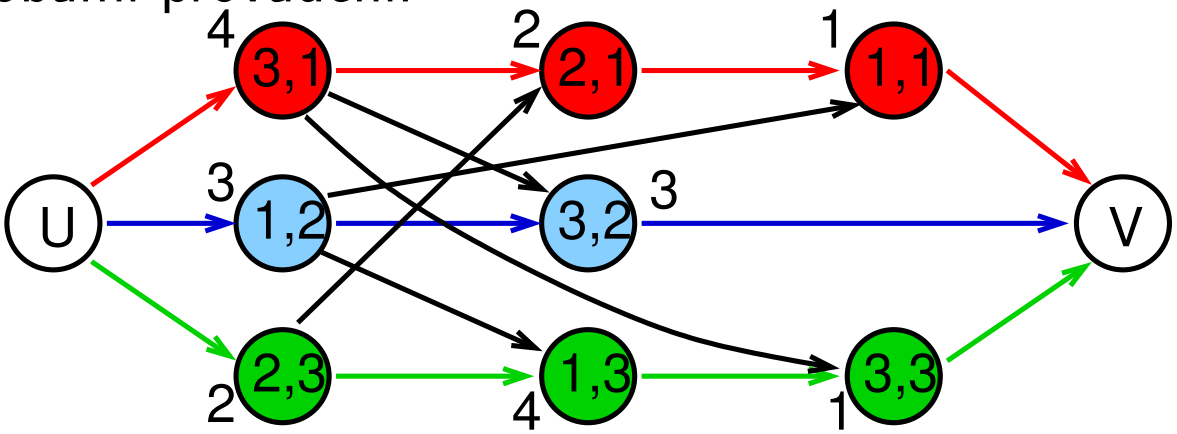
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

	modrá	zelená	červená
	$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
Δ_{12}			

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



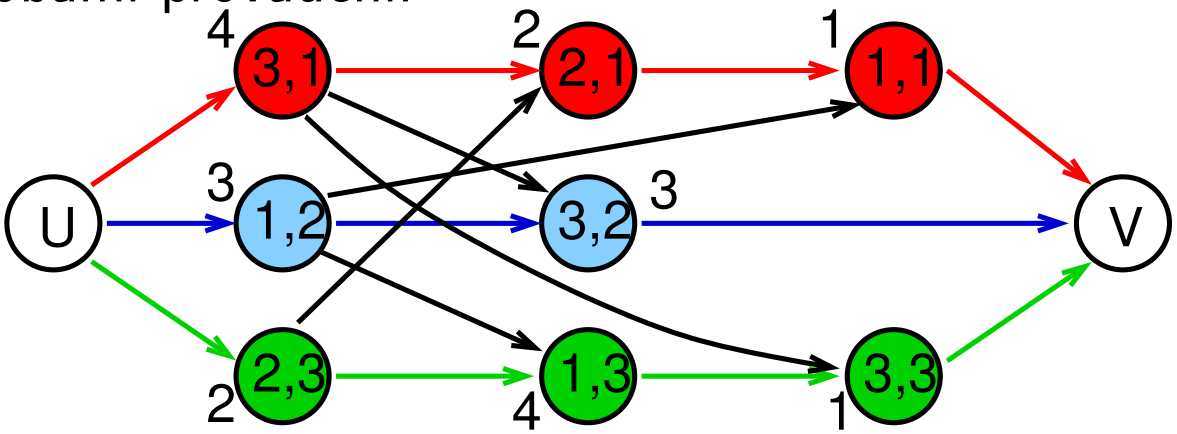
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$		

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



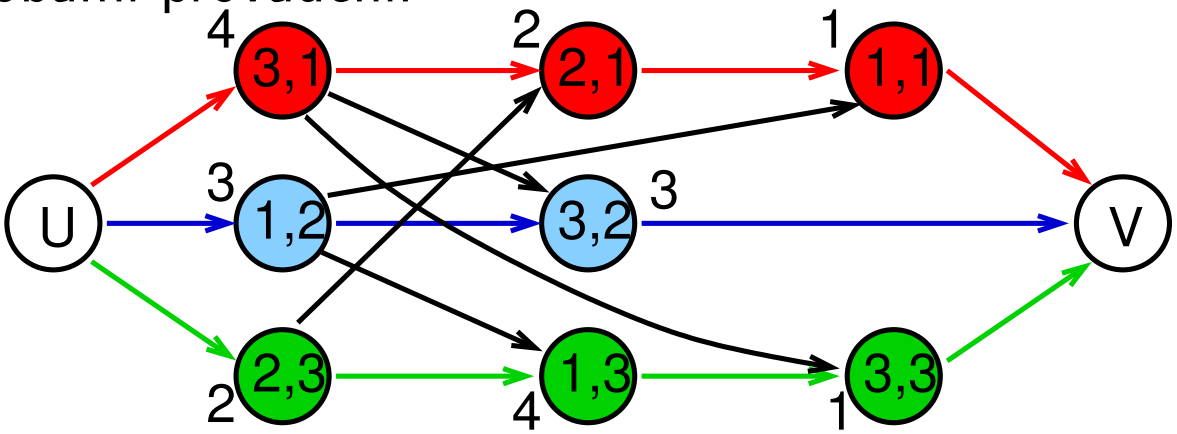
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	Δ_{13}	

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



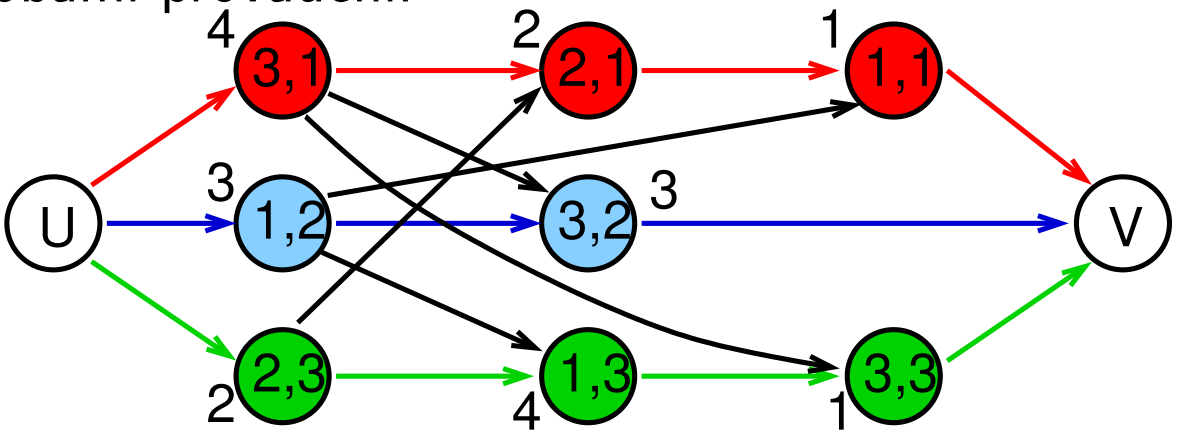
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



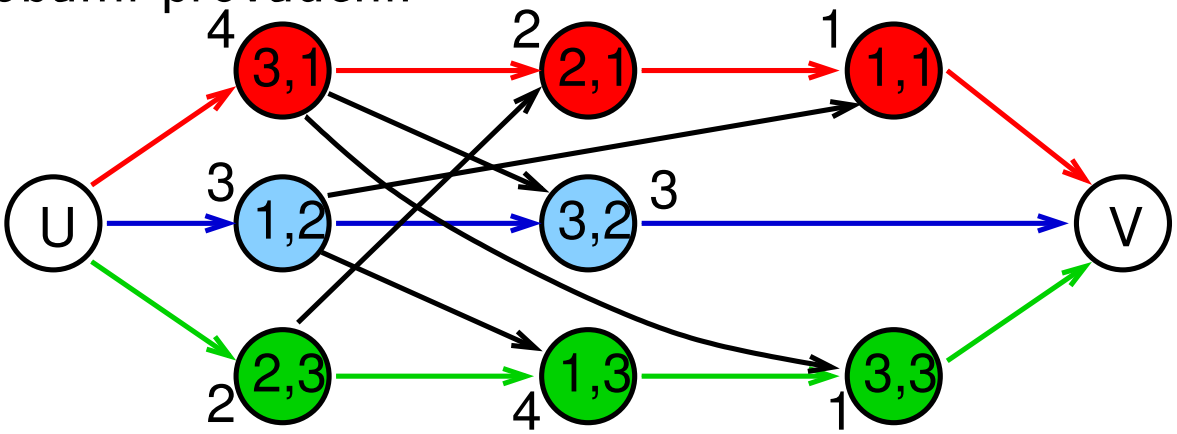
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	Δ_{11}

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



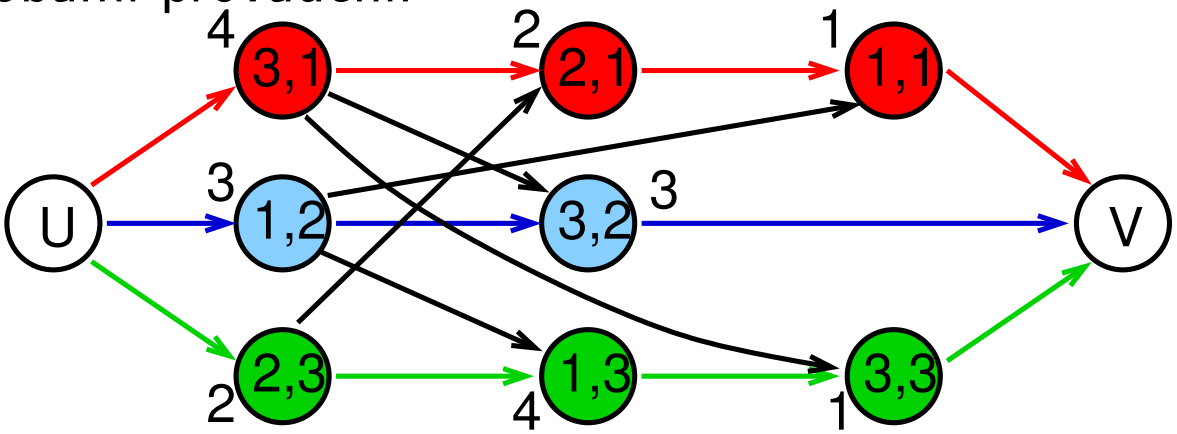
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	$\Delta_{11} = 1$

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

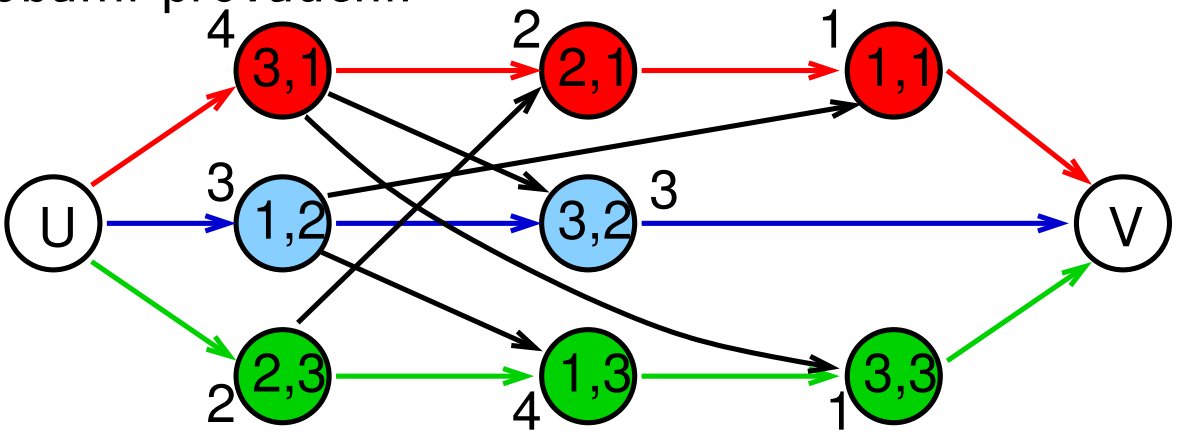
Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	$\Delta_{11} = 1$
d_{12}		

(víme: $d_{ij} = LB(V) - \Delta_{ij} + p_{ij}$)

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

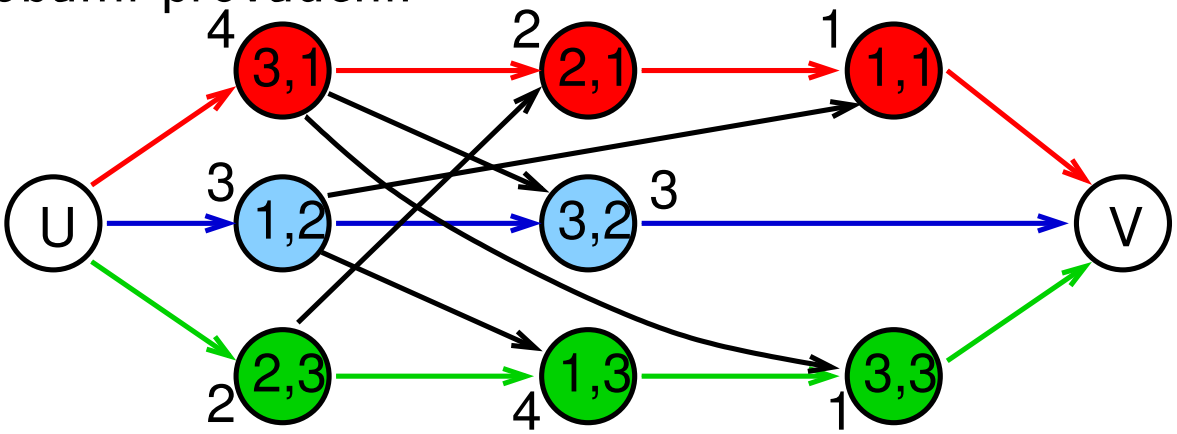
Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	$\Delta_{11} = 1$
$d_{12} = 3$		

(víme: $d_{ij} = LB(V) - \Delta_{ij} + p_{ij}$)

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

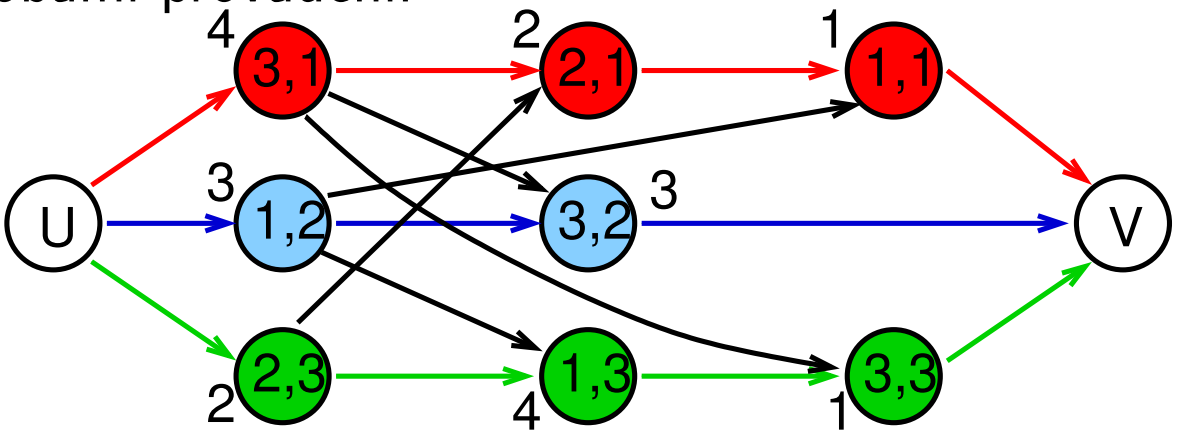
Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	$\Delta_{11} = 1$
$d_{12} = 3$	d_{13}	

(víme: $d_{ij} = LB(V) - \Delta_{ij} + p_{ij}$)

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

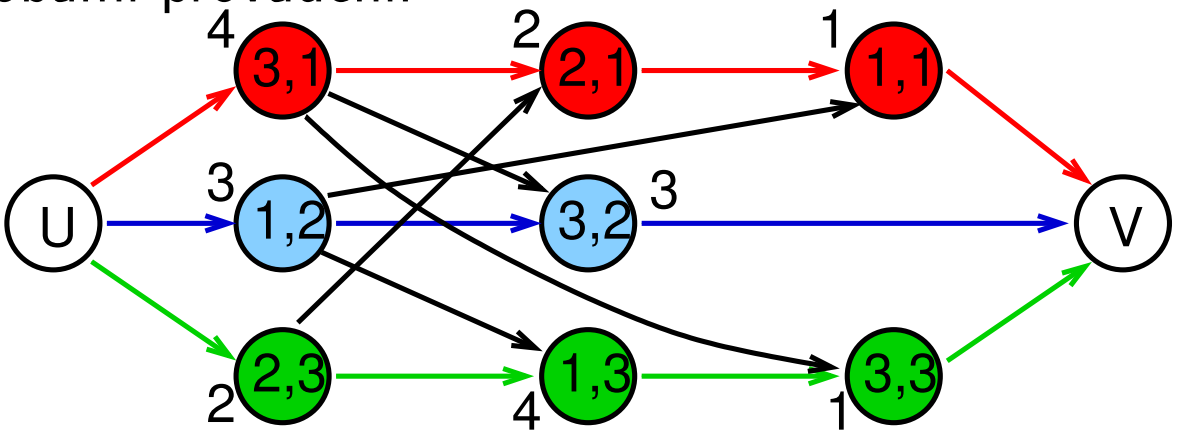
Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	$\Delta_{11} = 1$
$d_{12} = 3$	$d_{13} = 7$	

(víme: $d_{ij} = LB(V) - \Delta_{ij} + p_{ij}$)

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

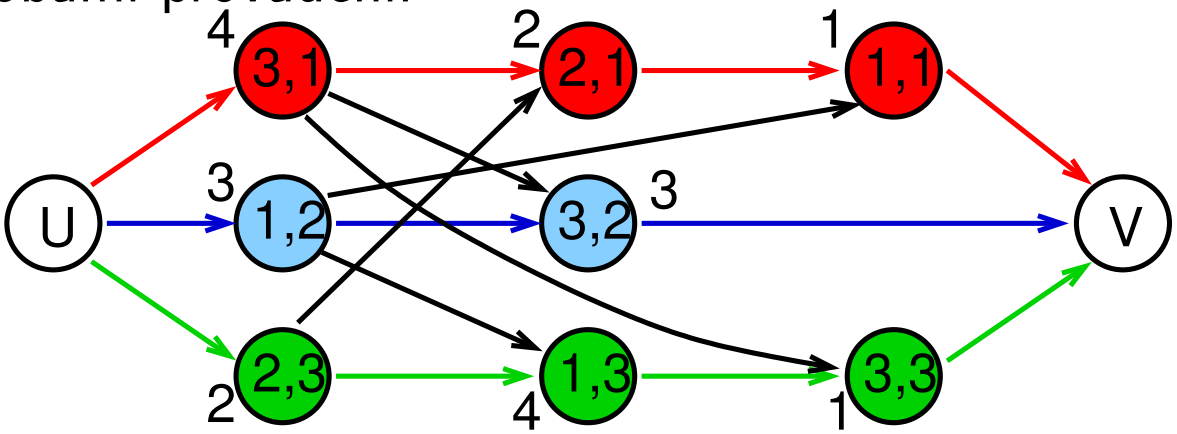
Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	$\Delta_{11} = 1$
$d_{12} = 3$	$d_{13} = 7$	d_{11}

(víme: $d_{ij} = LB(V) - \Delta_{ij} + p_{ij}$)

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

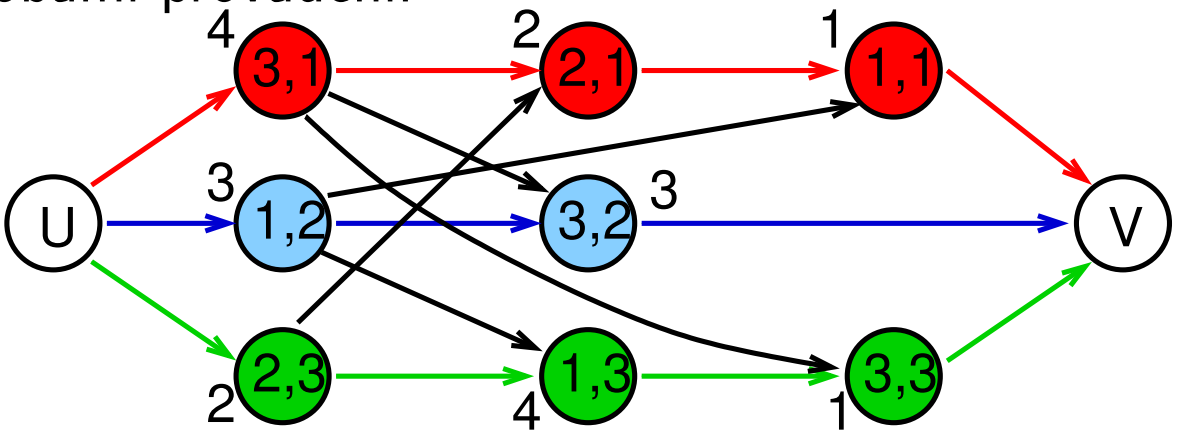
Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	$\Delta_{11} = 1$
$d_{12} = 3$	$d_{13} = 7$	$d_{11} = 8$

(víme: $d_{ij} = LB(V) - \Delta_{ij} + p_{ij}$)

Příklad: dolní hranice

Graf $G(D')$ s dobami provádění:



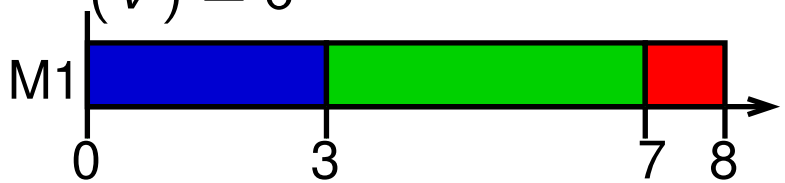
$LB(V) = l(U, (1,2), (1,3), (3,3), V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	$\Delta_{11} = 1$
$d_{12} = 3$	$d_{13} = 7$	$d_{11} = 8$

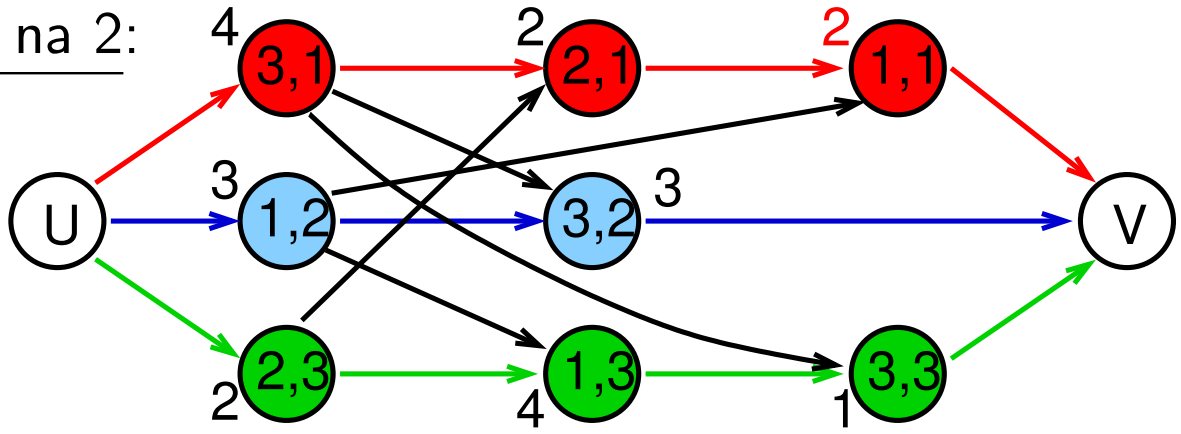
(víme: $d_{ij} = LB(V) - \Delta_{ij} + p_{ij}$)

Optimální řešení: $L_{max} = 0, LB^{new}(V) = 8$



Příklad: dolní hranice

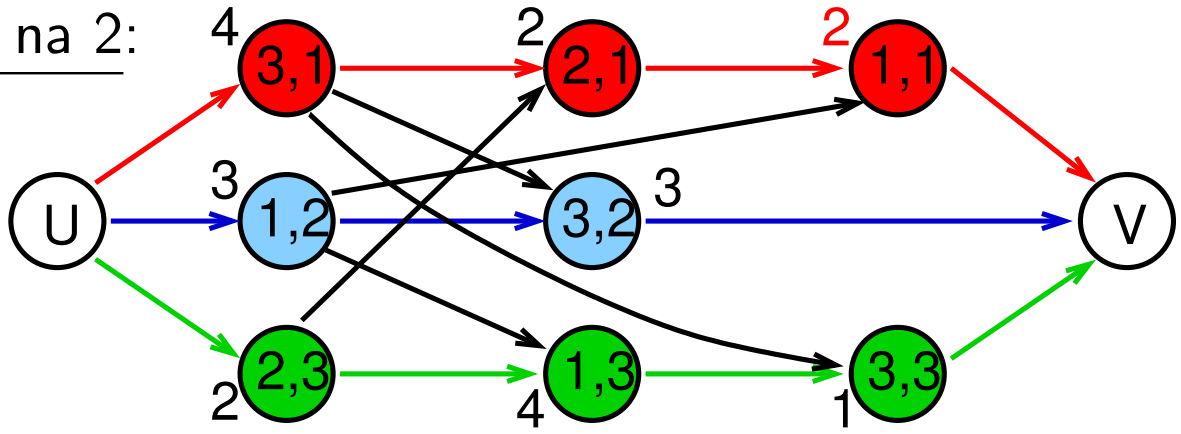
Změna $p11$ z 1 na 2:



$$LB(V) = l(U,(1,2),(1,3),(3,3),V)$$
$$= l(U,(3,1),(2,1),(1,1),V) = 8$$

Příklad: dolní hranice

Změna $p11$ z 1 na 2:



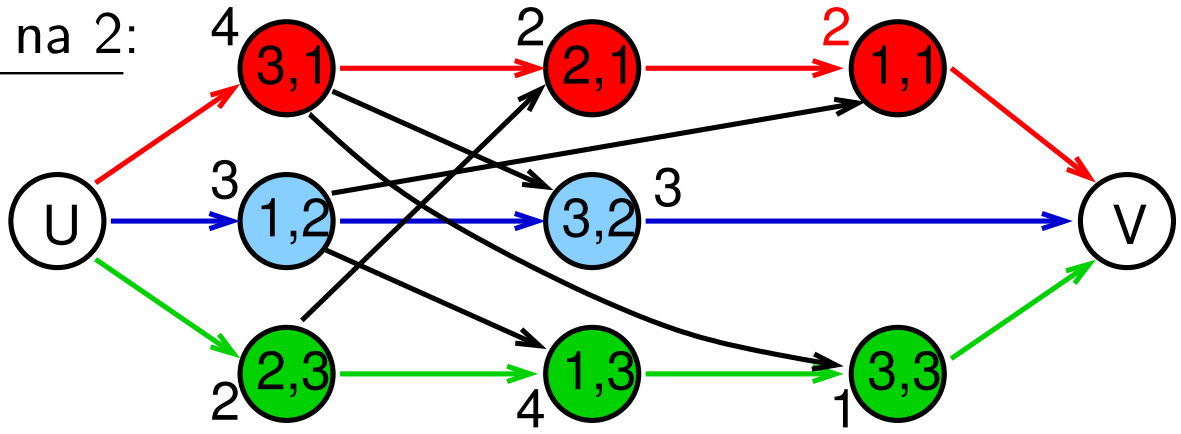
$$LB(V) = l(U,(1,2),(1,3),(3,3),V)$$
$$= l(U,(3,1),(2,1),(1,1),V) = 8$$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	$\Delta_{11} = \underline{2}$
$d_{12} = 3$	$d_{13} = 7$	$d_{11} = 8$

Příklad: dolní hranice

Změna $p11$ z 1 na 2:

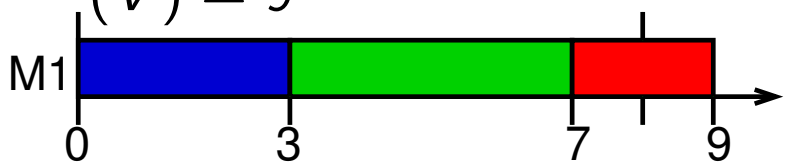


$LB(V) = l(U,(1,2),(1,3),(3,3),V)$
$= l(U,(3,1),(2,1),(1,1),V) = 8$

Data pro úlohy na stroji $M1$:

modrá	zelená	červená
$r_{12} = 0$	$r_{13} = 3$	$r_{11} = 6$
$\Delta_{12} = 8$	$\Delta_{13} = 5$	$\Delta_{11} = 2$
$d_{12} = 3$	$d_{13} = 7$	$d_{11} = 8$

Optimální řešení: $L_{max} = 1, LB^{new}(V) = 9$



Posunování kritického místa *(shifting bottleneck)*

- Úspěšná heuristika

  při řešení problému minimalizace *makespan* pro *job shop*

- Základní popis
  - iterativní heuristika
  - v každé iteraci je určen rozvrh pro jeden další stroj
  - používána re-optimalizace pro změnu už narozvrhovaných strojů

- Rozšiřitelnost: metoda lze rozšířit
  na obecnější *job shop* problémy
  - další objektivní funkce
  - *flexible flow shop*
    (paralelní stroj místo samostatných strojů)
  - nastavovací doba stroje

Princip algoritmu

- Značení
  - M je množina všech strojů
- Dáno
  - určen rozvrh pro podmnožinu $M_0 \subset M$ strojů
    - tj. je určen výběr disjunktivních hran
- Akce při jedné iteraci
  1. výběr stroje k, pro který ještě neexistuje rozvrh
     - tj. stroj z $M \setminus M_0$
  2. určení rozvrhu (výběru disjunktivních hran) pro stroj k na základě daných (zafixovaných) rozvrhů pro stroje z M_0
  3. nové rozvrhování (= přeplánování) strojů z M_0 na základě ostatních daných (zafixovaných) rozvrhů

Princip výběru stroje a určení jeho rozvrhu

- Myšlenka:
  - výběr ještě nerozvrženého stroje, který působí nejvíce problémů, tzv. kritický (*bottleneck*) stroj
- Realizace:
  - spočítej pro každou operaci na nenarozvrhovaném stroji
    - nejdřívější možný startovní čas a
    - minimální zdržení mezi koncem operace a koncem úplného rozvrhu

    založeného na
    - zafixovaných rozvrzích strojů v M_0 a
    - pořadí úloh
  - spočítej pro každý nenarozvrhovaný stroj rozvrh respektující tyto nejdřívější termíny dostupnosti a zdržení
  - vyber stroj s maximálním koncovým časem a zafixuj rozvrh na tomto stroji

Princip přeplánování strojů

- Myšlenka:
  - snaha redukovat *makespan* rozvrhu pro stroje v M_0
- Popis:
  - uvažuj stroje z M_0 jeden po druhém
  - smaž rozvrh vybraného stroje a spočítej nový rozvrh na základě
    - nejdřívějšího startovního času a
    - zdržení

    vyplývající z
    - ostatních strojů v M_0 a
    - pořadí úloh

Plánování job-shopu: shrnutí

Modelování a reprezentace

- disjunktivní grafová reprezentace
- matematické programování a job shop (+ řešení)

Řešení

- metoda větví a mezí pro job shop
- heuristika: posunování kritického místa

Rozvrhování montážních systémů

16. května 2022

Montážní linka

Job shop

- každá úloha má jednoznačnou identitu
- výroba na objednávku, malý objem výroby
- potenciálně komplikovaná cesta systémem
- velmi obtížný problém

Montážní linka
(flexible assembly system)

- limitovaný počet odlišných typů výrobků
- specifikováno vyráběné množství každého typu
- systém pro manipulaci s materiálem
  - startovní čas úlohy je funkcí času dokončení na předchozím stroji
  - limitovaný počet úloh čekajících ve frontě mezi stroji
- masová produkce
- ještě obtížnější problém

Montážní linka s flexibilním časem *(unpaced assembly system)*

- Několik strojů zapojených sériově *(flow system)*
- Flexibilní čas
  - stroj může strávit tolik času na úloze, kolik je třeba
- Blokování
  - následující stroj je plný a úloha na něj nemůže být přesunuta
- Fronty mezi stroji?
  - bez újmy na obecnosti lze uvažovat fronty nulové délky
    - frontu délky n lze simulovat n stroji, na kterých je doba provádění úlohy nulová

  $\Rightarrow$ budeme uvažovat fronty nulové délky
- Limitovaný počet odlišných typů výrobků
- Specifikováno vyráběné množství každého typu
- Cíl: maximalizace výkonu
  - výkon opět definován kritickým strojem

Příklad: výroba kopírek

- Různé typy kopírek montované na jedné lince
- Odlišné modely mají obvykle společný základ
- Odlišnost v rámci komponent
  - automatický podavač ano/ne, rozdílné optiky, ...
- Odlišnost typů vede k odlišným dobám výroby na jednotlivých strojích

Cyklické rozvrhy

- Rozvrhy jsou často cyklické nebo periodické
  - daná množina úloh rozvrhována v určitém pořadí
    - jsou obsaženy všechny typy výrobků
    - některé typy zde mohou být vícekrát
  - druhá identická množina rozvrhována, atd.
- Nevhodné pokud jsou dlouhé nastavovací doby
  - pak se vyplatí dlouhé běhy výrobku jednoho typu
- Praktické pokud nevýznamná nastavovací doba
  - nízké skladovací náklady
  - snadné na implementaci
  - nicméně: acyklický rozvrh může dávat maximální výkon
- V praxi
  - cyklické rozvrhy s malými odchylkami závislými na aktuálních objednávkách

Minimální podílová množina *(minimum part set)*

- Předpokládejme l typů výrobků
- N_k požadovaný počet výrobků typu k
- z největší společný dělitel
- Potom

$$N^* = \left(\frac{N_1}{z}, \ldots, \frac{N_l}{z} \right)$$

je nejmenší množina se „správnými" proporcemi

minimální podílová množina
(MPS, minimum part set)

Minimální podílová množina *(minimum part set)*

- Předpokládejme l typů výrobků
- N_k požadovaný počet výrobků typu k
- z největší společný dělitel
- Potom

$$N^* = \left(\frac{N_1}{z}, \ldots, \frac{N_l}{z} \right)$$

je nejmenší množina se „správnými" proporcemi

minimální podílová množina
(MPS, minimum part set)

- Uvažujme úlohy v MPS jako n úloh: $n = \frac{1}{z} \sum_{k=1}^{l} N_k$
  - p_{ij} jako dříve
  - cyklický rozvrh je rozvrh určen seřazením úloh v MPS
  - maximalizace výkonu = minimalizace doby cyklu

Příklad: MPS

- Systém se 4 stroji
- Tři odlišné typy výrobku, které musí být vyráběny ve stejném množství, tj.

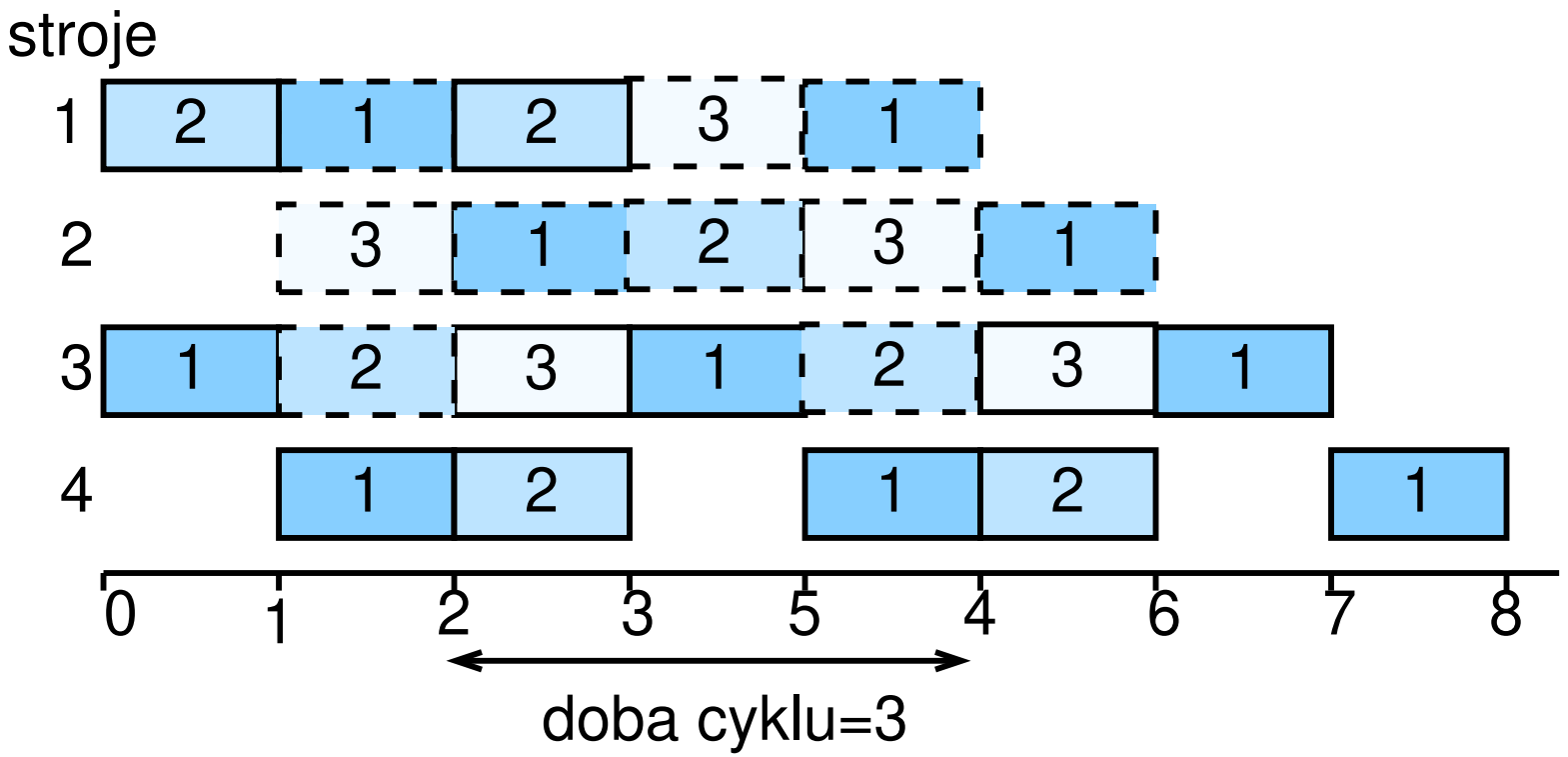
$$N^* = (1, 1, 1)$$

- Doby provádění

Úlohy	1	2	3	
p_{1j}	0	1	0	
p_{2j}	0	0	0	$\leftarrow$ fronta mezi strojem 1 a 3
p_{3j}	1	0	1	
p_{4j}	1	1	0	

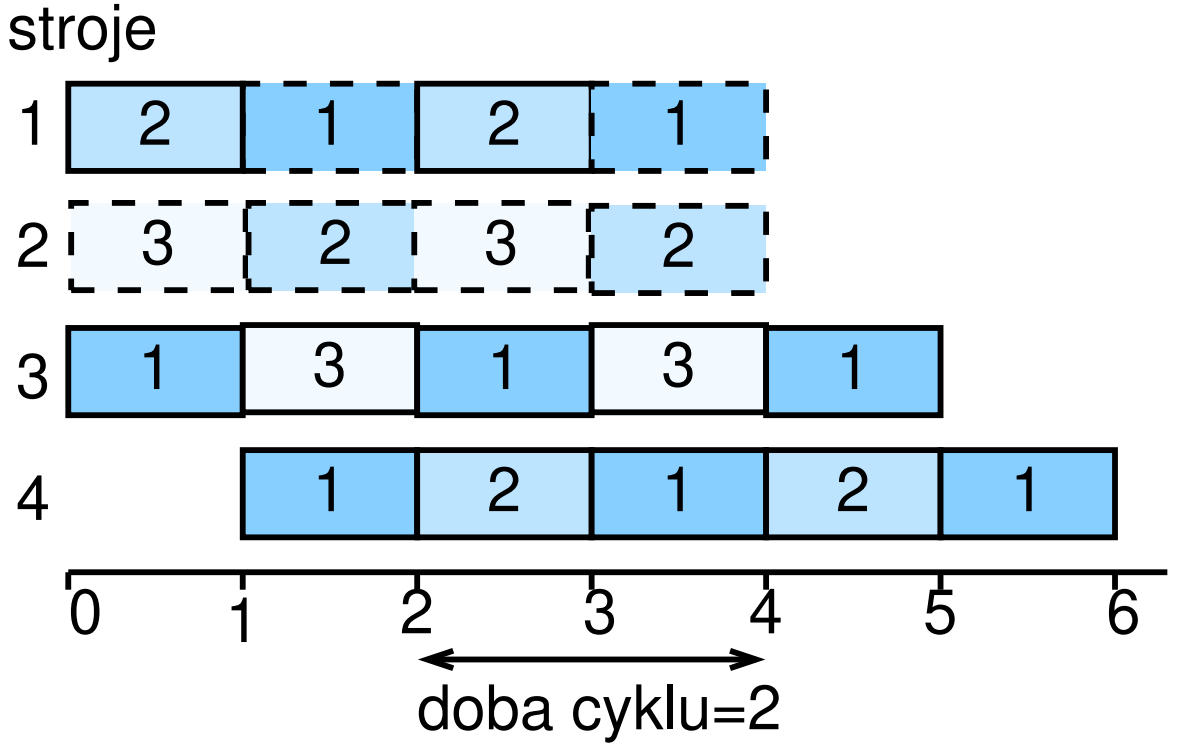
Příklad: posloupnost v MPS 1,2,3

Úlohy	1	2	3
p_{1j}	0	1	0
p_{2j}	0	0	0
p_{3j}	1	0	1
p_{4j}	1	1	0



Příklad: posloupnost v MPS 1,3,2

Úlohy	1	2	3
p_{1j}	0	1	0
p_{2j}	0	0	0
p_{3j}	1	0	1
p_{4j}	1	1	0



Minimalizace doby cyklu

Heuristika padnoucího profilu *(profile fitting heuristic, PF)*

- Vyber první úlohu j_1
  - výběr libovolné úlohy nebo
  - výběr úlohy s nejdelší celkovou dobou provádění
- Úloha generuje profil

$$X_{i,j_1} = \sum_{h=1}^{i} p_{h,j_1}$$

  - předpokládáme, že úloha j_1 poběží bez blokování

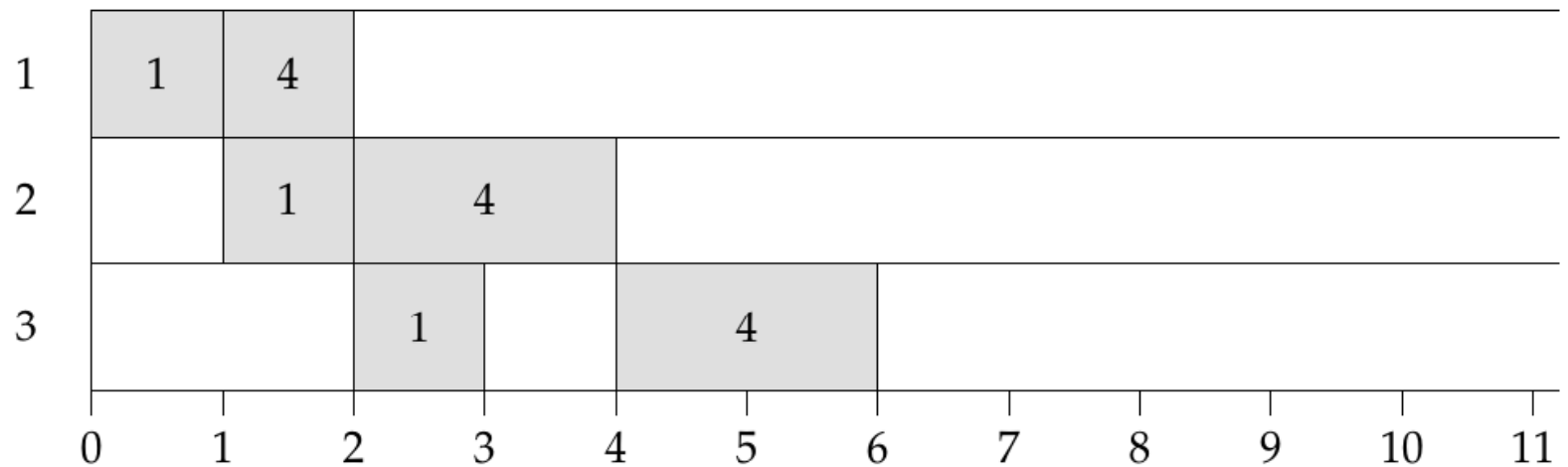
$\Rightarrow$ X_{i,j_1} čas odchodu: čas, kdy úloha j_1 opustí stroj i

PF: určení následující úlohy

Spočítej pro každou možnou úlohu

- dobu, kterou stroje čekají
- dobu, kdy je úloha blokována
- spočítej čas odchodu pro kandidáta na druhou pozici (j_2),
  např. pro úlohu c (j_1: úloha na první pozici)

$$
\begin{aligned}
X_{1,j_2} &= \max(X_{1,j_1} + p_{1c}, X_{2,j_1}) \\
X_{i,j_2} &= \max(X_{i-1,j_2} + p_{ic}, X_{i+1,j_1}) \quad i = 2, \ldots, m-1 \\
X_{m,j_2} &= X_{m-1,j_2} + p_{mc}
\end{aligned}
$$



PF: určení následující úlohy

Spočítej pro každou možnou úlohu

- dobu, kterou stroje čekají
- dobu, kdy je úloha blokována
- spočítej čas odchodu pro kandidáta na druhou pozici (j_2), např. pro úlohu c

$$\begin{aligned} X_{1,j_2} &= \max(X_{1,j_1} + p_{1c}, X_{2,j_1}) \\ X_{i,j_2} &= \max(X_{i-1,j_2} + p_{ic}, X_{i+1,j_1}) \quad i = 2, \ldots, m-1 \\ X_{m,j_2} &= X_{m-1,j_2} + p_{mc} \end{aligned}$$

- neproduktivní doba na stroji i, pokud je c na druhé pozici: $X_{i,j_2} - X_{i,j_1} - p_{ic}$
- celková neproduktivní doba (přes všechny stroje) pro c:

$$\sum_{i=1}^{m} (X_{i,j_2} - X_{i,j_1} - p_{ic})$$

- úloha s nejmenší neproduktivní dobou vybrána na druhou pozici (pro další pozice opakuj totéž) ... myšlenka padnoucího profilu

Diskuse

- PF heuristika se chová v praxi dobře
  - stále platí, že při použití heuristiky PF nehrají roli nastavovací doby
- Zjemnění:
  - neproduktivní doby nejsou stejně špatné na všech strojích
  - uvažuj kritický stroj
  - použití vah v součtu

Montážní linka s fixním časem

- Popis modelu
  - transportér, který se posunuje konstantní rychlostí
  - výrobky, které se vyrábí se posunují mezi stroji pevnou rychlostí
  - jednotková doba cyklu:
    určena jako doba mezi dvěma po sobě jdoucími úlohami na lince
  - každý stroj má danou kapacitu a omezení
  - cíl: uspořádat úlohy tak, aby
    - nebyly stroje přetíženy a
    - byla minimalizována cena za nastavení

Montážní linka s fixním časem

- Popis modelu
  - transportér, který se posunuje konstantní rychlostí
  - výrobky, které se vyrábí se posunují mezi stroji pevnou rychlostí
  - jednotková doba cyklu:
    určena jako doba mezi dvěma po sobě jdoucími úlohami na lince
  - každý stroj má danou kapacitu a omezení
  - cíl: uspořádat úlohy tak, aby
    - nebyly stroje přetíženy a
    - byla minimalizována cena za nastavení

- Příklad: výroba automobilu
  - rozdílné modely vyráběné na stejné lince
  - auta mají odlišné barvy a vybavení
  - vyráběná auta uspořádána tak, aby byla
    - minimalizována cena za nastavení a
    - stroje na lince byly rovnoměrně vytíženy

Skupiny a distribuce

- Atributy a charakteristiky každé úlohy
  - barva, vybavení, ...
- Cena za změny při výrobě
  - vytváření skupin výrobků, které mají vybrané atributy stejné
  - např. barva auta
- Časově náročné operace
  - distribuuj úlohy, které obsahují tyto operace
  - operace omezující kapacitu (index kritičnosti)
    - operace s vyšším indexem jsou kritičtější
  - např. instalace posuvné střechy
    - př. čtyřikrát časově náročnější, 10% aut má posuvnou střechu, tj. index kritičnosti = 4/10
    - sekce linky pro instalaci posuvné střechy musí být vytížena alespoň čtyřikrát méně než sekce pro automobily bez posuvné střechy, jinak nebude dostatek času na instalaci posuvné střechy

Kritéria

- Minimalizace celkové ceny na nastavení
- Splnění termínů dokončení pro výrobky na objednávku
  - celkové nezáporné vážené zpoždění
    - opakování: $T_j = \max(C_j - d_j, 0)$
- Distribuce operací omezujících kapacitu
  - $\psi_i(l)$ značí penalizaci, pokud stroj musí vyrábět dva výrobky, které jsou l pozic od sebe
- Pravidelné tempo spotřeby materiálu

Heuristika seskupování a distribuce *(grouping and spacing)*

1. Urči celkový počet úloh, které mají být rozvrhovány
   - vyšší počet úloh na rozvrhování umožní nižší cenu, ale snadněji dojde k narušení rozvrhu
   - typicky jeden den až týden
2. Seskup úlohy obsahující operace s vysokými cenami za nastavení
3. Uspořádej skupiny podle nastavovacích cen a termínů objednávky
4. Distribuuj úlohy v rámci skupiny tak,
   aby byly brány v úvahu operace omezující kapacitu
   - nejkritičtější operace distribuovány co nejlépe co nejdříve

   a zohledni přitom termíny objednávky

Jednoduchý matematický model

- 1 stroj, n úloh
- Každá úloha: $p_j = 1$, d_j (mohou být nekonečné), w_j, b atributů $a_{j1}, \ldots, a_{jb}$
  - 1. atribut reprezentuje barvu
  - 2. atribut je 1, pokud má úloha posuvnou střechu, jinak je 0
  - ...
- Jestliže úloha j následována úlohou k a $a_{j1} \neq a_{k1}$, pak je nutná cena na nastavení c_{jk}
  - c_{jk} je funkcí a_{j1} a a_{k1}
- Jestliže $a_{j2} = a_{k2} = 1$, pak je nutná penalizace $\psi_2(l)$
  - pokud jsou úlohy j a k od sebe vzdáleny l pozic
- Jestliže je úloha dokončena po termínu dokončení, uvažujeme vážené nezáporné zpoždění
- Cíl: minimalizovat celkovou cenu včetně
  - ceny za nastavení
  - ceny za distribuci
  - ceny za zpoždění

Příklad: heuristika seskupování a distribuce (zadání)

- 1 stroj, 10 úloh, $p_j = 1$
- Atributy úlohy j: a_{j1} a a_{j2}
- Cena za nastavení s využitím a_{j1} pro úlohu j:
  - dvě po sobě jdoucí úlohy mají $a_{j1} \neq a_{k1}$, pak $c_{jk} = |a_{j1} - a_{k1}|$
- $a_{j2} = a_{k2} = 1$ a mezi j a k je $(l-1)$ úloh, pak $\psi_2(l) = \max(3 - l, 0)$

Příklad: heuristika seskupování a distribuce (zadání)

- 1 stroj, 10 úloh, $p_j = 1$
- Atributy úlohy j: a_{j1} a a_{j2}
- Cena za nastavení s využitím a_{j1} pro úlohu j:
  - dvě po sobě jdoucí úlohy mají $a_{j1} \neq a_{k1}$, pak $c_{jk} = |a_{j1} - a_{k1}|$
- $a_{j2} = a_{k2} = 1$ a mezi j a k je $(l-1)$ úloh, pak $\psi_2(l) = \max(3 - l, 0)$
  - úlohy těsně po sobě $(0 = l - 1)$, pak je penalizace $\max(3 - 1, 0) = 2$

Příklad: heuristika seskupování a distribuce (zadání)

- 1 stroj, 10 úloh, $p_j = 1$
- Atributy úlohy j: a_{j1} a a_{j2}
- Cena za nastavení s využitím a_{j1} pro úlohu j:
  - dvě po sobě jdoucí úlohy mají $a_{j1} \neq a_{k1}$, pak $c_{jk} = |a_{j1} - a_{k1}|$
- $a_{j2} = a_{k2} = 1$ a mezi j a k je $(l-1)$ úloh, pak $\psi_2(l) = \max(3 - l, 0)$
  - úlohy těsně po sobě $(0 = l - 1)$, pak je penalizace $\max(3 - 1, 0) = 2$
  - jedna úloha mezi nimi $(1 = l - 1)$, pak je penalizace $\max(3 - 2, 0) = 1$

Příklad: heuristika seskupování a distribuce (zadání)

- 1 stroj, 10 úloh, $p_j = 1$
- Atributy úlohy j: a_{j1} a a_{j2}
- Cena za nastavení s využitím a_{j1} pro úlohu j:
  - dvě po sobě jdoucí úlohy mají $a_{j1} \neq a_{k1}$, pak $c_{jk} = |a_{j1} - a_{k1}|$
- $a_{j2} = a_{k2} = 1$ a mezi j a k je $(l-1)$ úloh, pak $\psi_2(l) = \max(3 - l, 0)$
  - úlohy těsně po sobě $(0 = l - 1)$, pak je penalizace $\max(3 - 1, 0) = 2$
  - jedna úloha mezi nimi $(1 = l - 1)$, pak je penalizace $\max(3 - 2, 0) = 1$
  - více než jedna úloha mezi nimi (s), pak je penalizace $\max(3 - s, 0) = 0$

Příklad: heuristika seskupování a distribuce (zadání)

- 1 stroj, 10 úloh, $p_j = 1$
- Atributy úlohy j: a_{j1} a a_{j2}
- Cena za nastavení s využitím a_{j1} pro úlohu j:
  - dvě po sobě jdoucí úlohy mají $a_{j1} \neq a_{k1}$, pak $c_{jk} = |a_{j1} - a_{k1}|$
- $a_{j2} = a_{k2} = 1$ a mezi j a k je $(l-1)$ úloh, pak $\psi_2(l) = \max(3-l, 0)$
  - úlohy těsně po sobě $(0 = l - 1)$, pak je penalizace $\max(3-1, 0) = 2$
  - jedna úloha mezi nimi $(1 = l - 1)$, pak je penalizace $\max(3-2, 0) = 1$
  - více než jedna úloha mezi nimi (s), pak je penalizace $\max(3-s, 0) = 0$
- Některé úlohy mají konečné termíny dokončení,
  pokud překročeny, pak $w_j T_j$ brána v úvahu

Úlohy	1	2	3	4	5	6	7	8	9	10
a_{j1}	1	1	1	3	3	3	5	5	5	5
a_{j2}	0	1	1	0	1	1	1	0	0	0
d_j	∞	2	∞	∞	∞	∞	6	∞	∞	∞
w_j	0	4	0	0	0	0	4	0	0	0

Příklad: heuristika seskupování a distribuce (řešení)

- 3 skupiny dle a_{j1}
- Skupina A: úlohy 1,2,3 skupina B: úlohy 4,5,6
  skupina C: úlohy 7,8,9,10
- Nejlepší pořadí skupin
  vzhledem k ceně za nastavení A, B, C nebo C, B, A
- Ale úloha 7 nebo 2 by nebyla dokončena včas a cena za zpoždění vysoká $\Rightarrow$ výběr pořadí A, C, B

Příklad: heuristika seskupování a distribuce (řešení)

- 3 skupiny dle a_{j1}
- Skupina A: úlohy 1,2,3 skupina B: úlohy 4,5,6
  skupina C: úlohy 7,8,9,10
- Nejlepší pořadí skupin
  vzhledem k ceně za nastavení A, B, C nebo C, B, A
- Ale úloha 7 nebo 2 by nebyla dokončena včas a cena za zpoždění vysoká $\Rightarrow$ výběr pořadí A, C, B
- Úlohy s atributem 2 nutno distribuovat, optimální posloupnosti minimalizující penalizaci ψ_2
  - A: 2,1,3 atributy 1 0 1
  - C: 8,7,9,10 atributy 0 1 0 0
  - B: 5,4,6 atributy 1 0 1
  - cena 3 (první-třetí=1, třetí-pátý=1, osmý-desátý=1)
- Celková cena: 6+3+0=9
  (cena za nastavení+cena za distribuci+cena za zpoždění)

Shrnutí

Montážní linka s flexibilním časem

- př. výroba kopírek
- cyklické rozvrhy
- heuristika padnoucího profilu PF

Montážní linka s fixním časem

- př. výroba automobilů
- heuristika seskupování a distribuce

Rezervace

16. května 2022

Rezervační systém

- m strojů zapojených paralelně
- n úloh s
  - dobou provádění $p_1, \ldots, p_n$
  - termíny dostupnosti $r_1, \ldots, r_n$
  - termíny dokončení $d_1, \ldots, d_n$
  - váhy $w_1, \ldots, w_n$
- Úloha musí být prováděna v rámci daného časového intervalu
  - termín dostupnosti a dokončení nelze porušit
- Nemusí být možné realizovat všechny úlohy
- Cíl: vyber podmnožinu úloh,
  - pro kterou existuje konzistentní rozvrh
  - která maximalizuje danou objektivní funkci
    - maximalizace počtu prováděných úloh
    - maximalizace celkového množství provádění
    - maximalizace profitu realizovaných úloh (zadané váhy)

Základní modely

- Systém bez rezervy
  - úlohy trvají od termínu dostupnosti do termínu dokončení, tj. $p_j = d_j - r_j$
  - mluvíme o pevných intervalech nebo o intervalovém rozvrhování
- Systémy s rezervou
  - interval mezi termínem dostupnosti a dokončení může mít nějakou rezervu, tj. $p_j \leq d_j - r_j$

Aplikace rezervačních systémů

- Pronájem aut
  - čtyři typy aut: subcompact, střední velikost, plná velikost, sportovní typ
  - pevné množství každého typu
  - stroj = typ auta
  - úloha = zákazník požadující auto
  - zákazník může požadovat určitý(é) typ(y) auta
    - úloha může být prováděna na podmnožině strojů
  - v případě nedostatku některého typu auta může být nabídnut v některých případech silnější typ auta
- Rezervace pokojů v hotelu
- Rezervace strojů v továrně

Intervalové rozvrhování

- m strojů zapojených paralelně
- n úloh, pro úlohu j zadán
  - termín dostupnosti r_j
  - termín dokončení d_j
  - doba provádění $p_j = d_j - r_j$
  - množina M_j strojů, na kterých může být úloha j prováděna
  - w_{ij}: profit z provádění úlohy j na stroji i
- Cíl: maximalizovat profit z prováděných úloh
  - $w_{ij} = 1$: maximalizovat počet realizovaných úloh
  - $w_{ij} = w_j$: maximalizovat vážený počet realizovaných úloh

Formulace celočíselného programování

- Časové periody $1, \ldots, H$
- J_l: množina úloh, která potřebuje provádění v čase l (známe předem!)
- $x_{ij} = 1$ pokud je úloha j prováděna na stroji i

  $x_{ij} = 0$ jinak
- Maximalizace

Formulace celočíselného programování

- Časové periody $1, \ldots, H$
- J_l: množina úloh, která potřebuje provádění v čase l (známe předem!)
- $x_{ij} = 1$ pokud je úloha j prováděna na stroji i

  $x_{ij} = 0$ jinak
- Maximalizace

$$\sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij}$$

Formulace celočíselného programování

- Časové periody $1, \ldots, H$
- J_l: množina úloh, která potřebuje provádění v čase l (známe předem!)
- $x_{ij} = 1$ pokud je úloha j prováděna na stroji i

  $x_{ij} = 0$ jinak
- Maximalizace

$$\sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij}$$

za předpokladu:

úloha běží nejvýše na jednom stroji

Formulace celočíselného programování

- Časové periody $1, \ldots, H$
- J_l: množina úloh, která potřebuje provádění v čase l (známe předem!)
- $x_{ij} = 1$ pokud je úloha j prováděna na stroji i

  $x_{ij} = 0$ jinak
- Maximalizace

$$\sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij}$$

za předpokladu:

úloha běží nejvýše na jednom stroji

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad j = 1, \ldots, n$$

Formulace celočíselného programování

- Časové periody $1, \ldots, H$
- J_l: množina úloh, která potřebuje provádění v čase l (známe předem!)
- $x_{ij} = 1$ pokud je úloha j prováděna na stroji i

  $x_{ij} = 0$ jinak
- Maximalizace

$$\sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij}$$

za předpokladu:

úloha běží nejvýše na jednom stroji

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad j = 1, \ldots, n$$

v každém čase běží na každém stroji nejvýše jedna úloha

Formulace celočíselného programování

- Časové periody $1, \ldots, H$
- J_l: množina úloh, která potřebuje provádění v čase l (známe předem!)
- $x_{ij} = 1$ pokud je úloha j prováděna na stroji i

  $x_{ij} = 0$ jinak
- Maximalizace

$$\sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij}$$

za předpokladu:

úloha běží nejvýše na jednom stroji

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad j = 1, \ldots, n$$

v každém čase běží na každém stroji nejvýše jedna úloha

$$\sum_{j \in J_l} x_{ij} \leq 1 \qquad i = 1, \ldots, m \quad l = 1, \ldots, H$$

provádění úlohy j na stroji i: $\quad x_{ij} \in \{0, 1\}$

Jednotková doba trvání

- Každá úloha je dostupná přesně 1 časovou jednotku
- Co z toho plyne?

Jednotková doba trvání

- Každá úloha je dostupná přesně 1 časovou jednotku
- Co z toho plyne? Problém lze rozdělit na nezavislé problémy
  - víme přesně, které úlohy i jsou prováděny v každé časové jednotce
  - jeden podproblém pro každou časovou jednotku
- Výsledný problém pro časovou jednotku l:

Jednotková doba trvání

- Každá úloha je dostupná přesně 1 časovou jednotku
- Co z toho plyne? Problém lze rozdělit na nezavislé problémy
  - víme přesně, které úlohy i jsou prováděny v každé časové jednotce
  - jeden podproblém pro každou časovou jednotku
- Výsledný problém pro časovou jednotku l:

  Maximalizace
  $$\sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}x_{ij}$$

  za předpokladu
  $$\sum_{i=1}^{m} x_{ij} \leq 1 \quad j = 1, \ldots, n$$
  $$\sum_{j \in J_l} x_{ij} \leq 1 \quad i = 1, \ldots, m$$
  $$x_{ij} \in \{0, 1\}$$

- Jedná se o problém přiřazení *(assignment problem)*
  - problém řešitelný v polynomiálním čase

Jednotková váha & identické stroje

- $w_{ij} = 1$; $M_j = \{1, \ldots, m\}$ pro všechna i, j; obecná p_j
  - tedy stroje jsou identické a
    cíl je maximalizovat počet realizovaných úloh
  - nejednotková p_j, nelze tedy řešit rozkladem v čase
- Algoritmus dávající optimální řešení

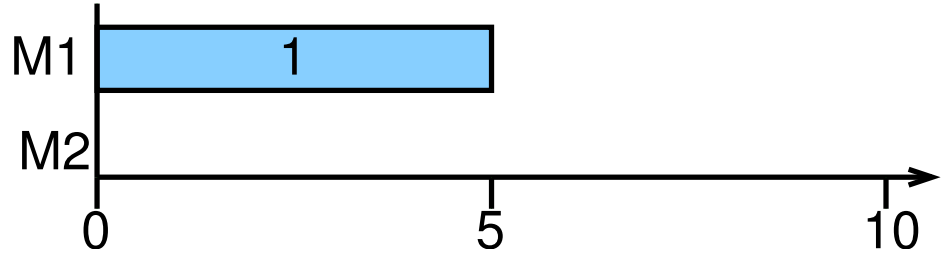
  Předpokládejme: $r_1 \leq \cdots \leq r_n$
  $J = \emptyset$ (J je množina už vybraných úloh pro provádění)
  for $j = 1$ to n do
    if existuje stroj dostupný v čase r_j then
      přiřaď j tomuto stroji
      $J := J \cup \{j\}$
    else urči úlohu j^* takovou, že $C_{j^*} = \max_{k \in J} C_k = \max_{k \in J}(r_k + p_k)$
      if $C_j = r_j + p_j > C_{j^*}$ then
        nezařazuj j do J
      else přiřaď j stroji j^*
        $J := J \cup \{j\} \setminus \{j^*\}$

Příklad: jednotková váha & identické stroje

2 stroje a 8 úloh:

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

Iterace 1: $j = 1$

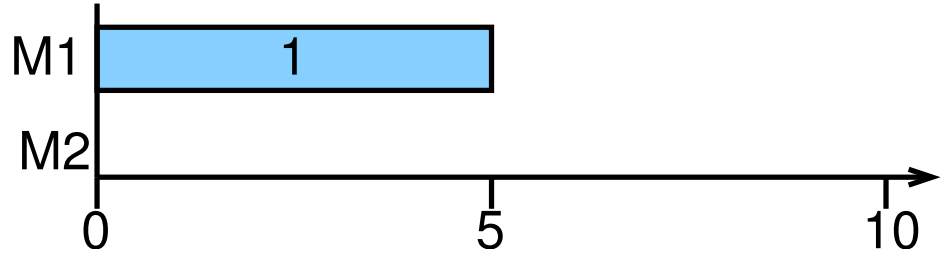


Příklad: jednotková váha & identické stroje

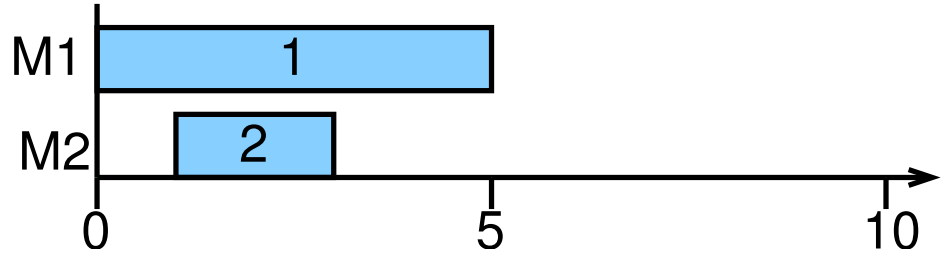
2 stroje a 8 úloh:

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

Iterace 1: $j = 1$



Iterace 2: $j = 2$

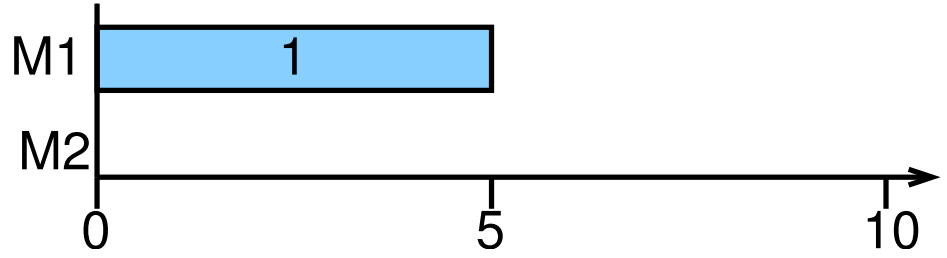


Příklad: jednotková váha & identické stroje

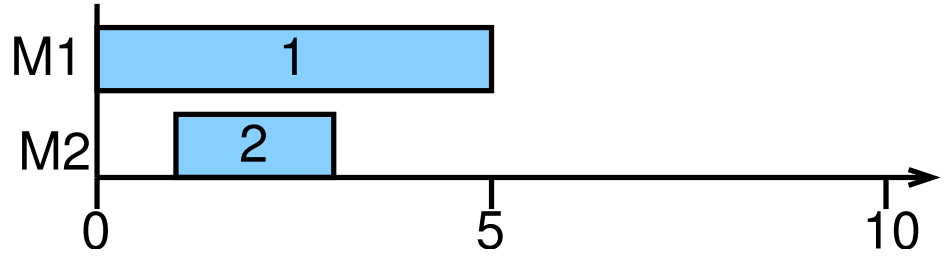
2 stroje a 8 úloh:

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

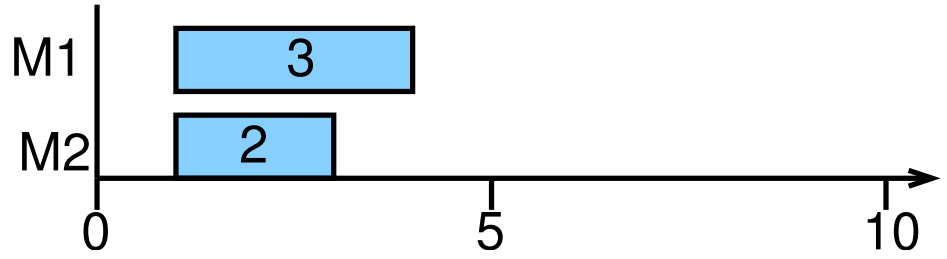
Iterace 1: $j = 1$



Iterace 2: $j = 2$



Iterace 3: $j = 3, j^* = 1$

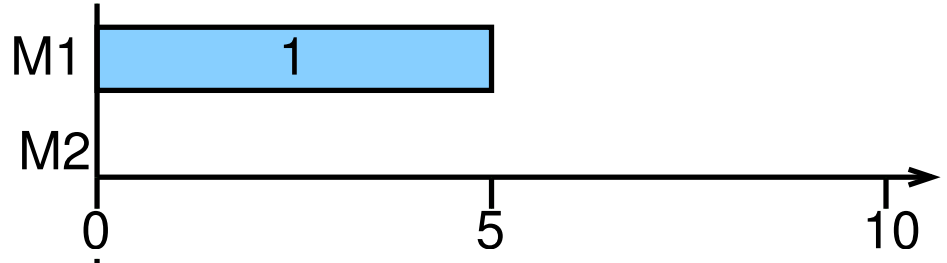


Příklad: jednotková váha & identické stroje

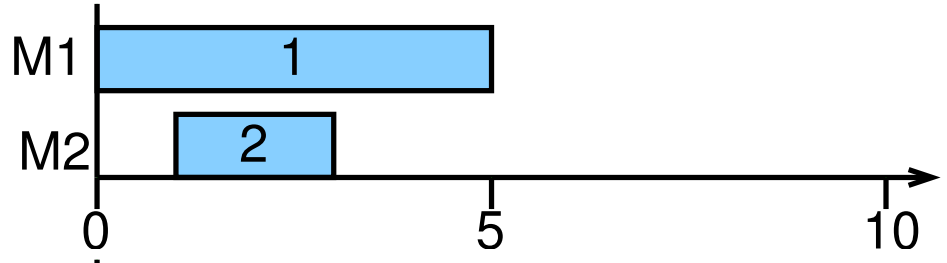
2 stroje a 8 úloh:

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

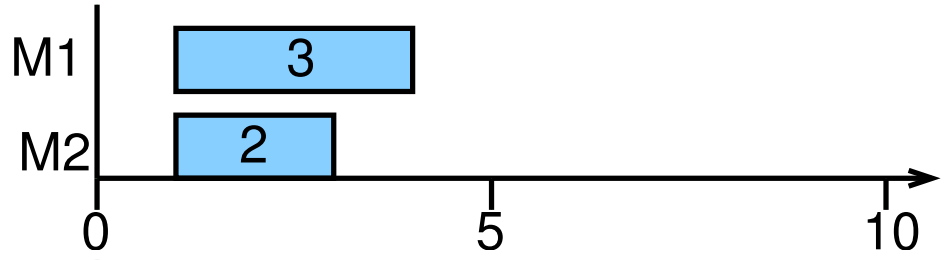
Iterace 1: $j = 1$



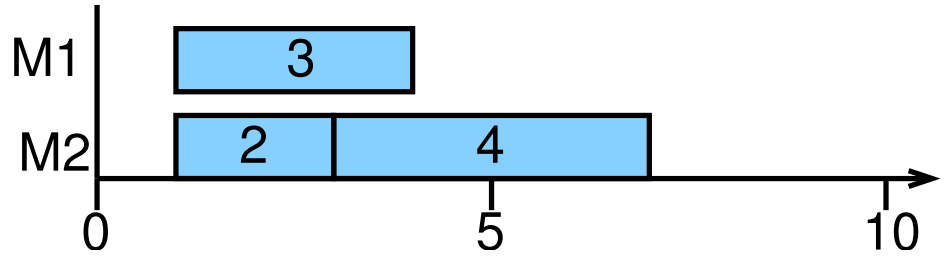
Iterace 2: $j = 2$



Iterace 3: $j = 3, j^* = 1$



Iterace 4: $j = 4$

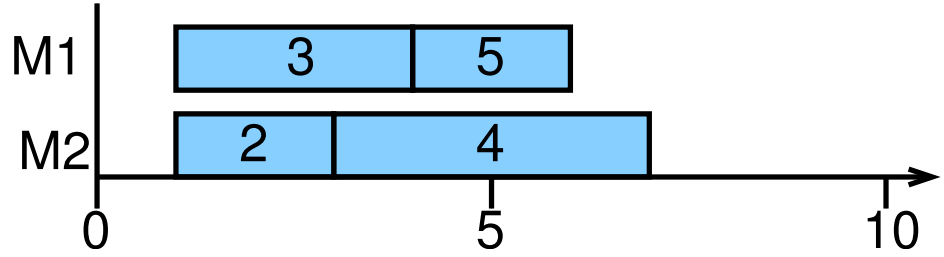


Příklad: jednotková váha & identické stroje

2 stroje a 8 úloh:

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

Iterace 5: $j = 5$

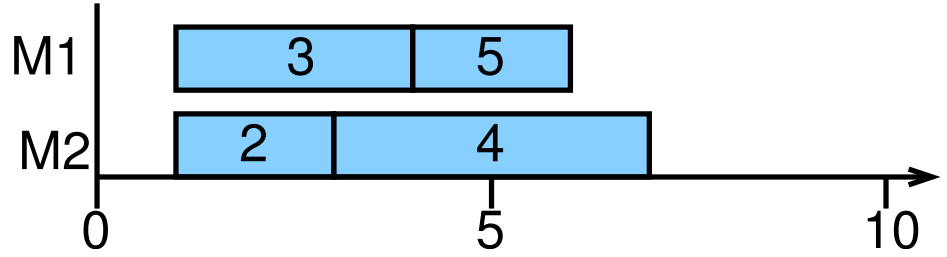


Příklad: jednotková váha & identické stroje

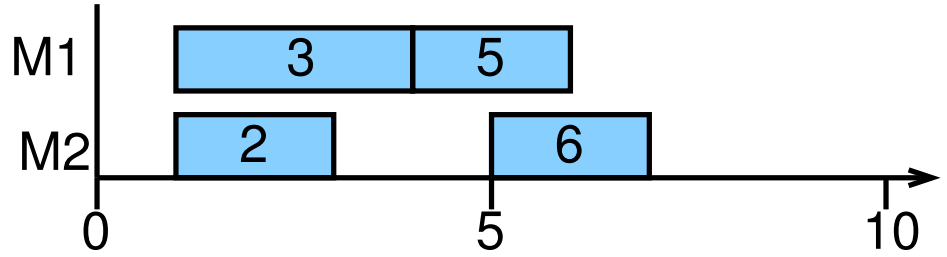
2 stroje a 8 úloh:

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

Iterace 5: $j = 5$



Iterace 6: $j = 6, j^* = 4$

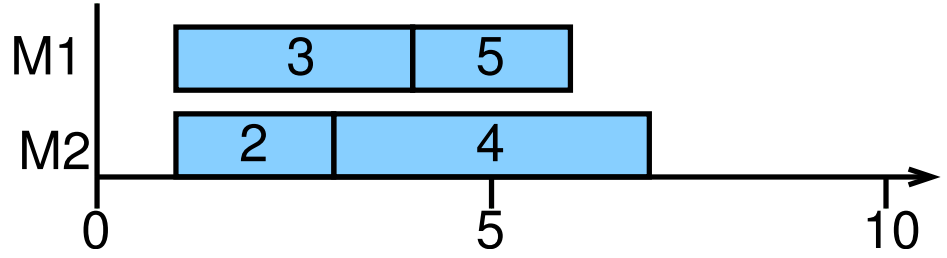


Příklad: jednotková váha & identické stroje

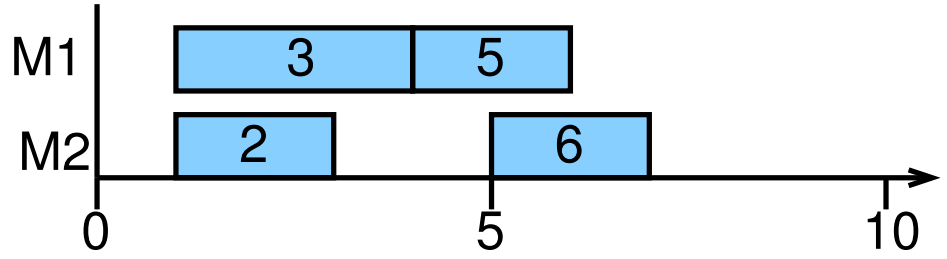
2 stroje a 8 úloh:

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

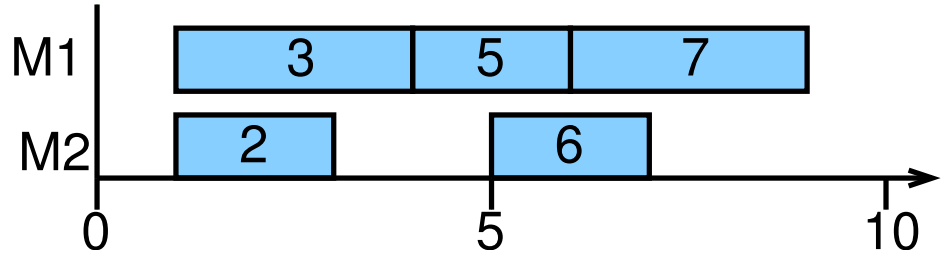
Iterace 5: $j = 5$



Iterace 6: $j = 6, j^* = 4$



Iterace 7: $j = 7$

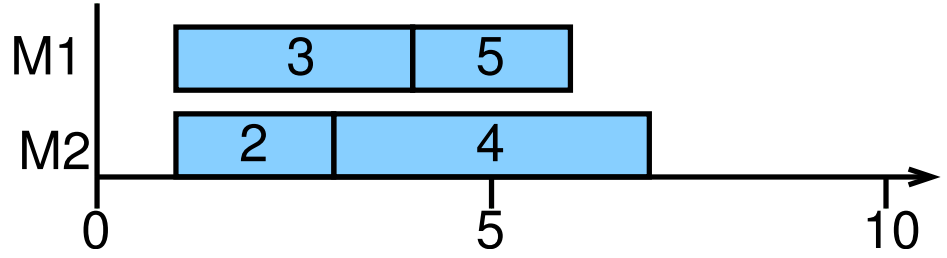


Příklad: jednotková váha & identické stroje

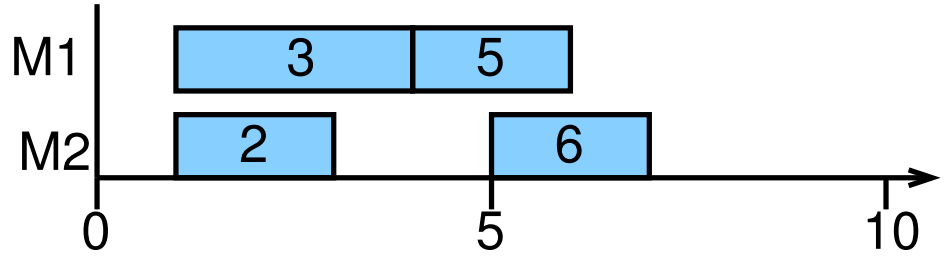
2 stroje a 8 úloh:

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

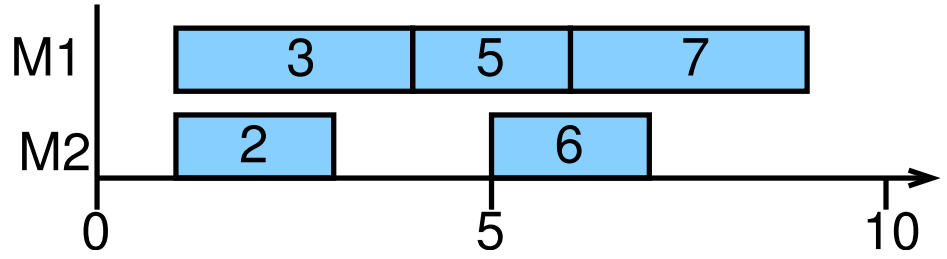
Iterace 5: $j = 5$



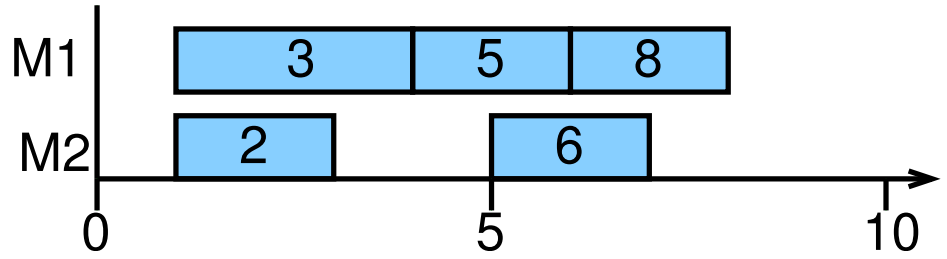
Iterace 6: $j = 6, j^* = 4$



Iterace 7: $j = 7$



Iterace 8: $j = 8, j^* = 7$



Jednotková váha a nelimitovaný počet identických strojů

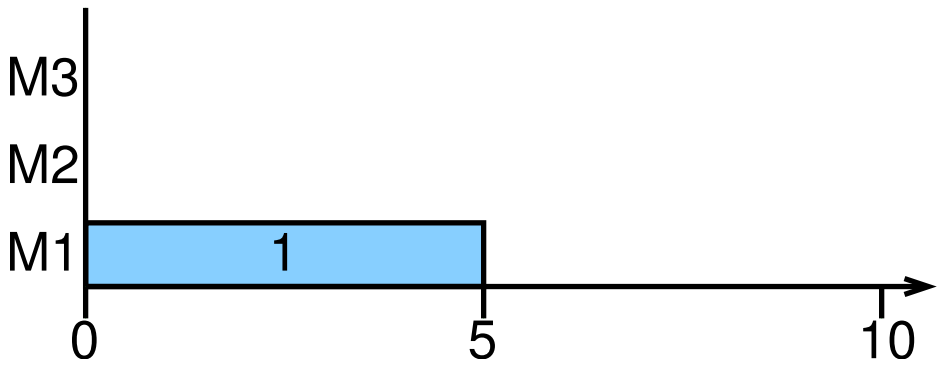
- $w_{ij} = 1$ pro všechna i, j
- Všechny úlohy musí být realizovány
- Cíl: použít minimální počet strojů
- Algoritmus dávající optimální řešení

Jednotková váha a nelimitovaný počet identických strojů

- $w_{ij} = 1$ pro všechna i, j
- Všechny úlohy musí být realizovány
- Cíl: použít minimální počet strojů
- Algoritmus dávající optimální řešení

Předpokladejme: $r_1 \leq \cdots \leq r_n$
$M = \emptyset$ (M množina použitých strojů)
$i := 0$
for $j = 1$ to n do
  if stroj z M je volný v r_j
  then
    přiřaď j na volný stroj
  else
    $i := i + 1$
    $M := M \cup \{i\}$
    přiřaď úlohu j na stroj i

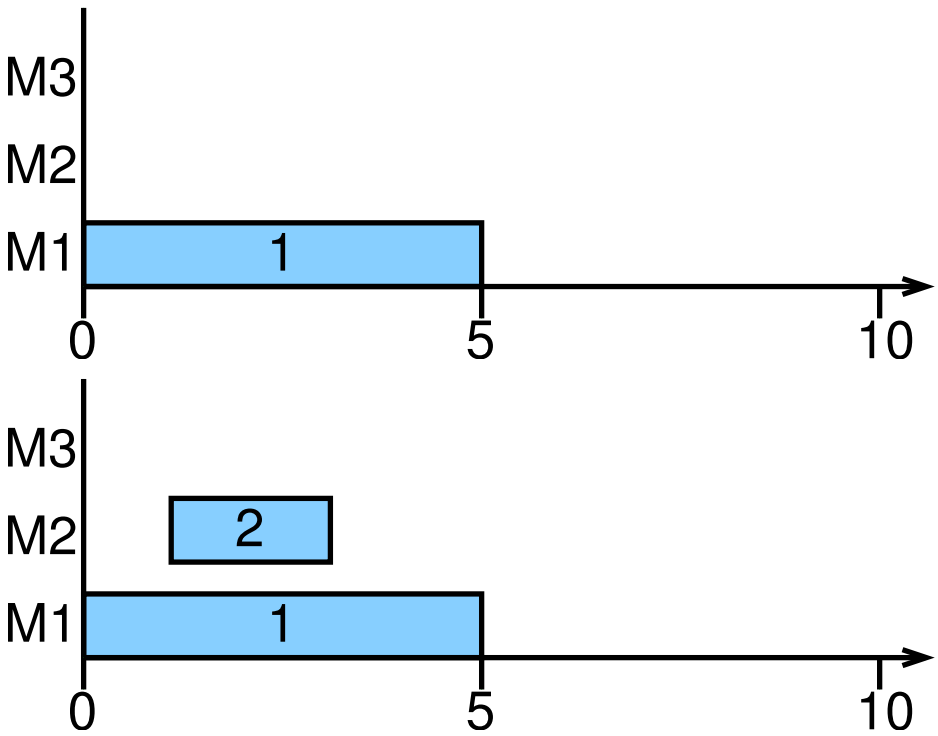
Příklad: jednotková váha a nelimitovaný počet identických strojů



j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

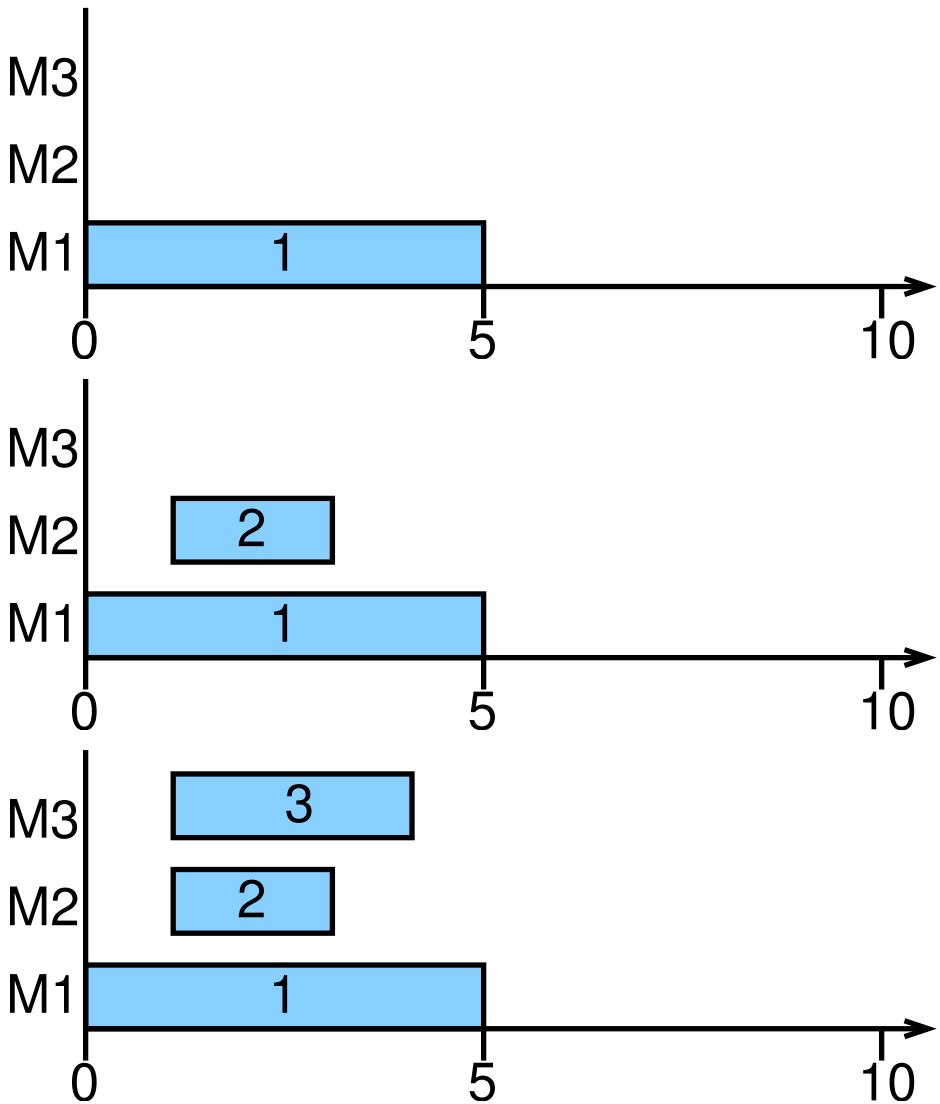
Příklad: jednotková váha a nelimitovaný počet identických strojů

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

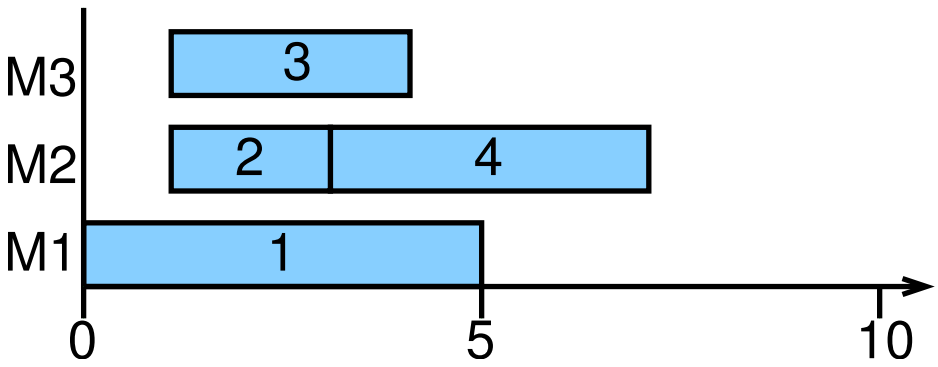


Příklad: jednotková váha a nelimitovaný počet identických strojů

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

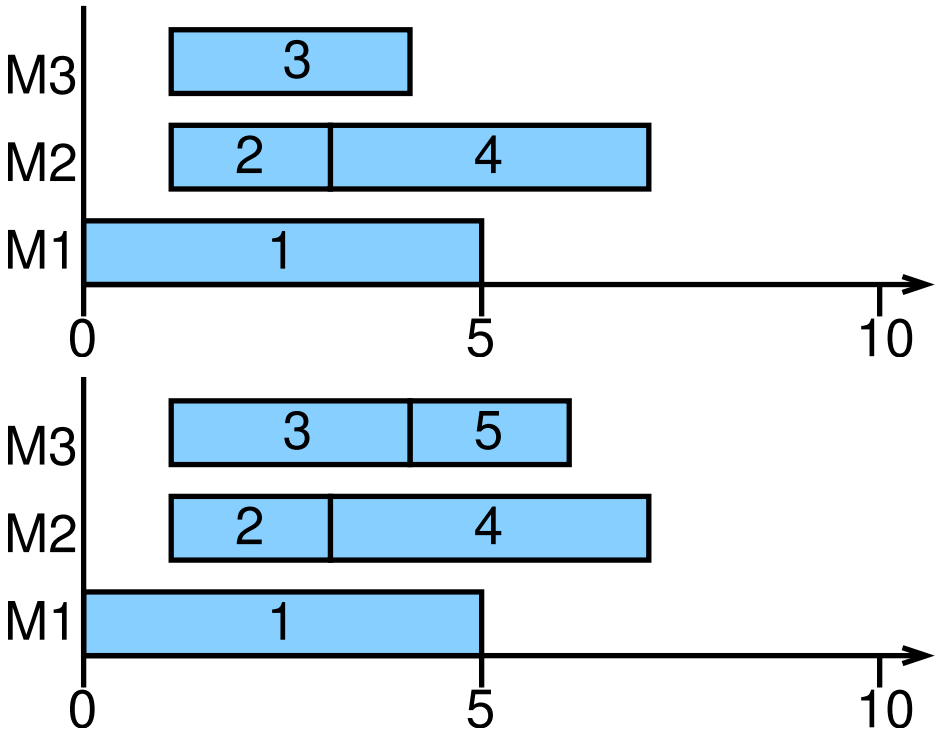


Příklad: jednotková váha a nelimitovaný počet identických strojů



j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

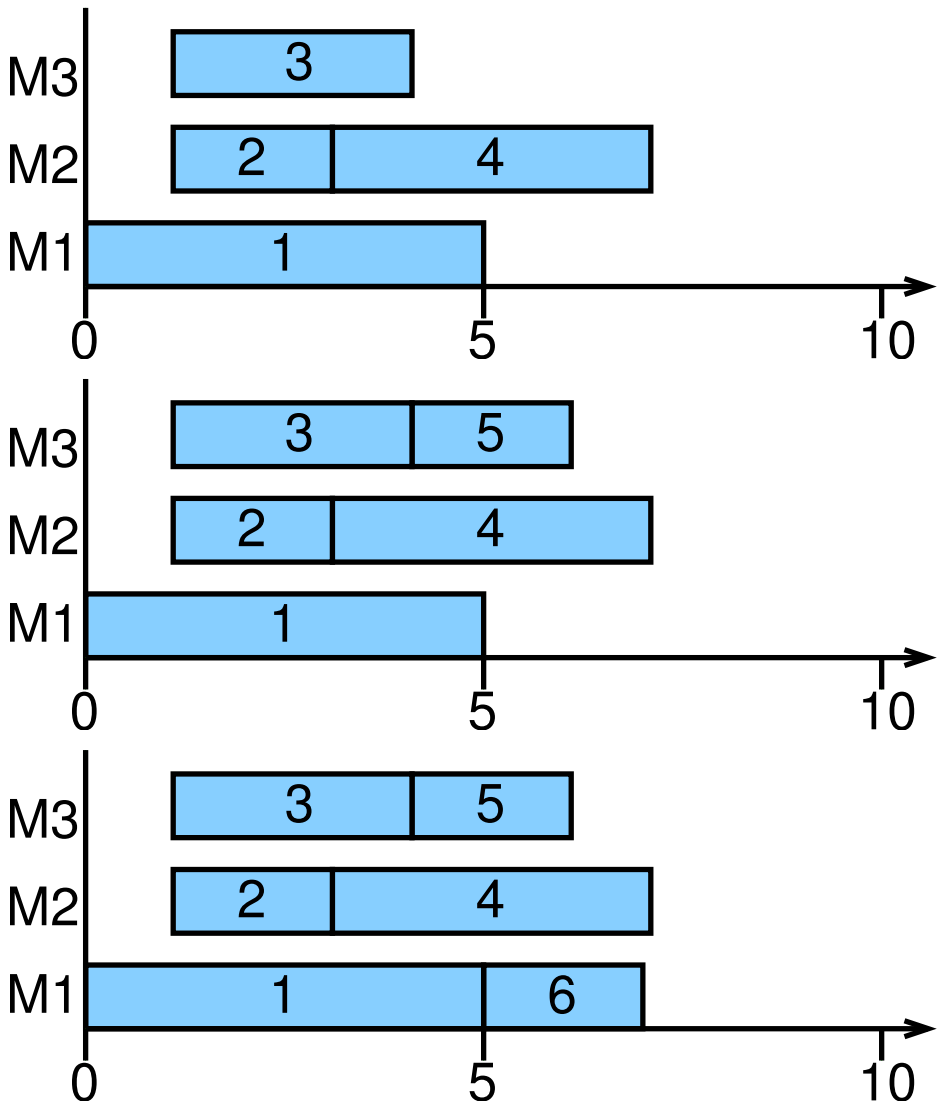
Příklad: jednotková váha a nelimitovaný počet identických strojů



j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

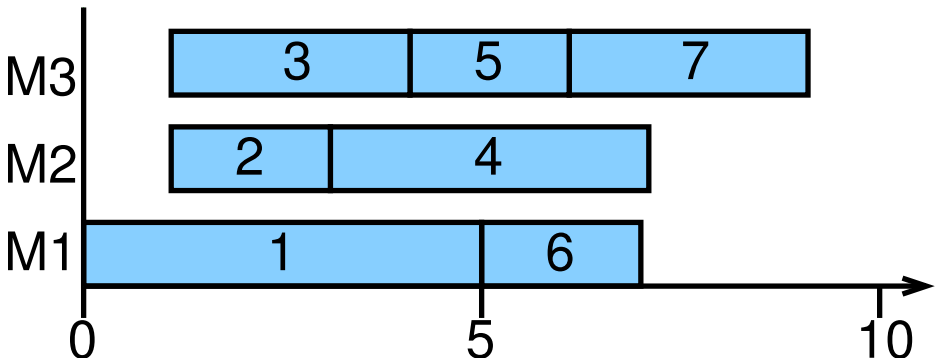
Příklad: jednotková váha a nelimitovaný počet identických strojů

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8



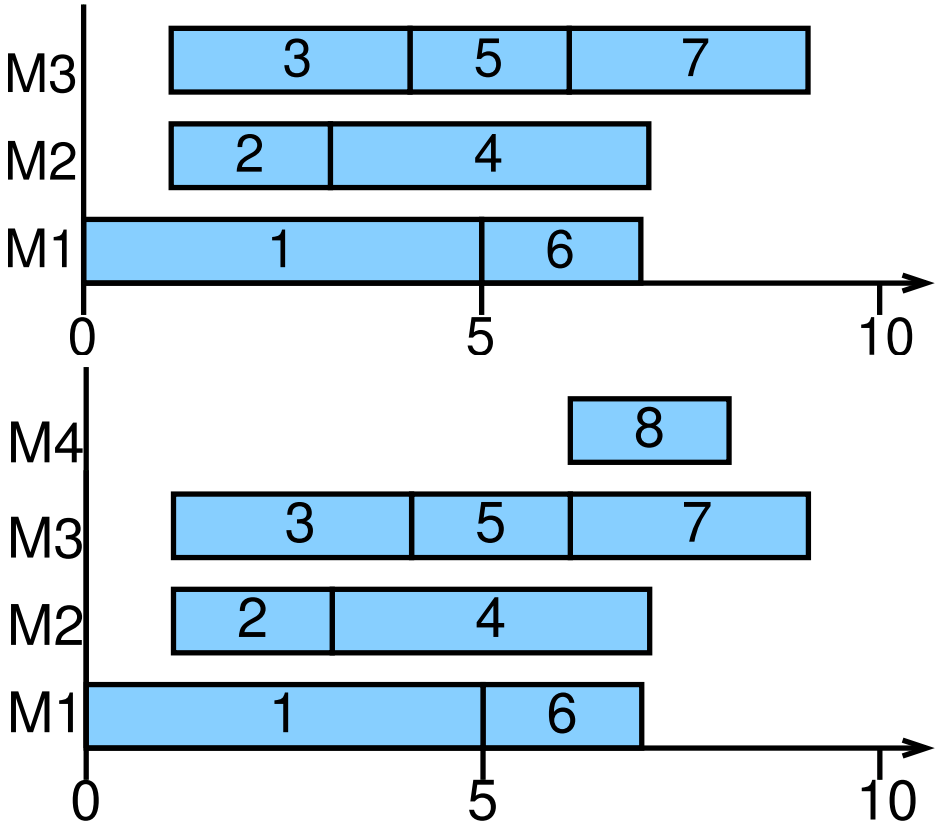
Příklad: jednotková váha a nelimitovaný počet identických strojů

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8



Příklad: jednotková váha a nelimitovaný počet identických strojů

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8



Barvení grafu

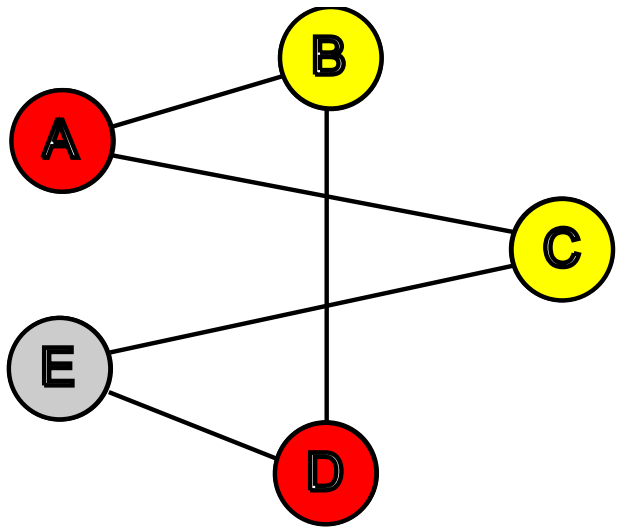
Problém barvení grafu

- Je možné obarvit vrcholy grafu s použitím n barev tak, aby žádné dva sousední vrcholy nebyly obarveny stejnou barvou?

Chromatické číslo grafu

- Minimální počet barev n nutný k obarvení grafu tak, by žádné dva sousední vrcholy nebyly obarveny stejnou barvou.

NP-úplný problém



Reformulace na problém barvení grafu

Problém s jednotkovou vahou a nelimitovaným počtem identických strojů lze reformulovat na problém barvení grafu

Reformulace na problém barvení grafu

Problém s jednotkovou vahou a nelimitovaným počtem identických strojů lze reformulovat na problém barvení grafu

- vrchol = úloha
- hrana (j, k) znamená, že se úlohy j a k překrývají v čase
- každá barva odpovídá jednomu stroji
- přiřaď barvu (tj. stroj) každému vrcholu tak, že dva sousední vrcholy (překrývají se v čase) mají různou barvu (tj. stroje)

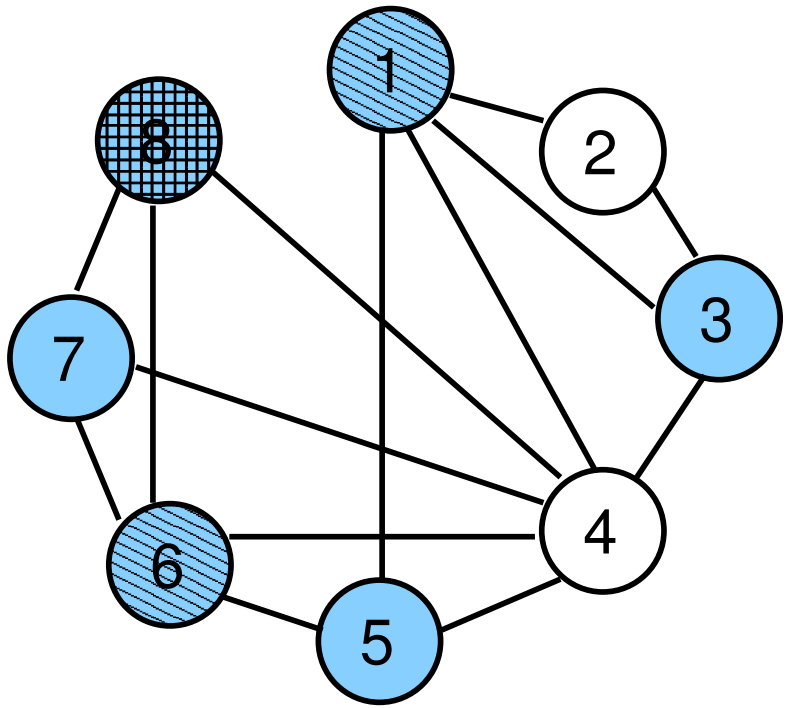
Poznámky:

- obecný problém existence obarvení grafu m barvami je NP-úplný
- uvažovaný rezervační problém je speciálním (jednodušším) případem problému barvení grafu
- heuristiky diskutovány v kapitole o *timetabling*

Příklad: reformulace

j	1	2	3	4	5	6	7	8
r_j	0	1	1	3	4	5	6	6
d_j	5	3	4	7	6	7	9	8

Odpovídající řešení problému barvení grafu:



Rezervační systém s rezervou

- Rezervační systém s rezervou: $p_j \leq d_j - r_j$
- Triviální případ
  - doby provádění = 1, identické váhy, identické stroje
  - řešení opět dekompozicí v čase
- Maximalizace váženého počtu aktivit
  - NP-těžký problém $\Rightarrow$ řešení heuristikami
  - kompozitní řídící pravidlo
    - předzpracování:
      určení flexibility úloh a strojů
    - algoritmus:
      nejméně flexibilní úloha na nejméně flexibilním stroji první

Předzpracování

- ψ_{it}: počet úloh, které mohou být přiřazeny
  na stroj i během intervalu $[t-1, t]$
  - možné využití stroje i v čase t
  - čím vyšší hodnota, tím je zdroj i v tomto čase flexibilnější

- $|M_j|$: počet strojů, které mohou být přiřazeny úloze j
  - čím vyšší hodnota, tím je úloha j flexibilnější

Prioritní indexy

- Prioritní index pro úlohu j: $I_j = f(w_j/p_j, |M_j|)$
  - uspořádání úloh podle: $I_1 \leq I_2 \leq \cdots \leq I_n$
  - čím menší je $|M_j|$, tím nižší je I_j
  - čím vyšší je w_j/p_j, tím nižší je I_j
    - snažíme se dát přednost kratším úlohám, protože maximalizujeme počet vážených provedených aktivit a delší úlohy jsou náročnější
  - např. $I_j = \frac{|M_j|}{w_j/p_j}$

Prioritní indexy

- Prioritní index pro úlohu j: $I_j = f(w_j/p_j, |M_j|)$
  - uspořádání úloh podle: $I_1 \leq I_2 \leq \cdots \leq I_n$
  - čím menší je $|M_j|$, tím nižší je I_j
  - čím vyšší je w_j/p_j, tím nižší je I_j
    - snažíme se dát přednost kratším úlohám, protože maximalizujeme počet vážených provedených aktivit a delší úlohy jsou náročnější
  - např. $I_j = \frac{|M_j|}{w_j/p_j}$
- Prioritní index pro stroj
  - vybírá stroj pro úlohu
  - jestliže úloha potřebuje stroj v $[t, t + p_j]$
    pak výběr stroje i záleží na funkci s faktory $\psi_{i,t+1}, \ldots, \psi_{i,t+p_j}$, tj. na $g(\psi_{i,t+1}, \ldots, \psi_{i,t+p_j})$
  - např. $g(\psi_{i,t+1}, \ldots, \psi_{i,t+p_j}) = \left(\sum_{l=1}^{p_j} \psi_{i,t+l} \right) / p_j$

    nebo $g(\psi_{i,t+1}, \ldots, \psi_{i,t+p_j}) = \max(\psi_{i,t+1}, \ldots, \psi_{i,t+p_j})$

Algoritmus maximalizace váženého počtu aktivit

1. $j = 1$
2. Vyber úlohu j s nejnižším l_j a

   vyber mezi stroji a dostupnými časy zdroj a čas s nejnižší $g(\psi_{i,t+1}, \ldots, \psi_{i,t+p_j})$

   Zruš úlohu j jestliže nemůže být přiřazena ani jednomu stroji v žádném čase
3. Jestliže $j = n$ STOP

   jinak nastav $j = j + 1$ a běž na krok 2

Cvičení: maximalizace váženého počtu aktivit

- Nalezněte rozvrh pro daný problém

Úlohy	1	2	3	4	5	6	7
p_j	3	10	9	4	6	5	3
w_j	2	3	3	2	1	2	3
r_j	5	0	2	3	2	4	5
d_j	12	10	20	15	18	19	14
M_j	$\{1,3\}$	$\{1,2\}$	$\{1,2,3\}$	$\{2,3\}$	$\{1\}$	$\{1\}$	$\{1,2\}$

- pro $l_j = \frac{|M_j|}{w_j/p_j}$ a

  $g(\psi_{i,t+1}, \ldots, \psi_{i,t+p_j}) = \left(\sum_{l=1}^{p_j} \psi_{i,t+l}\right) / p_j$

Cvičení: maximalizace váženého počtu aktivit

- Nalezněte rozvrh pro daný problém

Úlohy	1	2	3	4	5	6	7
p_j	3	10	9	4	6	5	3
w_j	2	3	3	2	1	2	3
r_j	5	0	2	3	2	4	5
d_j	12	10	20	15	18	19	14
M_j	$\{1,3\}$	$\{1,2\}$	$\{1,2,3\}$	$\{2,3\}$	$\{1\}$	$\{1\}$	$\{1,2\}$

- pro $l_j = \frac{|M_j|}{w_j/p_j}$ a

  $g(\psi_{i,t+1}, \ldots, \psi_{i,t+p_j}) = \left(\sum_{l=1}^{p_j} \psi_{i,t+l}\right) / p_j$

- Řešení:

Úlohy	7	6	1	4	5	2
Stroje	2	1	3	3	1	2
Časy	11-14	14-19	5-8	11-15	2-8	0-10

úlohu 3 nelze umístit

Shrnutí

Rezervace

- Intervalové rozvrhování (rezervační systém bez rezervy)
  - celočíselné programování
  - jednotková doba trvání: formulace problému přiřazení (řešení pro každou čas. jednotku zvlášť)
  - jednotková váha & identické stroje (maximalizace počtu provedených aktivit)
  - jednotková váha & nelimitovaný počet identických strojů
- Rezervační systém s rezervou
  - kompozitní řídící pravidlo

Rozvrhování jako timetabling

16. května 2022

Rozvrhování = *timetabling*

- Doposud: rozvrhování = *scheduling*
  Nyní: rozvrhování = *timetabling*
  - důraz kladen na časové umístění objektů
  - vazby na časové uspořádání mezi objekty hrají malou roli
- Omezení operátorů/nástrojů *(operator/tool)*
  - mnoho operátorů
  - úloha potřebuje jeden nebo více odlišných operátorů
  - úlohy vyžadující stejné operátory nemohou běžet zároveň
  - možné cíle:
    rozvržení všech úloh v rámci časového horizontu H
    nebo minimalizace *makespan*
- Omezení pracovní síly
  - R identických pracovníků = jeden zdroj kapacity R
  - každá úloha potřebuje několik pracovníků
  - celkový počet pracovníků nesmí být nikdy překročen

Rozvrhování jako problém plánování projektu s omezenými zdroji

- Problém plánování projektu s omezenými zdroji
  resource-constrained project scheduling problem (RCPSP)
  - n úloh
  - N zdrojů
  - R_i kapacita zdroje i
  - p_j doba provádění úlohy j
  - R_{ij} požadavek úlohy j na zdroj i
  - $Prec_j$ (přímí) předchůdci úlohy j
- Rozvrhování s operátory
  - $R_i = 1$ pro všechna $i = 1 \ldots N$
- Rozvrhování s pracovní sílou
  - $N = 1$
- Oba problémy rozvrhování stále velmi obtížné
  - i při $p_j = 1$ NP-těžké
    operátoři $\equiv$ barvení grafu
    pracovní síla $\equiv$ *bin-packing*

Rozvrhování s operátory jako barvení grafu

- Omezíme se na problém s jednotkovou dobou trvání

- Úloha = uzel
- Úlohy potřebují stejného operátora = hrana mezi uzly
- Přiřazení barev (= času) uzlům
  - sousední uzly musí mít různé barvy
    tj. úlohy se stejným operátorem musí být prováděny v různých časech

- Problém existence řešení
  - tj. zadán časový horizont H a hledám rozvrh v rámci horizontu
  - může být graf obarven H barvami?
- Optimalizační problém
  - tj. minimalizace *makespan*
  - jaký nejmenší počet barev je třeba?
  - chromatické číslo grafu

Heuristiky pro barvení grafu

- Stupeň uzlu
  - počet hran spojených s uzlem
- Úroveň saturace
  - počet různých barev spojených s uzlem
- Intuice
  - obarvi uzly s vyšším stupněm dříve
  - obarvi uzly s vyšší úrovní saturace dříve

Heuristiky pro barvení grafu

- Stupeň uzlu
  - počet hran spojených s uzlem
- Úroveň saturace
  - počet různých barev spojených s uzlem
- Intuice
  - obarvi uzly s vyšším stupněm dříve
  - obarvi uzly s vyšší úrovní saturace dříve
- Algoritmus
  1. uspořádej uzly v klesajícím pořadí podle jejich stupně
  2. použij barvu 1 pro první uzel
  3. vyber neobarvený uzel s maximální úrovní saturace
     v případě volby z nich vyber uzel
     s maximálním stupněm v neobarveném podgrafu
  4. obarvi vybraný uzel s nejmenší možnou barvou
  5. jestliže jsou všechny uzly obarveny STOP
     jinak běž na krok 3

Příklad: rozvrhování schůzek

Vytvoř rozvrh pro 5 schůzek se 4 lidmi

- schůzka = úloha, člověk = operátor
- všechny schůzky trvají jednu hodinu

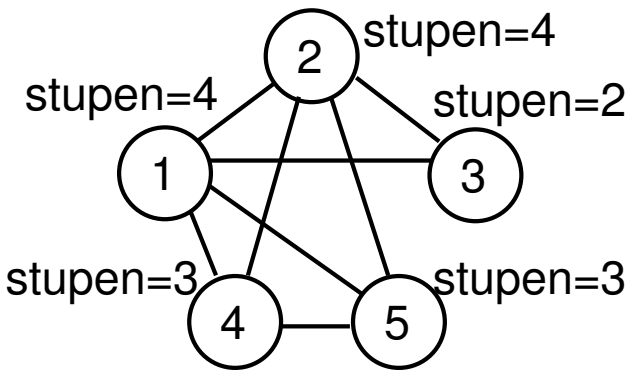
	1	2	3	4	5
Joe	1	1	0	1	1
Lisa	1	1	1	0	0
Jane	1	0	1	0	0
Larry	0	1	0	1	1

Příklad: rozvrhování schůzek

Vytvoř rozvrh pro 5 schůzek se 4 lidmi

- schůzka = úloha, člověk = operátor
- všechny schůzky trvají jednu hodinu

	1	2	3	4	5
Joe	1	1	0	1	1
Lisa	1	1	1	0	0
Jane	1	0	1	0	0
Larry	0	1	0	1	1

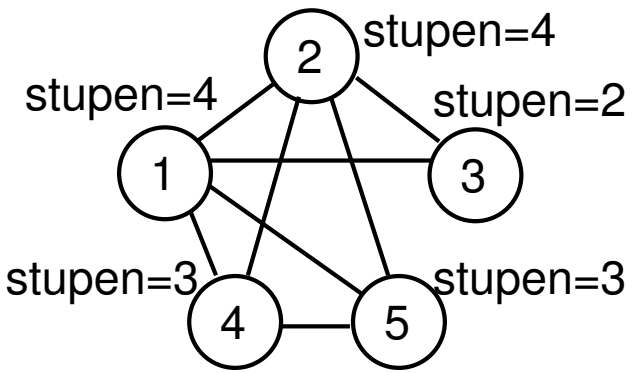


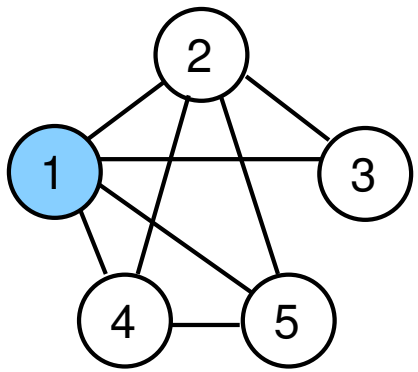
Příklad: rozvrhování schůzek

Vytvoř rozvrh pro 5 schůzek se 4 lidmi

- schůzka = úloha, člověk = operátor
- všechny schůzky trvají jednu hodinu

	1	2	3	4	5
Joe	1	1	0	1	1
Lisa	1	1	1	0	0
Jane	1	0	1	0	0
Larry	0	1	0	1	1

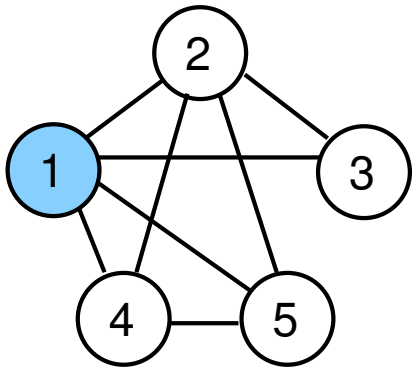




Můžeme vybrat buď úlohu 1 nebo úlohu 2

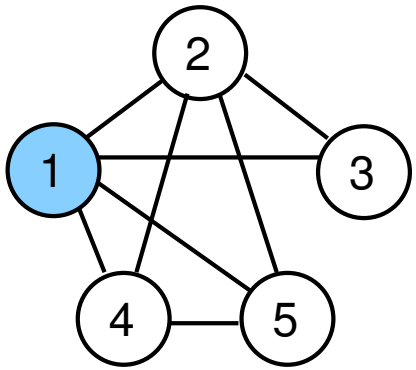
Např. vybereme 1 a obarvíme barvou 1

Příklad: rozvrhování schůzek (dokončení)

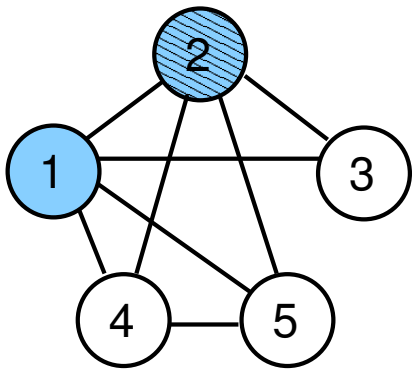


Úroveň saturace = 1 pro všechny úlohy
Vyber 2 vzhledem k nejvyššímu stupni

Příklad: rozvrhování schůzek (dokončení)

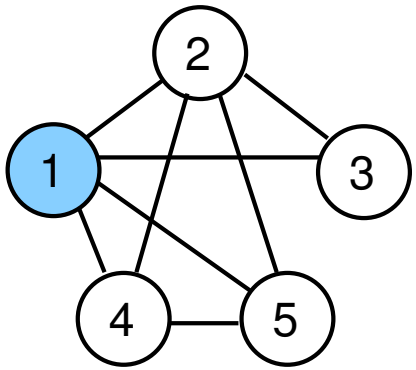


Úroveň saturace = 1 pro všechny úlohy
Vyber 2 vzhledem k nejvyššímu stupni

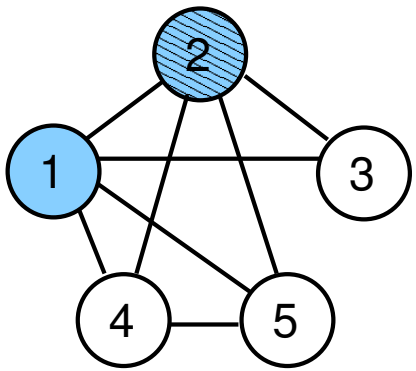


Úroveň saturace = 2 pro všechny uzly
Vyber 4 vzhledem k nejvyššímu stupni

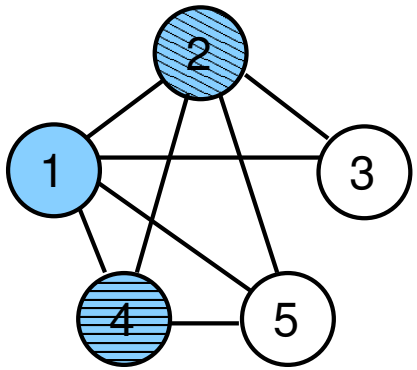
Příklad: rozvrhování schůzek (dokončení)



Úroveň saturace = 1 pro všechny úlohy
Vyber 2 vzhledem k nejvyššímu stupni

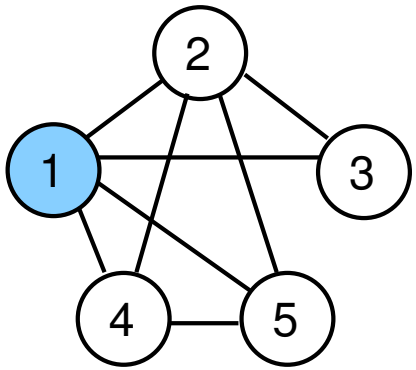


Úroveň saturace = 2 pro všechny uzly
Vyber 4 vzhledem k nejvyššímu stupni

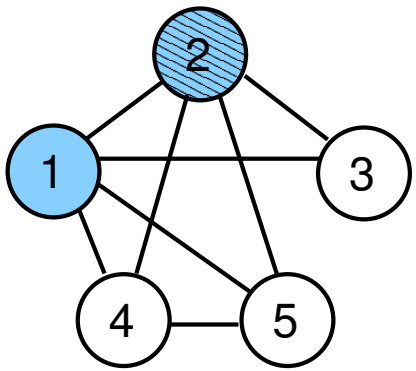


Úroveň saturace = 2 pro uzel 3
Úroveň saturace = 3 pro uzel 5
Vyber 5 na obarvení

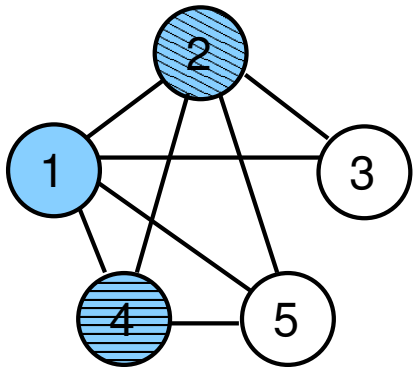
Příklad: rozvrhování schůzek (dokončení)



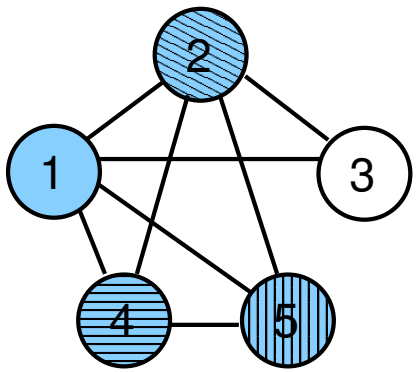
Úroveň saturace = 1 pro všechny úlohy
Vyber 2 vzhledem k nejvyššímu stupni



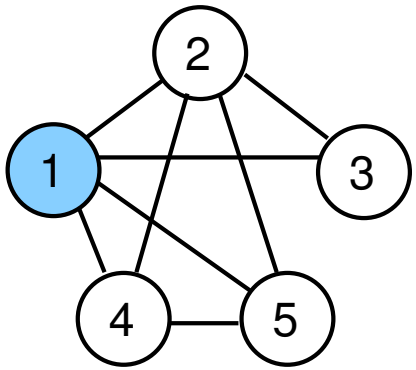
Úroveň saturace = 2 pro všechny uzly
Vyber 4 vzhledem k nejvyššímu stupni



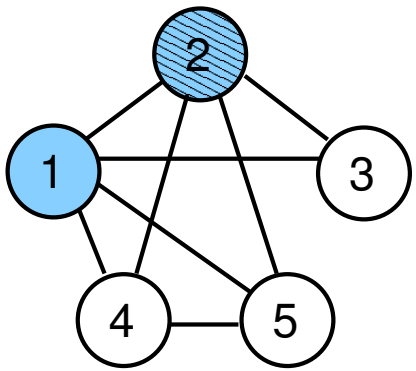
Úroveň saturace = 2 pro uzel 3
Úroveň saturace = 3 pro uzel 5
Vyber 5 na obarvení



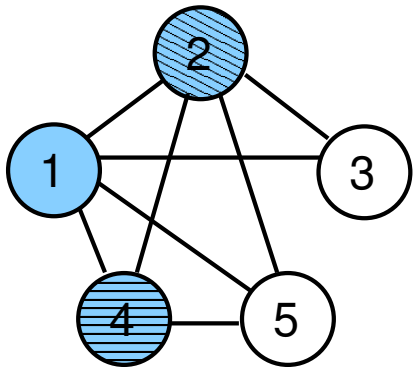
Příklad: rozvrhování schůzek (dokončení)



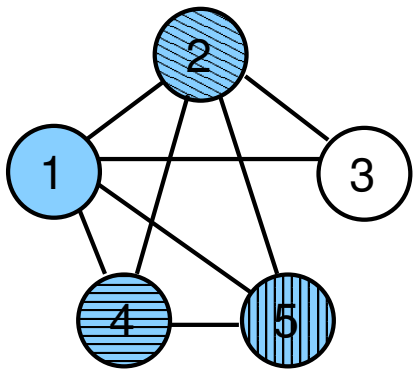
Úroveň saturace = 1 pro všechny úlohy
Vyber 2 vzhledem k nejvyššímu stupni



Úroveň saturace = 2 pro všechny uzly
Vyber 4 vzhledem k nejvyššímu stupni



Úroveň saturace = 2 pro uzel 3
Úroveň saturace = 3 pro uzel 5
Vyber 5 na obarvení



V posledním kroku obarvi 3
stejnou barvou jako 4
$\Rightarrow$celkem 4 barvy, tj. *makespan*=4

Příklad: rezervace vs. rozvrhování s operátory

Předpokládejme, že máme rezervační systém

Úloha	1	2	3
p_j	2	3	1
r_j	0	2	3
d_j	2	5	4

Lze přeformulovat na rozvrhování s operátory

Úloha	1	2	3
Operátor 1	1	0	0
Operátor 2	1	0	0
Operátor 3	0	1	0
Operátor 4	0	1	1
Operátor 5	0	1	0

Příklad: rezervace vs. rozvrhování s operátory

Předpokládejme, že máme rezervační systém

Úloha	1	2	3
p_j	2	3	1
r_j	0	2	3
d_j	2	5	4

Lze přeformulovat na rozvrhování s operátory

Úloha	1	2	3
Operátor 1	1	0	0
Operátor 2	1	0	0
Operátor 3	0	1	0
Operátor 4	0	1	1
Operátor 5	0	1	0

úloha 1 běží v čase [0,2]

úloha 2 běží v čase [2,5]
úloha 3 běží v čase [3,4]

Vztah k rezervačním systémům

- Rozvrhování s operátory blízké rezervačnímu systému bez rezervy
- Oba problémy reformulovány na problém barvení grafu
  - rezervace: hrana = dvě úlohy se překrývají v čase
  - rozvrhování: hrana = dvě úlohy požadují stejného operátora
- Rozdíl ve složitosti
  - rezervace: překrývající se časové jednotky jdou po sobě
  - rozvrhování: úlohy často nevyžadují pouze sousední operátory

  $\Rightarrow$ rozvrhování s operátory je obtížnější problém

Rozvrhování s pracovní sílou: aplikace

- Řízení projektu ve stavebním průmyslu
  - dodavatel má realizovat n aktivit
  - doba trvání aktivity j je p_j
  - aktivita j požaduje pracovní skupinu velikosti W_j
  - precedenční omezení na aktivity
  - dodavatel má W pracovníků
  - cíl: dokončit všechny aktivity v minimálním čase

Rozvrhování s pracovní sílou: aplikace

- Řízení projektu ve stavebním průmyslu
  - dodavatel má realizovat n aktivit
  - doba trvání aktivity j je p_j
  - aktivita j požaduje pracovní skupinu velikosti W_j
  - precedenční omezení na aktivity
  - dodavatel má W pracovníků
  - cíl: dokončit všechny aktivity v minimálním čase
- Rozvrhování zkoušek
  - všechny zkoušky mají stejnou dobu trvání
  - všechny zkoušky jsou v místnosti s W sedadly
  - počet studentů předmětu j, kteří skládají zkoušku, je W_j
  - několik zkoušek může být narozvrhováno ve stejné místnosti, pokud je postačující počet sedadel
  - cíl: zkonstruovat rozvrh tak, že jsou všechny zkoušky narozvrhovány v minimálním čase

Reformulace pomocí problému plnění košů *(bin-packing)*

- Problém plnění košů
  - každý koš má kapacitu W
  - předměty velikosti W_j
  - cíl: naplnit minimální počet košů

Reformulace pomocí problému plnění košů *(bin-packing)*

- Problém plnění košů
  - každý koš má kapacitu W
  - předměty velikosti W_j
  - cíl: naplnit minimální počet košů
- Uvažujme speciální problém rozvrhování s pracovní silou
  - předpokládejme jednotkovou dobu trvání
  - nelimitovaný počet strojů
  - minimalizace *makespan*
- Rozvrhování s pracovní silou jako problém plnění košů
  - předmět = úloha (s počtem pracovníků W_j)
  - kapacita koše = celkový počet pracovníků W
  - 1 koš = 1 časová jednotka
  - minimalizace počtu košů = minimalizace makespan

Reformulace pomocí problému plnění košů *(bin-packing)*

- Problém plnění košů
  - každý koš má kapacitu W
  - předměty velikosti W_j
  - cíl: naplnit minimální počet košů
- Uvažujme speciální problém rozvrhování s pracovní silou
  - předpokládejme jednotkovou dobu trvání
  - nelimitovaný počet strojů
  - minimalizace *makespan*
- Rozvrhování s pracovní silou jako problém plnění košů
  - předmět = úloha (s počtem pracovníků W_j)
  - kapacita koše = celkový počet pracovníků W
  - 1 koš = 1 časová jednotka
  - minimalizace počtu košů = minimalizace makespan
- Řešení problému plnění košů
  - NP-těžký problém
  - vyvinuta řada heuristik
  - heuristika prvního padnoucího *(first fit FF)* koše

  víme, že:
  $$C_{max}(FF) \leq \frac{17}{10} C_{max}(OPT) + 2$$

Příklad: heuristika prvního padnoucího koše (FF)

- Předpokládejme 18 úloh a W=2100
  - úloha 1-6 požaduje 301 jednotek zdroje
  - úloha 7-12 požaduje 701 jednotek zdroje
  - úloha 13-18 požaduje 1051 jednotek zdroje
- FF heuristika:

Příklad: heuristika prvního padnoucího koše (FF)

- Předpokládejme 18 úloh a W=2100
  - úloha 1-6 požaduje 301 jednotek zdroje
  - úloha 7-12 požaduje 701 jednotek zdroje
  - úloha 13-18 požaduje 1051 jednotek zdroje
- FF heuristika:
  - přiřadíme prvních 6 úloh na jeden zdroj (301×6=1806)
  - pak přiřadíme vždy dvě úlohy na další tři zdroje (701×2=1402)
  - pak přiřadíme právě jednu úlohu na každý zdroj
- Velmi nekvalitní řešení vzhledem k uspořádání úloh

Heuristika prvního padnoucí koše se zmenšováním úloh

- Zlepšení FF heuristiky
- Uspořádání úloh od nejdelší k nejkratší
- První padnoucí koš se zmenšováním úloh
  (first fit decreasing FFD)
- Řešení příkladu:

Heuristika prvního padnoucí koše se zmenšováním úloh

- Zlepšení FF heuristiky
- Uspořádání úloh od nejdelší k nejkratší
- První padnoucí koš se zmenšováním úloh
  (first fit decreasing FFD)
- Řešení příkladu:
  - seřadíme úlohy dle požadovaných jednotek zdroje, tj.
    13,14,15,16,17,18,7,8,9,10,11,12,1,2,3,4,5,6
  - úlohy bereme postupně a dáváme je na první zdroj, kam se vejdou
    13 dáme na zdroj 1, 14 dáme na zdroj 2, ..., 18 dáme na zdroj 6
    7 dáme na zdroj 1, 8 dáme na zdroj 2, ..., 12 dáme na zdroj 6
    1 dáme na zdroj 1, 2 dáme na zdroj 2, ..., 6 dáme na zdroj 6

Heuristika prvního padnoucí koše se zmenšováním úloh

- Zlepšení FF heuristiky
- Uspořádání úloh od nejdelší k nejkratší
- První padnoucí koš se zmenšováním úloh
  (first fit decreasing FFD)
- Řešení příkladu:
  - seřadíme úlohy dle požadovaných jednotek zdroje, tj.
    13,14,15,16,17,18,7,8,9,10,11,12,1,2,3,4,5,6
  - úlohy bereme postupně a dáváme je na první zdroj, kam se vejdou
    13 dáme na zdroj 1, 14 dáme na zdroj 2, ..., 18 dáme na zdroj 6
    7 dáme na zdroj 1, 8 dáme na zdroj 2, ..., 12 dáme na zdroj 6
    1 dáme na zdroj 1, 2 dáme na zdroj 2, ..., 6 dáme na zdroj 6
- Víme, že

$$C_{max}(FFD) \leq \frac{11}{9} C_{max}(OPT) + 4$$

- FF i FFD mohou být rozšířeny
  - různé termíny dostupnosti

Diskuse

- Uvažovali jsme reprezentanty různých problémů
- V praxi
  - obecnější problémy (kombinace všech uvažovaných rysů problémů zároveň)
  - dynamické rezervační problémy
  - uvažování ceny (management zisku)
  - přerušení aktivit
  - ...

Shrnutí

Rezervace

- Intervalové rozvrhování (rezervační systém bez rezervy)
  - celočíselné programování
  - jednotková doba trvání: formulace problému přiřazení (řešení pro každou čas. jednotku zvlášť)
  - jednotková váha & identické stroje (maximalizace počtu provedených aktivit)
  - jednotková váha & nelimitovaný počet identických strojů
- Rezervační systém s rezervou
  - kompozitní řídící pravidlo

Timetabling

- plánování projektu s omezenými zdroji (RCPSP)
- rozvrhování s operátory a barvení grafu
  - heuristika pro barvení grafu
- rozvrhování s pracovní silou a problém plnění košů
  - heuristika prvního padnoucího koše
  - heuristika prvního padnoucího koše se zmenšováním úloh

Rozvrhování předmětů na univerzitě

16. května 2022

Rozvrhování předmětů na univerzitě

Typy problémů

- rozvrhování se studijními obory (curriculum-based timetabling)
  - curriculum (studijní obor): množina předmětů
  - každý studen je zapsán do (jednoho nebo více) curricula
  - cíl: rozvrhování všech předmětů curricula bez překrývání
  - typické také pro rozvrhování na střední škole

Rozvrhování předmětů na univerzitě

Typy problémů

- rozvrhování se studijními obory (curriculum-based timetabling)
  - curriculum (studijní obor): množina předmětů
  - každý studen je zapsán do (jednoho nebo více) curricula
  - cíl: rozvrhování všech předmětů curricula bez překrývání
  - typické také pro rozvrhování na střední škole
- rozvrhování se zápisy studentů (enrollment-based timetabling)
  - každý student je individuálně zapsán/zaregistrován do nějaké množiny předmětů
  - studentský konflikt: student není schopen absolvovat (dva) zaregistrované předměty vzhledem k jejich překryvu
  - cíl: nalezení rozvrhu, který minimalizuje počet studenských konfliktů
  - př. rozvrhování na FI

Rozvrhování předmětů na univerzitě

Typy problémů

- rozvrhování se studijními obory (curriculum-based timetabling)
  - curriculum (studijní obor): množina předmětů
  - každý studen je zapsán do (jednoho nebo více) curricula
  - cíl: rozvrhování všech předmětů curricula bez překrývání
  - typické také pro rozvrhování na střední škole
- rozvrhování se zápisy studentů (enrollment-based timetabling)
  - každý student je individuálně zapsán/zaregistrován do nějaké množiny předmětů
  - studentský konflikt: student není schopen absolvovat (dva) zaregistrované předměty vzhledem k jejich překryvu
  - cíl: nalezení rozvrhu, který minimalizuje počet studenských konfliktů
  - př. rozvrhování na FI
- dělení studentů na skupiny (student sectioning/student scheduling)
  - dělení studentů na skupiny pro velké předměty, kde je nutné několik seminárních nebo přednáškových skupin

Iniciální tvorba rozvrhu (vytvoření rozvrhu ze začátku) vs.
Interaktivní rozvrhování: změny vytvořeného rozvrhu

Rozvrhovací systém UniTime `http://www.unitime.org`

Vyvinutý na Purdue University (USA) ve spolupráci s FI MU a MFF UK

Komplexní systém pro univerzitní rozvrhování

- rozvrhování předmětů se studijními obory i i se zápisy
  dělení studentů na skupiny
  iniciální tvorba rozvrhu, interaktivní rozvrhování
- rozvrhování studentů, zkoušek, událostí
- otevřený zdrojový kód
- webové rozhraní, Java, SQL+hibernate, XML

Rozvrhovací systém UniTime `http://www.unitime.org`

Vyvinutý na Purdue University (USA) ve spolupráci s FI MU a MFF UK

Komplexní systém pro univerzitní rozvrhování

- rozvrhování předmětů se studijními obory i i se zápisy
  dělení studentů na skupiny
  iniciální tvorba rozvrhu, interaktivní rozvrhování
- rozvrhování studentů, zkoušek, událostí
- otevřený zdrojový kód
- webové rozhraní, Java, SQL+hibernate, XML

Použití

- používáno pro rozvrhování na Purdue University
  - 70 různých problemů, celkem asi 40 000 studentů
- Masarykova univerzita: používáno na téměř všech fakultách
- např. MIT, USA, Lahore University of Management Sciences, Pakistán, University of Zagreb, Chorvatsko, AGH University of Science and Technology, Polsko, Antalya International University, Turecko, Universidad de Ingeniería y Tecnología (UTEC), Peru, American College of Middle East (ACM), Kuwait

Literatura

Materiál k přednášce

- přehledová práce – rozvrhování na Purdue University

  H. Rudová, T. Müller, K. Murray, Complex university course timetabling. *Journal of Scheduling*, 14(2): 187-207, Springer, 2011.

  `http://dx.doi.org/10.1007/s10479-010-0828-5`

Další materiály

- `http://www.unitime.org/publications.php`
- rozvrhování na Filozofické fakultě MU

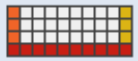
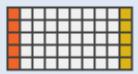
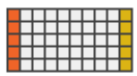
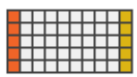
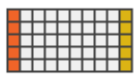
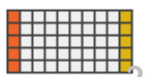
  H. Rudová and T. Müller, Rapid Development of University Course Timetables (extended abstract). Proceedings of the 5th Multidisciplinary International Scheduling Conference (MISTA 2011), pages 649–652. 2011

  `http://www.fi.muni.cz/~hanka/publ/mista11.pdf`
- rozvrhování na Pedagogické fakultě MU a FSpS

  T. Müller, H. Rudová, Real-life Curriculum-based Timetabling with Elective Courses and Course Sections. Annals of Operations Research, 239(1):153-170, Springer, 2016.

  `http://dx.doi.org/10.1007/s10479-014-1643-1`

Struktura předmětu s jeho třídami (vyučovanými částmi)

	Demand	Mins Per Week	Limit	Manager	Date Pattern	Time Pattern	Preferences: Time	Preferences: Room	Preferences: Distribution	Instructor
M E 263 M E 263H	98		96							
Lecture		150	96	LLR	Full Term	3 x 50 2 x 75		WTHR Computer		
Recitation		100	96	M E	Full Term	2 x 50		ME 120 ME 236 Classroom		
Laboratory		50	84-120	LAB	Even Wks	1 x 50		Windows XP		
Lec 1		150	96	LLR	Full Term	3 x 50 2 x 75		WTHR Computer		J. Smith C. Bing
Rec 1		100	48	M E	Full Term	2 x 50		ME 120 ME 236 Classroom	Back-To-Back M E 263 Rec 1 M E 263 Rec 2	J. Novak
Lab 1		50	14-20	LAB	Even Wks	1 x 50		Windows XP		
Lab 2		50	14-20	LAB	Even Wks	1 x 50		Windows XP		
Lab 3		50	14-20	LAB	Even Wks	1 x 50		Windows XP		
Rec 2		100	48	M E	Full Term	2 x 50		ME 120 ME 236 Classroom	Back-To-Back M E 263 Rec 1 M E 263 Rec 2	J. Novak
Lab 4		50	14-20	LAB	Odd Wks	1 x 50		Windows XP		
Lab 5		50	14-20	LAB	Odd Wks	1 x 50		Windows XP		
Lab 6		50	14-20	LAB	Odd Wks	1 x 50		Mac Os X		

Typická omezení

		Pevná omezení	Měkká omezení
Časy pro třídy*	časové vzory (pro pravidelné časy)	x	
	individuální časy	x	x
Místnosti pro třídy	individuální budovy/místnosti	x	x
	individuální vybavení místnosti	x	x
Omezení na zdroje	místnosti	x	
	vyučující	x	
Studenti	konflikty mezi dvěma třídami		x
Distribuční omezení mezi třídami	časové závislosti mezi třídami	x	x
	časové precedence mezi třídami	x	x
	třídy rozvrhované v podobných časech	x	x
	stejný nebo odlišný den/čas/místnost výuky pro třídy	x	x

*Třída je každá rozvrhovaná položka předmětu (přednáška, cvičení, ...)

Omezení a účelové funkce

Pevné podmínky

- musí být splněny

Měkké podmínky

- nemusí být splněny, pokud je to nutné

Měkké podmínky v rozvrhování: přehled

- studentské konflikty
- měkká omezení na čas
- měkká omezení na místností
- měkké distribuční podmínky

Vážený problém splňování podmínek (weighted CSP, WCSP) P

- zahrnuje pevné a měkké podmínky
- cíl: minimalizace F jako součtu vah nesplněných měkkých podmínek

Iterativní dopředné prohledávání (Iterative Forward Search, IFS) pro WCSP

P: WCSP, F: účelová funkce, $<$: komparátor

```
 1: function IFS(P, F, <)
 2:     i = 0
 3:     ω = ∅                                  (současné přiřazení/rozvrh)
 4:     σ = ∅                                  (nejlepší přiřazení/rozvrh)
 5:     while canContinue(ω, i) do
 6:         i = i + 1
 7:         v = selectVariable(P, ω)           (v reprezentuje třídu)
 8:         d = selectValue(P, ω, F, <, v)     (d reprezentuje umístění v rozvrhu)
 9:         γ = hardConflicts(P, ω, v/d)       (předměty, které musím odpřiřadit)
10:         ω = ω\γ ∪ {v/d}
11:         if F(ω) < F(σ) then σ = ω
12:     end while
13:     return σ
14: end function
```

Poznámka: není používána žádná propagace omezení
proměnná má buď přiřazenu iniciální hodnotu nebo má plnou iniciální doménu

Funkce v IFS

Nalezení konfliktních proměnných $\gamma = \text{hardConflicts}\,(P, \omega, v/d)$

- vypočítá množinu proměnných γ takovou, že $\omega \backslash \gamma \cup \{v/d\}$ neporuší žádnou pevnou podmínku
- lze aplikovat jednoduchý algoritmus – vyšší počet iterací je lepší než sofistikovány algoritmus

Výběr proměnné $\text{selectVariable}(P, \omega)$

- nevýznamný – chyby mohou být odstraněny budoucími iteracemi
- aplikováno náhodné uspořádání

Výběr hodnoty $\text{selectValue}(P, \omega, F, <, v)$

- velmi důležitý
- pro minimalizaci porušení měkkých podmínek:
  - výběr hodnoty d proměnné v s minimálním zhoršením $\Delta F(\omega, v/d)$ hodnoty účelové funkce s ohledem na měkká omezení
  - $\Delta F(\omega, v/d) = F(\omega \cup \{v/d\}) - F(\omega)$ (výpočet inkrementální)
- pro minimalizaci porušení pevných podmínek: konfliktní statistika

Konfliktní statistika pro třídu CS 101 Lab 2

Current Assignment of C S 101 Lab 2

Not assigned.

Room Locations: 1 (EDUC 108)

Time Locations: 3 (M 9:30a, M 11:30a, M 1:30p)

Conflict-based Statistics

- 2123× Room EDUC 108
  - 718× M 11:30a - 1:20p Full Term EDUC 108
    - 260× C S 101 Lab 3 ← M 11:30a - 1:20p Full Term EDUC 108
    - 235× C S 101 Lab 1 ← M 11:30a - 1:20p Full Term EDUC 108
    - 222× C S 101 Lab 4 ← M 11:30a - 1:20p Full Term EDUC 108
    - 1× ENGR 101 Lab 2 ← M 11:30a - 1:20p Full Term EDUC 108
  - 718× M 1:30p - 3:20p Full Term EDUC 108
    - 256× C S 101 Lab 1 ← M 1:30p - 3:20p Full Term EDUC 108
    - 235× C S 101 Lab 4 ← M 1:30p - 3:20p Full Term EDUC 108
    - 226× C S 101 Lab 3 ← M 1:30p - 3:20p Full Term EDUC 108
    - 1× ENGR 101 Lab 2 ← M 1:30p - 3:20p Full Term EDUC 108
  - 687× M 9:30a - 11:20a Full Term EDUC 108
    - 252× C S 101 Lab 1 ← M 9:30a - 11:20a Full Term EDUC 108
    - 240× C S 101 Lab 4 ← M 9:30a - 11:20a Full Term EDUC 108
    - 192× C S 101 Lab 3 ← M 9:30a - 11:20a Full Term EDUC 108

Konfliktní statistika

Předpoklad: při výběru hodnoty a proměnné A je nutné zrušit přiřazení hodnoty b proměnné B, tj. $[A = a \rightarrow \neg\, B = b]$

V průběhu výpočtu si tedy lze pamatovat:

$A = a \quad \Rightarrow \quad 3 \times \neg\, B = b, \quad 4 \times \neg\, B = c, \quad 1 \times \neg\, C = a, \quad 120 \times \neg\, D = a$

Konfliktní statistika

Předpoklad: při výběru hodnoty a proměnné A je nutné zrušit přiřazení hodnoty b proměnné B, tj. $[A = a \rightarrow \neg\, B = b]$

V průběhu výpočtu si tedy lze pamatovat:

$$A = a \quad \Rightarrow \quad 3 \times \neg\, B = b, \quad 4 \times \neg\, B = c, \quad 1 \times \neg\, C = a, \quad 120 \times \neg\, D = a$$

Při výběru hodnoty

- výběr hodnoty s nejnižším počtem konfliktů váženým jejich frekvencí
  - **konflikt započítáme pouze, pokud to vede k odstranění přiřazení**
- př. $A = a \quad \Rightarrow \quad 3 \times \neg\, B = b, \quad 90 \times \neg\, B = c, \quad 1 \times \neg\, C = a, \quad 120 \times \neg\, D = a$

  $A = b \quad \Rightarrow \quad 1 \times \neg\, B = a, \quad 3 \times \neg\, B = b, \quad 2 \times \neg\, C = a$

  za předpokladu, že máme přiřazení $B = c, C = a, D = b$
  - nechť A/a vede ke konfliktu s B/c: vyhodnoceno jako 90
    - není konflikt s C/a, tak se nezapočítává
  - nechť A/b vede ke konfliktu s C/a: vyhodnoceno jako 2
  - tj. vybereme hodnotu b pro proměnnou A

Konfliktní statistika II.

- $CBS[x = d_x \rightarrow \neg\, y = d_y] = c_{xy}$: při přiřazení $x = d_x$ bylo nutné zrušit přiřazení $y = d_y$ v minulosti c_{xy}-krát

Konfliktní statistika II.

- $CBS[x = d_x \rightarrow \neg\, y = d_y] = c_{xy}$: při přiřazení $x = d_x$ bylo nutné zrušit přiřazení $y = d_y$ v minulosti c_{xy}-krát

- Jestliže je hodnota d vybrána pro proměnnou v v IFS, potom hardConflicts$(P, \omega, v/d)$ vypočítá přiřazení $\gamma = \{v_1/d_1, v_2/d_2, \ldots v_n/d_n\}$, které musí být zrušeno, aby byla vynucena konzistence nového částečného přiřazení
  Jako důsledek jsou navýšeny čítače

$$CBS[v = d \rightarrow \neg\, v_1 = d_1], \quad CBS[v = d \rightarrow \neg\, v_2 = d_2], \ \ldots,$$
$$CBS[v = d \rightarrow \neg\, v_n = d_n] \ .$$

Konfliktní statistika II.

- $CBS[x = d_x \rightarrow \neg\, y = d_y] = c_{xy}$: při přiřazení $x = d_x$ bylo nutné zrušit přiřazení $y = d_y$ v minulosti c_{xy}-krát

- Jestliže je hodnota d vybrána pro proměnnou v v IFS, potom hardConflicts$(P, \omega, v/d)$ vypočítá přiřazení $\gamma = \{v_1/d_1, v_2/d_2, \ldots v_n/d_n\}$, které musí být zrušeno, aby byla vynucena konzistence nového částečného přiřazení
  Jako důsledek jsou navýšeny čítače

$$CBS[v = d \rightarrow \neg\, v_1 = d_1], \quad CBS[v = d \rightarrow \neg\, v_2 = d_2], \ \ldots,$$
$$CBS[v = d \rightarrow \neg\, v_n = d_n] \ .$$

- Konfliktní statistika je používána jako heuristika pro výběr hodnoty
  Evaluace hodnoty d proměnné v:

$$\sum_{v_i/d_i \in \omega \wedge v_i/d_i \in \text{hardConflicts}(P,\omega,v/d)} CBS[v = d \rightarrow \neg v_i = d_i]$$

tj. konflikt započítáme pouze tehdy, pokud to vede k odstranění přiřazení

Interaktivní rozvrhování: návrhy

Uvažovány změny s třídou PSY 120 Lec 5

Suggestions

Score	*Class*	*Date*	*Time*	*Room*	*Students*
+43	PSY 120 Lec 5	Full Term	MWF 7:30a	WTHR 200 → CL50 224	0
+48.4	PSY 120 Lec 5	Full Term	MWF 7:30a → TTh 7:30a	WTHR 200 → CL50 224	+10
+63.3	PSY 120 Lec 5	Full Term	MWF 7:30a → MWF 4:30p	WTHR 200 → LILY 1105	+14
	POL 130 Lec 2	Full Term	MWF 4:30p → MWF 9:30a	LILY 1105 → RHPH 172	
+63.9	PSY 120 Lec 5	Full Term	MWF 7:30a → MWF 4:30p	WTHR 200 → LILY 1105	+16
	POL 130 Lec 2	Full Term	MWF 4:30p	LILY 1105 → FRNY G140	
+63.9	PSY 120 Lec 5	Full Term	MWF 7:30a → MWF 4:30p	WTHR 200 → LILY 1105	+16
	POL 130 Lec 2	Full Term	MWF 4:30p	LILY 1105 → LYNN 1136	

(all 235 possibilities up to 2 changes were considered, top 5 of 22 suggestions displayed) Search Deeper

Opravná verze metody větví a mezí (Repair-based BB)

Algoritmus

- n nejlepších návrhů ω je vráceno uživateli
- prohledávání s časovým limitem
- hodnoty s nejlepší $\Delta F(\omega, v/d)$ prozkoumávány nejdříve
  - konfliktní statistika není brána v úvahu vzhledem k výpočetní náročnosti

Meze

- omezená hloubka prohledávání
  - abychom umožnili pouze malý počet změn proměnných
  - pro zahrnutí změny na jedné třídě
    nemá smysl měnit příliš mnoho dalších tříd
  - M: maximální hloubka
- hodnota účelové funkce F musí být lepší než n-tý nejlepší návrh
  - $\Omega[n]$: n-tý nejlepší návrh

Opravná verze metody větví a mezí (Repair-based BB)

Algoritmus

- n nejlepších návrhů ω je vráceno uživateli
- prohledávání s časovým limitem
- hodnoty s nejlepší $\Delta F(\omega, v/d)$ prozkoumávány nejdříve
  - konfliktní statistika není brána v úvahu vzhledem k výpočetní náročnosti

Meze

- omezená hloubka prohledávání
  - abychom umožnili pouze malý počet změn proměnných
  - pro zahrnutí změny na jedné třídě
    nemá smysl měnit příliš mnoho dalších tříd
  - M: maximální hloubka
- hodnota účelové funkce F musí být lepší než n-tý nejlepší návrh
  - $\Omega[n]$: n-tý nejlepší návrh

Opakování RepairBB: provádění nového RepairBB

- se zvětšenou hloubkou prohledávání a/nebo
- zvětšeným časovým limitem

Opravná verze metody větví a mezí

- P: WCSP
- ω: současné přiřazení
- v_{bb}: proměnná (třída), která bude přiřazována

1: **function** RepairBB(P, ω, v_{bb})
2: **if** $\{v_{bb}/d\} \subseteq \omega$ **then** $\omega = \omega \backslash \{v_{bb}/d\}$
3: **else** $d = nil$
4: $\gamma = \{v_{bb}/d\}$
5: **return** backtrack($P, \omega, \emptyset, \gamma, \emptyset, 0$)
6: **end function**

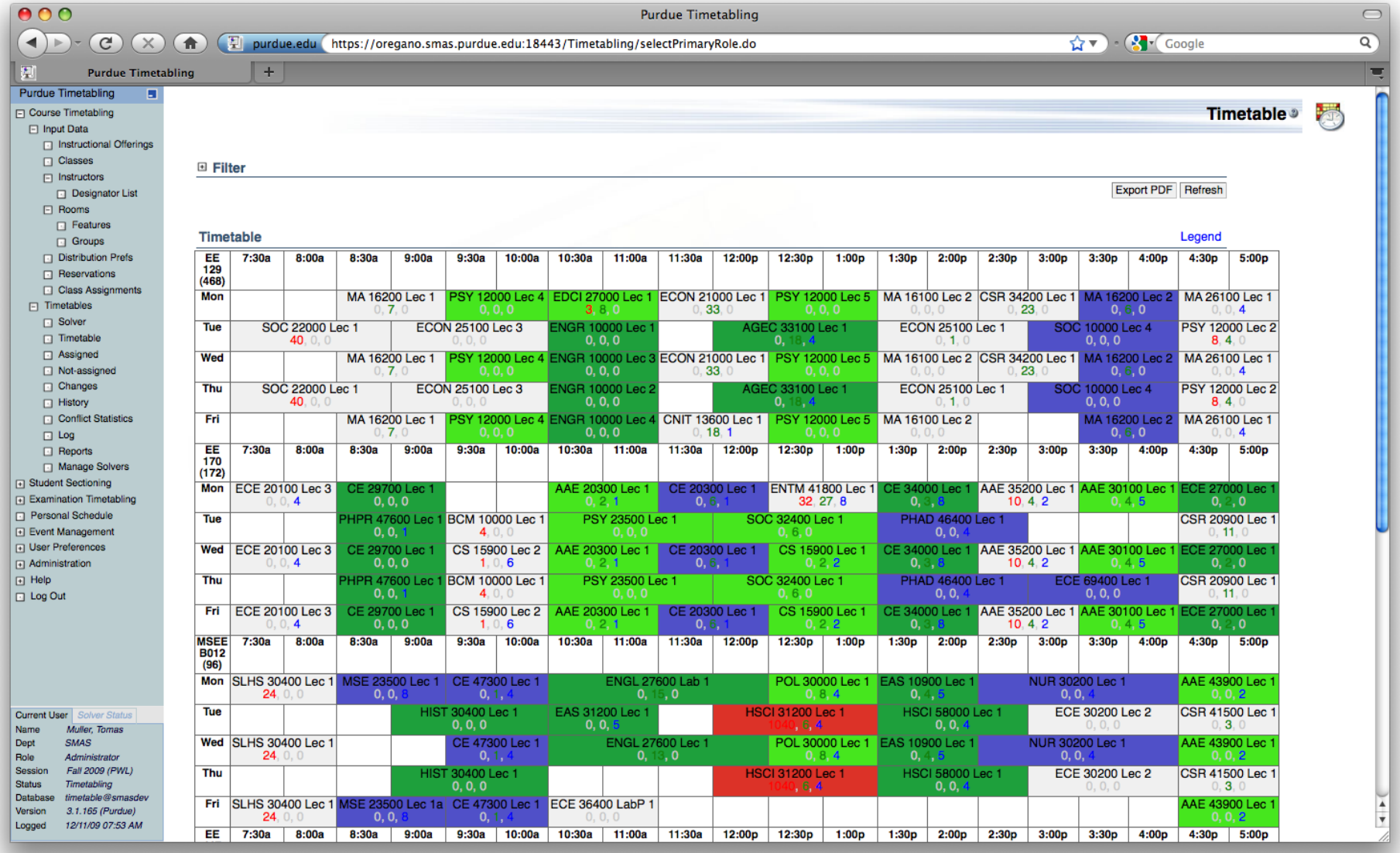
backtrack($P, \omega, \mu, \gamma, \Omega, m$)

- μ: nově vybrané přiřazení aktuálním prohledáváním do hloubky
- γ: proměnné (s případným původním přiřazením), pro které hledáme přiřazení
- Ω: návrhy (několik dosud nejlepších nalezených přiřazení)
- m: aktuální hloubka prohledávání

Funkce backtrack

1: **function** backtrack($P, \omega, \mu, \gamma, \Omega, m$)
2: **if** $\|\gamma\| + m > M$ **then return** $\emptyset$ *(M je maximální hloubka)*
3: **if** $\gamma = \emptyset$ **then return** ω *(konflikty vyřešeny)*
4: **if** timeout **then return** $\emptyset$
5: **if** *LowerBound*$(F(\omega \cup \gamma)) \geq F(\Omega[n])$ **then return** $\emptyset$
(odhad kvality nového přiřazení (po zahrnutí γ) je horší než n-tý návrh)
6: $v =$ selectVariableBB(γ) *(je vybrána některá nepřiřazená proměnná)*
7: **let** $v/d_o \in \gamma$ *(d_0 je původní hodnota proměnné v)*
8: **for** $d \in D_v$ **ordered by** $\Delta F(\omega, v/d)$ **do**
9: **if** $d = d_o$ **then continue** *(je vybrána původní hodnota)*
10: $\alpha =$ hardConflicts$(P, \omega, v/d)$
11: **if** $\alpha \cap \mu \neq \emptyset$ **then continue** *(konflikt s už vybraným přiřazením)*
12: $\Omega = \Omega \cup$ backtrack$(P, \omega \cup \{v/d\}, \mu \cup \{v/d\},$
$\gamma \backslash \{v/d_o\} \cup \alpha, \Omega, m+1)$
13: **end for**
14: **return** Ω
15: **end function**

UniTime.org: GUI s vygenerovaným rozvrhem



Fakulta informatiky MU

	Jaro 2014	Podzim 2014
Místnosti	14+5	14+6
Vyučující	197	231
Předměty	190	199
Třídy	500	604
Registrace	12 701^{+}	14 055^{+}
Registrace + obory	17 599	20 670
Konflikty dop.průchody		4x
Konflikty	958	1 440
Preference na čas	75,89 %	78,2 %

$^{+}$ odstraněny registrace cca 25 studentů s více než 20 předměty

International Timetabling Competition 2019

- Mezinárodní soutěž, kterou organizovala i FI MU
  - `https://www.itc2019.org`
  - rozvrhování univerzitních předmětů
- Vychází z reálných problémů shromážděných v systému UniTime
  - řešení problémů z celého světa včetně rozvrhování FI MU
  - anonymizovaná data
  - málo významné charakteristiky problémů odstraněny
    - snaha soustředit se na důležité rysy problému
- Průběh soutěže
  - 3×10 datových sad publikováno během soutěžního období
  - dostupný validátor řešení – založený na řešiči UniTime
  - submitování validních řešení přes web soutěže
- Výsledky
  - zahrnuty tři problémy z FI MU
  - vítězný řešič časově náročný
    matheuristika: matematické programování kombinované s heuristikami

International Timetabling Competition 2019

- Mezinárodní soutěž, kterou organizovala i FI MU
  - `https://www.itc2019.org`
  - rozvrhování univerzitních předmětů
- Vychází z reálných problémů shromážděných v systému UniTime
  - řešení problémů z celého světa včetně rozvrhování FI MU
  - anonymizovaná data
  - málo významné charakteristiky problémů odstraněny
    - snaha soustředit se na důležité rysy problému
- Průběh soutěže
  - 3×10 datových sad publikováno během soutěžního období
  - dostupný validátor řešení – založený na řešiči UniTime
  - submitování validních řešení přes web soutěže
- Výsledky
  - zahrnuty tři problémy z FI MU
  - vítězný řešič časově náročný (výsledky po 24 hodinách i déle)
    matheuristika: matematické programování kombinované s heuristikami
  - řešič založený na UniTime druhý nejlepší
    schopný produkovat výsledky v krátkém čase (hodina až dvě)

Rozvrhování předmětů na univerzitě: shrnutí

- Typy řešených problémů
  - studijní obory, zápisy, dělení na skupiny
  - iniciální vs. interaktivní rozvrhování
- UniTime
- Model problému
  - struktura předmětu
  - omezení a účelové funkce
- Prohledávání
  - iterativní dopředné prohledávání
  - konfliktní statistika
  - opravná verze metody větví a mezí

Zdroje, ze kterých průsvitky čerpají

V průsvitkách jsou použity obrázky a texty z uvedených zdrojů

- Michael Pinedo, Planning and Scheduling in Manufacturing and Services. Springer, 2005.
- Erwin Hans, Johann Hurink, Production Planning. Přednáška na University of Twente, Nizozemí. `http://www.stern.nyu.edu/om/faculty/pinedo/book2/dowload.html`
- Sanja Petrovič, Automated Scheduling. Přednáška na University of Nottingham, UK.
- Sigurdur Olafsson, Production Scheduling. Přednáška na Iowa State University, USA, `http://www.stern.nyu.edu/om/faculty/pinedo/book2/dowload.html`
- Roman Barták, Plánování a rozvrhování. Přednáška na MFF UK, `http://kti.ms.mff.cuni.cz/~bartak/planovani/prednaska.html`
- Thom Frühwirth and Slim Abdennadher. Essentials of Constraint Programming, Springer Verlag, 2003. `http://www.informatik.uni-ulm.de/pm/fileadmin/pm/home/fruehwirth/pisa/`
- Hana Rudová, Tomáš Müller and Keith Murray, Complex university course timetabling. Journal of Scheduling, 14(2): 187-207, Springer, 2011. `http://dx.doi.org/10.1007/s10951-010-0171-3`
- IBM ILOG CP optimizer for scheduling, Philippe Laborie et al., Constraints, 23(2):210-250, 2018