Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Tomáš Tichý

Pravděpodobnostní on-line algoritmy pro rozvrhování

Katedra aplikované matematiky

Vedoucí diplomové práce: **doc. RNDr. Jiří Sgall, PhD., MÚ AV ČR**

Studijní program: **Informatika**

Studijní obor: **Diskrétní matematika a optimalizace**

PRAHA 2001

Poděkování

Na tomto místě bych chtěl poděkovat zejména svému vedoucímu diplomové práce doc. RNDr. Jiřímu Sgallovi, PhD. za řadu podnětných nápadů, pomoc při odstraňování chyb, za náměty pro zdokonalování práce a za řadu cenných rad.

Použitý software

Tato práce byla vysázena českou verzí programu LaTeX pod operačním systémem **Linux 2.0 Debian** s použitím řady dalších pomocných programů s GNU licencí. Hlavní prezentované výpočty byly realizovány s pomocí programu pro matematické výpočty **BC** s GNU licencí. Některé maticové výpočty byly provedeny komerčním programem **Maple** s platnou licencí pro Univerzitu Karlovu.

Kontakt

E-mail: *tom@atrey.karlin.mff.cuni.cz* nebo *xtom@go.to*
Domácí stránka: *http://kiwi.ms.mff.cuni.cz/~tom*
Diplomová práce: *http://kiwi.ms.mff.cuni.cz/~tom/diplomka/*

Prohlášení

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 30. července 2001 Tomáš Tichý

Obsah

Kapitola 1

Úvod do problematiky

Tato diplomová práce se zabývá tzv. pravděpodobnostními on-line algoritmy pro rozvrhování. Následující kapitolky ukáží motivaci této práce, laicky vysvětlí základní pojmy a shrnou historický vývoj.

1.1 Základní myšlenka

Pro pochopení smyslu a motivace této práce je důležité se seznámit se základními definicemi a fakty týkajících se pravděpodobnostních algoritmů obecně.

1.1.1 Pravděpodobnostní vs. deterministické algoritmy

Běžné počítače, které se v praxi používají, jsou deterministickými zařízeními. Proto většina lidí považuje za přirozené, že počítačový program se má také chovat deterministicky - tedy za stejné situace stejně rozhodnout. Pokud se program takto nechová, tak se často usuzuje, že je špatný. Nutno poznamenat, že takovýto závěr je většinou správný a „nedeterministické“ chování programu je opravdu zapříčiněno chybou. Nekorektní chování programu může být způsobeno například kolizí s jinými programy, neboť program nemusí být operačním systémem zcela oddělen od svého okolí.

Avšak existuje i třída programů, pro které se přirozeně předpokládá nedeterministické chování, je to třída počítačových her. Nedeterministické chování je v jejich prospěch, neboť zvyšuje jejich variabilitu a zajímavost. Například u logických her s nedeterministickým chováním typicky nestačí nalézt konkrétní návod umožnující vyhrát narozdíl od deterministických.

Z matematického pohledu jsou předchozí úvahy zkreslující a situace vypadá úplně jinak. Při návrhu programu resp. algoritmu může být uvažován deterministický model, ve kterém se algoritmus musí v každém kroku pevně rozhodnout jen v závislosti na již přečteném vstupu, chování je tedy jednoznačně determinováno vstupem.

Jinou možností je pravděpodobnostní model. Takové algoritmy se nazývají pravděpodobnostní nebo randomizované. Pravděpodobnostní algoritmus se v každém kroku rozhoduje nejen na základě již přečteného vstupu, ale může při rozhodování použít (laicky řečeno) hod mincí. Poznamenejme, že hod mincí je chápán jako náhodný jev nabývající hodnoty 0 nebo 1 s pravděpodobností $\frac{1}{2}$.

1.1.2 Proč zkoumat pravděpodobnostní algoritmy?

Deterministický algoritmus je zřejmě speciálním případem pravděpodobnostního algoritmu, neboť při svém rozhodování nevyužívá hodů mincí. Při uvážení pravděpodobnostních algoritmů se nabízejí následující přirozené otázky:

- **Mají stejnou výpočetní sílu?**

  tj. zda každá úloha řešitelná pravděpodobnostním algoritmem je též řešitelná deterministickým algoritmem?

- **Mají nižší časovou nebo prostorovou složitost?**

  tj. zda pro některé úlohy umožní hledání v průměru „lepších“ (rychlejších či paměťově méně náročných) řešení?

- **Umožní sestrojit lepší deterministické algoritmy?**

  tj. zda pro některé úlohy dají návod, jak odstranit nedeterministické chování pravděpodobnostního algoritmu řešícího úlohu aniž by se příliš zvýšila časová složitost?

Všechny zde vyslovené otázky mají správnou kladnou odpověď. Zdůvodnění odpovědí lze nalézt v následujících odstavcích.

Stejná výpočetní síla

V teorii vyčílitelnosti se algoritmus formalizuje jako program pro Turingův stroj, zároveň je definován i pravděpodobnostní Turingův stroj. Pravděpodobnostní Turingův stroj je obohacený o nekonečnou pásku uniformně náhodných bitů. Existují existují různé ekvivalentní definice pravděpodobnostního Turingova stroje, které se liší zejména v podmínce zastavení.

Teorie vyčíslitelnosti dokazuje větu, že oba tyto modely jsou ekvivalentní. Konkrétní konstrukce důkazu závisí na zvolené definici.

Poznámka: *Determinizace pravděpodobnostních algoritmů provedená podle této věty je značně časově náročná.*

Závěr: Pravděpodobnostní algoritmy mají z hlediska teorie vyčíslitenosti stejnou výpočetní sílu jako deterministické algoritmy.

Typy pravděpodobnostních algoritmů

Pravděpodobnostní algoritmy se navrhují pro řešení různých úloh, sestrojené algoritmy jsou konfrontovány s existujícími deterministickými. Proto je třeba uvážit, co požadovat po algoritmu řešícím danou úlohu.

Nastíněná struktura nemá být brána jako definice. Algoritmy s uvedenou strukturou jsou často konstruované a dávají dobré výsledky, avšak existují i algoritmy s odlišnou strukturou.

- Pravděpodobnostní algoritmus pro **optimalizační úlohu** nalezne (při pevné náhodné posloupnosti) nějaké řešení. Dopočítá se pravděpodobnost, že takto získané řešení je optimální. Vícenásobným iterováním se pravděpodobnost, že řešení je optimální, zvyšuje. Spočte se průměrný počet iterací potřebných k tomu, aby bylo nalezeno hledané optimum. Za složitost takového algoritmu se považuje celková složitost všech těchto iterací.

  Je-li cílem nalézt optimální řešení, pak lze často použít vícenásobné iterování, pravděpodobnost chyby lze omezit libovolně malým kladným číslem. Je třeba, aby každá iterace vracela optimální řešení s pravděpodobností odraženou od 0, například alespoň 0.1.

- Pravděpodobnostní algoritmus pro **rozhodovací úlohu** skončí nějakou odpovědí, například to mohou být ANO, NE, NEPLATÍ ANO, NEPLATÍ NE, NEVÍM, je třeba spočítat (či odhadnout) pravděpodobnosti jednotlivých odpovědí, tyto pravděpodobnosti mohou záviset na správné odpovědi. „Rozumný“ algoritmus lze iterováním zlepšovat, proto se spočte průměrný počet iterací pro nalezení správného řešení, má-li to smysl.

- Má smysl uvažovat i **pravděpodobnostní aproximační algoritmy** pro optimalizační úlohy, v případě rozhodovacích úloh se stává, že algoritmus často neumí definitivně rozhodnout (ANO nebo NE), potom se iterováním ke správné odpovědi může blížit, to se týká například testů prvočíselnosti. V teorii složitosti je definována hierarchie tříd rozhodovacích pravděpodobnostních algoritmů.

V případě optimalizačních úloh může být hledán kompromis mezi přesností výsledku a složitostí výpočtu. Pro některé NP-úplné úlohy to může znamenat nalezení přijatelného řešení s polynomiální složitostí.

Návrh a analýza pravděpodobnostního algoritmu jsou značně individuální a závislé na konkrétní řešené úloze.

Snížení časové nebo prostorové složitosti

Časová resp. paměťová složitost pravděpodobnostního algoritmu se definuje jako průměr přes nekonečné posloupnosti náhodných bitů z časové resp. paměťové složitosti jednotlivých deterministických výpočtů. Pojmem „deterministický výpočet“ se rozumí výpočet algoritmu při pevné nekonečné náhodné posloupnosti.

Složitost se porovnává se složitostí deterministického algoritmu, který řeší stejnou úlohu.

Pravděpodobnostní algoritmy lze úspěšně využít při řešení mnoha úloh, neboť průměrná časová resp. paměťová složitost takového algoritmu může nižší než časová resp. paměťová složitost deterministického algoritmu. Příkladem je problém minimálního hranového řezu.

Derandomizace

Derandomizací pravděpodobnostního algoritmu se rozumí sestrojení deterministického algoritmu, který umožní řešit zadanou úlohu s pomocí tohoto pravděpodobnostního algoritmu a vždy nalezne správné (hledané) řešení.

Jak již bylo zmíněno, tak obecně takový algoritmus může mít neúměrně větší časovou a paměťovou složitost. Je-li pro výpočet algoritmu postačující tzv. polynomiálně velký pravděpodobnostní prostor, pak je možné jej celý probrat a spočítat tak přesné řešení. Příkladem je algoritmus hledání maximálního hranového řezu.

Polynomiálně velký pravděpodobnostní prostor je postačující, například pokud není algoritmem vyžadována nezávislost všech náhodných bitů, ale jen některých podmnožin z nich.

Poznámka: *Pro pravděpodobnostní on-line algoritmy se derandomizace neuvažuje, bude vysvětleno později.*

1.1.3 Co je on-line algoritmus

On-line algoritmus má ve vstupu určené okamžiky, ve kterých musí vypsat na výstup své „aktuální řešení," musí to provést dříve, než přečte další bit vstupu. Toto se samo o sobě od algoritmu, který není on-line, nijak neliší, proto je třeba další omezující požadavek. Klíčový požadavek pro definici on-line algoritmu je vytváření nového řešení z předcházejícího, tj. naposledy vypsaného. Je dovoleno používat jen operace definované zadáním úlohy.

On-line algoritmy mají řadu praktických aplikací ve výrobním procesu, ve kterém je třeba umět okamžitě „rozumně" rozhodnout, např. co zrovna vyrábět atp., bez znalosti budoucích požadavků. Typickým příkladem jejich použití je vytváření rozvrhů pro přidělování nějakých zdrojů, například výrobní stroj, počítač atp. Zde je vynucován požadavek o neměnnosti dřívějšího rozhodnutí, neboť např. o výrobku, který se začal vyrábět na základě rozhodnutí algoritmu, nelze zpětně rozhodnout, že bude vyráběn jiným strojem nebo v jiný čas.

Cílem je konstruovat takové on-line algoritmy, které v každé situaci rozhodnou tak, že toto rozhodnutí „při nejhorším možném pokračování" bude „co nejméně špatné." Toto je řečeno intuitivně a je to třeba definovat pro každou úlohu zvlášť.

Vlastnost „on-line" je pro algoritmus zásadní omezení. Je třeba si uvědomit, že hlavní prioritou bádání není snižování časové či paměťové složitosti, ale přibližování se optimálnímu tzv. „off-line" řešení, které může být spočteno bez on-line omezení.

Poznámka: *Jako protiklad k pojmu* ***on-line algoritmus*** *se používá pojem* ***off-line algoritmus****. Není nijak omezován průběh výpočtu takového algoritmu, ani nemusí hledat částečná řešení.*

Používá se řada algoritmů, které jsou on-line, aniž by to po nich bylo vyžadováno. Do této kategorie lze zařadit tzv. heuristické algoritmy, ve kterých se přidáním nového požadavku (např. úlohy) upraví již existující řešení, typickým příkladem je **hladový algoritmus**.

1.1.4 Pravděpodobnostní vs. deterministické on-line algoritmy

Pravděpodobnostní on-line algoritmy **mají větší výpočetní sílu** než deterministické on-line algoritmy.

Právě uvedený fakt může být na první pohled o to více překvapující, jak bylo v předchozím textu uvedeno, že pravděpodobnostní a deterministické algoritmy mají stejnou výpočetní sílu. Avšak má to velmi jednoduchou příčinu, sice lze pravděpodobnostní algoritmus derandomizovat, ale typicky nelze zachovat vlastnost on-line. V důsledku toho nemá smysl uvažovat derandomizaci pravděpodobnostních on-line algoritmů.

Například pro problém „rozvrhování úloh na dva počítače“ viz. Graham [8] je známý pravděpodobnostní on-line algoritmus, o kterém lze snadno dokázat, že je lepší než každý deterministický on-line algoritmus.

„Kvalita“ pravděpodobnostního on-line algoritmu je měřena funkcí porovnávající výsledky tohoto algoritmu a výsledky, které lze dostat nejlepším možným deterministickým algoritmem. Srovnání je většinou zajímavé. Navíc důkazy dolních odhadů pro deterministické algoritmy jsou snadné, většinou jsou provedeny konstrukcí konkrétního protipříkladu.

1.1.5 Kompetitivní poměr

Aby bylo možné pravděpodobnostní i deterministické on-line algoritmy porovnávat, tak je třeba jejich „kvalitu“ nějak měřit. Tyto algoritmy jsou optimalizační a jejich cílem je co nejvíce se přiblížit hledanému optimu. Proto se definuje tzv. kompetitivní poměr algoritmu jako poměr výsledku on-line algoritmu ku nejlepšímu možnému výsledku.

Nechť zobrazení $h(y)$ je ohodnocující funkce pro výsledný stav y, do kterého se algoritmus dostal svým rozhodováním. Výsledkem $h(y)$ je kladné reálné číslo. Čím je výsledek „lepší“ tím má být ohodnocující funkce menší. Ohodnocovací funkce je dána zadáním problému.

Poznámka: *V úlohách, pro které je stav y definován jako m-tice čísel, je často za ohodnocující funkci zvolena maximalizace nebo nějaký vážený průměr z těchto čísel.*

Nechť zobrazení $S_{\mathcal{A}}(x)$ určuje výsledek, ke kterému dospěje algoritmus $\mathcal{A}$ při vstupu x. Je-li algoritmus $\mathcal{A}$ pravděpodobnostní, potom se uvažuje pravděpodobnostní distribuce výsledků v závislosti na náhodné posloupnosti. Nechť je taková distribuce výsledků označena symbolem $S_{\mathcal{A}}(\mathcal{B}, x)$, kde $\mathcal{B}$ je pevná náhodná posloupnost.

Ohodnocení výsledku $H_{\mathcal{A}}(x)$ deterministického algoritmu $\mathcal{A}$ se spočítá s pomocí ohodnocující funkce jako

$$H_{\mathcal{A}}(x) \stackrel{def}{=} h(S_{\mathcal{A}}(x)) \tag{1.1}$$

Pro pravděpodobnostní algoritmus se ohodnocení výsledku $H_{\mathcal{A}}(x)$ spočte v průměru přes náhodné posloupnosti jako

$$H_{\mathcal{A}}(x) \stackrel{def}{=} \mathbb{E}_{\mathcal{B}}\left[h(S_{\mathcal{A}}(\mathcal{B}, x))\right] \tag{1.2}$$

Poznámka: *Definice* 1.1 *a* 1.2 *jsou zřejmě konzistentní, pro deterministický algoritmus dávají stejný výsledek.*

Nechť zobrazení $S(x)$ určuje optimální řešení úlohy spočtené nejlepším možným off-line algoritmem pro vstup x. Optimální výsledek $H(x)$ se potom definuje jako

$$H(x) \stackrel{def}{=} h(S(x)) \tag{1.3}$$

Potom se definuje zobrazení $c(x)$, které je **kompetitivním poměrem** pro algoritmus $\mathcal{A}$ a vstup x takto

$$c(x) = \frac{H_{\mathcal{A}}(x)}{H(x)} \tag{1.4}$$

Jako ohodnocení výsledku je klasické „nechť nepřítel zadá vstup," tedy nejhorší případ, který může nastat. **Kompetitivní poměr** pro algoritmus $\mathcal{A}$ je definován jako

$$c = \sup_{x} c(x) \tag{1.5}$$

Triviálně lze nahlédnout, že $c \geq 1$, dokonce $c(x) \geq 1$ pro každý vstup x. Neboť žádný algoritmus nemůže být lepší, než nejlepší možný.

Poznámka: *Typicky není cílem hledat přesná vyjádření těchto zobrazení, spíše jsou hledány jejich horní a dolní odhady. Na základě těchto odhadů se podle definice* 1.5 *spočte odhad kompetitivního poměru.*

1.1.6 Problém rozvrhování

Problém rozvrhování má řadu možných podob. Napřed budou shrnuty základní společné rysy a rozdíly.

Úloha rozvrhování známé posloupnosti úloh na m počítačů je NP-těžká. On-line algoritmy jsou leckdy vcelku dobrou aproximací. Je proto zajímavé zkoumat, jak moc dobrou aproximací jsou.

Společné rysy modelů

- Vstupem je posloupnost úloh.
- Každá úloha má určenou velikost.
- Úlohy se zpracovávají na počítačích.
- V jedom okamžiku smí být jeden počítač přiřazen nejvýše jedné úloze.
- V jedom okamžiku smí být jedna úloha přiřazena nejvýše jednomu počítači, případně nejvýše danému počtu počítačů.

Rozdíly mezi modely

Zmiňme příklady uvažovaných variant modelů. Existují i méně obvyklé varianty, které zde nejsou zmíněny.

- počítače
  - jejich počet je **pevný** nebo **proměnný**, v některých modelech lze za nějakou cenu počítače kupovat či prodávat (to se promítne v ohodnocující funkci)
  - mohou mít **různou** rychlost, potom čas dokončení úlohy se může lišit na různých počítačích
  - mohou mít různé **instrukční sady** či **zařízení**, úlohu je možné zpracovat jen na některých počítačích
- úlohy
  - mohou mít **priority**, váhy důležitosti úloh se zohlední v ohodnocující funkci
  - mohou mít **závislosti** na jiných úlohách, úlohu je možné zpracovat až po dokončení jiné, například z technologických důvodů ve výrobním procesu
  - mohou mít **časová omezení**, do kdy je třeba úlohu vykonat
  - mohou být **přerušitelné**, jejich vykonávání lze pozastavit a přenechat počítač jiné úloze
- operační systém
  - může **přerušit** nebo **restartovat** úlohu, to je tzv. preemptivní chování
  - může přesunout úlohu na jiný počítač (procesor)
- ohodnocující funkce
  - délka rozvrhu, tj. čas dokončení poslední úlohy

- součet časů dokončení všech úloh
- počet úloh nevykonaných včas (pro modely s omezeným časem)
- penalizace za nevčasné dokončení

Nejpoužívanější jsou následující modely.

- V **sekvenčním modelu** přicházejí jednotlivé úlohy v posloupnosti, každá má být přiřazena počítači (resp. počítačům) a časovému intervalu (resp. intervalům) dříve, než je známa další úloha posloupnosti. Součástí poskytované charakteristiky příchozí úlohy je její délka. Není omezeno umísťování do časových intervalů, případně úlohy mohou být i odkládány. Hodnotící funkcí může být například délka rozvrhu.

- V **modelu neznámých délek** algoritmus o běžící úloze (úlohách) ví jen to, zda již skončila nebo ne. Může libovolně umísťovat úlohy, které dosud přišly. Charakteristika úlohy neobsahuje její délku. Někdy bývá povolena preempce (odebírání) úloh, odebraná úloha může být později restartována (znovuspuštěna) resp. pokračována. Možné jsou i další charakteristiky, například paralelnost výpočtu na maximálním omezeném počtu počítačů. Hodnotící funkcí může být délka rozvrhu.

- **Model přicházení v čase** je podobný předchozímu modelu neznámých délek s rozdílem, že délka úlohy je známa v čase příchodu úlohy. Tedy komplikací je jen to, že není známa budoucnost. Hodnotící funkcí může být délka rozvrhu.

- **Intervalový model** se od předchozích modelů liší hlavně tím, že nepovoluje odložení úlohy, avšak úloha musí být zpracována v časovém intervalu daném charakteristikou úlohy. Hodnotící funkce je míra vah úloh zařazených do rozvrhu. Tento model se svým zadáním i metodami řešení zásadně liší od ostatních uvedených modelů.

Zvolený model

Deterministický případ problému rozvrhování je prozkoumán do značné hloubky. O pravděpodobnostním on-line případu se toho moc neví. Při zkoumání každého problému je důležité postupovat od jednodušších problémů ke složitějším a každému takovému problému předchází důkladné pochopení problému jednoduššího. Pro zkoumání byl proto zvolen jednoduchý model, tzv. **sekvenční model**. Jak se později ukáže, tak i pro takto jednoduchý model je analýza značně komplikovaná.

- Velikosti úloh jsou přirozená čísla, bez priorit, bez časových omezení, bez závislostí.
- K dispozici je m identických počítačů.
- Bez preempce.
- Ohodnocující funkce je délka rozvrhu (tj. čas dokončení poslední úlohy).

V tomto modelu se od pravděpodobnostního on-line algoritmu po přečtení velikosti úlohy očekavá odpověď, na který počítač má být příchozí úloha umístěna.

Laicky lze tento model dobře popsat na příkladu hry „Tetris.“ Hrajme tedy tuto hru na m sloupečcích (odpovídají počítačům), nepřítel posílá cihly o velikosti $1 \times d$, kde $d \in \mathbb{N}$ je nepevná délka konkrétní cihly, přicházejí svisle a nelze je otáčet. Cihla je umístěna do nějakého sloupečku na dříve umístěné cihly nebo počátek, uvažuje se „gravitace“ - mezery nejsou přípustné. Po umístění cihly nepřítel pošle další. Úkolem hráče je cihly umísťovat tak, aby výška této „stavby“ (poměr ku minimální možné výšce) nebyla „moc velká.“ Pojem „moc velká výška“ se chápe tak, aby kompetitivní poměr (poměr výšky ku minimální možné výšce stavby, jakou lze z daných cihel postavit) byl co nejmenší pro každou nepřítelem zadanou posloupnost cihel.

1.1.7 Shrnutí a vyjasňující poznámky

Pro čtenáře, který není zběhlý ve zkoumání složitosti a složitostních tříd, může být obtížnější si uvědomit, co se přesně zkoumá.

Následující poznámky mají za cíl upozornit na podstatné odlišnosti, kterých je třeba si všímat při čtení této nebo obdobné práce.

Algoritmus

- **deterministický algoritmus**

  Nepřítel zadá celý vstup najednou, rozhodne o umístění všech úloh, vypíše výstup.

- **pravděpodobnostní algoritmus**

  Rozšiřuje deterministický algoritmus, navíc může při rozhodování používat náhodné bity. Převoditelný na deterministický algoritmus za cenu zvýšení časové a paměťové složitosti.

- **determinitistický on-line algoritmus**

  Deterministický algoritmus. Nepřítel zadává úlohy po jedné. Algoritmus musí ihned rozhodnout o umístění úlohy.

- **pravděpodobnostní on-line algoritmus**

  Rozšiřuje determinitistický on-line algoritmus, navíc může při rozhodování používat náhodné bity. **Není** převoditelný na deterministický on-line algoritmus.

Typy odhadů

Odhady se dělí podle toho, jaké algoritmy jsou uvažovány pro hledaní řešení. Proto odhady mohou být tyto:

- **bez omezení**

  Počítáno přes všechny deterministické a pravděpodobnostní algoritmy (jsou převoditelné) řešící daný problém. Zadání dostanou předem. Zkoumá se časová a paměťová efektivita (složitost) algoritmů. Za zajímavé jsou považovány algoritmy počítající v polynomiálním čase.

- **deterministický on-line**

  Počítáno přes deterministické on-line algoritmy. Bez omezení složitosti. Zkoumá se kompetitivní poměr.

- **pravděpodobnostní on-line**

  Počítáno přes pravděpodobnostní on-line algoritmy. Bez omezení složitosti. Zkoumá se kompetitivní poměr.

Dolní odhady kompetitivního poměru

Dělení je podle typu (viz. předchozí odstavec) a následujících možností:

- **dolní odhad problému**

To je dolní odhad na kompetitivní poměr každého algoritmu řešícího daný problém. Speciálně je to dolní odhad i pro optimální (nejlepší) algoritmus řešící daný problém.

- **dolní odhad algoritmu**

  To je dolní odhad na kompetitivní poměr konkrétního algoritmu řešícího daný problém.

- **dolní odhad třídy algoritmů**

  To je obdobné „dolním odhadům," avšak odhad je počítán přes určitou třídu algoritmů. Třída je určena nějakým omezením toho, co algoritmus smí nebo nesmí. Je to tedy dolní odhad pro každý algoritmus v této třídě. Dolní odhad problému je dolním odhadem pro každou třídu algoritmů.

Poznámka: *Kompetitivní poměr je počítán jako supremum, tudíž popisuje nejhorší možný případ, který může nepřítel zadat.*
Poznámka: *Dolní odhad je obvykle konstruován nalezením „těžké" posloupnosti, na které má každý algoritmus kompetitivní poměr o velikosti alespoň dokazovaného dolního odhadu.*

Horní odhady kompetitivního poměru

Dělení je podle typu a následujících možností:

- **horní odhad algoritmu**

  To je horní odhad kompetitivního poměru konkrétního algoritmu řešícího daný problém. Je třeba provést analýzu dokazující, že pro žádnou posloupnost nebude kompetitivní poměr větší.

- **horní odhad problému**

  To je horní odhad kompetitivního poměru nejlepšího algoritmu řešícího daný problém. Kompetitivní poměr každého algoritmu je horním odhadem problému. Dokazuje se konstrukcí konkrétního algoritmu.

- **horní odhad třídy algoritmů**

  To je horní odhad kompetitivního poměru nejlepšího algoritmu z dané třídy řešícího daný problém. Třída je určena omezením toho, co algoritmus smí nebo nesmí. Horní odhad třídy algoritmů je horním odhadem problému. Dokazuje se konstrukcí konkrétního algoritmu.

1.2 Přehled známých výsledků

1.2.1 Historický vývoj

Historicky první, kdo se začal zajímat o tuto problematiku a kdo problém rozvrhování formuloval, byl Graham [8]. V této práci ještě nebyla problematika formulována v později ustálených termínech.

Prvním, kdo přišel s pojmy „on-line" a „off-line" je Knuth [12]. Pojem on-line použil pro algoritmus, který načítá vstupní posloupnost a spočtené výsledky vypisuje, jakmile je to možné. Pojem off-line definoval jako opak on-line. Jeho pojetí se značně liší od současného pojetí (definice).

První práce se současným pojetím definice on-line algoritmů se objevily v 70-tých letech. Týkaly se aproximačních algoritmů a zabývaly se tzv. „problémem batohu." Až poději se začal vyskytovat pojem „aproximační on-line algoritmy," poprvé v práci Johnsona [10]. Pojmy on-line a off-line byly použity v kryptografickém systému.

Dále byly nalezeny metody pro dolní odhady pomocí konstrukce „nepřítelem“ zadaného protipříkladu. Jedny z prvních aplikací této metody byly v práci Woodhal [18] při analýze tzv. „Bay Restaurant problému“ či práce Kierstead a Trotter [11] zabývající se „on-line barvením intervalových grafů.“

Yao [19] je první, kdo přišel s tvrzením začínajícím formulací „pro každý on-line algoritmus platí. . . .“ Dokázal nemožnost existence on-line algoritmu pro „problém batohu“ s kompetitivním poměrem lepším než $\frac{3}{2}$. Tento výsledek se stal základem pro další rozvoj aplikací on-line algoritmů pro řešení optimalizačních problémů.

Významný přelom ve výzkumu on-line algoritmů znamenaly práce publikované v roce 1985, viz. Sleator a Tarjan [16] a [17]. Tyto práce zapříčinily vzrůst zájmu o tento obor teoretické informatiky, který trvá dodnes. Publikované práce se zabývají problémem „stránkování (caching, paging)“ a tzv. „list update problémem.“

Od počátků výzkumu v této oblasti uplynulo mnoho let a byla dokázána řada zajímavých výsledků. Výzkum znamenal velký pokrok, ale přesto je nutné konstatovat, že existuje hodně dosud nevyřešených otevřených problémů. Lze očekávat, že tento obor čeká ještě dlouhý vývoj.

1.2.2 Graham

První on-line algoritmus, který řešil úlohu rozvrhování na m identických počítačích, byl **Grahamův algoritmus**. Tento algoritmus patří mezi tzv. hladové algoritmy, nebot' příchozí úloha je vždy umístěna na počítač, který má minimální zátěž (v „tetrisové analogii“ je úloha umístěna na co nejnižší sloupeček). Kompetitivní poměr algoritmu pro m počítačů je právě $2 - \frac{1}{m}$.

Dolní odhad se provede posloupností $m(m-1)$ jednotkových úloh následovaných úlohou velikosti m. Na každý počítač bylo umístěno $m-1$ jednotkových úloh a dále přišla poslední úloha. Rozvrh má délku $2m-1$. Délka optimálního rozvrhu je m. V optimálním rozvrhu jsou jednotkové úlohy umístěny na $m-1$ počítačích, zbylá úloha velikosti m zůstane samotná. Tudíž kompetitivní poměr je alespoň $\frac{2m-1}{m} = 2 - \frac{1}{m}$.

Optimalita, že algoritmus je $\left(2 - \frac{1}{m}\right)$-kompetitivní, se ukáže snadno. Důkaz se provede pro každou neprázdnou posloupnost (pro prázdnou to je triviální).

Označme t minimum zátěží počítačů před poslední úlohou a x velikost poslední úlohy.

Cílem, dle definice kompetitivního poměru, je dokázat vztah

$$t + x \leq \left(2 - \frac{1}{m}\right) \cdot \mathit{optimum} \tag{1.6}$$

Často se používá dolní odhad pro délku optimálního rozvrhu

$$\max\left\{t + \frac{x}{m}, x\right\} \leq \mathit{optimum} \tag{1.7}$$

Pro důkaz bude postačující varianta výrazu 1.6 s odhadem 1.7

$$t + x \leq \left(2 - \frac{1}{m}\right) \cdot \max\left\{t + \frac{x}{m}, x\right\} \leq \left(2 - \frac{1}{m}\right) \cdot \mathit{optimum} \tag{1.8}$$

Je postačující dokázat vztah 1.8, neboť z něj vyplývá vztah 1.6. Vztah 1.8 se rozborem podle maxima rozpadá na dva případy.

- případ $x \geq t + \frac{x}{m}$
  - z podmínky vyplývá $\frac{t}{x} \leq 1 - \frac{1}{m}$
  - odstranění maxima podle rozboru

$$\frac{t+x}{x} \leq \left(2 - \frac{1}{m}\right) \tag{1.9}$$

- důkaz posloupností úprav

$$\frac{t+x}{x} = \frac{t}{x} + 1 \leq 1 + 1 - \frac{1}{m} = 2 - \frac{1}{m} \tag{1.10}$$

- případ $t + \frac{x}{m} \geq x$
  - z podmínky vyplývá $\frac{t}{x} \geq 1 - \frac{1}{m}$
  - odstranění maxima podle rozboru

$$\frac{t+x}{t+\frac{x}{m}} \leq \left(2 - \frac{1}{m}\right) \tag{1.11}$$

  - důkaz posloupností úprav

$$\frac{t+x}{t+\frac{x}{m}} = \frac{\frac{t}{x}+1}{\frac{t}{x}+\frac{1}{m}} = \frac{\frac{t}{x}+\frac{1}{m}-\frac{1}{m}+1}{\frac{t}{x}+\frac{1}{m}} = 1 + \frac{1-\frac{1}{m}}{\frac{t}{x}+\frac{1}{m}} \leq 2 - \frac{1}{m} \tag{1.12}$$

Tudíž vztah 1.6 je splněn pro všechna $x > 0$, $t \geq 0$.

Závěr: Grahamův algoritmus je $\left(2 - \frac{1}{m}\right)$-kompetitivní. Odhad pro Grahamův algoritmus již nelze zlepšit.

1.2.3 Odhady deterministických algoritmů

Graham ve své práci [8] ukázal deterministický on-line $\left(2 - \frac{1}{m}\right)$-kompetitivní algoritmus pro rozvrhování na m počítačů.

Faigle, Kern a Turán ukázali, že Grahamův algoritmus je optimální pro $m = 2, 3$. Pro $m \geq 4$ dokázali dolní odhad $1 + \frac{1}{\sqrt{2}} \approx 1.707$.

Galambos a Woeginger dokázali existenci $\left(2 - \frac{1}{m} - \varepsilon_m\right)$-kompetitivního on-line algoritmu pro $m \geq 4$ počítačů. Avšak tento odhad podobně jako Grahamův odhad pro m jdoucí do nekonečna konverguje ke 2. Konstanty $\varepsilon_m > 0$ jsou dané důkazem.

Dlouho byl otevřený problém, zda existuje on-line algoritmus s kompetitivním poměrem menším než 2 pro libovolný počet počítačů (v limitě pro

m jdoucí do nekonečna). Problém vyřešili Bartal, Fiat, Karloff a Vohra, kteří sestrojili $\left(2 - \frac{1}{70} \approx 1.9857\right)$-kompetitivní on-line algoritmus pro libovolné m.

Dále odhad zdokonalili Karger, Phillips a Torng. Jejich algoritmus má menší (tedy lepší) kompetitivní poměr než Grahamův algoritmus pro $m \geq 6$, asymptoticky je algoritmus 1.945-kompetitivní.

Chen, van Vliet a Woeginger [5] dokázali další dolní odhad. Dokázali, že pro $m = 4$ počítače je každý deterministický on-line algoritmus alespoň 1.731-kompetitivní. Navíc sestrojili deterministický 1.7333-kompetitivní on-line algoritmus. Pro dostatečně velké m je každý deterministický on-line algoritmus alespoň 1.8319-kompetitivní. Výsledek zlepšili Bartal, Karloff a Rabani [3], kteří dokázali, že pro $m \geq 3454$ počítačů je každý deterministický on-line algoritmus alespoň 1.837-kompetitivní.

Albers [1] sestrojila 1.923-kompetitivní algoritmus pro $m \geq 2$ počítačů. Tento algoritmus má dobré asymptotické chování. Zároveň dokázala dolní odhad, že každý algoritmus je alespoň 1.852-kompetitivní.

1.2.4 Odhady pravděpodobnostních algoritmů

Ve srovnání s deterministickým případem je pravděpodobnostní případ mnohem méně prozkoumán.

Bartal, Fiat, Karloff a Vohra [2] pro $m = 2$ počítače sestrojili $\frac{4}{3}$-kompetitivní pravděpodobnostní on-line algoritmus a dokázali jeho optimalitu, neboť dosahuje dolní odhad. Pro $m = 3$ dokázali dolní odhad 1.4.

Chen, van Vliet a Woeginger [4], nezávisle Sgall [15], dokázali obecný dolní odhad. Pravděpodobnostní on-line algoritmus pro m počítačů má kompetitivní poměr alespoň

$$1 + \frac{1}{\left(\frac{m}{m-1}\right)^m - 1} \tag{1.13}$$

Pro m jdoucí k nekonečnu konverguje k $\frac{e}{e-1} \approx 1.5819$. Tento odhad dává stejný odhad pro $m = 2$, pro $m = 3$ počítače je dokázaný odhad 1.42105 lepší, než dokázali Bartal a kol.

Seiden [14] sestrojil algoritmy dávající lepší horní odhady pro $m = 3, \ldots, 7$. Jejich kompetitivní poměr je nižší (lepší) než nejlepší známý deterministický algoritmus. Dokonce pro $m = 3, 4, 5$ je horní odhad lepší než nejlepší dolní odhad pro deterministické algoritmy. Dokázané kompetitivní poměry jsou postupně 1.55870, 1.67516, 1.74113, 1.78809, 1.82081 pro $m = 3, 4, 5, 6, 7$.

Není znám žádný pravděpodobnostní on-line algoritmus, který by měl asymptoticky lepší kompetitivní poměr než nejlepší deterministický algoritmus, je tedy horním odhadem také pro pravděpodobnostní on-line algoritmy.

Nejdůležitější dosažené výsledky jsou shrnuty v tabulce 1.1.

Poznámka: *Přibližné hodnoty uvedené v tabulce jsou zaokrouhlené tak, aby dávaly příslušné odhady - dolní odhady dolů a horní odhady nahoru.*

1.2.5 Hlavní charakteristiky zkoumaných algoritmů

Smyslem této kapitoly není podrobné rozebírání toho, jak funguje ten či onen konkrétní algoritmus. Smyslem je jen nastínit, jak takové algoritmy vypadají a jaká mají omezení.

Důkazy horních odhadů kompetitivního poměru jsou koncipovány jako nalezení konkrétního algoritmu, o kterém se ukáže, jaký má kompetitivní poměr.

Počet počítačů	**Pravděpodobnostní**		**Deterministické**	
m	**Dolní odhad**	**Horní odhad**	**Dolní odhad**	**Horní odhad**
2	1.33333	1.33334	1.50000	1.50000
3	1.42105	1.55871	1.66666	1.66667
4	1.46285	1.67517	1.73101	1.73334
5	1.48738	1.74114	1.74625	1.77084
6	1.50352	1.78810	1.77301	1.80000
7	1.51496	1.82082	1.79103	1.82292
∞	1.58197	1.94500	1.83700	1.94500

Tabulka 1.1: Nejlepší známé odhady kompetitivních poměrů

Grahamův hladový algoritmus umísťuje příchozí úlohu vždy na počítač s minimální zátěží, proto je deterministický. Takovéto chování nedá dobrý výsledek v okamžiku, když přijde velká úloha. Vždy je proto potřeba mít připravený počítač s malou zátěží pro případ, že by přisla velká úloha.

Poznámka: *Velkou úlohou je myšlena úloha o velikosti přibližně jako je rozdíl maximální a minimální zátěže počítačů. To se netýká potenciálně nekonečných úloh, s jejich umístěním není problém. Umístění potenciálně nekonečné úlohy je stejné jako umístění jednotkové úlohy do prázdného rozvrhu.*

Často se algoritmy omezují jen na umísťování na několik vybraných počítačů v pořadí podle zátěží. Zřejmě jeden z nich bude minimální počítač (jinak by to byl problém pro menší m ztížený vyžadovaním menšího optima).

Důležitou charakteristikou deterministických i pravděpodobnostních on-line algoritmů je zachovávání nějakého invariantu. V deterministickém případě je používán invariant, aktuální konfigurace, příchozí úloha atp. pro určení počítače, na který bude úloha umístěna. V pravděpodobnostním případě se navíc používá průměrná konfigurace atp., jsou určeny pravděpodobnosti umístění příchozí úlohy na jednotlivé počítače a podle nich je náhodně vybrán počítač.

Seidenovy [14] algoritmy umísťují příchozí úlohy jen na minimální počítač a druhý minimální počítač. Seiden sestrojil dva algoritmy, které nazývá „Short Invariant“ a „Tall Invariant.“ První z nich používá invariant zaručující, že normalizovaná průměrná minimální zátěž nepřevýší konstantu závislou na m. Tím je algoritmus připraven na případnou velkou úlohu. Druhý z nich používá invariant, že maximální zátěž má být alespoň konstanta-násobek minimální. Za konstantu je volen kompetitivní poměr. Bylo očekáváno a z výsledků je patrné, že tyto algoritmy dobře fungují pro malý počet počítačů.

Jiným příkladem je algoritmus od Albers [1], tento umísťuje úlohy na minimální počítač a na prostřední počítač. Invariant, který má být splněn je, aby průměrná zátěž prostředního byla alespoň konstanta-násobek minimálního.

Zejména jsou konstruovány algoritmy, které umísťují úlohy na dva počítače. Hlavním důvodem je, že i pro dva je analýza kompetitivního poměru značně komplikovanou záležitostí.

1.3 Cíle a výsledky této práce

- Základním cílem práce je shrnout známé poznatky o pravděpodobnostních on-line algoritmech. K tomu je třeba vysvětlit a zavést potřebné pojmy. Součástí toho je i pokus o zavedení jednotného formalismu, který by mohl nahradit roztříštěné formalismy použité v literatuře.

- Důležitým cílem práce je prohloubení znalosti o problému rozvrhování. Teorii je třeba postupně budovat od základů a postupovat k věcem složitějším. Pro zkoumání byly zvoleny pravděpodobnostní on-line algoritmy, protože zadání problému rozvrhování je jednoduché, problém má praktické aplikace a existuje v něm i řada dlouho otevřených problémů.

- Dolní odhad, který dokázali Chen, van Vliet a Woeginger [4], nezávisle Sgall [15], se zdál býti optimální. Avšak objevily se důvody, proč se domnívat, že optimální není, důvody jsou založeny na intuici k důkazu citovaného odhadu. Cílem této práce bylo dokázat, že neexistuje pravděpodobnostní on-line algoritmus s kompetitivním poměrem rovným tomuto dolnímu odhadu. Tento cíl se bohužel nepodařilo zcela obecně dokázat. Zatím se podařilo tuto neoptimalitu dokázat pro $m = 3$ počítače. K tomuto důkazu je třeba vybudovat teorii dávající řadu zajímavých tvrzení o chování algoritmů na tzv. kritických posloupnostech.

- Dalším cílem je provést dolní odhad kompetitivního poměru pro třídu algoritmů umísťujících úlohy na dva počítače. To je třída algoritmů běžně konstruovaných v horních odhadech.

1.4 Komentáře

Zde jsou mé komentáře a názory založené na zkušenostech a intuici získané při psaní této práce. Proto nemusejí být nutně pravdivé, nicméně tyto komentáře mohou být chápány jako poznámky a náměty pro zkoumání.

- Na první pohled se může zdát, že deterministické on-line algoritmy pro rozvrhování jsou „diskriminovány“ definicí kompetitivního poměru. Diskriminace může být viděna v tom, že se po algoritmu chce, aby se choval „dobře“ již od počáteční prázdné konfigurace. Avšak těžkou posloupnost pro libovolnou konfiguraci lze získat asymptotickým zvětšením těžké posloupnosti na prázdné konfiguraci.

- U pravděpodobnostních on-line algoritmů je pochopitelně zkoumána průměrná délka rozvrhu. Ovšem s nenulovou pravděpodobností může být získaný rozvrh dost dlouhý. Zajímavé by mohlo být omezení maximální délky rozvrhu, například konstantou 2. Toto omezení by asi mělo vliv na odhady kompetitivních poměrů. Domnívám se však, že by tento vliv nebyl příliš velký.

- Dá se očekávat, že jeden z možných dalších směrů zdokonalování pravděpodobnostních on-line algoritmů může být i zvyšování počtu počítačů, na které algoritmus umísťuje úlohy. Asi to přinese lepší výsledky, ale nelze očekávat optimálnost těchto algoritmů. Pro konstrukci optimálních algoritmů je potřeba přijít s novým nápadem.

- V horních odhadech jsou konstruovány algoritmy umísťující příchozí úlohu na dva počítače (minimální a i-tý maximální počítač). Obdobně by však bylo možno umísťovat úlohy na minimální počítač a na počítač splňující nějakou podmínku (např. je po normalizaci maximální menší než konstanta). Pro velký (limitní) počet počítačů by tato úprava mohla zlepšit odhad, neboť může flexibilněji reagovat na příchozí úlohu.

- Na první pohled se může zdát, že dolní odhad přes všechny konfigurace může být počítán tak, že se odhad dokáže pro každou konfiguraci sestrojením protipříkladu. Takto jednoduše problém nelze řešit. Snadno lze sestrojit příklad, ve kterém se algoritmus může deterministicky rozhodnout mezi dvěma „špatnými“ možnostmi, zatímco nějaká jejich pravděpodobnostní kombinace může být v průměru „lepší.“

Kapitola 2

Základy

Tato kapitola sjednocuje značení, základní pojmy a definice o pravděpodobnostních on-line algoritmech, popisuje použitý formalismus. Je dokázáno i několik potřebných a zajímavých tvrzení.

2.1 Běžné symboly

- $\mathbb{N}$ označuje **přirozená čísla**, bez nuly, $1, 2, 3, \ldots$
- $\mathbb{N}_0$ označuje **přirozená čísla s nulou**, $0, 1, 2, 3, \ldots$
- $\mathbb{R}$ označuje **reálná čísla**
- $Pr_{x \in A}[I(x)]$ označuje pravděpodobnost, že jev I je splněn, počítáno přes distribuci A
- $\mathbb{E}_{x \in A}[f(x)]$ označuje střední hodnotu z funkce $f(x)$ přes distribuci A
- $[a|b]$ označuje tzv. blokové spojení matic či vektorů

2.2 Obecné definice

Definice 2.2.1 Pro $k, i \in \mathbb{N}$, $x \in \mathbb{N}$ *resp.* $\in \mathbb{R}$ *označme*

$[x]_i^k \stackrel{def}{=}$ *i-tý člen uspořádané k-tice x v pevně zvoleném kódovaní k-tic z/do přirozených resp. reálných čísel.*

Definice 2.2.2 Pro $m \in \mathbb{N}, m > 1$ *označme množinu*

$$[m] \stackrel{def}{=} \{i \mid (i \in \mathbb{N}) \wedge (i \leq m)\} \tag{2.1}$$

Tj. množina obsahující právě prvky $1, \ldots, m$.

Definice 2.2.3 Buď

$$\mathcal{R} \stackrel{def}{=} \langle 0, 1) \tag{2.2}$$

Tj. množina posloupností náhodných bitů, neboť ve dvojkové soustavě je reálné číslo nekonečnou posloupností hodnot 0 a 1.

Definice 2.2.4 $\mathcal{B} \in \mathcal{R}$ *nazveme* ***náhodná posloupnost****, pokud platí, že $\mathcal{B}$ je náhodné s uniformní distribucí pravděpodobnosti v $\mathcal{R}$.*

Taková posloupnost je nekonečnou posloupností náhodných bitů, tj. nezávislých náhodných $0/1$ veličin s pravděpodobností $\frac{1}{2}$ pro 0 resp. 1.

2.3 Posloupnosti úloh

Definice 2.3.1 *P nazveme* ***posloupnost úloh****, pokud platí, že $|P| \in \mathbb{N}_0$ je délka posloupnosti P a $P_i \in \mathbb{N}$, pro $i = 1, \ldots, |P|$, jsou prvky posloupnosti P.*

Definice 2.3.2 Buď

$$\mathcal{P} \stackrel{def}{=} \{P \mid P \textit{ je posloupnost úloh}\} \tag{2.3}$$

$\mathcal{P}$ je množina všech posloupností úloh.

Definice 2.3.3 Pro $P \in \mathcal{P}, i \in [|P|]$ *definujme*

$$P_{-i} \stackrel{def}{=} P_{|P|+1-i} \tag{2.4}$$

Tím jsou přehledněji označeny úlohy v pořadí od konce posloupnosti.

Definice 2.3.4 Pro $P, Q, R \in \mathcal{P}$, $j \in [|P|]$ *definujme posloupnost*

$$P^j \stackrel{def}{=} Q \tag{2.5}$$

kde

$$(|Q| = j) \wedge ((\forall i \in [j]) : \; Q_i = P_i) \tag{2.6}$$

Tj. prvních j členů posloupnosti P tvoří posloupnost P^j.

$$P^0 \stackrel{def}{=} Q \tag{2.7}$$

kde

$$|Q| = 0 \tag{2.8}$$

Tj. prázdná posloupnost.

$$P^{-j} \stackrel{def}{=} P^{|P|+1-j} \tag{2.9}$$

Tj. prvky posloupnosti P až po j-tý od konce včetně v nezměněném pořadí tvoří P^{-j}.

Definice je korektní, neboť je jednoznačná, důkaz zřejmý.

Tvrzení 2.3.5 Pro $P, Q, R \in \mathcal{P}$, $i, j \in [|P|]$, $Q = P^i$, $R = P^{-i}$

$$(i \geq j) \Longrightarrow (Q^j = P^j) \tag{2.10}$$

a pro záporné indexy platí

$$(i + j - 1) \in [|P|] \Longrightarrow (R^{-j} = P^{-(i+j-1)}) \tag{2.11}$$

Důkaz. Snadný, indukcí podle j s užitím definice 2.3.4. □

Definice 2.3.6 Pro $P \in \mathcal{P}$ *definujme*

$$\mathit{sum}(P) \stackrel{def}{=} \sum_{i \in [|P|]} P_i \tag{2.12}$$

$\mathit{sum}(P)$ je součet všech prvků v posloupnosti P.

Tvrzení 2.3.7 Pro $P \in \mathcal{P}$ *platí*

$$(|P| = 0) \Longrightarrow \mathit{sum}(P) = 0 \tag{2.13}$$

a současně

$$(|P| > 0) \Longrightarrow \mathit{sum}(P) = \mathit{sum}(P^{-2}) + P_{-1} \tag{2.14}$$

Důkaz. Zřejmý, indukcí. □

2.4 Pravděpodobnostní on-line algoritmus

Definice 2.4.1 Pro $\mathit{cub} : \mathbb{N} \times \mathbb{R} \times \mathcal{R} \longmapsto \mathbb{N} \times \mathcal{R}$

$\mathit{cub} : \mathbb{N} \times \mathbb{R} \times \mathcal{R} \longmapsto \mathbb{N} \times \mathcal{R}$ *nazveme **losující zobrazení**, pokud platí*

$$(\forall n \in \mathbb{N})(\forall p \in \mathbb{R})(\forall \mathcal{B} \in \mathcal{R}) : \tag{2.15}$$

$$(cub(n,p,\mathcal{B}) = X) \Longrightarrow ((\forall j \in [n]) :$$
$$Pr\left[[X]_1^2 = j\right] = [p]_j^n) \wedge ((\forall j \in [n]) :\ [X]_2^2 | [X]_1^2 = j$$
$$\textit{je uniformní jev v } \mathcal{R})$$

Losující zobrazení používá libovolné (tedy i iracionální) pravděpodobnosti, které lze efektivně (algoritmicky) vyčíslit.

Poznámka: *S pomocí losujícího zobrazení je možné sestrojit náhodný jev s libovolně předepsanými pravděpodobnostmi možných hodnot. Jen musí být splněna podmínka efektivní (algoritmické) vyčíslitelnosti těchto pravděpodobností. Časová složitost takové volby není omezena. Omezena je jen průměrná časová složitost.*

Tvrzení 2.4.2 $(\exists cub : \mathbb{N} \times \mathbb{R} \times \mathcal{R} \longmapsto \mathbb{N} \times \mathcal{R}) :$ *cub je losující zobrazení.*

Existuje losující zobrazení.

Důkaz. Algoritmická konstrukce takovéhoto zobrazení je popsána v literatuře, například viz. Nelson, Gailly [13]. Losující zobrazení má aplikace například v aritmetické kompresi dat. □

Definice 2.4.3 Pro $m \in \mathbb{N}, m > 1$

měřitelné zobrazení $\mathcal{A} : \mathcal{P} \times \mathcal{R} \times \mathbb{N} \longmapsto [m]$ *nazveme* ***algoritmus****, pokud platí*

$$(\forall \mathcal{B} \in \mathcal{R})(\forall P \in \mathcal{P})(\forall i \in [|P|])(\forall j \in [i]) :\ \mathcal{A}(P,\mathcal{B},j) = \mathcal{A}(P^i,\mathcal{B},j) \quad (2.16)$$

Slovem algoritmus se rozumí pravděpodobnostní on-line algoritmus pro rozvrhování na m počítačů. Výraz $\mathcal{A}(P,\mathcal{B},j)$ udává číslo počítače, na který byla umístěna j-tá zadaná úloha, tedy P_j, algoritmem $\mathcal{A}$ pracujícím s náhodnou posloupností $\mathcal{B}$ a vstupem P. V definici se požaduje, aby algoritmus rozhodl stejně (při pevné náhodné posloupnosti) pro všechny posloupnosti se stejným prefixem, což odpovídá tomu, že algoritmus je on-line.

Poznámka: *Podle tvrzení* 2.4.2 *má tato definice algoritmu smysl, každý pravděpodobnostní on-line algoritmus totiž lze realizovat s pomocí losujícího zobrazení.*

Definice 2.4.4 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $\mathcal{B} \in \mathcal{R}$ *definujme zobrazení* $cfg_{\mathcal{A}} : \mathcal{P} \times \mathcal{R} \longmapsto \mathbb{N}^m$ *tak, že*

$$(\forall P \in \mathcal{P})(\forall \mathcal{B} \in \mathcal{R})(\forall i \in [m]) : \ \left[cfg_{\mathcal{A}}(P, \mathcal{B})\right]_i^m \stackrel{def}{=} \sum_{(k \in [|P|]) \wedge (\mathcal{A}(P,\mathcal{B},k)=i)} P_k \tag{2.17}$$

Tím je definována konfigurace, tedy zátěže jednotlivých počítačů, do které se algoritmus dostane při dané posloupnosti úloh P a náhodné posloupnosti $\mathcal{B}$.

Definice 2.4.5 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$ *definujme množinu*

$$\mathcal{R}_{\mathcal{A}} \stackrel{def}{=} \left\{\mathcal{B} \in \mathcal{R} \mid (\forall P \in \mathcal{P}) : \ Pr_{\mathcal{B}_1 \in \mathcal{R}} \left[cfg_{\mathcal{A}}(P, \mathcal{B}_1) = cfg_{\mathcal{A}}(P, \mathcal{B})\right] > 0\right\} \tag{2.18}$$

Tím je definována množina náhodných posloupností, pro které se algoritmus nechová „špatně“ ve smyslu důkazu budoucího tvrzení 3.2.5, kde je nutné tuto podmnožinu použít. Ovšem tvrzení 2.4.8 ukáže, že tento požadavek není nijak omezující.

Definice 2.4.6 *Definujeme zobrazení* $OrdP : \mathcal{P} \longmapsto \mathbb{N}$, *že*

$$(\forall P \in \mathcal{P}) : \ OrdP(P) \stackrel{def}{=} 1 + \sum_{i=1}^{|P|} 2^{-1+\sum_{j=1}^{i} P_j} \tag{2.19}$$

Přičte se 1 za prázdnou posloupnost úloh, např. $38 = 1 + (100101)_2$ reprezentuje posloupnost $1, 2, 3$

Tvrzení 2.4.7 *Zobrazení OrdP je bijektivní.*

Důkaz. Zřejmý z definice 2.4.6. □

Tvrzení 2.4.8

$$Pr_{\mathcal{B}\in\mathcal{R}}\left[\mathcal{B}\in\mathcal{R}_{\mathcal{A}}\right]=1 \tag{2.20}$$

Tímto je ukázáno, že množina $\mathcal{R}_{\mathcal{A}}$ může být uvažována pro výběr náhodné posloupnosti, na které algoritmus bude pracovat namísto množiny $\mathcal{R}$, protože neporušuje uniformitu distribuce.

Důkaz. Definujme predikát $c^i_{\mathcal{A}}\left(\mathcal{B}_1,\mathcal{B}_2\right)$ takový, že algoritmus $\mathcal{A}$ pro náhodné posloupnosti $\mathcal{B}_1,\mathcal{B}_2$ na vstupní posloupnosti úloh $OrdP^{-1}(i)$ dává stejnou konfiguraci počítačů, takto

$$c^i_{\mathcal{A}}\left(\mathcal{B}_1,\mathcal{B}_2\right)\overset{def}{\Longleftrightarrow}\left(cfg_{\mathcal{A}}(OrdP^{-1}(i),\mathcal{B}_1)=cfg_{\mathcal{A}}(OrdP^{-1}(i),\mathcal{B}_2)\right) \tag{2.21}$$

Navíc z definice vyplývá, že tento predikát je relací ekvivalence s konečně mnoha rozkladovými třídami na množině $\mathcal{R}$.

Definujme množiny ${\mathcal{R}_{\mathcal{A}}}^i, i\in\mathbb{N}$ tak, aby platilo

$$\mathcal{R}_{\mathcal{A}}=\mathcal{R}-\bigcup_{i\in\mathbb{N}}{\mathcal{R}_{\mathcal{A}}}^i \tag{2.22}$$

takto

$${\mathcal{R}_{\mathcal{A}}}^i\overset{def}{=}\left\{\mathcal{B}\in\mathcal{R}\mid Pr_{\mathcal{B}_1\in\mathcal{R}}\left[c^i_{\mathcal{A}}\left(\mathcal{B},\mathcal{B}_1\right)\right]=0\right\} \tag{2.23}$$

vztah 2.22 zřejmě platí, přímo podle této definice 2.23 a definice 2.21 platí

$$Pr_{\mathcal{B}\in\mathcal{R}}\left[\mathcal{B}\in{\mathcal{R}_{\mathcal{A}}}^i\right]=0 \tag{2.24}$$

navíc v limitě platí

$$Pr_{\mathcal{B}\in\mathcal{R}}\left[\mathcal{B}\in\bigcup_{i\in\mathbb{N}}\mathcal{R_A}^i\right]=0 \tag{2.25}$$

odtud spolu se vztahem 2.22 platí dokazovaný vztah 2.20.

Poznámka: *Vztah* 2.20 *říká, že množina $\mathcal{R_A}$ má míru* 1 *v množině $\mathcal{R}$, míra množiny A v B, kde $A\subseteq B$, je totiž pravděpodobnost, že pro (uniformně) náhodné $x\in A$ platí $x\in B$.*

□

Tvrzení 2.4.9 Pro $m\in\mathbb{N}, m>1$, *algoritmus $\mathcal{A}$, $\mathcal{B}\in\mathcal{R}$, $P\in\mathcal{P}$ platí*

$$\sum_{i\in[m]}\left[cfg_{\mathcal{A}}(P,\mathcal{B})\right]_i^m=sum(P) \tag{2.26}$$

Součet zátěží všech počítačů je roven součtu velikostí všech úloh.

Důkaz. Z definice 2.4.4 se dosadí

$$\sum_{i\in[m]}\left[cfg_{\mathcal{A}}(P,\mathcal{B})\right]_i^m=\sum_{i\in[m]}\sum_{(k\in[|P|])\wedge(\mathcal{A}(P,\mathcal{B},k)=i)}P_k= \tag{2.27}$$

Prohodí se sumy, upraví se dle definice 2.4.3 a dále dle definice 2.3.6

$$=\sum_{k\in[|P|]}\sum_{(i\in[m])\wedge\mathcal{A}(P,\mathcal{B},k)=i}P_k=\sum_{k\in[|P|]}P_k=sum(P) \tag{2.28}$$

□

Tvrzení 2.4.10 Pro $m\in\mathbb{N}, m>1$, *algoritmus $\mathcal{A}$, $P\in\mathcal{P}$, $\mathcal{B}\in\mathcal{R}$, $i\in|P|$ platí*

$$(\forall j\in[m]):\ \left[cfg_{\mathcal{A}}(P,\mathcal{B})\right]_j^m=\left[cfg_{\mathcal{A}}(P^{-i},\mathcal{B})\right]_j^m+\sum_{k\in[i-1]\wedge\mathcal{A}(P,\mathcal{B},|P|+1-k)=j}P_{-k} \tag{2.29}$$

Toto je výhodnější tvar definice konfigurace, který ukazuje, jak se existující konfigurace rozšiřuje o další přidané úlohy.

Důkaz. Snadný z definic 2.4.4 a 2.4.3. □

Tvrzení 2.4.11 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $\mathcal{B} \in \mathcal{R}$, $i \in |P|$ *platí*

$$\left|\left\{j \in [m] \mid \left[cfg_{\mathcal{A}}(P, \mathcal{B})\right]_j^m \neq \left[cfg_{\mathcal{A}}(P^{-i}, \mathcal{B})\right]_j^m\right\}\right| \leq i - 1 \tag{2.30}$$

Deterministická konfigurace (tj. při pevné náhodné posloupnosti $\mathcal{B}$) pro posloupnost úloh a tutéž posloupnost zkrácenou o $i-1$ úloh se liší nejvýše na $i-1$ počítačích.

Důkaz. Indukcí přes i a použitím tvrzení 2.4.10. □

2.5 Zátěže, délka rozvrhu

Definice 2.5.1 Pro $m \in \mathbb{N}, m > 1, C \in \mathbb{N}_0^m$ *definujme bijektivní zobrazení* $ord_C : [m] \longmapsto [m]$ *tak, že*

$$(\forall i, j \in [m]) : \tag{2.31}$$

$$(i < j) \Longleftrightarrow ([C]^m_{ord_C(i)} < [C]^m_{ord_C(j)}) \vee (([C]^m_{ord_C(i)} = [C]^m_{ord_C(j)}) \wedge (i < j))$$

Takovéto zobrazení seřadí indexy sloupečků dané m-tice podle velikosti těchto sloupečků. Pro korektnost je třeba existence a jednoznačnost. Aby definice byla jednoznačná, tak jsou hodnoty primárně řazeny podle velikosti a sekundárně podle jejich indexu v m-tici. Takové zobrazení existuje, neboť jej lze realizovat libovolným třídícím algoritmem.

Poznámka: *m-tice je například deterministická konfigurace, pak $ord_C(1), \ldots, ord_C(m)$ jsou indexy od minimálního do maximálního sloupečku. Samotné ord_C může být chápáno jako m-tice těchto hodnot.*

Definice 2.5.2 Pro $m \in \mathbb{N}, m > 1$ *definujme zobrazení Cord* : $\mathbb{N}_0^m \longmapsto \mathbb{N}_0^m$ *tak, že*

$$(\forall C \in \mathbb{N}_0^m): \ Cord(C) \stackrel{def}{=} X \tag{2.32}$$

kde

$$[X]_i^m = [C]_{ord_C(i)}^m$$

Zobrazení *Cord* sloupečky dané m-tice (např. konfigurace) seřadí podle jejich velikosti.

Definice 2.5.3 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $\mathcal{B} \in \mathcal{R}$, $P \in \mathcal{P}$ *definujme zobrazení* $Load_{(\mathcal{A},P,\mathcal{B})} : [m] \longmapsto \mathbb{N}_0$ *tak, že*

$$(\forall i \in [m]): \ Load_{(\mathcal{A},P,\mathcal{B})}(i) \stackrel{def}{=} \left[Cord(cfg_{\mathcal{A}}(P,\mathcal{B}))\right]_i^m \tag{2.33}$$

Toto zobrazení seřadí počítače podle jejich zátěže, tedy minimální resp. maximální zátěž je $Load_{(\mathcal{A},P,\mathcal{B})}(1)$ resp. $Load_{(\mathcal{A},P,\mathcal{B})}(m)$.
Poznámka: *Pro přehlednost se používají i záporné indexy počítačů. Pro* $i \in [m]$ *je ztotožněno* $-i \equiv m + 1 - i$.
Poznámka: *Někdy se zjednodušeně mluví o „velikosti počítače," o „minimálním počítači" či o „maximálním počítači" místo o velikosti zátěže počítače, o počítači s minimální zátěží či o počítači s maximální zátěží.*

Definice 2.5.4 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$ *definujme zobrazení* $\mathbb{E}load_{(\mathcal{A},P)} : [m] \longmapsto \mathbb{R}$ *tak, že*

$$(\forall i \in [m]): \ \mathbb{E}load_{(\mathcal{A},P)}(i) \stackrel{def}{=} \mathbb{E}_{\mathcal{B}\in\mathcal{R}}\left[Load_{(\mathcal{A},P,\mathcal{B})}(i)\right] \tag{2.34}$$

Zobrazení počítá průměrné zatěže počítačů v pořadí dle jejich zatěží. Například $\mathbb{E}load_{(\mathcal{A},P)}(m)$ je průměr z maximálně zatížených počítačů v jednotlivých konfiguracích.

Poznámka: *Je třeba si uvědomit, že se zátěže počítačů v jednotlivých konfiguracích napřed seřadí a až následně se zprůměrují.*
Poznámka: *Pro přehlednost se používají i záporné indexy počítačů. Pro $i \in [m]$ je ztotožněno $-i \equiv m + 1 - i$.*

Definice 2.5.5 Pro $m \in \mathbb{N}, m > 1$, *algoritmus $\mathcal{A}$, $\mathcal{B} \in \mathcal{R}$ definujme zobrazení $makespan_{\mathcal{A}} : \mathcal{P} \times \mathcal{R} \longmapsto \mathbb{N}_0$ tak, že*

$$makespan_{\mathcal{A}}(P, \mathcal{B}) \stackrel{def}{=} \max_{i \in [m]} \left[cfg_{\mathcal{A}}(P, \mathcal{B}) \right]_i^m \tag{2.35}$$

Tj. délka rozvrhu vytvořená algoritmem $\mathcal{A}$ nad posloupností úloh P a při náhodné posloupnosti $\mathcal{B}$, tedy je to velikost maximálního sloupečku získané konfigurace.

Tvrzení 2.5.6 Pro $m \in \mathbb{N}, m > 1$, *algoritmus $\mathcal{A}$, $\mathcal{B} \in \mathcal{R}$, $P \in \mathcal{P}$ platí:*

$$makespan_{\mathcal{A}}(P, \mathcal{B}) = Load_{(\mathcal{A}, P, \mathcal{B})}(m) \tag{2.36}$$

Důkaz. Zřejmý z definic 2.5.3, 2.5.2 a 2.5.1. □

Proto lze místo definice $makespan_{\mathcal{A}}$ uvažovat následující obecnější verzi:

Definice 2.5.7 Pro $m \in \mathbb{N}, m > 1$, $C \in \mathbb{N}_0^m$ *definujme zobrazení makespan : $\mathbb{N}_0^m \longmapsto \mathbb{N}_0$ tak, že*

$$makespan(C) \stackrel{def}{=} \max_{i \in [m]} [C]_i^m \tag{2.37}$$

Speciálně platí rovnost $makespan(cfg_{\mathcal{A}}(P, \mathcal{B})) = makespan_{\mathcal{A}}(P, \mathcal{B})$.

Definice 2.5.8 Pro $m \in \mathbb{N}, m > 1$, *algoritmus $\mathcal{A}$, $P \in \mathcal{P}$ definujme zobrazení $\mathbb{E}makespan_{\mathcal{A}} : \mathcal{P} \longmapsto \mathbb{R}$ tak, že*

$$\mathbb{E}makespan_{\mathcal{A}}(P) \stackrel{def}{=} \mathbb{E}_{\mathcal{B} \in \mathcal{R}} \left[makespan_{\mathcal{A}}(P, \mathcal{B}) \right] \tag{2.38}$$

Průměrná délka algoritmem vytvořeného rozvrhu nad danou posloupností průměrovaná přes náhodné posloupnosti bitů, tedy tj. velikost maximálního sloupečku v průměrné konfiguraci.

2.6 Rozvrhovače

Definice 2.6.1 Pro $m \in \mathbb{N}, m > 1$ *definujme zobrazení* *sched* $: \{z \mid z : \mathbb{N} \longmapsto [m]\} \times \mathcal{P} \longmapsto \mathbb{N}_0^m$ *tak, že*

$$(\forall z : \mathbb{N} \longmapsto [m])(\forall P \in \mathcal{P}) : \ \mathit{sched}(z, P) \stackrel{def}{=} X \tag{2.39}$$

kde

$$[X]_i^m = \sum_{j : j \in [|P|] \wedge z(j) = i} P_j \tag{2.40}$$

Výsledkem zobrazení *sched* je konfigurace, do které se lze dostat umístěním i-té úlohy P_i na počítač $z(i)$ pro všechna i.

Tvrzení 2.6.2 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}, \mathcal{B} \in \mathcal{R}$, $P \in \mathcal{P}$, $z : \mathbb{N} \longmapsto [m]$, *splňující* $(\forall i \in [|P|]) : \ z(i) = \mathcal{A}(P, \mathcal{B}, i)$ *platí*

$$\mathit{sched}(z, P) = \mathit{cfg}_{\mathcal{A}}(P, \mathcal{B}) \tag{2.41}$$

Důkaz. Zřejmý. □

Definice 2.6.3 Pro $m \in \mathbb{N}, m > 1$, $P \in \mathcal{P}$, $z : \mathbb{N} \longmapsto [m]$ *zobrazení* z *nazveme* ***optimální rozvrhovač***, *pokud platí*

$$(\forall s : \mathbb{N} \longmapsto [m]) : \ \mathit{makespan}(\mathit{sched}(z, P)) \leq \mathit{makespan}(\mathit{sched}(s, P)) \tag{2.42}$$

Pojmem **optimální rozvrhovač** je nazváno každé zobrazení určující způsob, jak pro zadanou posloupnost vytvořit rozvrh o minimální délce.

Definice 2.6.4 Pro $m \in \mathbb{N}, m > 1$, $P \in \mathcal{P}$ *definujme offline*$_P : \mathbb{N} \longmapsto [m]$ *tak, že offline*$_P$ *je libovolný pevný optimální rozvrhovač.*

Existenci zobrazení a korektnost definice lze snadno ověřit podle definice 2.6.3.

Definice 2.6.5 Pro $m \in \mathbb{N}, m > 1$ *definujme zobrazení OPT* $: \mathcal{P} \longmapsto \mathbb{N}_0$ *tak, že*

$$(\forall P \in \mathcal{P}) : \; OPT(P) \stackrel{def}{=} makespan(sched(offline_P, P)) \tag{2.43}$$

Tím se definovala optimální, tj. minimální, délka rozvrhu, který lze sestrojit nad posloupností úloh P. Definice je korektní, neboť výsledek je invariantní vůči libovolné volbě optimálního rozvrhovače *offline*$_P$.

2.7 Kritická úloha, speciální posloupnosti

V této kapitolce jsou uvedeny základní definice a tvrzení se vztahem k řešení problému rozvrhování prezentovanému v této práci. Zejména jsou zde uvedeny definice kritické úlohy, kritické, slabě kritické a podkritické posloupnosti.

Definice 2.7.1 Pro $m \in \mathbb{N}, m > 1$ $P \in \mathcal{P}$ *nazveme* ***nasycená posloupnost****, pokud platí*

$$OPT(P) = \left\lceil \frac{sum(P)}{m} \right\rceil \tag{2.44}$$

Zátěže počítačů jsou „téměř“ stejné. K jejich vyrovnání stačí méně než m jednotkových úloh.

Poznámka: *Definice má smysl, neboť existují posloupnosti, pro které nastává uvedená rovnost, jsou to například kritické posloupnosti (viz. pozdější definice* 2.7.6*), bude ukázáno v důkazu tvrzení* 2.8.2.

Definice 2.7.2 Pro $m \in \mathbb{N}, m > 1$ $P \in \mathcal{P}$ *nazveme* ***úplně nasycená posloupnost****, pokud platí*

$$OPT(P) = \frac{sum(P)}{m} \tag{2.45}$$

Zátěže počítačů jsou stejné.

Poznámka: *Definice má smysl, neboť existují posloupnosti, pro které nastává uvedená rovnost, jsou to například kritické posloupnosti (viz. pozdější definice* 2.7.6*), bude ukázáno v důkazu tvrzení* 2.8.2.

Definice 2.7.3 Pro $m \in \mathbb{N}, m > 1$ *definujme predikáty, že pro* $P \in \mathcal{P}$, $|P| > 0$ *a* $i \in [|P|]$ *platí*

$$\mathcal{K}_K(P, i) \stackrel{def}{\Longleftrightarrow} P_i = \frac{sum(P^i)}{m} \tag{2.46}$$

$$\mathcal{K}_N(P, i) \stackrel{def}{\Longleftrightarrow} P_i \leq \frac{sum(P^i)}{m-1} \tag{2.47}$$

$$\mathcal{K}_P(P, i) \stackrel{def}{\Longleftrightarrow} P_i \leq \frac{sum(P^i)}{m} \tag{2.48}$$

Predikáty určují vlastnost úlohy P_i v posloupnosti. Predikát $\mathcal{K}_K(P, i)$ určuje, zda úloha je přesně kritická, predikát $\mathcal{K}_N(P, i)$, zda úloha je nejvýše „nadkritická“ a predikát $\mathcal{K}_P(P, i)$, zda úloha je podkritická (nejvýše kritická).

Podstatné pro definici kritické úlohy je, že její velikost je rovna délce úplně nasyceného rozvrhu pro tuto úlohu a jí předcházející úlohy.

Poznámka: *Tyto predikáty jsou vlastnostmi úlohy* P_i*, neboť výsledek predikátu se nemění při libovolném prodlužování posloupnosti, protože je funkcí jen posloupnosti* P^i.

Poznámka: *Pojem kritické úlohy byl převzat z výchozího článku viz. Sgall [15]. Pojem „nadkritické“ úlohy je nově definován a odpovídá pojmu kritické úlohy pro počet počítačů snížený o* 1.

Poznámka: *Pro přehlednost se používají i záporné indexy v posloupnosti. Pro $i \in [|P|]$ je ztotožněno $-i \equiv |P| + 1 - i$. Toto ztotožnění je konzistentní s definicí* 2.3.3.

Definice 2.7.4 Pro $m \in \mathbb{N}, m > 1$ *definujme symbol $\mathcal{K}$ tak, že*

$$\mathcal{K}(P) \stackrel{def}{=} \frac{sum(P)}{m-1} \tag{2.49}$$

Tímto je definována velikost tzv. kritické úlohy pro posloupnost úloh P.

Poznámka: *Zda je definice konzistentní s definicí* 2.7.3, *tj. zda platí*

$$(\forall P \in \mathcal{P}, |P| > 1) : \ \mathcal{K}(P^{-2}) = P_{-1} \Longleftrightarrow \mathcal{K}_K(P, -1) \tag{2.50}$$

lze snadno ověřit.

Definice 2.7.5 Pro $m \in \mathbb{N}, m > 1$ *definujme predikát pro $P \in \mathcal{P}$ a $n \in \mathbb{N}_0$ takto*

$$Crit(P, n) \stackrel{def}{\Longleftrightarrow} |P| \geq n \wedge (\forall i \in [n]) : \ \mathcal{K}_K(P, -i) \tag{2.51}$$

Predikát rozhoduje, zda posledních n úloh posloupnosti P je kritických.

Definice 2.7.6 Pro $m \in \mathbb{N}, m > 1$ $P \in \mathcal{P}$ *nazveme **kritická posloupnost**, pokud platí*

$$\begin{gathered} Crit(P, m) \wedge ((\exists h \in [|P|]) : \\ (\forall i \in [h]) : \ P_i = 1 \wedge \\ \wedge (\forall i \in [|P|], i > h) : \ \mathcal{K}_N(P, i)) \end{gathered} \tag{2.52}$$

Kritická posloupnost je tvořena počátečním úsekem jednotkových úloh následovaným „nadkritickými" úlohami (dle definice 2.7.3), navíc posledních m úloh je kritických.

Definice 2.7.7 Pro $m \in \mathbb{N}, m > 1$ *definujme množinu*

$$\mathcal{P}_K \stackrel{def}{=} \{P \in \mathcal{P} \mid P \textit{ je kritická posloupnost}\} \tag{2.53}$$

Množina kritických posloupností.

Definice 2.7.8 Pro $m \in \mathbb{N}, m > 1$ *definujme zobrazení Uni* : $\mathcal{P} \longmapsto \mathbb{N}_0$ *tak, že*

$$(\forall P \in \mathcal{P}): \ \mathit{Uni}(P) \stackrel{def}{=} \max\left(\{i \in [|P|] \mid (\forall j \in [i]): \ P_j = 1\} \cup \{0\}\right) \tag{2.54}$$

Zobrazení vrací délku maximálního počátečního úseku dané posloupnosti tvořeného jen úlohami o velikosti 1.

Tvrzení 2.7.9 Pro $m \in \mathbb{N}, m > 1$, $n \in \mathbb{N}_0, n \geq m$, $P \in \mathcal{P}_K$, $Crit(P, n)$

$$(\forall i \in [n - m + 1]): \ P^{-i} \in \mathcal{P}_K \tag{2.55}$$

speciálně

$$P^{-n+m-1} \in \mathcal{P}_K \tag{2.56}$$

Má-li kritická posloupnost na konci „dost" kritických úloh, tak jejím zkrácením se získá opět kritická posloupnost.

Důkaz. Snadný, podle definice 2.7.5 a volby $i \leq n - m + 1$ plyne, že $Crit(P^{-i}, m)$, což spolu s předpokladem, že P je kritická, podle definice 2.7.6 dává, že i posloupnost P^{-i} je kritická. Formule 2.56 je triviální důsledek formule 2.55 - speciální volbou $i = n - m + 1$. □

Tvrzení 2.7.10 Pro $m \in \mathbb{N}, m > 1$, $n \in [|P|]$, $P \in \mathcal{P}$, $Crit(P, n)$ *platí*

$$(\forall i \in [n]): \ P_{-i} = \left(\frac{m}{m-1}\right)^{n-i} \cdot \frac{\mathit{sum}(P^{-n})}{m} = \left(\frac{m}{m-1}\right)^{n-i} \cdot \frac{\mathit{sum}(P^{-n-1})}{m-1} \tag{2.57}$$

Určuje velikost kritických úloh v posloupnosti v tvaru, který bude dále výhodný pro počítání.

Důkaz. Snadný indukcí podle i pro pevné n s pomocí definice 2.7.3. □

Tvrzení 2.7.11 Pro $m \in \mathbb{N}, m > 1$, $P \in \mathcal{P}_K$, $i \in [m]$ *platí*

$$P_{-i} = \left(\frac{m}{m-1}\right)^{m-i} \cdot \frac{sum(P^{-m-1})}{m-1} \tag{2.58}$$

Důkaz. Plyne z tvrzení 2.7.10 ověřením předpokladů podle definice 2.7.4 a dosazením. □

Tvrzení 2.7.12 Pro $m \in \mathbb{N}, m > 1$, $n \in \mathbb{N}$

$$(\exists P \in \mathcal{P}_K)(\forall i \in [n]) : \ |P| \geq n \wedge P^{-i} \in \mathcal{P}_K \tag{2.59}$$

Existuje dostatečně dlouhá kritická posloupnost, že i po odebrání nejvýše $n-1$ posledních úloh zůstane kritická, tedy existuje kritická posloupnost taková, že i po přidání nejvýše $n-1$ kritických úloh zůstane kritická.

Důkaz. Důkaz je proveden konstrukcí takové kritické posloupnosti. Taková posloupnost splňuje

$$(\forall i \in [n+m-1]) : \ \mathcal{K}_K(P, -i) \tag{2.60}$$

Hodnota P_{-i} musí být celočíselná. Nutně podle tvrzení 2.7.10 musí být $sum(P^{-n-m})$ dělitelné číslem $(m-1)^{n+m-i}$. Proto $sum(P^{-n-m})$ má být dělitelné číslem $(m-1)^{n+m-1}$.

Ať posloupnost má prefix tvořený $(m-1)^{n+m-1}$ jednotkovými úlohami, tedy

$$\left(\forall i \in \left[(m-1)^{n+m-1}\right]\right) : \ P_i = 1 \tag{2.61}$$

Ať následuje $n+m-1$ kritických úloh. Podle definice 2.7.6 má platit

$$(\forall i \in [n+m-1]) : \ P_{(m-1)^{n+m-1}+i} = m^{i-1} \cdot (m-1)^{n+m-1-i} \tag{2.62}$$

Požadavky 2.61 a 2.62 je vlastně definice hledané kritické posloupnosti délky $|P| = (m-1)^{n+m-1} + n + m - 1$. Požadované vlastnosti plynou z definice 2.7.6. □

Definice 2.7.13 Pro $m \in \mathbb{N}, m > 1$ $P \in \mathcal{P}$ *nazveme **slabá kritická posloupnost**, pokud platí*

$$Crit(P, 1) \wedge ((\exists h \in [|P|]) : \tag{2.63}$$

$$(\forall i \in [h]) : \; P_i = 1 \wedge$$

$$\wedge (\forall i \in [|P|], i > h) : \; \mathcal{K}_N(P, i))$$

Slabá kritická posloupnost je tvořena počátečním úsekem jednotkových úloh následovaným „nadkritickými" úlohami (dle definice 2.7.3), navíc poslední úloha je kritická.

Definice 2.7.14 Pro $m \in \mathbb{N}, m > 1$ *definujme množinu*

$$\mathcal{P}_S \stackrel{def}{=} \{P \in \mathcal{P} \mid P \textit{ je slabá kritická posloupnost}\} \tag{2.64}$$

Množina slabých kritických posloupností.

Tvrzení 2.7.15 Pro $m \in \mathbb{N}, m > 1$ *platí*

$$\mathcal{P}_K \subseteq \mathcal{P}_S \tag{2.65}$$

Důkaz. Zřejmý z definic 2.7.6 a 2.7.13. □

Definice 2.7.16 Pro $m \in \mathbb{N}, m > 1$ $P \in \mathcal{P}$ *nazveme **podkritická posloupnost**, pokud platí*

$$(\exists h \in [|P|]) : \tag{2.66}$$

$$(\forall i \in [h]) : \; P_i = 1 \wedge$$

$$\wedge\,(\forall i \in [|P|], i > h) :\ \mathcal{K}_P(P, i)$$

Podkritická posloupnost je tvořena úsekem jednotkových úloh následovaným podkritickými úlohami.

Definice 2.7.17 Pro $m \in \mathbb{N}, m > 1$ *definujme množinu*

$$\mathcal{P}_P \overset{def}{=} \{P \in \mathcal{P} \mid P \textit{ je podkritická posloupnost}\} \tag{2.67}$$

To je množina podkritických posloupností.

Tvrzení 2.7.18 Pro $m \in \mathbb{N}, m > 1$, $X = \mathcal{P}_S$ *je množina slabě kritických posloupností na* m *počítačích,* $Y = \mathcal{P}_P$ *je množina podkritických posloupností na* $m - 1$ *počítačích platí*

$$X \subseteq Y \tag{2.68}$$

Každá slabá kritická posloupnost na m počítačích je podkritickou posloupností na $m - 1$ počítačích.

Důkaz. Ověření podle definice 2.7.13. □

2.8 Optimální rozvrh, kompetitivní poměr

Některá z následujících tvrzení jsou zobecněnými variantami k tvrzením v článku Sgall [15], ve kterém lze rovněž nalézt jejich původní důkazy. Ovšem pro účely této práce jsou tvrzení přeformulovaná (a zobecněná) a tudíž jsou zde uvedeny i jejich důkazy.

Tvrzení 2.8.1 Pro $m \in \mathbb{N}, m > 1$, $P \in \mathcal{P}_K$, $j \in [m]$ *platí*

$$sum(P^{-j}) = m \cdot P_{-j} \tag{2.69}$$

Důkaz. Přímo z definic 2.7.6 a 2.7.4. □

Tvrzení 2.8.2 Pro $m \in \mathbb{N}, m > 1$, $P \in \mathcal{P}_S \cup \mathcal{P}_P$ *platí*

$$OPT(P) = \left\lceil \frac{sum(P)}{m} \right\rceil \tag{2.70}$$

speciálně pro $P \in \mathcal{P}_S$ *platí*

$$OPT(P) = P_{-1} \tag{2.71}$$

Posloupnost P je nasycená, proto délka optimálního (míněno off-line) rozvrhu pro podkritickou odpovídá rovnoměrnému rozdělení úloh mezi všechny počítače, což je průměrná zátěž počítačů. Speciálně (pro kritické) slabé kritické posloupnosti to je délka její poslední (tudíž kritické) úlohy.
Poznámka: *Platí i pro kritické posloupnosti, neboť* $\mathcal{P}_K \subseteq \mathcal{P}_S$.

Důkaz. Důsledek 2.71 plyne z formule 2.70 podle definice 2.7.4, díky které zmizí i zaokrouhlování, neboť se dělí beze zbytku.

Dále případ pro $P \in \mathcal{P}_S$ se převede na případ $P \in \mathcal{P}_P$ pro $m-1$ počítačů takto: podle tvrzení 2.7.18 je P^{-1} podkritickou na $m-1$ počítačích, tedy i P^{-2} je podkritická na $m-1$ počítačích, tudíž podle formule 2.70 je dána délka optimálního rozvrhu **na** $m-1$ **počítačích** takto

$$OPT(P^{-2}) = \left\lceil \frac{sum(P^{-2})}{m-1} \right\rceil \tag{2.72}$$

Podle definice 2.7.4 a předpokladu, že posloupnost je kritická nebo slabá kritická, odpadá zaokrouhlování, neboť vnitřek je právě velikost poslední kritické úlohy, ta bude umístěna na jeden počítač sama, zbylé na $m-1$ zbylých počítačů, což je rozvrh požadované délky.

Dále bude dokázána formule 2.70 pro $m \in \mathbb{N}$ a $P \in \mathcal{P}_P$, čímž bude důkaz hotov. Důkaz je proveden konstrukcí rozvrhu hladovým algoritmem přes podkritické úlohy v posloupnosti, to jsou úlohy za prefixem z jednotkových úloh. Zbývající jednotkové úlohy již lze triviálně umístit a tím dorovnat na úplně nasycený rozvrh.

Konstrukce probíhá přes $j = |P|, \ldots, Uni(P)+1$. Při konstrukci off-line rozvrhu se vždy úloha P_j umístí na libovolný počítač takový, že délka zatím vytvořeného rozvrhu nepřevýší požadovanou délku $\left\lceil \frac{sum(P)}{m} \right\rceil$.

Nadefinujme posloupnost $Q \in \mathcal{P}$ tak, že

$$|Q| \stackrel{def}{=} |P| \wedge (\forall i \in [|P|]) : \; Q_i \stackrel{def}{=} P_{-i} \tag{2.73}$$

což je překlopená posloupnost P, tj. její úlohy jsou umístěny právě v opačném pořadí.

Označme $z^{\{j\}}$ rozvrhovač pro posloupnost Q^j. Pro $j = |P| + 1$ se definice rozšíří, posloupnost je prázdná, proto rozvrhovač je triviální. Bude dokázána platnost invariantu

$$(\exists i \in [m]) : \; \left[sched(z^{\{j+1\}}, Q^{j+1}) \right]_i^m + P_j \leq \left\lceil \frac{sum(P)}{m} \right\rceil \tag{2.74}$$

Zobrazení $z^{\{Uni(P)\}}$ definuje rozmístění všech úloh mimo počáteční jednotkový prefix posloupnosti P. Zbývá již jen rozmístit $Uni(P)$ jednotkových úloh, vzhledem k tomu, že bude dokázán výše uvedený invariant, tedy že délka rozvrhu z ostatních úloh bude nejvýše $\left\lceil \frac{sum(P)}{m} \right\rceil$ a tomu, že do rozvrhu o požadované délce lze umístit úlohy až o součtu $\left\lceil \frac{sum(P)}{m} \right\rceil \cdot m \geq sum(P)$, tak lze dále pokračovat v umísťování i těch jednotkových se zachováním téhož invariantu se získáním nasyceného rozvrhu. Tudíž $z^{\{1\}}$ je optimální rozvrhovač pro $Q^1 = Q$, ovšem tato posloupnost je překlopená P, tedy optimální rozvrhovač pro P se definuje jako

$$offline_P(i) \stackrel{def}{=} z^{\{1\}}(|P| + 1 - i) \tag{2.75}$$

Nyní se dokáže výše uvedený invariant sporem, předpokládá se pro spor, že

$$(\exists j \in [|P|])(\forall i \in [m]): \ \left[\mathit{sched}(z^{\{j+1\}}, Q^{\{j+1\}})\right]_i^m + P_j > \left\lceil \frac{\mathit{sum}(P)}{m} \right\rceil \tag{2.76}$$

Sečtením těchto nerovností pro nalezené j plyne nerovnost

$$\sum_{i \in [m]} \left[\mathit{sched}(z^{\{j+1\}}, Q^{\{j+1\}})\right]_i^m + m \cdot P_j > m \cdot \left\lceil \frac{\mathit{sum}(P)}{m} \right\rceil \tag{2.77}$$

Z definice $Q^{\{j+1\}}$ a definice 2.6.1 plyne

$$\sum_{k \in [|P|], k > j} P_k + m \cdot P_j > m \cdot \left\lceil \frac{\mathit{sum}(P)}{m} \right\rceil \tag{2.78}$$

Dle nerovnosti $m \cdot P_j \geq \mathit{sum}(P^j)$ z definic 2.7.3 a 2.7.13

$$\sum_{k \in [|P|], k > j} P_k + \mathit{sum}(P^j) > m \cdot \left\lceil \frac{\mathit{sum}(P)}{m} \right\rceil \tag{2.79}$$

Úpravou dle definice 2.3.6 a dolním odhadem zaokroulování dostaneme nerovnost, která je hledaným sporem

$$\mathit{sum}(P) > m \cdot \left\lceil \frac{\mathit{sum}(P)}{m} \right\rceil \geq \mathit{sum}(P) \tag{2.80}$$

Sestrojené zobrazení $\mathit{offline}_P$ dává nasycený rozvrh, viz. definice 2.7.1, tedy platí

$$\mathit{OPT}(P) = \left\lceil \frac{\mathit{sum}(P)}{m} \right\rceil \tag{2.81}$$

□

Definice 2.8.3 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $c \in \mathbb{R}$

*$\mathcal{A}$ nazveme **c-kompetitivní algoritmus**, pokud platí*

$$(\forall P \in \mathcal{P}): \ \mathbb{E}\mathit{makespan}_{\mathcal{A}}(P) \leq c \cdot \mathit{OPT}(P) \tag{2.82}$$

Takto nazveme algoritmus, který v průměru dává rozvrhy o délce nejvýše c krát větší, než je optimální délka, pro každou posloupnost úloh P.

Následují dvě pomocná tvrzení, jež jsou jen přeformulovaná původní ve smyslu zde uvedených definic.

Tvrzení 2.8.4 Pro $m \in \mathbb{N}, m > 1$, $c \in \mathbb{R}$, *c-kompetitivní algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $|P| \geq m$ *platí*

$$\frac{\sum_{i\in[m]} \mathbb{E}makespan_{\mathcal{A}}(P^{-i})}{\sum_{i\in[m]} OPT(P^{-i})} \leq \frac{\sum_{i\in[m]} c \cdot OPT(P^{-i})}{\sum_{i\in[m]} OPT(P^{-i})} = c \tag{2.83}$$

Důkaz. Přímo z definice 2.8.3 tj. definice c-kompetitivnosti. □

Poznámka: *Předpoklad o posloupnosti* P *splňuje každá kritická posloupnost, je-li* $P \leq m$, *pak optimální rozvrh má každou úlohu na jiném počítači. Zřejmě takový předpoklad není jakkoli omezující.*

Tvrzení 2.8.5 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $\mathcal{B} \in \mathcal{R}$ *platí*

$$sum(P) = \sum_{i\in[m]} Load_{(\mathcal{A},P,\mathcal{B})}(i) \tag{2.84}$$

Součet všech úloh v posloupnosti je roven součtu zátěží všech počítačů po jejich umístění algoritmem (pomocné technické tvrzení). Tvrzení platí i v průměru přes $\mathcal{B} \in \mathcal{R}$.

Důkaz. Důkaz snadný přímo z definic 2.3.6 a 2.5.3. □

Tvrzení 2.8.6 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $P^{-2} \in \mathcal{P}$, $\mathcal{B} \in \mathcal{R}$, $i \in [m-1]$ *platí*

$$Load_{(\mathcal{A},P,\mathcal{B})}(i) \leq Load_{(\mathcal{A},P^{-2},\mathcal{B})}(i+1) \tag{2.85}$$

Přidáním jedné úlohy zůstane pořadí počítačů téměř stejné, tj. buď počítač zůstal na stejném místě nebo se klesnul o jeden, nebyla-li na něj umístěna úloha, jinak nerovnost splní ten, který přišel na jeho místo.

Důkaz. Buď $j \in [m]$ počítač, na který byla umístěna úloha P_{-1}, podle tvrzení 2.4.10. Označme dále umístění tohoto počítače ve staré a nové konfiguraci.

$$a \stackrel{def}{=} ord_{cfg_{\mathcal{A}}(P^{-2},\mathcal{B})}(j) \tag{2.86}$$

a

$$b \stackrel{def}{=} ord_{cfg_{\mathcal{A}}(P,\mathcal{B})}(j) \tag{2.87}$$

Zátěž počítače j přidáním úlohy vzrostla a počítače jsou řazeny vzestupně, proto platí $a \leq b$, pořadí ostatních počítačů se podle definice 2.5.1 nemůže změnit. Proto počítače, které byly po sežazení před a nebo za b zůstanou, počítače mezi a a b se posunou, poklesnou o jeden, o vypuštěný j-tý počítač na místě a, který se objeví na místě b.

$$\begin{aligned}
(\forall i \in [m], i < a \vee i > b): \; ord_{cfg_{\mathcal{A}}(P,\mathcal{B})}{}^{-1}(i) &= ord_{cfg_{\mathcal{A}}(P^{-2},\mathcal{B})}{}^{-1}(i) \\
(\forall i \in [m], a \leq i < b): \; ord_{cfg_{\mathcal{A}}(P,\mathcal{B})}{}^{-1}(i) &= ord_{cfg_{\mathcal{A}}(P^{-2},\mathcal{B})}{}^{-1}(i+1) \\
ord_{cfg_{\mathcal{A}}(P,\mathcal{B})}{}^{-1}(b) &= ord_{cfg_{\mathcal{A}}(P^{-2},\mathcal{B})}{}^{-1}(a)
\end{aligned}$$

Což přímo podle definic 2.5.3 a 2.5.2 implikuje dokazované tvrzení. $\square$

Tvrzení 2.8.7 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $|P| \geq m$, $\mathcal{B} \in \mathcal{R}$ *platí*

$$(\forall i \in [m]): \; makespan_{\mathcal{A}}(P^{-i}, \mathcal{B}) \geq Load_{(\mathcal{A},P,\mathcal{B})}(-i) \tag{2.88}$$

Podle věty o linearitě střední hodnoty platí zde uvedené tvrzení i v průměru provedeném přes $\mathcal{B} \in \mathcal{R}$.

Důkaz. Postupuje se úpravami levé strany nerovnosti 2.88, makespan se přepíše dle jeho definice 2.5.5, dále se provede $(i-1)$-násobné použití tvrzení 2.8.6, čímž se dostane dokazovaná nerovnost. $\square$

Tvrzení 2.8.8 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, *kde* $|P| \geq m$ *platí*

$$\sum_{i \in [m]} \mathbb{E}makespan_{\mathcal{A}}(P^{-i}) \geq sum(P) \tag{2.89}$$

Důkaz. Důsledek tvrzení 2.8.7, protože formule 2.88 platí pro všechna $\mathcal{B} \in \mathcal{R}$, tak platí i v průměru, tedy

$$(\forall i \in [m]) : \ \mathbb{E}makespan_{\mathcal{A}}(P^{-i}) \geq \mathbb{E}load_{(\mathcal{A},P)}(-i) \tag{2.90}$$

Výraz z této formule lze posčítat přes i takto

$$\sum_{i \in [m]} \mathbb{E}makespan_{\mathcal{A}}(P^{-i}) \geq \sum_{i \in [m]} \mathbb{E}load_{(\mathcal{A},P)}(-i) = \tag{2.91}$$

dále použít větu o linearitě střední hodnoty dle definice 2.5.4, následně použít tvrzení 2.8.5 a odstranit průměrování konstanty

$$= \mathbb{E}_{\mathcal{B} \in \mathcal{R}} \left[Load_{(\mathcal{A},P,\mathcal{B})}(-i) \right] = \mathbb{E}_{\mathcal{B} \in \mathcal{R}} \left[sum(P) \right] = sum(P) \tag{2.92}$$

Tím je tvrzení dokázáno. □

Definice 2.8.9 Pro $m \in \mathbb{N}, m > 1$ *definujme*

$$\sigma_m \stackrel{def}{=} 1 + \frac{1}{\left(\frac{m}{m-1}\right)^m - 1} \tag{2.93}$$

Tato konstanta označuje dolní odhad kompetitivního poměru, který byl dokázán ve výchozím článku Sgall [15] a v této práci je dokazována jeho neoptimalita.

Tvrzení 2.8.10 *Platí*

$$\lim_{m \to \infty} \sigma_m = 1 + \frac{1}{e-1} \tag{2.94}$$

$$(\forall m \in \mathbb{N}, m > 1) : \ \sigma_m < 1 + \frac{1}{e-1} \tag{2.95}$$

Důkaz. Snadný, užitím tvrzení z matematické analýzy. □

Tvrzení 2.8.11 Pro $m \in \mathbb{N}, m > 1$, $P \in \mathcal{P}_K$ *platí*

$$\frac{sum(P)}{\sum_{i\in[m]} P_{-i}} = \sigma_m \tag{2.96}$$

Důkaz. Začne se úpravami levé strany rovnosti a dojde se postupně k pravé straně. Nejprve se použije tvrzení 2.8.1.

$$\frac{sum(P)}{\sum_{i\in[m]} P_{-i}} = \frac{m \cdot P_{-1}}{\sum_{i\in[m]} P_{-i}} = \tag{2.97}$$

Následně se dosadí podle tvrzení 2.7.11 a provede se posloupnost závěrečných elementárních úprav

$$= \frac{m \cdot \left(\frac{m}{m-1}\right)^{m-1} \cdot \frac{sum(P^{-m-1})}{m-1}}{\sum_{i\in[m]} \left(\left(\frac{m}{m-1}\right)^{m-i} \cdot \frac{sum(P^{-m-1})}{m-1}\right)} = \frac{m}{\sum_{i\in[m]} \left(\frac{m}{m-1}\right)^{1-i}} = \tag{2.98}$$

$$= 1 + \frac{1}{\left(\frac{m}{m-1}\right)^{m} - 1} = \sigma_m$$

□

Tvrzení 2.8.12 Pro $m \in \mathbb{N}, m > 1$, $c \in \mathbb{R}$, *c-kompetitivní algoritmus $\mathcal{A}$ platí*

$$c \geq \sigma_m \tag{2.99}$$

Kde σ_m je z definice 2.8.9.

Toto tvrzení je sice uvedeno ve výchozím článku viz. Sgall [15], avšak v méně obecné verzi. Zde bude ukázáno, že k jeho důkazu lze použít každou kritickou posloupnost definovanou v této práci. Každá kritická posloupnost je v tomto smyslu „špatná" a použitelná v dalších tvrzeních této práce.

Poznámka: *Definice použitá ve zmíněném článku uvažuje jen posloupnosti s právě m kritickými úlohami za spoustou jednotkových úloh.*

Důkaz. Důkaz je veden stejně, jako ve zmíněném článku. Je však použita zde uvedená definice kritické posloupnosti a výše přeformulovaná a zobecněná tvrzení. Nechť $P \in \mathcal{P}_K$ je kritická posloupnost. Důkaz je proveden přímo posloupností úprav a nerovností, začne se tvrzením 2.8.4, použije se nerovnost z tvrzení 2.8.8

$$c \geq \frac{\sum_{i\in[m]} \mathbb{E} makespan_{\mathcal{A}}(P^{-i})}{\sum_{i\in[m]} OPT(P^{-i})} \geq \frac{sum(P)}{\sum_{i\in[m]} OPT(P^{-i})} = \tag{2.100}$$

dále se použije tvrzení 2.8.2 a následně poslední krok je použití tvrzení 2.8.11

$$= \frac{sum(P)}{\sum_{i\in[m]} P_{-i}} = \sigma_m \tag{2.101}$$

a tím je získána dokazovaná nerovnost. □

2.9 Připomenutí lineárního programování

V důkazech bude použito lineární programování, proto je zde zmíněno několik základních faktů, které budou později použity.

Buď $m, n \in \mathbb{N}$, matice $m \times n : A \in \mathbb{R}^{n \cdot m}$, vektory $b \in \mathbb{R}^m$ a $c \in \mathbb{R}^n$.

Primární úloha je dána předpisem

$$\min \left\{ c^\top \cdot x \mid x \in \mathbb{R}^n \wedge Ax = b \wedge x \geq 0 \right\} \tag{2.102}$$

Duální úloha k primární úloze je dána předpisem

$$\max \left\{ b^\top \cdot y \mid y \in \mathbb{R}^m \wedge A^\top y \leq c \right\} \tag{2.103}$$

Pokud optimální hodnoty primárního a duálního řešení existují, potom se rovnají. Tím se zabývají následující tvrzení. Případné další podrobnosti lze nalézt v literatuře, například Grygarová [9].

Tvrzení 2.9.1 Nechť $m, n \in \mathbb{N}$, *matice* $m \times n : A \in \mathbb{R}^{n \cdot m}$, *vektory* $y, b \in \mathbb{R}^m$ *a* $x, c \in \mathbb{R}^n$, x, y *jsou přípustná řešení, tj.* $Ax = b \wedge x \geq 0$ *a* $A^\top y \leq c$, *jejichž ohodnocující funkce se rovnají* $c^\top x = b^\top y = H$. *Pak platí*

$$H = \min\left\{c^\top x \mid x \in \mathbb{R}^n \wedge Ax = b \wedge x \geq 0\right\} \tag{2.104}$$

a

$$H = \max\left\{b^\top y \mid y \in \mathbb{R}^m \wedge A^\top y \leq c\right\} \tag{2.105}$$

Důkaz. Nechť $p \in \mathbb{R}^n, q \in \mathbb{R}^m$ jsou přípusná řešení primární a duální úlohy. Potom platí

$$c^\top p \geq b^\top q \tag{2.106}$$

což se snadno ukáže úpravami: $c^\top p = p^\top c \geq p^\top A^\top q = (Ap)^\top q = b^\top q$, použily se podmínky primární a duální úlohy (v nerovnosti byl potřeba fakt $p \geq 0$).

Hodnota H je optimálním řešením úlohy primární i duální úlohy. Podle formule 2.106 je minimalizace i maximalizace omezená, za předpokladu existence přípustného řešení, což je předpoklad tohoto tvrzení. Pokud by H nebylo optimální řešení primární resp. duální úlohy, pak by existovalo $p\prime$ přípustné řešení primární úlohy resp. $q\prime$ přípustné řešení duální úlohy splňující nerovnost $c^\top p\prime < c^\top p$ resp. $b^\top q\prime > b^\top q$. Ovšem tyto nerovnosti spolu s předpokladem $c^\top p = b^\top q$ jsou ve sporu s formulí 2.106, která je splněna pro všechna přípustná řešení. $\square$

Poznámka: *Optimum lze nalézt tak, že se libovolným algoritmem nalezne optimální řešení pro primární i pro duální program. Pak již stačí jen ověřit přípustnost takovýchto řešení a splnění předpokladů tohoto tvrzení. Z toho už přímo vyplývá optimalita těchto řešení.*

Primární optimální řešení bylo spočteno simplexovým algoritmem. Duální optimální řešení bylo dopočítáno jako $y = \left(A_B{}^\top\right)^{-1} c_B$, kde B je báze v simplexové tabulce. Jak již bylo zmíněno, tak korektnost není třeba ověřovat, stačí jen ověřit předpoklady.

Tvrzení 2.9.2 Nechť $m, n \in \mathbb{N}$, *matice* $m \times n : A \in \mathbb{R}^{n \cdot m}$, *vektory* $b \in \mathbb{R}^m$ *a* $c \in \mathbb{R}^n$, *vektor* $y \in \mathbb{R}^m$ *je přípustné duální řešení, že* $A^\top y \leq c$, *optimální řešení primární úlohy je omezené.*

Platí, že

$$b^\top y \leq \min\left\{c^\top x \mid x \in \mathbb{R}^n \wedge Ax = b \wedge x \geq 0\right\} \tag{2.107}$$

Optimální hodnotu omezeného primárního lineárního programu lze zdola odhadnout hodnotou účelové funkce pro libovolné přípustné duální řešení.

Důkaz. S předpokladem omezenosti podle formule 2.106 je každé primární řešení odhadnuto každým duálním přípustným řešením, tedy to musí platit i pro optimální primární řešení. □

Tvrzení 2.9.3 Nechť $m, n \in \mathbb{N}$, *matice* $m \times n : A \in \mathbb{R}^{n \cdot m}$, *jednotková matice* $m \times m : I \in \mathbb{R}^{m \cdot m}$, *spojená matice* $m \times (n+m) : [A|I] \in \mathbb{R}^{(n+m) \cdot m}$, *vektor pravé strany* $b \in \mathbb{R}^m$, *vektory ohodnocující funkce* $c \in \mathbb{R}^n$, *jednotkový vektor* $e \in \mathbb{R}^m$ *a nulový vektor* $u \in \mathbb{R}^n$.

Platí, že ***primární úloha má přípustné řešení****, tedy*

$$\left\{c^\top x \mid x \in \mathbb{R}^n \wedge Ax = b \wedge x \geq 0\right\} \neq \emptyset \tag{2.108}$$

právě když následující úloha má nulové optimální řešení, tedy

$$\min\left\{[u|e]^\top [x|\tilde{x}] \mid x \in \mathbb{R}^n \wedge \tilde{x} \in \mathbb{R}^m \wedge [A|I]\,[x|\tilde{x}] = b \wedge [x|\tilde{x}] \geq 0\right\} = 0 \tag{2.109}$$

Toto tvrzení dává postup, jak pro libovolnou primární úloh nalézt přípustné řešení nebo dokázat, že neexistuje. Přípustné řešení nové úlohy je triviální, stačí zvolit $x = 0$ a $\tilde{x} = b$.

Důkaz. Důkaz se provede dokázáním obou implikací.

- Necht' primární úloha 2.108 má přípustné řešení x, potom x spolu s volbou $\tilde{x} = 0$ je přípustným a navíc optimálním řešením nové úlohy 2.109. Optimum je zdola omezeno 0 a toto řešení hodnotu 0 nabývá.
- Necht' nová úloha 2.109 má optimální řešení $x, \tilde{x}$ s hodnotou 0. Potom z přípustnosti tohoto řešení a nulovosti $\tilde{x}$ vyplývá, že x je přípustným řešením primární úlohy 2.108.

$\square$

Kapitola 3

Neoptimálnost existujícího odhadu

Chen, van Vliet a Woeginger [4], nezávisle Sgall [15], sestrojili obecný dolní odhad pro pravděpodobnostní on-line algoritmy. Dokázali, že každý pravděpodobnostní on-line algoritmus na m počítačích má kompetitivní poměr alespoň

$$1 + \frac{1}{\left(\frac{m}{m-1}\right)^m - 1} \tag{3.1}$$

Tento odhad je pro $m = 2$ počítače optimální. Vzhledem k tomu, že to je celkem „hezký“ výsledek, se intuitivně spekulovalo o tom, že tento odhad je optimální.

Avšak náplní této kapitoly je dokázat neoptimálnost tohoto dolního odhadu. Bohužel důkaz byl sestrojen zatím jen pro $m = 3$ počítače, ale i tento výsledek je značně netriviální.

3.1 Cíl této kapitoly

Dokazované tvrzení o neexistenci σ_m-kompetitivního algoritmu pro $m = 3$ počítače je formulováno takto

Tvrzení 3.1.1 Pro $m = 3$, $d \in \mathbb{R}$, *d-kompetitivní algoritmus* $\mathcal{A}$ *platí*

$$d > \sigma_m \tag{3.2}$$

Důkaz uvedeného tvrzení je náplní zbytku této kapitoly. Důkaz bude uveden později jako důsledek jiných tvrzení, až všechna potřebná tvrzení budou zformulována a dokázána. Zbývající tvrzení budou formulována obecněji, aby tento důkaz byl snadno rozšiřitelný nebo aby zobecnění alespoň usnadnil.

V důkazu sporu je vhodné předpokladat, že m je pevné a *algoritmus* $\mathcal{A}$ je pevný. Tedy pro spor předpokládaná negace tvrzení 3.1.1 je

$$\textit{algoritmus } \mathcal{A} \textit{ je } \sigma_m\textit{-kompetitivní algoritmus} \tag{3.3}$$

3.2 Výsledky této práce

Tvrzení 3.2.1 Pro $m \in \mathbb{N}, m > 1,$ σ_m*-kompetitivní algoritmus* $\mathcal{A}$, $P \in \mathcal{P}_K$ *platí*

$$(\forall i \in [m-1]) : \frac{\mathbb{E}\textit{load}_{(\mathcal{A},P)}(i+1)}{\mathbb{E}\textit{load}_{(\mathcal{A},P)}(i)} = \frac{m}{m-1} \tag{3.4}$$

Tvrzení tedy říká, že algoritmus bude mít průměrné zátěže v poměru $1 : \left(\frac{m}{m-1}\right)^1 : \cdots : \left(\frac{m}{m-1}\right)^{m-1}$.

Poznámka: *Byly zkoumány možnosti jak toto tvrzení zobecnit, avšak nepodařilo se vyhnout potřebnosti tvrzení* 2.8.11, *které požaduje kritickou posloupnost.*

Důkaz. Z tvrzení 2.8.5, 2.8.7, 2.8.4 a 2.8.2

$$sum(P) = \sum_{i\in[m]} \mathbb{E}load_{(\mathcal{A},P)}(i) \leq \sum_{i\in[m]} \mathbb{E}makespan_{\mathcal{A}}(P^{-i}) \leq \tag{3.5}$$

$$\leq \sigma_m \cdot \sum_{i\in[m]} OPT(P^{-i}) = \sigma_m \cdot \sum_{i\in[m]} P_{-i}$$

Z aplikace tvrzení 2.8.11 na předchozí nerovnost 3.5 plyne, že všechny nerovnosti musí být rovnostmi a tedy platí následující vztah

$$\sum_{i\in[m]} \mathbb{E}makespan_{\mathcal{A}}(P^{-i}) = \sum_{i\in[m]} \mathbb{E}load_{(\mathcal{A},P)}(-i) = \sigma_m \cdot \sum_{i\in[m]} P_{-i} \tag{3.6}$$

Podle tvrzení 2.8.7 platí nerovnost 2.88 v průměru přes $\mathcal{B} \in \mathcal{R}$, ovšem tyto nerovnosti jsou posčítáním první rovnosti z vztahu 3.6, tedy platí rovnost ve všech případech

$$(\forall i \in [m]) : \ \mathbb{E}makespan_{\mathcal{A}}(P^{-i}) = \mathbb{E}load_{(\mathcal{A},P)}(-i) \tag{3.7}$$

Podle vztahu 3.7, definice 2.8.3 a tvrzení 2.8.2 platí

$$(\forall i \in [m]) : \ \mathbb{E}load_{(\mathcal{A},P)}(-i) = \mathbb{E}makespan_{\mathcal{A}}(P^{-i}) \leq \sigma_m \cdot P_{-i} \tag{3.8}$$

Vztah 3.6 je posčítáním vztahu 3.8, ve vztahu 3.8 se proto nabývají vždy rovnosti. Proto platí vztah (odstraněno znaménko)

$$(\forall i \in [m]) : \ \mathbb{E}load_{(\mathcal{A},P)}(i) = \sigma_m \cdot P_{i-1-m} = \sigma_m \cdot P_{|P|-m+i} \tag{3.9}$$

Pro získání dokazované rovnosti stačí dosadit a upravit podle tvrzení 2.7.11

$$(\forall i \in [m-1]) : \ \frac{\mathbb{E}load_{(\mathcal{A},P)}(i+1)}{\mathbb{E}load_{(\mathcal{A},P)}(i)} = \frac{\sigma_m \cdot P_{|P|-m+i+1}}{\sigma_m \cdot P_{|P|-m+i}} = \frac{m}{m-1} \tag{3.10}$$

a tím je celé tvrzení dokázáno. □

Tvrzení 3.2.2 Pro $m \in \mathbb{N}, m > 1,$ σ_m*-kompetitivní algoritmus* $\mathcal{A}$, $P \in \mathcal{P}_K$ *platí*

$$(\forall i \in [m]) : \ \mathbb{E}load_{(\mathcal{A},P)}(i) = \left(\frac{m}{m-1}\right)^{i-1} \frac{sum(P)}{\sum_{j\in[m]} \left(\frac{m}{m-1}\right)^{j-1}} \tag{3.11}$$

$$(\forall i \in [m]) : \ \mathbb{E}load_{(\mathcal{A},P)}(i) = \sigma_m \cdot \left(\frac{m}{m-1}\right)^{i-1} \frac{sum(P^{-m-1})}{m-1} \tag{3.12}$$

Tvrzení explicitně vyjadřuje hodnoty průměrných zátěží pro danou kritickou posloupnost.

Důkaz. Tvrzení je důsledkem předchozího tvrzení 3.2.1. Podle tvrzení 2.8.5 je součet všech průměrných zátěží roven součtu velikostí všech úloh posloupnosti. Rovněž podle stejného tvrzení jsou jednoznačně určeny poměry průměrných zátěží, tedy mohou být přesně vyjádřeny formulí 3.11. Formule 3.12 se z formule 3.11 získá posloupností úprav. Ve formuli 3.12 se rozepíše $sum(P)$ na sumu velikostí posledních m kritických úloh a zbytek. Při dosazování za velikosti kritických úloh se použije tvrzení 2.7.11. Zbytek jsou již jen elementární matematické úpravy. □

Tvrzení 3.2.3 Pro $m \in \mathbb{N}, m > 1,$ σ_m*-kompetitivní algoritmus* $\mathcal{A}$, $P \in \mathcal{P}_K$, $P^{-2} \in \mathcal{P}_K$

$$(\forall i \in [m-1]) : \ \mathbb{E}load_{(\mathcal{A},P)}(i) = \mathbb{E}load_{(\mathcal{A},P^{-2})}(i+1) \tag{3.13}$$

$$\mathbb{E}load_{(\mathcal{A},P)}(m) = \mathbb{E}load_{(\mathcal{A},P^{-2})}(1) + P_{-1} \tag{3.14}$$

Nová průměrná zátěž i-tého minimálního počítače je rovna staré zátěži $(i+1)$-ního minimálního počítače.

Důkaz. Snadný s použitím tvrzení 3.2.1, které zaručuje stejné poměry všech průměrných zátěží. Dále s použitím tvrzení 2.7.11. □

Definice 3.2.4 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $\mathcal{B} \in \mathcal{R}$ *definujme množinu*

$$\mathcal{M}_{\mathcal{A}}(P, \mathcal{B}) \stackrel{def}{=} \left\{ j \in [m] \mid \left[cfg_{\mathcal{A}}(P, \mathcal{B}) \right]_j^m = \min_{k \in [m]} \left[cfg_{\mathcal{A}}(P, \mathcal{B}) \right]_k^m \right\} \tag{3.15}$$

Tato množina je množina indexů počítačů, které mají minimální zátěž v konfiguraci, do které se algoritmus na dané posloupnosti úloh a posloupnosti náhodných bitů dostane.

Tvrzení 3.2.5 Pro $m \in \mathbb{N}, m > 1$, σ_m*-kompetitivní algoritmus* $\mathcal{A}$, $P \in \mathcal{P}_K$, $P^{-2} \in \mathcal{P}_K$, $\mathcal{B} \in \mathcal{R}_{\mathcal{A}}$ *platí*

$$\left(\forall i \in [|P|], P^{i-1} \in \mathcal{P}_K \right) : \ \mathcal{A}(P, \mathcal{B}, i) \in \mathcal{M}_{\mathcal{A}}(P^{i-1}, \mathcal{B}) \tag{3.16}$$

a

$$Load_{(\mathcal{A},P,\mathcal{B})}(m) = Load_{(\mathcal{A},P^{-2},\mathcal{B})}(1) + P_{-1} \tag{3.17}$$

To znamená, že algoritmus se chová deterministicky a to tak, že vždy umísťuje kritickou úlohu (dostatečně dlouhé) kritické posloupnosti na počítač s minimální zátěží, navíc se z něj stane počítač s maximální zátěží.

Důkaz. Buď $i \in [m-1]$ pevné a $\mathcal{B}$ pevné. Pak z tvrzení 2.8.6 plyne

$$Load_{(\mathcal{A},P,\mathcal{B})}(i) \leq Load_{(\mathcal{A},P^{-2},\mathcal{B})}(i+1) \tag{3.18}$$

Cílem je dokázat v formuli 3.18 rovnost, proto se pro spor předpokládá ostrá nerovnost, tedy

$$Load_{(\mathcal{A},P,\mathcal{B})}(i) < Load_{(\mathcal{A},P^{-2},\mathcal{B})}(i+1) \tag{3.19}$$

Toto se ovšem zprůměruje přes $\mathcal{B} \in \mathcal{R}_{\mathcal{A}}$, z definice 2.4.5 plyne, že získaná konfigurace pro dané $\mathcal{B}$, P a i má nenulovou pravděpodobnost, nutně se musí projevit v průměru, tudíž ostrá nerovnost zůstane zachována. Formálně stačí dosadit podle definice 2.5.4 takto

$$\mathbb{E}load_{(\mathcal{A},P)}(i) = \mathbb{E}_{\mathcal{B}\in\mathcal{R}}\left[Load_{(\mathcal{A},P,\mathcal{B})}(i)\right] = \tag{3.20}$$

podle tvrzení 2.20 ukazuje, že $Pr_{\mathcal{B}_1\in\mathcal{R}}\left[\mathcal{B}_1 \in \mathcal{R} - \mathcal{R}_{\mathcal{A}}\right] = 0$, tedy množina odebraných hodnot má míru nula, tudíž neovlivní průměr (z jeho definice), proto platí

$$= \mathbb{E}_{\mathcal{B}\in\mathcal{R}_{\mathcal{A}}}\left[Load_{(\mathcal{A},P,\mathcal{B})}(i)\right] < \tag{3.21}$$

podle definice 2.4.5 množina náhodných posloupností mající stejnou zátěž nějakého počítače (obecněji stejnou konfiguraci) jako posloupnost $\mathcal{B} \in \mathcal{R}_{\mathcal{A}}$ má nenulovou míru. Tudíž se podle předpokladu 3.19 zvýší hodnota průměru a platí

$$< \mathbb{E}_{\mathcal{B}\in\mathcal{R}_{\mathcal{A}}}\left[Load_{(\mathcal{A},P^{-2},\mathcal{B})}(i+1)\right] = \tag{3.22}$$

následují dvě elementární úpravy

$$= \mathbb{E}_{\mathcal{B}\in\mathcal{R}}\left[Load_{(\mathcal{A},P^{-2},\mathcal{B})}(i+1)\right] = \mathbb{E}load_{(\mathcal{A},P^{-2})}(i+1) \tag{3.23}$$

Když se v zápisu vynechají mezikroky, tak se obdrží ostrá nerovnost

$$\mathbb{E}load_{(\mathcal{A},P)}(i) < \mathbb{E}load_{(\mathcal{A},P^{-2})}(i+1) \tag{3.24}$$

což je ve sporu s tvrzením 3.2.3.

Tedy platí

$$(\forall i \in [m-1]) : \; Load_{(\mathcal{A},P,\mathcal{B})}(i) = Load_{(\mathcal{A},P^{-2},\mathcal{B})}(i+1) \tag{3.25}$$

Toto se netýká $Load_{(\mathcal{A},P,\mathcal{B})}(m)$ ani $Load_{(\mathcal{A},P^{-2},\mathcal{B})}(1)$, přitom přidáním úlohy P_{-1} se může změnit zátěž jen jediného počítače, proto dle definic 2.5.3 a 2.4.3 platí

$$Load_{(\mathcal{A},P,\mathcal{B})}(m) = Load_{(\mathcal{A},P^{-2},\mathcal{B})}(1) + P_{-1} \tag{3.26}$$

což je právě dokazovaný vztah 3.17.

Úloha byla umístěna na počítač s minimální zátěží. Z definice 2.5.2 mohlo být více počítačů s minimální zátěží, právě proto je zde počítáno minimum.

Tím je tvrzení dokázáno pro $i = |P|$. Zbytek lze snadno dokázat indukcí. Omezující podmínka je dána předpokladem existence kratší kritické posloupnosti kvůli předpokladům tvrzení 3.2.3 a 2.8.6. □

Tvrzení 3.2.6 Pro $m \in \mathbb{N}, m > 1$, *σ_m-kompetitivní algoritmus $\mathcal{A}$, $P \in \mathcal{P}_K$, $i \in [m]$, $\mathcal{B} \in \mathcal{R}_\mathcal{A}$, $P^{-m} \in \mathcal{P}_K$ platí*

$$(\forall i \in [m]): \; Load_{(\mathcal{A},P^{-i},\mathcal{B})}(i) = Load_{(\mathcal{A},P,\mathcal{B})}(1) \tag{3.27}$$

Důkaz. Indukcí podle i s pomocí tvrzení 3.2.5. □

Tvrzení 3.2.7 Pro $m \in \mathbb{N}, m > 1$, *σ_m-kompetitivní algoritmus $\mathcal{A}$, $P \in \mathcal{P}_K$, $i \in [m]$, $\mathcal{B} \in \mathcal{R}_\mathcal{A}$, $P^{-m} \in \mathcal{P}_K$ platí*

$$Load_{(\mathcal{A},P,\mathcal{B})}(1) = Load_{(\mathcal{A},P^{-m-1},\mathcal{B})}(1) + \mathcal{K}(P^{-m-1}) \tag{3.28}$$

a obecněji

$$Load_{(\mathcal{A},P,\mathcal{B})}(i) = Load_{(\mathcal{A},P^{-m-1},\mathcal{B})}(i) + \mathcal{K}(P^{-m+i-2}) \tag{3.29}$$

Algoritmus od okamžiku, kdy začne dávat kritické úlohy na počítač s minimální zátěží a z něj se začne stávat počítač s maximální zátěží, již jen podle tohoto schématu cyklí a zvyšuje zátěže počítačů.

Důkaz. První formule se dokáže dle tvrzení 3.2.5. Druhá je jen posloupností úprav, ve kterých se použije první formule a tvrzení 3.2.6 na posloupnost P nebo na posloupnost ji prodloužující o nějaké další kritické úlohy, aby bylo možno použít tvrzení. □

Tvrzení 3.2.8 Pro $m \in \mathbb{N}, m > 1$, σ_m*-kompetitivní algoritmus* $\mathcal{A}$ *platí*

$$(\forall \varepsilon > 0)(\exists n \in \mathbb{N})(\forall P \in \mathcal{P}_K, Crit(P, n))(\forall \mathcal{B} \in \mathcal{R}_{\mathcal{A}})(\forall i \in [m-1]) : \tag{3.30}$$
$$\left| \frac{Load_{(\mathcal{A},P,\mathcal{B})}(i+1)}{Load_{(\mathcal{A},P,\mathcal{B})}(i)} - \frac{m}{m-1} \right| < \varepsilon$$

Má-li algoritmus na vstupu kritickou posloupnost, potom lze výslednou konfiguraci libovolně přesně (ve smyslu poměrů, ne absolutně) přibližovat průměrné konfiguraci. Přibližování závisí na „prodlužování“ vstupní posloupnosti.

Poznámka: *„Prodlužování“ je ve smyslu počtu kritických úloh v posloupnosti. Kvůli požadavku celočíselnosti nelze uvažovat jen přidávání nových úloh.*

Důkaz. Buď $n \in \mathbb{N}$ libovolně (dostatečně) veliké, podle tvrzení 2.7.12 existuje kritická posloupnost $P \in \mathcal{P}_K$ mající alespoň n posledních úloh kritických. Podle tvrzení 2.7.9 platí $P^{-n+m-1} \in \mathcal{P}_K$ (zrovna tak všechny podposloupnosti P s tímto prefixem jsou kritické), označme $Q = P^{-n+m-1}$.

Nyní bude ukázána konvergence konfigurace k průměrné konfiguraci. Z tvrzení 3.2.7 dosazením P^j plyne

$$(\forall j \in [|P|], j > |Q| + m)(\forall i \in [m]) : \tag{3.31}$$

$$Load_{(\mathcal{A},P^j,\mathcal{B})}(i) = Load_{(\mathcal{A},P^{j-m},\mathcal{B})}(i) + \mathcal{K}(P^{j-m+i-1})$$

Indukcí podle k lze dokázat následující tvrzení, které se získá iterováním právě odvozené rovnosti

$$(\forall j, k \in [|P|], j > |Q|, j + k \cdot m \leq |P|)(\forall i \in [m]) : \tag{3.32}$$

$$Load_{(\mathcal{A},P^{j+k\cdot m},\mathcal{B})}(i) = Load_{(\mathcal{A},P^{j},\mathcal{B})}(i) + \sum_{t=0}^{k-1} \mathcal{K}(P^{j+t\cdot m+i-1}) =$$

Dále stačí dosadit z tvrzení 2.7.10

$$= Load_{(\mathcal{A},P^{j},\mathcal{B})}(i) + \sum_{t=0}^{k-1} \left(\frac{m}{m-1}\right)^{t\cdot m+i-1} \cdot \frac{sum(P^j)}{m-1} = \tag{3.33}$$

a sečíst geometrickou řadu

$$= Load_{(\mathcal{A},P^{j},\mathcal{B})}(i) + \frac{sum(P^j)}{m-1} \cdot \left(\frac{m}{m-1}\right)^{i-1} \cdot \frac{\left(\frac{m}{m-1}\right)^{m\cdot k} - 1}{\left(\frac{m}{m-1}\right)^{m} - 1} \tag{3.34}$$

Buď pevné $\mathcal{B} \in \mathcal{R}_{\mathcal{A}}$, $i \in [m-1]$, $j \in [|P|]$, $j > |Q|$, $j + k \cdot m \leq |P|$, $\varepsilon > 0$, nerovnost dokazovaného z tvrzení (s drobnou úpravou, která bude následně napravena) se dokáže posloupností úprav, podmínka nenulovosti jmenovatele je triviálně splněna

$$\left| \frac{Load_{(\mathcal{A},P^{j+k\cdot m},\mathcal{B})}(i+1)}{Load_{(\mathcal{A},P^{j+k\cdot m},\mathcal{B})}(i)} - \frac{m}{m-1} \right| = \tag{3.35}$$

$$= \left| \frac{Load_{(\mathcal{A},P^{j+k\cdot m},\mathcal{B})}(i+1) - \frac{m}{m-1} \cdot Load_{(\mathcal{A},P^{j+k\cdot m},\mathcal{B})}(i)}{Load_{(\mathcal{A},P^{j+k\cdot m},\mathcal{B})}(i)} \right| = \tag{3.36}$$

dále se dosadí podle vztahu 3.32 výraz 3.34, získaný „velký člen“ se v čitateli odečtem, výsledkem je následující výraz

$$= \left| \frac{Load_{(\mathcal{A},P^{j},\mathcal{B})}(i+1) - Load_{(\mathcal{A},P^{j},\mathcal{B})}(i)}{Load_{(\mathcal{A},P^{j+k\cdot m},\mathcal{B})}(i)} \right| \leq \tag{3.37}$$

Vnitřní člen se shora odhadne hodnotou $sum(P^j)$, zároveň se odstraní absolutní hodnota

$$\leq \frac{sum(P^j)}{Load_{(\mathcal{A},P^{j+k\cdot m},\mathcal{B})}(i)} = \tag{3.38}$$

Do jmenovatele se dosadí výraz 3.34 podle rovnosti dané vztahem 3.32

$$= \frac{sum(P^j)}{Load_{(\mathcal{A},P^{j},\mathcal{B})}(i) + \frac{sum(P^j)}{m-1} \cdot \left(\frac{m}{m-1}\right)^{i-1} \cdot \frac{\left(\frac{m}{m-1}\right)^{m\cdot k} - 1}{\left(\frac{m}{m-1}\right)^{m} - 1}} \leq \tag{3.39}$$

Následuje horní odhad vynecháním prvního členu ve jmenovateli následovaný pokrácením s čitatelem, čímž odpadá závislost na proměnné j, takto

$$\leq \frac{(m-1)\cdot\left(\frac{m}{m-1}\right)^{-i+1}\cdot\left(\left(\frac{m}{m-1}\right)^{m}-1\right)}{\left(\frac{m}{m-1}\right)^{m\cdot k}-1} \leq \tag{3.40}$$

Dalším odhadem odpadne závislost na i, neboť $\left(\frac{m}{m-1}\right)^{i-1} \leq 1$, tedy

$$\leq \frac{(m-1)\cdot\left(\left(\frac{m}{m-1}\right)^{m}-1\right)}{\left(\frac{m}{m-1}\right)^{m\cdot k}-1} \leq \tag{3.41}$$

Toto se posloupností elementárních úprav a použitím nerovnosti z matematické analýzy $(\forall x \in \mathbb{R}, x > -1):\ \ln(1+x) \leq x$ upraví na tvar

$$\leq \frac{e\cdot m}{\left(\frac{m}{m-1}\right)^{m\cdot k}-1} \tag{3.42}$$

Bude-li k zvoleno tak, aby

$$k > \frac{\ln(\frac{e\cdot m}{\varepsilon}+1)}{m\cdot\ln\left(\frac{m}{m-1}\right)} \tag{3.43}$$

pak lze odvozovanou posloupnost nerovností zakončit horním odhadem ε, tedy tím bylo dokázáno, že

$$\left|\frac{\mathit{Load}_{(\mathcal{A},P^{j+k\cdot m},\mathcal{B})}(i+1)}{\mathit{Load}_{(\mathcal{A},P^{j+k\cdot m},\mathcal{B})}(i)} - \left(\frac{m}{m-1}\right)\right| < \varepsilon \tag{3.44}$$

K tomu je ještě nutný předpoklad, že máme takové P, tedy nutně $n \geq j + k\cdot m$. Složka j určuje minimální počet kritických úloh v kritické posloupnosti splňující předpoklady tvrzení 3.2.5, což je $m+1$ úloh (P^{-2} má být kritická).

Celkem je postačující požadavek

$$n \geq \frac{\ln(\frac{e\cdot m}{\varepsilon}+1)}{\ln\left(\frac{m}{m-1}\right)} + m + 2 \tag{3.45}$$

další $+1$ přibylo kvůli neostré nerovnosti.

Existence kritické posloupnosti splňující $Crit(P, n)$ vyplývá z tvrzení 2.7.12 a definice 2.7.5. □

Tvrzení 3.2.9 Pro $m \in \mathbb{N}, m > 1$, *σ_m-kompetitivní algoritmus $\mathcal{A}$ platí*

$$(\forall \varepsilon > 0)(\exists n \in \mathbb{N})(\forall P \in \mathcal{P}_K, Crit(P, n))(\forall \mathcal{B} \in \mathcal{R}_{\mathcal{A}})(\forall i \in [m]) : \tag{3.46}$$
$$\left| \frac{Load_{(\mathcal{A},P,\mathcal{B})}(i)}{sum(P)} - \frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j\in[m]} \left(\frac{m}{m-1}\right)^{j-1}} \right| < \varepsilon$$

Je důsledkem tvrzení 3.2.8 a jen jiným způsobem říká, že se algoritmus blíží průměrné konfiguraci. Explicitně vyjadřuje průměrnou konfiguraci. Určuje, jak moc dobrou aproximací je průměrná konfigurace pro konfiguraci při pevné náhodné posloupnosti.

Důkaz. Buď $\varepsilon\prime > 0$ dostatečně malé, potom podle tvrzení 3.2.8 existuje posloupnost, splňující stejné předpoklady a platí

$$(\forall i \in [m-1]) : \left| \frac{Load_{(\mathcal{A},P,\mathcal{B})}(i+1)}{Load_{(\mathcal{A},P,\mathcal{B})}(i)} - \frac{m}{m-1} \right| < \varepsilon\prime \tag{3.47}$$

Dále důkaz bude pokračovat tak, že se spočte $\varepsilon\prime$ v závislosti na ε, aby bylo splněno i toto tvrzení.

Následující nerovnosti lze snadno odvodit z tvrzení 3.2.8

$$(\forall i \in [m-1]) : \; Load_{(\mathcal{A},P,\mathcal{B})}(i+1) < Load_{(\mathcal{A},P,\mathcal{B})}(i) \cdot \left(\frac{m}{m-1} + \varepsilon\prime \right) \tag{3.48}$$

$$(\forall i \in [m-1]) : \; Load_{(\mathcal{A},P,\mathcal{B})}(i+1) > Load_{(\mathcal{A},P,\mathcal{B})}(i) \cdot \left(\frac{m}{m-1} - \varepsilon\prime \right) \tag{3.49}$$

Z čehož se indukcí podle i dokáží nerovnosti

$$(\forall i \in [m]) : \; Load_{(\mathcal{A},P,\mathcal{B})}(i) \leq Load_{(\mathcal{A},P,\mathcal{B})}(1) \cdot \left(\frac{m}{m-1} + \varepsilon\prime \right)^{i-1} \tag{3.50}$$

$$(\forall i \in [m]) : \ Load_{(\mathcal{A},P,\mathcal{B})}(i) \geq Load_{(\mathcal{A},P,\mathcal{B})}(1) \cdot \left(\frac{m}{m-1} - \varepsilon\prime\right)^{i-1} \tag{3.51}$$

A jejich posčítáním a úpravou se odvodí nerovnosti

$$\frac{Load_{(\mathcal{A},P,\mathcal{B})}(1)}{sum(P)} = \frac{Load_{(\mathcal{A},P,\mathcal{B})}(1)}{\sum_{i\in[m]} Load_{(\mathcal{A},P,\mathcal{B})}(i)} \geq \frac{1}{\sum_{i\in[m]} \left(\frac{m}{m-1} + \varepsilon\prime\right)^{i-1}} \tag{3.52}$$

$$\frac{Load_{(\mathcal{A},P,\mathcal{B})}(1)}{sum(P)} = \frac{Load_{(\mathcal{A},P,\mathcal{B})}(1)}{\sum_{i\in[m]} Load_{(\mathcal{A},P,\mathcal{B})}(i)} \leq \frac{1}{\sum_{i\in[m]} \left(\frac{m}{m-1} - \varepsilon\prime\right)^{i-1}} \tag{3.53}$$

Následovat bude posloupnost úprav výrazu, kterými se nakonec dospěje k hornímu odhadu $\varepsilon\prime$ v závislosti na ε, aby toto dokazované tvrzení platilo.

$$\left| \frac{Load_{(\mathcal{A},P,\mathcal{B})}(i)}{sum(P)} - \frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j\in[m]} \left(\frac{m}{m-1}\right)^{j-1}} \right| \leq \tag{3.54}$$

Aplikací zřejmé nerovnosti $(\forall a, b \in \mathbb{R}) : \ |a - b| \leq \max\{a, b\} - \min\{a, b\}$ se dostane

$$\leq \max\left\{ \frac{Load_{(\mathcal{A},P,\mathcal{B})}(i)}{sum(P)}, \frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j\in[m]} \left(\frac{m}{m-1}\right)^{j-1}} \right\} - \tag{3.55}$$

$$- \min\left\{ \frac{Load_{(\mathcal{A},P,\mathcal{B})}(i)}{sum(P)}, \frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j\in[m]} \left(\frac{m}{m-1}\right)^{j-1}} \right\} \leq$$

dále se aplikují spočtené horní a dolní odhady 3.50 a 3.51 takto

$$\leq \max\left\{ \frac{Load_{(\mathcal{A},P,\mathcal{B})}(1) \cdot \left(\frac{m}{m-1} + \varepsilon\prime\right)^{i-1}}{sum(P)}, \frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j\in[m]} \left(\frac{m}{m-1}\right)^{j-1}}) \right\} - \tag{3.56}$$

$$- \min\left\{ \frac{Load_{(\mathcal{A},P,\mathcal{B})}(1) \cdot \left(\frac{m}{m-1} - \varepsilon\prime\right)^{i-1}}{sum(P)}, \frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j\in[m]} \left(\frac{m}{m-1}\right)^{j-1}} \right\} \leq$$

dále se aplikují horní a dolní odhady 3.52 a 3.53 spočtené pro právě získané členy, tím se získá

$$\leq \max\left\{\frac{\left(\frac{m}{m-1}+\varepsilon\prime\right)^{i-1}}{\sum_{i\in[m]}\left(\frac{m}{m-1}-\varepsilon\prime\right)^{i-1}}, \frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j\in[m]}\left(\frac{m}{m-1}\right)^{j-1}})\right\} - \tag{3.57}$$

$$-\min\left\{\frac{\left(\frac{m}{m-1}-\varepsilon\prime\right)^{i-1}}{\sum_{i\in[m]}\left(\frac{m}{m-1}+\varepsilon\prime\right)^{i-1}}, \frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j\in[m]}\left(\frac{m}{m-1}\right)^{j-1}}\right\} \leq$$

s předpokladem $\frac{m}{m-1} > \varepsilon\prime$ se lze zbavit operátorů $\min$ a $\max$ takto

$$\leq \frac{\left(\frac{m}{m-1}+\varepsilon\prime\right)^{i-1}}{\sum_{i\in[m]}\left(\frac{m}{m-1}-\varepsilon\prime\right)^{i-1}} - \frac{\left(\frac{m}{m-1}-\varepsilon\prime\right)^{i-1}}{\sum_{i\in[m]}\left(\frac{m}{m-1}+\varepsilon\prime\right)^{i-1}} = \tag{3.58}$$

dále bude sečtena geometrická řada ve jmenovatelích

$$= \frac{\left(\frac{m}{m-1}+\varepsilon\prime\right)^{i-1}\cdot\left(\frac{m}{m-1}-\varepsilon\prime-1\right)}{\left(\frac{m}{m-1}-\varepsilon\prime\right)^{m}-1} - \frac{\left(\frac{m}{m-1}-\varepsilon\prime\right)^{i-1}\cdot\left(\frac{m}{m-1}+\varepsilon\prime-1\right)}{\left(\frac{m}{m-1}+\varepsilon\prime\right)^{m}-1} \leq \tag{3.59}$$

dále je převedeno na společný jmenovatel a ten je hrubě odhadnut s pomocí předpokladu $\frac{\left(\frac{m}{m-1}\right)}{2} > \varepsilon\prime$, roznásoben čitatel a proveden horní odhad členů, které tím zmizí

$$\leq \frac{\left(\frac{m}{m-1}\right)}{\left(\frac{1}{2}\cdot\frac{m}{m-1}\right)^{m}-1}\cdot\left(\left(\frac{m}{m-1}+\varepsilon\prime\right)^{m+i-1} - \left(\frac{m}{m-1}-\varepsilon\prime\right)^{m+i-1}\right) \leq \tag{3.60}$$

následně je použita nerovnost plynoucí z binomické věty a to, že platí

$$(\forall x \geq 1)(\forall \varepsilon\prime > 0)(\forall j \in \mathbb{N}):\ (x+\varepsilon\prime)^{j} \leq x^{j}\cdot(1+\varepsilon\prime)^{j} \tag{3.61}$$

což je aplikováno pro $x = \frac{m}{m-1} - \varepsilon\prime$, takto získaný člen se vytkne a odhadne

$$\leq \frac{\left(\frac{m}{m-1}\right)^{m+i}}{\left(\frac{1}{2}\cdot\frac{m}{m-1}\right)^{m}-1}\cdot\left((1+2\varepsilon\prime)^{m+i-1}-1\right) \leq \tag{3.62}$$

dalším krokem je horní odhad eliminací neznámé i takto

$$\leq \frac{\left(\frac{m}{m-1}\right)^{2m}}{\left(\frac{1}{2}\cdot\frac{m}{m-1}\right)^{m}-1}\cdot\left((1+2\varepsilon\prime)^{2m-1}-1\right) \tag{3.63}$$

Uvedený výraz má být menší než ε, tedy snadnou úpravou se získá požadavek na $\varepsilon\prime$, tedy

$$\varepsilon\prime < \frac{\sqrt[2m-1]{\varepsilon \cdot \frac{\left(\frac{m}{m-1}\right)^m - 1}{\left(\frac{m}{m-1}\right)^{2m}} + 1} - 1}{2} \tag{3.64}$$

Uvedený důkaz je korektní, neboť získaný požadavek je vždy kladný a tím je tvrzení dokázáno. □

3.3 Základní důkaz pro $m = 3$ počítače

Následuje zbytek důkazu tvrzení 3.1.1. Důkaz je rozdělen na logické části, každá část má vlastní kapitolku.

3.3.1 Myšlenka důkazu

Důkaz bude proveden sporem. Vyjde se z předpokladu existence takového algoritmu, avšak dojde se ke sporu s hodnotou jeho kompetitivního poměru. Pro snadné případné zobecnění bylo v místech, kde to není na úkor čitelnosti, ponecháno m místo dosazení jeho hodnoty.

Napřed se vezme kritická posloupnost, pro kterou σ_m-kompetitivní algoritmus pro libovolnou náhodnou posloupnost dospěje ke konfiguraci, která se v poměrech liší od průměrné konfigurace nejvýše o ε. S tímto předpokladem se „speciálně“ rozšíří opět na kritickou posloupnost tak, že je umístěna jedna maximální možná nadkritická úloha (dle definice 2.7.3) a m kritických úloh. Ukáže se, že žádný σ_m-kompetitivní on-line algoritmus nemůže na této posloupnosti umět sestrojit rozvrh lepší než $\frac{12969}{9125}$-kompetitivní. Ovšem to je spor s předpokladem, že zkoumaný algoritmus je σ_m-kompetitivní.

3.3.2 Konstrukce „dosvědčující“ posloupnosti

Buď $\varepsilon > 0$, podle tvrzení 3.2.9 existuje kritická posloupnost $P \in \mathcal{P}_K$, že platí

$$(\forall i \in [m-1])(\forall \mathcal{B} \in \mathcal{R}_\mathcal{A}) : \left| \frac{Load_{(\mathcal{A},P,\mathcal{B})}(i)}{sum(P)} - \frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j \in [m]} \left(\frac{m}{m-1}\right)^{j-1}} \right| < \varepsilon \quad (3.65)$$

a navíc ji lze rozšířit na kritickou posloupnost $Q \in \mathcal{P}_K$, jež dosvědčí hledaný spor. V posloupnosti Q je za prefixem P jedna nadkritická úloha velikosti $\frac{sum(P)}{m-2}$ následována m kritickými úlohami. Formálně je to zapsáno takto:

$$Q^{-m-2} \stackrel{def}{=} P \quad (3.66)$$

$$Q_{-m-1} \stackrel{def}{=} \frac{sum(P)}{m-2} \quad (3.67)$$

$$Crit(Q, m) \quad (3.68)$$

Poznámka: *Posloupnost Q s uvedenými vlastnostmi existuje. Podle tvrzení* 2.7.12 *existuje kritická posloupnost mající alespoň $|Q|$ kritických úloh navíc splňující, že $(m-2)$ dělí počet jejích jednotkových úloh. Existence Q potom plyne z nesoudělnosti $(m-2)$ a $(m-1)$.*

Z definice 2.7.13 posloupnost Q zkrácená až o $m-1$ posledních úloh zůstane slabě kritická, tudíž podle tvrzení 2.8.2 je známa délka optimálního off-line rozvrhu, tedy

$$(\forall i \in [m]) : \; Q^{-i} \in \mathcal{P}_S \wedge OPT(Q^{-i}) = Q_{-i} \quad (3.69)$$

3.3.3 Odvození omezujících podmínek

Pro vyřešení úlohy bude sestrojen lineární program, proto je třeba odvodit nějaké platné vztahy, které potom budou použity jako omezující podmínky v lineárním programu. Proměnné lineárního programu budou odpovídat neznámým zde uvedeným.

Pro každý možný předpis umístění posledních $m+1$ úloh posloupnosti Q bude uvážena pravděpodobnost jeho použití algoritmem $\mathcal{A}$, tyto pravděpodobnosti jsou $\left(\forall c \in [m]^{m+1}\right): \ p_c \in \mathbb{R}$, formálně splňují

$$p_c = Pr_{\mathcal{B} \in \mathcal{R}_{\mathcal{A}}}\left[(\forall i \in [m+1]): \ \mathcal{A}(Q, \mathcal{B}, |P|+i) = c_i\right] \tag{3.70}$$

Je zřejmé, že p_c jsou nezáporné (protože jsou to pravděpodobnosti). Dohromady dávají úplný jev, tedy platí

$$\sum_{c \in [m]^{m+1}} p_c = 1 \tag{3.71}$$

Dále bude ještě proměnná $\varrho \in \mathbb{R}$, nezáporná, dolní odhad kompetitivního poměru algoritmu $\mathcal{A}$, má-li na vstupu posloupnost Q.

Pro posloupnosti $(\forall i \in [m+1]): \ Q^{|P|+j}$ budou pravděpodobnosti pro umístění právě prvních i z nových úloh vyjádřeny z definice 2.4.3 a definice úplného jevu takto

$$(\forall i \in [m+1])\left(\forall d \in [m]^i\right): \ p_d = \sum_{c \in d \times [m]^{m+1-i}} p_c \tag{3.72}$$

Na základě definice 2.8.3 vzhledem k předpokládanému dolnímu odhadu ϱ kompetitivního poměru se sestrojí následující nerovnosti

$$(\forall i \in [m+1]): \ \mathbb{E}\mathit{makespan}_{\mathcal{A}}(Q^{|P|+i}) \le \varrho \cdot \mathit{OPT}(Q^{|P|+i}) \tag{3.73}$$

Podle tvrzení 2.8.2 a $Q \in \mathcal{P}_S$ platí

$$\mathit{OPT}(Q^{|P|+i}) = Q_{|P|+i} \tag{3.74}$$

a podle definice 2.5.8 a 2.5.5 pro $i \in [m+1]$ platí

$$\mathbb{E}makespan_{\mathcal{A}}(Q^{|P|+i}) \geq \sum_{d \in [m]^i} p_d \cdot \max_{j \in [m]} \left(L_j + \sum_{k \in [m+1] \wedge \mathcal{A}(Q,\mathcal{B},|P|+k)=j} Q_{|P|+k} \right) \tag{3.75}$$

kde symbol $L_j \in \mathbb{R}$ je dolní odhad pro hodnoty konfigurace, do které se algoritmus $\mathcal{A}$ dostane při vstupní posloupnosti P, buď tedy

$$(\forall j \in [m])(\forall \mathcal{B} \in \mathcal{R}_{\mathcal{A}}) : \; L_j \leq Load_{(\mathcal{A},P,\mathcal{B})}(j) \tag{3.76}$$

jež mohou být zvolená např. jako infimum.

3.3.4 Konstrukce lineárního programu

Problém řeší následující lineární program, (minimalizační) ohodnocující funkce je

$$x_{\varrho} \tag{3.77}$$

Všechny proměnné jsou nezáporné.

$$\left(\forall [m]^{m+1}\right) : \; x_{p_c} \geq 0 \tag{3.78}$$

$$(\forall i \in [m]) : \; x_{pom_i} \geq 0$$

$$x_{\varrho} \geq 0$$

Rovnice podle vztahu 3.71

$$\sum_{c \in [m]^{m+1}} x_{p_c} = 1 \tag{3.79}$$

Poslední podmínky jsou pro $i \in [m]$ tyto

$$\sum_{d \in [m]^i} \sum_{c \in d \times [m]^{m+1-i}} x_{p_c} \cdot \max_{j \in [m]} \left(\frac{mag \cdot L_j}{sum(P)} + \sum_{k \in [m+1] \wedge \mathcal{A}(Q,\mathcal{B},|P|+k)=j} \frac{mag \cdot Q_{|P|+k}}{sum(P)} \right) + \tag{3.80}$$

$$+x_{pom_i} - x_\varrho \cdot mag \cdot Q_{|P|+i} = 0$$

získané ze vztahu 3.75 dosazením za proměnné dle vztahu 3.72 (místo neznámých se použily odpovídající proměnné), dále je provedena „normalizace“ takovýchto nerovnic podělením číslem $sum(P) > 0$ a vynásobením číslem mag (pro $m = 3$ je $mag = 76$, zaručuje celočíselnost, obecně může být volena takto $mag = (m^m - (m-1)^m)(m-2)(m-1)^{(m-1)}$). Proměnné x_{pom_i} zajistí převod z nerovnic na rovnice, tedy x_{pom_i} dorovná i-tou nerovnici na rovnost. Proměnná x_ϱ je převedena na levou stranu rovnice.

3.3.5 Nezávislost na posloupnosti

Bude-li L_i voleno jako infimum, tedy

$$(\forall i \in [m]) : \ L_i \stackrel{def}{=} \inf_{\mathcal{B} \in \mathcal{R}_\mathcal{A}} Load_{(\mathcal{A},P,\mathcal{B})}(i) \tag{3.81}$$

potom podle definice infima. Navíc dosazením L_i do tvrzení 3.2.8 se získá

$$(\forall i \in [m]) : \left| \frac{L_i}{sum(P)} - \frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j\in[m]} \left(\frac{m}{m-1}\right)^{j-1}} \right| \leq \varepsilon \tag{3.82}$$

Z tohoto vztahu vyplývá, že koeficienty u proměnných x_{p_c} lineárního programu a lineárního programu získaného z něj nahrazením všech $\frac{L_i}{sum(P)}$ za $\frac{\left(\frac{m}{m-1}\right)^{i-1}}{\sum_{j\in[m]} \left(\frac{m}{m-1}\right)^{j-1}}$ se budou lišit nejvýše o $\varepsilon \cdot mag$. Velmi významou výhodou této úpravy je zbavení se závislosti lineárního programu na posloupnosti P, náhodné posloupnosti $\mathcal{B}$ i na výpočtu algoritmu $\mathcal{A}$.

3.3.6 Kritéria redukce počtu proměnných

Lineární program byl řešen v uvedeném tvaru. Ovšem ukázalo se, že vzhledem k jeho velikosti co do počtu proměnných bude v zájmu čitelnosti, přehlednosti a ověřitelnosti důkazu, ale i v zájmu „upočitatelnosti" při případném zobecnění, vhodnější pokusit se nalézt metody, jak počet těchto proměnných snížit. Základní fakt, o který se to opírá, je ten, že předpoklad nulovosti některých proměnných nezvýší (neovlivní) hodnotu optima. Takové proměnné budou nyní hledány.

Poslední úloha (tedy Q_{-1}) může být vždy umístěna na počítač s minimální zátěží. Provede-li algoritmus umístění této úlohy podle tohoto pravidla, tak rozvrh nepokazí (tj. neprodlouží). Proto nechť pro předpisy umísťování nesplňující toto pravidlo platí

$$(\forall d \in [m]^m \times ([m] - \{1\})) : \ x_{p_d} = 0 \tag{3.83}$$

Úvahu lze zobecnit pro i-tou úlohu Q_{-i}, má smysl uvažovat jen umístění na některý z prvních i počítačů sežazených podle zátěže, formálně nechť platí

$$(\forall i \in [m]) \, (\forall d \in [m]^m \times ([m] - [i])) : \ x_{p_d} = 0 \tag{3.84}$$

Poznámka: *Uvedené kritérium umožnilo pro $m = 3$ počítače snížit počet proměnných z $81 + 3 + 1 = 84$ na $18 + 3 + 1 = 23$ ($+3$ jsou pomocné proměnné x_{pom_i} a $+1$ je proměnná x_ϱ).*

Druhé kritérium využívá speciální vlastnosti této posloupnosti, kterou je rovnost $Q_{-m-1} = Q_{-m}$ nadkritické a další kritické úlohy. Na $m = 3$ počítačích je 6 možností (kombinací) jejich rozmístění, místo normálních 9. Lze předpokládat, že zátěž počítače s první z nich je nižší než s druhou z nich. Pokud by tomu tak nebylo, tak lze pořadí jejich umístění přehodit, délka rozvrhu se leda může zkrátit. Další výpočet algoritmu $\mathcal{A}$ tím není ovlivněn.

Poznámka: *Pro $m = 3$ se tímto kritériem dále sníží počet proměnných na $12 + 3 + 1 = 16$.*

Lineární program zjednodušený podle těchto kritérií nazveme „redukovaný.“ Aplikace těchto kritériích je zakomponována v počítačovém programu sestavujícím lineární program, jeho zdrojový kód bude později uveden. Byly nalezeny i další metody snížení počtu proměnných, budou zmíněny později.

3.3.7 Generování redukovaného lineárního programu

Nyní bude řešen tento „redukovaný“ lineární program z něhož bude následně odvozeno řešení původního lineárního programu. Pro nalezení jeho optimálního řešení byla použita simplexová metoda.

Vzhledem k tomu, že žádný program neposkytoval řešení s absolutní přesností (v racionálních číslech) nebo tuto soustavu nebyl schopen vyřešit, bylo nutné napsat vlastní počítačový program. Duální řešení bylo s pomocí takto získaných výsledků dopočítáno s použitím programu Maple. Budiž připomenuto, že podle tvrzení 2.9.1 není třeba ověřovat korektnost žádného z těchto programů.

Pro eliminování numerických chyb při ručních výpočtech, vyšší flexibilitu při zkoumání vlastností různých variant vstupních posloupností a pro leckoho i méně pracné ověřování správnosti koeficientů lineárního programu byl sestaven počítačový program v jazyku **C**, který lineární program generuje. Zdrojový kód programu je zde přiložen, také je uveden v příloze.

```
/* Diplomova prace, 2001, Tomas Tichy
 * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * overeni/odhad-m3/glp.c
 *
 * Program je soucasti Zakladniho dukazu neexistence
 * 27/19-kompetitivniho pravdepodobnostniho on-line
 * algoritmu pro m=3 pocitace.
```

```
 *
 * Vygeneruje linearni program definovany v tomto
 * dukazu. Tj. umistuje posloupnost uloh 76, 76, 114,
 * 171 na limitni deterministickou konfiguraci 16, 24,
 * 36 a zdola odhaduje pravdepodobnostni kompetitivni
 * pomer.
 */

#include <stdio.h>

#define M 3 /* pocet pocitacu */
#define P 4 /* delka vstupni posloupnosti */

int z[M] = { 16, 24, 36 }; /* pocatecni konfigurace */
int p[P] = { 76, 76, 114, 171 }; /* vstupni posloupnost */

int c[P]; /* aktualni konfigurace */
int um[P]; /* umisteni uloh (cisla pocitacu) */


int generuj_vse(int h, int w, int koef, int *loady,
                int *jmena)
/* Umistuje (zbyle) ulohy "p[h], ..., p[P-1]"
 * Vysledna konfigurace ma v linearnim programu
 * prirazen koeficient "koef", nebot ukolem teto
 * funkce je roznasobeni sumy, viz. definice LP.
 * "loady[]" jsou zateze pocitacu, jejich nazvy jsou
 * dany polem "jmena[]"
 * "w == 0" nevypisuje nic, "w < 0" vypisuje
 * koeficienty LP, "w > 0" vypisuje nazvy promennych
 * Navratova hodnota: pocet vygenerovanych promennych
 * (=konfiguraci)
 */
{
  int i, j, sum=0;
  /* Zateze se seradi podle velikosti na zacatku, nebo
   * byla-li predchazejici uloha ruzna od aktualni. M
   * je konstanta, tak nema smysl pouzivat lepsi
   * tridici algoritmus.
   */
  if (h==0 || p[h]!=p[h-1])
```

```
  {
    for (i=0; i<M-1; i++)
     for (j=0; j<M-1; j++)
      if (loady[j] > loady[j+1])
      {
        int s;
        s=loady[j]; loady[j]=loady[j+1]; loady[j+1]=s;
s=jmena[j]; jmena[j]=jmena[j+1]; jmena[j+1]=s;
      }
  }

  if (h<P)
  {
    /* Zbyva-li jeste nejaka neumistena uloha, pak ji
     * zkusi umistit.
     * Nasledujici tri radky obsahuji tzv. kriteria
     * redukce poctu promennych
     */
    int max = M;
    if (max > P-h) max=P-h;
    if (h>0 && p[h]==p[h-1] && max>c[h-1]+1)
       max=c[h-1]+1;

    for (i=0; i<max; i++)
    {
      /* uloha p[h] bude umistena na i-ty minimalni
       * pocitac, zateze a jmena pocitacu museji byt
       * pro rekurzivni duplikovany.
       */
      int loady_nove[M], jmena_nove[M];
      for (j=0; j<M; j++) { loady_nove[j]=loady[j];
                            jmena_nove[j]=jmena[j]; }
      /* umisteni ulohy p[h]: */
       /* h-ta uloha je umistena na i-ty minimalni */
      c[h]=i;
       /* jak puvodne se tento pocitac jmenoval */
      um[h]=jmena[i];
       /* zvyseni zateze prislusneho pocitace */
      loady_nove[i]+=p[h];
      /* rekurzivni dovygenerovani, pricti pocet
       * vygenerovanych koeficientu
```

```
       */
      sum+=generuj_vse(h+1, w, koef, loady_nove,
                       jmena_nove);
    }
    return sum;
  } else
  {
    /* vypsani koeficientu nebo nazvu promenne */
    if (w>0) printf("%d ", koef); else
    if (w<0)
    {
    printf("p_");
    for (j=0; j<P; j++) printf("%d", um[j]);
    printf(" ");
    }
    return 1; /* toto je jedna promenna */
  }
}

void generuj_mks(int h, int d, int *loady, int *jmena)
/* Umistuje (zbyle) ulohy "p[h], ..., p[d-1]"
 * tj. do predem dane hloubky d, tj. podle toho, ktera
 * rovnice LP to ma byt. V hloubce d je prirazen
 * koeficient odpovidajici delce rozvrhu v ziskane
 * konfiguraci. Funkce generuj_vse() vygeneruje sumu
 * promennych, z nichz se tato pseudopromenna sklada,
 * viz. odvozeni LP, je jim prirazen zmineny
 * koeficient.
 * "loady[]" jsou zateze pocitacu, jejich nazvy jsou
 * dany polem "jmena[]"
 */
{
  int i, j;
  /* Zateze se seradi podle velikosti na zacatku, nebo
   * byla-li predchazejici uloha ruzna od aktualni. M
   * je konstanta, tak nema smysl pouzivat lepsi
   * tridici algoritmus.
   */
  if (h==0 || p[h]!=p[h-1])
  {
    for (i=0; i<M-1; i++)
```

```
     for (j=0; j<M-1; j++)
      if (loady[j] > loady[j+1])
      {
        int s;
        s=loady[j]; loady[j]=loady[j+1]; loady[j+1]=s;
s=jmena[j]; jmena[j]=jmena[j+1]; jmena[j+1]=s;
      }
  }

  if (h<d)
  {
    /* Zbyva-li jeste nejaka neumistena uloha, pak ji
     * zkusi umistit.
     * Nasledujici tri radky obsahuji tzv. kriteria
     * redukce poctu promennych
     */
    int max = M;
    if (max > P-h) max=P-h;
    if (h>0 && p[h]==p[h-1] && max>c[h-1]+1)
       max=c[h-1]+1;

    for (i=0; i<max; i++)
    {
      /* uloha p[h] bude umistena na i-ty minimalni
       * pocitac, zateze a jmena pocitacu museji byt
       * pro rekurzivni duplikovany.
       */
      int loady_nove[M], jmena_nove[M];
      for (j=0; j<M; j++) { loady_nove[j]=loady[j];
                            jmena_nove[j]=jmena[j]; }
      /* umisteni ulohy p[h]: */
       /* h-ta uloha je umistena na i-ty minimalni */
      c[h]=i;
       /* jak puvodne se tento pocitac jmenoval */
      um[h]=jmena[i];
       /* zvyseni zateze prislusneho pocitace */
      loady_nove[i]+=p[h];
      /* rekurzivni dovygenerovani */
      generuj_mks(h+1, d, loady_nove, jmena_nove);
    }
  } else
```

```
  {
      /* vybere maximum zatezi, tj. delka rozvrhu po
       * umisteni uloh p[0],..,p[d-1], to je koeficient
       * linearniho programu, takze je u vsech
       * promennych s timto umisteni techto uloh
       */
      int max = loady[0];
      for (i=0; i<M; i++)
      if (max<loady[i]) max=loady[i];
      generuj_vse(h, 1, max, loady, jmena);
  }
}


int main()
/* hlavni funkce
 */
{
  int i, d, poc;
  int jmena[M] = { 1, 2, 3 };

  /* spocte pocet promennych linearniho programu,
   * promenne odpovidajici umistovani uloh jsou
   * generovane, P-1 je pomocnych pro prevedeni
   * na rovnice a 1 je pro odhad kompetitivniho pomeru
   * (rho).
   */
  poc=generuj_vse(0, 0, 1, z, jmena)+P-1+1;

  /* vystupem je popis linearniho programu,
   * vypise: <pocet rovnic> <pocet promennych>
   * jedna rovnice je trivialni a P-1 je generovanych,
   * celkem P
   */
  printf("%d %d\n\n", 1+P-1, poc);

  /* vypise ucelovou funkci, pro odhad kompetitivniho
   * pomeru (promenna rho) je jedna, jinak nula
   */
  for (i=0; i<poc-1; i++) printf("0 ");
  printf("1\n\n");
```

```
  /* vypise trivialni rovnici, dle definice uplneho
   * jevu, viz. definice LP, vsechny promenne
   * odpovidajici umistovani uloh maji koeficient 1,
   * ostatni maji 0, suma celkem ma byt 1.
   */
  generuj_vse(0, 1, 1, z, jmena);
  for (i=0; i<P-1; i++) printf("0 ");
  printf("0 1\n");

  /* vypise zbyvajici rovnice */
  for (d=1; d<P; d++)
  {
    /* napred koeficienty promennych odpovidajicich
     * umistovani uloh, generovane pro hloubku d, viz.
     * definice LP
     */
    generuj_mks(0, d+1, z, jmena);
    /* koeficienty pomocnych promennych pro doplneni
     * nerovnice na rovnici
     */
    for (i=0; i<P-1; i++)
       printf("%d ", (i==d-1) ? 1 : 0);
    /* promenna odhadu kompetitivniho pomeru (rho),
     * delka optimalniho rozvrhu ma byt rovna velikosti
     * posledni ulohy (pro slabe kriticke posl.)
     * a prava strana je 0
     */
    printf("%d 0\n", -p[d]);
  }
  printf("\n");
  /* vypise nazvy pouzitych promennych */
  generuj_vse(0, -1, 1, z, jmena);
  for (i=0; i<P-1; i++) printf("pom_%d ", i+1);
  printf("rho\n");

  return 0;
}
```

3.3.8 Řešení simplexovým algoritmem

Výstupem programu jsou p počet rovnic, n počet proměnných, dalších n čísel je účelová funkce lineárního programu. Účelová funkce je 1 právě pro proměnnou x_ϱ, jejíž hodnota se minimalizuje. Následuje p rovnic ve tvaru n koeficientů pro jednotlivé proměnné následované číslem, jež je příslušnou pravou stranou. Napřed jsou uvedeny koeficienty pro proměnné x_{p_c}, následují pomocné proměnné x_{pom_i} a poslední proměnná je x_ϱ. Proměnné budou v tomto pořadí kódovány ve vektoru $x \in \mathbb{R}^n$. Koeficienty jsou v matici $p \times n : A \in \mathbb{R}^{n \cdot p}$, pravá strana ve vektoru $b \in \mathbb{R}^p$ a účelová funkce ve vektoru $c \in \mathbb{R}^n$. Jejich hodnoty jsou tyto:

$$A = \begin{array}{c} \begin{array}{cccccccccccccccc} p_{1123} & p_{1132} & p_{2131} & p_{2113} & p_{2213} & p_{2231} & p_{3121} & p_{3112} & p_{3212} & p_{3221} & p_{3312} & p_{3321} & pom_1 & pom_2 & pom_3 & \varrho \end{array} \\ \left[\begin{array}{cccccccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 168 & 168 & 100 & 100 & 176 & 176 & 112 & 112 & 112 & 112 & 188 & 188 & 1 & 0 & 0 & -76 \\ 168 & 168 & 150 & 206 & 176 & 176 & 138 & 206 & 130 & 214 & 188 & 188 & 0 & 1 & 0 & -114 \\ 207 & 195 & 263 & 207 & 207 & 187 & 263 & 206 & 271 & 214 & 195 & 188 & 0 & 0 & 1 & -171 \end{array} \right] \end{array} \tag{3.85}$$

$$b = [1, 0, 0, 0] \tag{3.86}$$

$$c = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] \tag{3.87}$$

3.3.9 Další redukce počtu proměnných

V souvislosti s „jednoduchou“ strukturou této matice A bylo nalezeno další univerzální kritérium, umožnující odstranit některé další proměnné. Řešitelnost ani hodnotu optima neovlivní odstranění proměnné p_c (=sloupečku matice), jejíž sloupeček majorizuje sloupeček jiné proměnné p_d. „Majorizující “ proměnná p_c může být fixována na 0, „majorizovaná“ proměnná p_d může být zvýšena o původní hodnotu „majorizující “ proměnné p_c, první rovnice zůstane splněna, u ostatních stačí odpovídajícím způsobem zvýšit pomocné proměnné pom_i, aby zůstaly rovnicemi.

Poznámka: *Uvedenou úvahu lze dále zobecnit a odstranit proměnné, jejichž sloupečky majorizují nějakou konvexní kombinaci zbylých sloupečků odpovídajících proměnným p_c. Z praktického hlediska to není efektivní, neboť hledání takových proměnných je výpočetně náročné tak jako řešení tohoto lineárního programu.*

Pro tuto konkrétní matici lze podle tohoto pravidla fixovat na 0 následující proměnné $p_{1123}, p_{2213}, p_{3221}, p_{3312}$ a p_{3321} (ve stejném pořadí majorizují proměnné $p_{1132}, p_{2231}, p_{3112}, p_{2231}$ a p_{2231}). Bude proto uvážen tento zjednodušený lineární program

Vyřešení lineárního programu

$$A = \begin{array}{c} \begin{array}{ccccccccccc} p_{1132} & p_{2131} & p_{2113} & p_{2231} & p_{3121} & p_{3112} & p_{3212} & pom_1 & pom_2 & pom_3 & \varrho \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 168 & 100 & 100 & 176 & 112 & 112 & 112 & 1 & 0 & 0 & -76 \\ 168 & 150 & 206 & 176 & 138 & 206 & 130 & 0 & 1 & 0 & -114 \\ 195 & 263 & 207 & 187 & 263 & 206 & 271 & 0 & 0 & 1 & -171 \end{bmatrix} \end{array} \tag{3.88}$$

$$c = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] \tag{3.89}$$

$$b = [1, 0, 0, 0] \tag{3.90}$$

Poznamenejme, že tento lineární program byl řešen a jeho optimum má hodnotu $\frac{12969}{9125}$. Následuje vektor primárních proměnných $x \in \mathbb{R}^n$ a vektor duálních proměnných $y \in \mathbb{R}^p$, což jsou odpovídající optimální řešení. Dále následuje ověření přípustnosti duálního řešení.

$$x = [\frac{38}{9125}, 0, \frac{3207}{9125}, 0, \frac{1176}{1825}, 0, 0, 0, 0, 0, \frac{12969}{9125}] \tag{3.91}$$

$$y = [\frac{12969}{9125}, -\frac{368}{173375}, -\frac{26}{9125}, -\frac{521}{173375}] \tag{3.92}$$

$$A^\top y = [0, -\frac{1512}{173375}, 0, -\frac{2728}{173375}, 0, -\frac{41}{1825}, -\frac{216}{173375}, -\frac{368}{173375}, -\frac{26}{9125}, -\frac{521}{173375}, 1] \tag{3.93}$$

Budiž pro korektnost poznamenáno, že všechny vektory zde uvedené jsou chápány jako sloupce a takto jsou zde uvedeny jen pro úsporu místa.

Přípustnost primárního řešení x se ověří vynásobením.

Přípustnost duálního řešení se snadno ověří porovnáním výsledku $A^\top y$ s vektorem c, neboť tento má všechny složky až na jednu ≤ 0, odpovídající složka v c je 0, jediná výjimka je poslední složka, kdy v tomto i vektoru c je 1, tedy y je přípustné řešení.

Dále lze snadno spočítat hodnoty jim odpovídajících účelových funkcí

$$c^\top \cdot x = \frac{12969}{9125} \tag{3.94}$$

$$b^\top \cdot y = \frac{12969}{9125} \tag{3.95}$$

Hodnoty účelových funkcí primární a duální úlohy se rovnají. Podle tvrzení 2.9.1 je hledané optimum pro řešený lineární program $\frac{12969}{9125}$.

3.3.10 Návrat k původnímu zadání

Na základě tohoto řešení bude odvozeno přípusné „téměř optimální" řešení pro duální program původního zadaní před zaokrouhlením podle 3.76. Nechť

$$\tilde{y} \stackrel{def}{=} y - [m \cdot \varepsilon \cdot w, 0, 0, 0] \tag{3.96}$$

Konstanta w je volena jako $w \stackrel{def}{=} \mathit{mag} \cdot \left(1 + (m-1) \cdot \left(\left(\frac{m}{m-1}\right)^m - 1\right)\right)$, což je součet úloh posloupnosti po normalizaci, tedy maximální možný koeficient v maticích A a $\tilde{A}$.

Je nutné ověřit přípustnost $\tilde{y}$, tedy zda platí $A^\top \tilde{y} \leq c$. Matice A a původní $\tilde{A}$ se liší jen v koeficientech odpovídajících proměnným x_{p_c} a to nejvýše o $\varepsilon \cdot w$. Proměnné x_{p_c} odpovídají právě sloupečkům matice, které mají v prvním řádku hodnotu 1, ostatní sloupečky mají 0.

Výsledek součinu $\tilde{A}^\top y$ oproti $A^\top y$ je ve všech složkách větší nejvýše o $m \cdot \varepsilon \cdot w$. Tento rozdíl je kompenzován volbou $\tilde{y}$, to je tedy přípustné duální řešení původního zadání.

Hodnota účelové funkce pro toto duální řešení je $\frac{12969}{9125} - 1311 \cdot \varepsilon$, neboť $1311 = m \cdot w$ pro $m = 3$.

Podobně lze sestrojit i přípustné „téměř optimální" řešení pro primární úlohu původního zadání. Ovšem pro použití tvrzení 2.9.2 stačí duální řešení, které bylo spočítáno, viz. formule 3.96 a omezenost primárního řešení. Účelová funkce c má jedinou nenulovou složku, z její nezápornosti plyne omezení primární úlohy nulou. Tudíž optimální řešení tohoto lineárního programu je alespoň $\frac{12969}{9125} - 1311 \cdot \varepsilon$.

Tímto bylo ukázáno, že kompetitivní poměr algoritmu $\mathcal{A}$ je alespoň $\frac{12969}{9125} - 1311 \cdot \varepsilon$. Současně se však předpokládá, že kompetitivní poměr je nejvýše $\sigma_m = \frac{27}{19}$. Mělo by tedy být $\frac{12969}{9125} - 1311 \cdot \varepsilon \leq \frac{27}{19}$. Zřejmě $\frac{12969}{9125} > \frac{27}{19}$, tudíž pro spor stačí zvolit ε dostatečně malé. Ať je $\varepsilon < \frac{\frac{12969}{9125} - \frac{27}{19}}{1311}$, čímž je nalezen spor.

3.4 Silnější verze důkazu

Verze důkazu tvrzení 3.1.1 uvedená v předchozí kapitole je sice pro $m = 3$ počítače dostačující, avšak podařilo se nalézt zesílení důkazu, které však znamená zkomplikování důkazu. Je uvedeno proto, že by mohlo pomoci nalézt důkaz pro vyšší počet počítačů.

Základní idea důkazu zůstává stejná, změní se však soustava omezujících podmínek (rovnic a nerovnic). Opět bude zdola omezována hodnota kompetitivního poměru a hledána minimální hodnota vyhovující těmto požadavkům.

3.4.1 Obecnější posloupnost

Při příležitosti posilování důkazu byla zvolena i obecnější posloupnost. Speciální kritická posloupnost zvolená v tomto důkazu pro konstrukci lineárního programu má na svém konci $t \in \mathbb{N}$ nadkritických úloh následovaných m kritickými úlohami.

Poznámka: *Nadkritickými úlohami jsou zde míněny maximální takové.*

Nechť taková posloupnost je označena symbolem Q. Nechť posloupnost P je podposloupnost Q bez posledních $t+m$ úloh, tedy $P = Q^{-t-m-1}$.

3.4.2 Nové pomocné definice a tvrzení

Definice 3.4.1 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $q \in [t+m]$, $c \in [m]^q$ *řekneme, že algoritmus* $\mathcal{A}$ *umísťuje* P *podle předpisu* c*, pokud platí*

$$(\forall j \in [q])(\forall \mathcal{B} \in \mathcal{R}):\ \mathcal{A}(P, \mathcal{B}, -t-m+i-1) = [c]_i^q \tag{3.97}$$

Prvních q z $t+m$ posledních úloh umístí algoritmus $\mathcal{A}$ na počítače podle předpisu c (nezávisle na náhodné posloupnosti $\mathcal{B}$).

Definice 3.4.2 Pro $m \in \mathbb{N}, m > 1$, $t \in \mathbb{N}$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $|P| \geq t+m$ $q \in [t+m]$, $c \in [m]^q$, $\mathcal{B} \in \mathcal{R}$ *definujme symbol* $\mathit{ccfg}_{(\mathcal{A},\mathcal{B})}(P,c)$ *pro* $i \in [q]$ *předpisem*

$$\left[\mathit{ccfg}_{(\mathcal{A},\mathcal{B})}(P,c)\right]_i^m \stackrel{def}{=} \mathit{cfg}_{\mathcal{A}}(P^{-t-m-1}, \mathcal{B}) + \sum_{(j\in[q])\wedge([c]_j^q=i)} P_{-t-m+j-1} \tag{3.98}$$

Tento symbol označuje konfiguraci získanou umístěním úloh posloupnosti P^{-t-m-1} rozhodováním algoritmu $\mathcal{A}$ při náhodné posloupnosti $\mathcal{B}$ a prvních q úloh z $t+m$ posledních úloh posloupnosti P podle předpisu c.

Poznámka: *Snadno lze ověřit konzistentnost této definice s definicemi* 2.4.3 *a* 2.4.4.

Tvrzení 3.4.3 Pro $m \in \mathbb{N}, m > 1$, $P \in \mathcal{P}$, $|P| \geq t + m$, $q \in [t + m]$, $c \in [m]^q$, *algoritmus $\mathcal{A}$ umísťující úlohy P podle předpisu c platí*

$$(\forall \mathcal{B} \in \mathcal{R}) : \ \mathit{cfg}_{\mathcal{A}}(P, \mathcal{B}) = \mathit{ccfg}_{(\mathcal{A},\mathcal{B})}(P, c) \tag{3.99}$$

a také

$$(\forall i \in [m])(\forall \mathcal{B} \in \mathcal{R}) : \ \mathit{ord}_{\mathit{ccfg}_{(\mathcal{A},\mathcal{B})}(P,c)}(i) = \mathit{Load}_{(\mathcal{A},P,\mathcal{B})}(i) \tag{3.100}$$

Vyjadřuje získanou konfiguraci v závislosti na použitém předpisu c. Druhá formule navíc vyjadřuje vztah k zátěži počítačů.

Důkaz. Snadný z definic 3.4.2, 3.4.1, 2.4.4 a 2.4.3.

Formule 3.100 je podle definice 2.5.3 důsledkem formule 3.99. □

3.4.3 Snížení počtu proměnných

Napřed se dokáže pomocné tvrzení, které je vlastně kritériem pro významné snížení počtu proměnných v lineárním programu.

Tvrzení 3.4.4 Pro $m \in \mathbb{N}, m > 1$, *σ_m-kompetitivní algoritmus $\mathcal{A}$, $\mathcal{B} \in \mathcal{R}_{\mathcal{A}}$, $Q \in \mathcal{P}_K$, $i \in [m]$, $P \in \mathcal{P}$, $P = Q^{-i}$ platí*

$$\mathit{makespan}_{\mathcal{A}}(P, \mathcal{B}) = \mathit{Load}_{(\mathcal{A},Q,\mathcal{B})}(-i) \tag{3.101}$$

Zátěž i-tého maximálního počítače je rovna délce rozvrhu po umístění od konce i-té (kritické) úlohy.

Poznámka: *Motivace tvrzení je v tom, že většinou (až na nějaké singularity při rovnostech zátěží) každá z těchto kritických úloh je umístěna na jiný počítač.*

Důkaz. Podle tvrzení 2.8.7 pro $\mathcal{B} \in \mathcal{R}_{\mathcal{A}}$ platí

$$makespan_{\mathcal{A}}(P, \mathcal{B}) \geq Load_{(\mathcal{A},Q,\mathcal{B})}(-i) \tag{3.102}$$

Uvedená nerovnost musí platit i v průměru přes $\mathcal{R}_{\mathcal{A}}$ či $\mathcal{R}$. Je-li pro nějaké $\mathcal{B} \in \mathcal{R}_{\mathcal{A}}$ nerovnost ostrá, potom (jak již bylo dříve ukázáno, např. v důkazu tvrzení 3.2.5) bude ostrá nerovnost zachována i v průměru.

Pro spor budiž předpokládáno, že platí

$$\mathbb{E}makespan_{\mathcal{A}}(P) > \mathbb{E}load_{(\mathcal{A},Q)}(-i) \tag{3.103}$$

Obě tyto strany budou odhadovány pomocí neostrých nerovností. Dospěje se k omezení stejným číslem, což bude spor s ostrou nerovností.

První odhad podle předpokládaného kompetitivního poměru algoritmu $\mathcal{A}$, dosazeno podle tvrzení 2.8.2, dosazeno z definice P a definice 2.7.6.

$$\mathbb{E}makespan_{\mathcal{A}}(P) \leq \sigma_m \cdot OPT(P) = \sigma_m \cdot P_{-1} = \sigma_m \cdot Q_{-i} = \tag{3.104}$$

Další úpravou je již jen dosazení podle tvrzení 2.7.10.

$$= \sigma_m \cdot \left(\frac{m}{m-1}\right)^{m-i} \frac{sum(Q^{-m-1})}{m-1} \tag{3.105}$$

Podobně se odhadne podle tvrzení 3.2.2 takto

$$\mathbb{E}load_{(\mathcal{A},Q)}(-i) = \sigma_m \cdot \left(\frac{m}{m-1}\right)^{m-i} \frac{sum(Q^{-m-1})}{m-1} \tag{3.106}$$

Ovšem horní odhad je roven dolnímu (viz. formule 3.105 a 3.106), což si vynucuje rovnost místo ostré nerovnosti ve vztahu 3.103, to je spor. Podle uvedeného platí

$$\mathbb{E}makespan_{\mathcal{A}}(P) = \mathbb{E}load_{(\mathcal{A},Q)}(-i) \tag{3.107}$$

a navíc podle nerovnosti 3.102, rovnosti 3.107 a definice 2.4.5 také platí

$$\mathit{makespan}_{\mathcal{A}}(P,\mathcal{B}) = \mathit{Load}_{(\mathcal{A},Q,\mathcal{B})}(-i) \tag{3.108}$$

což bylo cílem důkazu. □

Každá proměnná přísluší nějakému předpisu rozmístění úloh, ovšem podle právě uvedeného tvrzení jsou tyto možnosti umísťování σ_m-kompetitivním algoritmem $\mathcal{A}$ značně omezené. Řada předpisů umístění je nepřípustná, tudíž pravděpodobnost, že je algoritmus použije je nulová. Jim odpovídající proměnné mohou být z lineárního programu vyloučeny. Tvrzení umožní velmi významnou redukci počtu proměnných v lineárním programu, který bude sestaven.

Toto kritérium není možno použít spolu s „sofistikovanými" kritérii uvedenými v předchozí kapitole, neboť se navzájem vylučují.

Poznámka: *Jediné kritérium z předchozí kapitoly nezávislé na chování algoritmu, ale jen na lineárním programu je tzv. „majorizování sloupců." Ani toto kritérium není použitelné, protože v soustavě se budou vyskytovat rovnice. Touto redukcí by se z rovnic staly nerovnice, což by vyžadovalo přidání m nových pomocných proměnných.*

Definice 3.4.5 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}_K$, $c \in [m]^{t+m}$ *definujme predikát* $\mathit{Sr}_{\mathcal{A}}(P,c)$ *formulí*

$$(\forall j \in [m])\Big(\forall e \in [m]^j, c \in e \times [m]^{t+m-j}\Big)(\forall \mathcal{B} \in \mathcal{R}) : \tag{3.109}$$

$$\max\Big(\mathit{ccfg}_{(\mathcal{A},\mathcal{B})}(P,e)\Big) = \mathit{ord}_{\mathit{ccfg}_{(\mathcal{A},\mathcal{B})}(P,c)}(j)$$

Tento predikát rozpoznává, zda předpis c rozmístění úloh posloupnosti P je vyhovující tvrzení 3.4.4. Korektnost definice lze ověřit z definice 3.4.2.

3.4.4 Limitní přechod

Provést precizní matematický výpočet s odhady pomocí malého ε je možné, avšak technicky komplikovanější. Takovýto výpočet lze realizovat obdobně, jako tomu je v kapitole 3.3.

Poznámka: *Hodnota ε klesá vzhledem k délce úseku kritických úloh před posledními $t{+}m$ úlohami v posloupnosti. Taková libovolně dostatečně dlouhá posloupnost P existuje podle tvrzení* 2.7.12, *tedy existuje i P rozšířená o t nadkritických a m kritických úloh.*

Vzhledem ke zmíněným komplikacím se ukázalo výhodnější použít v úvaze **limitní přechod**. Je tedy možné dokazovat odhad pro nekonečnou posloupnost, k níž je možné se dostat limitou z konečných posloupností.

Budou odvozeny vztahy, které musí být splněny pro každou posloupnost a σ_m-kompetitivní algoritmus. Tudíž musí být splněny i v jejich limitní verzi.

Kvůli korektnosti je třeba provést normalizaci posloupností, aby bylo možné „rozumně" manipulovat s dlouhými a nekonečnými posloupnostmi. Normalizace bude dána požadavkem, aby celkový součet úloh v P byl roven 1.

3.4.5 Nově požadované rovnosti

Podle tvrzení 3.2.1 pro σ_m*-kompetitivní algoritmus* $\mathcal{A}$ na kritické posloupnosti je známa průměrná zátěž na i-tém minimálním počítači po rozvrhnutí všech úloh posloupnosti. Podle tvrzení 3.2.2 je možné hodnoty těchto zátěží spočítat. Pro každou zátěž se takto sestaví jedna rovnice, celkem se sestaví m rovnic.

Napřed označme pravděpodobnosti p_c analogicky jako v první variantě důkazu bylo označeno vztahem 3.70, nyní pro $q \in [t+m]$ a $c \in [m]^q$ jsou p_c vyjádřena takto:

$$p_c = Pr_{\mathcal{B} \in \mathcal{R}_\mathcal{A}} \left[(\forall i \in [q]) : \ \mathcal{A}(Q, \mathcal{B}, |P| + i) = c_i \right] \tag{3.110}$$

což jsou pravděpodobnosti, že algoritmus $\mathcal{A}$ použije pro umístění úloh předpis c.

Skutečná zátěž se po normalizaci liší nejvýše o $\varepsilon > 0$, které lze libovolně dále zmenšovat. Jako v kapitole 3.3 jej lze zvolit podle tvrzení 3.2.2, hodnoty normalizovaných zátěží se potom podle tvrzení 3.2.9 liší nejvýše o ε. Proto pro $j \in [m]$ a $\mathcal{B} \in \mathcal{R}$ proto platí následující vztahy

$$\sum_{c \in [m]^{t+j} \wedge Sr_\mathcal{A}(Q,c)} p_c \cdot \frac{ord_{ccfg_{(\mathcal{A},\mathcal{B})}(Q,c)}(i)}{sum(P)} \in \left\langle \frac{\mathbb{E}load_{(\mathcal{A},Q)}(i)}{sum(P)} - \varepsilon, \frac{\mathbb{E}load_{(\mathcal{A},Q)}(i)}{sum(P)} + \varepsilon \right\rangle \tag{3.111}$$

V limitě nastává rovnost, tedy pro limitní posloupnost Q platí

$$\sum_{c \in [m]^{t+j} \wedge Sr_\mathcal{A}(Q,c)} p_c \cdot \mathbb{E}_{\mathcal{B} \in \mathcal{R}} \left[\frac{ord_{ccfg_{(\mathcal{A},\mathcal{B})}(Q,c)}(i)}{sum(P)} \right] = \frac{\mathbb{E}load_{(\mathcal{A},Q)}(i)}{sum(P)} \tag{3.112}$$

3.4.6 Požadavky na kompetitivní poměr

Další nerovnice se sestrojí z definice 2.8.3, to je definice kompetitivního poměru. Dosazením se dostanou následující nerovnosti pro $j \in [t+m]$

$$\sum_{c \in [m]^{t+j} \wedge Sr_\mathcal{A}(Q,c)} p_c \cdot \mathbb{E}_{\mathcal{B} \in \mathcal{R}} \left[\frac{\max(ord_{ccfg_{(\mathcal{A},\mathcal{B})}(Q,c)})}{sum(P)} \right] \leq \frac{\sigma_m \cdot OPT(Q^{-t-m+j-1})}{sum(P)} - \varepsilon \tag{3.113}$$

Pro limitní Q bude nerovnost takováto

$$\sum_{c \in [m]^{t+j} \wedge Sr_\mathcal{A}(Q,c)} p_c \cdot \mathbb{E}_{\mathcal{B} \in \mathcal{R}} \left[\frac{\max(ord_{ccfg_{(\mathcal{A},\mathcal{B})}(Q,c)})}{sum(P)} \right] \leq \frac{\sigma_m \cdot OPT(Q^{-t-m+j-1})}{sum(P)} \tag{3.114}$$

3.4.7 Konstrukce lineárního programu

Lineární program je sestrojen pro limitní případ vztahů 3.114 a 3.112. Proměnné jsou značeny stejně jako dříve.

Konstruovaný lineární program je minimalizační, účelová funkce je

$$x_{\varrho} \tag{3.115}$$

Podle vztahu 3.114 a tvrzení 2.8.2 rovnice pro $j \in [m]$ jsou

$$\sum_{c \in [m]^{t+j} \wedge Sr_{\mathcal{A}}(Q,c)} x_{p_c} \cdot \max\left(\frac{ccfg_{(\mathcal{A},\mathcal{B})}(Q,c)}{sum(P)}\right) + x_{pom_j} - x_{\varrho} \cdot \frac{Q^{-t-m+j-1}}{sum(P)} = 0 \tag{3.116}$$

a dle vztahu 3.112 pro $j \in [m]$ ještě rovnice

$$\sum_{c \in [m]^{t+j} \wedge Sr_{\mathcal{A}}(Q,c)} \frac{x_{p_c}}{sum(P)} \cdot ord_{ccfg_{(\mathcal{A},\mathcal{B})}(Q,c)}(j) = \frac{\mathbb{E}load_{(\mathcal{A},Q)}(j)}{sum(P)} \tag{3.117}$$

V koeficientech je za posloupnost Q dosazena limitní posloupnost, tj. rovnice a nerovnice jsou použity v jejich limitě. Limita má vliv jen na koeficienty tohoto lineárního programu. Koeficienty, podle tvrzení 3.2.8, normování a limitního přechodu, jsou konstantní vzhledem k náhodné posloupnosti $\mathcal{B}$.

Pro konstrukci tohoto lineárního programu byl napsán počítačový program v programovacím jazyku **C**, jehož zdrojový kód je uveden v příloze této práce. Tento počítačový program vyžaduje na vstupu čísla m a t.

3.4.8 Koeficienty lineárního programu

Pro ozřejmění výrazů použitých při inicializacích v zmíněném počítačovém programu budou na následujících řádcích uvedeny jejich odvození. Můžeme předpokládat $t \leq m$, alespoň prozatím nebude voleno více nadkritických úloh než m.

Aby výsledný program měl celočíselné koeficienty, tak bude po normalizaci vynásoben výrazem $(m-1)^{2m-1-t}(m-2)^t$.

- Přibližné zátěže počítačů (L_i) po umístění P

  Vyjádřené podle 3.2.9, tj. poměry $1 : \left(\frac{m}{m-1}\right) : \dots : \left(\frac{m}{m-1}\right)^{m-1}$. Zátěž i-tého počítače po úpravě na celá čísla pro $i = 0, \dots, m-1$ je

$$m^i \cdot (m-1)^{2m-1-t-i} \cdot (m-2)^t \tag{3.118}$$

- Součet všech úloh P, tj. součet všech zátěží

  Snadno sečtením zátěží.

$$(m^m - (m-1)^m) \cdot (m-1)^{m-t} \cdot (m-2)^t \tag{3.119}$$

- t nadkritických úloh navazujících na P

  Jsou vyjádřeny podle definice 2.7.3 pro $i = 1, \dots, m$ takto

$$(m^m - (m-1)^m) \cdot (m-1)^{m-1-t+i} \cdot (m-2)^{t-i} \tag{3.120}$$

- m kritických úloh navazující na t kritických navázaných na P

  Jsou vyjádřeny podle definice 2.7.3 pro $i = 0, \dots, m-1$ takto

$$(m^m - (m-1)^m) \cdot (m-1)^{m-1-i} \cdot m^i \tag{3.121}$$

3.4.9 Řešení pro $m = 3$

Pro $m = 3$ bylo zvoleno $t = 1$, což znamená, že se zkoumaná posloupnost skládá z dlouhé kritické posloupnosti rozšířené o jednu nadkritickou úlohu následovanou m kritickými úlohami.

Ukázalo se, že takto zvolený lineární program nemá přípustné řešení, proto v důkazu je třeba použít tvrzení 2.9.3 k dokázání jeho neexistence. Je proto řešen upravený lineární program.

Poznámka: *Simplexový algoritmus potřebuje na začátku výpočtu znát libovolné přípustné řešení, které se dále optimalizuje. Proto takovouto úpravu provádí simplexový algoritmus vždy.*

Byl napsán počítačový program - filtr, který transformuje popis lineárního programu v popis lineáního programu majícího nulové optimum právě, když původní má přípustné řešení. Na vstup tomuto filtru je dán výstup počítačového programu generujícího definovaný lineární program.

Získaný lineární program v maticovém zápisu, který rozhoduje o přípustnosti lineárního programu vytvořeného v předchozí kapitolce, vypadá takto:

$$A = \begin{array}{c}
\begin{array}{ccccccccccccccccccc}
p_{1231} & p_{1213} & p_{1321} & p_{2131} & p_{2113} & p_{2312} & p_{3121} & p_{3212} & pom_1 & pom_2 & pom_3 & rho & q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6
\end{array} \\
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
100 & 100 & 112 & 100 & 100 & 112 & 112 & 112 & 1 & 0 & 0 & -76 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
150 & 206 & 138 & 150 & 206 & 130 & 138 & 130 & 0 & 1 & 0 & -114 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
263 & 207 & 263 & 263 & 207 & 271 & 263 & 271 & 0 & 0 & 1 & -171 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
100 & 100 & 112 & 100 & 100 & 112 & 112 & 112 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
150 & 206 & 138 & 150 & 206 & 130 & 138 & 130 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
263 & 207 & 263 & 263 & 207 & 271 & 263 & 271 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\end{array} \tag{3.122}$$

Proměnné v matici A označené jako q_i jsou přidané pro převedení lineárního programu na nový lineární program rozhodující řešitelnost původního.

Pravá strana soustavy vypadá takto:

$$b = [1, 0, 0, 0, 108, 162, 243] \tag{3.123}$$

Odhodnocující funkce je opět dána převodem na nový lineární program takto:

$$c = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1] \tag{3.124}$$

Tento lineární program byl řešen simplexovým algoritmem, tím bylo spočteno optimální řešení primární úlohy a optimální hodnota, dále bylo dopočítáno optimální duální řešení.

Optimální řešení primární úlohy je

$$x = [0, \frac{2268}{6403}, \frac{4131}{6403}, 0, 0, 0, 0, 0, \frac{108}{337}, 0, 0, \frac{27}{19}, \frac{4}{6403}, 0, 0, 0, \frac{108}{337}, 0, 0] \quad (3.125)$$

a optimální řešení duální úlohy je

$$y = [1, 0, 0, 0, 1, -\frac{793}{6403}, -\frac{2335}{6403}] \quad (3.126)$$

Ověřením přípustnosti a optimálnosti těchto řešení se odstraní „počítačovost" tohoto důkazu. Přípustnost primárního řešení x plyne z nezápornosti x, vynásobením a porovnáním vztahu $Ax = b$. Pro duální řešení je třeba ověřit vztah $A^\top y \leq c$, vynásobení dává

$$A^\top y = [-\frac{86352}{6403}, 0, 0, -\frac{86352}{6403}, 0, -\frac{12336}{6403}, 0, -\frac{12336}{6403}, 0, 0, 0, 0, 1, 0, 0, 0, 1, -\frac{793}{6403}, -\frac{2335}{6403}] \quad (3.127)$$

což se triviálně porovná s uvedeným c.

Tedy tato řešení jsou přípustná, navíc hodnoty účelových funkcí se pro ně rovnají

$$c^\top x = \frac{2056}{6403} = b^\top y \quad (3.128)$$

Proto lze aplikovat tvrzení 2.9.1, odkud vyplývá, že optimální hodnota řešeného upraveného lineárního programu je $\frac{2056}{6403}$. Dále podle tvrzení 2.9.3 původní lineární program nemá přípustné řešení.

Tudíž pro $m = 3$ neexistuje σ_m-kompetitivní algoritmus, tím bylo dokázáno tvrzení 3.1.1.

3.5 Závěrečné poznámky

Byly nalezeny dva různé důkazy tvrzení 3.1.1. Podařilo se dokázat, že neexistuje $\frac{27}{19}$-kompetitivní pravděpodobnostní on-line algoritmus pro problém rozvrhování na $m = 3$ počítačích.

Původním záměrem bylo obecněji dokázat neexistenci σ_m-kompetitivního pravděpodobnostního algoritmu pro problém rozvrhování na m počítačích. To se bohužel přes všechnu snahu nepodařilo dokázat. Proto může být užitečné shrnout to, co bylo vyzkoušeno a nevedlo k lepšímu výsledku.

Byly vyzkoušeny následující modifikace uvedených důkazů

- více než $t > 1$ nadkritická úloha

  V silnější varianě způsobilo zvýšení počtu nadkritických úloh pro $m = 3$ počítače oslabení lineárního programu. Oslabení spočívá v přidání spousty nových proměnných a žádného silného omezení. Pro $m = 3$ a $t > 1$ nebo $m > 3$ a t libovolné byl spočtený odhad ve vyzkoušených případech roven σ_m.

- více či méně než m kritických úloh

  Pro méně kritických úloh ani nebyl dosažen odhad σ_m. Pro více kritických úloh byl dosažen stejný odhad jako pro m kritických úloh.

- různé velikosti úlohy předcházející m kritickým

  Nejlepší odhad byl pro maximální možnou nadkritickou úlohu. Vyšší nebyly zkoumány, neboť nesplňují předpoklady tvrzení 2.8.2.

- jiné zobecnění nadkritické úlohy

  Jedním z pokusů o zobecnění pojmu nadkritické úlohy bylo zvolit její velikost rovnou součtu předchozích úloh. Pro $m = 3$ definice splývají. Bohužel buď taková posloupnost (i s t takovými úlohami a m kritickými úlohami) nedala odhad lepší než σ_m, nebo nebylo možné sestrojit rozvrh podle tvrzení 2.8.2.

Úspěšnému důkazu tvrzení 3.1.1 předcházel dlouhý výzkum. Výzkum byl jak teoretický, tak i provázený řadou počítačových experimentů. Bylo provedeno hodně experimentů s cílem nalézt vhodnou těžkou posloupnost. V počátcích výzkumu, když nebyl žádný nápad, jak začít tvrzení dokazovat, bylo zkoušeno chování některých algoritmů a byla provedena řada výpočtů s cílem získat alespoň nějakou tolik potřebnou intuici v této oblasti.

Hlavní naděje pro důkaz neexistence σ_m-kompetitivního pravděpodobnostního on-line algoritmu pro $m > 3$ počítačů byly vkládány do zobecnění použité posloupnosti, která dala výsledek pro $m = 3$. Znamenalo to provést řadu výpočtů znovu a mnohem obecněji. Podle spočtených výsledků bylo třeba netriviálně upravit již existující počítačové programy.

Snad teorie vlastností σ_m-kompetitivních algoritmů vybudovaná v této práci a uvedená v této kapitole jednou pomůže nalézt důkaz neexistence takových algoritmů, případně pomůže nalézt lepší odhad, pro $m > 3$ počítačů.

Pro pokračování v tomto směru zatím bohužel schází nějaký nový nápad, jak předvedené důkazy zobecnit pro větší počet počítačů.

Shrňme zde nejzajímavější dokázané výsledky, zejména vlastnosti σ_m-kompetitivních pravděpodobnostních on-line algoritmů.

- dolní odhad pro $m = 3$, viz. tvrzení 3.1.1
- kritická úloha je umísťována na počítač s minimální zátěží, předchází-li jí alespoň m kritických úloh, viz. tvrzení 3.2.5
- přesné vyjádření průměrných zátěží počítačů po kritické posloupnosti, viz. tvrzení 3.2.2
- speciální rovnost zátěží a délek rozvrhu, viz. tvrzení 3.4.4
- délka optimálního rozvrhu pro slabou kritickou posloupnost, viz. tvrzení 2.8.2

Kapitola 4

Odhad pro běžné rozvrhovací algoritmy

V této kapitole jsou zkoumány pravděpodobnostní on-line algoritmy pro problém rozvrhování konstruované v důkazech horních odhadů. V kapitole 1.2.5 byla stručně nastíněna stuktura takovýchto algoritmů, tedy jejich hlavním rysem je to, že umísťují úlohy na dva počítače.

4.1 Úvodní poznámky

Cílem této kapitoly je sestrojit dolní odhad pro pravděpodobnostní on-line algoritmy umísťující příchozí úlohu na dva počítače řešící problém rozvrhování.

Je zdola odhadován kompetitivní poměr každého takového algoritmu.

Poznámka: *Pojmem „umísťovat na dva počítače" znamená, že algoritmus umístí úlohu buď na minimální nebo h-tý maximální počítač, kde h je pro algoritmus konstanta (při pevném m počtu počítačů).*

Takovéto algoritmy jsou zkoumány, protože jsou jednodušší než obecné pravděpodobnostní on-line algoritmy pro rozvrhování a výsledky pro ně by mohly přinést zkušenosti užitečné pro zkoumání těch obecných. Navíc takovéto existující algoritmy dávají celkem dobré výsledky, tudíž toto zjednodušení není přílišné.

Protože pro tento problém neexistují odhady, tak není výsledky s čím porovnávat. Původní záměr odhadu byl případně zjistit, že Seidenovy algoritmy, viz. Seiden [14], jsou dobré. Motivací bylo pokusit se dokázat dolní odhad blízký kompetitivnímu poměru Seidenových algoritmů.

Bohužel získaný odhad není těmto Seidenovým algoritmům moc blízký. To neznamená, že předvedený odhad je slabý, přesto může být blízký optimu.

Poznámka: *Zřejmě tento dolní odhad nesouvisí s dolními odhady pro Seidenovy algoritmy, neboť ty odhadují kompetitivní poměr konkrétního algoritmu nikoliv této třídy algoritmů umísťujících na dva počítače.*

Bohužel, vzhledem k formě důkazu, se zatím nepodařilo sestrojit standardní matematický důkaz. Protože je předveden počítačový důkaz, není možné se na výsledky zcela spoléhat. Avšak vzhledem k ověřitelnosti tohoto důkazu čtenářem může být považován za důvěryhodný. Přinejmenším spočtené odhady mohou být považovány za „odrazový můstek“ pro další výzkum v tomto směru.

4.2 Myšlenka

Základní myšlenka odhadu se odvíjí od pozorování, že pravděpodobnostní on-line algoritmy nemohou být postaveny na invariantu udržujícím nějaké pevné průměrné rozložení zátěží, viz. tvrzení 3.2.8, jak již bylo dříve zmíněno.

Proto bude konstruován obecný odhad počítaný přes všechny možné distribuce konfigurací na dlouhé posloupnosti jednotkových úloh. Postup, jak důkaz probíhá, lze schematicky popsat takto:

- **volba „těžké“ posloupnosti**

  Prefix posloupnosti je tvořen potenciálním nekonečnem jednotkových úloh následovaných $t \geq m$ kritickými úlohami. Ze zkušenosti je tato posloupnost považována za „těžkou,“ nelze však vyloučit existenci posloupnosti dávající lepší odhad.

- **konstrukce nerovností dle definice** 2.8.3 **kompetitivního poměru**

  Sestrojí se integrální rovnice, protože odhad je prováděn obecně přes libovolnou distribuci konfigurací, tedy průměr se počítá integrálem přes ní.

- **rozklad na intervaly**

  Interval, přes který se integruje, je rozložen na části, na nichž je integrovaná (po částech lineární) funkce lineární.

- **zjemnění intervalů**

  Intervaly vytvořené v předchozím kroku mají různorodou velikost. Později bude třeba přes každý takový interval odhadovat integrál, tudíž je výhodné je mít co nejmenší. Intervaly jsou proto rozkouskovány na dostatečně malé, velikost je parametrem řešení.

- **zjednodušení integrálních rovnic**

  Při bližším zkoumáním lze zjistit, že je integrována po částech lineární funkce. V této lineární funkci hraje hlavní roli konstanta. Samostatně se integruje po částech konstantní funkce, zbytek je integrován odděleně. Integrál je nahrazen sumou přes intervaly, na kterých je integrovaná (po částech lineární) funkce lineární. Konstanta se zintegruje, uvnitř sumy zůstane ještě integrál lineárního členu.

Poznámka: *Lineární člen nelze snadno zintegrovat, neboť se integruje přes distribuci.*

- **slepení intervalů**

  Někdy může být příliš mnoho vytvořených intervalů, pak je v zájmu snížení časové náročnosti žádoucí snížit počet těchto intervalů jejich spojováním v intervaly o nějaké minimální délce. Pro výsledný interval se použije koeficient, který je roven minimálnímu koeficientu podintervalů.

- **převod na lineární nerovnice**

  Pro získání lineární rovnice je třeba zbývající integrál zdola odhadnout. Odhad je snadný, neboť je integrována monotónní funkce. Zmíněné zjemnění bylo provedeno, aby tento odhad integrálu příliš neoslaboval dokazovaný výsledek.

- **vygenerování soustavy počítačovým programem**

  Počítačový program napsaný v jazyku **BC** sestrojí formálně odvozený lineární program, který bude dále řešen.

- **řešení lineárního programu**

  Byl napsán počítačový program v jazyku **BC** řešící lineární program simplexovým algoritmem s absolutní přesností v racionálních číslech.

4.3 Značení a základy

Následující symboly jsou pevné a nebudou se vyskytovat v jiném významu.

- $m \in \mathbb{N}$ počet počítačů
- $h \in [m]$ číslo druhého počítače pro umísťování úloh
- $\mathcal{A}$ pravděpodobnostní on-line algoritmus umísťující příchozí úlohu na minimální a h-tý maximální počítač
- $P \in \mathcal{P}_T$ těžká posloupnost
  množina těžkých posloupností je definována v následujících definicích

Definice 4.3.1 Pro $m \in \mathbb{N}, m > 1$, $t \in \mathbb{N}$, $t \geq m$ $P \in \mathcal{P}$ *nazveme* ***těžká posloupnost****, pokud platí*

$$Crit(P, t) \wedge (\forall i \in [|P| - t]) : \; P_i = 1 \tag{4.1}$$

Těžká posloupnost je tvořena prefixem jednotkových úloh následovaných t kritickými úlohami.

Definice 4.3.2 Pro $m \in \mathbb{N}, m > 1$ *definujme množiny*

$$\mathcal{P}_T^1 \stackrel{def}{=} \{P \in \mathcal{P} \mid P \textit{ je těžká posloupnost}\} \tag{4.2}$$

$$\mathcal{P}_T \stackrel{def}{=} \textit{uzávěr množiny } \mathcal{P}_T^1 \textit{ na limitu} \tag{4.3}$$

Množina těžkých posloupností a množina těžkých posloupností rozšířená o limity těžkých posloupností.

Poznámka: *Existuje nekonečně mnoho dlouhých těžkých posloupností. Lze tedy požadovat těžkou posloupnost o alespoň zvolené délce.*

Tvrzení 4.3.3 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $(\forall i \in [|P|]) : \; P_i = 1$ *platí*

$$(\forall \mathcal{B} \in \mathcal{R})(\exists a, d \in \mathbb{N}_0)(\forall i \in [m]) : \tag{4.4}$$

$$i \leq m - h : \; \mathit{Load}_{(\mathcal{A},P,\mathcal{B})}(i) \in \langle a, a+1 \rangle$$

$$i > m - h : \; \mathit{Load}_{(\mathcal{A},P,\mathcal{B})}(i) \in \langle a+d, a+d+1 \rangle$$

Výsledná konfigurace při pevně zvolené náhodné posloupnosti má dvě hladiny, tedy a a $a+d$.

Poznámka: *Člen* $+1$ *lze při dostatečně dlouhých posloupnostech zanedbat, v limitních posloupnostech zmizí úplně.*

Důkaz. Snadný, indukcí podle délky posloupnosti P. □

Tvrzení 4.3.4 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}$, $(\forall i \in [|P|]) : \; P_i = 1$ *platí*

$$(\exists a, d \in \mathbb{R})(\forall i \in [m]) : \tag{4.5}$$

$$i \leq m - h : \; \mathbb{E}\mathit{load}_{(\mathcal{A},P)}(i) \in \langle a, a+1 \rangle$$

$$i > m - h : \; \mathbb{E}\mathit{load}_{(\mathcal{A},P)}(i) \in \langle a+d, a+d+1 \rangle$$

Průměrná konfigurace má dvě hladiny a a $a+d$.

Důkaz. Snadno z tvrzení 4.3.3 a definice střední hodnoty. □

Poznámka: *Uvedená tvrzení mohou být analogicky formulována i pro algoritmy umísťující úlohy na více než dva počítače. V obecné verzi platí, že počet hladin získané konfigurace je roven počtu počítačů, na které algoritmus umísťuje příchozí úlohy. Rovnost předpokládá obecnou konfiguraci, protože hladiny mohou splývat v singulárních konfiguracích.*

4.4 Normalizace, limitní přechod

Pro účely dalšího textu je potřeba parametrizovat konfigurace získané na jednotkových prefixech těžkých posloupností. Získané konfigurace jsou normalizovány tak, aby součet všech zátěží byl přibližně 1. Vše je dále počítáno v reálných číslech.

Před normalizací podle tvrzení 4.3.3 pro těžkou posloupnost $P \in \mathcal{P}_T$ platí (rovnost z definice 4.3.1)

$$ma + hd \leq sum(P^{-t-1}) = |P| - t \leq ma + hd + m \tag{4.6}$$

Poznámka: *I po normalizaci se bude mluvit o jednotkovém prefixu (už normalizavané) posloupnosti, tím je míněn prefix úloh, které před normalizací byly jednotkové.*

Normalizace se provede „zmenšením měřítka“ a to tak, aby součet úloh v jednotkovém prefixu byl 1, tedy každá úloha je podělena délkou tohoto prefixu. Tudíž v normalizovaném tvaru platí:

$$ma + hd \leq 1 \leq ma + hd + \frac{m}{|P| - t} \tag{4.7}$$

Pro dolní odhad kompetitivního poměru může být použita libovolná těžká posloupnost. Triviálně může být vytvořena posloupnost takovýchto posloupností s rostoucí délkou posloupnosti. Tudíž člen $\frac{m}{|P|-t}$ konverguje k 0 podle délky takové posloupnosti.

Použitím **limitního přechodu** se získá limitní posloupnost, dolní odhad dokázaný pro limitní posloupnost bude dokazovaným dolním odhadem. Budou odvozeny nerovnosti, které musí být splněny pro každou posloupnost. Tedy podle limitního přechodu musí být tyto nerovnosti splněny i v limitě.

Poznámka: *Limitní přechod je možné obejít podobně, jako tomu bylo v kapitole* 3.3 *počítáním s posloupností malých* ε_i *konvergujících k* 0. *Ovšem to by znamenalo zbytečné prodloužení a formální komplikace důkazu na úkor čitelnosti.*

Poznámka: *Limitní posloupnost je tvořena nekonečně dlouhým prefixem úloh o celkové velikosti 1, po kterých se dostane do konfigurace mající na* $m-h$ *počítačích zátěž právě* a *a na zbývajících* h *počítačích zátěž právě* $a+d$.

Limitou výrazu 4.7 je rovnost

$$ma + hd = 1 \tag{4.8}$$

Z tohoto vztahu lze v limitě jednoznačně vyjádřit d rozdíl hladin takto

$$d = \frac{1 - ma}{h} \tag{4.9}$$

Možné konfigurace, ke kterým algoritmus dospěje pro jednotkový prefix limitní posloupnosti, lze popsat na základě vztahu 4.9 a požadavků $a, d \geq 0$. Konfigurace jsou dány parametrem $a \in \left\langle 0, \frac{1}{m} \right\rangle$, hodnota d je dopočítána podle 4.9.

Nyní lze uvést další značení

- $a \in \left\langle 0, \frac{1}{m} \right\rangle$ značí nižší hladinu konfigurace po jednotkovém prefixu
- $d \in \langle 0, 1 \rangle$ značí rozdíl mezi vyšší a nižší hladinou konfigurace po jednotkovém prefixu
- $f(a) \in \langle 0, 1 \rangle$ pro $a \in \left\langle 0, \frac{1}{m} \right\rangle$ je pravděpodobnostní distribuce deterministických konfigurací po prefixu jednotkových úloh posloupnosti P pro algoritmus $\mathcal{A}$. Funkce $f(a)$ je neznámá měřitelná funkce. Důkaz je veden přes všechny možné takové funkce.

- $k \in \{0,1\}^t$ je předpis chování algoritmu $\mathcal{A}$ na posledních t kritických úlohách při pevné náhodné posloupnosti. Hodnota $[k]_j^t$ určuje umístění kritické úlohy P_{-t+j-1}, 0 znamená na minimální, 1 na h-tý maximální. Algoritmus $\mathcal{A}$ je pravděpodobnostní kombinací těchto předpisů.

  Poznámka: *Algoritmus a náhodná posloupnost určují právě jeden takový předpis. Používají se i částečné předpisy, tj. nějaký prefix k, určující umístění zkrácené posloupnosti.*

- $p_k(P,a)$ značí pravděpodobnost, že se algoritmus $\mathcal{A}$ rozhodnul provést umístění t kritických úloh předpisem k pro posloupnost P a konfiguraci a. Analogicky se definuje pro prefix předpisu.

Pro $f(a)$ pravděpodobnostní distribuci přes $a \in \left\langle 0, \frac{1}{m} \right\rangle$ platí z definice pravděpodobnostní distribuce následující vztah

$$\int_0^{\frac{1}{m}} f(a)da = 1 \tag{4.10}$$

Z definice úplného jevu platí

$$(\forall w \in [t]) : \sum_{k \in \{0,1\}^w} p_k(P,a) = 1 \tag{4.11}$$

a pravděpodobnost lze vyjádřit takto

$$(\forall w \in [t])(\forall k \in \{0,1\}^w) : \; p_k(P,a) = \sum_{l \in k \times \{0,1\}^{t-w}} p_l(P,a) \tag{4.12}$$

Pro další odvozování je velmi důležitá délka rozvrhu, tzv. makespan. Stejně jako konfigurace bude používán v normalizovaném tvaru, takže jeho hodnota bude reálné číslo.

Definice 4.4.1 Pro $m \in \mathbb{N}, m > 1$, $w \in [t]$, $k \in \{0,1\}^w$, $a \in \left\langle 0, \frac{1}{m} \right\rangle$, $P \in \mathcal{P}_T$ *označme symbolem mks*$_k(P,a)$ *normalizovanou délku rozvrhu při umístění prvních w úloh z posledních t kritických úloh posloupnosti P na konfiguraci danou parametrem a. Úlohy byly umístěny podle předpisu k.*

Poznámka: *Formálnost není třeba, navíc by snížila čitelnost.*

Pro zkoumání vlastností této funkce je třeba získat její lepší popis. Délka rozvrhu *mks*$_k(P,a)$ se bude počítat součtem jednodušších složek:

- základní hladina a
  jednotkový prefix dává konfiguraci o hladinách a a $a+d$, nutně a bude složkou délky rozvrhu

- $Q_P(k,a) \cdot d$ je případný rozdíl hladin o d
  funkce $Q_P(k,a)$ nabývá hodnotu 0 nebo 1, určuje, zda na maximálním počítači je celkem a nebo $a+d$ jednotkových úloh bez případné poslední před normalizací nepřičtené.

- $M_P(k,a)$ je součet úloh z posledních t kritických úloh posloupnosti P umístěných na maximálním počítači.

- $\varepsilon_k(P,a)$ odchylka (korekce) kvůli případné $+1$ v hladině před normalizací. Hodnota (stejnoměrně) konverguje dle délky posloupnosti k 0. Pro limitní posloupnost je rovno 0.

Poznámka: *Uvažovaný maximální počítač je předem pevně zvolen (může jich být více).*

Tedy pro posloupnost $P \in \mathcal{P}_T$ platí

$$\textit{mks}_k(P,a) = a + Q_P(k,a)\frac{1-ma}{h} + M_P(k,a) + \varepsilon_k(P,a) \tag{4.13}$$

Vztah 4.13 platí pro každou posloupnost. Musí platit i v limitě, tedy pro limitní posloupnost P, takto

$$mks_k(P, a) = a + Q_P(k, a)\frac{1 - ma}{h} + M_P(k, a) \tag{4.14}$$

snadnou úpravou se získá

$$mks_k(P, a) = \left(M_P(k, a) + \frac{Q_P(k, a)}{h}\right) + a \cdot \left(1 - \frac{m}{h} Q_P(k, a)\right) \tag{4.15}$$

Tvrzení 4.4.2 Pro $m \in \mathbb{N}, m > 1$, $P \in \mathcal{P}_T$, $w \in [m+1]$, $k \in \{0,1\}^w$ *platí, že zobrazení* $M_P(k, a)$ *a* $Q_P(k, a)$ *na* $a \in \left\langle 0, \frac{1}{m} \right\rangle$ *jsou po částech konstantní funkce.*

Důkaz. Snadný, neboť může nabývat jen některou z m hodnot, kterou je součet velikostí kritických úloh umístěných na počítačích podle předpisu k.

Poznámka: *V těchto funkcích není přičtena hladina konfigurace, určují jen koeficienty zkoumané po částech lineární funkce. Hodnota* a *ovlivňuje jen výběr výsledné konfigurace.* □

Definice 4.4.3 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}_T$, $w \in [m+1]$, $k \in \{0,1\}^w$ *definujme rozklad intervalu* $\left\langle 0, \frac{1}{m} \right\rangle$ *na podintervaly, na kterých má po částech lineární funkce* $mks_k(P, a)$, *ve vyjádření formulí* 4.15, *konstantní koeficienty. Tj.* $M_P(k, a) + \frac{Q_P(k,a)}{h}$ *a* $1 - \frac{m}{h} Q_P(k, a)$ *jsou konstanty.*

Nechť jsou tyto intervaly označeny $I_1, \ldots, I_T$, *kde* T *je jejich počet.*

Poznámka: *Definice je korektní podle tvrzení* 4.4.2.

4.5 Podmínka kompetitivního poměru

Nyní již je možné přikročit k formulaci požadavku kompetitivního poměru. Opět je požadavkem splnit definici kompetitivního poměru, tedy průměrná délka rozvrhu musí být nejvýše ϱ-krát optimální délka rozvrhu.

Tvrzení 4.5.1 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}_T$, $w \in [t]$, $a \in \left\langle 0, \frac{1}{m} \right\rangle$ *je průměrná délka rozvrhu při konfiguraci jednotkového prefixu s parametrem* a *rovna*

$$\sum_{k \in \{0,1\}^w} \left(p_k(P, a) \cdot mks_k(P, a) \right) \tag{4.16}$$

Tj. průměrná délka rozvrhu po umístění prvních w úloh z t posledních kritických úloh posloupnosti P na konfiguraci danou parametrem a.

Důkaz. Přímo z definice střední hodnoty. □

Tvrzení 4.5.2 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}_T$, $w \in [m+1]$, *pravděpodobnostní distribuce* $f(a)$ *na* $a \in \left\langle 0, \frac{1}{m} \right\rangle$ *je průměrná délka rozvrhu*

$$\int_0^{\frac{1}{m}} f(a) \cdot \sum_{k \in \{0,1\}^w} \left(p_k(P, a) \cdot mks_k(P, a) \right) da \tag{4.17}$$

Tj. průměrná délka rozvrhu po umístění prvních w z t posledních kritických úloh posloupnosti P.

Důkaz. Snadno z definice střední hodnoty a tvrzení 4.5.1. □

Podle tvrzení 4.5.2 a 2.8.2 lze vytvořit dolní omezení pro kompetitivní poměr ϱ, neboť délka optimálního rozvrhu pro $w \in [t]$ je $OPT(P^{-t+w-1})$. Proto lze formálně definovat omezující podmínku pro $w \in [t]$, $P \in \mathcal{P}_T$ jako

$$\int_0^{\frac{1}{m}} f(a) \cdot \sum_{k \in \{0,1\}^w} \left(p_k(P, a) \cdot mks_k(P, a) \right) da \leq \varrho \cdot OPT(P^{-m-1+w-1}) \tag{4.18}$$

Uvedená nerovnost obsahuje integrál a proměnnými jsou funkce, tudíž v tomto tvaru není použitelná pro konstrukci lineárního programu. Je proto třeba změnit integrál na sumu.

Jednak je potřeba délku rozvrhu $mks_k(P, a)$ pro $a \in I_i$ zdola odhadnout konstantou. To je triviální operace, vzhledem k tomu, že meze intervalů I_i pro $i \in [T]$ jsou známy a $mks_k(P, a)$ je známá lineární funkce na I_i.

Definice 4.5.3 Pro $m \in \mathbb{N}, m > 1$, $P \in \mathcal{P}_T$, $i \in [T]$, $w \in [t]$, $k \in \{0,1\}^w$ *definujme*

$$C_k(P, i) \stackrel{def}{=} \min_{a \in I_i} mks_k(P, a) \tag{4.19}$$

Je konstanta, která je dolním odhadem pro výraz $mks_k(P, a)$ *na* $a \in I_i$.

Poznámka: *Definice je korektní, protože* $mks_k(P, a)$ *je lineární funkce v* a. *Pro limitní posloupnost nastává rovnost.*

Tvrzení 4.5.4 Pro $m \in \mathbb{N}, m > 1$, *algoritmus* $\mathcal{A}$, $P \in \mathcal{P}_T$, $i \in [T]$, $w \in [m+1]$ *platí*

$$\int_{a \in I_i} f(a) \sum_{k \in \{0,1\}^w} \big(p_k(P, a) \cdot C_k(P, i)\big) da = \tag{4.20}$$

$$= \int_{a \in I_i} f(a) da \cdot \sum_{k \in \{0,1\}^w} \left(\frac{\int_{a \in I_i} f(a) \cdot p_k(P, a) da}{\int_{a \in I_i} f(a) da} \cdot C_k(P, i) \right)$$

Důkaz. Pomocí věty o linearitě střední hodnoty. Snadné, neboť platí pro každé k.

Poznámka: *Pro limitní posloupnost nastává rovnost.* □

Podle tvrzení 4.5.4 a definice 4.5.3 lze zdola odhadnout průměrnou délku rozvrhu. Je odstraněněn integrál, použité symboly dále viz. 4.22 a 4.23. Odhad vypadá takto

$$\int_{a \in I_i} f(a) \sum_{k \in \{0,1\}^w} \big(p_k(P, a) \cdot C_k(P, i)\big) da = \sum_{i \in [T]} \left(F_i \cdot \sum_{k \in \{0,1\}^w} (p_k(i) \cdot C_k(P, i)) \right) \tag{4.21}$$

Kde jsou použité symboly definovány takto

$$F_i \stackrel{def}{=} \int_{a \in I_i} f(a) da \tag{4.22}$$

$$p_k(i) \stackrel{def}{=} \frac{\int_{a \in I_i} f(a) \cdot p_k(P, a) da}{\int_{a \in I_i} f(a) da} \tag{4.23}$$

Navíc podle vztahů 4.10 a 4.11 lze snadno ukázat, že platí

$$\sum_{i \in [T]} F_i = 1 \tag{4.24}$$

$$\sum_{k \in \{0,1\}^w} p_k(i) = 1 \tag{4.25}$$

tudíž platí

$$\sum_{i \in [T], k \in \{0,1\}^w} F_i \cdot p_k(i) = 1 \tag{4.26}$$

Požadavek kompetitivního poměru nyní vypadá takto 2.8.2):

$$(\forall w \in [t]) : \sum_{i \in [T]} \left(F_i \cdot \sum_{k \in \{0,1\}^w} (p_k(i) \cdot C_k(P, i)) \right) \leq \varrho \cdot P_{-t+w-1} \tag{4.27}$$

I tento vztah je splněn pro každou posloupnost, tedy platí i v limitě. Potřebné koeficienty $C_k(P, i)$ jsou spočteny v limitě (pro limitní posloupnost) podle definice 4.5.3.

Lineární program pro výpočet dolního odhadu kompetitivního poměru se sestrojí podle vztahů 4.26 a 4.27. Podstatná je volba třídy těžkých posloupností, parametrem $\mathcal{P}_T$ je $t \geq m$. Výpočtem bylo zjištěno, že zajímavé výsledky se získají volbou $t = m + 1$.

Proměnné lineárního programu odpovídají součinu $F_i \cdot p_k(i)$, nechť příslušná proměnná je označena $x_{i,k}$. Hodnoty $p_l(i)$ jsou vyjádřeny pomocí $p_k(i)$ pro něž $k \in \{0, 1\}^t$. Lineární program s minimalizací účelové funkce vypadá takto:

- účelová funkce je

$$x_\varrho \tag{4.28}$$

- rovnice podle 4.26

$$\sum_{i \in [T], k \in \{0,1\}^t} x_{i,k} = 1 \tag{4.29}$$

- dále nerovnice podle 4.27 pro $w \in [t]$

$$\sum_{i \in [T]} \sum_{k \in \{0,1\}^w} \left(C_k(P, i) \cdot \sum_{l \in k \times \{0,1\}^{t-w}} x_{i,l} \right) \leq \varrho \cdot P_{-t+w-1} \tag{4.30}$$

Za neznámé koeficienty se dosadí z definice posloupnosti 4.3.1. Získaný lineární program je řešen simplexovou metodou.

4.6 Počítačové řešení

Protože konstrukce je komplikovaná a v této formě je téměř „nad lidské síly," tak byl sestaven počítačový program, který sestaví tento lineární program. Počítačový program byl napsán v počítačovém jazyku **BC** a jeho zdrojový kód je uveden v příloze této práce.

Získaný lineární program bylo potřeba nějakým způsobem vyřešit. Žádný dostupný software nebyl vyhovující, neboť nebyl schopen vyřesit tak rozsáhlý lineární program s absolutní přesností v racionálních číslech. Proto byl napsán program v jazyku **BC** řešící primární lineární program simplexovou metodou. Jeho zdrojový kód je rovněž uveden v příloze této práce.

Poznámka: *Užití a ověření zdrojových kódů vyžaduje znalosti operačního systému UNIX či Linux a programovacího jazyku* ***BC****. Mezi různými verzemi jsou určité odlišnosti, které mohou způsobit drobné technické problémy. Pro účely této práce byla použita verze je 1.06, která musela být upravena tak, aby podporovala dostatečně velká pole. Úprava tohoto programu je rovněž uvedena v příloze této práce.*

4.7 Výsledky

Výsledky byly spočteny pro $m = 2, 3, 4, 5, 6$, výpočet pro více počítačů je komplikován neustále rostoucí časovou složitostí, neboť počet proměnných lineárního programu značně roste.

Poznámka: *Časová složitost záměrně není rozebírána, neboť závisí na zjemnění intervalů, na kterém závisí „kvalita" výsledku.*

Výsledky jsou shrnuty v tabulkách 4.1 a 4.2. V tabulce 4.1 jsou rozepsány dolní odhady pro algoritmy umísťující na minimální a h-tý maximální počítač podle h, výsledný dolní odhad je minimalní z nich. Tabulka 4.2 srovnává dosažené dolní odhady s dolními odhady pravděpodobnostních on-line algoritmů a horními odhady Seidenových algoritmů.

4.8 Závěrečné poznámky

Podle tabulky 4.2 se podařilo spočítat zajímavý výsledek pro $m = 3, 5, 6$, tedy všechny tyto výsledky jsou lepší než odhad σ_m pro obecné pravděpodobnostní on-line algoritmy.

V případě pro $m = 2$ počítače je znám optimální algoritmus, tomuto výsledku se získaný odhad hodně přiblížil. Odhad pro $m = 2$ byl počítán jen pro ilustraci toho, že získané odhady nemusí být příliš vzdálené optimu.

V případě pro $m = 4$ počítače se přes všechno zjemňování nepodařilo v rozumném čase dokázat lepší odhad než pro obecné algoritmy. Obtížným případem je umísťování úloh na minimální a druhý maximální počítač. V tomto případě se získaný odhad dost přiblížil obecnému odhadu.

Poznámka: *Na základě zkušeností a pozorování se domnívám, že pro $m = 4$ počítače existuje algoritmus umísťující příchozí úlohy na minimální a druhý maximální počítač, který je lepší než Seidenovy algoritmy.*

m	Dolní odhad pro pevné h: 1	2	3	4	5	Dolní odhad
2	$\frac{127}{96}$ 1.32291					$\frac{127}{96}$ 1.32291
3	$\frac{315}{221}$ 1.42533	$\frac{200149}{139824}$ 1.43143				$\frac{315}{221}$ 1.42533
4	$\frac{1224708}{810299}$ 1.51142	$\frac{45894095371}{31397765060}$ 1.46169	$\frac{437479}{288576}$ 1.51599			$\frac{45894095371}{31397765060}$ 1.46169
5	$\frac{8}{5}$ 1.60000	$\frac{38563235419399785}{25793244858256829}$ 1.49509	$\frac{52454839}{34975955}$ 1.49974	$\frac{3779491}{2391040}$ 1.58068		$\frac{38563235419399785}{25793244858256829}$ 1.49509
6	$\frac{5}{3}$ 1.66666	$\frac{2875913448861574021 4}{1911194708608951587 9}$ 1.50477	$\frac{797052565279793}{526289586249368}$ 1.51447	$\frac{1695768673820548}{1117453753428309}$ 1.51752	$\frac{47981194157}{29502571875}$ 1.62633	$\frac{28759134488615740214}{19111947086089515879}$ 1.50477

Tabulka 4.1: Dolní odhady algoritmů umísťující úlohy na dva počítače

m	**Dolní odhad na dva**	**Obecný dolní odhad**	**Horní odhad na dva**
2	1.32291	1.33333	1.33333
3	1.42533	1.42105	1.55871
4	1.46169	1.46285	1.67517
5	1.49509	1.48738	1.74114
6	1.50477	1.50352	1.78810

Tabulka 4.2: Srovnání získaných a známých výsledků

Spočtená hodnota odhadu pro jednotlivá h může pomoci při konstrukci nových algoritmů. Odhad dává určitou představu o tom, na které počítače je více a na které je méně výhodné umísťovat úlohy.

Poznámka: *Po zkušenostech z výpočtů za nejnáročnejší pro malý počet počítačů považuji počítání dolních odhadů při umísťování úloh na minimální a druhý maximální počítač.*

Tato metoda byla uvedena také proto, že se jedná o novou obecnou metodu pro konstrukci dolních odhadů. Navíc umožňuje počítat odhady přes všechny možné pravděpodobnostní distribuce konfigurací.

Dá se očekávat, že uvedená metoda může být zlepšena, například volbou jiné těžké posloupnosti atp., pro získání silnějšího dolního odhadu.

Poznámka: *Bylo propočítáváno, zda jsou výsledky podstatně ovlivněny (oslabeny) „zaokrouhlením" vzniklým při převodu integrálu na sumu, avšak ukázalo se, že rozdíl je velmi malý (v řádu setin).*

Kapitola 5

Popis příloh na kompaktním disku

Přílohou této práce je kompaktní disk. Tato kapitola se zabývá popisem a použitím souborů ulozených na tomto kompaktním disku. Pro použití této přílohy se předpokládají znalosti operačního systému UNIXového typu, například Linux, vybavený standardními UNIXovými příkazy (sh, make, awk, sed, gcc, rm, mv, . . .).

5.1 Instalace

Soubory uložené na kompaktním disku je možné prohlížet pod libovolným operačním systémem na počítači s přístupem k CD-ROM. Kopii kompaktního disku lze také nalézt na internetu na adrese

http://kiwi.ms.mff.cuni.cz/~tom/diplomka/

Kopírování

Pro použití ověřovacích skriptů je potřeba si tento disk zkopírovat do domovského adresáře na počítači vybeveným operačním systémem UNIXového typu. Například pomocí příkazů:

```
$ mount /mnt/cdrom
$ mkdir ~/diplomka
$ cp -Rf /mnt/cdrom ~/diplomka
$ umount /mnt/cdrom
$ cd ~/diplomka
```

Zkopírování zabalené přílohy z internetu se provede buď pomocí internetového prohlížeče uložením souboru **diplomka.tar.gz** na domovské stránce této práce nebo pomocí programu **lynx**, je-li nainstalován, příkazem:

```
$ lynx -source http://kiwi.ms.mff.cuni.cz/~tom/diplomka/
download/diplomka.tar.gz > ~/diplomka.tar.gz
```

Zabalenou přílohu je možné rozbalit příkazem:

```
$ mkdir ~/diplomka
$ tar xvfz ~/diplomka.tar.gz -C ~/diplomka
$ cd ~/diplomka
```

Adresář download

Neexistuje-li adresář **download**, tak jeho vygenerování se provede příkazem

```
$ cd ~/diplomka
$ make download
```

Tento adresář je sice uložen na kompaktním disku, avšak není uložen v souborech obsahující zabalené adresáře a soubory kompaktního disku kvůli zřejmému „zacyklení.“

Překompilování

Smazání všech vygenerovaných souborů se provede příkazem

```
$ make clean
```

Opětovné vygenerování souborů, včetně překompilování zdrojového kódu v jazyku LaTeX do různých formátů a vytvoření adresáře **download** spolu s vygenerováním zabalených kopií přílohy, se provede příkazem

```
$ make
```

Poznámka: *Toto přegenerování není nutné.*

HTML stránka

Přílohu je možné prohlížet internetovými prohlížeči. Je možné si prohlížet zmíněnou domovskou stránku diplomové práce nebo ji prohlížet lokálně. Pro lokální prohlížení stačí otevřít soubor **index.html** na tomto kompaktním disku nebo v jeho kopii. Prohlížení programem **lynx** je možné příkazem:

```
$ lynx ~/diplomka/index.html
```

nebo obdobně programem **Netscape**:

```
$ netscape ~/diplomka/index.html
```

5.2 Obsah přílohy

Hlavní adresář

Hlavní adresář kompaktního disku obsahuje následující soubory a adresáře:

- **README.txt**

  Obsahuje hlavičku diplomové práce a komentář k tomuto disku.

- **index.html**

  Je HTML dokument obsahující kopii domovské stránky diplomové práce. Prostřednictvím této stránky je možné si prohlížet obsah kompaktního disku.

- **Makefile**

  Obsahuje definice příkazů pro program **make**. Jsou to příkazy:

  ```
  $ make
  $ make download
  $ make clean
  ```

- **overeni/**

  Tento adresář obsahuje skripty, programy a jejich výstupy pro snadné ověření správnosti spočtených odhadů v této práci.

- **latex-src/**

  Tento adresář obsahuje zdrojový kód diplomové práce a jeho překlady do různých datových formátů (PostScript, PDF, DVI).

- **images/**

  Tento adresář obsahuje obrázky použité v HTML stránce. V svém podadresáři **mff-logo/** obsahuje logo Matematicko-fyzikální fakulty Univerzity Karlovy použité na titulní straně diplomové práce.

- **download/**

  Tento adresář obsahuje vygenerované soubory obsahující zabalené ostatní adresáře, dále obsahuje text diplomové práce v různých datových formátech. Tento adresár je určen pro kopírování.

  Pokud tento adresář neexistuje, tak může být vygenerován příkazem:

  ```
  $ make download
  ```

Adresář latex-src/

Tento adresář obsahuje zdrojový kód diplomové práce, je napsaný v typografickém systému LaTeX. Navíc se v tomto adresáři nacházejí i překlady diplomové práce do různých datový formátů pro jejich snadné prohlížení a tisk.

Následující soubory jsou původní:

- **src.tex**

  Je hlavním souborem této práce napsaný v systému LaTeX. Jeho kompilováním se dostanou vysázené formáty pro snadné prohlížení a tisk.

- **debugfiltr.sh**

  Je pomocný skriptík použitý při kompilaci. V zdrojovém kódu rozlišuje, které řádky budou jen v ladící verzi a které ne. Je používán interně příkazem **make**.

- **Makefile**

  Obsahuje definice příkazů pro program **make**. Definovány jsou následující příkazy:

  ```
  $ make
  $ make clean
  $ make test
  $ make debug
  ```

  Tyto příkazy provádějí postupně kompilaci zdrojových kódů, smazání generovaných souborů, rychlou testovací kompilaci a kompilaci s ladícími poznámkami v textu (návěští atp.).

Zbylé soubory jsou generované a obsahují text diplomové práce přeložený do různých datových formátů:

- **diplomka.ps, diplomka.pdf, diplomka.dvi**

  PostSciptová verze, PDF - Portable Document Format, Device Independent verze.

- **diplomka.ps.gz, diplomka.pdf.gz, diplomka.dvi.gz**

  Stejné jako předchozí, navíc zkomprimované programem GZip.

- **diplomka.tex**

  Upravený původní zdrojový kód po přefiltrování pomocným skriptem. U některých řádků bylo rozhodováno, zda budou vloženy. Rozhodování záleželo na tom, zda výsledkem má být finální nebo ladící verze.

Adresář overeni/

Tento adresář obsahuje skripty, program a jejich výstupy pro snadné ověření správnosti odhadů provedených v této práci. Nachází se zde několik adresářů:

- **bin**

  Obsahuje pomocné filtry. Filtr **g2t.awk** posílá na výstup právě jen ohraničenou část. Filtr **tab2bc.awk** slouží pro vygenerování kódu v jazyku **BC**, který inicializuje číselná pole podle vstupní tabulky popisující lineární program. Filtr **tab2xtab.awk** převádí vstupní lineární program na lineární program mající nulové optimální řešení právě, když původní má optimální řešení, viz. tvrzení 2.9.3. Po zkompilování se zde nachází i program **bc** interpretující jazyk **BC**, jazyk pro matematické výpočty.

- **odhad-m3**

  Programy v jazyku **C** použité v základním dolním odhadu pro $m = 3$ počítače. Tyto programy generují lineární programy popsané v kapitolce 3.3. Program **glp** je uveden v této práci přímo v textu a generuje lineární program právě pro $m = 3$. Program **glpm** je zobecněný předchozí pro obecné m.

  Soubory **glp.c** a **glpm.c** jsou jejich zdrojové kódy. Programy jsou zkompilovány automaticky při prvním použití testovacích skriptů.

- **silnejsi**

  Obsahuje program **silnejsi** v jazyku **C** generující lineární program pro silnější verzi důkazu dolního odhadu kompetitivního poměru pro pravděpodobnostní on-line algoritmy. Program na vstupu dostane počet počítačů a počet nadkritických úloh, které mají být obsaženy ve vstupní posloupnosti, podrobnosti viz. kapitolka 3.4.

Zdrojový kód programu je v souboru **silnejsi.c**.

- **na-dva**

  Obsahuje program **gen.bc** v jazyku **BC** generující lineární program počítající dolní odhad pro třídu pravděpodobnostních on-line algoritmů pro rozvrhování umísťujících příchozí úlohy na dva počítače. Tento lineární program je definovaný v kapitole 4.

  Soubor **gen.bc** je zdrojový a současně spustitelný, neboť je interpretován interpreterem jazyka **BC**, interpreter je program **bin/bc**.

- **simplex**

  Obsahuje soubor **simplex.bc0**, který je částí programu v jazyku **BC**, která umí řešit lineární program simplexovým algoritmem, viz. Grygarová [9].

  Tato práce se nezabývá rozebíráním simplexového algoritmu, proto nechť se každý podívá do okomentovaného zdrojového kódu a ověří jeho správnost podle uvedené literatury.

  Spojením výstupu z filtru **tab2bc.awk** s tímto souborem dojde k vytvoření programu v jazyku **BC**, který řeší lineární program, jež měl filtr na vstupu.

- **ruzne**

  Obsahuje soubor **bc_1.06.tar.gz**, ve kterém jsou zdrojové soubory od interpretru programovacího jazyka **BC** pro matematické výpočty ve verzi 1.06.

Tento program je volně šiřitelný s GNU licencí.

Ve zdrojovém kódu je zde uveden, protože pro účely této práce musí být upraven a překompilován, aby podporoval větší pole, než jaká poskytuje standardně. Tato úprava a kompilace je provedena příkazem spuštěným z adresáře **overeni/**:

```
$ make bc
```

Tento příkaz je proveden automaticky. Zmíněnou opravu realizuje **sed**-ový skript **fix.sed** spuštěný na soubor **consts.h** ve zdrojových kódech programu **bc**.

- **vystupy**

  Obsahuje uložené výstupy pro jednotlivé ověřovací skripty, což bude u každého skriptu popsáno zvlášť.

Dále se v tomto adresáři nacházejí testovací skripty, které poskytují rozhraní pro snadné ověření správnosti prezentovaných výsledků. Výstupy těchto skriptů pro některé parametry jsou uloženy v adresáři **vystupy/**. Pokud se v důsledku kopírování ztratila spouštěcí práva, tak příkazem

```
$ make
```

dojde k správnému nastavení přístupových práv a zkompilování potřebných programů. Jsou-li přístupová práva nastavena správě, tak tento příkaz není potřeba zadávat, nebot' je těmi testovacími skripty automaticky spuštěn. Tento příkaz je proveden jen při jeho prvním použití, aby se zbytečně neprováděly duplicitní operace. Připadné smazání vygenerovaných souborů může být provedeno příkazem

```
$ make clean
```

Nyní již následuje popis testovacích (ověřovacích) skriptů:

- **test-m3.sh**

  Ověřuje dolní odhad pro $m = 3$ počítače, že kompetitivní poměr musí být alespoň $\frac{12969}{9125}$, což vypíše jako optimální hodnotu řešeného lineárního programu. Spouští se příkazem

  ```
  $ ./test-m3.sh
  ```

  Lineární program je vygenerován programem **odhad-m3/glp**, jeho výstupem je textový popis matice. Výstup je transformován filtrem **bin/tab2bc.awk** do formátu jazyku **BC** a dále je připojen skript simplexové metody **simplex/simplex.bc0**. Výsledek je interpretován interpreterem jazyka **BC**, tj. program **bin/bc**.

  V adresáři **vystupy/test-m3** se nachází již spočtený výstup z tohoto skriptu. Výstup je v souboru **test-m3.txt**, řádek obsahující výsledek byl pro přehlednost uložen i do souboru **vysledky.txt**.

  Výstup lze v tomto adresáři (se souborem s výstupem) přegenerovat příkazem

  ```
  $ make
  ```

- **test-m.sh**

  Předvádí základní verzi důkazu obecně pro m počítačů, tedy spočte dolní odhad kompetitivního poměru vyplývající z lineárního programu sestrojeného s předpokladem existence σ_m-kompetitivního algoritmu. Číslo m dostane tento skript jako parametr, může být (z tohoto adresáře) spuštěn například příkazem

```
$ ./test-m.sh 4
```

který spočte odhad pro $m = 4$ a vypíše jej na výstup. Na výstup je rovněž vypsán lineární program, údaje potřebné pro ověření optimality a informace o průběhu výpočtu simplexové metody.

Zmíněný lineární program je vygenerovaný programem **odhad-m3/glpm**, jehož výstupem je textový popis matice, dále je použit filtr **bin/tab2bc.awk**. Získaný výstup je spojen se skriptem simplexové metody **simplex/simplex.bc0** a výsledek je interpretován programem **bin/bc**.

V adresáři **vystupy/test-m** se nacházejí předem spočtené výstupy pro $m = 2, 3, 4, 5, 6$ v souborech **test-m2.txt**, $\ldots$, **test-m6.txt**. Výsledky jsou shrnuty v souboru **vysledky.txt**. Tyto výsledky mohou být z tohoto adresáře přegenerovány příkazem

```
$ make
```

Z těchto spočtených výsledků je zajímavý jen odhad pro $m = 3$, což je $\frac{12969}{9125}$, ostatní jsou rovny odhadu σ_m.

- **test-silnejsi.sh**

  Tento skript ověřuje důkaz v kapitolce 3.4 o silnější verzi důkazu. Tedy cílem je rozhodnout o řešitelnosti lineárního programu sestaveného na základě předpokladu existence σ_m-kompetitivního algoritmu. Parametry lineárního programu jsou m počet počítačů a p počet nadkritických úloh ve vstupní posloupnosti (viz. její definice). Čísla m a p jsou vyžadována jako parametry tohoto skriptu. Skript může být použit například příkazem

```
$ ./test-silnejsi.sh 4 2
```

který provede výpočet pro $m = 4$ počítače a $p = 2$ nadkritické úlohy.

Lineární program je sestrojen programem **silnejsi/silnejsi**, jeho zdrojový kód v jazyku **C** je v souboru **silnejsi/silnejsi.c**. Výstupem tohoto program je popis tohoto lineárního programu, ten je transformován filtrem **bin/tab2xtab.awk** v lineární program mající optimální řešení 0 právě, když původní má přípustné řešení, viz. tvrzení 2.9.3. Dále po transformaci filtrem **bin/tab2bc.awk** a připojení kódu **simplex/simplex.bc0** je získaný program v jazyku **BC** interpretován programem **bin/bc**.

V adresáři **vystupy/test-silnejsi** jsou uloženy výstupy tohoto skriptu pro některé hodnoty m a p. Pro $m = 3$ a $p = 1$ je výstup uložen v souboru **test-m3-p1.txt**. Optimální hodnoty lineárních programů jsou opět shrnuty v souboru **vysledky.txt** a mohou být přegenerovány z tohoto adresáře příkazem

```
$ make
```

Spočtená optimální hodnota pro $m = 3$ a $p = 1$ je $\frac{2056}{6403}$. Tudíž pro původní lineární program nemá řešení a tedy neexistuje σ_m-kompetitivní algoritmus pro $m = 3$. Pro ostatní zkoušené parametry je spočtené optimum 0.

- **test-silnejsi-hod.sh**

  Tento skript provádí téměř totéž co skript **test-silnejsi.sh**, avšak je vynecháno použití filtru **bin/tab2xtab.awk**, tedy přímo je počítána optimální hodnota lineárního programu.

Výsledky jsou v adresáři **vystupy/test-silnejsi-hod**, pro $m = 3$ a $p = 1$ lineární program nemá řešení a jinak je získaný odhad roven hodnotě σ_m.

- **test-na2.sh**

  Tento skript počítá dolní odhady kompetitivního poměru pro pravděpodobnostní on-line algoritmy pro rozvrhování umísťující příchozí úlohy na dva počítače. Podrobnosti lze nalézt v kapitole 4. Tento skript má následující parametry, které musí být uvedeny v tomto pořadí:

  - m počet počítačů
  - i číslo počítače

    Zkoumaný algoritmus umísťuje příchozí úlohy na minimální a i-tý maximáľní počítač.

  - *st* zjemnění

    Určuje, na jak malé intervaly má být interval $\left\langle 0, \frac{1}{m} \right\rangle$ rozkouskován pro zpřesnění výpočtu. Intervaly budou mít velikost nejvýše $\frac{1}{xst \cdot m^{st}}$. Tento parametr není povinný, není-li uveden, použije se defaultní hodnota $st = 2$.

  - *xst* zjemnění

    Spolu s *st* určuje zjemnění. Není povinný, avšak je-li uveden, pak vždy následuje *st*, jeho defaultní hodnota je $xst = 1$.

  - L příznak lepení

    Tento příznak určuje, zda má být použita fáze zpětného slepování intervalů po tom, co jsou spočteny koeficienty lineárního programu. Při slepování jsou intervaly spojovány na maximální velikost nejvýše však $\frac{1}{xst \cdot m^{st}}$. Není-li tento parametr uveden, potom tato fáze není provedena.

Příklady možného použití jsou následující:

```
$ ./test-na2.sh 3 1 4
$ ./test-na2.sh 4 2 4 2 L
$ ./test-na2.sh 6 4 0 16 L
```

Tento skript napřed vyhodnotí zadané parametry skriptu. Podle těchto parametrů je spuštěn skript **na-dva/gen.bc**, který vygeneruje lineární program odhadující kompetitivní poměr. Protože tento program vypisuje i nějaké jiné údaje, tak lineární program je z jeho výstupu získán filtrem **bin/g2t.awk**. Lineární program je transformován do jazyka **BC** filtrem **bin/tab2bc.awk**, je připojen skript simplexové metody **simplex/simplex.bc0** a výsledný program je interpretován programem **bin/bc**.

Spočtené výsledky jsou uloženy v adresáři **vystupy/test-na2**. Název souborů s výstupem odpovídá zadaným parametrům skriptu, název začíná vždy **test-** a končí **.txt** a mezi jsou jednotlivé parametry oddělené znakem **-**. Například pro výše uvedené příklady to jsou soubory **test-3-1-4.txt, test-4-2-4-2-L.txt, test-6-4-0-16-L.txt**. Odhady jsou shrnuty v souboru **vysledky.txt**, podle nichž byla vytvořena tabulka 4.1.

Výsledky je možné vygenerovat příkazem

```
$ make
```

avšak je třeba počítat s dlouhým časem výpočtu. Prezentované výsledky byly počítány paralelně na více počítačích.

Poznámka: *Rychlost výpočtu velmi záleží na rychlosti použitých počítačů a může se pohybovat od několika hodin až po několik desítek hodin.*

Kapitola 6

Závěr

Tato práce víceméně splnila všechny předem zvolené cíle. Práce obsahuje základní fakta z teoretické informatiky k pravděpodobnostním (on-line) algoritmům. Kromě toho historický úvod mapuje vývoj v této oblasti od počátků vědního oboru teoretická informatika.

Snažil jsem se o čitelnost a snadnou pochopitelnost této práce i čtenářem bez hlubších znalostí tohoto oboru. K tomu mě vedlo to, že problém rozvrhování pomocí pravděpodobnostních on-line algoritmů je velmi speciální podobor teoretické informatiky, se kterým vůbec není seznámena většina lidí zabývajících se teoretickou informatikou. Práce je proto budována od elementárních základů až po komplikovaná tvrzení a zavádí potřebný formalismus sjednocující různorodé formalismy používané v literatuře pro jednotlivé podproblémy.

Jádro práce však tvoří nové výsledky dokázané v této práci, jež jsou podrobněji shrnuty v závěrech jednotlivých kapitol. Hlavně se podařilo dokázat, že neexistuje $\frac{27}{19}$-kompetitivní pravděpodobnostní on-line algoritmus pro rozvrhovaní na $m = 3$ počítače. Spolu s tím je vybudována potřebná teorie nezávislá na počtu počítačů a použitelná v případném zobecnění důkazu.

Dalším novým výsledkem jsou dolní odhady kompetitivního poměru pravděpodobnostních on-line algoritmů umísťujících příchozí úlohy na dva počítače. Tento výsledek je zcela původní, neboť zatím se pouze Seiden [14] zabýval konstrukcí takových algoritmů, tedy horními odhady této třídy algoritmů. Tato třída přináší další zjednodušení problému, avšak i přes toto zjednodušení zůstává hodně těžkým problémem.

Získané dolní dolní odhady mohou pomoci při návrhu a konstrukci nových algoritmů řešících problém rozvrhování. Spočtené výsledky proto mohou být považovány za přínos pro tento obor.

Literatura

[1] S. Albers: *Better Bounds for On-Line Scheduling.* Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, 130-139, 1997

[2] Y. Bartal, A. Fiat, H. Karloff, R. Vohra: *New algorithms for an ancient scheduling problem.* Proceedings of the 24th Annual ACM Symposium on the Theory of Computing, 51-58, 1992

[3] Y. Bartal, H. Karloff, Y. Rabani: *A better lowerbound for on-line scheduling.* Information Processing Letters 50,3:113-116, 1994

[4] B. Chen, A. van Vliet, G. Woeginger: *A better lowerbound for randomized on-line scheduling algorithms.* Information Processing Letters 51,5:219-222, 1994

[5] B. Chen, A. van Vliet, G. Woeginger: *New lowe and upper bounds for on-line scheduling.* Operations Research Letters 16,4,221-230, 1994

[6] L. Epstein, J. Sgall: *A Lower Bound for On-Line Scheduling on Uniformly Related Machines.* 1999

[7] A. Fiat, G. J. Woeginger (Eds.): *Online algorithms - The state of the Art.* Springer, Berlin 1998

[8] R. L. Graham: *Bounds for certain multiprocessing anomalies.* Bell System Technical Journal 45,1563-1581, 1966

[9] L. Grygarová: *Úvod do lineárního programování.* SPN, Praha 1975

[10] D. S. Johnson: *Near-optimal bin packing algorithms.* PhD. thesis, MIT, Cambridge, MA, 1973

[11] H. A. Kierstead, W. T. Trotter: *An extremal problem in recursive combinatorics.* Congressus Numerantium, 33:143-153, 1981

[12] D. E. Knuth: *The Art of Computer Programming.* Addison-Wesley, první vydání, druhý svazek, 1968

[13] M. Nelson, J.-L. Gailly: *The Data Compression Book.* M&T Books, New York 1996

[14] S. Seiden: *Randomized Algorithms for that Ancient Scheduling Problem.* Proceedings of the 5th Workshop on Algorithms and Data Structures, 210-223, 1997

[15] J. Sgall: *A Lower Bound for Randomized On-Line Multiprocessor Scheduling.* Information Processing Letters 63-1:51-55

[16] D. D. Sleator, R. E. Tarjan: *Amortized efficiency of list update and paging rules.* Communications of the ACM, 28:202-208, 1985

[17] D. D. Sleator, R. E. Tarjan: *Self-adjusting binary search trees.* Journal of the ACM, 32:652-686, 1985

[18] D. R. Woodal: *The bay restaurant - a linear storage problem.* American Mathematical Monthly, 81:240-246, 1974

[19] A. C. C. Yao: *New algorithms for bin packing.* Journal of the ACM, 27:207-227, 1980