Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Ondřej Zajíček

Algoritmy pro rozvrhování s konflikty

Matematický ústav AV ČR
Vedoucí diplomové práce: Doc. RNDr. Jiří Sgall, DrSc.

Studijní program: Informatika, teoretická informatika

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

V Praze dne Ondřej Zajíček

Obsah

Název práce: Algoritmy pro rozvrhování s konflikty
Autor: Ondřej Zajíček
Katedra (ústav): Katedra aplikované matematiky
Vedoucí diplomové práce: Doc. RNDr. Jiří Sgall, DrSc.
e-mail vedoucího: sgall@math.cas.cz
Abstrakt: Problém rozvrhování s konflikty předpokládá graf konfliktů, jehož vrcholy reprezentují stroje a hrany reprezentují konflikty mezi nimi. Každý stroj může být vypnutý nebo zapnutý. Stroje, které jsou v konfliktu, nemohou být zároveň zapnuté. Na jednotlivé stroje čas od času přicházejí úlohy a řadí se do vstupních front jednotlivých strojů. Zapnuté stroje průběžně úlohy zpracovávají a vyprazdňují tak své fronty. Algoritmus rozhoduje o vypínání a zapínání jednotlivých strojů, přičemž musí dodržet omezení vyplývající z grafu konfliktů. Cílem je najít takový rozvrh, který minimalizuje maximální dosaženou délku vstupních front. Problém je online, algoritmus tedy musí reagovat průběžně, přičemž neví, jaké úlohy se objeví později. Navrhl jsem algoritmus založený na maximalizaci skalárního součinu pracovního vektoru (vektoru, reprezentujícího konfiguraci jednotlivých strojů) a vektoru délek front. V této práci dokazuji, že tento algoritmus je dobře definován, je vždy konečný a pro konkrétní graf (cestu délky 3) je jeho kompetitivní poměr 7/3. Dále v práci rozebírám možnosti implementace tohoto algoritmu.
Klíčová slova: rozvrhování, konflikty, online algoritmy

Title: Algorithms for scheduling with conflicts
Author: Ondřej Zajíček
Department: Department of Applied Mathematics
Supervisor: Doc. RNDr. Jiří Sgall, DrSc.
Supervisor's e-mail address: sgall@math.cas.cz
Abstract: Scheduling with conflicts supposes graph of conflicts. Vertices of that graph represent machines and edges represent conflicts between them. Every machine can be switched on or switched off. Two conflicting machines cannot be both switched on at the same time. At certain times new tasks arrive to specific machines and enqueue to its input buffers. Each machine continuously processes tasks from its input buffer whenever it is switched on. An algorithm decides which machines should be switched on or switched off at any time, obeying conflict constraints. The objective is to schedule machine switching to minimize the maximum buffer size of all processors. The problem is online, so an algorithm has to make decisions about current configuration without knowledge of future tasks. In this thesis I consider the algorithm based on maximization of scalar product of work vector (vector describing configuration of machines) and vector of buffer lengths. I prove that this algorithm is well defined, finite on every input and for specific graph (path of length 3) it has competitive ratio of 7/3. Further I consider possibilities of implementation of that algorithm.
Keywords: scheduling, conflicts, online algorithms

1 Úvod

Rozvrhovací problémy je souhrnné označení pro skupinu optimalizačních problémů, které mají základní schéma tohoto typu: Je zadaná množina strojů a množina úloh a je třeba pro každou úlohu najít stroj, který ji zpracuje. Zpracování každé úlohy na daném stroji trvá nějaký čas, po který je zpravidla daný stroj nepoužitelný pro ostatní úlohy. Přiřazení strojů jednotlivým úlohám se označuje jako rozvrh. Jestliže rozvrh splňuje omezující podmínky (například že v jeden čas na jednom stroji se zpracovává právě jedna úloha), tak se označuje jako korektní. Korektních rozvrhů je obvykle mnoho, cílem rozvrhování tedy bývá nalézt rozvrh, který je optimální vzhledem k nějaké kriteriální funkci (typicky celková délka běhu).

Existuje mnoho variant rozvrhování. Pokud je možné čas zpracování dané úlohy na daném stroji vyjádřit jako podíl délky úlohy a rychlosti stroje, pak se takové rozvrhování nazývá v anglické literatuře *scheduling on related machines* (nebo dokonce *scheduling on identical machines*, pokud jsou rychlosti všech strojů stejné). V opačném případě se nazývá *scheduling on unrelated machines*. Pokud je možné zpracovaní úlohy kdykoliv přerušit, stroj uvolnit a ve zpracování pokračovat později, pak se mluví o preemptivním rozvrhování. Rozvrhování může pracovat offline (algoritmus má přístup k celému zadání najednou) nebo online (algoritmus dostává informace o úlohách postupně a musí průběžně rozhodovat o rozvrhu). Mohou existovat různá omezení které úlohy je možné umístit na které stroje.

Rozvrhovací problém, který bude diskutován v této diplomové práci, se podstatně liší od velké části ostatních rozvrhovacích problémů. V tomto problému je dána množina strojů a množina podmínek, které dva stroje nemohou pracovat ve stejný okamžik — množina strojů tvoří vrcholy grafu a množina podmínek tvoří jeho hrany. Dále je zadaná posloupnost úloh. Úloha je popsaná svou velikostí, časem příchodu a určením, na kterém stroji má běžet. Příchozí úlohy se ukládají do front u jednotlivých strojů a pokud daný stroj pracuje, tak jsou postupně zpracovávány. Zpracování úlohy je možné kdykoliv přerušit a později se k ní vrátit.

Cílem algoritmu je najít takový rozvrh (plán udávající, která množina strojů bude v který časový okamžik pracovat), který minimalizuje délku nejdelší fronty (kde délka fronty je měřena jako součet délek úloh v dané frontě). Problém je formulován jako online, tedy algoritmus dostává postupně úlohy uspořádané podle času příchodu a než dostane informaci o úloze, musí být rozhodnuto o rozvrhu až do času příchodu dané úlohy. Po příchodu dané úlohy se tedy již příslušná počáteční část rozvrhu nesmí měnit.

Cílem této diplomové práce je navrhnout vhodné algoritmy pro řešení problému rozvrhování s konflikty, nalezené algoritmy jasně definovat a analyzovat jejich chování (konečnost, kompetitivní poměr) na různých třídách grafů, zejména na cestách. V současné době není známo mnoho algoritmů pro řešení tohoto problému a existující algoritmy (popsané zejména v [1]) jsou pro některé třídy grafů (například cesty a kružnice) nepoužitelné nebo nevhodné, jak ukazuje sekce 4.

1.1 Optimalizační a online problémy

Na problém je možné nahlížet jako na funkci z množiny vstupů do potenční množiny možných výstupů, která pro pevný vstup určuje množinu přípustných (feasible) řešení. Pokud navíc je zadaná funkce z kartézského součinu množiny vstupů a množiny výstupů do (například) $\mathbf{R}^+$ (nazývaná často cílová funkce) a cílem algoritmu je nalézt nejenom přípustné řešení, ale navíc s co možná největší (nebo nejmenší) hodnotou cílové funkce, pak se jedná o optimalizační problém.

Online problémy je souhrnné označení pro skupinu problémů, při kterých vstup je rozdělen na části a algoritmus musí vyprodukovat část svého výstupu dřív, než dostane k disposici další část vstupu. Příkladem takového problému může být třeba klasický rozvrhovací problém, kdy algoritmus postupně dostává úlohy a musí je ihned přiřadit nějakému stroji, ale dopředu neví, jaké úlohy přijdou později. Online problémy obvykle bývají zároveň i optimalizačními problémy, pak má smysl definovat kompetitivní poměr online problému a kompetitivní poměr online algoritmu.

Kompetitivní poměr online algoritmu ALG (vzhledem k nějakému pevnému optimalizačnímu problému, kde je cílem minimalizovat cílovou funkci F s oborem hodnot $\mathbf{R}^+$) je definován takto:

Definice 1.1 *Definuji* RATIO(ALG) *jako funkci algoritmu* ALG*:*

$$\mathrm{RATIO}(\mathrm{ALG}) = \sup_{X \in \mathrm{INSTANCES}} \left\{ \frac{F(X, \mathrm{ALG}(X))}{F(X, \mathrm{OPT}(X))} \right\}$$

Kde INSTANCES je množina všech instancí daného problému a OPT je optimální offline algoritmus (tedy optimální algoritmus, který má k přístup k celému zadání bez online omezení).

Kompetitivní poměr problému je pak infimum přes kompetitivní poměry všech online algoritmů, který daný problém řeší.

Na online problém se často pohlíží jako na hru, ve které spolu soupeří algoritmus a oponent. Oponent zná v každou chvíli stav algoritmu, ale algoritmus nezná stav oponenta. Algoritmus dostává postupně vstup a generuje postupně výstup, oponent taktéž zpracovává stejný vstup, ale navíc určuje, jaký vstup bude v následujícím kroku. Cílem oponenta je dosáhnout co možná největšího poměru mezi hodnotou cílové funkce na výstupu algoritmu a na výstupu oponenta (vzhledem k tahům algoritmu), Oponent zde vlastně zastupuje jak funkci optimálního offline algoritmu, tak funkci suprema z definice kompetitivního poměru algoritmu.

1.2 Značení

V následujícím textu budu obvykle skalární hodnoty značit malými písmeny, vektory značit velkými písmeny a množiny značit slovy z velkých písmen (např. DIRS). V textu se často vyskytují posloupnosti vektorů, a proto využívám dolního indexu na indexaci prvku v rámci posloupnosti, zatímco pro indexaci složky v rámci jednoho vektoru využívám dolní index v hranatých závorkách. Například V_i tedy značí i-tý vektor v rámci posloupnosti vektorů $V_1, V_2, \ldots$, zatímco $V_{[i]}$ značí i-tou složku vektoru V. Vektory zapisuji bez čárek mezi prvky, například nulový třídimenzionální vektor zapíši jako $(0\ 0\ 0)$. Standardní skalární součin vektorů V, W značím $\langle V, W \rangle$.

2 Definice problému

V úvodu jsem zde diskutovaný problém zběžně popsal. Ten popis však byl zamýšlen spíše pro získání celkové představy, pro účely analýzy problému bude třeba přesnější definice problému, která bude uvedena v této kapitole.

Nejdříve ale navážu na zběžný popis z úvodu a popíšu, v čem se liší od definice, která bude následovat. V popisu jsem zmiňoval, že nezpracované úlohy se ukládají do front. Protože však úlohy jsou přerušitelné a v rámci jednoho stroje zaměnitelné, tak není třeba uvažovat jednotlivé posloupnosti úloh ve frontách, ale stačí uvažovat pouze součet délek úloh pro každou frontu. Tyto hodnoty budu nadále sdružovat do vektoru. Obdobně budu sdružovat do vektoru úlohy, které se objeví ve stejný čas na různých strojích.

Aktuální konfiguraci strojů popisuje pracovní vektor. Ten pro každý stroj udává (pomocí čísla z intervalu $\langle 0, 1 \rangle$), jak intenzivně daný stroj pracuje. Zběžný popis z úvodu předpokládal, že v jeden okamžik mohou pracovat pouze stroje, které tvoří nezávislou množinu v grafu konfliktů. Přípustné pracovní vektory by tedy byly charakteristické vektory nezávislých množin z grafu konfliktů. Následující definice však umožňuje, aby mezi přípustné pracovní vektory patřily také jejich konvexní kombinace. Pokud bych tuto možnost neměl, tak ji mohu nasimulovat tím, že v rozvrhu budu opakovaně střídat jednotlivé nezávislé množiny (tvořící danou konvexní kombinaci) po krocích o velikosti $\epsilon\alpha_i$, kde α_i je příslušný koeficient konvexní kombinace a ϵ je libovolně malá konstanta. Délka rozvrhu a délky jednotlivých front pro takový rozvrh a pro původní rozvrh (obsahující přímo konvexní kombinace nezávislých množin) se bude lišit jen o nějaké libovolně malé ϵ'.

V následujících definicích nadále předpokládám pevný graf $G = (V, E)$ a $|V| = n$. Výraz I_j reprezentuje charakteristický vektor j-té nezávislé množiny v grafu G, který má celkem m nezávislých množin.

Definice 2.1 *Definuji* INSTANCES *jako množinu všech X splňujících:*

$$X = \Big(k, \{P_i\}_{i=1}^k, \{r_i\}_{i=1}^k\Big)$$
$$k \in \mathbf{N}; \quad \forall i \in \{1 \dots k\}: \ P_i \in \mathbf{R}_0^{+n}, r_i \in \mathbf{R}_0^+$$
$$r_1 = 0; \quad \forall i, j \in \{1 \dots k\}: \ (i < j) \rightarrow (r_i < r_j)$$

INSTANCES je množina všech instancí problému. Jednotlivá instance udává, kdy přijdou jaké nové úlohy na jednotlivé stroje. V čase r_i přijde i-tá úloha, popsaná vektorem délek P_i. Celkem je v instanci k úloh. Předpokládáme rostoucí časy příchodů a příchod první úlohy je v čase 0.

Definice 2.2 *Definuji množinu* DIRS *jako množinu všech D splňujících:*

$$\exists \alpha_1 \dots \alpha_n \in \langle 0, 1\rangle: \ \sum_{i=1}^m \alpha_i = 1 \wedge \sum_{i=1}^m \alpha_i I_i = D$$

DIRS je množina všech přípustných pracovních vektorů. Je to omezený konvexní mnohostěn, který má jako vrcholy jednotlivé nezávislé množiny grafu G.

Definice 2.3 *Definuji* SCHEDULES *jako množinu všech S splňujících:*

$$S = \Big(l, \{W_i\}_{i=1}^l, \{t_i\}_{i=1}^l\Big)$$
$$l \in \mathbf{N}; \quad \forall i \in \{1 \dots l\}: \ W_i \in \mathrm{DIRS}, t_i \in \mathbf{R}_0^+$$
$$t_1 = 0; \quad \forall i, j \in \{1 \dots l\}: \ (i < j) \rightarrow (t_i < t_j)$$

SCHEDULES je množina všech přípustných rozvrhů. Rozvrh udává, kdy a jak pracují jednotlivé stroje a tedy jakou má hodnotu pracovní vektor v kterýkoliv časový okamžik. V čase 0 se stroje nakonfigurují na pracovní vektor W_1, v čase t_i proběhne rekonfigurace strojů a pracovní vektor se změní na W_i. Celkový počet rekonfigurací (včetně úvodní konfigurace) je l. Předpokládáme rostoucí časy rekonfigurací.

Definice 2.4 *Definuji* $\mathrm{LOAD}(X, S, t)$ *jako funkci instance* X*, rozvrhu* S *a nezáporného času* t*. V definici používám značení složek* X *a* S *podle definic instance a rozvrhu (*k*,* P_i*,* r_i*,* l*,* W_i *a* t_i*):*

$$\begin{aligned}
\mathrm{LOAD}(X,S,0) &= P_1 \\
\text{a pro } t > 0\text{:} & \\
\mathrm{LOAD}(X,S,t) &= \left(A' - (t-t')\,W_j\right)^+ + P \\
\text{kde: } t' &= \max\left\{x \mid x \in \{r_i\}_{i=1}^{k} \cup \{t_i\}_{i=1}^{l} \wedge x < t\right\} \\
A' &= \mathrm{LOAD}(X,S,t') \\
j &= \operatorname*{argmax}_{i\in\{1\ldots l\}} \{t_i \mid t_i < t\} \\
P &= \begin{cases} P_i, & \text{když } \exists i\colon r_i = t \\ \mathbf{0} & \text{jinak} \end{cases}
\end{aligned}$$

$\mathrm{LOAD}(X, S, t)$ je funkce, která pro danou instanci X, rozvrh S a čas t vrátí vektor délek front v čase t. Definice je provedena rekurzivně a je korektní, neboť pro libovolný pevný čas t existuje konečně mnoho zanoření rekurze — funkce se při rekurzi odkazuje jen na striktně menší časy z konečné množiny. Funkce využívá čas poslední události t' před požadovaným časem, vektor délek front A' v předchozím čase, aktuální pracovní vektor W_j a případný vektor délek příchozí úlohy P_i, pokud je aktuální čas t časem příchodu nějaké úlohy.

Definice 2.5 *Definuji* $\mathrm{BUFLEN}(X, S)$ *jako funkci instance* X *a rozvrhu* S*:*

$$\mathrm{BUFLEN}(X,S) = \max_{t\in\langle 0,\infty)} \left\{|\mathrm{LOAD}(X,S,t)|_\infty\right\}$$

$\mathrm{BUFLEN}(X, S)$ je funkce, která pro danou instanci X a rozvrh S vrátí maximální délku fronty, které bylo během zpracování X dosaženo. Tato funkce je cílová funkce problému, algoritmus pro zadané X hledá S takové, které minimalizuje $\mathrm{BUFLEN}(X, S)$.

Definice 2.6 *Definuji* $\mathrm{OPT}(X)$ *jako funkci instance* X*:*

$$\mathrm{OPT}(X) = \operatorname*{argmin}_{S \in \mathrm{SCHEDULES}} \{\mathrm{BUFLEN}(X, S)\}$$

$\mathrm{OPT}(X)$, uveden zde pro úplnost, reprezentuje optimální offline algoritmus.

Definice 2.7 *Definuji množinu* $\mathrm{INSTANCES}^-$*:*

$$\mathrm{INSTANCES}^- = \{X \in \mathrm{INSTANCES} \mid \mathrm{BUFLEN}(X, \mathrm{OPT}(X)) = 1\}$$

$\mathrm{INSTANCES}^-$ je množina instancí pro omezený problém, který se liší od základního problému (rozvrhování s konflikty) pouze v tom, že množina instancí problému je omezena na ty instance, pro které optimální offline algoritmus potřebuje fronty o velikosti právě 1. Vzhledem k tomu, že každou netriviální instanci základního problému je možné seškálovat do instance omezeného problému, tak každý algoritmus pro omezený problém odpovídá nějakému algoritmu pro základní problém, kterému navíc bude podána informace o potřebné délce fronty optimálního offline algoritmu. Definuji $\mathrm{buf}(G)$ jako kompetitivní poměr základního problému rozvrhování s konflikty pro pevný graf G a obdobně definuji $\mathrm{buf}^-(G)$ jako kompetitivní poměr omezeného problému pro pevný graf G.

3 Přehled existujících výsledků

[1] je významný článek týkající se tohoto problému. Jsou v něm uvedeny zejména tyto výsledky:

- Pro každý konečný graf G je $\mathrm{buf}(G)$ a $\mathrm{buf}^-(G)$ konečné.
- Pro každý úplný graf K_n je $\mathrm{buf}(K_n) = \mathrm{buf}^-(K_n) = H_n$ (kde H_n je n. harmonické číslo), pro každý úplný bipartitní graf $K_{m,n}$ je $\mathrm{buf}(K_{m,n}) = \mathrm{buf}^-(K_{m,n}) = 2$ a pro úplné k-partitní grafy je jejich $\mathrm{buf}(G)$ mezi H_k a $H_{k-1} + 1$. Tyto výsledky dosahuje jednoduchý hladový algoritmus.

  Hladový algoritmus (v článku [1] označován Greedy) je možné stručně popsat metodou výběru pracovní množiny — v každém okamžiku pracuje množina strojů vybraná tak, že z množiny kandidátů (která na začátku výběru pracovní množiny obsahuje všechny stroje s neprázdnou frontou) vybere stroj s nejdelší frontou a ten a všechny jeho sousedy vyřadí z množiny kandidátů. Tento krok se iteruje, dokud množina kandidátů je neprázdná. Tento stručný popis však neříká jak řešit případy, kdy v grafu je více sousedních vrcholů s nejdelšími frontami. Formální definice hladového algoritmu uvedena ve zmiňovaném článku tento problém řeší výběrem libovolného vrcholu a iterací po krocích, jejichž délka jde limitně k 0, není však jasné, zda tento postup na každém grafu konverguje.

- Pro každý strom T je $\mathrm{buf}(T) \leq D(T)/2 + 1$, kde $D(T)$ je poloměr stromu T. Tento výsledek je dosažen hierarchickým hladovým algoritmem.

  Hierarchický hladový algoritmus předpokládá pro graf G pevnou konečnou posloupnost indukovaných podgrafů $\{H_i\}_{i=1}^{p}$, kde $H_0 = G$ a H_{i+1} je indukovaný podgraf grafu H_i. Pro stromy je posloupnost definována tak, že H_{i+1} vznikne odstraněním všech vrcholů stupně 1 z grafu H_i. Hierarchický hladový algoritmus pracuje tak, že v každém kroku je pomocí algoritmu GREEDY nejdříve zvolena pracovní množina pro H_p, a pak pro každé i od $p-1$ do 1 je zvolena (pomocí modifikace GREEDY) pracovní množina pro H_i, která je nadmnožinou pracovní množiny pro H_{i+1}. Zvolená pracovní množina pro H_0 se pak použije jako pracovní množina pro G. Modifikace algoritmu GREEDY pak spočívá v tom, že modifikovaný algoritmus na začátku výběru pracovní množiny nemá v množině kandidátů všechny vrcholy grafu, ale jen ty, které nejsou prvkem zadané množiny nezávislých vrcholů, ani s ní nesousedí. K této zadané množině nezávislých vrcholů jsou pak iterativně přidávány další vrcholy z množiny kandidátů, jak je popsáno výše v popisu algoritmu GREEDY.

Velice zajímavý je článek [2], který ukazuje, jak je možné převést tento problém do podoby geometrické hry na honěnou. Tato hra je formulována takto: V zadaném metrickém prostoru se pohybuje zloděj a policista (body v prostoru). Maximální rychlost pohybu zloděje i policisty je stejná. Zloděj zná v každém okamžiku souřadnice policisty a kromě svého pohybu také ovládá návnadu. Návnada je bod v prostoru, jehož vzdálenost v každém okamžiku od zloděje musí být menší než nějaká pevná konstanta, jinak na pohyb návnady nejsou kladeny žádné požadavky. Policista nezná polohu zloděje, ale zná polohu návnady. Cílem zloděje je vzdálit se co možná nejdál od policisty, cílem policisty je udržovat alespoň pevnou vzdálenost od zloděje (když už ho nemá — vzhledem ke stejným rychlostem — šanci chytit).

Článek dokazuje, že pro eukleidovský prostor existuje strategie pro zloděje, která mu umožňuje vzdálit se libovolně daleko, zatímco pro prostor s metrikou definovanou konvexním mnohostěnem existuje strategie pro policistu udržující pevnou vzdálenost mezi ním a zlodějem. Mnohostěnová metrika je metrika odvozená z mnohostěnové normy, ve které všechny body na stěnách daného konvexního mnohostěnu (který musí být středově symetrický podle středu **0**) mají jedničkovou normu. Norma ostatních vektorů je dána linearitou normy.

Článek dále ukazuje, že problém rozvrhování s konflikty je možné snadno převést do této podoby. Stačí vzít n-rozměrný prostor (n je stále počet strojů) a metriku definovat mnohostěnem DIRS obohaceným o své symetrické kopie v ostatních kvadrantech. Souřadnice policisty pak budou dány odpracovanou dobou příslušných strojů algoritmu (každý stroj odpovídá příslušné ose v n-rozměrném prostoru), souřadnice návnady jsou dány celkovou délkou přišlých úloh příslušných strojů, a souřadnice zloděje jsou dány odpracovanou dobou příslušných strojů oponenta navýšenou o konstantu. V článku se tvrdí, že drobnými úpravami důkazu existence strategie udržující pevnou vzdálenost pro policistu je možné (pomocí této transformace) dokázat, že pro každý konečný graf G je $\mathrm{buf}(G)$ konečné.

Článek [3] se zabývá řízením přepínání paketů v počítačové síti a používá model, který odpovídá speciálnímu případu rozvrhování s konflikty, kde graf konfliktů je úplný graf. Na rozdíl od rozvrhování s konflikty je v tomto článku problém diskrétní a nazývá se *Balanced Scheduling Problem*. V článku je analyzován jednoduchý hladový algoritmus, který odpovídá algoritmu GREEDY z [1], a je o něm dokázáno, že má kompetitivní poměr $\Theta(\log n)$ a z tohoto hlediska je takřka optimální, neboť (jak je v článku dále dokázáno) dolní odhad kompetitivního poměru pro deterministické algoritmy je $1 + \lfloor \log_2 n \rfloor$ a pro randomizované algoritmy je $1 + \lfloor \log_2 n \rfloor / 2$.

Dále je v článku analyzován algoritmus *ROUND ROBIN*, který pravidelně střídá všechny stroje. O tomto algoritmu je dokázáno, že jeho kompetitivní poměr je n. Kromě *Balanced Scheduling Problem* je v článku ještě definován a řešen *Delay Balanced Scheduling Problem*, ve kterém se namísto celkové délky úloh ve frontě minimalizuje součet zpoždění (rozdíl času dokončení a času přijetí) jednotlivých úloh ve frontě, avšak tento problém již nemá přímou souvislost s tématem mé diplomové práce.

Článek [4] se také zabývá řízením přepínání paketů v počítačové síti, na rozdíl od [3] však používá spojitý model, který lépe odpovídá problému rozvrhování s konflikty. V článku je také analyzován jednoduchý hladový algoritmus (nazýván *Longest Queue First*) a o něm je dokázáno. že má kompetitivní poměr H_n a je optimální. Dále se autoři v článku zabývají konstrukcí efektivního offline algoritmu.

4 Algoritmus GREEDY

Algoritmus Greedy je jednoduchý algoritmus, který hladově volí pracovní množinu strojů. Tedy vždy vybere ten stroj z možných strojů, který má nejdelší frontu. Stručně jsem ho popsal na začátku minulé sekce a přesně definován je v [1], v drobných modifikacích také v [3] a [4]. Je o něm dokázáno, že na úplných grafech je jeho kompetitivní poměr H_n (podle [1], obdobně také podle [3] a [4]). Jaký je však kompetitivní poměr na jiných třídách grafů, zejména na cestách a kružnicích?

Algoritmus Greedy má jeden důležitý rys. Pokud algoritmus pro instanci X vygeneroval rozvrh S a během tohoto rozvrhu nedošlo k situaci, že některý stroj nepracoval jen z toho důvodu, že už měl prázdnou frontu, pak když upravíme instanci X na instanci X' zvětšením první úlohy o kladný násobek vektoru $\mathbf{1}$, tak algoritmus pro instanci X' vygeneruje rozvrh S', který se od S liší nejvýše v posledním pracovním vektoru (který zpracuje přidaný násobek vektoru $\mathbf{1}$).

Tento rys je způsoben tím, že algoritmus nevyužívá absolutní délku front, ale stará se pouze o to, která fronta je nejdelší, což bude po celou dobu běhu algoritmu stejně vycházet na X i na X', neboť délky odpovídajících front se budou lišit o stejnou konstantu. Toho může využít oponent — pokud získá nějaký náskok a rozloží ho rovnoměrně mezi své fronty, tak může stejný postup mnohokrát opakovat a získat tím libovolně velký náskok.

Následující příklad ukazuje, jak je toho možné dosáhnout na kružnici délky 6. Algoritmus tedy není kompetitivní (nemá konečný kompetitivní poměr) na tomto grafu. V tabulce je uveden úsek instance, na jehož počátku má algoritmus i oponent všechny fronty délky 2 (což je možné snadno docílit přidáním jedné úlohy) a po jeho konci má algoritmus stále všechny fronty délky 2, ale oponent má všechny fronty délky 1. Pak už stačí poslat úlohu (1 1 1 1 1 1) a, vzhledem k výše zmíněné vlastnosti algoritmu, postup libovolněkrát opakovat. Tento postup lze snadno modifikovat pro delší kružnice a cesty.

V tabulce 1 je uveden úsek instance (časy v prvním sloupku a vektory úloh v druhém) a úseky rozvrhů algoritmu a oponenta (časy v prvním sloupku a pracovní vektory v třetím a čtvrtém sloupku). Časy jsou uvedeny relativně k počátku úseku. Úsek trvá šest časových jednotek. V tabulce 2 jsou uvedeny délky front algoritmu (vektor A) a oponenta (vektor Z). V této tabulce jsou obvykle dva řádky po sobě se stejným časem — v daný čas přišla úloha a první řádek udává délky front před započtením této úlohy, zatímco druhý řádek udává délky front po jejím započtení.

Tabulka 1: Problémová instance a rozvrh algoritmu GREEDY a oponenta

čas	instance	algoritmus	oponent
0	(1 0 0 1 0 0)	(1 0 0 1 0 0)	(1 0 1 0 1 0)
1	(0 1 0 0 1 0)	(0 1 0 0 1 0)	(0 1 0 1 0 1)
2	(0 0 1 0 0 1)	(0 0 1 0 0 1)	(1 0 1 0 1 0)
3	(1 0 0 1 0 0)	(1 0 0 1 0 0)	(0 1 0 1 0 1)
4	(0 1 0 0 1 0)	(0 1 0 0 1 0)	(1 0 1 0 1 0)
5	(0 0 1 0 0 1)	(0 0 1 0 0 1)	(0 1 0 1 0 1)

Tabulka 2: Běh algoritmu GREEDY a oponenta na problémové instanci

čas	$A_{[1]}$	$A_{[2]}$	$A_{[3]}$	$A_{[4]}$	$A_{[5]}$	$A_{[6]}$	$Z_{[1]}$	$Z_{[2]}$	$Z_{[3]}$	$Z_{[4]}$	$Z_{[5]}$	$Z_{[6]}$
0	2	2	2	2	2	2	2	2	2	2	2	2
0	3	2	2	3	2	2	3	2	2	3	2	2
1	2	2	2	2	2	2	2	2	1	3	1	2
1	2	3	2	2	3	2	2	3	1	3	2	2
2	2	2	2	2	2	2	2	2	1	2	2	1
2	2	2	3	2	2	3	2	2	2	2	2	2
3	2	2	2	2	2	2	1	2	1	2	1	2
3	3	2	2	3	2	2	2	2	1	3	1	2
4	2	2	2	2	2	2	2	1	1	2	1	1
4	2	3	2	2	3	2	2	2	1	2	2	1
5	2	2	2	2	2	2	1	2	0	2	1	1
5	2	2	3	2	2	3	1	2	1	2	1	2
6	2	2	2	2	2	2	1	1	1	1	1	1

5 Algoritmus MAXLOAD

Dále se budu zabývat algoritmem MAXLOAD, což je algoritmus založen na myšlence, že v každém okamžiku by měl aktuální pracovní vektor maximalizovat lineární funkci $F_A(W) = \langle A, W \rangle = \sum_{i=1}^{n} A_{[i]} W_{[i]}$, kde vektor A je parametr udávající aktuální délku front ($A_{[i]}$ je délka fronty i-tého stroje). Protože tato představa vznáší otázky ohledně konvergence a nejasnosti ohledně efektivní implementace, tak algoritmus MAXLOAD formuluji jinak:

Algoritmus pracuje ve fázích. Na začátku každé fáze vybere vhodný pracovní vektor (postupem popsaným později), který maximalizuje $F_A(W)$, tento vektor (spolu s aktuálním časem) vypíše na výstup a dále spočítá (postupem popsaným později), jak dlouho může tato fáze maximálně trvat (tedy po jak dlouhou dobu bude vybraný vektor stále maximalizovat $F_A(W)$). Tato hodnota je vždy kladná a po této době fáze skončí. Fáze může také předčasně skončit příchodem další úlohy na vstup. Po skončení fáze ihned následuje další fáze (dokud nebude vyprázdněn vstup i jednotlivé fronty).

Jakým způsobem vybrat pracovní vektor? V první řadě je třeba si uvědomit, že množina přípustných pracovních vektorů DIRS je omezený konvexní mnohostěn. Funkce $F_A(X)$ je lineární, a tedy na omezeném konvexním mnohostěnu nabývá maxima na některé z jeho zobecněných stěn. První krok je tedy nalézt tuto stěnu — množinu MAX_A:

Definice 5.1 *Pro pevný vektor délek front A definuji c_A a množinu* MAX_A*:*

$$\begin{aligned} c_A &= \max_{X \in \mathrm{DIRS}} \langle A, X \rangle \\ \mathrm{MAX}_A &= \{X \in \mathrm{DIRS} \mid \langle A, X \rangle = c_A\} \end{aligned}$$

Množina MAX_A je zobecněná stěna mnohostěnu DIRS, a tedy je sama konvexním obalem některých vrcholů mnohostěnu DIRS. Nejjednodušší možnost, jak nalézt množinu MAX_A, je probrat všechny nezávislé množiny v grafu G, nalézt ty, pro jejichž charakteristické vektory W je $\langle A, W \rangle$ maximální, množina MAX_A je pak konvexní obal těchto vektorů.

Libovolný vektor z MAX_A je tedy vhodný kandidát na pracovní vektor podle základní představy algoritmu. Pokud však vezmeme v potaz požadavek, aby daný vektor byl optimální nejenom ihned, ale alespoň ještě nějaký neprázdný časový interval, pak je třeba vybrat vektor z následující podmnožiny:

Definice 5.2 *Pro pevný vektor délek front A definuji* STABLE_A *jako množinu všech X splňujících:*

$$X \in \mathrm{MAX}_A \quad \wedge \quad (\forall Y \in \mathrm{MAX}_A, \forall \epsilon > 0\colon\ \langle X, (A - \epsilon X) \rangle \geq \langle Y, (A - \epsilon X) \rangle)$$

Pokud bychom vybrali vektor $X \in \mathrm{MAX}_A$, který není v STABLE_A, pak po libovolně malém kroku ϵ by X bylo vyřazeno z $\mathrm{MAX}_{A'}$ (kde $A' = A - \epsilon X$), neboť by existovalo $Y \in \mathrm{MAX}_A$, pro které $\langle X, A' \rangle < \langle Y, A' \rangle$.

Lemma 5.3 *Pro každý vektor délek front A a reálný vektor X platí:*

$$X \in \mathrm{STABLE}_A \quad \Leftrightarrow \quad X \in \mathrm{MAX}_A \quad \wedge \quad (\forall Y \in \mathrm{MAX}_A\colon\ \langle X, X \rangle \leq \langle Y, X \rangle)$$

Důkaz:

$$\begin{array}{ll}
\forall Y \in \mathrm{MAX}_A, \forall \epsilon > 0\colon & \langle X, (A - \epsilon X)\rangle \geq \langle Y, (A - \epsilon X)\rangle \\
\forall Y \in \mathrm{MAX}_A, \forall \epsilon > 0\colon & \langle X, A\rangle \ - \langle X, \epsilon X\rangle \geq \langle Y, A\rangle \ - \langle Y, \epsilon X\rangle \\
\forall Y \in \mathrm{MAX}_A, \forall \epsilon > 0\colon & c_A - \langle X, \epsilon X\rangle \geq c_A - \langle Y, \epsilon X\rangle \\
\forall Y \in \mathrm{MAX}_A, \forall \epsilon > 0\colon & \epsilon \langle X, X\rangle \leq \epsilon \langle Y, X\rangle \\
\forall Y \in \mathrm{MAX}_A\colon & \langle X, X\rangle \leq \langle Y, X\rangle
\end{array}$$

■

Lemma 5.4 *Pro každý vektor délek front A a souřadnici i platí:*

$$A_{[i]} = 0 \quad \Rightarrow \quad \forall X \in \mathrm{STABLE}_A\colon\ X_{[i]} = 0$$

Důkaz: Nechť $A_{[i]} = 0$ a $X_{[i]} \neq 0$. Z definice DIRS plyne, že pro každý vektor V z DIRS a pro každou souřadnici j existuje v DIRS vektor V', který z V dostaneme vynulováním j-té souřadnice. Když $A_{[i]} = 0$, pak tedy pro každý vektor V z MAX_A existuje v MAX_A vektor V', který z V dostaneme vynulováním i-té souřadnice. Tedy i pro vektor X existuje takový vektor X'. Pak ale $\langle X, X\rangle > \langle X', X\rangle$, neboť všechny prvky sumy $\langle X, X\rangle$ až na i-tý prvek se shodují s prvky sumy $\langle X', X\rangle$ a i-tý prvek je v první sumě kladné číslo (neboť $X_{[i]} \neq 0$), zatímco v druhé sumě se rovná 0. To je však ve sporu s předchozím lemmatem. ■

Toto lemma ukazuje, že když budou za pracovní vektory vybírány vektory ze STABLE_A, tak A bude mít stále nezáporné hodnoty ve všech souřadnicích.

Lemma 5.5 *Pro každé dva reálné vektory X, Y platí:*

$$\langle X, Y\rangle \geq \langle X, X\rangle \quad \wedge \quad \langle X, Y\rangle \geq \langle Y, Y\rangle \quad \Rightarrow \quad X = Y$$

Důkaz:

$$\begin{array}{rcl}
\langle X, Y\rangle & \geq & \langle X, X\rangle \\
0 & \geq & \langle X, X\rangle - \langle X, Y\rangle = \langle X, X - Y\rangle \\
\langle X, Y\rangle & \geq & \langle Y, Y\rangle \\
0 & \geq & \langle Y, Y\rangle - \langle X, Y\rangle = \langle Y, Y - X\rangle = \langle -Y, X - Y\rangle \\
0 & \geq & \langle X, X - Y\rangle + \langle -Y, X - Y\rangle = \langle X - Y, X - Y\rangle \\
X & = & Y
\end{array}$$

■

Lemma 5.6 *Pro každý vektor délek front A a reálné vektory X, Y platí:*

$$X \in \mathrm{STABLE}_A \quad \wedge \quad Y \in \mathrm{STABLE}_A \quad \Rightarrow \quad X = Y$$

Důkaz: Plyne z lemmat 5.3 a 5.5. ∎

Lemma 5.7 *Pro každý vektor délek front A a reálný vektor X platí:*

$$X \in \mathrm{MAX}_A \quad \wedge \quad (\forall Y \in \mathrm{MAX}_A \colon \langle X, X\rangle \leq \langle Y, Y\rangle) \quad \Rightarrow \quad X \in \mathrm{STABLE}_A$$

Důkaz: Zvolme pevné Y různé od X a předpokládejme pro spor, že $\langle X, Y\rangle < \langle X, X\rangle$. Pak pro libovolné $\epsilon \in (0, 1)$:

$$\begin{aligned}
Z &\stackrel{\text{def}}{=} (1-\epsilon)\, X + \epsilon Y \\
\langle Z, Z\rangle &= \langle (1-\epsilon)\, X + \epsilon Y, (1-\epsilon)\, X + \epsilon Y\rangle \\
\langle Z, Z\rangle &= (1-\epsilon)^2 \langle X, X\rangle + 2\epsilon\,(1-\epsilon)\,\langle X, Y\rangle + \epsilon^2 \langle Y, Y\rangle \\
\langle Z, Z\rangle &= (1-\epsilon)^2 \langle X, X\rangle + (2\epsilon\,(1-\epsilon)\,\langle X, X\rangle - 2\epsilon\,(1-\epsilon)\,\langle X, X\rangle) + \\
&\quad\ 2\epsilon\,(1-\epsilon)\,\langle X, Y\rangle + \epsilon^2 \langle Y, Y\rangle \\
\langle Z, Z\rangle &= \left(1-\epsilon^2\right) \langle X, X\rangle - 2\epsilon\,(1-\epsilon)\,(\langle X, X\rangle - \langle X, Y\rangle) + \epsilon^2 \langle Y, Y\rangle \\
\langle Z, Z\rangle &= \langle X, X\rangle + \epsilon^2\,(\langle Y, Y\rangle - \langle X, X\rangle) - 2\epsilon\,(1-\epsilon)\,(\langle X, X\rangle - \langle X, Y\rangle) \\
\langle Z, Z\rangle &= \langle X, X\rangle + \epsilon^2\,(\Delta_1 + 2\Delta_2) - 2\epsilon\Delta_2
\end{aligned}$$

kde $\Delta_1 = \langle Y, Y\rangle - \langle X, X\rangle$ a $\Delta_2 = \langle X, X\rangle - \langle X, Y\rangle$. Δ_1 i Δ_2 jsou podle předpokladů kladné, takže je možné zvolit dostatečně malé ϵ, aby $\langle Z, Z\rangle < \langle X, X\rangle$, což je spor, neboť $Z \in \mathrm{MAX}_A$ z konvexity MAX_A. Takže pro každé Y platí $\langle X, Y\rangle \geq \langle X, X\rangle$ a závěr plyne z lemmatu 5.3. ∎

Na tento důkaz je možné nahlédnout trochu více geometricky: zvolíme-li pevné Y různé od X, pak úhel $YX\mathbf{0}$ je pravý nebo tupý, jinak by na úsečce XY ležel vrchol Z, pro který by platilo $\langle X, X\rangle > \langle Z, Z\rangle$ a z konvexity $Z \in \mathrm{MAX}_A$, což by bylo ve sporu s předpokladem. Z Kosinovy věty pak plyne $\langle Y, Y\rangle \geq \langle X, X\rangle + \langle Y - X, Y - X\rangle$, z čehož již snadno plyne $\langle X, Y\rangle \geq \langle X, X\rangle$.

Lemma 5.8 *Pro každý vektor délek front A platí:*

$$|\mathrm{STABLE}_A| = 1$$

Důkaz: MAX_A je konvexní mnohostěn a tedy $\langle X, X\rangle$ na něm nabývá minima. Podle lemmatu 5.7 je tedy tento vektor v STABLE_A. Podle lemmatu 5.6 je to jediný vektor v STABLE_A. ■

Výběr pracovního vektoru je tedy jasný — nejdříve projít všechny nezávislé množiny a najít vrcholy MAX_A a následně v MAX_A nalézt vektor s nejmenší normou (tedy vektor eukleidovsky nejblíže počátku souřadnic). Je ale ještě potřeba spočítat, po jak dlouhou dobu Δ bude setrvávat vybraný vektor W v množině $\mathrm{MAX}_{A-\Delta W}$

Definice 5.9 *Pro pevný nenulový vektor délek front A a pracovní vektor W definuji funkci $d_{A,W}(\epsilon, X)$, funkci $\delta_{A,W}(X)$, množinu $\mathrm{NEW}_{A,W}$ a konstantu $\Delta_{A,W}$:*

$$
\begin{aligned}
d_{A,W}(\epsilon, X) &= \langle A - \epsilon W, X\rangle = \langle A, X\rangle - \epsilon \cdot \langle W, X\rangle \\
\delta_{A,W}(X) &= \frac{\langle A, W\rangle - \langle A, X\rangle}{\langle W, W\rangle - \langle W, X\rangle} \\
\mathrm{NEW}_{A,W} &= \{X \in \mathrm{DIRS} \mid (\langle A, W\rangle > \langle A, X\rangle) \wedge (\langle W, W\rangle > \langle W, X\rangle)\} \\
\Delta_{A,W} &= \min_{X \in \mathrm{NEW}_{A,W}} \{\delta_{A,W}(X)\}
\end{aligned}
$$

$d_{A,W}(\epsilon, X)$ ukazuje, jaká bude hodnota funkce $F_{A'}(X) = \langle A', X\rangle$ (kde A' je budoucí hodnota vektoru délek front) za čas ϵ při vybraném pracovním vektoru W. $\delta_{A,W}(X)$ je výsledek řešení rovnice $d_{A,W}(\epsilon, X) = d_{A,W}(\epsilon, W)$ (vzhledem k neznámé ϵ). $\mathrm{NEW}_{A,W}$ je množina potenciálních pracovních vektorů, které mají šanci 'sesadit' W z pozice pracovního vektoru a $\Delta_{A,W}$ je čas, dokdy k tomu určitě nedojde. Protože je požadováno, aby A byl nenulový, tak $\mathrm{NEW}_{A,W}$ je neprázdná (neboť obsahuje $\mathbf{0}$) a $\Delta_{A,W}$ je tedy konečná.

Lemma 5.10 *Pro pevný nenulový vektor délek front A a pracovní vektor W platí:*

$$\forall X \in (\mathrm{DIRS} \setminus \mathrm{NEW}_{A,W})\ \forall \epsilon > 0: \quad d_{A,W}(\epsilon, X) \leq d_{A,W}(\epsilon, W)$$

Důkaz: Rozborem případů:

$\langle A, W\rangle = \langle A, X\rangle \rightarrow$ Pak $X \in \mathrm{MAX}_A$ a zbytek plyne z lemmatu 5.3

$\langle A, W\rangle < \langle A, X\rangle \rightarrow$ Tento případ nemůže nastat, neboť $W \in \mathrm{MAX}_A$

$\langle A, W\rangle > \langle A, X\rangle \rightarrow$ Pak tedy $\langle W, W\rangle \leq \langle W, X\rangle$ a tedy hodnota $d_{A,W}(\epsilon, X)$ při rostoucím ϵ klesá stejně rychle nebo rychleji než $d_{A,W}(\epsilon, W)$

■

Lemma 5.11 *Pro pevný nenulový vektor délek front A a pracovní vektor W platí:*

$$\forall \epsilon \in \langle 0, \Delta_{A,W} \rangle \ \forall X \in \mathrm{DIRS}: \quad d_{A,W}(\epsilon, X) \leq d_{A,W}(\epsilon, W)$$

$$\forall \epsilon \in \langle 0, \Delta_{A,W} \rangle : \quad W \in \mathrm{MAX}_{A-\epsilon W}$$

Důkaz: První tvrzení stačí dokázat jen pro $X \in \mathrm{NEW}_{A,W}$, pro ostatní X plyne z předchozího lemmatu. Pro pevné $X \in \mathrm{NEW}_{A,W}$ je $\langle A, W\rangle > \langle A, X\rangle$ a tedy $d_{A,W}(0, W) > d_{A,W}(0, X)$. $\delta_{A,W}(X)$ je pak výsledek řešení rovnice $d_{A,W}(\epsilon, X) = d_{A,W}(\epsilon, W)$. Pro libovolné ϵ z intervalu $\langle 0, \delta_{A,W}(X)\rangle$ tedy $d_{A,W}(\epsilon, X) \leq d_{A,W}(\epsilon, W)$ platí. Pokud vezmu $\Delta_{A,W}$ jako minimum z hodnot $\delta_{A,W}(X)$, pak to platí pro všechna požadovaná X. Druhé tvrzení triviálně plyne z prvního. ∎

Lemma 5.12 *Pro pevný nenulový vektor délek front A a pracovní vektor W platí:*

$$\forall \epsilon \in (0, \Delta_{A,W}) : \quad \mathrm{MAX}_{A-\epsilon W} \subseteq \mathrm{MAX}_A$$

$$\forall \epsilon \in (0, \Delta_{A,W}) : \quad \mathrm{STABLE}_{A-\epsilon W} = \mathrm{STABLE}_A$$

Důkaz: První tvrzení se dokazuje obdobně jako předchozí dvě. Pokud vektor X není v MAX_A ani v $\mathrm{NEW}_{A,W}$, pak se do $\mathrm{MAX}_{A-\epsilon W}$ nemůže dostat, neboť $d_{A,W}(\epsilon, X)$ při rostoucím ϵ klesá stejně rychle nebo rychleji než $d_{A,W}(\epsilon, W)$. Pokud vektor X není v MAX_A, ale je v $\mathrm{NEW}_{A,W}$, pak se do $\mathrm{MAX}_{A-\epsilon W}$ nedostane dříve než při $\epsilon = \delta_{A,W}(X)$, což není menší než $\Delta_{A,W}$. Druhé tvrzení plyne z prvního tvrzení a z předchozího lemmatu. ∎

Tato lemmata tedy ukazují, že doba, po kterou algoritmus může mít zvolen vektor W, je alespoň $\Delta_{A,W}$. Ačkoliv $\delta_{A,W}(X)$ není lineární funkce, tak musí minima (hodnoty $\Delta_{A,W}$) nabývat na nějakém vrcholu DIRS, neboť pokud by nabývala minima v bodě V, který není vrchol, pak by to znamenalo, že bod V se jako první dostane do $\mathrm{MAX}_{A'}$, kde dříve nebyl. Ale $\mathrm{MAX}_{A'}$ je konvexní mnohostěn a jeho vrcholy jsou pouze některé nezávislé množiny z G, a tedy pokud se v něm objevil nový bod V, který není vrchol, tak se v něm musel objevit i nový vrchol.

Lemma 5.13 *Jestliže algoritmus má na začátku fáze nenulový vektor délek front A a zvolí pracovní vektor W a fáze nebude přerušena příchodem nové úlohy, pak fáze bude trvat čas $\Delta_{A,W}$ a pracovní vektor W' zvolený v následující fázi bude splňovat $\langle W', W'\rangle < \langle W, W\rangle$.*

Důkaz: Nechť V je vektor z $\mathrm{NEW}_{A,W}$, pro který platí $\delta_{A,W}(V) = \Delta_{A,W}$. Tedy vektor nabývající minima z definice $\Delta_{A,W}$. Podle lemmatu 5.12 může fáze trvat nejméně $\Delta_{A,W}$. Nechť $A' = A - \Delta_{A,W}W$. Zřejmě $\langle A', V\rangle = \langle A', W\rangle$ a tedy $V \in \mathrm{MAX}_{A'}$. Protože $V \in \mathrm{NEW}_{A,W}$ a tedy $\langle W, W\rangle > \langle W, V\rangle$, tak $W \notin \mathrm{STABLE}_{A'}$ a fáze tedy musí skončit. Protože však podle lemmatu 5.11 platí $W \in \mathrm{MAX}_{A'}$, tak nový pracovní vektor W' splňuje $\langle W', W'\rangle < \langle W, W\rangle$ (podle lemmat 5.6 a 5.7). ■

Věta 5.14 *Algoritmus vždy skončí po konečně mnoha fázích.*

Důkaz: Během běhu algoritmu přijde pouze konečně vstupních úloh. Stačí tedy ukázat, že každá posloupnost fází, během které nedošlo k příchodu nové úlohy, je konečná. Podle předchozího lemmatu v rámci jedné takové posloupnosti fází stále ostře klesá eukleidovská norma pracovního vektoru a protože mnohostěn DIRS má konečně mnoho stěn a tedy možných pracovních vektorů je jen konečně mnoho, tak každá taková posloupnost je konečná. ■

6 Implementace algoritmu MAXLOAD

Algoritmus MAXLOAD obsahuje tři implementačně náročné kroky — nalezení množiny MAX_A, vektoru $W \in \mathrm{STABLE}_A$ a určení času $\Delta_{A,W}$.

Množina MAX_A je konvexní mnohostěn a tedy při hledání této množiny stačí najít jen její vrcholy, což jsou nezávislé množiny v grafu. Jednoduchý algoritmus pro nalezení těchto vrcholů projde všechny nezávislé množiny a vybere ty, pro jejichž charakteristické vektory W je $\langle A, W\rangle$ maximální.

Nalezení vektoru W, pro který platí $W \in \mathrm{STABLE}_A$, je komplikovanější. Víme, že má jít o vektor s nejmenší normou mezi všemi vektory z MAX_A. W je konvexní kombinace vrcholů MAX_A. Využijeme následující tvrzení:

Lemma 6.1 *Nechť W je prvkem STABLE_A, $V_1 \dots V_p$ jsou některé (ne nutně všechny) vrcholy z MAX_A a W je jejich konvexní kombinací, existují tedy nějaké $\lambda_1 \dots \lambda_p$) splňující:*

$$\sum_{i=1}^{p} \lambda_i V_i = W \quad (1)$$

$$\sum_{i=1}^{p} \lambda_i = 1 \quad (2)$$

$$\forall i \in \{1 \dots p\} : \lambda_i > 0 \quad (3)$$

Pak platí:

$$\forall i \in \{1 \dots p\}: \ \langle W, W \rangle = \langle V_i, W \rangle$$

Důkaz:

$$\langle W, W \rangle = \sum_{i=1}^{p} \lambda_i \langle W, W \rangle \leq \sum_{i=1}^{p} \lambda_i \langle V_i, W \rangle = \langle W, W \rangle$$

První rovnost plyne z (2), druhá rovnost plyne z (1) a linearity skalárního součinu. Nerovnost uprostřed tedy nabývá rovnosti. Tato nerovnost je součet nerovností (přenásobených kladnými koeficienty) z lemmatu 5.5 (tedy pro každé V_i: $\langle W, W \rangle \leq \langle V_i, W \rangle$), tudíž všechny tyto nerovnosti nabývají rovnosti. ∎

Věta 6.2 *Nechť W je prvkem* STABLE_A*, $V_1 \dots V_p$ jsou některé (ne nutně všechny) vrcholy z* MAX_A*, splňují předpoklady lemmatu 6.1 a jsou dohromady lineárně nezávislé. Pak soustava rovnic*

$$\begin{aligned} \sum_{i=1}^{p} \langle V_j - V_{j+1}, V_i \rangle x_i &= 0 \qquad \text{pro } j \in \{1 \dots p-1\} \\ \sum_{i=1}^{p} x_i &= 1 \end{aligned}$$

má právě jedno řešení a toto řešení jsou právě koeficienty konvexní kombinace dávající W, tedy ty $\lambda_1 \dots \lambda_p$, jejichž existenci předpokládá lemma 6.1.

Důkaz: Podle předpokladu můžeme vyjít z rovností $\langle V_j, W \rangle = \langle W, W \rangle$ ze závěru lemmatu 6.1 a ty formulovat takto:

$$\sum_{i=1}^{p} \langle V_j, V_i \rangle x_i = e \qquad \text{pro } j \in \{1 \dots p\}$$

Na levé straně rovnosti je W vyjádřeno jako neznámá kombinace vektorů V_i a skalární součin roznásobil vzniklou sumu. e je konstanta rovna $\langle W, W \rangle$, jejíž hodnotu neznáme. Podle lemmatu 6.1 koeficienty λ_i (které reprezentují W jako konvexní kombinaci vrcholů V_i) tvoří řešení této soustavy rovnic.

Tato soustava má právě jedno řešení, neboť matici této soustavy je možné zapsat jako součin MM^T, kde M je matice, jejíž řádky tvoří vektory V_i. Matice M má hodnost p, součin MM^T hodnost zachovává a tedy matice soustavy rovnic (o rozměrech $p \times p$) je regulární. Pokud jednotlivé rovnice od sebe postupně odečteme a zbývající rovnici vyhodíme, tak dostaneme soustavu rovnic, ve které již nefiguruje neznámá konstanta e:

$$\sum_{i=1}^{p} \langle V_j - V_{j+1}, V_i \rangle \, x_i = 0 \qquad \text{pro } j \in \{1 \ldots p-1\}$$

Tato soustava má jednodimenzionální prostor řešení, neboť vznikla vyhozením jedné rovnice ze soustavy rovnic, která měla právě jedno řešení. Tento prostor samozřejmě stále obsahuje řešení tvořené koeficienty $\lambda_1 \ldots \lambda_p$. Jestliže k této soustavě přidáme ještě rovnici:

$$\sum_{i=1}^{p} x_i = 1$$

Pak výsledná soustava rovnic má právě jedno řešení, neboť tato nová rovnice zřejmě není lineární kombinací ostatních a řešení tvořené koeficienty $\lambda_1 \ldots \lambda_p$ je zřejmě řešením i této rovnice. ∎

Tato věta nám umožňuje nalézt W, pokud uhodneme, ze kterých vrcholů je možné ho složit jako konvexní kombinaci. Soustava rovnic pak neobsahuje jiné neznáme parametry a tudíž ji můžeme vyřešit a nalézt hodnoty $\lambda_1 \ldots \lambda_p$ a tedy W.

Jednoduchý algoritmus pro nalezení vektoru W je projít všechny lineárně nezávislé podmnožiny vrcholů MAX_A (kvůli nezávislosti je možné se omezit na ty množiny, jejichž kardinalita je menší nebo rovna n) a pro každou takovou množinu vyřešit výše zmíněnou soustavu rovnic a pokud je řešení validní (tedy hodnoty všech proměnných jsou větší než 0), tak spočítat konvexní kombinaci a získat kandidáta na W. Ze všech nalezených kandidátů pak vzít toho s nejmenší normou.

Protože W vždy existuje a je možné ho vyjádřit jako konvexní kombinaci n nebo méně vrcholů MAX_A, tak některá z probíraných podmnožin bude splňovat požadavky věty 6.2 a tedy příslušný kandidát bude W a tedy tento postup vždy najde vrchol W, který je prvkem STABLE_A.

Jednoduchý algoritmus pro určení času $\Delta_{A,W}$ projde všechny nezávislé množiny, pro každou vyhodnotí, zda patří do $\mathrm{NEW}_{A,W}$, a v takovém případě spočítá $\delta_{A,W}(X)$ a udržuje si nalezené minimum (a příslušnou nezávislou množinu). Toto minimum bude (podle poznámky ze závěru minulé sekce) $\Delta_{A,W}$.

6.1 Maximální nezávislé množiny

Pro tyto algoritmy se nabízí jedno vylepšení založené na myšlence, zda by nestačilo v těchto algoritmech procházet pouze maximální nezávislé množiny (maximální ve smyslu inkluze). S drobnými modifikacemi to možné je. Stačí si uvědomit, že nemaximální nezávislé množiny se berou v potaz pouze kvůli tomu, že některé fronty již mohou být prázdné. Pokud tomu tak je (vektor délek front A je v některé souřadnici nulový), pak vybraný pracovní vektor bude v příslušných souřadnicích také nulový (to tvrdí lemma 5.4) a naopak, pokud je i-tá fronta neprázdná, pak $\langle A, X\rangle > \langle A, X'\rangle$, kde X je charakteristický vektor nezávislé množiny obsahující i-tý vrchol a X' je charakteristický vektor množiny, která vznikla z předchozí odebráním i-tého vrcholu.

Definice 6.3 *Nechť A je vektor délek front, G' je indukovaný podgraf grafu G vzniklý odebráním všech vrcholů, které mají prázdné fronty (tedy odebráním množiny vrcholů $\left\{i \mid A_{[i]} = 0\right\}$), a $F(X)$ je funkce, která pro podmnožinu vrcholů grafu G vrátí její charakteristický vektor. Definuji množinu* MIS_A*:*

$$\mathrm{MIS}_A = \{F(X) \mid X \textit{ je maximální nezávislá množina v grafu } G'\}$$

Modifikovaný zmíněný algoritmus pro první krok (hledání vrcholů MAX_A) pracuje tak, že projde všechny prvky MIS_A vybere ty, pro jejichž charakteristické vektory W je $\langle A, W\rangle$ maximální. Tento postup sice nenajde všechny vrcholy MAX_A, ale ty nenalezené vrcholy MAX_A jsou podle lemmatu 5.4 nezajímavé, neboť se nepodílí na pracovním vektoru. Tím jsme zároveň urychlili druhý krok — namísto vrcholů MAX_A se v něm zpracovává menší množina vrcholů.

Zmíněný algoritmus pro třetí krok (určení času $\Delta_{A,W}$) se modifikuje obdobně — průchod přes DIRS se nahradí průchodem přes MIS_A. Je však třeba provést ještě jednu úpravu, neboť je třeba počítat s tím, že některá souřadnice v A se může v průběhu vyprázdnit, což by změnilo množinu MIS_A a tedy i výsledky algoritmu prvního kroku. Modifikovaný algoritmus tedy spočítá $\Delta'_{A,W}$ a $\Delta''_{A,W}$ (podle následující definice) a hledaný čas je ten menší z nich.

Definice 6.4

$$\begin{aligned}
\mathrm{NEW}'_{A,W} &= \{X \in \mathrm{MIS}_A \mid (\langle A, W\rangle > \langle A, X\rangle) \wedge (\langle W, W\rangle > \langle W, X\rangle)\} \\
\Delta'_{A,W} &= \min_{X \in \mathrm{NEW}'_{A,W}} \{\delta_{A,W}(X)\} \\
\Delta''_{A,W} &= \min_{i \in \{1 \ldots n\}, W_{[i]} \neq 0} \left\{A_{[i]}/W_{[i]}\right\}
\end{aligned}$$

6.2 Algoritmus pro cesty

Pro některé třídy grafů je možné najít specifické algoritmy pro jednotlivé kroky. Například pro cesty je možné hledat vrcholy MAX_A pomocí přímočarého algoritmu pracujícího v lineárním čase. Tento algoritmus bere v potaz poznámky o maximálních nezávislých množinách a stejně jako algoritmus z minulé podsekce negeneruje některé nepodstatné vrcholy MAX_A. Při popisu předpokládám pevný graf G jako cestu délky n, s přirozenými čísly $1 \dots n$ jako vrcholy, kde hrany spojují sousední čísla.

Definice 6.5 Pro přirozená čísla a, d splňující $1 \leq a \leq d \leq n$ definuji $\mathrm{IP}^{a,d}$ jako množinu obsahující charakteristické vektory všech těch nezávislých množin v grafu G, které obsahují vrcholy a a d, neobsahují vrcholy před a, ani vrcholy po d a mezi vrcholy a a d jsou maximální ve smyslu inkluze, tedy nelze do nich přidat žádný vrchol mezi a a d, aniž by přestaly být nezávislé.

Definice 6.6 *Pro pevný vektor délek front A a pevná přirozená čísla a, d, která splňují $1 \leq a \leq d \leq n$, definuji $m_A^{a,d}$ a množinu $\mathrm{MIP}_A^{a,d}$:*

$$m_A^{a,d} = \max_{X \in \mathrm{IP}^{a,d}} \{\langle A, X \rangle\}$$

$$\mathrm{MIP}_A^{a,d} = \left\{ X \in \mathrm{IP}^{a,d} \mid \langle A, X \rangle = m_A^{a,d} \right\}$$

Definice 6.7 Pro pevný vektor R, který je prvkem $\mathrm{IP}^{a,d}$ (pro nějaké a, d), a jemu odpovídající nezávislou množinu M definuji dělící bod vektoru R jako každou dvojici čísel (b, c), pro kterou platí $b < c$, $b \in M$, $c \in M$ a pro všechna i splňující $b < i < c$: $i \notin M$. Zřejmě platí $2 \leq c - b \leq 3$ kvůli maximalitě a nezávislosti M.

Definice 6.8 Rozdělení vektoru R ($R \in \mathrm{IP}^{a,d}$) podle dělícího bodu (b, c) je operace vracející vektory S a T. Vektor S vznikl z R vynulováním všech souřadnic ostře větších než b, Vektor T vznikl z R vynulováním všech souřadnic ostře menších než c. Vektory S a T zřejmě splňují $S \in \mathrm{IP}^{a,b}$ a $T \in \mathrm{IP}^{c,d}$.

Definice 6.9 Spojení vektorů S a T ($S \in \mathrm{IP}^{a,b}$, $T \in \mathrm{IP}^{c,d}$) je operace vracející vektor R. Tato operace je definovaná pouze pokud $2 \leq c - b \leq 3$. V tom případě $R = S + T$ (kde $+$ zastupuje disjunktní sjednocení odpovídajících nezávislých množin). Výsledný vektor R zřejmě splňuje $R \in \mathrm{IP}^{a,d}$. Spojení vektoru S s nulovým vektorem je dodefinováno jako vracející S.

K těmto definicím ještě poznamenejme, že rozdělení a spojení vektorů jsou inverzní operace, spojení vektorů je vlastně jen vektorový součet s omezeným definičním oborem a pro každý vektor A v obou případech platí $\langle A, R\rangle = \langle A, S\rangle + \langle A, T\rangle$.

Lemma 6.10 *Pro každý vektor délek front A, každý vektor $R \in \mathrm{IP}^{a,d}$ a vektory S, T vzniklé z R rozdělením podle libovolného jeho dělícího bodu (b, c), platí:*

$$R \in \mathrm{MIP}_A^{a,d} \quad \Rightarrow \quad S \in \mathrm{MIP}_A^{a,b} \quad \wedge \quad T \in \mathrm{MIP}_A^{c,d}$$

Důkaz: Z definice rozdělení plyne, že $S \in \mathrm{IP}^{a,b}$ a $T \in \mathrm{IP}^{c,d}$, takže stačí dokázat, že $\langle A, S\rangle$ a $\langle A, T\rangle$ jsou maximální. Pokud by některý z nich nebyl maximální, tedy by existoval $S' \in \mathrm{IP}^{a,b}$ nebo $T' \in \mathrm{IP}^{c,d}$ splňující $\langle A, S'\rangle > \langle A, S\rangle$ (respektive $\langle A, T'\rangle > \langle A, T\rangle$), pak spojením S' a T (respektive S a T') vznikne $R' \in \mathrm{IP}^{a,d}$ splňující $\langle A, R'\rangle > \langle A, R\rangle$ (podle vlastnosti spojení), což je ve sporu s předpokladem $R \in \mathrm{MIP}_A^{a,d}$. ∎

Věta 6.11 *Pro každý vektor délek front A a přirozená čísla a, b, která splňují $1 \leq a$, $a + 3 \leq b \leq n$, platí:*

$$m_A^{a,b} = \max\left\{m_A^{a,b-3}, m_A^{a,b-2}\right\} + m_A^{b,b}$$

Důkaz: Nechť R je libovolný vektor z $\mathrm{MIP}_A^{a,b}$. Tento vektor má (mimo jiné) jeden z dělících bodů $(b-3, b)$, $(b-2, b)$, neboť v příslušné nezávislé množině jsou mezi a a b mezery velikosti 1 nebo 2. Nechť c je tedy ta hodnota, aby (c, b) byl tento dělící bod. Nechť S a T jsou vektory vzniklé z R rozdělením podle tohoto dělícího bodu. Podle lemmatu 6.10 platí $S \in \mathrm{MIP}_A^{a,c}$ a $T \in \mathrm{MIP}_A^{b,b}$ a tedy:

$$m_A^{a,b} = \langle A, R\rangle = \langle A, S\rangle + \langle A, T\rangle = m_A^{a,c} + m_A^{b,b} \leq \max\left\{m_A^{a,b-3}, m_A^{a,b-2}\right\} + m_A^{b,b}$$

Opačná nerovnost se dokáže takto: Nechť c je prvek z $\{b-3, b-2\}$ takový, že $m_A^{a,c}$ nabývá maxima ze znění věty, S je libovolný vektor z $\mathrm{MIP}_A^{a,c}$, T je libovolný (ten jediný) vektor z $\mathrm{MIP}_A^{b,b}$ a R je vektor vzniklý spojením vektorů S a T. Zřejmě $R \in \mathrm{IP}^{a,b}$ a tedy:

$$m_A^{a,b} \geq \langle A, R\rangle = \langle A, S\rangle + \langle A, T\rangle = m_A^{a,c} + m_A^{b,b} = \max\left\{m_A^{a,b-3}, m_A^{a,b-2}\right\} + m_A^{b,b}$$

∎

Cílem algoritmu je najít všechny maximální nezávislé množiny z MAX_A. Protože tyto množiny mohou anebo nemusí obsahovat jednotlivé krajní vrcholy cesty G, tak je třeba nalézt $m_A^{1,n-1}$, $m_A^{1,n}$, $m_A^{2,n-1}$ a $m_A^{2,n}$, vzít z nich maximum m_A (které se rovná c_A z definice MAX_A) a sjednotit ty $\mathrm{MIP}_A^{a,b}$ (z těch čtyřech možných), jejichž $m_A^{a,b}$ je maximální.

Pro $b - a \leq 4$ je možné $m_A^{a,b}$ spočítat přímo, neboť množina $\mathrm{IP}^{a,b}$ je jednoprvková, anebo v případě $b - a = 1$ prázdná. Ostatní potřebné hodnoty se spočítají induktivně: pro rostoucí i z $\{6 \ldots n\}$ algoritmus spočítá $m_A^{1,i}$ a $m_A^{2,i}$ podle věty 6.11, přičemž si pamatuje tři poslední hodnoty v každé řadě: $\left\{m_A^{a,i-b} \mid a \in \{1 \ldots 2\}, b \in \{1 \ldots 3\}\right\}$. Tento algoritmus má zřejmě časovou složitost $O(n)$ a paměťovou složitost $O(1)$.

Jak generovat množinu $\mathrm{MIP}_A^{1,b}$? Pro $b \leq 5$ triviálně, pro větší b je možné využít tuto myšlenku: Protože každý vektor R, který je prvkem $\mathrm{MIP}_A^{1,b}$, je možné získat složením vektoru S, který je prvkem $\mathrm{MIP}_A^{1,b-3}$ nebo $\mathrm{MIP}_A^{1,b-2}$, a vektoru T, který je prvkem $\mathrm{MIP}_A^{b,b}$ (viz důkaz věty 6.11), tak stačí zjistit, pro které $c \in \{b-3, b-2\}$ platí $m_A^{1,b} = m_A^{1,c} + m_A^{b,b}$ a ke každému vektoru z $\mathrm{MIP}_A^{1,c}$ připojit jediný vektor z $\mathrm{MIP}_A^{b,b}$ (tedy vlastně přidat k nezávislé množině vrchol na konec), čímž se vygenerují všechny vektory z $\mathrm{MIP}_A^{1,b}$.

Pokud se algoritmus na hledání $m_A^{1,b}$ modifikuje, aby si zapamatoval všechna spočítaná $m_A^{1,b}$ (čímž se jeho paměťová složitost změní na $O(n)$), pak je tedy možné vygenerovat množinu $\mathrm{MIP}_A^{1,b}$ jednoduchým rekurzivním algoritmem, který pracuje pro zadané i (na počátku b) a sufix T (na počátku nulový vektor) takto:

- Jestliže $i \leq 5$, tak vypsat vektor, který vznikne spojením jediného pevného prvku množiny $\mathrm{IP}^{1,i}$ se sufixem T, a skončit.
- Nechť T' je nový sufix vzniklý spojením jediného pevného prvku množiny $\mathrm{IP}^{i,i}$ a T.
- Jestliže $m_A^{1,i} = m_A^{1,i-2} + m_A^{i,i}$, pak se rekurzivně zavolat s $i - 2$ a T'.
- Jestliže $m_A^{1,i} = m_A^{1,i-3} + m_A^{i,i}$, pak se rekurzivně zavolat s $i - 3$ a T'.
- Skončit.

Tento algoritmus má paměťovou složitost $O(n)$ a časovou složitost $O(n+q)$, kde q je velikost výstupu. Jiná možnost je algoritmus na hledání $m_A^{1,b}$ modifikovat, tak aby průběžně sestavoval orientovaný acyklický graf, na kterém jsou jednotlivé prvky $\mathrm{MIP}_A^{1,b}$ reprezentované cestami od listů ke kořeni. Tato reprezentace $\mathrm{MIP}_A^{1,b}$, která odpovídá struktuře rekurze v předchozím rekurzivním algoritmu, má paměťovou složitost $O(n)$ a takový algoritmus na generování $\mathrm{MIP}_A^{1,b}$ má tedy časovou i paměťovou složitost v $O(n)$.

Vraťme se k původnímu problému — hledání vrcholů MAX_A. Podle předchozí podsekce je tedy třeba nalézt ty vrcholy MAX_A, které jsou maximální ve smyslu inkluze, pokud se neberou v potaz ty souřadnice, na kterých je vektor A nulový, a naopak nalezené vektory mají mít tyto souřadnice nulové. Algoritmus popsaný v této sekci najde právě vrcholy MAX_A, které jsou maximální ve smyslu inkluze. Opravit ho, aby splňoval výše zmíněné požadavky, je možné dvěma způsoby: Buď se po dokončení výpočtu inkriminované souřadnice vynulují, nebo se původní graf (cesta) rozdělí na několik podcest vynecháním vrcholů, které odpovídají inkriminovaným souřadnicím, a na každou cestu se tento algoritmus pustí nezávisle. Výsledek pak vznikne zkombinováním dílčích výsledků z jednotlivých podcest.

7 MAXLOAD na 3-cestě

V této sekci dokážeme, že kompetitivní poměr algoritmu MAXLOAD na cestě délky 3 je 7/3. Dolní odhad bude demonstrován pomocí instance, na které MAXLOAD dosahuje tohoto poměru oproti offline algoritmu, horní odhad bude dokázán pomocí invariantů udávají vztah mezi délkou front algoritmu a oponenta (offline algoritmu).

Před analýzou chování na 3-cestě se však ještě zmíním, jak se algoritmus MAXLOAD chová na 2-cestě. V tomto případě je jeho chování stejné jako chování algoritmu Greedy, což je možné snadno ověřit rozborem případů. 2-cesta je úplný graf a Greedy tedy podle [1] je na ní 2-kompetitivní. Algoritmus MAXLOAD je tedy na 2-cestě 2-kompetitivní, což je optimální hodnota (podle [1]).

Aktuální délky jednotlivých front algoritmu budeme v této sekci značit písmeny a, b a c (kde b je délka fronty stroje, jemuž odpovídá prostřední vrchol v grafu konfliktů, a a c jsou délky front strojů, jimž odpovídají krajní vrcholy), aktuální délky front oponenta budeme značit obdobně písmeny a', b' a c'. Maximální délku nejdelší z front (tedy hodnotu funkce $\mathrm{BUFLEN}(X, S)$) oponenta budeme značit r. Platí tedy $a' \leq r$, $b' \leq r$, a $c' \leq r$.

Rozborem případů zjistíme, které všechny vektory mohou být vybrané jako pracovní a za jakých podmínek:

$$
\begin{array}{llll}
a+c<b & & & (0\ 1\ 0) \\
a+c>b & a\neq 0 & c\neq 0 & (1\ 0\ 1) \\
 & a\neq 0 & c=0 & (1\ 0\ 0) \\
 & a=0 & c\neq 0 & (0\ 0\ 1) \\
a+c=b & a\neq 0 & c\neq 0 & \left(\frac{1}{3}\ \frac{2}{3}\ \frac{1}{3}\right) \\
 & a\neq 0 & c=0 & \left(\frac{1}{2}\ \frac{1}{2}\ 0\right) \\
 & a=0 & c\neq 0 & \left(0\ \frac{1}{2}\ \frac{1}{2}\right) \\
 & a=0 & c=0 & (0\ 0\ 0)
\end{array}
$$

Lemma 7.1 *Po celou dobu běhu algoritmu platí:*

$$
\begin{aligned}
b &\leq b'+a+c && (4)\\
a &\leq a'+b && (5)\\
c &\leq c'+b && (6)
\end{aligned}
$$

Důkaz: Nejdříve dokážeme první invariant. Na začátku (kdy všechny fronty jsou prázdné) invariant platí. Pokud $b \leq a+c$, pak invariant triviálně platí. Pokud $b > a+c$, pak pracovní vektor algoritmu bude (0 1 0). V tom případě však b klesá maximální možnou rychlostí, a a c se nemění a b' nemůže klesat rychleji než b. Invariant tedy zůstává zachován i v tomto případě.

Důkaz druhého invariantu bude obdobný: Na začátku platí, pokud $a \leq b$, tak také triviálně platí, pokud $a > b$, pak pracovní vektor algoritmu bude (1 0 1) nebo (1 0 0) (podle toho, zda $c = 0$), a tedy klesá maximální možnou rychlostí a b se nemění. Invariant tedy zůstává zachován. Třetí invariant se dokáže symetricky podle druhého (přehodí se role a a c). ∎

Lemma 7.2 *Po celou dobu běhu algoritmu platí:*

$$
\begin{aligned}
a+b &\leq a'+b'+r && (7)\\
c+b &\leq c'+b'+r && (8)
\end{aligned}
$$

Důkaz: Důkazy obou invariantů je třeba provádět zároveň. Na začátku oba invarianty platí. Pokud by poprvé měl být některý z nich porušen (bez újmy na obecnosti předpokládejme první z nich), pak těsně před porušením se levá strana bude rovnat pravé a levá musí klesat pomaleji než pravá a tedy (v prvním případě) součet prvních dvou složek pracovního vektoru musí být menší než 1 (součet stejných složek pracovního vektoru u oponenta totiž může být až 1). To nastává na dvou možných pracovních vektorech: (0 1/2 1/2) a (0 0 1). V obou případech platí $a = 0$ a $b \leq c$. pak ale:

$$2(a' + b' + r) = 2\,(a + b) = 2b \leq c + b \leq c' + b' + r \leq b' + 2r$$

$$2a' + b' \leq 0$$

a tedy $a' + b' = 0$ a pravá strana tedy neklesá vůbec. Druhý případ (nejdříve porušen druhý invariant) je symetrický. ■

Lemma 7.3 *Po celou dobu běhu algoritmu platí:*

$$\begin{aligned} b &\leq 7/3\, r \\ a &\leq 2r \\ c &\leq 2r \end{aligned}$$

Důkaz: Součtem invariantů (4), (7) a (8) dostaneme $3b \leq 3b' + a' + c' + 2r$ a tedy zřejmě $3b \leq 7r$. Obdobně součtem (5) a (7) dostaneme $2a \leq 2a' + b' + r \leq 4r$ a součtem (6) a (8) dostaneme $2c \leq 2c' + b' + r \leq 4r$. ■

Lemma 7.4 *Existuje instance, na které algoritmus MAXLOAD potřebuje frontu délky* $7/3 - \epsilon$ *(pro libovolně malé* ϵ*), zatímco oponent si vystačí s frontou délky* 1.

Důkaz: Zápis běhu algoritmu a oponenta je v následující tabulce. Tabulka je rozdělena do třech částí. Nejdříve se aplikuje 1. část, pak se aplikuje libovolněkrát 2. část nebo její symetrická varianta (s prohozeným a, a' za c, c'), přičemž každé opakování sníží ϵ na polovinu, a na závěr se aplikuje 3. část (nebo její symetrická varianta). Čas v prvním sloupečku je počítán od začátku části. Pokud v nějaký čas přijde nová úloha, tak jsou v tabulce dva řádky se stejným časem — první před započítáním dané úlohy a druhý po započítání. Pokud má řádek v komentáři dva vektory a skalár, pak první vektor je pracovní vektor algoritmu, druhý je pracovní vektor oponenta — jedná se o pracovní vektory v časovém intervalu mezi časem daného a předchozího řádku, skalár pak udává rozdíl časů.

Tabulka 3: Instance dokládající dolní odhad kompetitivního poměru

čas	a	b	c	a'	b'	c'	komentář
0	1	1	0	1	1	0	přišla úloha (1 1 0)
1	1/2	1/2	0	1	0	0	$(\frac{1}{2}\ \frac{1}{2}\ 0)$, (0 1 0), 1
1	1/2	1/2	1	1	0	1	přišla úloha (0 0 1)
3/2	0	1/2	1/2	1/2	0	1/2	(1 0 1), (1 0 1), 1/2
2	0	1/4	1/4	0	0	0	$(0\ \frac{1}{2}\ \frac{1}{2})$, (1 0 1), 1/2
2	0	5/4	5/4	0	1	1	přišla úloha (0 1 1)
3	0	3/4	3/4	0	0	1	$(0\ \frac{1}{2}\ \frac{1}{2})$, (0 1 0), 1
3	1	3/4	3/4	1	0	1	přišla úloha (1 0 0)
0	1	$1-2\epsilon$	$1-2\epsilon$	1	0	1	počátek z minula
1/2	1/2	$1-2\epsilon$	$1/2-2\epsilon$	1/2	0	1/2	(1 0 1), (1 0 1), 1/2
1	1/3	$2/3-2\epsilon$	$1/3-2\epsilon$	0	0	0	$(\frac{1}{3}\ \frac{2}{3}\ \frac{1}{3})$, (1 0 1), 1/2
1	4/3	$5/3-2\epsilon$	$1/3-2\epsilon$	1	1	0	přišla úloha (1 1 0)
$2-6\epsilon$	$1+2\epsilon$	$1+2\epsilon$	0	1	6ϵ	0	$(\frac{1}{3}\ \frac{2}{3}\ \frac{1}{3})$, (0 1 0), $1-6\epsilon$
2	$1-\epsilon$	$1-\epsilon$	0	1	0	0	$(\frac{1}{2}\ \frac{1}{2}\ 0)$, (0 1 0), 6ϵ
2	$1-\epsilon$	$1-\epsilon$	1	1	0	1	přišla úloha (0 0 1)
0	$1-\epsilon$	$1-\epsilon$	1	1	0	1	počátek z minula
0	$1-\epsilon$	$2-\epsilon$	1	1	1	1	přišla úloha (0 1 0)
1	$2/3-\epsilon$	$4/3-\epsilon$	2/3	1	0	1	$(\frac{1}{3}\ \frac{2}{3}\ \frac{1}{3})$, (0 1 0), 1
1	$2/3-\epsilon$	$7/3-\epsilon$	2/3	1	1	1	přišla úloha (0 1 0)

■

Věta 7.5 *Kompetitivní poměr algoritmu MAXLOAD na cestě délky* 3 *je* 7/3.

Důkaz: Předchozí lemma udává dolní odhad, horní odhad plyne z lemmatu 7.3. ■

8 Experimenty

Během psaní diplomové práce jsem prováděl výpočetní experimenty, jejichž cílem bylo odhalit, jaké invarianty pro algoritmus platí. Experimenty byly založeny na systematickém generování instancí a aplikování algoritmu na vygenerované instance.

Testovací program pracoval tak, že metodou prohledávání do hloubky (s limitem na maximální hloubku) procházel prostor možných konfigurací (konfigurací myslím dvojici vektorů délek front pro algoritmus a pro oponenta). Program na počátku inicializoval vektory délek front pro algoritmus a oponenta na 0 a pak v každém kroku nejdříve zkontroloval, jestli už počet kroků (hloubka ve stromě rekurze) překročil stanovenou mez nebo jestli daná konfigurace byla již dříve navštívena (pomocí hashovací tabulky klíčované konfigurací). V takovém případě tato větev výpočtu skončila. V opačném případě zvolil úlohu z množiny $\{0,1\}^n$, přičetl ji k délkám front algoritmu a oponenta. Výpočet dál pokračoval v této větvi, pokud délky všech front oponenta byly menší nebo rovny 1 a zároveň pokud oponent mohl zvolit takový pracovní vektor, který by odpovídal jedné maximální nezávislé množině (a nikoliv netriviální konvexní kombinaci) a který by mohl běžet (aniž by došlo k předčasnému vyprázdnění některé fronty) po dobu 1 časové jednotky — například pro cestu délky 3 při značení podle minulé sekce to znamená, že buď $a = 1$, $b \leq 1$ a $c = 1$, nebo $a \leq 1$, $b = 1$, $c \leq 1$. Oponent tedy jeden z možných pracovních vektorů zvolil a nechal ho běžet po dobu 1 časové jednotky, stejně tak po stejnou dobu algoritmus zpracovával své fronty. V opačném případě (oponent měl frontu délky větší než 1 nebo nemohl zvolit požadovaný pracovní vektor) tato větev výpočtu skončila. Tento krok (od kontroly hloubky zanoření) se opakoval, dokud nebyla splněna některá z výše zmíněných podmínek pro ukončení větve výpočtu.

Předchozí odstavec popisoval pouze jednu větev výpočtu. Výpočet se větvil v místech, kde si zvolil úlohu nebo pracovní množinu pro oponenta. Pokud některá větev výpočtu skončila, tak se výpočet vrátil k místu posledního větvení a zvolil tam jinak (backtracking). Tak postupně vyzkoušel všechny možné volby. Během výpočtu se kontrolovala platnost zadaných kandidátů na invarianty, počítal se dolní odhad kompetitivního poměru (porovnáním délek front algoritmu s délkami front aktuálního oponenta, který však nebyl optimální offline algoritmus) a ukládaly se všechny konfigurace. které algoritmus navštívil.

Prohledávaný prostor byl záměrně zúžen celou řadou omezení:

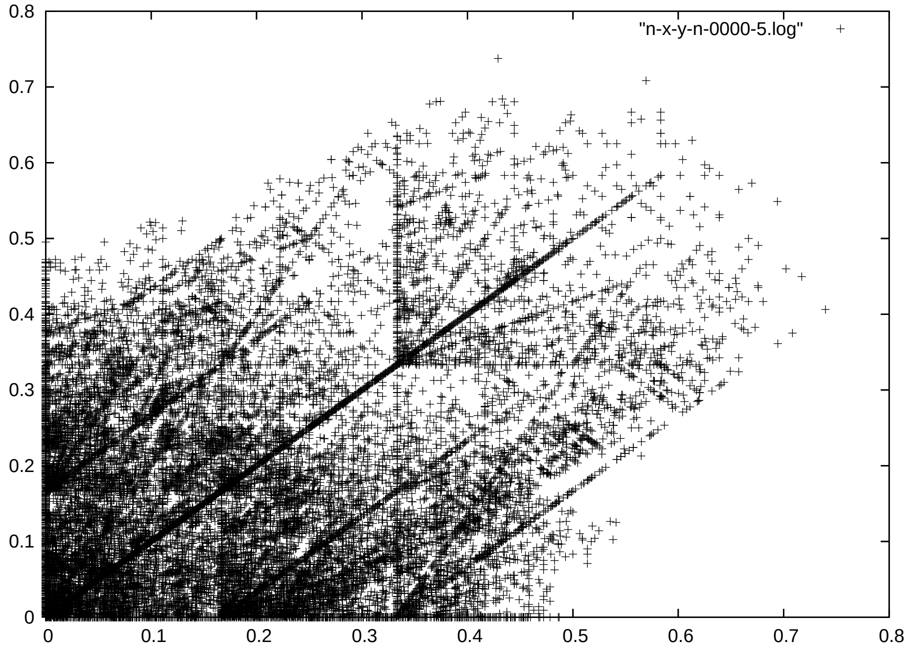
- Přicházející úlohy jsou složeny jen z nul a jedniček.
- Je pevně daná maximální oponentova délka fronty.
- Časové pojetí je diskrétní (úlohy přicházejí v pravidelných časových intervalech).
- Oponent má omezenou množinu tahů (smí volit jen maximální nezávislé množiny, které je možné zpracovávat po dobu daného časového intervalu).
- Celkové množství tahů je omezené.
- Omezená přesnost při testování navštívenosti konfigurace (konfigurace, lišící se ve všech souřadnicích o méně než dané ϵ, jsou spolu ztotožněny).

Díky těmto omezením výpočty doběhly pro cesty délky 3 a 4 v reálném čase. Pro 3-cestu se tento postup osvědčil — při rozumné přesnosti testování navštívenosti konfigurace (zaokrouhlování na 2^{-24}) stihl program během několika minut až několika desítek minut projít celý stavový prostor (přičemž ani nenarazil na omezení hloubky rekurze). Přestože byl stavový prostor radikálně omezen, tak v něm program našel posloupnosti konfigurací, jejichž kompetitivní poměr se blížil 7/3, a krajní body v prostoru konfigurací ležely takřka na hranicích, které plynou z invariantů obsažených v důkazech. Pro 4-cestu se však tento postup příliš neosvědčil. I při snížení přesnosti na 2^{-10} a omezení počtu kroků na 19 trvalo mnoho hodin, než program dokončil výpočet. Z množiny takto dosažených konfigurací se nám však nepovedlo vyvodit žádné netriviální závěry.

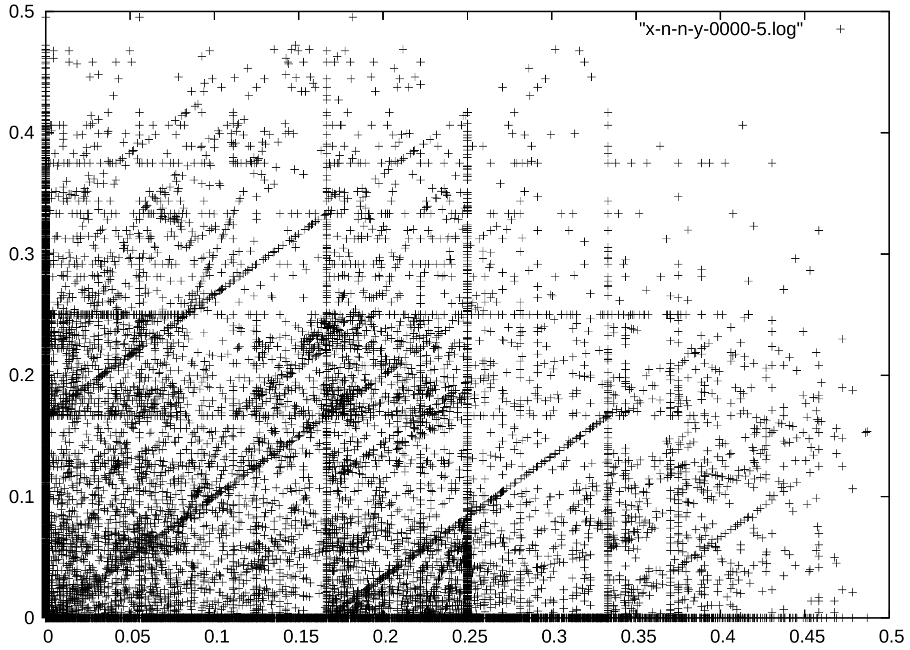
Je zajímavé, že pro běh algoritmu při tomto pojetí platí další invarianty. Například pro 3-cestu v každém okamžiku těsně před příchodem nové úlohy platí, že součet délek krajních front se rovná délce prostřední fronty. Tento invariant platí proto, že maximální velikost příchozí úlohy je dostatečně malá, aby se případný rozdíl stihl vyrovnat do konce kola.

Pro ilustraci zde uvádím některé obrázky získané vizualizací výsledků experimentů. Obrázky 1, 2 a 3 ukazují délky front algoritmu v jednotlivých stavech výpočtu na 4-cestě. Pro tyto obrázky se využily jen stavy, ve kterých měl oponent všechny fronty prázdné. Každý stav je reprezentován křížkem. V obrázku 1 se jako souřadnice křížku použily délky 2. a 3. fronty algoritmu, v obrázky 2 se použily délky 1. a 4. fronty a v obrázku 3 se použily délky 1. a 2. fronty.

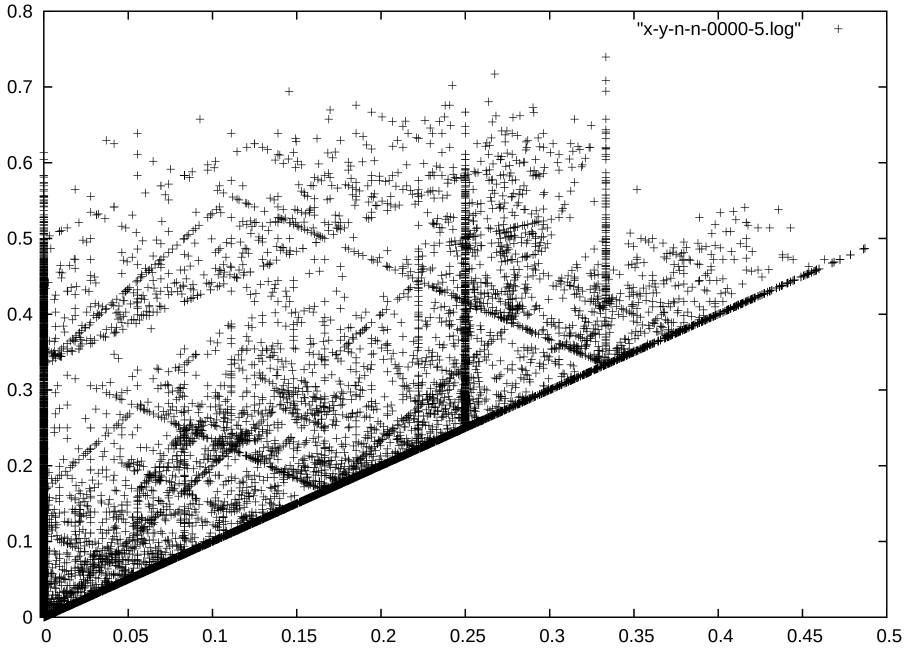
Obrázek 4 ilustruje výpočet na 3-cestě. Obrázek je získán obdobně jako v předchozím případě, zde se využily jen stavy, ve kterých měl oponent prostřední frontu prázdnou a krajní fronty délky 1. Pro souřadnice se využily délky krajních front algoritmu. Na obrázku jsou vidět, že krajní body vytyčují dvě poloroviny omezující dosažitelnost ve stavovém prostoru — $2a + c \leq 2$, $2c + a \leq 2$ (využívám značení z předchozí sekce). Tyto podmínky v podstatě odpovídají podmínkám z lemmatu 7.2. protože platí $a' = c' = 1$, $b' = 0$ (neboť v tomto obrázku jsou zahrnuty jen stavy, které tyto podmínky splňují), $r = 1$ (oponent nemá povoleno delší fronty) a $a + c = b$ (nový invariant uvedený v předminulém odstavci).



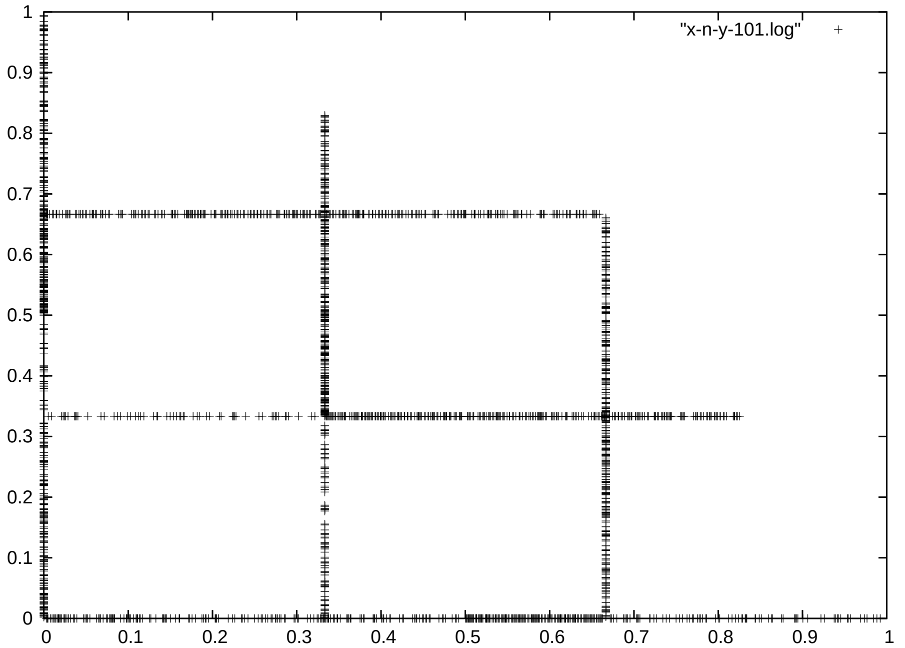
Obrázek 1: Stavy vnitřních front algoritmu na 4-cestě



Obrázek 2: Stavy vnějších front algoritmu na 4-cestě



Obrázek 3: Stavy 1. a 2. fronty algoritmu na 4-cestě



Obrázek 4: Stavy krajních front algoritmu na 3-cestě

9 Závěr

Během práce se zvolené cíle povedlo splnit jen částečně. Navrhnul jsem jediný algoritmus pro řešení rozvrhování s konflikty, algoritmus MAXLOAD. Tento algoritmus byl v práci dostatečně podrobně definován, byly probrány jeho možnosti implementace a povedlo se dokázat jeho konečnost pro libovolný vstup. Kompetitivní poměr se však povedlo dokázat pouze pro cestu délky 3 a v tomto případě je kompetitivní poměr (7/3) horší, než u již známých algoritmů (2 pro hierarchický hladový algoritmus z [1]). Na druhou stranu jsme během výzkumu nenarazili na žádný případ, kdy by si algoritmus MAXLOAD vedl vyloženě špatně, například pro žádný graf neznáme instanci, na které by MAXLOAD nebyl kompetitivní.

Výzkum by dále mohl pokračovat několika směry, zejména pak hledáním důkazu kompetitivnosti algoritmu pro všechny grafy a hledáním hodnoty kompetitivního poměru pro konkrétní grafy, zejména pro delší cesty. Další směr výzkumu by se mohl týkat možností, jak algoritmus lépe implementovat. V případě popsání mnohostěnu DIRS pomocí jeho stěn by nejspíš bylo možné použít v implementaci lineární programování pro nalezení množiny MAX. Další možnost je dokázat konvergenci iteračního algoritmu, který jako pracovní vektor volí náhodné vrcholy MAX a zvolený vektor nechává pracovat po krátký časový okamžik.

Nabízí se také možnost zkoumat podobné algoritmy. Například po přečtení [2] se domnívám, že by bylo zajímavé zkoumat algoritmus, který namísto $\langle A, V \rangle$ minimalizuje $\langle A, V \rangle / |V|$.

Literatura

[1] Marek Chrobak, János Csirik, Csanád Imreh, John Noga, Jiří Sgall, Gerhard J. Woeginger [2001]. The buffer minimization problem for multiprocessor scheduling with conflicts, *ICALP '01: Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, 862–874.

[2] Günter Rote [2003]. Pursuit-evasion with imprecise target location, *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, 747–753.

[3] Hisashi Koga [2001]. Balanced Scheduling toward Loss-Free Packet Queuing and Delay Fairness, *ISAAC '01: Proceedings of the 12th International Symposium on Algorithms and Computation*, 61–73.

[4] Amotz Bar-Noy, Ari Freund, Shimon Landa, and Joseph (Seffi) Naor [2002]. Competitive on-line switching policies, *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, 525–534.