České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra řídicí techniky

Rozvrhování výrobních procesů s blokovacími operacemi

Scheduling of manufactoring processes with blocking operations

Diplomová práce

Studijní program: Elektrotechnika a informatika, strukturovaný
Studijní obor: Kybernetika a měření (magisterský)
Vedoucí práce: Ing. Přemysl Šůcha, Ph.D.

Michal Smola

Praha 2010

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Michal Smola**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Obor: Kybernetika a měření, blok KM1 - Řídicí technika

Název tématu: **Rozvrhování výrobních procesů s blokujícími operacemi**

Pokyny pro vypracování:

1. Seznamte se s algoritmy pro rozvrhování s blokujícími operacemi.
2. Seznamte se se systémy umožňující plánování výroby více uživateli (např. Google DOCs).
3. Pro vybranou případovou studii navrhněte a implementujte algoritmus pro rozvrhování úloh výroby s blokujícími operacemi. Řešení srovnejte s jiným vybraným již existujícím řešením. Snažte se navrhnout takové řešení, aby data algoritmu mohlo zadávat více uživatelů.
4. Odzkoušejte a zdokumentujte Vámi navržené řešení.

Seznam odborné literatury:

[1] Peter Brucker, Sigrid Knust. Complex Scheduling, Springer Berlin Heidelberg, 2006.
[2] Peter Brucker, Silvia Heitmann, Johann Hurink and Tim Nieberg, Job-shop scheduling with limited capacity buffers, OR Spectrum, Springer Berlin / Heidelberg, 2006.

Vedoucí: Ing. Přemysl Šůcha, Ph.D.

Platnost zadání: do konce zimního semestru 2009/10

prof. Ing. Michael, Šebek, DrSc.
vedoucí katedry

L.S.

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 27. 2. 2009

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 5. 1. 2010 ..... Smola .....

podpis

Abstrakt

Obsahem práce je návrh a implementace paralelního algoritmu lokálního prohledávání s tabu seznamem (local search with tabu list, tabu search) řešícího úlohu zobecněného blokovacího job shopu (generalized blocking job shop). Za základ posloužil algoritmus prezentovaný v [11], který byl také implementován, a rychlost výpočtu pomocí obou algoritmů byla změřena a srovnána.

Součástí práce je i porovnání softwarových platforem MS Sharepoint a Google Docs vhodných pro implementaci uživatelského rozhraní umožňujícího úpravu zadání více uživateli a implementace takového rozhraní v Google Docs.

Abstract

The aim of the work is the design and implementation of an parallel tabu search algorithm for solving generalized blocking job shop problem. The algorithm is based on an algorithm presented in [11] which was also implemented and the speed of both algorithms was measured and compared.

Part of the thesis is the comparision of software platforms MS Sharepoint and Google Docs suitable for implementing a user interface allowing multiple users to edit input of the algorithm. Such an interface was implemented in Google Docs.

Obsah

1 Úvod **8**
1.1 Modifikace základních úloh 8
1.1.1 Buffery 9
1.1.2 Kumulativní zdroje 10
1.1.3 Prostorové zdroje 10
1.2 Reprezentace úlohy 11
1.2.1 Disjunktivní graf 11
1.2.2 Alternativní graf 11
1.2.3 Pořadí úloh 13
1.3 Přehled publikovaných prací 13
1.4 Obsah práce 17

2 Blokovací job shop **18**
2.1 Formulace problému 18
2.2 Grafová reprezentace 20
2.3 Příklad 22

3 Popis algoritmu **26**
3.1 Lokální prohledávání 26
3.2 Tabu prohledávání 26
3.3 Popis algoritmu 27
3.3.1 Postup nalezení topologického uspořádání vrcholů . . 27
3.3.2 Postup nalezení kritické cesty v grafu 28
3.3.3 Generování sousedů 28
3.3.4 Oprava souseda 29
3.3.5 Vlastní tabu algoritmus 30

4 Paralelizace algoritmu **34**
4.1 Paralelizace algoritmů lokálního prohledávání 34
4.1.1 Paralelní prohledávání z více míst 34
4.1.2 Paralelní průzkum sousedství 34
4.1.3 Paralelní ohodnocování řešení 35
4.2 Podpora vícevláknového programování v .NET frameworku . 35
4.3 Vlastní paralelní algoritmus 38

5 Softwarové platformy vhodné pro implementaci nástroje pro plánování výroby s více uživateli **40**
5.1 Google Docs 40
5.1.1 Přístup do Google Docs pomocí knihovny pro .NET . 42
5.1.2 Implementace uživatelského rozhraní 43
5.2 Microsoft SharePoint 44
5.3 Porovnání technologií 46

6 Experimentální měření rychlosti **48**
6.1 Vstupní zadání pro měření 48
6.2 Implementační detaily 48
6.3 Výsledky měření 49
6.4 Shrnutí výsledků 54

7 Závěr **55**

1 Úvod

Rozvrhování je důležitá součást každodenního života každého z nás. Každým okamžikem se rozhodujeme, co budeme kdy dělat a jak si svůj čas uspořádáme. Rozhodujeme se na základě našich přání a priorit. U jednodušších rozhodnutí si to možná ani neuvědomíme, ale ve složitějších případech, kdy času není nazbyt a termíny neúprosně klepou na dveře, dá hodně přemýšlení a práce vše uspokojivě uspořádat. V případě výrobních procesů celý problém dostává nový rozměr. Již není potřeba rozvrhnout činnost jednoho člověka, ale celé skupiny lidí a strojů a i množství úkolů významně roste. Mezi úkoly je také mnohem více precedenčních omezení a úkoly často mají určen čas, do kdy musí být hotové. To vše zvyšuje náročnost nalezení optimálního rozvrhu, kdy jsou všechny operace dokončené v nejkratším možném termínu, a vybízí k využití strojů pro řešení této úlohy.

V teorii rozvrhování se většinou zkoumají částečně zjednodušené úlohy, nejznámější jsou open shop, flow shop a job shop. Zvláště job shop, z uvedených zástupců nejuniverzálnější a výpočetně nejnáročnější, přitahuje pozornost výzkumu (např. [2, 3, 4, 6]). U klasických úloh flow shop i job shop je však přítomen předpoklad, že mezi stroji je dostatek místa pro skladování libovolného množství výrobků a že operace, které stroje provádějí, jsou různé (tedy že každý stroj je unikátní). Tento předpoklad ale v praxi samozřejmě splněn není, proto je vhodné úlohy rozšířit a vzít v úvahu možnost výskytu více shodných strojů, které se mohou zastupovat, a také uvažovat omezenou (případně i nulovou) velikost odkládacích míst mezi stroji - bufferů. Omezení velikosti bufferů je vhodné i v případě dostatečných prostor v místě výroby. Čím méně výrobků se skladuje, tím je výroba efektivnější.

Úloha rozvrhování výrobních procesů je značně výpočetně náročná, často se jedná o NP úplnou úlohu [3, 4, 5]. Z tohoto důvodu je větší pozornost věnována heuristickým algoritmům, které nemusí vést k optimálnímu výsledku, ale jsou určeny k nalezení výsledku dobrého v kratším čase.

1.1 Modifikace základních úloh

Základní úloha job shop je tvořena joby, kde každý job představuje skupinu operací mezi nimiž jsou precedenční omezení. Každá operace má navíc ur-

čený stroj, na kterém má být vykonána. Na rozdíl od úlohy flow shop může být v případě job shopu počet operací v jobech různý, stejně jako může být různé pořadí strojů, v jakém je budou joby využívat.

1.1.1 Buffery

Omezení velikosti bufferů dále zesložiťuje již tak výpočetně náročné úlohy [3]. Pokud jeden stroj dokončí operaci a stroj, na kterém má být provedena další operace, je zaneprázdněn, je potřeba job uložit do bufferu. Pokud je ale buffer plný, nebo dokonce vůbec neexistuje, job nelze nikam přesunout a blokuje tak stroj pro další použití. Job zůstává na místě tak dlouho, dokud se neuvolní následující stroj, nebo místo v bufferu. Operace, která nemá k dispozici buffer a blokuje stroj, se nazývá blokovací.

Pokud uvažujeme model výrobního procesu s buffery, rozlišujeme podle možnosti využití bufferů následující typy modelů [2, 3]:

- Buffery náležející jobům (job dependent buffers) - Každý job má přidělen svůj buffer s kapacitou nula nebo jedna. Job může tento buffer využít kdykoliv.

- Model s párovými buffery (pairwise buffer model) - Buffer vždy přísluší dvěma strojům. Job může použít buffer pouze pokud se přesouvá mezi těmito dvěma stroji.

- Model s výstupními buffery (output buffer model) - Buffer vždy náleží jednomu stroji a mohou ho využít pouze joby, které stroj právě opouští.

- Model se vstupními buffery (input buffer model) - Buffer opět náleží jednomu stroji, ale může ho použít pouze job, který čeká na zpracování na tomto stroji.

- Obecný model s buffery (general buffer model) - Takový model, ve kterém se mohou vyskytovat všechny předchozí případy.

Situaci, kdy se objeví cyklus dvou a více jobů, které čekají na uvolnění strojů blokovaných ostatními joby v cyklu, říkáme uvíznutí (deadlock). Situaci je možné vyřešit současným přesunem všech uvíznutých jobů. Takovou operaci

nazýváme prohození (swap). V některých případech, např. [4, 5], je tedy ještě rozlišováno mezi swap a no-swap modely. V případě no-swap modelu se při uvíznutí nedá nic dělat a rozvrh vedoucí k této situaci je nepřípustný.

1.1.2 Kumulativní zdroje

Obecnější pohled poskytuje např. [8, 9], kde je zaveden pojem kumulativní zdroje. Takové zdroje nemají jen dva stavy (plný - prázdný), ale těchto stavů mají více. Tyto stavy vyjadřují, jak moc je daný zdroj zaplněn (např. v případě skladu) anebo jak moc je využíván (např. surovina v cisterně). Od bufferů se liší tím, že různé operace mohou spotřebovat/produkovat různá množství daného zdroje a že jako zdroje jsou uvažovány i stroje. Každý zdroj má zdola i shora omezenou kapacitu. Při překročení těchto mezí je rozvrh nepřípustný. Jedná se například o již zmíněné sklady, nebo o různé zásobníky a nádrže, které mohou být naplňovány a vyprazdňovány. Podle povahy zdroje rozlišujeme mezi spojitou a diskrétní variantou (která je vlastně jen speciálním případem spojité varianty). Obě varianty vyžadují jiný přístup k řešení. Spojité zdroje (například v případě výroby nápojů nebo chemických provozů) se označují jako rezervoáry a nespojité zdroje (sklad produktů, stroje) jako diskrétní zdroje. Operace mají zadány minimální a maximální časy počátku a konce, minimální čas trvání a také kolik jakého zdroje potřebují či produkují.

1.1.3 Prostorové zdroje

Prostorové zdroje (spatial resources), kterým je věnován např. [10], jsou dalším možným rozšířením klasických job shop a flow shop úloh. Jedná se o zdroje, které jsou použity pro více po sobě následujících operací. Jako příklad může posloužit úloha ze stavebnictví, kdy při zateplení domu potřebujeme lešení. Pro operaci „stavba lešení" potřebujeme prostorový zdroj „lešení", který zůstává nedostupný pro jiné skupiny operací, dokud není dokončena skupina operací „odstranění staré omítky", „instalace izolační vrstvy", „nahození nové omítky" a nakonec „rozebrání lešení". Od klasických zdrojů se tedy liší tím, že jakmile je využíván skupinou operací, zůstává blokován až do dokončení poslední operace ze skupiny, a to i při případných prodlevách mezi operacemi, kdy např. jedna z operací čeká na uvolnění jiného zdroje.

Při výskytu prostorových zdrojů v rozvrhovací úloze je častější výskyt tzv. deadlock situací, kdy dvě a více operací vzájemně čekají na dokončení a na uvolnění zdrojů, protože při uvažování prostorových zdrojů vzniká deadlock i při vzájemném blokování zdrojů skupinami operací.

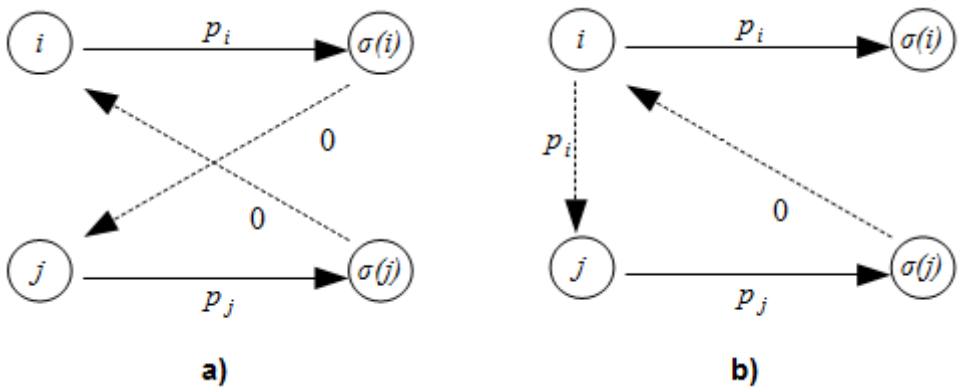
1.2 Reprezentace úlohy

1.2.1 Disjunktivní graf

Klasická job shop úloha je většinou reprezentována jako disjunktivní graf [1, 2, 3]. Disjunktivní graf je možno zapsat jako trojici $G = (N; F; D)$, kde N je množina vrcholů (nodes), F množina orientovaných hran a D množina párů orientovaných hran. Množina vrcholů představuje všechny operace ze všech jobů plus zdrojový a koncový vrchol, které jsou začátkem a koncem rozvrhu. Hrany $(i, j) \in F$ jsou pevné (fixed arcs) a zajišťují správné pořadí operací v jobech. Mají váhu f_{ij}, rovnou době trvání operace i. Do množiny pevných hran patří i hrany vedoucí ze zdrojového vrcholu do vrcholů odpovídajícím prvním operacím jobů, ty mají váhu nula, a hrany vedoucí z posledních vrcholů (operací) jobů, ty mají váhu rovnu době trvání poslední operace. Hrany v množině D nazýváme disjunktivní (disjunctive arcs). Jsou ve tvaru $((i, j), (j, i))$, kde i a j jsou operace prováděné na jednom stroji. Rozvrh je pak určen výběrem S, který vybírá z každého páru disjunktivních hran právě jednu a tím určuje pořadí operací na strojích.

1.2.2 Alternativní graf

Zobecněním disjunktivního grafu získáme alternativní graf [4], pomocí něhož můžeme modelovat i blokovací operace [1, 2, 3, 4, 5, 7]. Alternativní graf se také skládá ze tří množin $G = (N; F; A)$. První dvě množiny N a F mají stejný význam jako v případě disjunktivního grafu. Třetí množina se nazývá množina alternativních hran (alternative arcs) a také obsahuje páry hran, které určují, která z operací určených na jeden stroj se vykoná dříve. Tentokrát ale obě hrany z páru nemusí vést pouze mezi těmito dvěma operacemi. Páry alternativních hran mají tedy obecně následující tvar: $((i, j), (h, k))$. Výrazem $\sigma(i)$ označme operaci, která v příslušném jobu následuje za operací i. Pokud je operace i blokovací, vychází jí odpovídající alternativní



Obrázek 1: Alternativní hrany blokovacích operací, obrázek převzat z [2] str. 156

hrana s váhou nula z operace $\sigma(i)$ a vede k operaci j. To zaručí, že operace j začne až když job obsahující operaci i opustí stroj. Pokud operace i blokovací není, vede alternativní hrana přímo z ní, protože job stroj opustí okamžitě po dokončení operace. V tomto případě má alternativní hrana váhu odpovídající době trvání operace i. Situace je zřejmá z obrázku 1. Na obrázku 1a) jsou obě operace blokovací, proto obě alternativní hrany vedou z následníků operací ($\sigma(i)$ a $\sigma(j)$) a mají nulovou váhu. Pár alternativních hran má tedy tvar $((\sigma(i), j), (\sigma(j), i))$. Na obrázku 1b) operace i blokovací není, proto alternativní hrana míří přímo z ní a má váhu p_i. Pár alternativních hran v tomto případě je ve tvaru $((i, j), (\sigma(j), i))$.

Pomocí alternativního grafu lze namodelovat nejen blokovací operace (tedy ty, které nemají k dispozici buffer), ale i buffery samotné v jejich nejobecnější variantě [3]. Buffer o velikosti n je rozdělen na n bufferů o velikosti jedna. S těmito buffery je pak nakládáno jako se stroji, na kterých je doba trvání operace nulová. Mimo bufferů lze pomocí alternativního grafu namodelovat mnoho dalších situací [5], např. maximální prodlevu mezi dvěma operacemi, nedostupnost stroje v určenou dobu (odstávka stroje), časové okno pro provedení operace atd.

Při reprezentaci úlohy alternativním grafem musí algoritmus vybrat jednu z každé dvojice alternativních hran. Graf tvořený pouze pevnými a vybranými alternativními hranami nesmí obsahovat cyklus kladné délky (výběr splňující tuto podmínku se nazývá konzistentní). Doba rozvrhu je pak ur-

čena délkou kritické (tj. nejdelší) cesty v grafu.

1.2.3 Pořadí úloh

Pro reprezentaci můžeme využít také pořadí úloh, a to jak pro úlohu s buffery [8] a pro úlohu s kumulativními zdroji [9], tak pro úlohu s prostorovými zdroji [10]. V případě úlohy s buffery je počítáno s pořadími, v jakých jsou joby zpracovávány na strojích a v jakých vstupují a vystupují z bufferů. V [2] je ukázáno, že tento typ reprezentace je vhodný pouze pro speciální typy bufferů, konkrétně pro vstupní buffery, výstupní buffery, párové buffery a buffery náležející jobům. V těchto případech lze, narozdíl od obecného modelu s buffery, z pořadí efektivně vypočítat výsledný rozvrh.

V případě kumulativních zdrojů jsou používána pouze pořadí jobů na zdrojích (zdroje zde představují jak stroje a buffery, tak např. materiál potřebný k práci na jobu).

1.3 Přehled publikovaných prací

V práci Job-shop scheduling with limited capacity buffers [2] se autoři zabývají hlavně reprezentací různých typů bufferů pomocí alternativního grafu a konstrukcí rozvrhů z již známých sekvencí jobů na strojích, kde ukazují, že mimo případ obecných bufferů k tomu existují efektivní algoritmy.

V disertační práci Job-shop scheduling with limited buffer capacities [3] se autorka zabývá modelováním různých druhů bufferů v alternativním grafu. Ukazuje, že nalezení optimálního výběru alternativních hran je NP-úplná úloha. Popisuje opravující tabu algoritmus, kdy umožňuje generovat nepřípustná řešení, která pak upravuje do přípustné podoby. K tomu využívá tzv. *block approach* teorém, který zavádí pojem blok (zjednodušeně několik po sobě následujících operací na jednom stroji), a ukazuje, že zlepšení rozvrhu lze dosáhnout pouze přesunutím některé operace bloku před první, nebo za poslední operaci bloku.

V práci Job-shop scheduling with blocking and no-wait constraints [4] autoři definují alternativní graf, zabývají se složitostí problému a pro jeho různé varianty (ideální job shop, blokovací job shop no-swap, blokovací job shop se swap a no-wait job shop) uvádějí čtyři heuristické algoritmy a jeden

algoritmus větví a mezí (branch and bound). Heuristické algoritmy však často přinášejí nepřípustné řešení a autoři sami uvádějí, že jejich výsledky jsou neuspokojivé.

V práci Machine scheduling via alternative graphs [5] autoři také definují alternativní graf, modelují pomocí něj výrobnu oceli, vlakový provoz, sítě s přepínanými pakety a ukazují, že nalezení optimálního výběru alternativních hran je NP-úplná úloha, a to i v případě, že všechny hrany mají stejnou váhu.

Práce Cyclic job shop scheduling problems with blocking [6] se zabývá speciálním případem job shop rozvrhování, a to cyklickou variantou, kdy každý job je prováděn opakovaně. Autoři publikují tabu algoritmus s opravováním a algoritmus smíšeného celočíselného programování.

V práci A Tabu Search Algorithm for the Railway Scheduling Problem [7] autoři opět definují alternativní graf a publikují opravující tabu algoritmus. Opravování je založené na tom, že nepřípustné řešení obsahuje cyklus kladné délky. Když tedy takové řešení vygenerují, najdou cyklus a změní v něm jednu alternativní hranu. Dále ukazují, že změnou alternativních hran, které nejsou součástí kritické cesty grafu, nelze dosáhnout kratšího rozvrhu.

V knize Project Scheduling with Time Windows and Scarce Resources: Temporal and Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions [8] se autoři v kapitole 2.12 zabývají kumulativními zdroji, a to jak diskrétní variantou (kapitola 2.12.1), tak variantou spojitou (kapitola 2.12.2). V algoritmech pracují s pořadími operací. Porušení omezení u zdrojů (přeplnění či přečerpání) detekují pomocí tzv. zakázaných množin a vypořádávají se s nimi pomocí posunutí začátku operace, která omezení poruší, za operaci, která zdroj dostane do takového stavu, aby kritickou operaci zvládl. Při přeplnění zdroje se tedy kritická operace posune za operaci, která zdroj dostatečně vyčerpá, při přečerpání za operaci, která zdroj dostatečně naplní.

V článku Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results [9] autor podává přehled o existujících přístupech k propagaci zdrojových omezení a prezentuje dva nové algoritmy. Existující přístupy představují dva proudy:

Timetabling spočívá v součtu minimálních požadavků operací na zdroje v čase, kdy budou tyto operace s jistotou aktivní. Pokud výsledný součet porušuje omezení zdroje, musí se pořadí operací změnit. Také se zpřesňují odhady začátků a konců trvání jednotlivých operací. Výhodou je malá algoritmická náročnost. Nevýhodou je, že pokud nejsou operace dostatečně časově omezené (mají velká rozmezí, kdy mohou proběhnout), nepropaguje tento přístup dostatečně, případně nepropaguje vůbec.

Activity interactions je založeno na zkoumání vztahů operací, které soutěží o jeden zdroj. Např. když časové okno operace A končí před začátkem časového okna operace B, je pořadí těchto operací jasné. Dále se zkoumá např. vztah jedné operace k množině ostatních operací na jednom zdroji, kdy lze zjistit, že daná operace nemůže být zařazena jako poslední nebo jako první na daném zdroji. Tento přístup lze použít pouze v případě diskrétních zdrojů. Nevýhodou je stejně jako u *timetabling* slabé nebo žádné propagování v případě velkých časových oken operací.

Autor přichází se dvěma novými propagačními algoritmy. První, který je definovaný jen na diskrétních zdrojích, nazval *energy precedence constraint* a hlavní myšlenka je následující: Mějme zdroj s kapacitou Q, operaci x a množinu Ω, která představuje operace rozvržené před x. Zdroj musí poskytnout dostatek energie pro všechny tyto události, tedy x nemůže začít dříve, než je součet požadavků na zdroj všech operací z Ω podělený kapacitou zdroje Q. Stejný postup je aplikovaný i „zezadu", kdy se nepočítá s časy začátků operací, ale s jejich *deadline* časy.

Druhý propagační algoritmus se jmenuje *balance constraint* a je definovaný jak pro reservoáry, tak pro diskrétní zdroje. Spočívá ve výpočtu minimální a maximální hladiny zdroje po zahájení a ukončení každé operace. Operace nemusí mít požadavky na zdroje zadané přesně, ale lze je zadat jako intervaly, tedy i hladiny zdroje pak budou ve formě intervalů. Pomocí těchto hladin lze odhalit porušení omezení (když vypočtená maximální úroveň hladiny je pod spodní hranicí omezení nebo když minimální úroveň hladiny je nad horní hranicí omezení), lze najít nové hranice požadavků operací (když je aktuální hladina zdroje $[3, 5]$, spodní omezení zdroje je 1 a následující operace spotřebuje $[1, 3]$ jednotek zdroje, musíme upravit požadavky operace na $[1, 2]$, aby nemohlo dojít k přečerpání zdroje), případně

lze objevit nutnost změny pořadí operací tak, aby byl rozvrh přípustný.

V práci A Heuristic Methodology for Solving Spatial Resource - Constrained Project Scheduling Problems [10] autoři publikují heuristický algoritmus řešící úlohy s prostorovými zdroji. Prostorové zdroje jsou využívány skupinami operací. Operace, která prostorový zdroj využívá ve skupině jako první, je označena jako call operace. Skupina může obsahovat i více call operací, a to v případě, kdy využívá více prostorových zdrojů. Prezentovaný algoritmus se skládá ze tří algoritmů a jako vstup požaduje seznam operací λ seřazený podle priorit (pokud nejsou priority definovány, tak v náhodném pořadí) a přednostní omezení určující pořadí operací v jobech. První algoritmus zpracovává seznam κ, který obsahuje všechny call aktivity ve stejném pořadí v jakém jsou uvedeny v seznamu λ. Tento seznam transformuje do seznamu call aktivit β, který splňuje přednostní a prostorové omezení. Druhý algoritmus přetváří prioritní seznam všech aktivit λ do prioritního seznamu všech aktivit μ, který splňuje všechna přednostní a prostorová omezení. Využívá při tom omezení vygenerovaná prvním algoritmem, což zaručuje, že nebudou porušena omezení týkající se prostorových zdrojů. Pro urychlení výpočtu jsou použita dvě nadřazená pravidla (dominance rules). Pokud v průběhu prvního algoritmu nelze pokračovat v tvorbě seznamu, algoritmus se vrátí o krok zpět a zkusí jinou možnost. První nadřazené pravidlo umožňuje odhalit situaci, kdy ani vracení nepomůže, a ukončí aktuální hledání. Druhé nadřazené pravidlo spočívá v pamatování prozkoumaných kombinací operací a zabraňuje prozkoumávání již prozkoumaných množin.

Po vygenerování seznamu, který splňuje všechna omezení, je z něj sestaven rozvrh pomocí sekvenčního algoritmu na tvorbu rozvrhu (serial schedule generation scheme). Autoři ukazují, že paralelní varianta tohoto algoritmu není vhodná, protože umožňuje tvorbu rozvrhu obsahujícího uvíznutí.

V práci A new neighborhood and tabu search for the Blocking Job Shop [11] je publikován tabu algoritmus pro zobecněný blokovací job shop. Autoři uvádějí, že u blokovacího job shopu v případě použití lokálního prohledávání je problémem časté generování těch sousedů, kteří jsou nepřípustným řešením úlohy. Uvádějí tedy algoritmus, který generuje pouze přípustné sousedy, a to tak, že ty nepřípustné opravuje. Pro reprezentaci autoři používají

upravený disjunktivní graf a poznamenávají, že délku rozvrhu tvoří kritická cesta v tomto grafu. Zkrácení rozvrhu lze tedy dosáhnout pouze vyjmutím některé disjunktivní hrany na kritické cestě a přidáním hrany, která s ní tvoří dvojici.

1.4 Obsah práce

Obsahem této práce je návrh a implementace algoritmu pro rozvrhování úloh výroby typu job shop, kde nejsou přítomny žádné buffery mezi stroji. Na základu algoritmu prezentovaného v [11] byl vytvořen paralelní metaheuristický algoritmus umožňující využít naplno moderní počítače, zrychlit tak výpočet rozvrhu a dosáhnout za stejnou dobu výpočtu lepšího výsledku. Zároveň bylo implementováno uživatelské rozhraní umožňující úpravu zadání více uživateli najednou.

Text je rozvržen do sedmi kapitol. Obsahem druhé kapitoly, Blokovací job shop, je popis a formulace úlohy, její reprezentace pomocí grafu a ilustrující příklad. Ve třetí kapitole, Popis algoritmu, je popsán algoritmus ve své sekvenční variantě a jsou uvedeny pomocné algoritmy, které jsou při výpočtu nezbytné. Ve čtvrté kapitole, Paralelizace algoritmu, jsou uvedeny metody používané při paralelizaci algoritmů lokálního prohledávání, představeny jsou možnosti paralelizace v .NET frameworku od firmy Microsoft a na závěr je popsána struktura vlastního paralelního algoritmu. V páté kapitole, Softwarové platformy vhodné pro implementaci nástroje pro plánování výroby s více uživateli, jsou porovnány dvě softwarové platformy Google Docs a MS Sharepoint vhodné pro implementaci uživatelského rozhraní s možností úpravy zadání více uživateli. V šesté kapitole, Experimentální měření rychlosti, jsou uvedeny a porovnány výsledky měření původního i nově prezentovaného algoritmu. Konečně v sedmé kapitole, Závěr, jsou shrnuty výsledky celé práce.

Během psaní diplomové práce jsem dostal možnost upřesnit si některé otázky související s terminologií. Na základě doporučení pracovníků jazykové poradny jsem v názvu práce nahradil přídavné jméno slovesné *blokující* přídavným jménem účelovým *blokovací*.

2 Blokovací job shop

Blokovací job shop (blocking job shop - BJS) je taková verze job shopu, kde nejsou mezi stroji přítomny žádné buffery pro uchovávání částečně zpracovaných jobů, a job tedy musí zůstat po zpracování na stroji a tím ho blokovat, dokud není možné přesunout ho na další stroj. Zobecněný blokovací job shop (generalized blocking job shop - GBJS) bere v úvahu dva další parametry. Prvním parametrem je doba potřebná k přesunu jobu z jednoho stroje na další a druhým parametrem je doba nastavení stroje při výměně jobů v závislosti na pořadí těchto jobů.

Jak bylo uvedeno v minulé kapitole, job se skládá z operací. V GBJS si operaci můžeme představit jako čtyři po sobě následující kroky: (i) krok převzetí (take-over step), kdy je job přesunut na stroj z předchozího stroje (případně umístěn na stroj, pokud se jedná o první operaci daného jobu), (ii) krok zpracování (processing step), kdy je na jobu provedena úprava, (iii) (možná nepřítomný) krok blokování (blocking period), kdy job čeká na uvolnění následujícího stroje, a sám tak blokuje pro další použití současný stroj a (iv) krok odevzdání (hand-over step), kdy je job přesunut na další stroj (nebo kdy je job uvolněn, pokud se jedná o poslední operaci daného jobu). Také je nutné stroje nastavovat, seřizovat. Tato nastavení se provádějí před přijetím prvního jobu na stroj, při výměnách jobů na stroji a po zpracování posledního jobu. Časy nastavení jsou závislé na pořadí jobů, v jakém jsou na stroji zpracovávány. GBJS úloha spočívá v nalezení nejkratšího přípustného rozvrhu všech operací. BJS je speciálním případem GBJS, kdy jsou časy přesunů jobů nulové, stejně jako časy nastavení strojů.

2.1 Formulace problému

Mějme množinu strojů M, množinu jobů $\mathcal{J}$ a množinu operací I. Prázdná počáteční operace $\sigma \in I$ je určena k vykonání před všemi ostatními operacemi a reprezentuje začátek rozvrhu. Naproti tomu $\tau \in I$ je prázdná koncová operace, je určena k vykonání po všech ostatních operacích a reprezentuje konec rozvrhu. Obě tyto operace mají nulovou délku a nejsou přiřazeny k žádnému stroji. Job $J \in \mathcal{J}$ je definovaný svou množinou operací, tedy $J \subseteq I$. Operace patřící do jobu J jsou značeny písmenem J s dolním indexem označujícím pořadí operace v jobu. Sled operací v jobu J

je tedy $J_1, J_2, \ldots, J_{|J|}$. Pro každý job $J \in \mathcal{J}$, množina uspořádaných dvojic po sobě jdoucích operací J_r a J_{r+1} je značena jako $A_J = \{(J_r, J_{r+1}) : r = 1, \ldots, |J| - 1\}$. Pro každý stroj $m \in M$, I_m značí množinu operací využívajících m. V GBJS každá operace $i \in I$ představuje čtyři kroky: krok převzetí s dobou trvání d_i^t, krok zpracování s dobou trvání $d_i > 0$, krok čekání (blokování) s neznámou dobou trvání a krok odevzdání s dobou trvání d_i^h. Pro každou operaci $i \in I$, značíme $t(i)$ krok převzetí a $h(i)$ krok odevzdání.

Pro každý stroj $m \in M$ a operace $i, j \in I_m, i \neq j$, d_{ij}^s značí čas nastavení stroje v případě, že operace j na stroji bezprostředně následuje operaci i. Podobně pro každý stroj $m \in M$ a operaci $i \in I_m$, $d_{\sigma i}^s$ značí čas nastavení stroje před převzetím operace i, která je určena ke zpracování na stroji m jako první a $d_{i\tau}^s$ značí čas nastavení stroje po operaci i, která je zpracována na stroji m jako poslední. Počáteční časy kroku převzetí a kroku odevzdání operace $i \in I$ jsou značeny $x_{t(i)}$, resp. $x_{h(i)}$ a časy začátků prázdných operací x_σ a x_τ . GBJS je pak možné formulovat následovně:

$$\text{Minimalizujte } x_\tau \text{ při dodržení:} \tag{1}$$

$$x_{t(j)} - x_{h(i)} \geq d_i^h + d_{ij}^s$$

$$\text{nebo } x_{t(i)} - x_{h(j)} \geq d_j^h + d_{ji}^s \text{ pro všechna } \{i, j\} \subseteq I_m,\ m \in M \tag{2}$$

$$x_{h(i)} - x_{t(j)} = 0 \text{ pro všechna } (i, j) \subseteq A_j\ ,\ J \in \mathcal{J} \tag{3}$$

$$x_{h(i)} - x_{t(i)} \geq d_i^t + d_i \text{ pro všechna } i \in I \tag{4}$$

$$x_{t(i)} - x_\sigma \geq d_{\sigma i}^s,$$

$$x_\tau - x_{h(i)} \geq d_i^h + d_{i\tau}^s \text{pro všechna } i \in I \tag{5}$$

$$x_{t(i)}, x_{h(i)}, x_\sigma, x_\tau \geq 0 \text{ pro všechna } i \in I \tag{6}$$

Cílová funkce minimalizuje délku celého rozvrhu reprezentovanou časem začátku koncové operace τ. Disjunktivní omezení 2 zajišťuje, že se žádné dvě operace i a j na stroji m v čase nepřekrývají (krok odevzdání operace i musí předcházet kroku převzetí operace j, nebo obráceně), a tedy že stroj m je po dobu vykonávání operací i a j blokovaný. Synchronizační omezení

3 zaručuje požadované chování jobu při přechodu od operace i k operaci j: čas začátku kroku odevzdání $x_{h(i)}$ operace i a čas začátku kroku převzetí $x_{t(j)}$ operace j se musí rovnat (tyto kroky se v čase překrývají). Omezení 4 pro každou operaci i zaručuje, že krok odevzdání operace i je proveden až po dokončení kroků převzetí a zpracování operace i. Omezení 5 zaručuje respektování času nastavení stroje před první a po poslední operaci na každém stroji.

2.2 Grafová reprezentace

Stejně jako v případě klasického job shop problému, může být GBJS formulován jako optimalizační úloha v (zobecněném) disjunktivním grafu $G = \langle V, A, E, \varepsilon, c \rangle$. G je prostý orientovaný graf s množinou vrcholů V, množinou hran $A \cup E$, kde A je množina konjunktivních hran, E množina disjunktivních hran, c určuje váhy hran a ε je skupina disjunktivních množin reprezentující disjunktivní strukturu grafu G. Vrcholy reprezentují jedlotlivé operace, ze kterých se skládají joby, konjunktivní hrany vyjadřují precedenční omezení v rámci jednotlivých jobů a disjunktivní hrany určují pořadí operací určených k vykonání na jednom stroji. Každá disjunktivní množina $D \in \varepsilon$ obsahuje právě dvě disjunktivní hrany, tj. $D = \{e, \overline{e}\}$, kde hranu $\overline{e}$ nazýváme dvojčetem hrany e (a obráceně), a každá disjunktivní hrana $e \in E$ se nachází právě v jedné disjunktivní množině D. Zobecněný disjunktivní graf je téměř shodný s klasickým disjunktivním grafem, odlišuje se pouze tím, že jeho disjunktivní množiny mohou obsahovat dvě libovolné hrany, zatímco v případě klasického disjunktivního grafu disjunktivní množiny obsahují pouze hrany protichůdné $e = (v, w)$ a $\overline{e} = (w, v)$ pro nějaké $v, w \in V$.

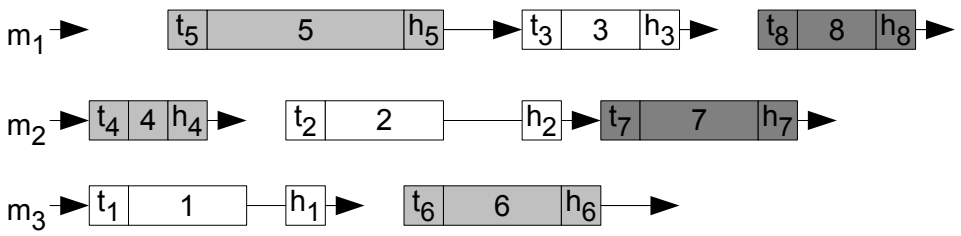
Podmnožinu disjunktivních hran $S \subseteq E$ nazýváme výběrem v G. Podgraf $G(S)$ vznikne z grafu G sloučením disjunktivních hran S s konjunktivními hranami A, tedy $G(S) = (V, A \cup S, c)$ (pro zjednodušení značení píšeme c, místo abychom zaváděli nějaké omezené c, které by obsahovalo váhy pouze pro hrany obsažené v $A \cup S$). Výběr S je kompletní v G, pokud obsahuje hranu z každé disjunktivní množiny $D \in \varepsilon$, a pozitivně acyklický v G, pokud $G(S)$ neobsahuje cyklus pozitivní délky. Výběr je proveditelný v G, pokud je kompletní a pozitivně acyklický.

Disjunktivní graf spojený s instancí GBJS je definován následovně. Množina vrcholů V obsahuje dva vrcholy pro každou operaci i, vrchol převzetí t_i a vrchol odevzdání h_i, takže $V = V^I \cup \{\sigma, \tau\}$ kde $V^I = \{t_i, h_i : i \in I\}$. Množina konjunktivních hran $A = A^0 \cup A^1 \cup A^{\sigma,\tau}$, kde A^0 je množina synchronizačních hran spojující v každém jobu J po sobě jdoucí operace $(i, j) \in A_J$ párem protichůdných hran s nulovou délkou, tedy $A^0 = \{(h_i, t_j), (t_j, h_i) : (i, j) \in A_J, J \in \mathcal{J}\}$, A^1 je množina hran zpracování, spojující v každé operaci vrchol převzetí a vrchol odevzdání, tedy $A^1 = \{(t_i, h_i) : i \in I\}$, a $A^{\sigma,\tau}$ obsahuje hrany spojující počáteční vrchol σ a koncový vrchol τ se všemi ostatními vrcholy, tedy $A^{\sigma,\tau} = \{(\sigma, t_i) : i \in I\} \cup \{(h_i, \tau) : i \in I\}$. Množina disjunktivních hran E obsahuje pro každé dvě operace $i, j \in I_m$ na stroji m dvě hrany spojující vrchol odevzdání operace i s vrcholem přijetí operace j a obráceně, tedy $E = \{(h_i, t_j), (h_j, t_i) : i, j \in I_m, i \neq j, m \in M\}$. Disjunktivní množina D obsahuje jeden pár hran $\{(h_i, t_j), (h_j, t_i)\}$ pro operace $i, j \in I_m, i \neq j$ a skupina disjunktivních množin $\varepsilon = \{\{(h_i, t_j), (h_j, t_i)\} : i, j \in I_m, i \neq j, m \in M\}$. Konečně hrany $e \in A \cup E$ mají váhy c_e definované následovně: $c_e = 0$ pro $e \in A^0$, $c_e = d_i^t + d_i$ pro $e = (t_i, h_i) \in A^1$, $c_e = d_{\sigma i}^s$ pro $e = (\sigma, t_i) \in A^{\sigma,\tau}$, $c_e = d_i^h + d_{i\tau}^s$ pro $e = (h_i, \tau) \in A^{\sigma,\tau}$ a $c_e = d_i^h + d_{ij}^s$ pro $e = (h_i, t_j) \in E$.

GBJS formulován pomocí G: Mezi všemi proveditelnými výběry v G najděte výběr S, který minimalizuje délku nejdelší cesty z σ do τ v $G(S)$.

Všimněme si, že jakákoliv disjunktivní množina $D = \{(h_i, t_j), (h_j, t_i)\}$ vytvoří spolu s hranami $(t_i, h_i), (t_j, h_j) \in A^1$ v grafu $G(D)$ cyklus pozitivní délky. Z toho vyplývá, že v každém proveditelném výběru musí být obsažena nejvýš jedna hrana z každé disjunktivní množiny.

Disjunktivní graf je možné převést do kompaktnějšího tvaru, který je sice méně názorný, ale přináší nižší paměťovou náročnost algoritmu a umožňuje efektivnější výpočet nejdelší cesty v grafu. Nejdříve je možné dvojice vrcholů spojené párem synchronizačních hran nulové délky sloučit do jednoho vrcholu. Dále je možné u každého jobu sloučit vrcholy představující poslední operaci. V tom případě je nutné k vahám hran vedoucích z posledního vrcholu připočíst váhu odebrané procesní hrany. V implementaci algoritmu byla použita tato kompaktnější reprezentace.



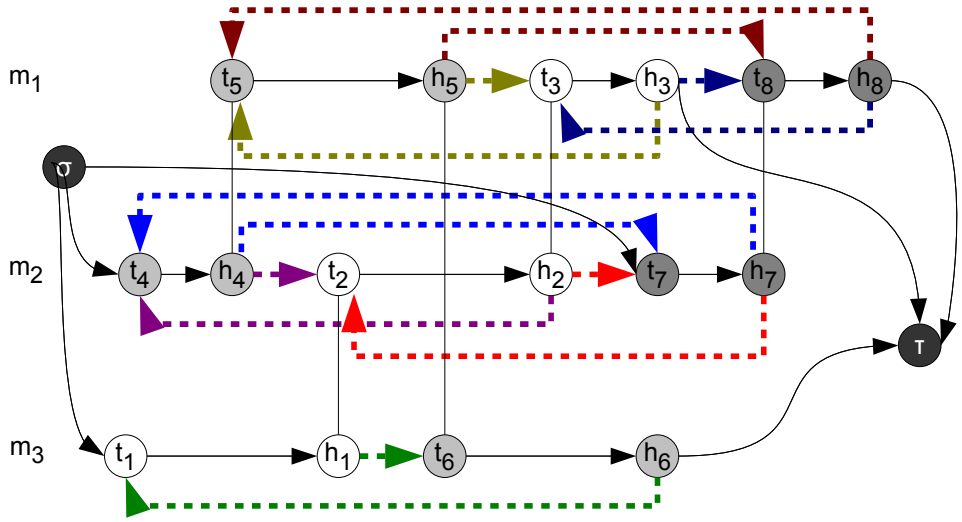
Obrázek 2: Ganttův diagram

2.3 Příklad

Následuje příklad ilustrující popis problému. Mějme instanci GBJS se třemi stroji $M = \{m_1, m_2, m_3\}$, třemi joby $\mathcal{J} = \{J, K, L\}$ a osmi operacemi $I = \{1, 2, \ldots, 8\}$. Joby jsou určeny množinami svých operací $J = \{1, 2, 3\}$, $K = \{4, 5, 6\}$, $L = \{7, 8\}$, kde např. pro job J posloupnost operací je $J_1 = 1$, $J_2 = 2$ a $J_3 = 3$ a množina uspořádaných dvojic po sobě jdoucích operací je $A_J = \{(1, 2), (2, 3)\}$. Operace jsou určeny na stroje následovně: $I_{m_1} = \{3, 5, 7\}$, $I_{m_2} = \{2, 4, 8\}$ a $I_{m_3} = \{1, 6\}$, takže job J je nejdříve zpracován na stroji m_3, poté na stroji m_2 a nakonec na stroji m_1.

Číselné hodnoty jsou následující: trvání kroků zpracování d_i pro $i \in I$ jsou $(d_1, \ldots, d_8) = (30, 50, 30, 20, 50, 30, 40, 20)$, trvání kroků převzetí a kroků odevzdání jsou $d_i^t = d_i^h$=10 pro $i \in I$, a časy nastavení strojů jsou $d_{53}^s = 20$, $d_{38}^s = 10$, $d_{42}^s = 20$, $d_{27}^s = 10$, $d_{16}^s = 10$ a $d_{35}^s = d_{83}^s = d_{58}^s = d_{85}^s = d_{24}^s = d_{72}^s = d_{47}^s = d_{74}^s = d_{61}^s = 0$.

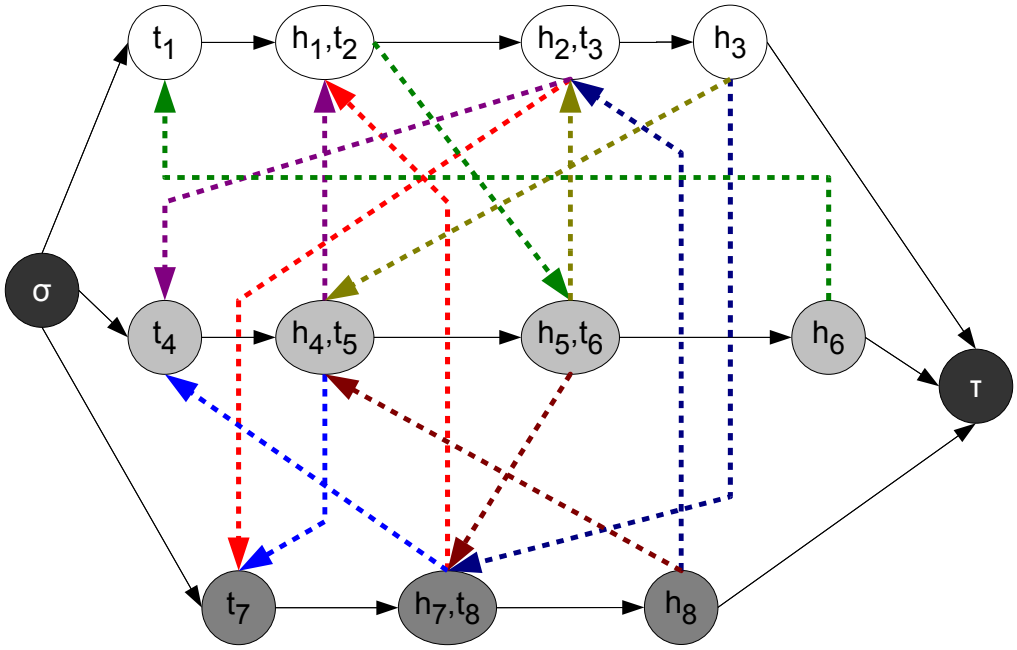
Obrázek 2 zobrazuje Ganttův diagram pro uvedenou instanci GBJS. Každá operace zahrnuje čtyři kroky: např. operace 1 zahrnuje krok převzetí o délce d_1^t, když je job umisťován na stroj, krok zpracování o délce d_1, kdy je job na stroji upravován, krok čekání (značený jako neorientovaná úsečka) kdy job blokuje stroj m_1 při čekání na uvolnění stroje m_2 a krok odevzdání o délce d_1^h, když je job přesunut ze stroje m_1 na stroj m_2. Krok odevzdání operace 1 a krok převzetí operace 2 jsou synchronyzovány, probíhají ve stejném časovém úseku. Doby nastavení strojů závislé na pořadí operací jsou zobrazeny jako šipky. Např. nastavení stroje m_3 mezi operacemi 1 a 6 je hotové před ukončením operace 5 na stroji m_1, takže job může být ihned



Obrázek 3: Disjunktivní graf

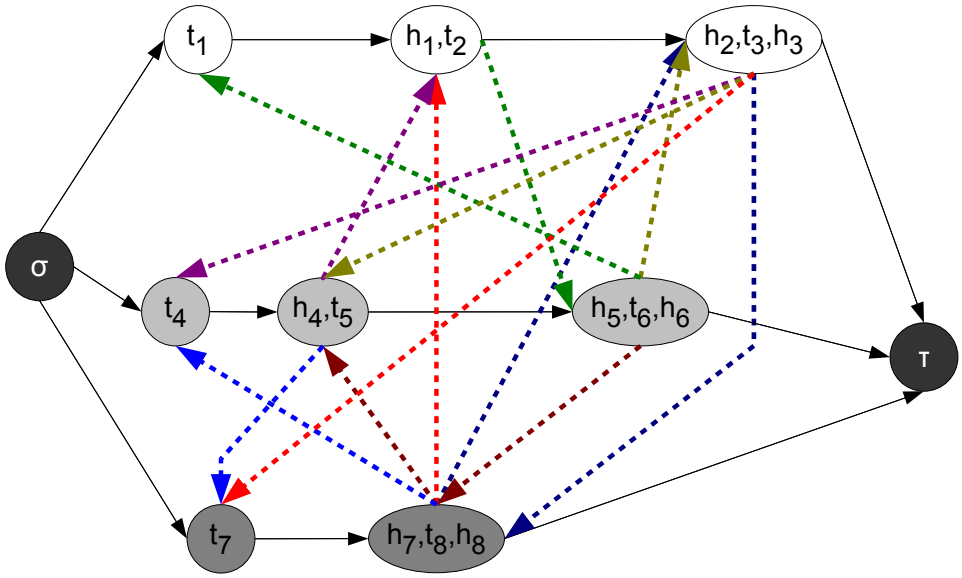
přesunut a krok čekání operace 5 má nulovou délku.

Na obrázku 3 je pro uvedenou instanci GBJS zobrazen disjunktivní graf $G = (V,\ A,\ E,\ \varepsilon,\ c)$. Pro zjednodušení nejsou uvedeny číselné hodnoty délek jednotlivých operací. Dvojice protichůdných synchronizačních hran A^0 nulových délek byly nahrazeny neorientovanými hranami. Disjunktivní hrany E jsou zobrazeny čárkovaně a disjunktivní množiny D (obsahující vždy dvě disjunktivní hrany) jsou navzájem odlišeny barvami. Graf G je definován následovně: množina vrcholů $V = \{t_i, h_i : i = 1, 2, \ldots, 8\} \cup \{\sigma, \tau\}$, množina konjunktivních hran $A = A^0 \cup A^1 \cup A^{\sigma,\tau}$, kde množina synchronizačních hran nulových délek $A^0 = \{(h_1, t_2), (t_2, h_1), (h_2, t_3), (t_3, h_2), (h_4, t_5), (t_5, h_4), (h_5, t_6), (t_6, h_5), (h_7, t_8), (t_8, h_7)\}$, množina hran zpracování s kladnými délkami $A^1 = \{(t_i, h_i) : i = 1, 2, \ldots, 8\}$ a množina $A^{\sigma,\tau}$, obsahující hrany spojující vrchol σ s prvním vrcholem každého jobu a hrany spojující poslední vrchol každého jobu s vrcholem τ, obsahuje $\{(\sigma, t_1),\ (\sigma, t_4),\ (\sigma, t_7),\ (h_3, \tau),\ (h_6, \tau),\ (h_8, \tau)\}$. Skupina disjunktivních množin $\varepsilon = \{\{(h_1, t_6), (h_6, t_1)\},\ \{(h_2, t_7), (h_7, t_2)\},\ \{(h_4, t_2), (h_2, t_4)\},\ \{(h_4, t_7), (h_7, t_4)\},\ \{(h_3, t_5), (h_5, t_3)\},\ \{(h_3, t_8), (h_8, t_3)\},\ \{(h_5, t_8), (h_8, t_5)\}$. Množina disjunktivních hran $E = \bigcup_{D \in \varepsilon} D$. Rozvrh zobrazený na obrázku 2 odpovídá přípustnému výběru $S = \{(h_1, t_6),\ (h_4, t_2),\ (h_2, t_7),\ (h_4, t_7),\ (h_5, t_3),\ (h_3, t_8),\ (h_5, t_8)\}$.



Obrázek 4: První redukce disjunktivního grafu

Na obrázku 4 je zobrazen disjunktivní graf po první redukci, kdy byly sloučeny vždy dva vrcholy spojené synchronizačními hranami nulových délek do jednoho vrcholu. V tomto případě už se nedá určit, který vrchol patří na který stroj (sloučené vrcholy patřily na jiné stroje), a proto jsou vždy v řádku vrcholy patřící k jednomu jobu, namísto vrcholů patřících na jeden stroj, jak to bylo na obrázku 3. Při této redukci grafu zůstávají váhy hran nezměněny.



Obrázek 5: Druhá redukce disjunktivního grafu

Na obrázku 5 je zobrazen disjunktivní graf po obou redukcích, kdy oproti první redukci byly navíc sloučeny vrcholy představující poslední operaci každého jobu. V tomto případě je již potřeba upravit váhy některých hran. Ke každé hraně vycházející z před sloučením posledního vrcholu v jobu je potřeba přičíst váhu sloučené hrany. Např. u prvního jobu (s bílými vrcholy) slučujeme vrchol h_2, t_3 s vrcholem h_3. Váhu hrany, která tyto dva vrcholy spojuje, musíme přičíst k hranám $(h_3, \tau), (h_3, h_4, t_5), (h_3, h_7, t_8)$.

3 Popis algoritmu

Použitý algoritmus řešící problém GBJS je metaheuristika využívající lokální prohledávání s tabu seznamem.

3.1 Lokální prohledávání

Lokální prohledávání je iterativní proces, který postupuje od jednoho řešení k dalšímu, dokud není splněna ukončující podmínka [12]. Aby bylo hledání systematické, přípustné pohyby od jednoho řešení k dalšímu jsou omezeny na sousední řešení. Pro každé řešení $s \in S$ množina $\mathcal{N}(s)$ označuje podmnožinu všech řešení, a obsahuje všechna řešení, která mohou být dosažena z s v jednom kroku. Množina $\mathcal{N}(s)$ se nazývá sousedství s. Nalezené sousedy, prvky množiny $\mathcal{N}(s)$, můžeme ohodnotit nějakou cílovou funkcí $c(s)$. Lokální prohledávání pak může být popsáno následovně: V každé iteraci začínáme s jedním řešením $s \in S$, pro toto řešení nalezneme jeho sousedy, ohodnotíme je cílovou funkcí $c(s)$ a vybereme souseda s nejlepším ohodnocením. Toho použijeme jako výchozí řešení pro další iteraci. Podle způsobu výběru souseda rozlišujeme několik variant lokálního prohledávání:

- Metoda největšího stoupání (hill climbing): jako nové řešení je vždy vybrán soused s nejlepším ohodnocením
- Náhodná procházka (random walk): podobně jako metoda největšího stoupání, ale s určitou (malou) pravděpodobností je proveden náhodný skok ve stavovém prostoru
- Simulované žíhání (simulated annealing): podobně jako náhodná procházka, ale pravděpodobnost náhodných skoků se s časem snižuje (simulace ochlazování kovu)
- Tabu prohledávání (tabu search)

3.2 Tabu prohledávání

Základní lokální prohledávání (metoda největšího stoupání) je rozšířeno o tabu seznam, ve kterém jsou uchovávána již použitá řešení. Tato řešení

nejsou při výběru nejlepšího souseda uvažována. Tím je zabráněno zacyklení při hledání. Kvůli paměťové náročnosti není často možné uchovávat všechna použitá řešení, proto je uchováváno jen několik posledních. Množství je kompromisem mezi již zmíněnou paměťovou náročností a potřebou zábránit zacyklení, protože tabu seznamem je zabráněno pouze výskytu zacyklení kratších, než je délka tabu seznamu. Dále je možné neuchovávat v tabu seznamu celá řešení, ale jen jejich typické vlastnosti. Ze sousedů jsou pak vyloučena všechna řešení, která tyto vlastnosti mají. To má nevýhodu, že mohou být vyloučena i ještě neprozkoumaná řešení. K eliminaci této nevýhody se používají tzv. aspirační kritéria, která definují případy, kdy je možné řešení s vlastností z tabu seznamu přijmout.

3.3 Popis algoritmu

V použitém algoritmu je často nutné ověřit přípustnost výběru disjunktivních hran v grafu a najít kritickou cestu v grafu. Připomeňme si, že přípustný výběr je kompletní a neobsahuje cyklus kladné délky. Kompletnost výběru snadno ověříme pomocí kontroly výběru právě jedné hrany z každé disjunktní množiny $D \in \varepsilon$. Zda graf obsahuje cyklus, zjistíme pomocí topologického uspořádání vrcholů [26].

3.3.1 Postup nalezení topologického uspořádání vrcholů

1. Položme $i = 1$;
2. najdeme vstupní vrchol grafu (vstupní vrchol je takový, do kterého nevedou žádné hrany; v případě výskytu několika vstupních vrcholů vybereme jeden náhodně) a přiřadíme mu číslo i;
3. odebereme očíslovaný vrchol z grafu i se všemi hranami, které z něj vedou a inkrementujeme i o jedna;
4. opakujeme body 2 a 3, dokud existuje nějaký vstupní vrchol;
5. pokud jsou po provedení postupu všechny vrcholy grafu očíslované, graf je acyklický.

3.3.2 Postup nalezení kritické cesty v grafu

Hranu vedoucí z vrcholu i do vrcholu j značíme (i, j) a váhu takové hrany $w((i, j))$.

1. Očíslujeme vrcholy grafu G podle věty o acyklických grafech;
2. najdeme minimální časové ohodnocení $t(i)$ všech vrcholů (nejkratší možná doba pro dosažení uzlu):
   (a) vrcholu s číslem 1 přiřadíme $t(1) = 0$;
   (b) pro $i = 2, 3, \ldots, n$, kde n značí počet vrcholů, vrcholu i přiřadíme $t(i) = max\{t(j) + w((j, i))\}$;
3. najdeme maximální časové ohodnocení $T(i)$ všech vrcholů:
   (a) vrcholu s číslem n přiřadíme $T(n) = t(n)$;
   (b) pro $i = n-1, n-2, \ldots, 1$ vrcholu i přiřadíme $T(i) = min\{T(j) - w((i, j))\}$;
4. kritická cesta prochází těmi vrcholy i, pro něž $t(i) = T(i)$ a těmi hranami (i, j), pro něž $w((i, j)) = t(j) - t(i)$.

3.3.3 Generování sousedů

Sousedé jsou tvořeni výměnami disjunktivních hran obsažených ve výchozím grafu. V [7] je ukázáno, že zlepšení rozvrhu je možné dosáhnout pouze výměnou hrany, která leží na kritické (nejdelší) cestě v grafu. V každém kroku algoritmu je tedy ve výchozím grafu nalezena kritická cesta a pomocí výměny každé disjunktivní hrany na této cestě je vytvořen jeden soused. Pouhou výměnou disjunktivní hrany za její dvojče je však možné vytvořit v grafu cyklus. Proto je přípustnost generovaných sousedů ověřována a nepřípustní sousedé jsou opraveni.

Pseudokód algoritmu pro nalezení sousedů vypadá následovně:

1. Máme výchozí graf s přípustným výběrem $G(S)$;
2. ve výchozím grafu $G(S)$ najdeme kritickou cestu, $\lambda(S)$ značí množinu disjunktivních hran na této cestě;
3. pro každou disjunktivní hranu $e \in \lambda(S)$:

   (a) vytvoříme kopii (sousední graf) výchozího grafu;
   (b) v kopii vyměníme hranu e za její dvojče $\overline{e}$;
   (c) pokud jdou vrcholy sousedního grafu očíslovat, skončíme;
   (d) pokud sousední graf obsahuje cyklus, opravíme ho.

3.3.4 Oprava souseda

Oprava souseda vychází z toho, že nově vytvořený cyklus obsahuje právě přidanou hranu a tedy i některou z hran, která vede z nebo do vrcholů příslušejících k jobům, které nově přidaná hrana spojuje. Při opravě by tedy v grafu měly být vyměněny některé z hran vedoucí z nebo do vrcholů, které přísluší těmto dvěma jobům. Ve skutečnosti jsou vyměněny pouze některé z hran vedoucí z nebo do vrcholů, které přísluší jobu, do kterého vede přidávaná hrana. Tato modifikace algoritmu byla využita již ve zdrojové práci [11], kde také byly provedeny numerické testy ověřující její správnost.

Poté jsou do grafu přidány hrany z těch disjunktních množin, které nejsou v grafu obsaženy (jejich hrany byly v předchozím kroku odebrány). Aby se sousedé od původního grafu lišili co nejméně, snaží se algoritmus vrátit co nejvíce hran vyskytujících se v původním grafu. V první fázi cyklicky prochází odebrané hrany a hledá ty, které by při navrácení do grafu v něm vytvořily cyklus. Pokud takovou hranu najde, přidá do grafu její dvojče (druhou hranu z disjunktní množiny). V momentě, kdy při průchodu cyklem není přidána hrana z žádné ještě v grafu nepřítomné disjunktní množiny, přijde na řadu druhá fáze, ve které jsou do grafu vráceny z těchto množin hrany, které byly přítomné v původním grafu.

Algoritmus opravy souseda zapsaný v pseudokódu vypadá následovně:

1. Najdeme všechny vrcholy náležející k jobu, do kterého vede hrana vyměněná za hranu z kritické cesty (její dvojče);

2. odebereme z grafu všechny disjunktivní hrany, které vedou z nebo do těchto vrcholů;

3. vrátíme do grafu hranu, kterou jsme vyměnili za hranu z kritické cesty;

4. cyklicky procházíme množinu odebraných hran a pro každou hranu testujeme, zda nevytvoří v grafu s neúplným výběrem cyklus;

   (a) pokud hrana cyklus vytvoří, přidáme do grafu její dvojče a hranu odebereme z množiny odebraných hran;

   (b) pokud hrana cyklus nevytvoří, nepřidáme žádnou hranu;

5. pokud jsme při průchodu množinou odebraných hran již žádné dvojče nepřidali, vrátíme do grafu zbylé hrany z množiny odebraných hran.

3.3.5 Vlastní tabu algoritmus

V každé iteraci algoritmu je z výchozího řešení (výběru v grafu) vytvořena množina sousedních řešení. Pro pamatování posledních pohybů (výměn hran) používá algoritmus tabu seznam fixní délky který obsahuje hrany, které nesmí být při tvorbě sousedů použity (tzv. tabu hrany). Je-li tabu seznam plný při přidávání další hrany, je z něj odebrána hrana kterou obsahuje nejdéle.

Při výběru kandidáta na nové výchozí řešení ohodnotíme všechna sousední řešení a pak je postupně procházíme. Pokud soused nevznikl výměnou tabu hrany a je lepší než dosud nejlepší ze sousedů, zapamatujeme si ho jako dosud nejlepšího kandidáta. Pokud soused vznikl výměnou tabu hrany a není lepší než nejlepší dosud nalezený graf, ignorujeme ho. Pokud je lepší než nejlepší dosud nalezený graf, zapamatujeme si ho jako dosud nejlepšího kandidáta (vliv tzv. aspiračního kritéria).

Výběr nejlepšího kandidáta zapsaný v pseudokódu vypadá následovně:

1. Ohodnoť všechna sousední řešení (najdi v nich kritickou cestu a urči její délku);

2. nejKandidát := null;

3. postupně pro všechny sousední řešení proveď:
   (a) pokud soused vznikl pomocí hrany, která není v tabu seznamu;
      i. pokud je soused lepší než nejKandidát, ulož souseda do nejKandidát;
   (b) pokud soused vznikl pomocí tabu hrany;
      i. pokud je soused lepší než nejKandidát a zároveň je lepší než nejlepší dosud nalezený graf vůbec, ulož souseda do nejKandidát;
4. konec, nejKandidát obsahuje nejlepší sousední řešení.

Pokud při výběru nejlepšího kandidáta žádné vhodné řešení nenajdeme (všechny hrany z kritické cesty výchozího grafu jsou v tabu seznamu a grafy vytvořené s jejich pomocí nejsou lepší než nejlepší dosud nalezené řešení), použijeme souseda vzniklého výměnou té tabu hrany, která je v tabu seznamu nejdéle.

Dále je použit seznam ohraničené délky obsahující tzv. elitní řešení. Elitní řešení je nově nalezené řešení, které je kratší než dosud nejlepší nalezené řešení. Kdykoliv tedy v průběhu algoritmu dospějeme k řešení lepšímu, než je dosud nejlepší nalezený rozvrh, uložíme ho do tohoto seznamu. Když prohledávání po určitém množství iterací nezlepší dosud nejlepší nalezené řešení, přeruší tuto větev, vrátí se k poslednímu elitnímu řešení a pokračuje ve výpočtu od něj. Společně s elitním řešením je uchováván i seznam prozkoumaných hran, které již nesmí být použity, aby se výpočet ubíral jinou cestou. Společně s elitním řešením je uchováván i jemu odpovídající tabu seznam. Řešení je ze seznamu elitních řešení odebráno, pokud nemá žádnou neprozkoumanou hranu. Algoritmus obsahuje i detekci cyklů, která sleduje posloupnost délek výchozích řešení a v případě opakování těchto hodnot obnoví poslední elitní řešení.

Celý algoritmus zapsaný v pseudokódu s pomocí již uvedených algoritmů vypadá následovně:

1. Načti zadání;
2. vytvoř počáteční přípustný výchozí graf;

3. najdi ve výchozím grafu kritickou cestu, disjunktivní hrany z této cesty ulož do neprozkoumaných hran;
4. pro každou disjunktivní hranu z kritické cesty, která je v neprozkoumaných hranách proveď:
   (a) vytvoř kopii výchozího grafu a vyměň v ní určenou disjunktivní hranu z kritické cesty výchozího grafu za její dvojče;
   (b) pokud je vzniklý graf nepřípustný, oprav ho;
   (c) ulož nově vytvořený graf do seznamu kandidátů na nový výchozí graf;
5. ze seznamu kandidátů vyber toho nejlepšího;
6. pokud neexistuje nejlepší kandidát (všechny disjunktivní hrany z kritické cesty výchozího grafu jsou v tabu seznamu a žádný kandidát není lepší než nejlepší dosud nalezený graf), vytvoř náhradního nejlepšího kandidáta výměnou disjunktivní hrany z kritické cesty výchozího grafu, která je v tabu seznamu nejdéle;
7. hranu použitou k tvorbě nejlepšího kandidáta přidej do tabu seznamu;
8. pokud je výchozí řešení v seznamu elitních řešení, odeber z jeho seznamu neprozkoumaných hran hranu použitou k tvorbě nejlepšího kandidáta;
9. vymaž aktuální seznam neprozkoumaných hran a naplň ho disjunktními hranami z kritické cesty nejlepšího kandidáta;
10. pokud je kandidát lepší než dosud nejlepší nalezený graf, ulož ho i s jeho seznamem neprozkoumaných hran a tabu seznamem do seznamu elitních řešení;
11. pokud je detekováno opakování hodnot v posloupnosti délek výchozích grafů (zacyklení výpočtu), jako nový výchozí graf použij poslední elitní řešení (společně s obnovou jeho tabu seznamu a seznamu neprozkoumaných hran). Pokud již žádné elitní řešení neexistuje, nastav ukončovací příznak;

12. pokud již dlouho (předem určený počet iterací) nedošlo ke změně nejlepšího dosud nalezeného řešení, použij jako nový výchozí graf poslední elitní řešení (společně s obnovou jeho tabu seznamu a seznamu neprozkoumaných hran). Pokud již žádné elitní řešení neexistuje, nastav ukončovací příznak;

13. pokud je nastaven ukončovací příznak, nebo algoritmus dosáhl předem určeného počtu iterací, případně bylo dosaženo časového omezení, skonči;

14. jako nové výchozí řešení použij nejlepšího kandidáta, najdi v něm kritickou cestu a přejdi na bod 4.

4 Paralelizace algoritmu

Hlavní přínos této práce spočívá v paralelizaci algoritmu lokálního prohledávání s tabu seznamem [11]. V této kapitole je ukázáno, jaké možnosti se v této oblasti nabízejí a je popsána struktura výsledného algoritmu.

4.1 Paralelizace algoritmů lokálního prohledávání

Existují tři hlavní paralelní metody lokálního prohledávání [12]: paralelní prohledávání z více míst, paralelní průzkum sousedství a paralelní ohodnocování řešení.

4.1.1 Paralelní prohledávání z více míst

Metoda spočívá ve spuštění několika nezávislých, nebo spolupracujících instancí lokálního prohledávání. Může se jednat o jednu, nebo i o více různých variant lokálního prohledávání. Každá instance obvykle začíná v jiném počátečním řešení. Nejčastěji se používá varianta s nezávislými instancemi, protože je snadněji implementovatelná. Tato varianta má nejblíže k sekvenčnímu algoritmu, kdy N paralelních nazávislých instancí lokálního prohledávání dojde ke stejnému výsledku jako N běhů sekvenční varianty na jednom stroji. Paralelismus umožňuje zvýšit robustnost algoritmu [12]. V případě spolupracujících instancí dochází v průběhu výpočtu k výměně informací. Často se jedná o informaci o dobrých nalezených řešeních, o provedených pohybech a další informace umožňující sledovat postup hledání a eliminovat tak situace, kdy dvě větve prohledávají stejnou oblast.

4.1.2 Paralelní průzkum sousedství

Průzkum okolí probíhá na základě modelu farmář - pracovník (farmer - worker) a umožňuje zrychlit průzkum možných pohybů. Na začátku každé iterace algoritmu farmář pošle současné řešení svým pracovníkům, každý pracovník prozkoumá nějakou část okolí a výsledky odešle farmářovi. Model je efektivní, pokud je tvorba sousedního řešení časově náročná, nebo pokud jich existuje velké množství. U lokálního prohledávání ale efektivita tohoto postupu není zaručena, protože počet sousedů může být velmi proměnlivý.

4.1.3 Paralelní ohodnocování řešení

Pokud se ohodnocovací funkce skládá z několika oddělitelných částí, lze využít paralelizmu při ohodnocování jednotlivých řešení. Paralelní větve algoritmu počítají každá svou část a výsledky se pak sloučí do výsledného ohodnocení. Výhodné je to při velké časové náročnosti ohodnocení řešení.

4.2 Podpora vícevláknového programování v .NET frameworku

V .Net frameworku od firmy Microsoft se nachází jmenný prostor (namespace) *System.Threading* poskytující třídy a rozhraní umožňující vícevláknové programování [13]. Kromě tříd na synchronizaci vláken a přístupu ke sdíleným datům (*Mutex*, *Monitor*, *Interlocked*, *AutoResetEvent*, atd.), obsahuje i třídu *ThreadPool* umožňující použít zásobník vláken (pool), která jsou řízena běhovým prostředím. Následuje výpis a stručný popis nejdůležitějších tříd ze jmenného prostoru *System.Threading*:

Thread Její instance slouží k reprezentaci jednotlivých vláken. Kód, který má být vláknem vykonáván, je určen *ThreadStart* delegátem nebo *ParameterizedThreadStart* delegátem. *ParameterizedThreadStart* delegát umožňuje předat vláknu vstupní data. Třída dále umožňuje nastavení priorit vláken a získání informace o jejich stavu (zda vlákno běží, je blokováno, zda běží na popředí či na pozadí atd.). Vytvořené vlákno je spuštěno metodou *Start*, jeho vykonávání se pozastaví metodou *Sleep* a k předčasnému ukončení vlákna dojde po volání metody *Abort*.

Monitor Poskytuje nástroje k synchronizaci přístupu k objektům. To je řešeno pomocí jejich zamykání, kdy každý objekt může být zamknutý v jednom okamžiku jen jedním vláknem. Objekt může být odemknut pouze vláknem, kterým byl uzamknut. Zámek také může být využit k omezení přístupu k části kódu, jehož vykonávání více vlákny současně by mohlo způsobit chybu. Takovému kódu se říká kritická sekce. Při použití třídy *Monitor* se nevytvářejí její instance, ale volají se metody, kterým se jako parametr

předává objekt, se kterým mají pracovat. Pro zamknutí objektu je určena metoda *Enter*, pro odemknutí metoda *Exit*. Pro usnadnění práce s *Monitory* při zamykání kritické sekce existuje v jazyku C# konstrukce *lock* do jejíhož bloku se sekce uzavře. Tato konstrukce zastupuje volání metod *Enter*, blok *try* obsahující kritickou sekci kódu a blok *final* obsahující metodu *Exit*.

Mutex Zajišťuje podobnou funkci jako konstrukce *lock* v popisu třídy *Monitor* s tím rozdílem, že *Mutex* může být použit k synchronizaci vláken běžících v různých aplikacích. Pokud je použit k synchonizaci vláken z různých aplikací, musí mít přiděleno jméno, aby obě aplikace mohly přistupovat ke stejnému *Mutexu*. Pokud není tato vlastnost požadována, je vhodnější použít konstrukci *lock*, protože je rychlejší. *Mutex* se využívá např. pro zajištění běhu pouze jedné instance programu.

Interlocked Metody této třídy pomáhají chránit proti chybám způsobených rozvrhovačem při přepnutí kontextu během aktualizace sdílené proměnné, nebo při běhu dvou vláken na různých procesorech.

EventWaitHandle Tato třída umožňuje komunikaci vláken pomocí signálů. Typicky je jedno či více vláken blokováno pomocí instance této třídy dokud na ni neblokované vlákno nezavolá metodu *Set* a neuvolní tak jedno či více blokovaných vláken. Chování třídy *EventWaitHandle* závisí na způsobu resetování. Ten se určuje pomocí příznaku - buď *AutoReset* nebo *ManualReset* (viz třídy *AutoResetEvent* a *ManualResetEvent*)

AutoResetEvent Jedná se o podtřídu třídy *EventWaitHandle*. Umožňuje mezivláknovou komunikaci pomocí signálů. Typicky se používá při potřebě exkluzivního přístupu vláken ke zdroji. Po vytvoření je instance třídy *AutoResetEvent* v nesignalizujícím stavu. Vlákno se zablokuje a čeká na signál po zavolání metody *WaitOne* na instanci této třídy. Signál mu může poslat nějaké neblokované vlákno zavoláním metody *Set*, která nastaví instanci třídy *AutoResetEvent* do signalizujícího stavu a uvolní tak če-

kající vlákno. Instance třídy *AutoResetEvent* zůstává v signalizujícím stavu jen dokud neuvolní nějaké čekající vlákno. Pokud žádné vlákno na signál nečeká, instance počká na první vlákno volající metodu *WaitOne* a okamžitě ho uvolní. Volání metody *Set* se nesčítají, pokud tedy žádné vlákno nečeká na uvolnění a na instanci třídy je metoda *Set* zavolána několikrát, druhé a další volání metody *Set* nebudou mít žádný efekt, propuštěno bude jen jedno vlákno. Po uvolnění vlákna se instance automaticky resetuje do nesignalizujícího stavu. V případě více čekajících vláken je jako první uvolněno to vlákno, které metodu *WaitOne* zavolalo jako první.

ManualResetEvent Podtřída třídy *EventWaitHandle*, která se od *AutoResetEvent* liší tím, že instance se po uvolnění blokovaného vlákna do nesignalizujícího stavu nepřepíná automaticky, ale je nutné zavolat metodu *Reset*. Používá se v případech, kdy nějaké vlákno musí dokončit svou činnost dříve než mohou začít pracovat jiná vlákna. Když takové vlákno začne svou činnost, nastaví instanci třídy *ManualResetEvent* do nesignalizujícího stavu metodou *Reset*. Vlákna, která pak zavolají metodu *WaitOne* jsou blokována a čekají na signál. Když vlákno, které instanci zablokovalo, zavolá na tuto instanci metodu *Set*, odblokuje ji a dá zablokovaným vláknům signál, že mohou pokračovat ve své činnosti. Jakmile je jednou instance třídy *ManualResetEvent* v signalizujícím stavu, zůstává v něm do dalšího volání metody *Reset* a do té doby neblokuje vlákna volající metodu *WaitOne*.

Semaphore Omezuje počet vláken, které mohou využívat sdílený zdroj. Při vytváření instance této třídy jsou jí předávány dva parametry - volná kapacita semaforu a celková kapacita semaforu. Pokud chce vlákno použít sdílený zdroj, který je řízen semaforem, zavolá na instanci třídy *Semafor* metodu *WaitOne*. Pokud má semafor volnou kapacitu, vlákno propustí a sníží svoji volnou kapacitu o jedna. Pokud semafor nemá volnou kapacitu, je vlákno blokováno dokud jiné vlákno, které používalo sdílený

zdroj, kapacitu neuvolní zavoláním metody *Release*. V případě uvolnění kapacity semaforu s více čekajícími vlákny není nijak garantováno, které vlákno volnou kapacitu dostane.

4.3 Vlastní paralelní algoritmus

V algoritmu prezentovaném v této práci byla z variant uvedených v kapitole 4.1 použita varianta paralelního průzkumu sousedství. Hlavní vlákno - farmář - vytvoří tolik pomocných vláken - pracovníků - kolik je na kritické cestě výchozího řešení disjunktivních hran. Každému z pomocných vláken je jedna z těchto hran přiřazena a pomocí její výměny v grafu se pomocné vlákno snaží získat sousední řešení. Po výměně hrany a případné obnově přípustnosti grafu je v něm nalezena kritická cesta a tedy i délka celého sousedního řešení. Nalezená řešení jsou ukládána a po skončení všech pomocných vláken hlavní vlákno vybere nejvhodnější řešení pro další iteraci.

Fakt, že tato varianta paralelizace algoritmů lokálního prohledávání nemusí být kvůli proměnnému množství sousedních řešení efektivní, není podstatný, protože počet sousedů bude (zvlášť v případě velkých zadání) mnohem vyšší než počet jader procesoru. Aby v souvislosti s touto početní převahou pomocných vláken nedocházelo k častému přepínání kontextu procesoru, je počet aktivních vláken omezen pomocí instance třídy Semaphore na počet jader procesoru.

Kromě semaforu je z možností .NET frameworku pro paralelizaci využita konstrukce lock pro zamykání sdílených proměnných (seznamu nalezených sousedů) a pole instancí třídy AutoResetEvent, pomocí kterých pomocná vlákna signalizují, že dokončila svoji činnost. Hlavní vlákno čeká, dokud pomocná vlákna nedokončí své výpočty pomocí volání statické metody WaitAll třídy WaitHandle, které je jako parametr předáno právě pole s instancemi třídy AutoResetEvent.

Činnost hlavního vlákna zapsaná v pseudokódu:

1. Načti zadání;
2. vytvoř počáteční přípustný graf;
3. najdi v grafu kritickou cestu;

4. pro každou disjunktivní hranu z kritické cesty proveď:
   (a) vytvoř pomocné vlákno a jako vstupní parametr mu předej jednu disjunktivní hranu z kritické cesty;
5. počkej na signály od všech pomocných vláken oznamující dokončení jejich činností;
6. ze seznamu nalezených sousedních řešení (grafů) vyber nejlepší;
7. pokud nelze žádné řešení vybrat, vytvoř řešení pomocí disjunktivní hrany z kritické cesty výchozího řešení, která je nejdéle v tabu seznamu;
8. pokud byl detekován cyklus (opakující se délky výchozích řešení), případně při dlouhé době bez zlepšení nejlepšího dosud nalezeného řešení použij elitní řešení jako nové výchozí řešení. Pokud již žádné elitní řešení neexistuje, nastav ukončovací příznak;
9. pokud je nastaven ukončovací příznak, nebo byla dosažena hranice počtu iterací, případně bylo dosaženo časového omezení, skonči;
10. přejdi na bod 3.

Činnost pomocného vlákna zapsaná v pseudokódu:

1. Vezmi si semafor;
2. vytvoř kopii výchozího řešení;
3. v kopii vyměň disjunktivní hranu získanou od hlavního vlákna (vstupní parametr);
4. pokud je nové řešení nepřípustné, oprav ho;
5. zamkni si pro sebe seznam kandidátních řešení a ulož do něj nově získané řešení;
6. odemkni seznam kandidátních řešení;
7. pošli hlavnímu vláknu signál oznamující dokončení činnosti;
8. uvolni semafor.

5 Softwarové platformy vhodné pro implementaci nástroje pro plánování výroby s více uživateli

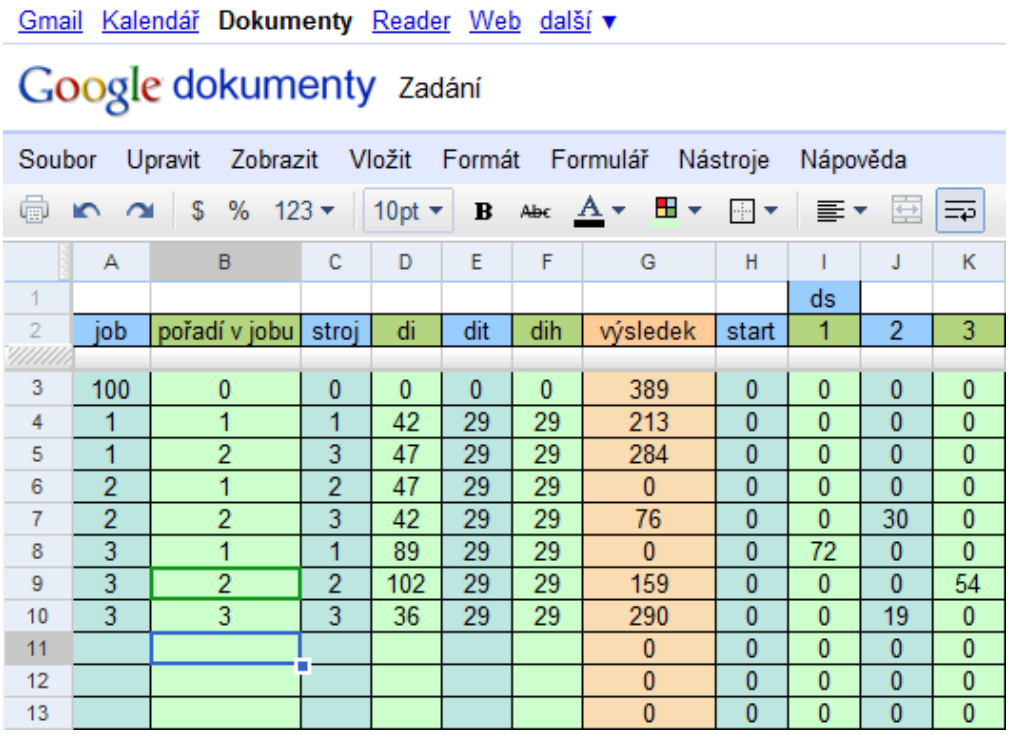
V této kapitole jsou popsány dvě softwarové platformy vhodné k implementaci nástroje pro plánování výroby s možností současné práce více uživatelů. Jedná se o Google Docs a o Microsoft SharePoint.

5.1 Google Docs

Google Docs je webová aplikace obsahující textový procesor, tabulkový procesor, nástroj na tvorbu prezentací a nástroj na tvorbu formulářů. Je součástí balíku webových služeb známých pod jménem Google Apps a umožňuje uživatelům vytvářet, upravovat a sdílet dokumenty online. Nachází se na adrese http://docs.google.com/ a v základní verzi je poskytována pro osobní i firemní použití zdarma firmou Google. Aplikace existuje i v Premier verzi, která stojí 40 € na rok za jeden uživatelský účet a která má garantovanou dostupnost 99,9% času [14].

Pro přístup ke službě v základní verzi stačí mít založen u Google uživatelský účet. Pro rozvrhování se dá využít hlavně tabulkový procesor jako uživatelské rozhraní, a to jak pro zadávání úloh, tak v omezené míře pro prezentaci výsledků výpočtů. Ukázka uživatelského rozhraní tabulkového procesoru je na obr. 6.

Dokumenty mohou být uživateli sdíleny. Sdílení je definováno na více úrovních, dokument tedy může být soukromý, může být sdílen s vybranými uživateli (identifikace pomocí účtu Google), anebo může být veřejný. Pokud je dokument veřejný, lze na něj odkazovat pomocí unikátní adresy. Dokument je možné sdílet jen k prohlížení, nebo i k úpravám. Dále lze určit, zda uživatel s právy k úpravám dokumentu (editor) má právo pozvat další uživatele k prohlížení, či k úpravám dokumentu. Při sdílení dokumentu aplikace umožňuje současnou práci více uživatelů najednou (na obr. 6 zelený kurzor patří druhému uživateli). Každý editor má svůj kurzor a vidí kurzory ostatních editorů a v reálném čase i jimi prováděné úpravy. Aplikace si pamatuje předešlé verze dokumentů, lze se k nim kdykoliv vrátit.



Obrázek 6: Ukázka uživatelského rozhraní tabulkového procesoru Google Docs

Do služby Google Docs je možné nahrávat i dokumenty vytvořené v jiných aplikacích. V případě tabulek je možný import dokumentů typu: s hodnotami oddělenými čárkami (.csv), soubory Microsoft Excel (.xls a .xlsx) a tabulky ve formátu OpenDocument (.ods), a to až do velikosti 1 MB. V podmínkách používání služby je napsáno, že sešity mají omezení v podobě maximálně 256 sloupců, 200000 buněk nebo 100 listů podle toho, které omezení je dosaženo jako první. Počet řádků omezen není. Dále je omezen počet sešitů na 1000 s tím, že otevřeno najednou jich může být maximálně 11.

Do dokumentů je možné přidávat tzv. miniaplikace (gadgets), což jsou malé aplikace, které mohou pracovat s obsahem tabulky. Jedná se o XML soubor který, popisuje strukturu miniaplikace [18]. Ta dále může obsahovat HTML část pro popis statických prvků gadgetu a může používat i JavaScipt pro dynamické prvky gadgetu. Celý kód miniaplikace může být v jednom souboru, nebo v něm mohou být URL odkazy na části umístěné jinde. Miniaplikace se dají použít i v jiných částech balíku Google Apps pro různé účely (zobrazení map, jednoduché hry atd.), v tabulkovém procesoru se nejčastěji

používají pro zobrazení dat v grafu.

Firma Google poskytuje Data API [19] k přístupu do aplikace z klientských aplikací, které ke komunikaci využívá čisté XML/HTTP. Pro usnadnění vývoje klientských aplikací Google poskytuje i 4 knihovny pro jazyky Java, .NET, PHP a Python. V této práci byla využita knihovna pro .NET.

5.1.1 Přístup do Google Docs pomocí knihovny pro .NET

Pro využití knihovny je potřeba mít Microsoft Visual Studio 2005 nebo 2008, z adresy http://code.google.com/p/google-gdata/downloads/list stáhnout a nainstalovat Google Data API SDK a do projektu s klientskou aplikací přidat reference na DLL knihovny, které budou potřeba (v našem případě se jedná o Google.GData.Extensions.dll, Google.GData.Client.dll a Google.GData.Spreadsheets.dll). V programu je pak potřeba použít direktivy using Google.GData.Client; using Google.GData.Extensions; a using Google.GData.Spreadsheets;. Třída SpreadsheetsService představuje připojení klientské aplikace k tabulkovým službám. Parametrem při vytváření její instance je textový řetězec ve tvaru: jméno společnosti-jméno aplikace-číslo verze. Po vytvoření instance třídy je nutné zadat přihlašovací údaje ke Google účtu a pak je již možné přistupovat k tabulkám. Součástí dokumentace ke knihovně [15] je i ukázkový příklad, který obsahuje následující kód demonstrující použití knihovny pro výpis všech sešitů a v nich obsažených listů, ke kterým má uživatel přístup.

```
private static void
PrintAllSpreadsheetsAndWorksheets(SpreadsheetsService service)
{
    SpreadsheetQuery query = new SpreadsheetQuery();
    SpreadsheetFeed feed = service.Query(query);
    Console.WriteLine("Your spreadsheets:");
    foreach (SpreadsheetEntry entry in feed.Entries)
    {
        Console.WriteLine("Spreadsheet: {0}",
            entry.Title.Text);
        PrintAllWorksheets(service, entry);
```

```
    }
}
```

Pomocí knihovny lze:

- Vytvářet a mazat listy v sešitech a upravovat jejich vlastnosti
- Přidávat, mazat a aktualizovat řádky a měnit jejich pořadí
- Získat záznamy v řádcích podle dat, která obsahují
- Měnit obsah jednotlivých buněk

5.1.2 Implementace uživatelského rozhraní

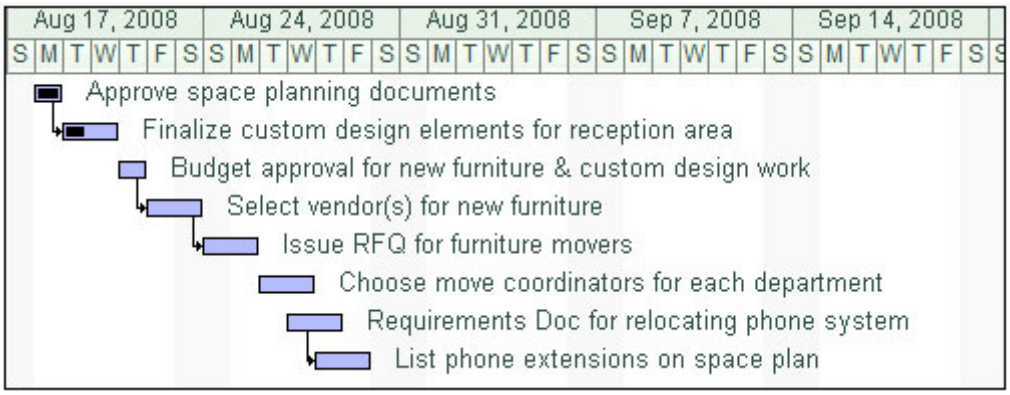
V rámci práce bylo vytvořeno jednoduché uživatelské rozhraní pro zadání vstupu pro algoritmus a pro prezentaci jeho výsledků. Ukázku je možné vidět na obr. 6. Do řádků uživatelé zadají parametry jednotlivých operací. Jsou to: číslo jobu, do kterého operace patří, její pořadí v jobu, číslo stroje na kterém má být vykonána, délka trvání úprav, trvání umístění operace na stroj a trvání odebrání operace ze stroje. Další sloupec je připraven pro zapsání výsledků - časů začátků operací. Pak následuje matice s časy nastavení stroje při výměně operace, jejíž index odpovídá sloupci za operaci, jejíž index odpovídá řádku. První řádek je vyhrazen pro poslední (prázdnou) operaci značící konec rozvrhu.

Součástí tabulky je miniaplikace pro spuštění rozvrhovacího algoritmu. Jedná se o jednoduchou aplikaci s tlačítkem odkazujícím na php soubor umístěný na serveru, jehož funkcí je spuštění programu na výpočet rozvrhu. Kód miniaplikace vypadá následovně:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
    <ModulePrefs title="Button" height="250"
        scaling="false" />
    <Content type="html">
        <![CDATA[
        <form method="GET"
```

```
                    action="http://adresa.serveru/index.php" >
                <input type="submit" value="Odeslat">
            </form>
            ]]>
        </Content>
    </Module>
```

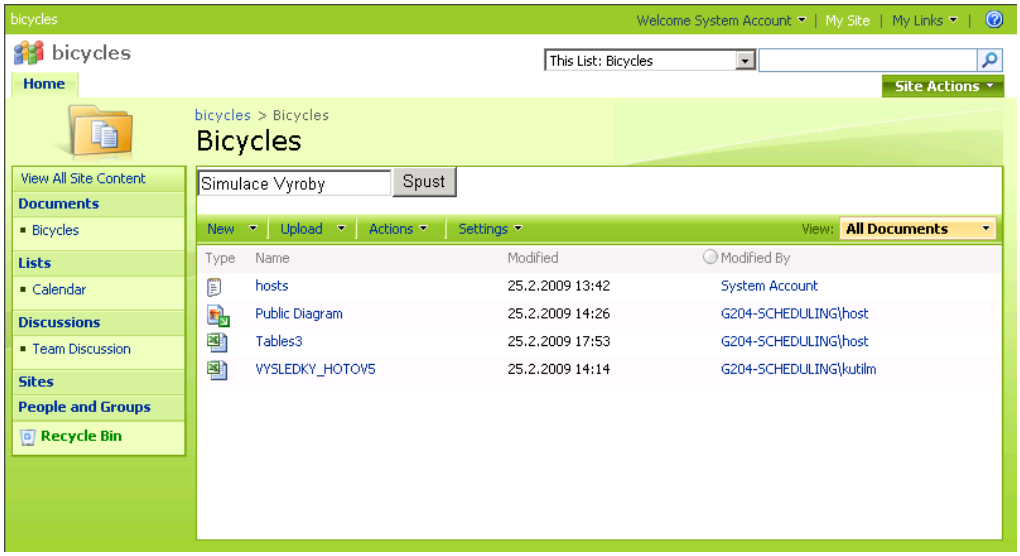
Aplikace na serveru si sama z tabulky načte zadání, vypočte výsledek, který zase do tabulky zapíše. Výsledný rozvrh je možné reprezentovat v miniaplikaci jako např. Ganttův diagram (ukázka Ganttova diagramu připraveného ve službě Docs pro prezentaci stavu projektu na obr. 7).



Obrázek 7: Ukázka Ganttova diagramu v tabulce Google Docs (převzato z [27])

5.2 Microsoft SharePoint

Nezkrácený název zní Microsoft SharePoint Products and Technologies. Jedná se o sadu produktů a technologií určených pro použití v oblasti portálů a spolupráce [16]. Základní platforma se nazývá Windows SharePoint Services 3.0 (WSS 3.0), při vlastnictví produktu Windows Server 2003 je k dispozici zdarma. Existuje několik služeb, které rozšiřují její funkcionalitu a na které je potřeba mít zvláštní licenci. Jsou to např. Microsoft Office SharePoint Server 2007, Search server 2008, Forms Server 2007, Microsoft Office Groove Server 2007 a další. Pro správu celého SharePoint řešení je k dispozici Microsoft Office SharePoint Designer, ve kterém lze např. upra-



Obrázek 8: Ukázka webové stránky Windows SharePoint Services 3.0 (převzato z [17])

vovat vzhled webů vytvořených pomocí šablon. Úprava je prováděna pomocí myši, není nutná znalost programování. MS SharePoint používá webové rozhraní (ukázka na obr. 8) dostupné pomocí webového prohlížeče. Ačkoliv jsou podporovány všechny hlavní webové prohlížeče, plně integrovaný a schopný využít všech funkcí je pouze Internet Explorer. Stránky MS SharePointu jsou webové aplikace založené na ASP.NET 2.0 využívající webového serveru IIS a databázi SQL Server. Mohou být nastaveny, aby vypadaly jinak (a měly jinou funkci) v Intranetu, Extranetu a na Internetu.

Windows SharePoint Services 3.0 se skládají z několika částí:

- Základního rámce obsahujícího objektový model, trvalé úložiště obsahu a konfiguraci databází na SQL serveru a ASP.NET ovládací prvky pro prezentaci obsahu.
- Konfigurační webové stránky pro správu webové farmy skládající se z jednoho nebo více serverů hostujících jednu nebo více webových stránek.
- Vestavěných šablon webů, které mohou být využity pro rychlou tvorbu vlastních webů a pro přidávání stránek a knihoven do již existujících webů.

- Indexeru, který prochází obsahem databáze a indexuje ho pro rychlé vyhledávání.
- Sbírku dalších šablon webů a stránek, známou pod jménem „Application Templates" obsahujících další logiku.
- Konfiguračního průvodce, který pomáhá rychle zprovoznit základní stránku.

Všechny šablony lze upravovat a dosáhnout tak požadovaného vzhledu a funkčnosti. Součástí stránky může být i knihovna dokumentů, a to jak sdílená knihovna, tak soukromá. Knihovna může obsahovat soubory jakéhokoliv typu. WSS 3.0 umožňuje vracet se k předchozím verzím dokumentů. Editace dokumentu probíhá v editoru nainstalovaném na lokálním počítači, a to tak, že je dokument otevřen buď přímo ze serveru nebo, je stažena jeho kopie. V obou případech je na serveru zamknut proti změnám a ostatní uživatelé nemohou vidět prováděné změny. Po dokončení editace dokumentu je uložen zpět na server a uvolněn pro úpravy dalších uživatelů. U dokumentů sady Microsoft Office lze soubory rezervovat a uvolňovat přímo z editoru.

Přidání dalších prvků na web (např. správu dokumentů, obrázků, formulářů) může přidat jakýkoliv uživatel s dostatečným oprávněním. Může přidávat buď knihovny poskytované WSS 3.0 či Microsoft Office SharePoint Serverem 2007, nebo může použít takzvané webové části (web parts). Jako příklad implicitně připravených webových částí lze uvést různé seznamy, diskuze, kalendář s úkoly atd., je však možné si vytvořit vlastní webové části pomocí Microsoft Visual Studia. Webová část se chová jako klasická .NET aplikace, lze v ní tedy pracovat se soubory, ať už fyzicky uloženými na disku serveru, či nahranými v SharePointu. Lze také použít „Excel services" pro přístup k tabulkám Excelu uloženým na SharePoint serveru.

5.3 Porovnání technologií

Pří výběru z obou technologií bude záležet na požadavcích výsledné aplikace a zákazníka, který ji bude využívat. MS SharePoint je v porovnání s Google Docs mnohem univerzálnější nástroj s velkými možnostmi přizpůsobení. Na druhou stranu je také mnohem náročnější na pořízení, instalaci a

údržbu. U Google Docs je výhoda v tom, že služba nefunguje na hardwaru zákazníka, ten se tedy o něj nemusí starat, stejně jako o aktualizace a konfiguraci software. To však může být pro řadu zákazníků z oblasti průmyslu nevýhoda, protože nemohou ovlivnit dostupnost aplikace. Ovšem v případě pořízení Premier účtu by tato překážka mohla padnout. Další nevýhodou Google Docs je omezení velikosti dokumentu, kdy hlavně počet sloupců nemusí pro rozsáhlá zadání stačit.

V rámci této práce bylo úspěšně vyzkoušeno použití technologie Google Docs. Služba pracovala spolehlivě a poskytovaná knihovna pro jazyk C# je přehledná a s dobrou dokumentací. Projevily se však dvě nevýhody použití této služby, a to omezený počet sloupců tabulky a zpomalování aplikace se vzrůstajícím počtem tabulek. Ukázka implementovaného rozhraní je na obr. 6. Výsledky jsou prezentovány jednoduše v oranžově zvýrazněném sloupci, jehož hodnoty představují rozvržené začátky provádění operací zapsaných v řádcích. Bylo by možné vytvořit gadget, který by výsledky prezentoval graficky, ale těžiště práce leží ve vývoji algoritmu, proto od toho autor upustil.

6 Experimentální měření rychlosti

Součástí práce bylo měření zlepšení algoritmu, a to jak ve smyslu zrychlení výpočtu, tak ve smyslu zlepšení výsledných rozvrhů. Testy byly prováděny na dvou počítačích, jeden disponoval čtyřjádrovým procesorem, druhý dvoujádrovým.

6.1 Vstupní zadání pro měření

Jako vstupní data bylo použito 72 standardních benchmarků používaných pro job shop problémy. Konkrétně se jednalo o ft01/04/06/10/20 od Fishera a Thompsona [21], abz5-9 navržený Adamsem a dalšími [20], la01-40 od Lawrence [22], orb01-10 od Applegatea a Cooka [23], swv01-05/10-12 od Storera a dalších [24] a yn1-4 od Yamady a Nakano [25]. Z důvodu omezeného přístupu k počítači se čtyřjádrovým procesorem a zdržením způsobeným laděním algoritmu nebyly změřeny na tomto počítači všechny benchmarky. Vynechány byly la23-40, swv01-05/10-12, abz7-9 a yn1-4. Celkem tedy bylo na čtyřjádrovém procesoru změřeno 32 benchmarků.

Jelikož benchmarky jsou navržené pro základní úlohu job shop, bylo je nutné pro úlohu zobecněného job shopu doplnit o časy umístění jobu na stroj, odebrání jobu ze stroje a časy nastavení strojů mezi operacemi. Tyto časy byly nastaveny pevně, a to na hodnotu 10 pro umístění a odebrání jobu na a ze stroje a na hodnotu 25 pro nastavení strojů při výměnách operací.

6.2 Implementační detaily

Algoritmus byl kvůli možnosti využít knihovnu pro komunikaci se službou Google Docs implementován v jazyku C#. Při měření kvalit algoritmu nehrála komunikace s uživatelským rozhraním roli, proto byla tato část nahrazena a zadání se načítalo ze souboru z lokálního disku. Benchmarky jsou uložené jednotlivě v souborech ve formátu dat, který se liší od formátu dat s kterými pracuje algoritmus, musely být proto přetransformovány a doplněny. Tato transformace se do času běhu algoritmu nezapočítávala.

Počáteční přípustné řešení bylo vytvořeno následovně. Algoritmus postupně prochází disjunktivní hrany, zkouší vždy jednu z dvojice přidat do neúplného grafu, otestuje přípustnost takového výběru a pokud detekuje cyklus, hranu z grafu odebere a vloží do něj její dvojče. Takovým postupem bylo získáno pokaždé stejné počáteční řešení a jelikož i ostatní parametry zadání jsou stejné, výpočet trval vždy stejnou dobu. Doba trvání výpočtu každého zadání proto byla změřena jen jednou.

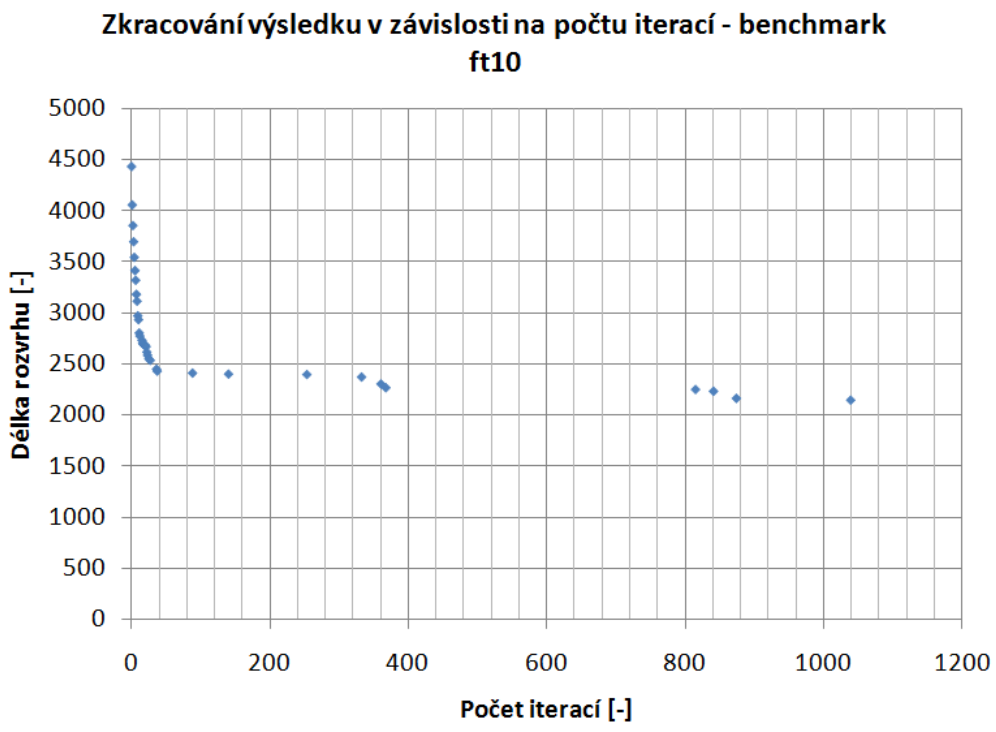
Běh algoritmu byl omezen buď na 20000 iterací, nebo na 10 minut podle toho, kterého omezení bylo dosaženo dříve. Zvlášť u menších zadání nebylo dosaženo ani jednoho omezení, a přesto algoritmus ukončil svou činnost. To je způsobeno vyčerpáním seznamu elitních řešení, která se používají při detekci cyklu nebo při velkém počtu iterací bez zlepšení dosud nejlepšího výsledku.

6.3 Výsledky měření

V tabulkách 1 a 2 jsou uvedeny výsledky měření na čtyřjádrovém, resp. dvoujádrovém procesoru. V prvním sloupci je vždy uveden název benchmarku, ve druhém jeho velikost, kdy první číslo vyjadřuje počet jobů v úloze a druhé číslo počet strojů, na které mají být joby rozvrženy. Další tři sloupce obsahují údaje o výsledné délce rozvrhu. V prvním sloupci je uveden výsledek paralelní verze algoritmu, ve druhém sloupci výsledek sekvenční verze algoritmu a ve třetím sloupci je uvedeno o kolik procent je výsledek paralelního algoritmu oproti sekvenčnímu lepší ($(par. - sekv.)/sekv. * 100$). Výsledky jsou uvedeny bezrozměrně, jsou v takových časových jednotkách, v jakých jsou uvedeny časy v zadání. Další dva sloupce obsahují délky dob výpočtů. V tomto případě nemá smysl uvádět rozdíl mezi paralelní a sekvenční variantou, protože u většiny zadání byl výpočet ukončen při dosažení desetiminutové hranice. Pokud je někde desetiminutová hranice o několik vteřin překročena, je to z toho důvodu, že byla vždy dokončena právě rozpracovaná iterace. V posledních třech sloupcích jsou uvedeny počty iterací algoritmu, tedy kolikrát bylo prozkoumáno okolí současného řešení a vybrán nejlepší soused jako nové výchozí řešení do další iterace. V prvním sloupci je opět uveden výsledek pro paralelní verzi algoritmu, ve druhém pro sekvenční a ve třetím je v procentech uvedeno zlepšení paralelního algoritmu (výpočet

je stejný jako u výsledné délky rozvrhu). Konečně v posledním řádku jsou uvedeny průměry z údajů o zlepšení výsledků a z údajů o zvětšení počtu iterací algoritmu při použití paralelní verze.

Na obrázku 9 je zobrazen graf znázorňující zkracování výsledného rozvrhu v závislosti na počtu iterací algoritmu pro zadání ft10. Je patrné, že k nejrychlejšímu zkracování dochází v počáteční fázi, ale je také vidět, že algoritmus je schopný opustit lokální minimum a dosáhnout lepšího rozvrhu i v pokročilé fázi výpočtu.



Obrázek 9: Ukázka zlepšování rozvrhu v závislosti na počtu iterací - benchmark ft10

Tabulka 1: Výsledky měření benchmarků na čtyřjádrovém procesoru

benchmark		výsledný čas [-]			trvání výp. [s]		počet iterací [-]		
název	velikost	par.	sekv.	rozd. [%]	par.	sekv.	par.	sekv.	rozd. [%]
ft01	2×3	172	172	0	0	0	7	7	0
ft04	3×6	286	286	0	0	0	69	69	0
ft06	6×6	531	531	0	3	6	608	608	0
la01	10×5	1575	1575	0	88	220	3568	3438	4
la02	10×5	1483	1483	0	218	509	10377	9972	4
la03	10×5	1649	1649	0	8	17	576	576	0
la04	10×5	1467	1467	0	94	213	4110	4087	1
la05	10×5	1373	1396	2	144	316	7220	7221	0
la06	15×5	2320	2320	0	600	600	9306	3766	147
la07	15×5	2159	2226	3	477	600	7088	3646	94
la08	15×5	2189	2189	0	600	600	7901	2961	167
la09	15×5	2354	2383	1	454	600	8200	3971	106
la10	15×5	2214	2281	3	600	600	7153	3768	90
orb02	10×10	2155	2155	0	600	600	5131	2163	137
orb07	10×10	1416	1461	3	600	600	4402	1901	132
orb05	10×10	2033	2049	1	600	600	1809	623	190
orb10	10×10	2218	2263	2	600	601	1421	554	156
orb01	10×10	2234	2237	0	600	600	3137	1759	78
orb06	10×10	2328	2328	0	600	600	6309	2637	139
orb03	10×10	2046	2074	1	600	600	3194	1050	204
orb04	10×10	2241	2241	0	600	600	4414	1576	180
orb08	10×10	2040	2040	0	68	170	947	947	0
orb09	10×10	2030	2148	5	600	600	3316	1581	110
abz6	10×10	2143	2176	2	600	600	4889	2120	131
ft10	10×10	2141	2141	0	600	600	4133	1742	137
abz5	10×10	2522	2522	0	600	600	3474	1148	203
la16	20×10	2052	2149	5	600	600	4681	1754	167
la17	20×10	1815	1815	0	600	600	5711	2376	140
la18	20×10	1865	1930	3	600	600	3828	1757	118
la19	20×10	2027	2148	6	600	600	3592	1805	99
la20	20×10	2039	2189	7	600	600	4655	1930	141
ft20	20×5	2938	2938	0	600	600	4614	1761	162
la11	20×5	2908	2970	2	600	600	2759	1118	147
la12	20×5	2856	2914	2	600	600	2542	1022	149
la13	20×5	3042	3048	0	600	600	2805	1096	156
la14	20×5	3084	3084	0	600	600	4095	1713	139
la15	20×5	3002	3002	0	600	600	3204	1218	163
la21	15×10	3077	3133	2	600	600	1651	740	123
la22	15×10	2736	2949	7	600	600	1799	731	146
průměr				1,46					109,25

Tabulka 2: Výsledky měření benchmarků na dvoujádrovém procesoru (pokračování na další straně)

benchmark		výsledný čas [-]			trvání výp. [s]		počet iterací [-]		
název	velikost	par.	sekv.	rozd. [%]	par.	sekv.	par.	sekv.	rozd. [%]
ft01	2×3	172	172	0	0	0	7	7	0
ft04	3×6	286	286	0	0	0	69	69	0
ft06	6×6	531	531	0	6	8	608	608	0
la01	10×5	1575	1575	0	209	247	3560	3438	4
la02	10×5	1483	1483	0	473	582	10407	10377	0
la03	10×5	1649	1649	0	16	20	576	576	0
la04	10×5	1467	1467	0	195	240	4088	4087	0
la05	10×5	1396	1396	0	380	364	9204	7221	27
la06	15×5	2320	2320	0	600	600	4083	3195	28
la07	15×5	2159	2271	5	600	600	3841	3173	21
la08	15×5	2189	2189	0	600	600	3254	2555	27
la09	15×5	2383	2383	0	600	600	4474	3122	43
la10	15×5	2267	2335	3	600	600	3976	3273	21
orb02	10×10	2155	2155	0	600	600	2226	1792	24
orb07	10×10	1461	1478	1	600	600	2010	1593	26
orb05	10×10	2049	2049	0	600	600	849	491	73
orb10	10×10	2218	2263	2	600	602	698	486	44
orb01	10×10	2237	2237	0	600	600	1850	1523	21
orb06	10×10	2328	2328	0	600	600	2799	2182	28
orb03	10×10	2074	2149	3	600	600	1216	881	38
orb04	10×10	2241	2455	9	600	600	2368	1369	73
orb08	10×10	2040	2040	0	159	208	947	947	0
orb09	10×10	2148	2148	0	600	600	1608	1276	26
abz6	10×10	2176	2176	0	600	600	2123	1857	14
ft10	10×10	2141	2141	0	600	600	1833	1588	15
abz5	10×10	2522	2522	0	600	600	1325	1025	29
la16	20×10	2149	2149	0	600	600	1897	1457	30
la17	20×10	1815	1815	0	600	600	2460	1910	29
la18	20×10	1930	1937	0	600	600	1882	1555	21
la19	20×10	2148	2148	0	600	600	1949	1441	35
la20	20×10	2189	2189	0	600	600	2011	1496	34
ft20	20×5	2938	2938	0	600	600	1943	1421	37
la11	20×5	2970	3019	2	600	600	1190	816	46
la12	20×5	2914	2914	0	600	600	1130	864	31
la13	20×5	3048	3048	0	600	600	1212	957	27
la14	20×5	3084	3084	0	600	600	1833	1496	23
la15	20×5	3002	3039	1	600	600	1320	1046	26

benchmark		výsledný čas [-]			trvání výp. [s]		počet iterací [-]		
název	velikost	par.	sekv.	rozd. [%]	par.	sekv.	par.	sekv.	rozd. [%]
la21	15×10	3077	3142	2	600	600	846	610	39
la22	15×10	2949	2949	0	600	600	840	623	35
la23	15×10	3214	3214	0	601	601	637	529	20
la24	15×10	3146	3248	3	600	601	707	698	1
la25	15×10	3057	3057	0	600	601	864	716	21
la26	20×10	4521	5041	10	601	600	465	431	8
la27	20×10	4940	4940	0	600	600	786	607	29
la28	20×10	4309	4309	0	601	601	446	331	35
la29	20×10	4131	4132	0	601	601	373	269	39
la30	20×10	4205	4205	0	600	601	418	331	26
la36	15×15	3987	4150	4	600	601	588	480	23
la37	15×15	4157	4157	0	601	604	599	415	44
la38	15×15	3861	3999	3	601	600	490	362	35
la39	15×15	4687	4798	2	601	602	652	539	21
la40	15×15	4470	4478	0	600	601	623	489	27
swv01	20×10	3993	4035	1	601	600	386	248	56
swv02	20×10	3967	4053	2	600	600	302	194	56
swv03	20×10	3954	3989	1	602	600	364	261	39
swv04	20×10	4026	4087	1	600	600	298	218	37
swv05	20×10	4183	4210	1	601	600	361	243	49
la31	30×10	6776	6878	1	601	602	257	191	35
la32	30×10	7562	7763	3	600	600	199	153	30
la33	30×10	7143	7143	0	602	603	301	189	59
la34	30×10	7289	7444	2	601	605	241	162	49
la35	30×10	7645	8052	5	604	605	215	153	41
abz7	15×20	3728	3907	5	600	603	346	280	24
abz8	15×20	4110	4170	1	602	600	350	274	28
abz9	15×20	3940	3940	0	601	601	380	311	22
swv10	20×15	4794	4780	0	601	608	203	167	22
yn1	20×20	6279	6382	2	602	600	254	175	45
yn2	20×20	6466	6603	2	600	601	241	179	35
yn3	20×20	6224	6224	0	601	602	242	182	33
yn4	20×20	6135	6236	2	600	606	229	166	38
swv11	50×10	11061	11774	6	607	604	53	38	39
swv12	50×10	10978	11522	5	605	603	56	40	40
průměr				1,26					29,19

6.4 Shrnutí výsledků

Výsledky ukazují, že v případě využití dvou jader paralelní algoritmus vypočítal průměrně o 1,26 % lepší výsledek a provedl při tom o 29,19 % více iterací než sekvenční algoritmus. V případě použití čtyř jader byl výsledek oproti sekvenční variantě lepší průměrně o 1,46 % a proběhlo o 109,25 % více iterací.

Při srovnání výsledků na vícejádrových procesorech vezmeme v úvahu jen ty benchmarky, které byly vypočítány na obou procesorech. Zlepšení výsledku při výpočtu dvoujádrovým procesorem oproti sekvenčnímu algoritmu pak činilo 0,4 % (na čtyřjádrovém 1,46%) a počet iterací byl oproti sekvenční variantě vyšší o 13,85 % (109,25 % na čtyřjádrovém).

7 Závěr

Cílem této práce byl návrh a implementace rozvrhovacího algoritmu řešícího rozvrh výrobních procesů s blokovacími operacemi. Součástí práce je přehled úloh obsahujících blokovací operace a přehled publikovaných prací, které se jimi zabývají. Z několika variant problému byla vybrána úloha blokovacího job shopu doplněná o praktické rozšíření týkající se dob nastavení strojů mezi jednotlivými operacemi a dob přesunů jobů mezi stroji. Výsledný algoritmus je paralelní algoritmus lokálního prohledávání s tabu seznamem založený na algoritmu prezentovaném v [11]. Oba algoritmy byly implementovány v jazyku C# a doby jejich běhů byly změřeny na dvou počítačích s vícejádrovými procesory (dvoujádrovým a čtyřjádrovým) pomocí sady 72 benchmarků užívaných pro testování algoritmů řešících úlohy job shop. Pro tento účel byla tato široce používaná zadání doplněna o údaje potřebné v rozšířené úloze. Kvůli časové tísni při měření a omezenému přístupu k počítači se čtyřjádrovým procesorem na něm bylo změřeno jen 32 benchmarků. Paralelní algoritmus prohledává možné rozvrhy výrazně rychleji, na čtyřjádrovém procesoru bylo za stejný čas jako pomocí původního algoritmu prohledáno o 109,25 % rozvrhů více, a díky tomu za stejný čas výpočtu nachází lepší výsledky, na čtyřjádrovém procesoru o 1,46%.

Součástí práce je i srovnání softwarových platforem MS Sharepoint a Google Docs z pohledu vhodnosti platformy pro implementaci uživatelského rozhraní umožňující zadávání dat více uživateli. Vlastní rozhraní bylo implementováno pomocí Google Docs a úspěšně odzkoušeno. Zároveň však byly nalezeny limity, které by použití této platformy mohly znemožňovat.

Reference

[1] BRUCKER, PETER; KNUST, SIGRID. *Complex Scheduling.* Berlin : Springer, 2006. 284 s. GOR-Publications . ISBN 978-3-540-29545-7.

[2] BRUCKER, PETER ET AL. Job-shop scheduling with limited capacity buffers. *OR Spectrum.* 2006, vol. 28, no. 2, s. 151-176. Dostupný z WWW: <http://www.springerlink.com/content/755n4124w1j4707l/fulltext.pdf>.

[3] HEITMANN, SILVIA. *Job-shop scheduling with limited buffer capacities.* Osnabrück, 2007. 110 s. Fachbereich Mathematik/Informatik der Universität Osnabrück. Dizertační práce. Dostupný z WWW: <http://deposit.ddb.de/cgi-bin/dokserv?idn=985083344&dok_var=d1&dok_ext=pdf&filename=985083344.pdf>.

[4] MASCIS, ALESSANDRO; PACCIARELLI, DARIO. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research.* 2002, vol. 143, is. 3, s. 498-517. Dostupný z WWW: <http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6VCT-44HXD6H-4-7W&_cdi=5963&_user=1490772&_orig=na&_coverDate=12%2F16%2F2002&_sk=998569996&view=c&wchp=dGLzVtz-zSkzV&md5=6ce6a9b7edcd82fe94ffa50f96ad432f&ie=/sdarticle.pdf>.

[5] MASCIS, A.; PACCIARELLI, D. *Machine scheduling via alternative graphs.* Technical report, Report DIA 46-2000. Dipartimento di Informatica e Automazione, Universitá Roma Tre, 2000. 32 s.

[6] BRUCKER, PETER; KAMPMEYER, THOMAS. Cyclic job shop scheduling problems with blocking. *Annals of Operations Research* [online]. 2008, vol. 159, no. 1 [cit. 2010-01-03], s. 161-181. ISSN 1572-9338. Dostupný z WWW: <http://www.springerlink.com/content/523tq21u4xj35316/fulltext.pdf>.

[7] PACCIARELLI, DARIO; PRANZO, MARCO. A Tabu Search Algortihm for the Railway Scheduling Problem. In PINHO DE

SOUSA, JORGE. *Proceedings of the 4th Metaheuristics International Conference.* Porto (Portugal) : [s. n.], 2001, s. 159-163. Dostupný z WWW: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.5631&rep=rep1&type=pdf>.

[8] NEUMANN, KLAUS; SCHWINDT, CHRISTOPH; ZIMMERMANN, JÜRGEN. *Project Scheduling with Time Windows and Scarce Resources : Temporal and Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions.* 2nd edition. Berlin : Springer-Verlag, 2003. 385 s. ISBN 978-3-540-40125-4.

[9] LABORIE, PHILIPPE. Algorithms for propagating resource constraints in AI planning and scheduling : Existing approaches and new results. *Artificial Intelligence.* 2003, vol. 143, is. 2, s. 151-188. Dostupný z WWW: <http://www.sciencedirect.com/science/article/B6TYF-47HKD3Y-1/2/0214cd9342510478e1e2fc75cf155ad4>.

[10] DE FRENE, E.; SCHATTEMAN, D.; HERROELEN, W.; VAN DE VONDER, S., *A Heuristic Methodology for Solving Spatial Resource-Constrained Project Scheduling Problems.* Leuven, June 2007, 32 s. Katholieke Universiteit Leuven, Faculty of Economics and Applied Economics, Department of Decision Sciences and Information Management. Dostupný z WWW: <http://ssrn.com/abstract=1089355>.

[11] GROFLIN, HEINZ; KLINKERT, ANDREAS. A new neighborhood and tabu search for the Blocking Job Shop. *Discrete Applied Mathematics.* 2009, vol. 157, is. 17, s. 3643-3655. Dostupný z WWW: <http://www.sciencedirect.com/science/article/B6TYW-4VYW658-1/2/fd535025ef3bef8bcb8c19890e8e3d24>.

[12] TALBI, EL-GHAZALI. *Parallel combinatorial optimization.* Hoboken (New Jersey) : John Wiley and Sons, 2006. 330 s. ISBN 9780471721017.

[13] Microsoft Corporation. *MSDN Library : System.Threading Namespace* [online]. c2009 [cit. 2009-12-20]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/system.threading.aspx>.

[14] Google. *Smlouva o užívání služby Google Apps Premier* [online]. c2009 [cit. 2009-12-17]. Dostupný z WWW: <http://www.google.com/apps/intl/cs/terms/premier_terms.html>.

[15] Google. *Google Spreadsheets APIs and Tools : Client Libraries and Sample Code* [online]. c2009 [cit. 2009-12-18]. Dostupný z WWW: <http://code.google.com/intl/cs/apis/spreadsheets/code.html>.

[16] Microsoft Corporation. *Microsoft Office Online : Začínáme se službou SharePoint Services systému Windows* [online]. c2009 [cit. 2009-12-21]. Dostupný z WWW: <http://office.microsoft.com/cs-cz/sharepointtechnology/HA100242771029.aspx>.

[17] ELBL, Marek. *Simulace a rozvrhování výrobních procesů.* Diplomová práce. Praha, 2010. ČVUT v Praze, Fakulta elektrotechnická, Katedra řídicí techniky. Vedoucí diplomové práce Ing. Přemysl Šůcha Ph.D.

[18] Google. *Google code : gadgets.* API Developer\'s Guide* [online]. c2009 [cit. 2009-10-08]. Dostupný z WWW: <http://code.google.com/intl/cs/apis/gadgets/docs/dev_guide.html>.

[19] Google. *Google Spreadsheets APIs and Tools : Developer\'s Guide: Protocol* [online]. c2009 [cit. 2009-08-16]. Dostupný z WWW: <http://code.google.com/intl/cs/apis/spreadsheets/data/3.0/developers_guide_protocol.html>.

[20] ADAMS, Joseph; BALAS, Egon; ZAWACK, Daniel. The shifting bottleneck procedure for job shop scheduling. *Management Science.* 1988, vol. 34, is. 3, s. 391-401.

[21] FISHER, H.; THOMPSON, G. L. Probabilistic learning combinations of local job-shop scheduling rules. In MUTH, J. F.; THOMPSON, G. L. *Industrial Scheduling.* Englewood Cliffs (New Jersey) : Prentice Hall, 1963. s. 225-251.

[22] LAWRENCE, S. *Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques.* Gra-

duate School of Industrial Administration, Carnegie-mellon University, Pittsburgh (Pennsylvania). 1984

[23] APPLEGATE, David; COOK, William. A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing.* 1991, vol. 3, no. 2, s. 149-156.

[24] STORER, Robert H.; WU, S.David; VACCARI, Renzo. New search spaces for sequencing problems with application to job shop scheduling. *Management Science.* 1992, vol. 38, is. 10, s. 1495-1509.

[25] YAMADA, Takeshi; NAKANO, Ryohei. A genetic algorithm applicable to large-scale job-shop problems. In MANNER, Reinhard, MANDERICK, Bernard. *Parallel instance solving from nature, 2 : Proceedings of the Second Conference on Parallel Problem Solving from Nature.* [s. l.] : Elsevier Science Inc., 1992, s. 281-290. ISBN 0444897305. Dostupný z WWW: <http://www.kecl.ntt.co.jp/as/members/yamada/PPSN92.PDF>.

[26] DVOŘÁK, Jiří. *Úlohy teorie grafů* [online]. 2009 [cit. 2009-10-18]. Dostupný z WWW: <www.uai.fme.vutbr.cz/~jdvorak/vyuka/tsoa/Pred09.ppt>.

[27] Viewpath. Google *Project Management with Visualization Gadget* [online]. c2008 [cit. 2009-12-26]. Dostupný z WWW: <http://www.viewpath.net/Website/ganttcharts/googlegadget.aspx>.