VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY
INSTITUTE OF INFORMATICS

VYUŽITÍ UMĚLÉ INTELIGENCE PŘI ŘEŠENÍ ROZVRHOVACÍHO PROBLÉMU

THE USE OF ARTIFICIAL INTELLIGENCE FOR SOLVING A SCHEDULING PROBLEM.

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jan Rybníček**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. Petr Dostál, CSc.**
SUPERVISOR

BRNO 2016

Vysoké učení technické v Brně
Fakulta podnikatelská

Akademický rok: 2015/2016
Ústav informatiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Rybníček Jan, Bc.

Informační management (6209T015)

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách, Studijním a zkušebním řádem VUT v Brně a Směrnicí děkana pro realizaci bakalářských a magisterských studijních programů zadává diplomovou práci s názvem:

Využití umělé inteligence při řešení rozvrhovacího problému

v anglickém jazyce:

The Use of Artificial Intelligence for Solving a Scheduling problem.

Pokyny pro vypracování:

Úvod
Cíle práce, metody a postupy zpracování
Teoretická východiska práce
Analýza současného stavu
Vlastní návrhy řešení
Závěr
Seznam použité literatury
Přílohy

Seznam odborné literatury:

DOSTÁL, P. Pokročilé metody rozhodování v podnikatelství a veřejné správě. Brno: CERM, 2012. 718 s. ISBN 978-80-7204-798-7.
DOSTÁL, P. Advanced Decision Making in Business and Public Services. Brno: CERM, 2011. 168 s. ISBN 978-80-7204-747-5.
HANSELMAN, D. a B. LITTLEFIELD. Mastering MATLAB. Pearson Education International Ltd., 2012. 852 s. ISBN 978-0-13-185714-2.
MAŘÍK, V., O. ŠTĚPÁNKOVÁ a J. LAŽANSKÝ. Umělá inteligence. Praha: ACADEMIA, 2013. 2473 s. ISBN 978-80-200-2276-9.

Vedoucí diplomové práce: prof. Ing. Petr Dostál, CSc.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2015/2016.

L.S.

_____	_____
doc. RNDr. Bedřich Půža, CSc.	doc. Ing. et Ing. Stanislav Škapa, Ph.D.
Ředitel ústavu	Děkan fakulty

V Brně, dne 29.2.2016

Abstrakt

Tato diplomová práce se zabývá rozvrhovacími problémy a algoritmy využitelnými k jejich řešení. Rozvrhovací algoritmy se snaží o optimální přidělení zdrojů v čase podle omezujících podmínek. Rozvrhovací problémy se často liší svojí podstatou a zadáním omezujících podmínek. V praktické části práce je vyřešen jeden konkrétní typ plánovacího problému, který je omezenou verzí obecného plánovacího problému.

Abstract

This thesis deals with scheduling problems and algorithms usable to solve them. Scheduling algorithms seek an optimal allocation of resources over time while using constraints. Scheduling problems are often different in nature and type of constraint conditions. In the practical part is solved one particular type of scheduling problem, which is a constrained version of the common scheduling problem.

Klíčová slova:

Rozvrhovací problém, umělá inteligence, průmysl cestovního ruchu

Key words:

Scheduling problem, artificial intelligence, tourism industry

Bibliografická citace

Rybníček, J. *Využití umělé inteligence při řešení rozvrhovacího problému*. Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2016. 89 s. Vedoucí diplomové práce prof. Ing. Petr Dostál, CSc.

Prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 23. Května 2016

……

Bc. Jan Rybníček

Poděkování

Děkuji vedoucímu diplomové práce prof. Ing. Petru Dostálovi, CSc za odbornou pomoc a další cenné rady při vypracovávání diplomové práce.

Obsah

1 Úvod

Tato práce se zabývá teoretickým zmapováním rozvrhovacího problému a metod použitelných k jeho řešení. Jsou zkoumány standardní algoritmy, heuristické funkce i metody využívající umělé inteligence. V praktické části je podrobněji rozebrán jeden z rozvrhovacích problémů a jsou zkoumány některé algoritmy použitelné na jeho řešení.

Rozvrhování je obecný pojem, který zasahuje do mnoha oblastí a který vyžaduje různé přístup ke svému řešení. Můžeme rozlišit jak rozvrhování do budoucnosti, kdy známe hodnoty všech vstupů, tak i průběžné rozvrhování, kdy vstupy nejsou dopředu známy a kdy musí být zaručeno, že nově příchozí úlohy budou mít dostatek zdrojů pro svůj průběh.

Umělá inteligence je vlastnost algoritmů, které pro nalezení nejlepšího řešení nepotřebují projít celý stavový prostor úlohy. Často se při řešení problémů mohou učit ze svých minulých průběhů a podle vlastní úvahy najít nejlepší možné řešení zadaného problému.

2 Vymezení problému a cíle práce

Hlavním cílem této diplomové práce je vytvoření optimalizačního systému pro řešení rozvrhovacího problému při určování obsazení hotelových pokojů. Budou využity prvky umělé inteligence. Výstupem by měla být aplikace schopná optimalizovat rozvržení rezervací pokojů tak, aby vyhovovalo zadaným požadavkům. Nejdůležitějším z požadavků bývá eliminace příliš krátkých neobsazených míst mezi jednotlivými rezervacemi.

Dílčími cíli jsou:

- Analýza současného stavu ve společnosti
- Analýza existujících algoritmů a porovnání jejich výkonnosti
- Návrh vylepšení algoritmů
- Zhodnocení přínosů pro společnost a pro zákazníky

2.1 Metody a postup zpracování

Pro implementaci optimalizačních algoritmů bude využit programovací jazyk C# a program bude vytvořen ve vývojovém prostředí Visual Studio. Pro účely této diplomové práce bude vytvořeno i jednoduché grafické rozhraní pro demonstraci funkčnosti, v praxi se výsledný program bude moci zavést jako knihovna do již existujícího systému.
V současnosti neexistuj ve společnosti žádný automatický systém pro optimalizaci, a ta je tak prováděna ručně. Vytvořená aplikace by měla být schopna v krátkém čase optimalizovat rozvržení rezervací pokojů.

Diplomová práce je rozdělena do několika dílů. První část obsahuje teoretická východiska, vysvětluje základní pojmy rozvrhování a umělé inteligence. V další části jsou popsány algoritmy umělé inteligence použitelné pro rozvrhovací problém. Poslední část se zabývá vlastním řešením, kde je zkoumáno několik algoritmů vhodných pro řešení rozvrhovacího problému.

3 Rozvrhování

Rozvrhovací problém je problémem optimalizačním. Rozvrhování lze definovat jako činnost, která určuje kdy začít s kterými úlohami. Musí se brát v úvahu různá omezení vyplývající z dostupnosti výrobních, lidských i časových zdrojů. Optimalizace pak hledá nejvhodnější kombinaci zdrojů a úloh, která dosáhne nejlepšího výsledku.

Rozvrhovací problém je definován proměnnými, omezeními a optimalizační funkcí. Proměnné mají množinu hodnot, kterých mohou nabývat. Omezení dále určují, kterých hodnot může proměnná nabývat. Při tvorbě rozvrhů vzniká vždy mnoho realizovatelných rozvrhů, které však nejsou optimální. Optimalizační funkce pak rozhoduje, která řešení jsou lepší než ostatní.

Pro potřeby rozvrhování je použit čas s diskrétními hodnotami.

3.1 Popis

Základní popis situace vypadá následovně: máme *m* strojů, které musí zpracovat *n* úloh. Plán pak představuje přiřazení časových zdrojů na každém stroji jednotlivým úlohám. Grafickou reprezentaci je možné provést pomocí Ganttových diagramů, které mohou být zaměřeny buď na stroje, nebo úlohy. Obě tyto varianty jsou rovnocenné a vzájemně převoditelné.

Každá úloha se skládá z několika operací. Každá operace má určité požadavky a parametry jako například vyžadované stroje a dobu běhu. Operace mají na výběr z množiny strojů. V množině strojů se mohou nacházet stroje jednoúčelové, vzájemně zastupitelné stroje a víceúčelové stroje, které zvládají vykonávat více druhů operací. Dále je zde nákladová funkce, která určuje cenu dokončení úlohy za nějaký čas.

Rozvrh je realizovatelný pokud se žádné dva intervaly na jednom stroji nepřekrývají, pokud se nepřekrývají dva intervaly alokované jedné úloze, a pokud jsou splněny další podmínky vyplývající ze specifikace úlohy.

Plán je optimální, pokud je minimalizována optimalizační funkce.

3.2 Pojmy

Pro každou úlohu je definováno několik základních vlastností [1], [2], [3]:

- Doba zpracování (p) – doba, kterou úloha vyžaduje pro zpracování na jednom stroji.
- Dostupnost (r) – označuje okamžik, kdy úloha vstupuje do systému. Je to okamžik, kdy může začít práce na dané úloze.
- Dokončení (d) – v tomto čase by měla být úloha již zpracována, ale není to podmínkou. Narůstají však náklady.
- Váha (w) – důležitost s jakou je k úloze přistupováno

Rozvrhovací problémy mohou být zapisovány podle Grahamovy notace jako **α|β|γ**, kde alfa značí použité stroje, beta značí typ zpracovávaných úloh a gama obsahuje funkci, která má být optimalizována.

Typy strojů (α) mohou být následující:

- Jeden stroj (1) – V celém systému je pouze jeden stroj. Tento případ lze považovat za speciální variantu všech následujících.
- Paralelní identické stroje (P) – Existuje m identických strojů fungujících paralelně. Není žádné pravidlo, podle kterého by se měly úlohy řadit k jednotlivým strojům. S každou úlohou je provedena pouze jedna operace.
- Job shop (J) – Existuje více strojů. Každá úloha navštíví jeden nebo více strojů, některé stroje může vynechat, některé může navštívit vícekrát. Cesta je předem známá.
- Flow shop (F) – Sériově zapojené stroje. Úlohy postupně postupují přes všechny stroje v přesně daném pořadí.
- Open shop (O) – Každá úloha je zpracována přesně jednou na každém stroji. Jejich pořadí není důležité.

Charakteristiky úlohy a omezení rozvrhování (β):

- Preempce (pmtn) – Pokud jsou úlohy preemptivní, může jejich vykonávání být dočasně přerušeno ve prospěch jiné úlohy. Pokud jsou nepreemptivní, zůstává jim přidělen stroj až do jejich dokončení.

- Čekání (W) – Týká se pouze úloh typu flow shop. Úlohám může být nastaveno, že mezi dvěma po sobě jdoucími stroji nesmí být žádná přestávka.
- Precedence (prec) – Udává pořadí, v jakém jednotlivé úlohy mají být vykonávány. Nejčastěji jsou tyto omezení zadána ve formě acyklického grafu, kde každý uzel představuje jednu úlohu. Pokud má každá úloha právě jednoho předchůdce a právě jednoho následníka, jedná se o zřetězené zpracování. Pokud každá úloha má maximálně jednoho následníka, říkáme, že cesty v grafu směřují ke kořeni (intree). Pokud má každá úloha maximálně jednoho předchůdce, všechny cesty směřují od kořene (outtree).
- Dostupnost (r) – čas kdy nejdříve lze začít na úloze pracovat.
- Počet úloh – Počet úloh současně zpracovávaných v systému může být omezen
- Další omezení: Počet operací v úloze, doba zpracování, deadline…

Optimalizace (γ):

- Doba zpracování (C_{max}) – Určuje čas kdy je dokončena poslední úloha na posledním stroji. Definován bývá jako $C_{max}=max(C_1, ..., C_n)$. Minimalizace této funkce maximalizuje propustnost systému a zajišťuje rovnoměrné vytížení strojů.
- Zpoždění (L_{max}) – $L_j=C_j - d_j$ kde C je doba dokončení úlohy a d je doba očekávaného dokončení. $L_{max}=(L_1, ..., L_n)$.
- Opožděnost (nezáporné zpoždění, tardiness) (T_{max}) – Vypočítá se jako $T_j=max(C_j - d_j,0)$. Je snaha o minimalizaci celkového zpoždění.
- Případně lze nastavit speciální funkci podle potřeb úlohy (f (x))

3.3 Složitost

Teorie složitosti je velice důležitá při řešení různých typů rozvrhovacího problému. Pro každý rozvrhovacím problémem se snažíme vytvořit vhodný algoritmus. Bohužel se často stává, že neexistuje jiná možnost, než projít celý stavový prostor úlohy. Proto je velmi důležité mít znalosti NP třídách problémů, a být schopni určit, zda existuje efektivní způsob řešení [1].

3.4 Časová složitost algoritmů

Základem zkoumání složitosti algoritmů je měření doby běhu algoritmu jako funkce velikosti vstupu. Každý algoritmu se skládá z postoupnosti kroků. V závislosti na počtu opakování algoritmu pak můžeme tvrdit, že třeba krok 1 má lineární závislost, krok 2 je kvadraticky závislý a krok 3 třeba exponenciálně závislý. O celém algoritmu, pak z důvodu zjednodušení, říkáme, že má složitost jako nejsložitější z jeho kroků. Zanedbávají se multiplikační konstanty a jednodušší části algoritmu. Také není důležitá konstanta před funkcí, ale řád, se kterým funkce roste. Máme-li tedy například algoritmus se složitostí definovanou jako $n+3n^2+3*3^n$, mluvíme o exponenciální složitosti [3].

Rozlišujeme následující základní druhy složitosti [4]:

O (1) – Konstantní: Násobení dvou čísel, návrat z funkce, porovnání hodnot

O (log N) – Logaritmická: Nezávisí na základu logaritmu, jde třeba o binární vyhledávání (metoda půlení intervalu)

O (N) – Lineární: „for-cyklus“ a v něm třeba operace s konstantní složitostí

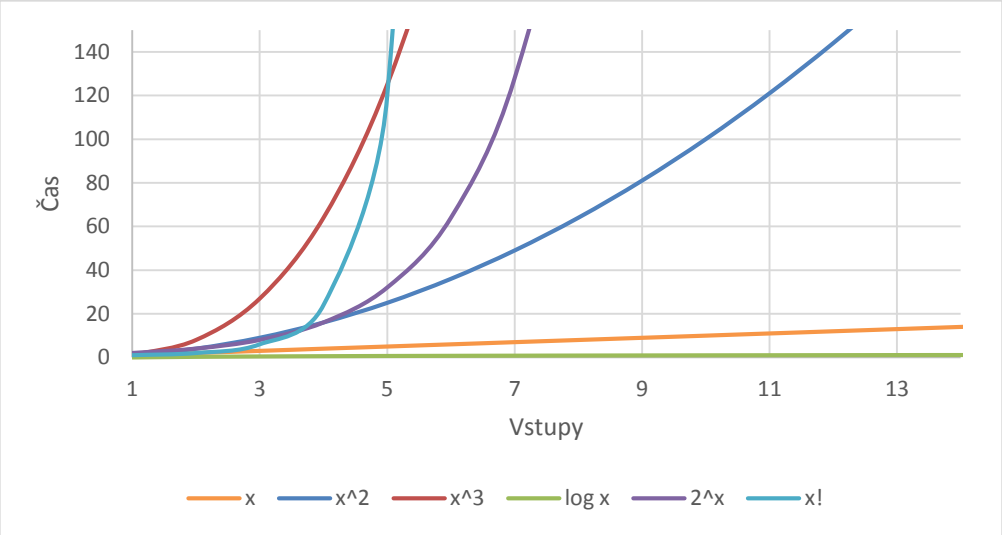
O (N*log N) – Lineárně-logaritmická: Metody řazení HeapSort a MergeSort

O (N^2) – Kvadratická - dva zanořené „for-cykly“, BubbleSort

O (N^3) – Kubická: Násobení matic

O (2^N) – Exponenciální

O (N!) – Faktoriálová: Neoptimalizovaný problém obchodního cestujícího



Obrázek 1: Porovnání různých druhů složitosti

	n=10	n=100	n=1000	n=1000000
log n	3.3ns	6.7ns	10ns	20ns
n	10ns	100ns	1μs	1ms
n*log n	33ns	664ns	9.9μs	20ms
n^2	100ns	10μs	1ms	16.5min
n^3	1μs	1ms	1s	31let
2^n	1μs	3 * 10^14 let	3 * 10^286 let	≈ ∞
n!	3ms	3 * 10^142 let	≈ ∞	≈ ∞

Tabulka 1: Porovnání doby výpočtu [4]

Z tabulky je vidět, že pokud uvažujeme o počítači, který zvládne 10^9 operací za sekundu, tak už problémy s kubickou složitostí trvají nepřiměřeně dlouho a pro větší vstupy jsou algoritmy s exponenciální a faktoriálovou složitostí nepoužitelné. Snažíme se tedy při návrhu algoritmů co nejvíce o používání logaritmické a lineární složitosti.

O polynomiální složitosti hovoříme u algoritmů, které mají složitost maximálně jako $O(N^k)$, kde k je konstanta. Obecně se uvažuje, že algoritmus, který lze vyjádřit s polynomiální složitostí je řešitelný v konečném čase. Nepolynomiální algoritmy jsou již pro nízké hodnoty vstupů neřešitelné.

Kromě časové složitosti je potřeba uvažovat i složitost paměťovou, která určuje, kolik paměti algoritmus pro svůj běh spotřeboval. Uvažuje se s elementárními paměťovými jednotkami o velikosti byte, integer atd. Pro účely této práce se však tímto nebudu zabývat.

Třídy P a NP

Pro definování P a NP problému existuje několik způsobů. Jeden z nejzajímavějších je pomocí nedeterminstického algoritmu [3] [5].

Nedeterministický algoritmus je takový algoritmus, který pro jednu množinu vstupů dává při několika průbězích rozdílné výsledky. V praxi mají jen omezené využití. Deterministické algoritmy jsou všechny ostatní, kdy na jednu kombinaci vstupů připadá za každých podmínek ten stejný výstup.

Nedeterminismus bývá implementován několika způsoby a na několika úrovních, například pomocí generování náhodných čísel, nebo při více vláknovém běhu aplikace. Nedeterministické algoritmy se používají při problémech, které mají více správných řešení.

Třída **P** zahrnuje rozhodovací úlohy, pro které existuje polynomiální algoritmus
Třída **NP** je skupinou úloh, pro které existuje nedeterministický algoritmus pracující v polynomiálním čase. Pro tyto úlohy platí, že v polynomiálním čase nejsme schopni nalézt řešení, ale pokud už řešení máme, můžeme ověřit, jestli je platné. Toho se využívá například v kryptografii. Můžeme rychle zjistit, jestli nějaké heslo patří k zašifrovaným datům, ale nalezení samotného hesla z dat je velmi obtížné.
Pokud bychom měli stroj schopný řešit všechna možná řešení v jednom okamžiku, tak by se k řešení přišlo v polynomiálním čase.
Platí, že P je podmnožinou NP. Důkazem je, že deterministický algoritmus je vlastně algoritmem nedeterministickým, který v sobě neobsahuje část, která by se dala považovat za nedeterministickou [3].

Polynomiální redukce

Mnoho reálných problémů lze popsat jako optimalizační úlohy, kdy se hledá nejlepší možné řešení z množiny existujících řešení. Rozlišujeme mezi rozhodovacím problémem a optimalizačním problémem. Každý optimalizační problém lze převést na rozhodovací problém zavedením dodatečného parametru ω, a následným dotazováním, jestli existuje řešení, které je menší (větší) než ω. Výstupem řešení rozhodovacího problému jsou odpovědi „ano“ a „ne“ [3].
Teorie okolo NP třídy platí jen pro rozhodovací problémy. Složitost optimalizačního problému je závislá na složitosti jeho odpovídajícího rozhodovacího problému. To znamená, že optimalizační problém je řešitelný v polynomiálním čase, právě když je jeho odpovídající rozhodovací problém řešitelný v polynomiálním čase.
Na příkladu obchodního cestujícího to lze popsat následovně. Obchodní cestující má za úkol projít N měst nejkratší možnou cestou. Optimalizační úloha by byla nalezení nejkratší cesty. Rozhodovací úlohou pak řeší, zda je možné nalézt trasu takovou, že je kratší než nějaká zvolená hodnota. Lze dokázat, že když je rozhodovací úloha řešitelná

polynomiálně, je polynomiálně řešitelná i optimalizační úloha. Pro problém obchodního cestujícího ještě nebyl nalezen polynomiální algoritmus. Uvažujme, že máme u úlohy instanci I, o Ano-instanci uvažujeme, pokud I dává odpověď ano, jinak mluvíme o Ne-instanci. Nechť P a Q jsou dva rozhodovací problémy. P je polynomiálně redukovatelné na Q, zapsáno jako $P \propto Q$, pokud existuje funkce f, která všechny instance úlohy P na instance úlohy Q, a to tak, že všechny ano-instance P jsou namapovány na ano-instance Q. To stejné s ne-instancemi. Pokud je funkce f polynomiální, mluvíme o *polynomiální redukci*. Pokud tvrdíme, že $P \propto Q$, neznamená to, že i $Q \propto P$. Z toho vychází zjištění, že pokud máme problémy $P \propto Q$, a Q je řešitelné v polynomiálním čase, tak i P je řešitelné v polynomiálním čase. A také když P není řešitelné v polynomiálním čase, tak i Q nemůže být řešeno v polynomiálním čase.

NP-těžké úlohy

Problém U je NP-těžký, pokud všechny problémy ve třídě NP jsou polynomiálně redukovatelné na U.

NP-úplné úlohy

Rozhodovací problém U je NP-úplný, pokud U náleží do NP třídy, a zároveň platí, že všechny problémy ve třídě NP jsou polynomiálně redukovatelné na U.

Za předpokladu, že P i Q jsou NP-úplné, pak platí $P \propto Q$ a $Q \propto P$.

Když nalezneme algoritmus na řešení problému z NP-úplné třídy, tak jej můžeme použít na každý problém z NP třídy. Jestliže je jeden z problémů NP-úplné třídy řešitelný v polynomiálním čase, pak všechny úlohy z NP-úplné třídy jsou.

NP-těžké úlohy

Problém U je NP-těžký, pokud všechny problémy ve třídě NP jsou redukovatelné na U.

Další třídy

Existují ještě další třídy s ještě větší složitostí, například problémy, jejich řešení nelze najít v polynomiálním čase a v polynomiálním čase nelze ani ověřit jejich správnost. Příkladem je hraní šachů. Tyto třídy však nejsou pro tuto práci důležité, a tak se jimi nebudu dále zabývat.

3.5 Druhy rozvrhovacích úloh

3.5.1 Rozvrhování na jednom stroji

Plánování úloh pro jeden stroj je nejjednodušší skupina úloh v teorii rozvrhování. Při plánování úloh pro jeden stroj určujeme, v jakém pořadí jsou stroji přidělovány úlohy tak, aby byly splněny různé podmínky. Každá úloha má termín dokončení a termín dostupnosti. Cílem optimalizace je minimalizování zpoždění. Obecně se jedná o NP-těžký problém, ale s určitými omezeními a povolenou preempcí se lze redukovat na polynomiální problém.

Typický zástupce tohoto problému zapsaný podle Grahamovy konvence vypadá následovně: **1| |L_{max}** , neboli jeden stroj obsluhující úlohy bez omezení a optimalizováno je celkové zpoždění systému [3] [6].

EDD earliest due date

Seřazení plánovaných úloh do fronty podle termínu dokončení do neklesající posloupnosti. Pokud ještě není úloha v daném termínu dostupná, je naplánována následující. Z fronty jsou pak vybírány úlohy (takže vlastně podle nejdřívější doby dokončení). Tento algoritmus je optimální.

FCFS First come fist served

Algoritmus neprovádí žádné plánování, ale úloze, která je na začátku fronty, je přiřazen stroj, na kterém zůstává, dokud není dokončena. Rozšíření této metody spočívá v zavedení preempce, kdy je úloha na stroji pouze omezenou dobu a před dokončením je pozastavena a znovu zařazena do fronty. Metoda není optimální. Ve verzi rozšířené o preempci byla používána pro plánování úloh na procesoru v prvních počítačích.

SPT Shortest processing time

Obdobou EDD je SPT. Úlohy jsou seřazeny od nejkratší doby vykonávání na stroji. Nevýhodou je, že je nutné znát dobu zpracování. Pokud nejsou na začátku výpočtu známy všechny úlohy (procesy v PC), hrozí dlouhým úlohám, že budou stále předbíhány krátkými.

WSPT

Rozšíření předchozí metody o váhu každé úlohy. Je optimální pro $1||\sum w_j C_j$. Úlohy jsou plánovány podle $w_j/p_j < w_{j+1}/p_{j+1}$.

Round Robin

Používaná pro plánování procesů na procesoru. Rozšíření metody FCFS. Úlohy čekají ve frontě a každé z nich je přiděleno časové kvantum, které může úloha strávit na procesoru. Po uběhnutí časového kvanta je úloha zařazena zpět do fronty.

Metoda větví a mezí (Branch and Bound)

Tato metoda pracuje na principu dělení problému na podproblémy. Není již triviální jako předchozí metody a lze ji použít na celou škálu rozvrhovacích problémů, nejenom pro problémy s jedním strojem. Snaží se minimalizaci nebo maximalizaci účelové funkce. Podstata spočívá v tom, že množina přípustných řešení se dělí na podmnožiny, kterým je spočítáno ohodnocení účelovou funkcí. Toto ohodnocení prohlásíme za horní (dolní) mez. Pokud je u nějakého přípustného řešení účelová funkce vyšší (nižší), nepokračuje se dále ve výpočtu v této větvi. Algoritmus lze zapsat i pomocí rekurze [6].

```
LIST := počáteční uzel;
UB := horní mez z heuristické funkce, nebo nekonečno;
BEST := heuristické řešení problému;
WHILE LIST není prázdný DO
      BEGIN
      Vyber uzel U z LIST;
      Vygeneruj následníky uzlu U a jejich dolní mez LB
      FOR i := 1 TO nU DO
            IF LBi < UB THEN
                  IF potomek[i] má jedno řešení THEN
                        U := LBi;
                        BEST := řešení potomka[i]
                  ELSE zařadit potomka[i] do LIST
      END
```

Algoritmus 1: Metoda větví a mezí

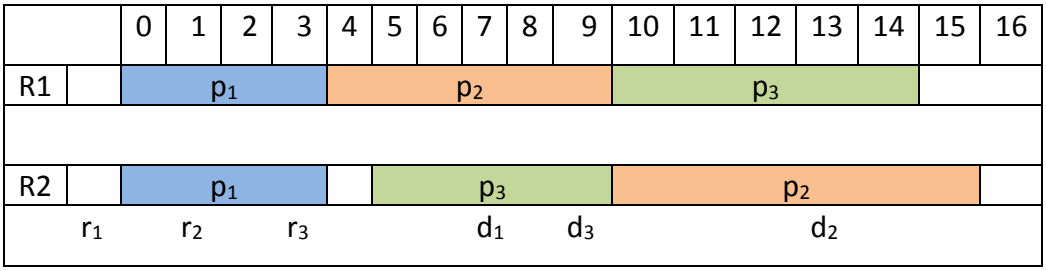
Mezi další zajímavé problémy patří **1| r_j |L_{max} ,** který je již NP úplný. Na rozdíl od předchozího problému je zde navíc omezení v rozdílných termínech dostupnosti. Může se tak stát, že optimální rozvrh má v sobě období nečinnosti. Znázorněno je to na následujícím příkladu.

U	**1**	**2**	**3**
p_j	4	6	5
r_j	0	3	5
d_j	8	14	10
C_j	**4**	**10**	**15**
L_j	**-4**	**-4**	**5**

U	**1**	**3**	**2**
p_j	4	5	6
r_j	0	5	3
d_j	8	10	14
C_j	**4**	**10**	**16**
L_j	**-4**	**0**	**2**

Tabulka 2: Příklad optimalizovaného rozvrhu

První rozvrh je plánován metodou EDD. V tabulce je jako C_j označen čas dokončení a jako $L_j = C_j - d_j$ zpoždění za požadovaným termínem dokončení. Metoda EDD vytvořila rozvrh bez volných míst, ale maximální zpoždění je 5 časových jednotek.
V druhém rozvrhu vytvořeném podle pokročilejších metod vznikla nečinnost po dobu jedné časové jednotky, ale maximální doba zpoždění klesla na 2 časové jednotky. Je to dáno tím, že termín dokončení úlohy 3 je před termínem úlohy 2, je tedy naplánován dříve, ale zároveň úloha 3 nemůže začít v čase 4, protože je dostupná až v čase 5. Na obrázku je vše graficky znázorněno [6].



Tabulka 3: Porovnání rozvrhů

Problémů lze definovat nepřeberné množství, proto zde uvedu již jen jeden zajímavý. **1|pmtn, r_j|L_{max}**. Jde o rozšíření předchozího problému o preempci. Pro jeho řešení lze použít preemptivní EDD algoritmus. V každé časové jednotce je naplánována úloha s nejbližším termínem dokončení (due date). Tento algoritmus je typu online, to znamená, že v době plánování neznáme všechny úlohy, které se v systému vyskytnou [7].

3.5.2 Rozvrhování paralelních strojů

V problému s paralelními stroji řešíme průchod úloh přes alternativní stroje. Všechny stroje vykonávají ten samý druh činnosti, ale liší se dobou činnosti. Pokud je povolen preempce, může být úloha zpracovávána postupně na několika různých strojích [1].

- Identické stroje: doba zpracování je stejná na všech strojích
- Uniformní stroje: doba zpracování je násobkem nějaké základní hodnoty
- Libovolné stroje: dobu zpracování má každý stroj různou

Problém $\mathbf{P_m||C_{max}}$, spočívá v rozvrhování *m* paralelních strojů, když účelová funkce je minimalizace celkové doby běhu všech úloh. Existuje řada heuristických metod pro řešení tohoto problému, které vytváří téměř optimální rozvrh. Tento problém je NP-těžký [3].

LPT (longest processing time first)
Úlohy jsou seřazeny do klesající posloupnosti podle doby běhu. Úlohy, které mají delší dobu běhu, se dostanou na řadu nejdříve. Bylo dokázáno, že existuje závislost mezi poměrem řešení získaného metodou LPT a optimálním řešením.

$$\frac{C_{\max(LPT)}}{C_{\max(OPT)}} \leq \frac{4}{3} - \frac{1}{3 * m}$$

Pro příklad uvažujme následující problém. Máme devět úloh a čtyři dostupné stroje.

U	1	2	3	4	5	6	7	8	9
p_j	5	5	4	4	3	3	2	2	2

Tabulka 4: Zadané úlohy

	1	2	3	4	5	6	7	8	9
M1	p_1					p_7		p_9	
M2	p_2					p_8			
M3	p_3				p_5				
M4	p_4				p_6				

	1	2	3	4	5	6	7	8	9
M1	p_1					p_5			
M2	p_2					p_6			
M3	p_3				p_4				
M4	p_7		p_8		p_9				

Tabulka 5: LPT rozvrh a optimální rozvrh

V prvním rozvrhu vytvořeném metodou LPT je poslední úloha dokončena v čase 9. V optimálním rozvrhu je poslední dokončena již v čase 8. Pokud dosadíme do dříve uvedeného vzorce, zjistíme, že poměr vypočítaného a optimálního by měl být ≤ 1,25. Ve skutečnosti jsme na tom ještě lépe na 1,125.

Problémy typu $\mathbf{P_m|pmtn|C_{max}}$ minimalizují dobu dokončení preemptivních úloh na m strojích. Vyřešit tento problém lze se složitostí O (n) [1].

Wrap around pravidlo

Tato metoda využívá ve výpočtu dolní mez jako odhad optimálního řešení.

$$LB = \max(max_i^n(p_i, \frac{\sum_{i=1}^{n} p_i}{m}))$$

Úlohy nejsou nijak seřazeny a postupně se přidělují prvnímu stroji, dokud nedojde k překročení LB, poté je úloha rozdělena a pokračuje na dalším stroji.

Například pro následující zadání dostaneme LB = 7.

U	1	2	3	4	5	6
p_j	2	5	4	3	1	6

Tabulka 6: Zadání rozvrhu

Znázornění rozvrhu vypadá následovně:

	1	2	3	4	5	6	7	8
M1	P_1		P_2					
M2	P_2		P_3			P_4		
M3	P_4	P_5	P_6					

Tabulka 7: Rozvrh vytvořený pomocí Wrap around pravidla

V případě že p_i>LB, tedy že doba zpracování jedné z úloh je větší, než LB, musíme LB zvýšit na její hodnotu. Úlohu nemůžeme rozdělit, protože ji nelze zpracovávat na dvou strojích současně. Dolní mez se tedy nastaví na hodnotu této speciální úlohy [2].

LRPT (longest remaining processing time first)

Tento algoritmus, stejně jako předchozí tvoří optimální rozvrh pro problémy typu $\mathbf{P_m|pmtn|C_{max}}$. Je vytvořen seznam úloh a v každém časovém kvantu jsou vybrány pro stroje úlohy s nejdelším zbývajícím časem.

	1	2	3	4	5	6	7	8
M1	p2						p_4	
M2	p_6			p_1	p_4	p_1	p_5	
M3	p_3		p_4	p_3	p_6	p_3	p_6	

Tabulka 8: Rozvrh vytvořený metodou LRPT

Pokud problém rozšíříme na $\mathbf{Q_m|pmtn|C_{max}}$, tedy zpracování na strojích pracujících různou rychlostí, musíme využít modifikaci LRPT metody. Pokaždé, když je na nejrychlejším stroji dokončena úloha, jsou na něj přesunuty úlohy z druhého nejrychlejšího stroje [6].

3.5.3 Shop scheduling

Rozvrhovací problémy typu „shop" jsou rozsáhlá skupina příbuzných problémů, které představují speciální případy obecného „shop" problému. Jsou zde problémy jako Job shop, Open shop, Flow shop, Cycle shop a další. Slovo „shop" je zde ve významu dílna, továrna nebo výrobna. Snaží se tím naznačit, že tyto úlohy původně vznikly pro potřeby průmyslové výroby [1].

Obecný „shop“ problém je definován následovně: Máme *n* úloh (J_1, …, J_n), a *m* strojů (M_1, …, M_m). Každá úloha se skládá z množiny operací O_{ij} (j = 1, …, n_i), které mají dobu zpracování p_{ij}. Každá operace O_{ij} musí být zpracována na stroji $\mu_{ij} \in \{M_1, \dots, M_m\}$. Mohou existovat prioritní omezení mezi operacemi každé úlohy. Každá úloha může být zpracovávána v jednom okamžiku pouze jedním strojem a každý stroj může v jednom okamžiku zpracovávat pouze jednu úlohu. Smyslem snažení je nalézt vhodný rozvrh, který optimalizuje účelovou funkci [3].

Reprezentace pomocí disjunktních grafů

Disjunktní grafy mohou být využity k popsání některých obecných „shop“ problémů. Pokud je účelová funkce regulární, pak se v množině možných řešení nachází optimální řešení [8].

Disjunktní graf je definován jako ***G***=(V,C,D), kde:

V je množin uzlů reprezentujících operace všech úloh. Každý uzel má nastaven dobu zpracovávání. K nim jsou přidány dva pomocné uzly reprezentující začátek a konec grafu. Pomocné uzly mají dobu zpracování nastavenou na 0.

C je množina orientovaných konjuktivních hran. Tyto hrany znázorňují priority mezi jednotlivými operacemi. Navíc se zde nachází hrany mezi počátkem a všemi operacemi bez předchůdce a mezi koncem a všemi operacemi bez následovníka.

D je množina disjunktivních hran. Tyto hrany existují pro každou dvojici operací jedné úlohy, které nejsou spojeny hranou z množiny C a dále jsou zde hrany, které spojují operace vykonávané na jednom stroji a které nejsou spojeny hranami z množiny C.

Rozvrh na grafu ***G*** je množina počátečních časů $T=\{t_i \mid i \in V\}$. Dále platí:

Pro konjunktivní omezení:

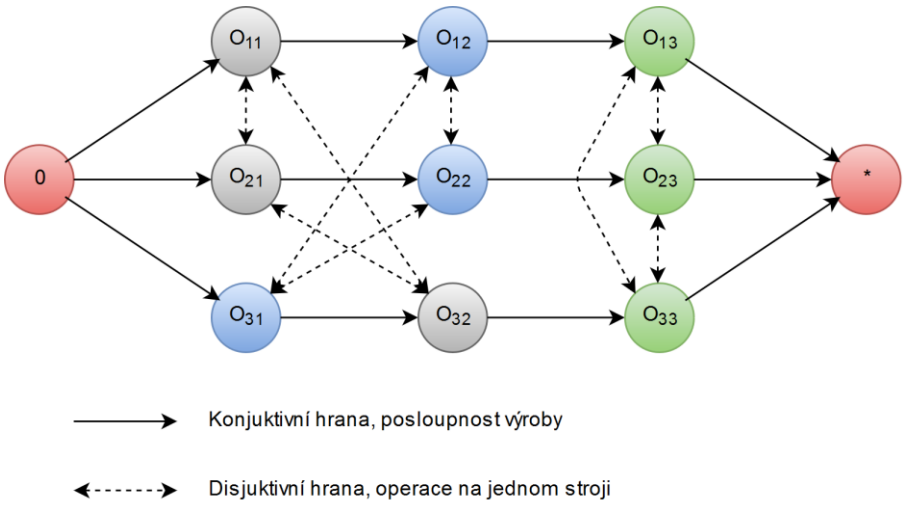
$$t_j - t_i \geq p_j \ , \forall (i,j) \in C$$

Pro disjunktní omezení:

$$t_j - t_i \geq p_j \ \vee \ t_i - t_j \geq p_j \ , \forall (i,j) \in D$$

Účelová funkce hledá nejkratší možnou cestu přes všechny vrcholy grafu při splnění všech omezení.

Zápis pomocí grafu může být použit k výpočtu začátku a konce jednotlivých operací. Začátek operace je udán kritickou cestou, což je nejdelší trasa od počátečního uzlu k uzlu operace j. Je to nejdřívější čas, kdy může operace začít.



Obrázek 2: Disjunktní graf

Pro vytvoření rozvrhu musíme vytvořit posloupnost všech úloh na všech strojích. Toho se dosáhne změnou neorientovaných hran na orientované tak, že vznikne acyklický graf. Všechny neorientované hrany musí být převedeny na orientované, vznikne tak množina S. Výsledný graf ***G(S)***=(V,C ∪ S) je acyklický.

3.5.3.1 Job shop

Rozvrhování typu Job shop je speciálním případem obecného „shop“ problému a zobecněním „flow shop“ problému. Máme n úloh (J_1,J_2,… J_n), a m strojů (M_1,M_2,…M_m). Každá úloha J_j se navíc skládá z několika operací ($O_{j,1}, O_{j,2}, \dots, O_{j,n_j}$) které mají dané pořadí, v jakém musí být provedeny. Úlohy mohou mít různý počet operací. Navíc existují precedenční omezení ve formě $O_{i,j} \rightarrow O_{i,j+1}$ pro $j = 1, \dots, n_i - 1$. Celkový počet možných rozvrhů se blíží $n!^m$ [3].

Každá z operací může být zpracována pouze na jednom ze strojů. Preempce není dovolená a stroje zvládají zpracovávat v jednom okamžiku pouze jednu operaci. Každá operace má pevně daný čas zpracování. Optimalizačním úkolem je nejčastěji nalézt takovou posloupnost operací pro každý stroj, která minimalizuje dobu dokončení.

Jenom několik specifických příkladů lze optimálně vyřešit v polynomiálním čase. Job shop problém je NP-těžký, když počet strojů nebo úloh je větší nebo rovno 3. Tvorba rozvrhu se čtyřmi stroji je NP kompletní i když jsou všechny operace proveditelné v jednotkovém čase.
Problém obchodního cestujícího je speciální případ Job shop problému s počtem strojů m=1 [1].

Johnsonův algoritmus pro Job shop problém

Tento algoritmus řeší úlohy, kde jsou dva stroje a každá úloha je zpracovávána na maximálně dvou strojích. Například $\mathbf{J_2|\ n_i<2\ |Cmax}$. Jeho výhodou je, že může být převeden na flow shop problém $F_2|\ |C_{max}$.

Úlohy jsou rozděleny do několika podmnožin.
I_1 – úlohy, které jsou zpracovávány pouze na stroji 1
I_2 – úlohy, které jsou zpracovávány pouze na stroji 2
$I_{1,2}$ – úlohy zpracovávané nedříve na stroji 1, poté na stroji 2
$I_{2,1}$ – úlohy zpracovávané nedříve na stroji 2, poté na stroji 1

Nyní podle Jacksonova algoritmu:

1) Vypočítat optimální sekvenci $R_{1,2}$ podle Johnosonova algoritmu pro množinu $I_{1,2}$
2) Vypočítat optimální sekvenci $R_{2,1}$ podle Johnosonova algoritmu pro množinu $I_{2,1}$
3) Na stroji 1 provést úlohy z $L_{1,2}$ podle sekvence $R_{1,2}$, poté v jakémkoli pořadí I_1 a nakonec úlohy z $L_{2,1}$ podle sekvence $R_{2,1}$
4) Na stroji 2 provést úlohy z $L_{2,1}$ podle sekvence $R_{2,1,}$ poté v jakémkoli pořadí I_2 a nakonec úlohy z $L_{1,2}$ podle sekvence $R_{1,2}$

Mějme následující zadání. Úlohy 1-4 musí být zpracovány nejdříve na stroji 1. Úlohy 5-7 musí být nejdříve zpracovány na stroji 2, úlohy 8 a 9 musí být zpracovány pouze na stroji 1 a úlohy 10 a 11 pouze na stroji 2.

J	1	2	3	4	5	6	7	8	9	10	11
p_{1j}	3	2	1	1	2	4	3	1	2	0	0
p_{2j}	2	1	2	1	4	8	9	0	0	2	1

Tabulka 9: Zadání rozvrhu

Vytvořené množiny podle Jacksonova algoritmu jsou tyto:

$I_1 = \{J_8, J_9\}$

$I_2 = \{J_{10}, J_{11}\}$

$I_3 = \{J_1, J_2, J_3, J_4\}$

$I_4 = \{J_5, J_6, J_7\}$

Vytvořený rozvrh pak vypadá následovně:

čas	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
M1	J3	J1	J1	J1	J2	J2	J4	J4	J8	J9	J9	J5	J5	J6	J6	J7	
M2	J7	J7	J5	J5	J5	J6	J6	J6	J10	J10	J11	J1	J1	J2	J3	J3	J4

Tabulka 10: Vytvořený rozvrh

Pokud by se stalo, že by jedna úloha měla probíhat na dvou strojích zároveň, tak na jednom nezačne, dokud neskončí na předchozím a daný stroj bude chvíli neaktivní. Lze dokázat, že alespoň jeden stroj nemá žádné neaktivní období.

Pro řešení Job shop problému byla navržena řada heuristických algoritmů. Například prioritní pravidla, simulované žíhání, tabu search, genetické algoritmy a další. Lze využít i již dříve uvedenou metodu Branch and Bound [3].

Shifting bottleneck heuristika

Tato metoda byla vyvinuta pro snížení celkové doby zpracování. Pracuje s úvahou, že úlohy spolu soupeří o zdroje. A v takovém systému je jedno nebo více úzkých míst, které brzdí dokončení zpracování. Tato heuristika se snaží o minimalizaci tohoto efektu. Metoda funguje v systémech s konečným počtem úloh a s konečným počtem strojů.

Jedná se o iterativní heuristiku. Na začátku je konjunktní graf, který není nijak optimalizován, jsou v něm jen zaznamenána precedenční omezení.
Na začátku je zadaný problém zapsán pomocí grafu [6].

```
M0 = Ø; //naplánované stroje;
M = všechny stroje;
G = zadání ve formě grafu s pouze konjunktivními hranami
C_max = kritická cesta v G
K = Ø;  //stroj který maximalizuje L_max
F = Ø; // pomocná proměnná s nejlepší L_max

  WHILE M != M0 // dokud nejsou naplánovány všechny stroje
     //Zjištění stroje způsobujícího úzké hrdlo
     FOREACH Mi in M – M0
          Naplánuj stroj Mi podle 1|r_j|L_max
          Z = zpoždění naplánovaného rozvrhu
          IF Z < F
               K = Mi
               F = Z
     //Naplánování úzkého místa
     Naplánuj stroj K podle 1|r_j|L_max
     Úprava hodnot G // přidání disjunktních hran
     M0=M0-K;

     //Přeplánování již naplánovaných strojů
     FOREACH Mi in M0 – K
          Zruš disjunktní hrany patřící Mi v G
          Naplánuj stroj Mi podle 1|r_j|L_max
```

Algoritmus 2: Pseudokód metody Shifting bottleneck

3.5.3.2 Cycle shop

Cycle shop je zvláštní případ problému job shop, který se zabývá periodickými procesy. Je to zároveň i rozšíření problému Flow shop, kde všechny úlohy mají stejné pořadí operací, ale některé operace mohou být opakovány na některých strojích. Byl vytvořen pro řešení problémů spojených s výrobou procesorů, kde často dochází k opakování jedné operace. Problémy L2||C_{max} a L2|pmtn|C_{max}. I nejjednodušší úlohy minimalizující maximální čas dokončení výroby, kde doby zpracování nejsou omezeny je NP-těžké [1].

3.5.3.3 Open shop

Rozvrhování podle modelu open shop, je zvláštní případ obecného „shop" problému.
Máme zde m strojů (M_1,M_2,…M_m), které provádějí různé operace a n úloh (J_1,J_2,… J_n).

- Každá úloha i se skládá z m operací O_{ij} (j = 1, … , m), kde O_{ij} musí být zpracováno na stroji M_j.
- Neexistují prioritní omezení mezi operacemi.

Úkolem je nalézt pořadí operací patřících k jedné úloze a zároveň pořadí operací zpracovávaných na jednom stroji.
Jako $T_{i,j}$, pak označujeme pořadí operace na stroji M_j a doba zpracování je $p_{i,j} \geq 0$. Celkový čas zpracování úlohy J_i je tedy $p_i = \sum_j p_{i,j}$.
Požadavek na čas stroje M_j je $m_j = p_j = \sum_i p_j$.
Z těchto dvou hodnot pak vybereme maximum $h = max(p_i, m_j)$. Rozvrhovací omezení jsou pak následující:

- V každém okamžiku může každý stroj zpracovávat pouze jednu operaci.
- Každá úloha může být zpracovávána v jednom okamžiku pouze jedním strojem.
- Zpracování každé operace $T_{i,j}$ na stroji M_j trvá $p_{i,j}$ časových jednotek.

Koncový čas každého rozvrhu je nejméně h. Hlavní rozdíl mezi Open shop a Flow shop je v tom, že flow shop potřebuje, aby operace jednotlivých úlohy byly zpracovávány v určitém pořadí. V open shop nezáleží na tom, v jakém pořadí jsou jednotlivé operace vykonávány.
V případě job shop, úlohy mohou mít jakýkoliv počet operací. Každá operace každé úlohy je přidělena jednomu ze strojů. Oproti tomu u open shop a flow shop, operace jsou

přesně přiřazeny jednotlivým strojům. Pořadí, v jakém jsou operace prováděny v job shop a flow shop je sekvenční. Open shop se chová jako flow shop, u kterého nezáleží, v jakém pořadí jsou vykonány jednotlivé operace.
Open shop byl vytvořen, protože řada problémů z reálného světa nezapadala přesně do kategorie flow shop. V současné době existuje řada algoritmů na řešení tohoto problému. Pokud se v modelu nachází pouze dva stroje a je preemptivní, je to problém s lineární složitostí. Se dvěma stroji a nepreemptivními úlohami musíme počítat s polynomiální složitostí. Pokud jsou však stroje 3 a úlohy jsou nepreemptivní, již se jedná o problém NP-těžký.
V teorii rozvrhování se open shop často vyskytne jako část jiných rozvrhovacích problémů. Často se tak stává při řešení problémů pomocí lineárního programování. Nejdříve se definuje množina intervalů a pomocí LP se určí množství času, které připadne na kterou úlohu na určitém stroji v konkrétní časový interval. Nyní nám zůstává jeden nebo více open shop problém.

Příklad praktického využití open shop je v satelitní komunikaci (SS/TDMA), kde spolu komunikuje více pozemních stanic. Používá se také při komunikaci v optických počítačových sítích, které pracují s více vlnovými délkami světla. Použití je však i v mnoha jiných oblastech [1].

Zobecněním rozvrhování open shop je tvorba školního rozvrhu. Učitelé jsou stroje a úlohy jsou třídy. Je třeba nalézt takový rozvrh, kdy profesor učí v jednom okamžiku pouze v jedné třídě a zároveň v každé třídě neučí více než jeden učitel. Dále jsou zde omezení v čase, kdy mohou lekce probíhat.

Pokud jsou časy zpracování $p_{i,j}$ jednotlivých operací neurčité, a preempce není povolena, pak jen málo problémů je řešitelných v polynomiálním čase. Například
$O_2| |C_{max}$ nebo O| pmtn | C_{max}. Problém O| pmtn | $\sum C_i$ je NP těžký. [1]

Jednotková doba zpracování

Pro vyřešení open shop problému s jednotkovou dobou zpracování byl navržen tento postup, který transformuje obecný problém P na P':

- Všechny stroje jsou nahrazeny m paralelními stroji.
- Všechny úlohy i jsou změněny na řetězec jednotkových operací O_{ik} (k = 1, …, m)

Rozvrh je pak zapsán binární maticí A=($a_{i,t}$), kde $a_{i,t}$ =1 právě když O_{ik} je naplánován na čas t. Takový rozvrh lze převést na realizovatelný rozvrh původního problému přiřazováním strojů jednotlivým jednotkovým operacím podle následujícího klíče:

- Všechny jednotkové operace náležící jednomu řetězci jsou přiřazeny různým strojům
- Všechny operace naplánované na jeden okamžik jsou přiřazeny různým strojům.

Toto přiřazování strojů je ekvivalentní problému barvení hran bipartitního grafu *G* definovaném podle *A*, který má přesně *m* barev [3].

Bipartitní graf je takový graf, který lze rozdělit na dvě množiny takovým způsobem, že vrcholy v jedné množině mezi sebou nemají žádnou hranu. Problém barvení hran grafu se snaží vyřešit obarvení všech hran grafu tak, aby každé dvě hrany mající společný vrchol byly obarveny odlišnou barvu.

Pro barvení hran existují vhodné algoritmy.

Graf G má n*m hran. Pokud je úloh více nebo rovno jako strojů, lze dosáhnout s použitím vhodných algoritmů časové složitosti O (nm $\log^2$(nm)).

Algoritmus LPTF

Jeden z algoritmů pro řešení open shop problémů se nazývá LPTF (Longest processing time first). Principem tohoto algoritmu je, že kdykoliv je uvolněn stroj, je mu přidělena úloha s nejdelším zbývajícím časem zpracování [6].

Již dříve zmíněný problém $O_2| |C_{max}$ je zadán následující tabulkou, kde například p_{1j} je doba zpracování úlohy j na stroji 1.

Úloha	**J1**	**J2**	**J3**	**J4**
$\mathbf{p_{1j}}$	6	5	4	7
$\mathbf{p_{2j}}$	3	8	9	4

Tabulka 11: Zadání rozvrhu

Postup:

1) Nejdelší dobu zpracování na stroji 1 má úloha 4, obsadí tedy tento stroj.
2) Úloha 3 obsadí stroj 2.
3) Stroj 1 skončí před strojem 2 a vybere tedy nejdelší zbývající čas, což je úloha 1.
4) Nyní je volný stroj 2. Nejdelší vhodná úloh je č. 2.
5) …

Tímto způsobem se pokračuje, dokud nejsou žádné v úlohy připravené k naplánování. Výsledek lze znázornit následujícím Ganttovým diagramem.

čas	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
M1	4	4	4	4	4	4	4	1	1	1	1	1	1					2	2	2	2	2	3	3	3	3
M2	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	4	4	4	4	1	1	1		

Tabulka 12: Vytvořený rozvrh

V čase 14, kdy na stroji 1 skončila úloha 1, a měl být vybrána další úloha, došlo k tomu, že nejdelší nezpracovaná úloha byla úloha 2, která však byla zpracovávána na stroji 2. Došlo tedy ke vzniku nečinnosti na 4 časové jednotky.

3.5.3.4 Flow shop

Rozvrhování podle modelu flow shop, je konkretizovaný případ obecného „shop“ problému [2].

V systému se nachází m strojů a n úloh. Každá úloha má přesně m operací. Operace mají určeno, na kterém stroji mají být provedeny. Platí zde i obecná pravidla jako že každý stroj může v jednom okamžiku zpracovávat jen jednu úlohu a jedna úloha může být zpracovávána jen jedním strojem.

- Každá úloha i se skládá z m operací O_{ij} které mají čas zpracování $p_{i,j}$ (j = 1, … , m), kde O_{ij} musí být zpracováno na stroji M_j.

- Existují prioritní omezení mezi operacemi ve formě $O_{i,j}$ -> $O_{i,j+1}$, (i = 1, … , m-1) pro každé i = 1, … , n je každá úloha zpracována na stroji 1, pak na stroji 2, atd.
- Všechny úlohy mají předem dané pořadí, v jakém jsou na strojích vykonávány.
- Pořadí operací na strojích může být různé.

Pokud máme pouze dva stroje, může být problém vyřešen s časovou složitostí O (n*log n). Pro více strojů je tento problém NP- těžký [3].

F_2| |C_{max}

Problém F_2| |C_{max} je jedním z mála flow shop problémů, který je řešitelný v polynomiálním čase. K tomu slouží **Johnsonův algoritmus** [1][6].

$p_{1,j}$ značí dobu vykonávání j-té operace na stroji 1.

$p_{2,j}$ značí dobu vykonávání j-té operace na stroji 2.

- Rozděl úlohy do dvou množin tak, že množina N1 bude obsahovat úlohy s $p_{1,j} < p_{2,j}$ a množina N_2 bude obsahovat úlohy s $p_{1,j} \geq p_{2,j}$.
- Úlohy v množině N_1 jdou nejdříve a to ve vzestupném pořadí podle $p_{1,j}$.
- Úlohy z množiny N_2 jdou v sestupném pořadí podle $p_{2,j}$

Existuje několik předpokladů:

- Doba provádění operace se nemění
- Operace jedné úlohy nemohou být prováděny současně
- Nejdříve proběhne operace 1, poté operace 2
- Úlohy mezi sebou nemají závislosti

$p_{1,j}$ značí dobu vykonávání j-té operace na stroji 1.

Následující tabulka ukazuje zadání flow shop problému s jednotlivými stroji, úlohami a prováděcím časem operací jednotlivých úloh.

	J1	J2	J3	J4	J5
M1	4	3	3	1	8
M2	8	3	4	4	7

Tabulka 13: Zadání rozvrhu

U první tabulky se postupně vyhledává nejkratší $p_{1,j}$ a pokud je $p_{1,j} < p_{2,j}$, tak se sepíše do tabulky N1. U zbylých se postupuje od největšího $p_{2,j}$ a zapisují se do N2.
Vzniklé množiny vypadají takto:

N1 = { J4,J3,J1}

N2 = {J5,J2}

Nyní se postupně vybírají nejdříve úlohy z množiny N1 a postupně se provádějí jejich operace. Vytvořený rozvrh vypadá takto.

čas	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
M1	J4	J3	J3	J3	J1	J1	J1	J1	J5	J5	J5	J5	J5	J5	J5	J2	J2	J2								
M2		J4	J4	J4	J4	J3	J3	J3	J3	J1	J1	J1	J1	J1	J1	J1	J5	J5	J5	J5	J5	J5	J5	J2	J2	J2

Tabulka 14: Vytvořený rozvrh

Johnosonův algoritmus je optimální a pracuje s časovou složitostí O (n*log n).

$F_3 | |C_{max}$

Zkusme nyní do systému přidat ještě jeden stroj. Obecně nelze použít Johnsonův algoritmus. Pokud jsou však splněny dvě podmínky, můžeme jej použít [6].

$$\min(p_{1j}) \geq \max(p_{2j})$$

nebo

$$\min(p_{3j}) \geq \max(p_{2j})$$

Doba nejkratšího zpracování úlohy na stroji 2 musí být vždy menší nebo rovna době zpracování na jednom ze zbývajících strojů. Je potřeba vytvořit náhradní zadání. Pro každou úlohu se vezme doba zpracování na druhém stroji a přičte se k dobám zpracování na prvním i třetím stroji. Johnsonův algoritmus poté proběhne stejným způsobem jako pro dva stroje ale v tomto případě ignoruje stroj č. 2.

Bylo dokázáno, že po splnění podmínek i v tomto případě vytváří Johnsonův algoritmus optimální řešení.

Pokud nejsou splněny všechny požadavky, tak Johnsonův algoritmus vytváří rozvrh, který sice není optimální, ale dobrým východiskem pro jiné optimalizační algoritmy [3].

$F_m| |C_{max}$ pro m>3

Toto je rozšíření flow shop problému se třemi stroji. V současné době neexistuje žádné přesné efektivní řešení. Ačkoliv to na první pohled může vypadat jednoduše, s rostoucím počtem strojů se rychle dostaneme k velmi komplexním problémům, což je způsobeno kombinatorickou povahou flow shop problému. Kromě Johnsonova algoritmu pro optimalizaci doby výroby na dvou strojích neexistuje žádný algoritmus, který by prováděl optimalizaci pro jiný aspekt nebo pro více strojů. Bylo dokázáno, že nepreemptivní flow shop s minimalizací doby výroby je NP kompletní problém [1].

Permutační flow shop

Zvláštním případem flow shopu je permutační flow shop, kde pořadí zpracování operací je stejné pro všechny úlohy. Má tu výhodu, že počet možných řešení je $n!$. Pokud máme systém se třemi úlohami a dvěma stroji, můžeme vytvořit následující sekvence úloh: J1-J2-J3, J3-J1-J2, J1-J3-J2, J3-J2-J1, J2-J1-J3 a J2-J3-J1. Tedy 3!. Ve standardním flow shop problému máme takovýchto sekvencí $(n!)^m$ [6].

4 Umělá inteligence

4.1 Úvod do umělé inteligence

Umělá inteligence se v poledních letech dostává do hledáčku odborné i neodborné veřejnosti stále častěji. Nejdůležitější roli hraje ve specializovaných oborech, jako je například robotika, ale často se s ní můžeme setkat třeba i při zpracování obrazu, v expertních systémech, a při zpracování dat finančními institucemi. S nárůstem výpočetního výkonu a komplexnosti technologií se zvyšuje i složitost zpracovávaných úloh. Umělá inteligence by se tak mohla stát odpovědí na tyto zvýšené požadavky a přispět tak k dalšímu vývoji mnoha technických oborů a třeba i k vývoji sama sebe.

4.2 Historie

Za první odbornou činnost spadající do problematiky umělé inteligence jsou považovány práce, na jejichž základě Warren McCulloch a Walter Pitts definovali v roce 1943 umělý neuron a naznačili možnosti umělých neuronových sítí, což je označení pro výpočetní modely inspirované funkcemi a chováním biologických struktur. Další výrazný pokrok v tématice neuronových sítí nastal v roce 1949, když Donald Hebb formuloval základní pravidlo pro učení neuronových sítí, které se v oblasti neuronových sítí používá dodnes [11].

V rámci umělé inteligence má obzvlášť důležité místo Alan Mathison Turing, který se zabýval myšlenkou výpočetních strojů, které by se dokázaly učit z vlastních zkušeností a řešit problémy. V roce 1950 představil návrh testu počítačové inteligence. Tento takzvaný Turingův test zkoumá, zda počítač vybavený umělou inteligencí dokáže člověka přesvědčit o tom, že není pouhým strojem, ale skutečnou lidskou bytostí. Test byl samozřejmě prováděn tak, aby se hodnotilo pouze chování a komunikační schopnosti počítače a zabránilo se hodnocení založenému na prvcích, které s inteligencí nesouvisí (vzhled, hlas atd.). Testovaná osoba předem neví, zda bude komunikovat s počítačem nebo s člověkem a pokud to v průběhu testu nedokáže rozpoznat, testovaná verze umělé inteligence úspěšně splní Turingův test. Pokládané otázky mohou být libovolně

komplikované a obsáhlé a testovaný počítač může dělat vše pro to, aby působil jako lidská bytost. I přesto, že tento test existuje už více než půl století, žádná umělá inteligence ho zatím jednoznačně nesplnila.

Vývoj práce na umělé inteligenci lze rozčlenit do několika etap [12]. První etapa probíhala v padesátých a v průběhu šedesátých let a byla charakterizována nadšenou prací a mnoha velkolepými vizemi a nadějemi. Byl vyvinut programovací jazyk LISP. Objevily se programy určené k řešení obecných úloh, první programy určené pro hraní her, první program pro zpracování přirozeného jazyka a mnoho dalšího. Ukázalo se, že problematika UI je složitější, než se z počátku zdálo.

Druhá etapa vývoje umělé inteligence nastala v sedmdesátých letech. Ve výzkumu došlo ke znatelné stagnaci, ale pokrok se i přesto zcela nezastavil. Objevily se například první expertní systémy pro identifikaci organických sloučenin, hledání ložisek rud, či diagnostiku infekčních onemocnění, programy pro rozpoznávání řeči a komunikaci v přirozeném jazyce. Byl vytvořen nový jazyk pro umělou inteligenci zvaný PROLOG.

Třetí etapa započala v letech osmdesátých, kdy se umělá inteligence konečně stala uznávanou vědou a začala se dostávat do komerční sféry. Začaly vznikat expertní systémy, syntetizátory řeči, systémy pro zpracování přirozeného jazyka, systémy pro zpracování obrazů, návrhy počítačů s architekturami vhodnými pro umělou inteligenci (LISPovské a PROLOGovské počítače). Obnovil se také výzkum problematiky neuronových sítí.

Poslední etapa vývoje umělé inteligence začala přibližně na přelomu osmdesátých a devadesátých let a trvá dodnes. Charakterizována je především přenosem zájmu o umělou inteligenci z výzkumných laboratoří do reálného světa. Umělé inteligence se zařadila mezi ostatní počítačové vědy a pozornost výzkumných pracovníků se zaměřila na práci s nejistými a neúplnými informacemi, neboli takzvaný softcomputig, kam spadají neuronové sítě, genetické algoritmy, fuzzy logika, hrubé množiny, chaos atd.

4.3 Definice umělé inteligence

Definice pojmu umělá inteligence není jednoduchá a existuje jich celá řada, shodují se však v tom, že jde o vlastnost člověkem vytvořeného systému, který je schopen přijímat podněty z okolí, analyzovat vztahy mezi nimi, objevovat zákonitosti, vytvářet si modely a s jejich pomocí provádět rozhodnutí a řešit problémy.

Konečným cílem výzkumu umělé inteligence je zkonstruování stroje nebo programu, který se při komunikaci s člověkem jeví jako skutečný člověk (Turingův test). Umělá inteligence se v podstatě snaží vybudovat výpočetní modely poznávacích procesů [11].

4.4 Definice inteligence

Obecně by se dalo říct, že inteligentní chování je takové chování, které se dokáže přizpůsobit novým okolnostem. Lidská inteligence však není pouze jedna specifická schopnost, ale spíše celá řada oddělených složek. Mezi těmito složkami je především učení, logické myšlení, řešení problémů, vnímání a porozumění jazyku.

4.4.1 Učení

Učení existuje v mnoha různých formách. Vůbec nejzákladnější proces učení je ten, který je založený na systému pokusů a omylů. Tento systém by se dal definovat tím, že postupy s pozitivními odezvami jsou v jeho rámci ukládány pro pozdější použití a postupy s negativními odezvami jsou odsouvány do pozadí. Pokud do učení zapojíme i generalizaci umožníme tím dosažení pozitivních výsledku i v situacích, se kterými se daný inteligentní systém nikdy předtím nesetkal.

4.4.2 Uvažování

Uvažování je schopnost vyvozovat závěry odpovídající dané situaci. Schopnost vyvozovat závěry však není jasný důkaz, že daný počítač nebo program dokáže rozumově uvažovat. Závěry musí být relevantní pro daný úkol nebo situaci. A právě potřeba rozlišovat mezi relevantními a irelevantními informacemi je jedním z nejpodstatnějších problémů, kterým musí umělá inteligence dnes čelit.

4.4.3 Vnímání

Vnímání je proces využívající smyslové orgány pro snímání okolního prostředí a následnou analýzu vztahů a vlastností dané scény a objektů, které jsou v ní zahrnuty. Tato analýza je komplikována tou skutečností, že každý z objektů může při různých příležitostech zaujímat různé podoby. Vliv zde může mít úhel pohledu, směr a intenzita světla a mnoho dalšího. Umělé vnímání je v současné době dostatečně pokročilé například na to, aby umožnilo speciálním automobilům bez řidičů relativně bezpečný pohyb v běžném provozu.

4.4.4 Porozumění jazyku

Jazyk se dá definovat jako systém znaků, kterým je přiřazen význam podle jisté konvence. Mohou sem spadat například určité kombinace písmen, zvuků nebo třeba obrázků. V současnosti existují počítačové programy, které jsou v omezených kontextech schopny poměrně plynule používat konkrétní jazyk a reagovat pomocí něj na různé otázky. Problém však spočívá v tom, že tyto programy danému jazyku ve skutečnosti nerozumí. Aby se dalo říci, že jazyk opravdu chápe, je potřeba, aby se ho naučil od jeho základů a dokázal se s jeho pomocí zapojit do plnohodnotné konverzace s ostatními členy komunity, která tento jazyk používá.

4.4.5 Řešení problémů

Řešení problémů je velmi důležitá a obsáhlá součást umělé inteligence. Do řešení problémů můžeme například zahrnovat hledání vítězných tahů v deskových hrách, identifikaci osob na fotografiích, plánování série pohybů robota a mnoho dalšího. Umělá inteligenci si musí ze získaných informací vybrat relevantní znalosti a s jejich pomocí vyvozovat určité závěry.

4.5 Metody umělé inteligence

V této kapitole budou vysvětleny některé základní metody využívané v umělé inteligenci. Nejdříve budou popsány jednoduché metody a postupně se propracujeme ke složitějším [12].

Úlohy umělé inteligence lze charakterizovat jako množinu počátečních stavů, množinu koncových stavů a množinu operátorů popisující přechod mezi jednotlivými stavy.
Každou metodu lze hodnotit podle čtyř kritérií.

- úplnost: pokud existuje řešení, tak bude nalezeno
- časová náročnost
- paměťová náročnost
- optimálnost: pokud existuje nejlepší řešení, tak ho daná metoda nalezne

4.5.1 Metody prohledávání stavového prostoru

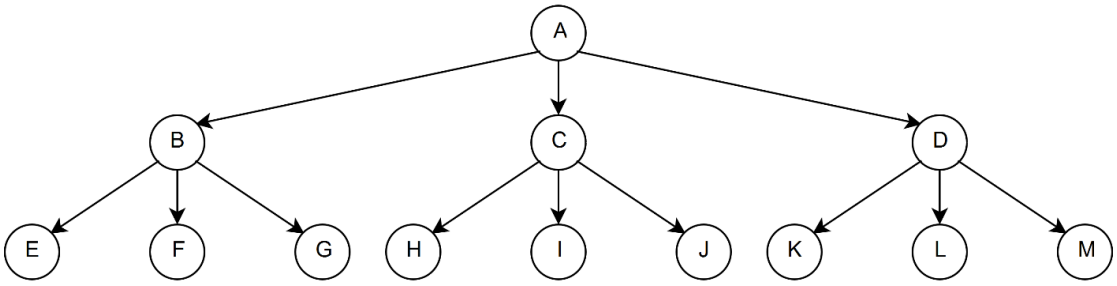
Úlohu lze popsat orientovaným grafem, kdy uzly jsou jednotlivé stavy úlohy a hrany jsou přechody mezi stavy. Máme množinu počátečních stavů, množinu koncových stavů a množinu operátorů popisující přechod mezi jednotlivými stavy. Využívají se, když je důležité zaznamenat celou cestu od počátku k cíli. Tyto metody lze dále dělit neinformované a informované [12].

4.5.1.1 Neinformované prohledávání stavového prostoru.

Tyto metody se slepě snaží dostat do cílového stavu bez znalosti jeho umístění. Prostě postupně zkouší všechny možné kombinace a doufají, že jedna z nich bude vyhovovat koncovým podmínkám.

BFS (Breadth first search)
Metoda spoléhá na použití fronty při procházení stavů. Metoda vždy nalezne optimální řešení, její paměťová a časová náročnost je však exponenciální, proto není vhodná pro náročnější úlohy.

1) Do fronty je umístěn počáteční stav.
2) Pokud je fronta prázdná, řešení neexistuj a následuje konec výpočtu.
3) Vybrání prvního uzlu z fronty, pokud je cílovým, tak konec, jinak nalezení jeho následovníků a jejich umístění do fronty. Pokračování na bod 2.



Obrázek 3: Stavový prostor

Na tomto obrázku lze fungování vysvětlit následovně: Ve frontě se nachází počáteční uzel A. Vyjmeme vrcholový uzel A a zjistíme, jestli je koncovým stavem. Pokud není, jsou do fronty vloženy jeho následovníci B, C a D. Nyní se na začátku fronty nachází stav B, je tedy vyjmut a jsou nalezeni jeho následovníci a umístěni do fronty. Následuje expanze uzlu C a pak D a tak dále. Algoritmus postupuje po jednotlivých úrovních.

DFS (Depth first search)

Velice podobná metodě BFS, hlavním rozdílem je použití zásobníku místo fronty. Neprohledává graf po úrovních ale po listech. Expanze uzlů bude probíhat v pořadí A, D, M, L, K, C, J… Algoritmus není ani úplný ani optimální. Časová složitost je exponenciální, ale prostorová lineární.

UCS (Uniform cost search)

Tato metoda využívá místo fronty nebo zásobníku seznam. V grafu je každá cesta mezi dvěma uzly ohodnocena, tzn. je nalezena cena přechodu mezi dvěma uzly. Ze seznamu je pro expanzi vybrán vždy uzel, který má nejnižší ohodnocení, tedy takový uzel, ke kterému jsme se dostali s nejmenšími náklady. Jinak metoda funguje stejným způsobem jako předchozí metody. Algoritmus je úplný a optimální, časová i prostorová náročnost je exponenciální.

4.5.1.2 Informované metody

Tato skupina metod využívá informace o cílovém stavu a mohou je využít pro rozhodování při expanzi uzlů v grafu.

BS (Best first search)

Podobně jako u metody UCS je využit seznam. Rozdíl je ve způsobu výběru nejvhodnějšího uzlu. Pro výpočet je vytvořena heuristická funkce *h (n), která* zjišťuje odhad ceny cesty z daného uzlu do cíle. Hodnota této funkce je pak sečtena s cenou již uražené cesty *g (h).*

$$f(n) = g(n) + h(n)$$

Greedy search

Zvláštní případ BS, kdy se uvažuje pouze s odhadem ceny cesty do cíle. Úspěšnost metody silně závisí na vhodně zvolené heuristické funkci.

Hill climbing

Jedná se o úlohu typu lokálního prohledávání, je pro ně typické že nezaznamenávají celou cestu od počátku k cíli, ale pouze koncový stav. Nehledají optimální cestu, ale optimální stav. Výhodou je minimální paměťová náročnost a skutečnost, že naleznou řešení i v případě, že z časových důvodů nelze použít metody popsané v předchozí kapitole.
Podobně jako Gready search využívá Hill climbing heuristickou funkci, která však nepopisuje vzdálenost od cíle, ale kvalitu nalezeného řešení.

1) Je vybrán uzel a je nalezen nejlepší následník.
2) Pokud má současný uzel lepší ohodnocení než následník, jedná se o hledaný výsledek.
3) Pokračuj bodem 1 s uzlem nalezeným v bodě 2.

Metoda má nevýhodu, že se může zaseknout v lokálním extrému. Vyřešit to lze spuštěním algoritmu několikrát s nastavením jiného počátečního uzlu. Nebo využitím seznamu, do kterého se v každém kroku ukládá několik nejlépe ohodnocených uzlů.

4.5.2 Metody s omezujícími podmínkami

Algoritmy pro výpočty s omezujícími podmínkami zkoumají vnitřní strukturu jednotlivých stavů.
Formálně lze problém popsat následovně: Máme množinu proměnných P a dále množinu omezení O. Kandidát na koncový stav nemá na začátku přiřazenu žádnou proměnnou.

Postupně jsou mu proměnné nastavovány tak, aby zároveň byly splněny všechny podmínky. Konečný stav nastane v okamžiky, kdy jsou nastaveny všechny proměnné.

Backtracking pro problém s omezujícími podmínkami

Problémy s omezujícími podmínkami se často řeší pomocí algoritmu Backtracking. Tento algoritmus je vhodný pro problémy, které

1) Na zásobník je umístěn počáteční uzel
2) Pokud je zásobník prázdný, úloha nemá řešení
3) Z uzlu na vrcholu zásobníku vytvoř další možné řešení, pokud to nelze, uzel odstraň a běž na 2
4) Pokud nově vygenerované řešení je koncovým stavem, tak úspěšný konec. Jinak vložit uzel na zásobník a pokračovat bodem 2.

Na příkladu Problému osmi dam lze algoritmus jednoduše vysvětlit. V tomto problému jde o způsob, jak na šachovnici naskládat osm dam tak, aby na žádném řádku, v žádném sloupci a ani v žádné úhlopříčce nebyly dvě dámy.

Algoritmus umístí první dámu na souřadnice [1,1]. Druhou zkusí hned vedle na [1,2], ale to nevyhovuje podmínce, že na řádku nesmí být dvě dámy. Takto projede celý řádek. Následovně zkouší na [2,1], to nevyhovuje kvůli podmínce o dvou dámách ve sloupci. Zkusí tedy [2,2], ale to nevyhovuje kvůli podmínce o úhlopříčce. Úspěšný je až v pozici [2,3]. Algoritmus se pokusí umístit všechny dámy, ale zjistí, že všechny již umístit nelze. Musí se tedy postupně vracet až na začátek. Zde přesune první dámu na [1,2] a obvyklým způsobem pokračuje dále. Pokud má úloha řešení, tak ho algoritmus najde.

4.5.3 Další metody

Pro další typu úloh byly vyvinuty ještě jiné zajímavé metody. Například při rozkladu úloh na podproblémy se spoléhá na skutečnost, že pokud máme problém P, tak ten je řešitelný když je řešitelný alespoň jeden z jeho podproblémů, případně když jsou řešitelné všechny jeho podproblémy. Tento způsob je využit v metodách hraní jednoduchých her. Jsou myšleny hry, kde proti sobě soupeří dva spoluhráči a vzájemně se střídají v tazích ve hře. Počítá se s tím, že stavový prostor celé hry je relativně malý a lze předem spočítat všechny možné tahy dvou hráčů. Když tedy jeden hráč má přístup k vypočítanému stavovému

prostoru, ví jakým způsobem reagovat na protihráče tak, aby vyhrál. Tyto metody však nelze použít třeba pro šachy, protože stavový prostor je příliš velký a nelze jej v rozumném čase spočítat.

4.6 Soft computing

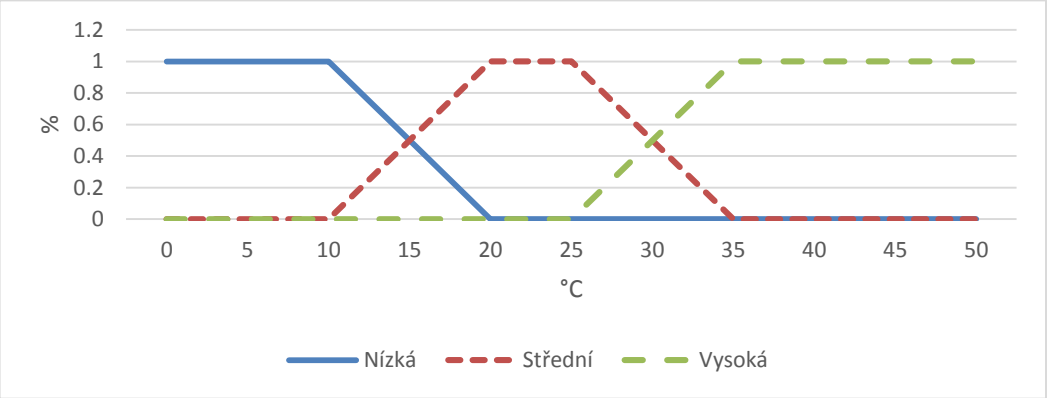
Až do teď uváděné metody a algoritmy vždy hledaly řešení exaktním způsobem. Neuvažovaly ve výpočtu s neurčitostmi a předpokládaly, že hodnoty všech proměnných jsou přesně zadány a informace jsou kompletní. Dá se tedy označit za Hard computing.
Soft computing se pouští do oblastí, kde toto neplatí. Dokáže si poradit s informacemi, které jsou popsány nepřesně a neúplně. Soft computing je spojením mnoha různých, zdánlivě nesouvisejících metod, které jsou využity pro hledání řešení. Metody soft computingu se soustředí na co nejrychlejší nalezení řešení, které však nemusí být vždy optimální, je mu však velmi blízko. Problém s optimálním řešením je ten, že k jeho výpočtu spotřebujeme příliš mnoho zdrojů (čas, peníze, procesorový výkon, …). Pokud jsme schopni nalézt řešení, které se jen blíží tomu optimálnímu, ale spotřebujeme na něj jen zlomek zdrojů, má pro nás takové řešení mnohem větší hodnotu [9] [10].
Inspirace Soft computingu vychází z lidského chování. Člověk má o reálném světě okolo sebe často jen nepřesné a neúplné informace, avšak i s takovýmto vstupem je schopen problémy vyřešit. Například hod míčem. Existují rovnice, které přesně určují pod jakým úhlem a jakou silou míč hodit aby dopadl na požadované místo. Lidé si ale většinou nic nepočítají a ze zkušenosti vědí, že když hodí míčem danou silou určitým směrem, že je velká pravděpodobnost že dopadne velmi blízko cíle. Metody Soft computingu také často vycházejí z naučených znalostí. Nejdříve je jim poskytnuta množina dat, na které se mohou trénovat a až později jsou jim předložena data, ze kterých mají vyvodit nějaké závěry. V případě řízení strojů je důležité, že metody mají zpětnou vazbu a mají možnost zasahovat do činnosti po celou dobu jejího provádění, například řízení automobilu.
Typickými oblastmi, kde se Soft computing využívá, jsou rozpoznání textu a hlasu, počítačové vidění, bankovnictví, plánování a kontrola výroby, rozmístění součástek na plošném spoji, řízení automobilu, atd.

4.6.1 Fuzzy logika

V klasické logice se pracuje pouze se dvěma hodnotami ANO nebo NE, výrok může být pravdivý nebo nepravdivý. Tato vlastnost se hodí pro přesný popis vlastností. Pro popis neurčitostí a částečných pravd je však potřeba nalézt lepší nástroj. Již v antice byl formulován paradox, který dobře popisuje problematiku fuzzy množin: Máme-li malou hromadu kamení, tak přidáním jednoho dalšího kamenu máme stále malou hromadu. Je tedy každá hromada malá?

Pro řešení tohoto problému byla navržena fuzzy logika, která místo striktního rozlišování 0/1, vyjadřuje stupeň pravdivosti nějakého výroku, míru příslušnosti k určité množině. V systému popsaném pomocí fuzzy logiky se nachází tři základní kroky [9]:

- Fuzzifikace: převod numerických hodnot na fuzzy množiny
- Inference: výpočet s fuzzy hodnotami
- Defuzzifikace: převod zpět na numerické hodnoty



Graf 1: Členské funkce při fuzzifikaci teploty

Obrázek ukazuje, jak lze pomocí fuzzy množin reprezentovat hodnoty v systému s jednou proměnou, zde konkrétně s teplotou. Po převedení číselné hodnoty na slovní popis, se na ni uplatní pravidla typu <když><pak>. Každé pravidlo může mít nastaveno svoji váhu.

<když> teplota = nízká <pak> topit hodně

<když> teplota = střední <pak> topit málo

<když> teplota = vysoká <pak> netopit

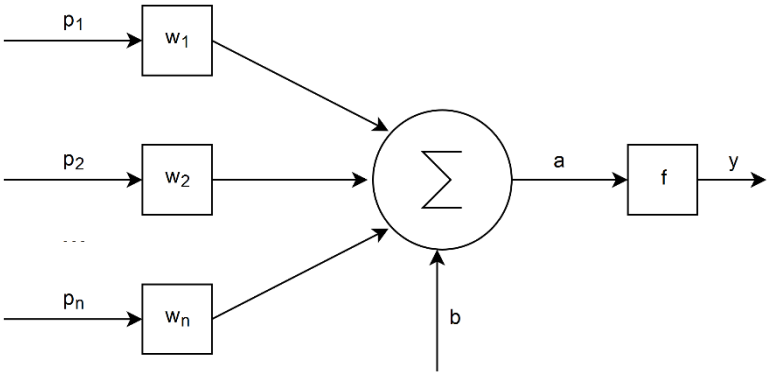
Například pro teplotu 30°C zde dostaneme dvě hodnoty: topit málo na 50% a netopit na 50%. Tyto dvě hodnoty pak projdou agregační funkcí, která podle vlastních pravidel vytvoří jednu fuzzy proměnnou, která je následně převedena na konkrétní numerickou proměnnou.

V reálných systémech figuruje velké množství vstupů a k nim patřících pravidel. Po vytvoření fuzzy systému je důležitá jeho validace a verifikace na testovacích datech. Musí být správně nastaveny hodnoty vah jednotlivých pravidel a limity členských funkcí. To může být provedeno buď manuálně, nebo s využitím dalších metod umělé inteligence, například genetických algoritmů.
Fuzzy logika se používá v rozhodovacích systémech, expertních systémech, řízení výroby, Data miningu, ale i jako součást jiných systémů.

4.6.2 Neuronové sítě

S rostoucím výkonem počítačů se začali výzkumníci věnovat možnosti simulace lidského nebo zvířecího mozku. Složení mozku na buněčné úrovni bylo již známo, a tak byl učiněn pokus o nasimulování funkce mozkových buněk nazývaných neurony. Základem funkčnosti umělých neuronových sítí je skutečnost, že neznáme vnitřní stav sítě a způsob výpočtu. Síť se chová jako černá skříňka, která má pouze vstup a výstup. Samotný výpočet zůstává skryt. Výpočet se skládá ze dvou fází: učení a vlastní výpočet. V učící fázi jsou neuronové síti předložena testovací data a systém se na nich pokouší hledat závislosti. V druhé fázi již síť dostane ostrá data a podle znalostí z první fáze tvoří výstupy [9].

Základem systému je umělý neuron, perceptron.



Obrázek 4: Perceptron

$$y = f(\sum_{i=1}^{n} w_i * p_i + b)$$

Tradiční perceptron má jeden nebo více vstupů, každý vstup je násoben určitou vahou a poté jsou všechny, společně s prahovou hodnotou b, sečteny. Tato hodnota ještě projde přes přenosovou funkci f. Přenosová funkce může být lineární (y=a), skoková (y=1 pro a> x, jinak y=0), případně složitější v závislosti na typu řešeného problému (logsig, tansig, …). Smyslem přenosové funkce je zamezit přenos extrémních hodnot mezi neurony. Upravuje výstupní hodnotu do vhodného intervalu.

Spojením perceptronů do dvourozměrné struktury vznikají neuronové sítě. V síti rozlišujeme vrstvy. Na začátku je X neuronů ve vstupní vrstvě, na které se napojuje Y neuronů v další vrstvě, které se říká skrytá, protože nemáme žádnou možnost ji ovlivnit. Počet skrytých vrstev a neuronů v nich je libovolný, stejně tak počet spojů mezi neurony. Mohou existovat i neurony bez vstupů, které zajišťují, aby výpočet neuvázl v lokálním extrému. Na výstupu je pak výstupní vrstva s jedním nebo více výstupy.

Učení neuronové sítě spočívá ve vhodném nastavení vah vstupů a prahové hodnoty jednotlivých neuronů. Rozlišujeme dva druhy učení [10]:

- S učitelem: algoritmu je předložena množina vstupních dat a množina výstupů. Algoritmus se pak snaží nastavit parametry sítě tak, aby pro každý vstup generoval požadovaný výstup.
- Bez učitele: algoritmus nemá přístup k množině výsledků, má pouze množinu vstupů. Toto omezení nevadí pro některé specifické úlohy, například třídění dat

do skupin podle předem neznámých vlastností. Příkladem je rozpoznávání písma.

Algoritmus učení neuronové sítě se nazývá backpropagation. Nejprve jsou náhodně nastaveny váhy a prahové hodnoty. Poté je podle jednoho učícího vstupu a výstupu spočítán současný výstup sítě a jeho odchylka od očekávané hodnoty. Poté se od výstupu ke vstupu mění hodnoty vah a prahových hodnot. Toto proběhne pro všechny prvky učící množiny. Algoritmus se opakuje, dokud nedosáhne požadované přesnosti. Pokud není učící množina dostatečně velká, tak hrozí, že se algoritmus natrénuje pouze na její řešení a nebude si umět poradit s obecným zadáním.

4.6.3 Genetické algoritmy

Jinými slovy též evoluční algoritmy. Nacházejí svou inspiraci v přírodě ve způsobu, jakým evoluce vytváří specializované organizmy. Při řešení problému je vytvořena množina jedinců, kteří řeší zadaný problém s různou úspěšností. V každé generaci jsou vybrány nejlépe ohodnocení jedinci, ze kterých jsou vytvořeni potomci pro následující generaci. Výpočet končí po nalezení optimálního řešení, nebo zadaném počtu generací.
V tradičním přístupu jsou chromozomy jednotlivých jedinců popsány binárními řetězci, které kódují vlastnosti jednotlivců. Máme-li například systém řízení automobilu, tak pro reprezentaci směru můžeme použít jeden bit pro úhel natočení kol a rychlost můžeme použít pokaždé 8 bitů. Řetězec 1 00110101 00011001 pak vyjadřuje, že automobil jede dopředu, kola má natočena pod úhlem 53 stupňů a jede rychlostí 25 km/h. Tento řetězec reprezentuje reakci systému na nečekanou událost. Je náhodně vygenerováno několik možných variant reakce a poté je podle genetických pravidel vytvořena optimální varianta [9] [10] [13].

Pro výpočet se používá několik základních genetických pravidel

- Selekce: Z generace jedinců jsou vybráni nejlépe ohodnocení jedinci. Jsou však vybíráni s určitou pravděpodobností a tak se do další generace může prosadit i méně schopný jedinec. Tímto se zabrání uvíznutí výpočtu v lokálním extrému. Výběr zajišťuje speciální funkce, která má za úkol i tvorbu párů pro další kroky.

- Křížení: Vybrané dvojice jsou podrobeny křížení. Existuje celá řada heuristických pravidel, jak dosáhnou nejlepšího výsledku. Nekříží se například všechny bity, ale pouze jen některé části, které mohou být definovány pevně nebo se průběžně měnit. Při křížení dochází k výměně části chromozomů. Křížení může probíhat mezi dvěma nebo více jedinci.

- Mutace: Jsou vybrány náhodné bity v chromozomu a je změněna jejich hodnota. Mutace není prováděna příliš často. Zabraňuje uvíznutí v lokálním extrému, případně se objeví vlastnost, ke které by se křížením nedalo dojít.

Nová generace může být ze zcela nových jedinců, nebo se do ní mohou dostat i nejlépe ohodnocení z předchozí generace. Počet jedinců v generaci by měl být konstantní. Populace by neměla být příliš malá ani příliš velká. Malá populace nemusí najít použitelné řešení a velká zase potřebuje delší výpočetní čas. Optimální velikost je přibližně n až 2n, kde n je počet bitů v chromozomu.

Formálně zapsaný algoritmus má následující kroky:

1) Inicializace: Nastavení velikosti populace, vytvoření hodnotící funkce, tvorba jedinců první generace. Tvorba jedinců může být náhodná, nebo podle heuristik, závisí na typu problému, heuristicky generovaní jedinci, kteří jsou si velmi podobní, nemusí být schopni nalézt optimální řešení.
2) Selekce, křížení, mutace.
3) Ohodnocení jedinců v generaci. Pokud byl nalezen dostatečně kvalitní jedinec nebo uběhl nastavený počet generací, tak konec.
4) Tvorba nové generace a pokračování na bodu 2.

5 Využití algoritmů umělé inteligence pro řešení rozvrhovacího problému

V této sekci se budu zabývat některými zajímavými metodami umělé inteligence, které řeší rozvrhovací problém. Ačkoliv i některé metody zmíněné v předchozích kapitolách by se daly považovat umělou inteligenci, teprve v této kapitole jsou plně rozvinuty.

5.1 Genetické algoritmy pro Job shop problém

Na jakém principu fungují genetické algoritmy, bylo obecně vysvětleno v předcházejících kapitolách, zde je popsána konkrétní aplikace na Job shop problém. Přístup ke genetickým algoritmům je velice komplexní a jednotlivé přístupy se mohou velice lišit, zde bude popsán jeden z jednodušších přístupů [14], [15].

Prvním problémem při používání genetických algoritmů je způsob, jakým jsou vytvořeny chromozomy, na kterých jsou pak prováděny genetické operace. Lze použít reprezentaci založenou na operacích, na strojích, na úlohách nebo na seznamu preferencí. Touto poslední možností se teď bude popsána blíže.

J3|n=3,prec|C_{max}

V tomto typu zadání máme 3 úlohy a 3 stroje. Každá úloha se skládá z 3 operací, které každá musí být provedena na jednom ze strojů. Pořadí operací na strojích může být různé pro každou úlohu, ale nesmí se měnit. Lestan a kolektiv navrhli jednoduchý způsob řešení pomocí genetických algoritmů [16].

	Doba zpracování				Pořadí zpracování		
	O1	O2	O3				
J1	2	3	5		M1	M2	M3
J2	3	4	2		M3	M2	M1
J3	7	1	3		M1	M3	M2

Tabulka 15: Zadání rozvrhovacího problému

Pro jednotlivé úlohy vytvoříme řetězce ve formátu *(operace, stroj, trvání),* a dostáváme následující řetězce.

J1: (J1 M1 2) (J1 M2 3) (J1 M3 5)

J2: (J2 M3 3) (J2 M2 4) (J2 M1 2)

J3: (J3 M1 7) (J3 M3 1) (J3 M2 3)

Tento zápis říká, že úloha 1 má být zpracována nejdříve na stroji 1, pak na stroji 2 a nakonec na stroji 3. Jsou uvedeny i doby zpracování jednotlivých operací. Nyní dojde k vytvoření řetězce operací reprezentujícího jeden neoptimální rozvrh. Ze seznamů operací pro každou úlohu jsou náhodně vybírány první položky a umisťovány do jednoho řetězce. Nakonec bychom mohli mít například následující řetězec:

(J3 M1 7) (J1 M1 2) (J2 M3 3) (J1 M2 3) (J2 M2 4) (J1 M3 5) (J3 M3 1) (J3 M2 3) (J2 M1 2)

Tento řetězec reprezentuje jednoho jedince (chromozom). Vygenerovaný řetězec zachovává precedenční omezení. Lze vygenerovat všechna možná řešení, včetně optimálního. V praxi je vygenerováno rozumné množství jedinců, na kterých se pak provádějí operace klasických genetických algoritmů.

Strojově orientovaný Ganttův diagramu reprezentující rozvrh zadaný vytvořeným řetězcem se získá tak, že se z řetězce zleva odebírají operace a umisťují se zleva do Ganttova diagramu na první vhodné místo. Operace, které se nacházejí na začátku řetězce, mají větší prioritu než ty, které se nacházejí na konci. Rozvrh, a s ním i optimalizovaná celková doba zpracování, závisí pouze na pořadí operací zapsaných v řetězci. Ganttův diagram pro zmíněný řetězec vypadá následovně:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
M1	J3							J1		J2							
M2				J2						J1			J3				
M3	J2							J3					J1				

Tabulka 16: Náhodně vygenerovaný rozvrh

Náhodně vygenerovaný rozvrh má celkovou dobu zpracování 17 časových jednotek. Lze předpokládat, že existuje rozvrh s lepšími parametry. Nyní se začneme zabývat genetickými operacemi nad tímto počátečním rozvrhem.

Selekce

Selekce je nejjednodušší z genetických operací. Má za úkol vybrat nejlepší řešení dané generace a přesunout je do generace následující. Aby se zabránilo ztrátě nejlepších řešení, jsou do následující generace přeneseny beze změn i nejlepší řešení současné generace. Funkce, která hodnotí kvalitu řešení, počítá koncový čas poslední operace. Je snaha o minimalizaci této hodnoty.

Výběr řešení se děje pomocí pravděpodobnostní funkce, kde f je účelová funkce, i zkoumané řešení, PS je počet jedinců v populaci a P (i) pravděpodobnost výběru jedince i do další generace.

$$P(i) = \frac{f(i)}{\sum_{k=1}^{PS} f(k)}$$

Křížení

Křížení nebylo v návrhu řešení tohoto problému využito, protože příliš často tvořilo nerealizovatelné rozvrhy.

Mutace

Celá tíže genetických metod tedy zůstává na mutaci.

1) Z řetězce operací je vybrána náhodná operace O.
2) Od operace O se v řetězci postupuje doleva i doprava a hledá se další operace patřící k dané úloze nebo hranice řetězce. Dostaneme tak podřetězec, ve kterém je úloha operace O zastoupena pouze operací O. Tj. pokud O patřilo k J1, tak ve vytvořeném podřetězci je pouze jeden prvek úlohy J1.
3) Operace O je vložena na náhodné místo v podřetězci.
4) Je zjištěna kvalita nového řešení.

Pojďme zpět k našemu příkladu. Řekněme, že v řetězci popisujícím naše úvodní řešení přesuneme operaci *(J3 M1 7) a zkusme rozvrh znovu znázornit v Ganttově grafu.*

(J3 M1 7) *(J1 M1 2) (J2 M3 3) (J1 M2 3) (J2 M2 4) (J1 M3 5) (J3 M3 1) (J3 M2 3) (J2 M1 2)*

=>

(J1 M1 2) (J2 M3 3) (J1 M2 3) ***(J3 M1 7)*** *(J2 M2 4) (J1 M3 5) (J3 M3 1) (J3 M2 3) (J2 M1 2)*

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
M1	J1		J3							J2				
M2			J1			J2						J3		
M3	J2					J1					J3			

Tabulka 17: Rozvrh po mutaci

Při mutaci v tomto konkrétním případě se doba zpracování zkrátila o 3 časové jednotky na 14 časových jednotek.

Na evoluci má vliv několik parametrů:

- Velikost populace: Počet jedinců, kteří se účastní evoluce.
- Počet generací: Kolikrát se celý algoritmus iteruje.
- Tvrdost výběru nové generace: Určuje, která řešení se budou vybrána do další generace. Jestli jen nejlepší, nebo i méně kvalitní. Pokud je tvrdost nastavena příliš přísně, hrozí uváznutí v lokálním extrému, při měkkém nastavení pak hrozí prohledávání celého stavového prostoru.
- Množství změn: Změna může být provedena přesunutím jedné nebo více operací.

Problém byl prezentován na třech strojích a třech úlohách. Pro tak malé problémy je rychlejší použít standardních algoritmů. Pro problém deseti strojů a deseti úloh již stavový prostor naroste do obrovských rozměrů a je rychlejší využít tuto metodu. Autoři tohoto modelu byli schopni dosáhnou po 500 generacích výsledku, který se od optima lišil o méně než 4 %.

5.2 Rozvrhování pomocí neuronových sítí.

Existuje mnoho přístupů k použití umělých neuronových sítí při řešení rozvrhovacího problému. Zde je popsán jeden z nich, konkrétně Hopfieldovy neurnové sítě. Sekvence úloh na daném stroji je reprezentována maticí neuronů *n x n*. V této reprezentaci každý řádek značí jednu úlohu a každý sloupek určuje pozici v rozvrhu. Protože každá úloha může být umístěna v maximálně jedné pozici, výsledná matice má v každém řádku a každém sloupci maximálně jeden aktivovaný neuron [17], [18].

Neuron může nabývat hodnot od 0 do 1. Pokud $n_{i,j}=1$, tak je neuron aktivní, to znamená, že úloha i byla umístěna na pozici j. Pokud $n_{i,j}=0$, neuron je deaktivovaný a na jeho pozici tedy není naplánována žádná úloha. Pokud se hodnota nachází mezi 0 a 1, tak to znamená, že úloha i umístěna na pozici j s určitou pravděpodobností $P(n_{i,j})$.

Pokud máme jeden stroj a tři úlohy, lze matici znázornit následovně:

$$\begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A výsledný rozvrh pak vypadá takto.

	1	2	3	4	5	6
M1	J1		J2			J3

Tabulka 18: Rozvrh

V Job shop problému ale máme více jak jeden stroj. Musíme proto použít větší matici pro reprezentaci všech úloh v neuronové síti. Tato matice má rozměry *mn x (mn+1).* Kromě realizovatelnosti, neuronová síť zároveň hledá optimální řešení podle účelové funkce.

Pro další výpočet je nutné znát hodnotu energetické funkce. Tato funkce popisuje každý stav neuronové sítě. Snažíme se najít její minimum.

$$E = \frac{A}{2}\sum_{x=1}^{mn}\sum_{j=1}^{mn+1}\sum_{j=1,j\neq i}^{mn+1} v_{xi}v_{xj} + \frac{B}{2}\left(\sum_{x=1}^{mn}\sum_{j=1}^{mn+1} v_{xi-mn}\right)^2$$

V energetické funkci je v_{xi} výstupní hodnota neuronu na pozici (x, i). Konstanta A je řádkový snižovací koeficient a B je globální snižovací koeficient. První část výrazu je rovna 0 právě když se n řádku nachází pouze jeden neuron s hodnotou 1. Druhá část výrazu je rovna 0, když v matici je právě *mn* neuronů s hodnotou 1.

Inicializace

Do neuronové sítě jsou nahrány hodnoty tak, aby splňovaly všechny podmínky, tedy aby z nich šel vytvořit realizovatelný rozvrh.

Trénování

Učení v Hopfieldově síti spočívá ve snižování hodnoty energetické funkce stavů, které jsou v síti zaznamenány. Síť funguje jako paměťový systém, který bude konvergovat k naučené hodnotě. Během učícího procesu jsou součty hodnot v řádcích i sloupcích

udržovány na konstantní hodnotě (=1), takže nakonec pouze jeden neuron dosáhne aktivační hodnoty rovno 1, zatímco ostatní se dostanou na 0. Na konci tohoto procesu dostaneme permutační matici vzniklou z původní matice, která reprezentuje realizovatelný rozvrh.

5.3 Rozvrhování pomocí inteligence hejna

Metody umělé inteligence pracující na principu inteligence hejna (swarm inteligence) využívají kolektivní chování mnoha jednoduchých jedinců, kteří mají možnost komunikovat mezi sebou. Výsledek je vytvořen interakcemi jedinců. Každý jedinec se v takovémto systému řídí pomocí jednoduchých pravidel, neexistuje globální řízení. Typickým zástupcem této skupiny algoritmů je optimalizace mraveništěm (ant colony). Tyto algoritmy někdy bývají považovány za odnož genetických algoritmů.

5.3.1 Rozvrhování pomocí mraveniště

Tento algoritmus se inspiruje v mravenčí kolonii, kde jeden mravenec je bezvýznamný, ale ve společnosti svých druhů je schopen vyřešit složité problémy. Teprve spoluprací s ostatními mravenci má jeden mravenec schopnost něco dokázat. Z pozorování živých mravenců bylo zjištěno, že při hledání potravy se mravenec okolo hnízda pohybuje nejdříve chaoticky, náhodně, a po nalezení potravy se snaží dostat zpět do hnízda nejkratší možnou cestou. Při zpáteční cestě klade na trase feromonovou stopu, po které se bude moci vrátit. Ostatní mravenci, kteří se také pohybují náhodně, se po nalezení feromonové stopy rozhodují, jestli se dále pohybovat náhodně, nebo pokračovat po feromonové stopě. Feromonová stopa není věčná, ale s časem se její intenzita snižuje, díky tomu cesty ke zdrojům, které přestaly existovat, postupně zaniknou [19], [20], [21].

Základem pro úspěšné aplikování této metody na nějaký problém je převedení problému na graf. Na začátku algoritmu máme populaci mravenců, kteří vytvářejí řešení problému. Všichni mravenci jsou naráz vypuštěni do grafu a čeká se, až všichni naleznou řešení.

Následující rovnice vyjadřuje pravděpodobnost přechodu mravence z uzlu i do uzlu j.

$$p_{ij}(t) = \frac{\left[T_{ij}(t)\right]^{\alpha} * \left[\frac{1}{d_{ij}}\right]^{\beta}}{\sum_{j \in M}\left[T_{ij}(t)\right]^{\alpha} * \left[\frac{1}{d_{ij}}\right]^{\beta}}$$

T_{ij} – množství feromonu mezi uzly i a j

d_{ij} – odhad vzdálenosti mezi uzly i a j

M – seznam možných cest z uzlu i

α,β – parametry určující relativní váhu feromonů a odhadu zbývající cesty

t – čas

Poté co první skupina mravenců proběhne grafem, je podle určitých pravidel aktualizována feromonová stopa, a mravenci se vyšlou do grafu znovu. Pro aktualizaci hodnoty feromonu existuje mnoho strategií. Feromony může pokládat jen nejlepší mravenec, několik nejlepších nebo nejlepší a ostatní s určitou pravděpodobností. Zde je uveden výpočet množství feromonu v případě, že jej pokládá pouze nejlepší jedinec [19].

$$T_{ij}(t+1) = (1-p) * T_{ij}(t) + p * \Delta T_{ij}(t+1)$$

$$\Delta T_{ij}(t+1) = \begin{cases} \frac{F}{f(x)}, pokud\ nejlepší\ jednec\ použil\ tuto\ cestu \\ 0, jinak \end{cases}$$

p – koeficient odpařování feromonu

F – množství ukládaného feromonu

f(x) – cena cesty nejlepšího mravence

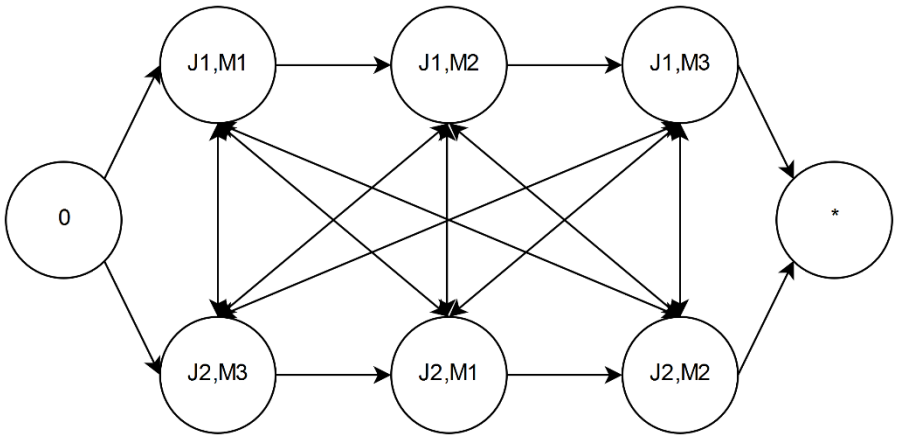
V další iteraci budou mít hrany projité nejlepším mravencem vyšší ohodnocení a největší pravděpodobnost, že se ostatní mravenci vydají také touto cestou. Je však i určitá pravděpodobnost, že se mravenec vydá cesto úplně náhodnou, nebo se v průběhu cesty odchýlí z trasy. Algoritmus končí po předem daném počtu iterací, nebo pokud se dostatečně dlouho nezměnilo nejlepší řešení.

Uvažujme následující zadání **J3|n=2,prec|C_{max}:**

	Doba zpracování				Pořadí zpracování		
	O1	O2	O3				
J1	2	3	5		M1	M2	M3
J2	3	4	2		M3	M1	M2

Tabulka 19: Zadání problému

Pro řešení pomocí mraveniště, je nezbytné reprezentovat problém pomocí grafu.



Obrázek 5: Disjunktní graf

V uzlech grafu jsou jednotlivé operace. J1,M1 značí, že tato část úlohy 1 má být vykonána na stroji M1. Precedenční omezení jsou znázorněna jednosměrnými hranami. Navíc je přidán počáteční a koncový uzel. Přechod z a do tohoto uzlu má nulovou cenu. Každá hrana má definovánu hodnotu feromonu a dobu vykonávání operace. Hrana z i do j má zadánu délku jako dobu trvání operace j. Protože jedou některé hrany obousměrné, je třeba definovat hodnoty pro oba směry [19].

Nyní každý mravenec vyjde z počátečního uzlu, projde přes všechny uzly a skončí v koncovém uzlu. Pro to je využita některá málo náročná standardní metoda, například Greedy search, Backtracking, atd. Aby bylo zajištěno, že každý mravenec bude generovat realizovatelný rozvrh, je vybaven třemi seznamy. V prvním jsou všechny ještě nenavštívené uzly, v druhém jsou uzly, které může navštívit v následujícím kroku, a ve třetím jsou již navštívené uzly.

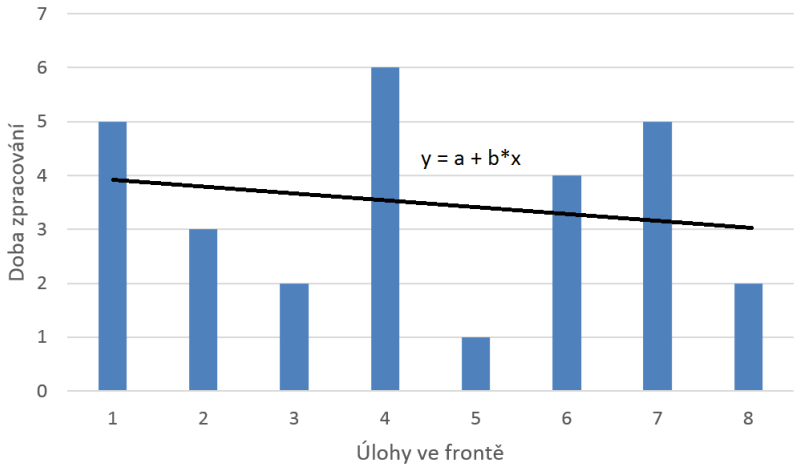
Časová složitost tohoto algoritmu je O (I*A*n*m), kde I je počet iterací, A je počet mravenců, n je počet úloh a m je počet strojů.

Důležitou součástí tohoto algoritmu je pečlivé vyvážení všech parametrů. Parametry α,β, které rozhodují o způsobu pokládání feromonů, je vhodné volit α>2, β>6. Pokud by byly obě hodnoty nastaveny na 0, procházeli by mravenci grafem náhodně. Počet mravenců je vhodné nastavit na m*n. Vypařovací koeficient je dobré nastavit na 0,01. Jeho správné nastavení silně ovlivňuje celkovou úspěšnost algoritmu [20].

Hlavní nevýhodou tohoto algoritmu je jeho časová složitost. Každý z mravenců musí najít řešení celého problému. I přes to, že mravenci o řešení příliš „nepřemýšlí“, tak vzhledem k jejich počtu a množství iterací trvá výpočet příliš dlouho. Důležité je, aby generování řešení jedním mravencem bylo co nejrychlejší i za cenu, že nebude příliš kvalitní [20].

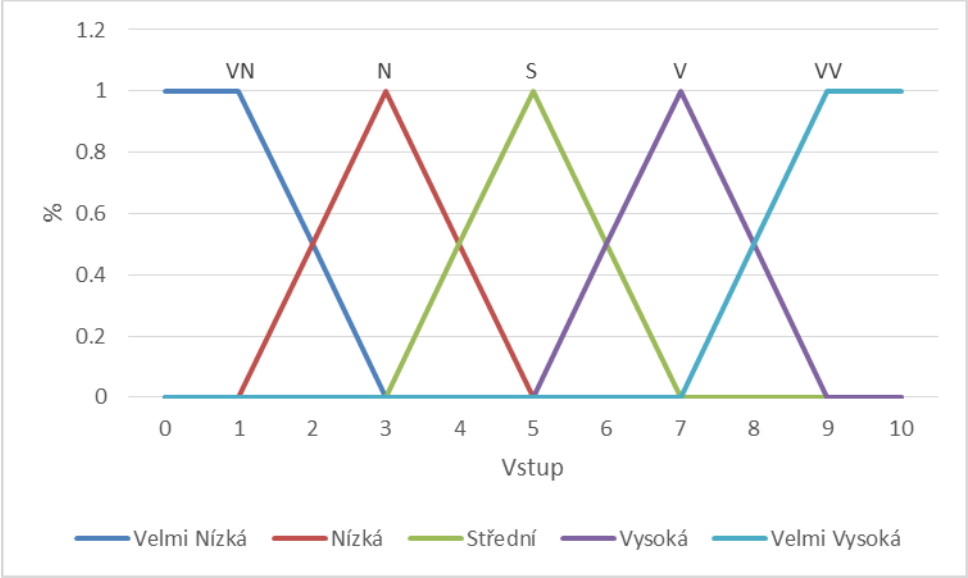
5.4 Fuzzy logika a rozvrhování

Pro některé problémy rozvrhování lze využít vlastností fuzzy logiky[22], [23].
Uvedeme si příklad online rozvrhování s jedním strojem a postupně přicházejícími úlohami. V tomto jednoduchém případě bude na vstupu fronta úloh, které mají různou dobu zpracování. Fuzzy systém bude rozhodovat, jestli z fronty odebírat úlohy podle strategie FIFO nebo LIFO. Pokaždé když stroj dokončí práci na úloze, fuzzy systém zjistí potřebnou strategii.
Jako vstup do fuzzy systému jsou dvě hodnoty popisující směrnici lineárního trendu úloh ve frontě (hodnoty a, b). Je to znázorněno na následujícím obrázku. Pro další výpočet budou hodnoty normalizovány do rozsahu 0 až 10.



Graf 2: Zjištění vstupů pro fuzzy systém

Fuzzifikace proběhne následujícím způsobem:



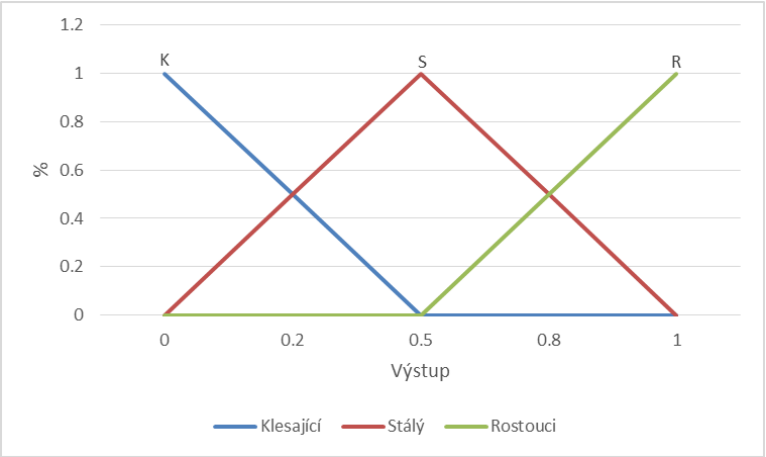
Graf 3: Fuzzifikace vstupů

Na vstupu tedy budeme mít dvě proměnné (a, b), které se převedou na fuzzy proměnné. Nyní se dostaneme k uplatnění pravidel. Máme skupinu pravidel ve tvaru:

IF vstup1 ==N AND vstup2 == S THEN vystup=K

IF vstup1 ==V AND vstup2 == VV THEN vystup=R

...



Graf 4: Defuzzifikace

Po uplatnění všech pravidel dostaneme množinu výstupních parametrů. Vyšlo-li například, že fuzzy pravidla vytvořila výstupní množinu {K, K, S}, tak zprůměrováním jejich středních hodnot dostaneme:

$$výstup = \frac{K + K + S}{3} = \frac{0 + 0 + 0{,}5}{3} \cong 0{,}17$$

Tuto defuzzifikovanou hodnotu je ještě potřeba normalizovat do potřebného rozsahu. Vzhledem k tomu, že očekáváme binární výstup,(FIFO nebo LIFO), použijeme pravidlo

IF výstup <0.5 výstup = FIFO
ELSE výstup = LIFO

Tímto algoritmem lze řídit výrobní online frontu s jedním strojem a přitom minimalizovat maximální zpoždění.

5.5 CSP a rozvrhovací problém

Za Constraint Satisfaction Problem se považuje struktura (X, D, C), kde X je množina proměnných, která může nabývat hodnot z množiny D. Množina C pak obsahuje

jednotlivá omezení. Každé omezení je ve formě $C_i=(V_i,S_i)$, kde V_i je podmnožina X a Si obsahuje povolené hodnoty pro V_i [24], [25].

Mějme zadání se třemi úlohami a třemi stroji. Pokud ho zapíšeme jako CSP, mohlo by vypadat následovně:

X = {st1,st2 …, st9}, kde st_i je startovní čas operace

D = {0, …, 20}, D nabývá hodnot od 0 do 20. Značí možné startovní časy jednotlivých operací, 20 bylo zvoleno jako heuristický odhad maximálního trvání rozvrhu.

C = {$C_{1,2}, C_{2,3}$, … }, kde $C_{i,j} = \{V_{i,j}, S_{i,j}\}$, $V_{i,j}= \{st_i, st_j\}$, $S_{i,j}$={omezení}

Máme dva druhy omezení:

- Precedenční omezení definuje, kdy mohou jednotlivé operace začít. Jsou ve tvaru:
$$st_i + du_i \leq st_j$$
tedy že startovní čas druhé operace musí být větší než součet startovního času a trvání předchozí operace.
- Kapacitní omezení zajišťuje, že na jednom stroji nemohou probíhat v jednom okamžiku dvě operace. Jsou ve tvaru:
$$\min(st_i + du_i, st_j + du_j\ ) \leq \max(st_i, st_j)$$

Takto zadaný CSP se dále řeší třeba pomocí backtrackingu

```
function  CSP_Backtracking (solution, csp)
     IF solution == complete THEN  RETURN solution;
     var = další nepřiřazená proměnná
     FOREACH val IN možné_hodnoty(var)
          IF val je v soulad se solution
               solution.add(var, val)
               result = CSP_Backtracking(solution, csp);
               IF result == OK RETURN result;
               solution.remove(var, val)
     RETURN fail;
```

Algoritmus 3: Backtracking pro CSP

6 Vlastní řešení

V této kapitole je popsán řešený případ a také společnost, pro kterou bylo řešení vytvářeno.

6.1 Popis společnosti

Práce vznikla ze zadání společnosti Pear s.r.o., která se zabývá tvorbou rezervačních systémů pro cestovní kanceláře a ubytovací zařízení [27].

Firma Pear s.r.o. byla založena v roce 1990 pěticí společníků za účelem zjednodušení a vylepšení software pro obchodní procesy v turistice. Pear začal vyvíjet a nabízet software, servis a řešení pro cestování - a to nejen v Česku, ale i v dalších zemích.

Všechen software od firmy Pear je vytvořen na míru pro turistický průmysl. Poskytované služby zahrnují individuální konzultace, hotline podporu a úplné zaškolení. Cílem firmy je efektivní partnerství s našimi zákazníky.

V průběhu let firma zpracovala zakázky pro desítky cestovních kanceláří s tisíci terminály a upevnila si pozici na trhu nejen v oblasti dodávek software, ale i v oblasti služeb a poradenství.

6.1.1 Produkty

Rezervační systém
Rezervační terminál je nativní Windows aplikace, která nabízí nejlepší uživatelský zážitek a největší funkcionalitu. K dispozici je vizuální zobrazení volných a rezervovaných kapacit v autobuse nebo letadle. Rezervační terminál poskytuje veškerou funkcionalitu potřebnou k odbavení zákazníka včetně tisku dokumentů (faktur, cestovní smluv, voucherů, jízdenek, atd.). Historie všech objednávek se uchovává v databázi a je přístupná z ostatní modulů jako jsou Statistika, CRM nebo Účetnictví.

Rezervační systém pro pobočky a podobný systém pro cestovní agentury/agenty má méně funkcí protože je to Internetová aplikace a jako taková má další výhody. Pobočky mohou obvykle používat jak systém pro pobočky, tak systém pro centrálu, je-li to potřeba.

Rezervační systém pro cestovní agentury/agenty a koncové uživatele (zákazníky) je webová aplikace s limitovanou funkčností, která se však jednoduše používá. Může být buďto integrována do webové prezentace firmy nebo může běžet nezávisle na ní, pokud má firma webovou prezentaci od třetí strany, jež nedovoluje její integraci.

Kalkulace

WinCK poskytuje jednotný a flexibilní kalkulační model založený na definovaných smlouvách a velkoobchodních cenách. Výstupem jsou koncové ceny. Uživatelské rozhraní integruje tabulkový editor podobný Microsoft Excelu. Velkoobchodní ceny se obyčejně sestávají ze třech částí - zahraniční náklady, tuzemské náklady a zisková marže. Výsledkem jsou ceníky, které obsahují informace pro automatické zpracování DPH a sestavování individuálních objednávek. Umožněna je současná existence více měn v jednom ceníku. Rovněž mohou vedle sebe existovat různé verze jednoho ceníku, tj. jedna pro cestovní agentura, další pro koncové klienty, atd.

WinCK také umožňuje tvořit kalkulace přímo v MS Excelu, který načítá data potřebná pro tvorbu kalkulací z databáze a po nakalkulování nadefinovanou kalkulaci uloží zpět do databáze. Slevy mohou být v kalkulacích nadefinovány několika způsoby: pevné slevy, procentní slevy, individuální slevy. Navíc lze tvořit pravidla, kdy jsou slevy vázány na datum rezervace nebo odjezdu.

Redakční systém

Redakční systém shromažďuje informace o hotelech a destinacích. Informace mohou být textového charakteru, obrázky, mapy nebo ikony. Některé jsou definovány systémem, další si může volně dodefinovat cestovní kancelář. Informace uložené v redakčním systému slouží jako podklady pro dynamické webové stránky s propracovanými vyhledávacími schopnostmi. Na těchto webových stránkách mohou být pochopitelně zobrazeny i informace z ostatních modulů, které spolu sdílí společnou databázi. Data z redakčního systému mohou být sdíleny s ostatními cestovními kancelářemi nebo agenturami prostřednictvím webových služeb a XML.

CRM

Modul CRM obstarává práci s informacemi o zákaznících. Tam, kde je potřebný, jsou jeho části přístupné z ostatních modulů, třeba z rezervačního terminálu. Každý zákazník

má historii smluv. Základní hledán mezi zákazníky vyhledává podle téměř jakéhokoliv systémem nadefinovaného kritéria - atributu zákazníka (křestní jméno, příjmení, věk, adresa, atd.). Další atributy mohou být nadefinovány a přiřazeny zákazníkovi. Jejich počet a obsah není nijak limitován, mohou to být pole jako zájmy, preference, handikapy, Komplexní vyhledávání je další silnou stránkou CRM. Takto lze hledat podle detailů v objednávkách, které zákazníci učinili, destinací, do kterých cestovali, částky, kterou utratily a mnoha dalších.

Statis

Modul Statis zahrnuje obecné a univerzální statistiky. Každá statistika může být filtrována a tříděna dle různých kritérií jako jsou datum odjezdu, příjezdu nebo rezervace, zákazník nebo rezervační číslo. Modul obsahuje několik desítek statistik: statistiky rezervací a zákazníků, roomingů, obsazeností v letadlech a autobusech dle půdorysů, odjezdových řádů, odletů, … Dále existují propracované analytické nástroje pokrývající analýzu nákladů, analýzu cash-flow, analýzu výnosů, analýzu destinací a produktů.

Kromě výše uvedených statistik lze vytvořit specializované statistiky a reporty na zakázku přímo podle potřeba zákazníka. Nabízí se i exporty do jiných formátů. Pro analytické nástroje jiných firem nabízíme datové kostky a datové mraky.

Účetnictví

V modulu účetnictví je zahrnuta veškerá funkcionalita, kterou by mohly vyžadovat společnosti podnikající v cestovním ruchu.

Dále společnost nabízí tvorbu webových prezentací přesně podle požadavků zákazníka

6.1.2 Porterova analýza pěti konkurenčních sil

Stávající konkurence

Na trhu působí několik málo společností nabízejících takto specializovaný produkt. Velcí hráči na poli informačních systémů, jako například SAP, nejsou našimi konkurenty. Náš

produkt je mnohem lépe připraven a hlavně je prodáván za nižší cenu. Naše společnost je zaměřena na malé a střední cestovní kanceláře. Pokud se některá z nich pokusí zavést SAP, obvykle to končí po několika letech krachem nebo nasazením jiného systému. Slabým místem naší firmy je marketing. Ačkoliv nabízíme kvalitnější produkt, často se zákazník rozhodne pro jiné řešení, i když nenabízí tolik možností jako to naše.

Nová konkurence

Hrozbu nové konkurence nelze vyloučit, protože však nabízíme velice specifický produkt na malém trhu s již existující konkurencí, je nepříliš pravděpodobné, že by se nová společnost byla schopna prosadit. Existuje možnost, že by se velcí dodavatelé informačních systémů pokusili prosadit i v tomto oboru, nemají však mnohaleté zkušenosti s potřebami tohoto odvětví.

Vliv dodavatelů

Společnost nemá dodavatele v tradičním slova smyslu. Nejdůležitější z nich je asi Microsoft a jeho vývojové nástroje, zejména Visual Studio a serverová platforma. Pokud by výrazným způsobem zkazil novou generaci svých produktů a zamezil používání starších verzí, byl by to pro společnost velký problém. Některé činnosti firmy jsou outsorcovány, zejména správa lokální sítě a starost o servery a ostatní hardware. Všechny vztahy jsou zajištěny kvalitními smlouvami a dosud nebyl jediný problém. Dodání kancelářských potřeb není potřeba řešit.

Vliv odběratelů

Odběratelé mají velkou vyjednávací sílu, často se opožďují platby za dodané služby, případně jsou podmiňovány dalšími pracemi. Ačkoliv to není podle podepsaných smluv, vedení se bojí s tím cokoliv děla ze strachu ze ztráty zákazníka.

Substituční produkty

Neexistují substituční produkty. Zákazník, pokud vyžaduje určitou úroveň správy svých dat, nemá na výběr než využít některý z nabízených informačních systémů. I v případě velmi malých cestovních kanceláří je velmi nepraktické řešit vše pomocí Excelu.

6.1.3 SLEPT analýza

Sociální faktory

Fungování společnosti je přímo závislé na úspěšnosti svých zákazníků, jimiž jsou cestovní kanceláře a agentury. Všechny faktory, které ovlivňují výběr zájezdů jednotlivými osobami se tak týkají i této společnosti. Existuje vliv stáří osob, kdy se mění preference vybíraných dovolených. Stárnutí populace není velkým rizikem, protože i starší občané budou chtít na dovolenou, mění se jen cílové destinace. Na to musí vhodně zareagovat cestovní kanceláře vhodnou nabídkou. S postupně rostoucí životní úrovní lze předpokládat, že občané budou chtít cestovat stále častěji.

Legislativní faktory

Pro úspěch firmy je kritické, aby splňovala všechny legislativní požadavky a byla schopna je implementovat do svého informačního systému. Externí zaměstnanec zaměřený na právo musí být schopen správně interpretovat nové zákony a vysvětlit je ostatním zaměstnanců, kteří navrhnou jejich zapracování do systému. Většinou se v poslední době jednalo o jednoduché změny ve výši DPH. Složitější však je sledování legislativy všech států, kde působí naši zákazníci a navrhnout proveditelná opatření. V poslední době se například jednalo o způsob placení daní (spotřební, ze mzdy,…) při jízdě autobusu přes několik států. V turistickém ruchu lze předpokládat snahu o podporu ze strany státu. Největším nebezpečím je amatérismus politiků, kteří dočasně schválí špatný zákon.

Ekonomické faktory

Rizikem je stav celé společnosti, jestli se ekonomika nachází v krizi a zákazníci šetří a omezují své výdaje na dovolenou. Dále kurz české koruny k euru a americkému dolaru. Zájezdy do západní Evropy cestovní kanceláře platí v eurech, jinde po světě v dolarech. Přílišné výkyvy mohou značně zdražit celkovou cenu zájezdů a omezit tím i počet cestovatelů. Míra inflace je v posledních letech nízká, což pozitivně ovlivňuje rozhodování zákazníků.

Politické faktory

Existuje určitá podpora turismu ze strany státu. Většinou to ale nemá větší dopad na hospodaření cestovních kanceláří. V minulých letech byly oblíbené zájezdy do severní Afriky, zejména Tunis, Egypt, Maroko, ale i Turecko. Vzhledem k proběhnuvším revolucím a následné nestabilitě výrazně klesl počet turistů ochotných do těchto zemí cestovat, což mělo za následek i krach několika cestovních kanceláří. Není předpoklad, že by se situace v nejbližší době zlepšila a tak se cestovní kanceláře přeorientovávají na jiné destinace. Zkoušejí nalákat turisty do zemí jako je Bulharsko a Rumunsko. V nadcházející letní sezóně bude zvýšená poptávka po dovolených ve střední Evropě, Itálii, Španělsku. V Řecku, které se neumí vypořádat s migranty, výrazně klesla poptávka po dovolené na východních ostrovech, v západní části neklesla tak výrazně. Pokud bude růst počet migrantů v západní Evropě, očekává se pokles poptávky po poznávacích zájezdech do daných lokalit.

Technologické faktory

V oblasti IT dochází k výrazným pokrokům téměř neustále. Společnost sleduje vývoj nových technologií a snaží se je využít ve svých produktech. V nejbližší době plánuje zavést cloudové verze svých programů. Nabízené informační a účetní programy nejsou zaměřeny na každého uživatele, ale pouze na úzce specializovanou skupinu vyškolených pracovníků. Udržení kroku s posledními technologiemi není kritické, důležité je pouze absolutní splnění legislativních požadavků. Společnost například stále prodává produkt, jehož design vznikl před více jak deseti lety. Funkčnost je ale vylepšována několikrát ročně.

6.1.4 7s analýza

Strategie

Každá společnost, která se plánuje dlouhodobě udržet na trhu, potřebuje několik strategií. Nejdůležitější z nich je dlouhodobá strategie. Společnost Pear s.r.o. je menším podnikem, který je vlastněn dvěma společníky, kteří jsou zároveň i jednatelé. Nemá tedy písemně zaznamenanou strategii v žádné podnikové dokumentaci. Hlavní cíle firmy Pear by se daly popsat jako snaha o upevnění své pozice na trhu, získání dobrého jména a udržení a zvýšení svého podíl na trhu s informačními systémy. Společnost působí na trhu od roku

1994 a tak má oproti konkurenci výhodu v dlouhodobém působení. Mezi potencionálními zákazníky je společnost známá a nemá tedy potřebu výrazných investic do reklamy. Získávání nových zákazníků pak probíhá aktivním marketingem.

Struktura

Společnost je vlastněna dvěma společníky, kteří jsou zároveň i jednateli. Kromě nich jsou ve společnosti ještě zaměstnáni 4 zaměstnanci a proměnlivý počet brigádníků. Dále zde vystupuje ještě neupřesněný počet externích zaměstnanců. Vzhledem k velikosti podniku není potřeba, aby byly pravomoci striktně oddělené, a proto jednotliví zaměstnanci vykonávají širokou škálu činností.

Systém řízení

Ve společnosti je zaveden informační systém vlastní výroby, který eviduje přijaté zakázky a sleduje práci na nich. Většina komunikace mezi zaměstnanci probíhá ústně, komunikace se zákazníky bývá přes email a je archivována. Strategická rozhodnutí přijímají oba majitelé společně. Oba průběžně kontrolují plnění cílů zaměstnanci, ekonomickou výtěžnost v jednotlivých obdobích a spokojenost zákazníků. Je kladen důraz na osobní zodpovědnost každého zaměstnance za svou práci.

Styl manažerské práce

Řízení této společnosti lze charakterizovat jako velmi demokratické. Při řešení všech problémů, panuje na poradách či brainstormingu otevřená diskuze a všichni zúčastnění se snaží dohodnout na společném závěru. V případě neshody při řešení má rozhodující slovo ředitel společnosti. Úkoly stanovuje management, kdy oba majitelé jsou ale zároveň zaměstnanci, způsob jejich plnění je již týmová práce, kdy se většinou již nepožaduje vlastní úvaha zaměstnanců.

Spolupracovníci

Je důležitá podrobná znalost legislativy okolo turistického ruchu. Je využíváno odborných znalostí několika externích pracovníků, včetně právníků a daňových poradců. Všichni zaměstnanci musí být profesionály ve svém oboru, je kladen velký důraz na jejich schopnosti a pečlivost práce. Pracovníci jsou motivováni ke zvyšování svého vzdělání a

schopností. Odměňování zaměstnanců probíhá fixně, každoročně jsou zde ale příplatky, které reflektují osobní nasazení jednotlivých pracovníků. Další možností přivýdělku je práce na malých úkolech mimo pracovní dobu.

Sdílené hodnoty

Sdílené hodnoty firmy by šlo popsat jako snahu o naplňování vize a poslání firmy. U této společnosti tak lze mluvit o snahu o dodávání vysoce kvalitních informačních systémů s úzkým zaměřením. Dále se společnost snaží vycházet vstříc osobním potřebám svých zaměstnanců, což je možné pozorovat ve volnější pracovní době. Podnik věří, že spokojený zaměstnanec je produktivní zaměstnanec, proto pro dosažení tohoto stavu dělá maximum (pracovní podmínky, benefity, vzdělání, kariérní růst). Společnost doufá, že tato spokojenost se pozitivně odráží nejen na pracovní výkonnosti, ale i na vztahu se zákazníky a dodavateli.

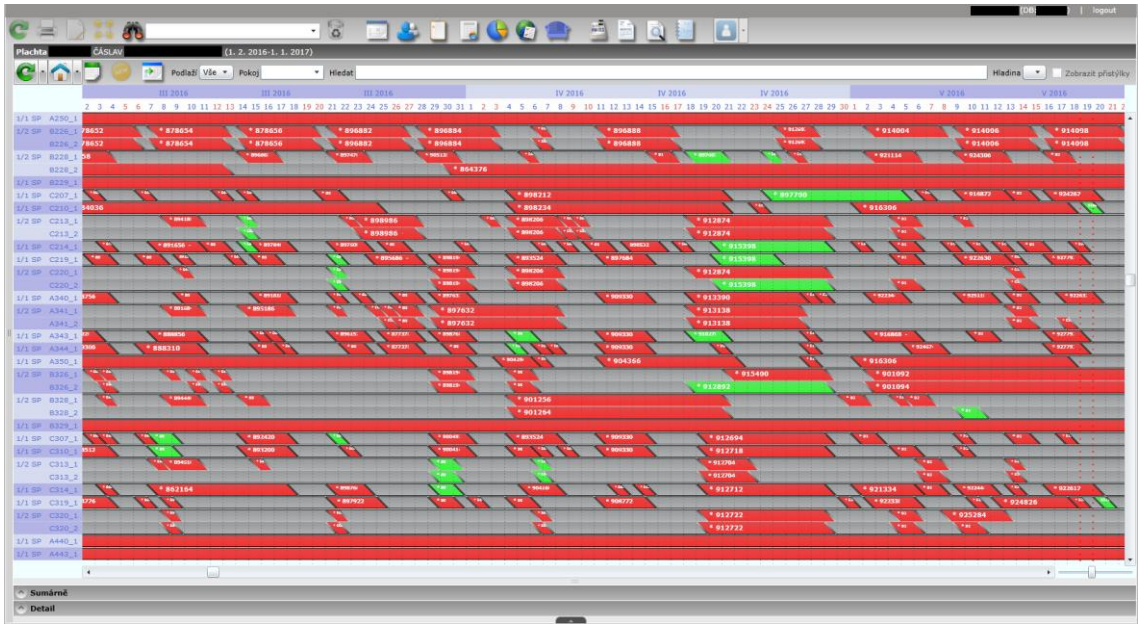
Schopnosti

Je kladen důraz na osobní rozvoj zaměstnanců a vzdělávání. Na trhu se objevují neustále nové možnosti i omezení, se kterými se společnost musí umět vypořádat. Je zcela nezbytné získávat nové poznatky a znalosti všemi dostupnými cestami. Společnost působí na trhu již dlouhý čas a tak si již vybudovala detailní znalosti o všech vlastnostech trhu.

6.2 Popis problému

Cílem této práce je vytvořit aplikaci na řešení rozvrhovacího problému při umisťování rezervací v ubytovacích zařízeních.

V současné době není ve společnosti použit žádný software na optimalizaci rozvrhu, protože společnost byla zaměřena na větší cestovní kanceláře, které ani tuto funkčnost nevyžadovaly. Pro velké hotely, které se zabývají ubytováváním turistických zájezdů, nemá rozvrhování ve smyslu této práce velký smysl. Cestovní kanceláře si u hotelů objednávají celou kapacitu a tu samy posléze dělí podle délky jednotlivých zájezdů. Zájezdy mají neměnnou délku (7, 10, 14 dní) a v hotelu pak dochází pouze ke střídání rezervací známých délek a není potřeba řešit nezarezervovanou kapacitu.



Obrázek 6: Rezervační systém

Na obrázku 6 je znázorněno současné řešení použité ve společnosti, do kterého bude jako nová funkčnost zakomponován výsledek této diplomové práce. Zde se konkrétně jedná o zařízení typu školních kolejí, které rozprodává volnou kapacitu i pro jiné potřeby. Na řádcích jsou jednotlivé pokoje a k nim přiřazené rezervace. Dny jsou znázorněny ve sloupcích. Lichoběžníkový tvar rezervací je dán tím, že standardně pobyt začíná i končí v poledne. Barvami je označen stav jednotlivých rezervací, zde pouze zaplaceno/nezaplaceno, ale existují i jiné stavy. Šedou jsou znázorněny volné termíny.

V poslední době se společnost rozhodla nabízet produkty zaměřené i do jiných oborů než jsou cestovní kanceláře. Snaží se nabízet software hotelům, které nabízejí ubytování přímo koncovým zákazníkům. Protože každý jejich zákazník si může zarezervovat libovolnou délku termínu a s libovolným začátkem. Tímto způsobem postupně vznikají rozvrhy, které jsou „řídké“, mezi jednotlivými rezervacemi se tvoří krátké sekvence (1-3 dny), u kterých je velká pravděpodobnost, že si je nikdo nezarezervuje. Je tedy nutné jednotlivé rezervace poskládat tak, aby byly eliminovány krátké sekvence volného místa a zároveň se prodloužily dlouhé volné sekvence.

V Grahamově konvenci by šlo tento problém popsat jako $P_m| r_j, d_j |f$, kde f je funkce popsaná na následujících stránkách.

Účelová funkce

Jak popsat účelovou funkci nebylo na počátku úplně zřejmé. Byl sdělen jen neurčitý požadavek „přeskládej to tak, aby tam bylo co nejvíc volného místa“. Jak ale matematicky vyjádřit požadavek na „co nejvíc volného místa“? Bylo nutné nalézt funkci, která bude preferovat delší volné sekvence a zároveň bude jednoduchá na výpočet. Po bližší analýze problému jsem se rozhodl pro součet kvadrátů délek volného místa.

Na následujícím grafu je znázorněn neoptimalizovaný rozvrh, částečně optimalizovaný rozvrh a i jeho optimální varianta.

	1	2	3		1	2	3		1	2	3
P1	R1				R1	R2			R1	R2	R3
P2		R2		=>				=>			
P3			R3				R3				

Tabulka 20: Optimalizace rozvrhu

Máme zde 3 pokoje a 3 rezervace. Nejdříve je každá rezervace ve vlastním pokoji, posléze se je podařilo přemístit do jednoho pokoje. Pro ohodnocení kvality nalezeného řešení je použit následující výraz:

$$score = \sum_{i=1}^{PP} \sum_{j=1}^{PV_i} \left(délka(sekvence_{i.j})^2 \right)$$

Kde PP je počet pokojů, PV je počet volných sekvencí v každém pokoji. Sekvencí se rozumí nepřerušená posloupnost nezarezervovaných dat. Na uvedeném příkladu dostáváme:

pro první pokoj je $\text{score}_1 = 2^2 = 4$

pro druhý pokoj je $\text{score}_2 = 1^2 + 1^2 = 2$

a pro třetí pokoj je $\text{score}_3 = 2^2 = 4$

Celkové score neoptimalizované varianty je tedy $4 + 2 + 4 = 10$.

Pro částečně optimalizovanou variantu:

pro první pokoj je $\text{score}_1 = 1^2 = 1$

pro druhý pokoj je $\text{score}_2 = 3^2 = 9$

a pro třetí pokoj je $\text{score}_3 = 2^2 = 4$

Celkové score částečně optimalizované varianty je tedy 14.

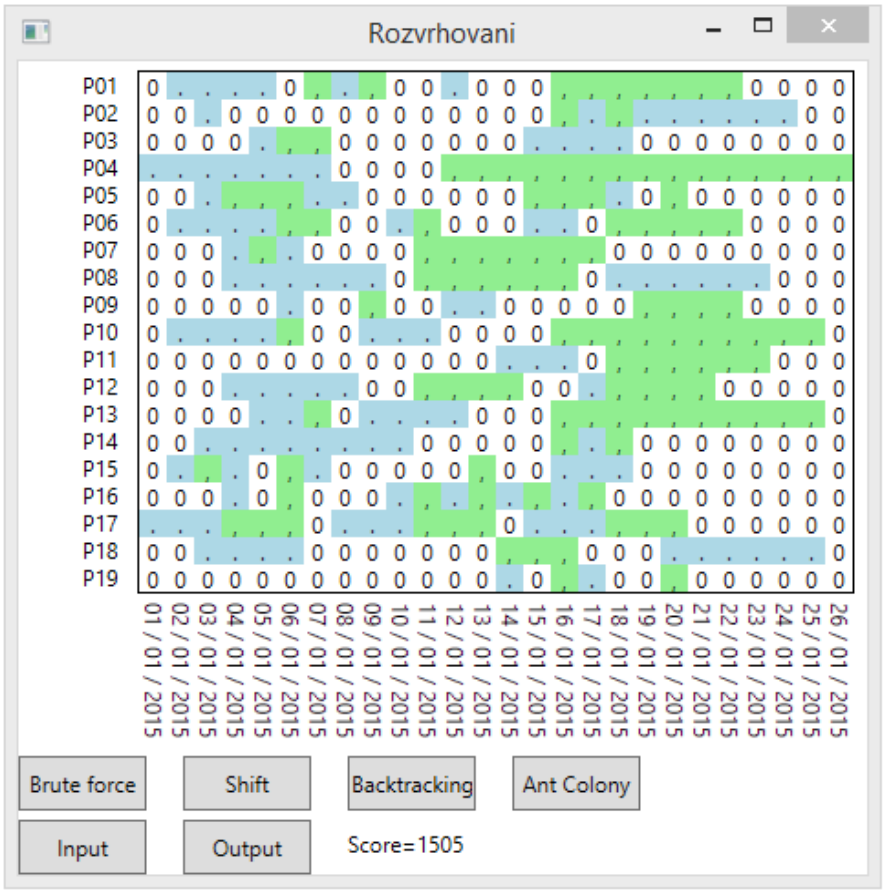
A nakonec pro plně optimalizovanou variantu dostaneme $score = 0^2 + 3^2 + 3^2 = 18$

Je vidět, že se zlepšováním řešení dochází ke zvyšování hodnoty účelové funkce, je tedy použitelná pro potřeby dalších výpočtů. Nutným předpokladem je, že všechny pokoje jsou stejné (počet lůžek, vybavení, poloha, …), tzn., že jsou jakkoli zaměnitelné a rezervace mezi nimi mohu libovolně přesunovat. To lze zajistit tak, že metody řešení popsané v dalších kapitolách budou vždy volány nad množinou stejných pokojů.

6.3 Aplikace

Aplikace byla vytvořena v prostředí Visual Studio v jazyce C#. Používá podobný design jako současné řešení využívané ve společnosti. Na následujícím obrázku je znázorněn počáteční stav vytvořené aplikace. Na řádcích jsou jednotlivé pokoje, ve sloupcích jsou dny. Nulou je označena volná kapacita. Jednotlivé rezervace jsou od sebe odděleny barevně. Pod pracovní plochou se nachází ovládací tlačítka. Jsou zde 4 metody rozvrhování a tlačítka pro řízení vstupu a výstupu. Po startu aplikace se na pracovní ploše hned ukáže cvičný rozvrh, na kterém lze demonstrovat funkčnost aplikace. Pro serióznější práci je nutné využít tlačítek pro vstup a výstup. Jako vstup slouží XML soubor, jehož struktura je popsána dále.

Uživatelské rozhraní bylo vytvořeno jen pro potřeby demonstrace funkčnosti řešení. V budoucnu bude tato aplikace ve formě knihovny zapracována do současné firemní aplikace.



Obrázek 7: Okno aplikace

Formát vstupních a výstupních dat

Jako vstup i výstup vytvořené aplikace slouží XML soubor. Na každém řádku je jedna rezervace. Má definován jedinečný identifikátor, identifikátor pokoje, start a konec. Pro případ, že je potřeba předat prázdný pokoj, je vložena „speciální rezervace“ s id=0.

```
<rows>
  <row id="175" room="P1" start="2015-01-02T21:45:00" end="2015-01-05T04:30:00" />
  <row id="69" room="P2"  start="2015-01-07T07:30:00" end="2015-01-07T19:30:00" />
 <row id="0" room="P3"  start="0" end="0" />
</rows>
```

Algoritmus 4: Struktura XML souboru

Vnitřní reprezentace dat

Pro vnitřní potřeby programu je vytvořen seznam struktur pro reprezentaci jednotlivých rezervací.

```
public class res
  {
      public int id { get; set; } // id rezervace
      public string room { get; set; } //který pokoj
      public Int64 start { get; set; } // začátek rezervace
      public Int64 end { get; set; } // konec rezervace
     …
  }
```

Algoritmus 5: Vnitřní reprezentace rezervace

Ukázalo se, že použití datového typu *DateTime* pro čas je nevhodné, protože vnitřně je reprezentován strukturou obsahující jednotlivé složky data. V algoritmech dochází k velkému počtu porovnání mezi dvěma daty, což prodlužovalo dobu výpočtu. Lepší je reprezentace jako Int64 do kterého je standardní čas převeden následovně:

$$Čas = rok * 100000000 + měsíc * 1000000 + den * 10000 + hodina * 100 + minuta$$

Řetězec "2015-01-02T21:45:00" je tak převeden na 201501022145. Zpětný převod lze jednoduše provést pomocí dělení a operace modulo. Porovnání dvou proměnných typu *DateTime* je více než šest krát delší než porovnání dvou proměnných typu Int64. Pokud je počet porovnání v řádu desetitisíců, je již úspora znatelná.

Pro fungování rozvrhovacích metod bylo nutno vytvořit několik podpůrných funkcí, které jsou mezi metodami sdíleny. Jedná se o funkce zobrazení rozvrhu, výpočet score, manipulaci s rezervacemi, tvorba kopií, atd. Mnohé z těchto funkcí jsou pouze několikařádkové, ale zvyšují přehlednost kódu. Velké množství krátkých funkcí se ale ukázalo jako nevýhoda, pokud docházelo k velkému počtu volání dotyčných funkcí. Režijní náklady na volání funkce byly příliš vysoké, proto bylo v několika případech upuštěno od samostatné funkce.

Pro metody je občas výhodné používat dva seznamy rezervací. V jednom jsou již umístěné rezervace a v jednom jsou čekající na umístění. Abych mohl používat jen jeden

seznam, tak rezervace, která ještě není umístěná má proměnnou room=““. Při tomto přístupu by se mohlo stát, že se ztratí informace o dostupných pokojích, a tak jsou do rozvrhu zařazeny speciální záznamy s id=0, které označují pokoj bez přiřazených rezervací.

V následující kapitole budou popsány metody, které byly využity při tvorbě řešení.

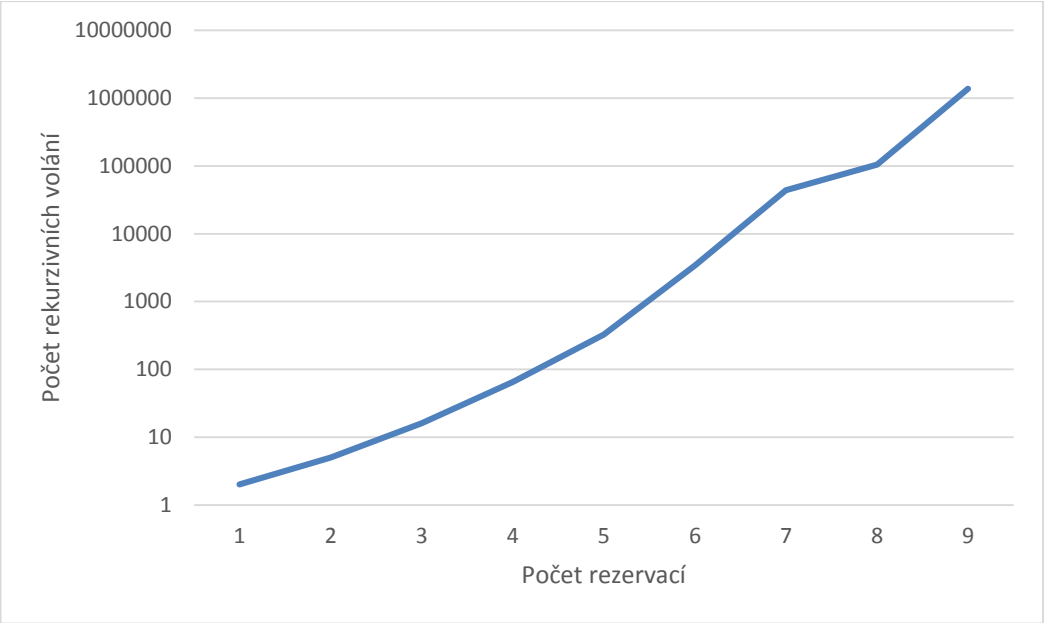
6.3.1 Bruteforce

První zkoušená metoda bylo Bruteforce, tedy nalezení všech možných řešení a vybrání nejlepšího. Byla naprogramována jen jako doplnění ostatních metod a k ověření na malém vzorku dat, že ostatní metody skutečně poskytují optimální řešení. Byla použita rekurzivní varianta algoritmu. Na vstupu má seznam pokojů a seznam všech rezervací.

```
Function Bruteforce(rooms_sch, reserv_list)
     IF reserv_list je prázdný
          Zjisti score vytvořeného rozvrhu v rooms_sch
          RETURN nalezené řešení  rooms_sch + score
     Vezmi první rezervaci z reserv_list a nalezni možné
     kandidátní pozice v rooms_sch
     IF neexistuje možné umístění RETURN FAIL
     FOREACH kandidátní pozice
          Vyjmi rezervaci z reserv_list a vlož ji do
          rooms_sch na kandidátní pozici
          výsledky[]=Bruteforce(rooms_sch, reserv_list)
     RETURN nejlepší řešení z pole výsledky[]
```

Algoritmus 6: Bruteforce

Složitost tohoto algoritmu je exponenciální, proto je nepoužitelný již pro 15 rezervací a 10 pokojů. Byl učiněn pokus o napsání více vláknové verze tohoto algoritmu, ve které se rekurzivně volá hlavní metoda v novém vlákně. To vede k vyčerpání systémových prostředků a k pádu aplikace.



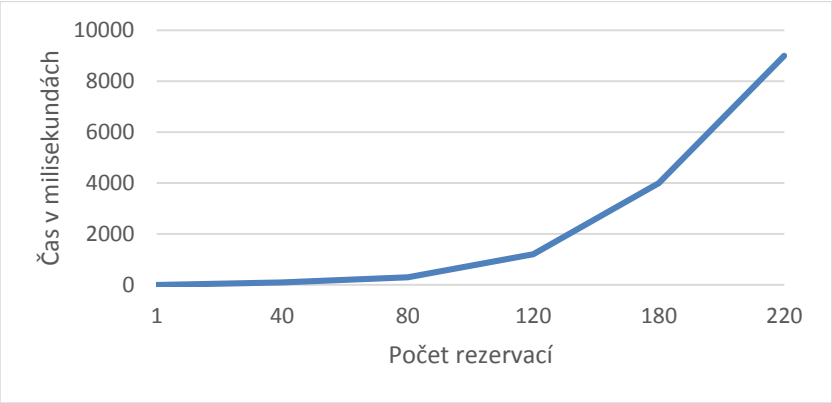
Graf 5: Závislost počtu rekurzí na počtu rezervací

6.3.2 Shift

Druhá zkoušená metoda byla pracovně nazvaná Shift. Jedná se o naivní heuristiku, která se snaží pohybovat jednotlivými rezervacemi a zjišťuje, jestli se nepodaří najít lepší řešení. Na vstupu má již vytvořený rozvrh a snaží se jej optimalizovat.

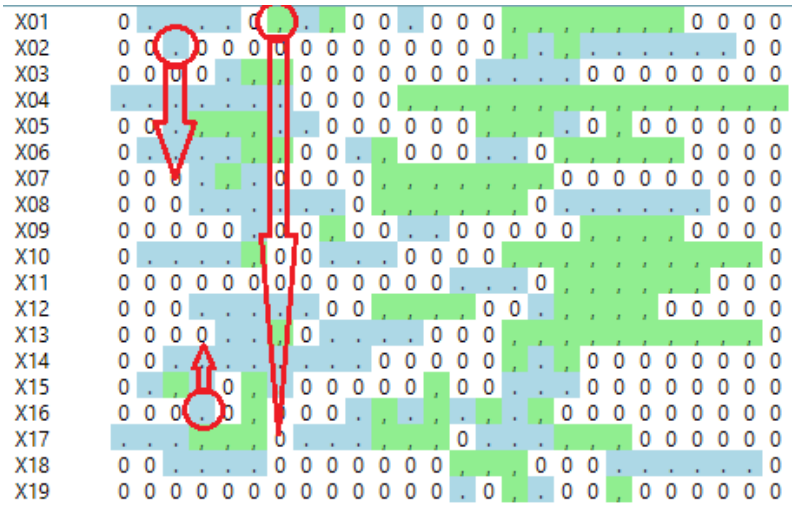
```
Shift (rozvrh)
     WHILE TRUE
          FOREACH rezervace IN rozvrh
               Nalezni volné pozice pro rezervaci
               IF existují pozice
                    Vyber pozici s nejlepším score
                    Přesuň rezervaci na novou pozici
                    BREAK
          IF nebyla přesunuta žádná rezervace
               RETURN rozvrh
```

Algoritmus 7: Shift



Graf 6: Závislost výpočetního času na počtu rezervací

Algoritmus pracuje s kubickou časovou složitostí. Algoritmus bude postupovat po rezervacích seřazených podle počátečního data, když nalezne rezervaci, kterou lze přesunout a zároveň zvýšit score, tak ji přesune. Poté algoritmus pokračuje znovu od první rezervace. Na obrázku je znázorněn přesun prvních dvou vhodných rezervací. Algoritmus není optimální a je příliš časově náročný.



Obrázek 8: Příklad přesunu rezervací

Metoda postupuje po sloupcích a hledá první rezervaci, se kterou by šlo pohnout a zvýšilo by to kvalitu rozvrhu.

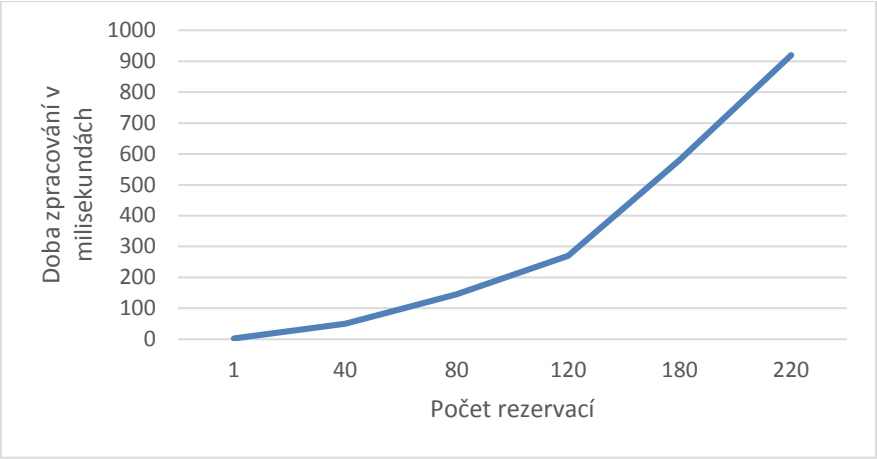
6.3.3 Backtracking

Tradiční provedení backtrackingu uzpůsobené na použití při rozvrhování. Do struktury popisující jednu rezervaci byl přidán parametr určující, kterou z množných pozic vybrat v současném kroku. Pro každou rezervaci se zjišťuje seznam povolených pozic, seřazených od nejvýhodnější. Pokud je takový seznam prázdný, musí se algoritmus o krok vrátit a zkusit alternativní pozici předcházející rezervace.

```
Backtracking (rozvrh)
     FOR (r=0; r< počet rezervací v rozvrhu; r++)
               Nastav rezervace[r] jako nepřiřazeno
     WHILE (i < počet rezervací)
          Najdi kandidátní pozice pro rezervace[i]
          Seřaď pozice od nejlepší // na 0 bude nejlepší
          IF (rezervace[i].index > počet kandidátů – 1)
               IF (i == 0) RETURN FAIL
               i--;
               CONTINUE;
          Vlož rezervace[i] na pozici rezervace[i].index
          rezervace[i].index ++
          i++
     RETURN OK
```

Algoritmus 8: Backtracking

Backtracking se ukázal jako velmi efektivní pro řešení tohoto typu problémů. Ze všech zkoušených algoritmů má nejkratší dobu běhu. Zásluhu na tom má i navržená heuristika. Na začátku jsou rezervace seřazeny podle začátku a u rezervací začínajících v jeden den ještě podle doby trvání. Tímto způsobem se omezí počet nutných návratů.



Graf 7: Doba zpracování v závislosti na počtu rezervací (Backtracking)

6.3.4 Ant colony optimalization

V tradičním pojetí jde o nalezení cesty při průchodu mravence grafem. Mravenec se pohybuje s určitou pravděpodobností buď náhodně, nebo po předem označených cestách. Při tomto pojetí se mravenec "nepohybuje", ale "vyplňuje" rozvrh.
Na začátku se stejně jako u backtrackingu vyprázdní rozvrh a naplní seznam rezervací. Jednotliví mravenci pak konstruují řešení, které je v každé iteraci ohodnoceno. Každý mravenec pracuje samostatně na svém řešení.
Předpokladem je existence mapy feromonů. Mapa je ve formě seznamu trojic, kde je uvedeno identifikační číslo pokoje, číslo rezervace a množství feromonu pro tuto variantu. V mapě jsou všechny kombinace pokoje a rezervace, jakou zvolil některý z mravenců a zároveň hodnota feromonu určující její prioritu.

Na začátku je vytvořeno N vláken, jedno pro každého mravence. Každý mravenec se pohybuje pomocí upraveného Backtrackingu. Oproti variantě uvedené dříve, nemá při rozhodování o umístění rezervace informaci o score.

Při použití tohoto algoritmu je nutné dbát na správné nastavení vnitřních parametrů algoritmu. Jde zejména o počet mravenců, počet iterací, množství ukládaného feromonu a množství odpařovaného feromonu. Roli při tvorbě algoritmu má i strategie ukládání feromonu. Může jej ukládat nejlepší mravenec, ale i několik nejlepších s různě odstupňovaným množstvím.

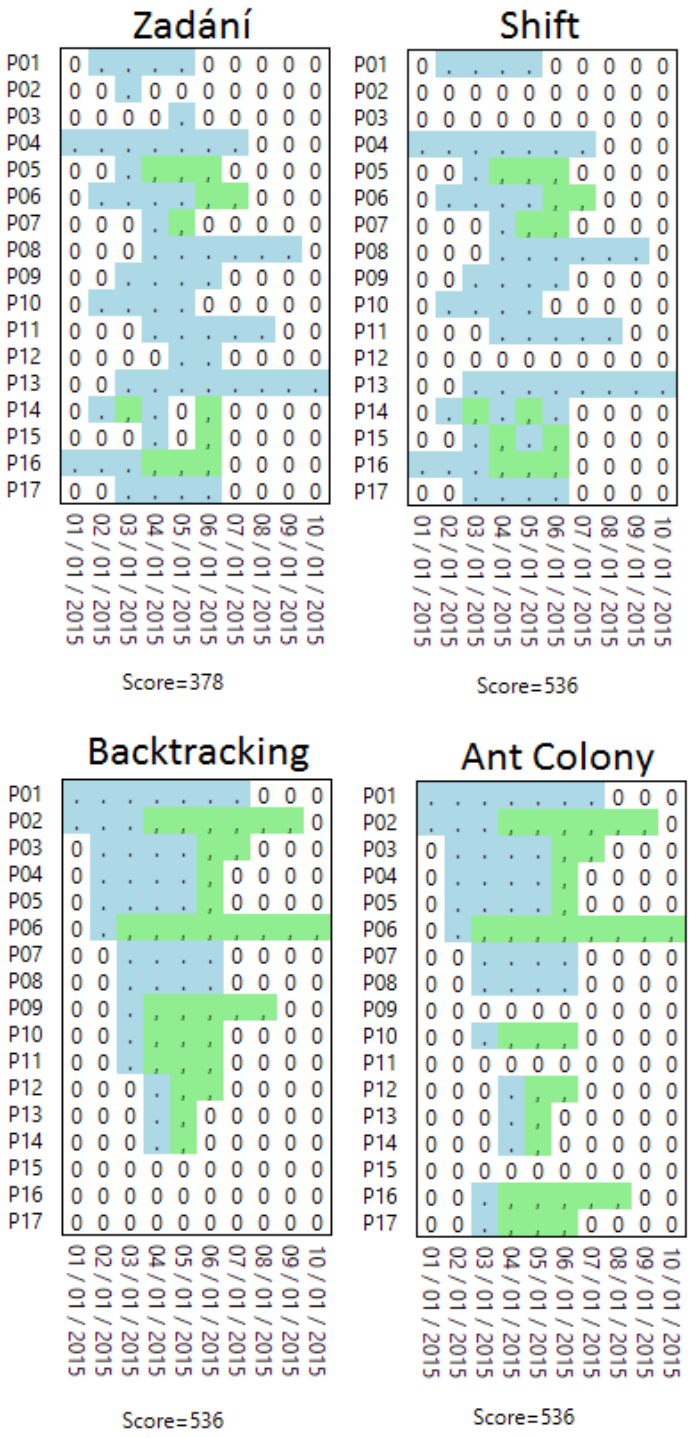
```
Start_Ants()
     FOR i IN počet_iterací
          FOR j IN počet_mravenců
               Vytvoř vlákno s jedním mravencem
               Výsledky[j] = Ant()
          Vyber nejlepšího z Výsledky[]
          a uprav podle něj mapu feromonů
     RETURN nejlepší řešení z Výsledky[]



Ant(rozvrh)
     FOR (r=0; r< počet rezervací v rozvrhu; r++)
          Nastav rezervace[r] jako nepřiřazeno
     WHILE (i < počet rezervací)
          Najdi kandidátní pozice pro rezervace[i]
          IF (rezervace[i].index > počet kandidátů – 1)
               IF (i == 0) RETURN FAIL
               i--;
               CONTINUE;
          Vlož rezervace[i] na pozici rezervace[i].index
          rezervace[i].index ++
          i++
     RETURN OK
```

Algoritmus 9: Ant Colony Optimalization

Teorie tohoto algoritmu vypadala nadějně, bohužel v praxi se ukázalo, že pro dobrý výsledek je nutné použít velké množství mravenců a velký počet iterací a ani poté není zaručeno nalezení optimálního řešení. I v literatuře se uvádí, že další vývoj tohoto algoritmu není perspektivní.

Na obrázku 9 je porovnání výsledků vytvořených různými algoritmy. Shodou okolností dosáhly všechny metody výsledku se stejným ohodnocením. Ale to bylo možné jen díky tomu, že zadání obsahovalo nízký počet rezervací a mělo relativně vysoký počet volných

míst. Pro větší rozvrhy metoda Shift ani Ant Colony nedosahují optimálního řešení, i když se mu velmi blíží.



Obrázek 9: Porovnání různých metod rozvrhování

6.4 Přínos navrhnutého řešení

Vytvořený program zjednodušuje práci pracovníka s rezervačním systémem. V současnosti je ve formě samostatné aplikace, kterou ale lze jednoduše zakomponovat do již existujícího rezervačního systému. Umožňuje tvorbu optimalizovaného rozvrhu s efektivitou jen stěží dosažitelnou některým z pracovníků. Při tvorbě rozvrhu pro delší časová období je vhodnější použít počítač s vyšším výpočetním výkonem.

Pro společnost je vytvořený program konkurenční výhodou, která je velmi důležitá na omezeném trhu informačních systémů pro cestovní kanceláře. Ekonomický přínos pro společnost je tedy spíše ve schopnosti udržet si současné zákazníky, případně získat nějaké nové.

Ekonomický přínos vytvořeného řešení pro zákazníka není jednoduché přímo vyčíslit. Zákazníkům přináší možnost zvýšit obsazenost svých pobytových zařízení a tím i zvýšit příjem financí. Dále přináší úsporu času pracovníka pracujícího s rezervačním systémem. Práce s rozvrhem představuje pouze část pracovníkovy náplně práce, je však jednou z jejích nejobtížnějších součástí.

Náklady na zavedení vytvořeného programu do systému budou relativně nízké. Pokud se uvažuje, že tato práce byla vytvořena zdarma, je potřeba počítat pouze s náklady na propojení. Ty představuje především plat odpovědného pracovníka. Odhad doby práce na propojení aplikací je přibližně 3 dny. Samotné propojení je otázka několika desítek minut, ale následuje delší doba testování. Náklady by tedy mohly být přibližně 10 000 Kč.

Závěr

Cílem práce bylo vytvoření optimalizačního systému pro rozvrhovací problém při obsazování hotelových pokojů rezervacemi. Při tvorbě byly použity prvky umělé inteligence. Systém byl vyvíjen pro společnost Pear s.r.o. Je ale obecně využitelný v kterékoli společnosti, která řeší obsazenost pokojů ve svých ubytovacích zařízeních. Program lze použít i pro speciální případ plánování autobusové dopravy, kdy mají autobusy jeden výchozí bod, do kterého se vždy vracejí (MHD). Dochází k jednoduché abstrakci, že jeden autobus je „pokoj“ a jedna jízda autobus je „rezervace pokoje“. Je však nutné zdvojit délku rezervace kvůli cestě tam a zpět.

V úvodní části této práce byla provedena analýza různých druhů rozvrhovacích problémů a způsobů jejich řešení. V druhé části jsou prozkoumány metody umělé inteligence. Ve třetí části jsou spojeny poznatky z předchozích dvou a jsou zde popsány metody využívající prvků umělé inteligence při řešení rozvrhovacího problému.

V praktické části byla nejprve popsána společnost, pro kterou bude mít tato práce skutečný význam. Následně je provedena Porterova analýza konkurenčních sil a SLEPT analýza. Dále je popsán problém který tato práce řeší.

Rozvrhovací problém při obsazování hotelových pokojů rezervacemi byl řešen několika různými způsoby. Prvním z algoritmů byla tradiční metoda Bruteforce, tedy nalezení všech realizovatelných rozvrhů a vybrání nejlepšího. Tato metoda byla použita pouze pro potvrzení výsledků ostatních algoritmů nad malým vzorkem dat. Mezi dalšími algoritmy bych zmínil Backtracking a Ant Colony Optimalization.

Podařilo se úspěšně implementovat vybrané algoritmy. Společnost může v budoucnu jednoduchým způsobem propojit vytvořené řešení se současným rezervačním systémem a zvýšit tím svojí konkurenceschopnost.

Seznam použité literatury

[1] JOSPEH Y-T. LEUNG. *Handbook of Scheduling Algorithms, Models, and Performance*. London: Chapman & Hall/CRC, 2004. ISBN 0203489802.

[2] MICHAEL L. PINEDO. *Scheduling theory, algorithms, and systems*. 4th ed. New York: Springer, 2012. ISBN 1461423619.

[3] BRUCKER, Peter. Scheduling algorithms. 5th ed. Berlin: Springer, 2007. ISBN 978-35-406-9515-8.

[4] ČERNÝ, Jakub. *Základní grafové algoritmy* [online]. 2010 [cit. 2016-05-21]. Dostupné z: http://kam.mff.cuni.cz/~kuba/ka/ka.pdf

[5] SKIENA, Steven S. The algorithm design manual. 2nd ed. London: Springer, 2008. ISBN 978-18-480-0070-4.

[6] ALHARKAN, Ibrahim. *Algorithms for Sequencing and Scheduling* [online]. 2011 [cit. 2016-05-21]. Dostupné z: http://faculty.ksu.edu.sa/ialharkan/IE428/ Algorithms_for_Sequencing_and_Scheduling1.pdf

[7] RUDOVÁ, Hana. *Plánování úloh na jednom stroji* [online]. [cit. 2016-05-21]. Dostupné z: http://www.fi.muni.cz/~hanka/rozvrhovani/prusvitky/sedma.pdf

[8] YAMADA, Takeshi a Ryohei NAKANO. *Job-shop scheduling* [online]. 1997 [cit. 2016-05-19]. Dostupné z: www.kecl.ntt.co.jp/as/members/yamada/galbk.pdf

[9] DOSTÁL, P. Pokročilé metody rozhodování v podnikatelství a veřejné správě. Brno: CERM, 2012. 718 s. ISBN 978-80-7204-798-7.

[10] DOSTÁL, P. Advanced Decision Making in Business and Public Services. Brno: CERM, 2011. 168 s. ISBN 978-80-7204-747-5.

[11] MAŘÍK, V., O. ŠTĚPÁNKOVÁ a J. LAŽANSKÝ. Umělá inteligence. Praha: ACADEMIA, 2013. 2473 s. ISBN 978-80-200-2276-9.

[12] ZBOŘIL, František. *Základy umělé inteligence*. Brno, 2006.

[13] MAN, TANG, KWONG. *Genetic Algorithms Concepts and Designs*. London: Springer London, 1999. ISBN 9781447105770.

[14] WALL, Matthew. *A Genetic Algorithm for Resource-Constrained Scheduling* [online]. 1996 [cit. 2016-05-22]. Dostupné z: http://lancet.mit.edu/~mbwall/phd/thesis/thesis.pdf

[15] WERNER, Frank. *Genetic Algorithms for shop Scheduling Problems* [online]. 2011 [cit. 2016-05-22]. Dostupné z: http://mat.uab.cat/~alseda/MasterOpt/p11-31.pdf

[16] LESTAN, BREZOCNIK, BUCHMEISTER, BALIC. *Solving the Job shopscheduling problem with simple genetic algorithm* [online]. 2009 [cit. 2016-05-22]. Dostupné z: http://ww.ijsimm.com/Full_Papers/Fulltext2009/text8-4_197-205.pdf

[17] WANG, XU, CHEN. *A Novel Hopfield Neural Network Approach to Job-Shop Scheduling Problems* [online]. 2004 [cit. 2016-05-22]. Dostupné z: http://www.wseas.us/e-library/conferences/miami2004/papers/484-399.pdf

[18] SABUNCUOGLU, GURGUN. *A neural network model for scheduling problems* [online]. 1994 [cit. 2016-05-22]. Dostupné z: http://yoksis.bilkent.edu.tr/pdf/files/1237.pdf

[19] VENTRESCA a OMBUKI. *Ant Colony Optimization for Job Shop Scheduling Problem* [online]. 2004 [cit. 2016-05-22]. Dostupné z: https://www.cosc.brocku.ca/sites/all/files/downloads/research/cs0404.pdf

[20] FLORES, GOMEZ a BAUTISTA. *Ant Colony Optimization for Job Shop Scheduling Problem* [online]. 2013 [cit. 2016-05-22]. Dostupné z: https://arxiv.org/ftp/arxiv/papers/1309/1309.5110.pdf

[21] RITCHIE, Graham. *Static Multi-processor Scheduling with Ant Colony Optimisation & Local Search* [online]. Edinburgh, 2003 [cit. 2016-05-19]. Dostupné z: https://www.inf.ed.ac.uk/publications/thesis/online/IM030077.pdf

[22] HASSAN, CHOWDHURY a MASUD. *A Fuzzy - Multicritaria Based Approach for Job Sequencing* [online]. 2012 [cit. 2016-05-22]. Dostupné z: https://globaljournals.org/GJRE_Volume12/6-A-Fuzzy-Multicritaria-Based-Approach.pdf

[23] RAMKUMAR, TAMILARASI a DEVI. *Multi Criteria Job Shop Schedule Using Fuzzy Logic Control for Multiple Machines Multiple Jobs* [online]. 2010 [cit. 2016-05-22]. Dostupné z: https://globaljournals.org/GJRE_Volume12/6-A-Fuzzy-Multicritaria-Based-Approach.pdf

[24] PANG a GOODWIN. *Aplication of CSP Algorithms to Job Shop Scheduling Problems* [online]. 1997 [cit. 2016-05-22]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.9563 &rep=rep1&type=pdf

[25] SADEH a FOX. *Variable and value ordering heuristics for the Job shop Scheduling constraint satisfaction problem* [online]. 1998 [cit. 2016-05-22]. Dostupné z: https://www.ri.cmu.edu/pub_files/pub1/ sadeh_koniecpol_norman_1995_1/sadeh_koniecpol_norman_1995_1.pdf

[26] FERA, Marcello a Fabio FRUGGIERO. *Production Scheduling Approaches for Operations Management* [online]. 2015 [cit. 2016-05-19]. Dostupné z: http://cdn.intechopen.com/pdfs-wm/43001.pdf

[27] *PEAR s.r.o.* [online]. Brno, 2016 [cit. 2016-05-22]. Dostupné z: http://www.pear.cz/

Seznam tabulek, obrázků, grafů

Seznam příloh

Příloha A: CD s touto prací a zdrojovými kódy