UNIVERZITA PARDUBICE
FAKULTA EKONOMICKO-SPRÁVNÍ

BAKALÁŘSKÁ PRÁCE

2009 **Barbora KAVÁLKOVÁ**

Univerzita Pardubice
Fakulta ekonomicko-správní

Možnosti řešení nedeterministicky polynomiálních úloh

Barbora Kaválková

Bakalářská práce

2009

Univerzita Pardubice
Fakulta ekonomicko-správní
Ústav systémového inženýrství a informatiky
Akademický rok: **2008/2009**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Barbora KAVÁLKOVÁ**

Studijní program: **B6209 Systémové inženýrství a informatika**

Studijní obor: **Regionální a informační management**

Název tématu: **Možnosti řešení NP úloh**

Zásady pro vypracování:

· Teorie výpočetní složitosti algoritmů a úloh · Nedeterministicky polynomiální problém · Turingův stroj · Úlohy třídy P a NP

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

MAŘÍK, Vladimír, ŠTĚPÁNKOVÁ, Olga, LAŽANSKÝ, Jiří. Umělá inteligence (1). 1. vyd. Praha : Academia, 2000. 264 s. ISBN 80-200-0496-3.

COVENEY, Peter, HIGHFIELD, Roger. Šíp času : cesta vědou za rozluštěním největší záhady lidstva. 1. vyd. Ostrava : Oldag, 1995. 472 s. ISBN 80-85954-08-7.

PELIKÁN, Jan. Diskrétní modely v operačním výzkumu. 1. vyd. 1 : Professional Publishing, 2001. 150 s. ISBN 80-86419-17-7.

DEVLIN, Keith. Problémy pro třetí tisíciletí : Sedm největších nevyřešených otázek matematiky. Luboš Pick. [s.l.] : Argo, Dokořán, 2005. 272 s. Aliter; sv. 9788073630164. ISBN 80-7363-016-8.

Vedoucí bakalářské práce: **Ing. Jan Panuš, Ph.D.**
Ústav systémového inženýrství a informatiky

Datum zadání bakalářské práce: **6. října 2008**

Termín odevzdání bakalářské práce: **24. srpna 2009**

L.S.

Ing. Jana Myšková
děkanka

doc. Ing. Jiří Křupka, Ph.D.
vedoucí ústavu

V Pardubicích dne 6. října 2008

Prohlašuji:

Tuto práci jsem vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.

Byla jsem seznámena s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně Univerzity Pardubice.

V Pardubicích dne 20. srpna 2009 Barbora Kaválková

Poděkování

Na tomto místě bych ráda poděkovala vedoucímu bakalářské práce Ing. Janu Panušovi za jeho konzultace, trpělivost a ochotu. Dále bych chtěla poděkovat doc. RNDr. Evě Milkové, Ph.D. za její ochotu, odborné rady a konzultace při tvorbě práce.

Souhrn

Tato práce je věnována především možnosti řešení nedeterministicky polynomiálních úloh. Částečně se zaměřuje na historii, definice, algoritmy a na varianty řešení. Dále se práce zabývá problémem batohu, problémem obchodního cestujícího, barvením grafu a RSA problémem, některé tyto pojmy na praktických příkladech vysvětluje.

Klíčová slova

NP, úlohy NP, NP-úplné úlohy, problém P versus NP, polynomiální, úplnost, problém batohu, hamiltonovská kružnice a graf, metoda obchodního cestujícího, barvení grafů, algoritmy

Title

The possibilities of solving non-deterministic polynomial problems

Abstract

This bachelor thesis is especially dedicated to the possibility of solving non-deterministic polynomial problems. It is particularly focused on history, definition, algoritms and varieties of solving. The thesis also deals with explaining of the travelling salesman problem.

Keywords

NP problems, NP-complete problems, travelling salesman method

OBSAH

Úvod

Téma mé bakalářské práce se týká problému možnosti řešení nedeterministicky polynomiálních úloh. Jde o jeden z nejdůležitějších problémů v informatice. Jeho důležitost a význam si uvědomuje stále více matematiků. Jedná se v podstatě o otázku rovnosti dvou tříd, P a NP (viz. dále), neboli o slavném problému, označovaném "P = NP?", který se týká složitosti výpočtů. Vznikl v teoretické informatice v době, kdy se klasičtí matematici na tuto disciplínu dívali s despektem jako na obor, kde se dají dobře prodávat lacino získané výsledky.

Za cíl své práce jsem si zvolila charakteristiku nedeterministicky polynomiálních úloh. V návaznosti na to jsem se rozhodla podrobně popsat a na praktických příkladech ukázat problém batohu, obchodního cestujícího, barvení grafu a některé vybrané algoritmy.

V první části bakalářské práce se budu zabývat teorií nedeterministicky polynomiálních úloh, polynomiální versus nedeterministicky polynomiální zahrnující historii, definice a též jeho modifikaci. Dále popisuji heuristické algoritmy, které se používají při řešení NP úloh. Na závěr se budu věnovat problému obchodního cestujícího, kde rozsáhleji popisuji jeho princip a řešení. Tyto dva uvedené problémy patří mezi typické NP - úplné úlohy.

V její druhé části se věnuji popisu jednotlivých NP – úplných problémů a některým praktickým aplikacím.

Třetí část práce je zaměřena na některé algoritmy. Je ukázán přístup k jejich popisu a řešení.

V závěru bakalářské práce provádím konečné shrnutí poznatků a cílů této práce.

1 Úvod do NP problému

Ze všech problémů milénia je odpověď na hádanku vztahu P versus NP nejnáchylnější k tomu, aby ji vyřešil „neznámý amatér“ – tedy někdo bez matematické průpravy, možná někdo velice mladý a matematické komunitě neznámý. Všechny ostatní problémy jsou pohřbeny hluboko pod masou hodně těžké matematiky. To není případ problému P versus NP, který se zabývá tím, jak efektivně mohou počítače v dostatečně krátkém čase řešit určitý typ úloh. Nejenže je poměrně snadné pochopit, v čem problém spočívá, ale je zapotřebí jeden nový dobrý nápad. Právě na dobré nápady není třeba hlubokých znalostí, stačí fantazie.

1.1 Historie

Pro většinu lidí představují počítače prostředek komunikace, který slouží k posílání a přijímání vzkazů a k získávání informací z internetu. V tom ale jejich původní idea nespočívá. Počítače byly prvotně stvořeny k tomu, pro provádění aritmetických operací. Vznikly na základě předcházejícího dlouhodobého teoretického výzkumu matematické koncepce „vypočitatelnosti“. Ve třicátých letech minulého století začalo nezávisle několik matematiků studovat otázky „vypočitatelnosti“. Řešili, co je vlastně výpočet a které funkce v oboru přirozených čísel lze vypočítat. Samotný výzkum probíhal dříve, než byla dostupná moderní výpočetní technika. Původní zájem matematiků nebyl motivován žádnou myšlenkou na provádění nějakých výpočtů, ať už ručně nebo výpočetní technikou. Otázka „vypočitatelnosti“ byla zkoumána výhradně pro vlastní matematickou zajímavost bez širších návazností.

Zájem o „vypočitatelnost“ vznikl po zásadním objevu rakouského matematika Kurta Gödela v roce 1931. Podnětem k tomu byl inspirativní seznam nevyřešených matematických problémů německého matematika Davida Hilberta z roku 1900 [5].

Hilbert byl ovlivněn úspěchy tehdy převládajícího axiomatického přístupu k matematice. Budování matematického odvětví pomocí axiomatického přístupu znamenalo zformulovat několik základních tvrzení – „axiomů“ – a z nich logicky odvodit všechny ostatní poznatky daného oboru. Otázka „pravdivosti“ nějakého výroku se redukuje na otázku jeho „vyvoditelnosti z axiomů“. Tato původní Thalesova představa o matematice (cca 600 před naším letopočtem) se stala základem matematického učení starověkých Řeků [5].

Úspěch této metody závisí na tom, jak si budeme při formulování axiomů vést. Aby se matematické tvrzení mohlo kvalifikovat jako přijatelný axiom, mělo by být poměrně

jednoduché a dostatečně fundamentální, abychom je mohli považovat za zřejmě pravdivé. Dosáhnutí této podmínky nemusí být vždy lehké. Například Eukleidovy axiomy geometrie vedly dlouhodobě k diskusím o pravdivosti jednoho z jeho axiomů, takzvaného postulátu o rovnoběžkách. Kritici se domnívali, že tento výrok je příliš složitý na to, aby mohl být axiomem. Proto bylo podniknuto mnoho pokusů o jeho odvození z jednodušších předpokladů. Vznikaly všelijaké „neeukleidovské geometrie“ popisující postulát rovnoběžnosti jinými axiomy. Při sepisování axiomů může být velice obtížné nalézt všechna „samozřejmá“ tvrzení, která se do seznamu hodí. Eukleides vynechal některé předpoklady, které jsou pro vývoj geometrie nezbytné a které ve své knize využíval. Až do konce devatenáctého století, kdy začal Hilbert tuto problematiku důkladně zkoumat, nebyl úplný seznam axiomů sepsán.

Po úspěšném zformulování vhodné soustavy axiomů eukleidovské geometrie navrhl Hilbert, toto učinit v kterémkoli jiném matematickém oboru. Hledání axiomů pro několik různých matematických odvětví později vešlo ve známost jako takzvaný Hilbertův program.

Hilbert věřil tomu, že v kterémkoli matematickém oboru bude možné sepsat soustavu základních tvrzení – axiomů, ze kterých mohou být v podstatě všechny poznatky onoho oboru odvozeny. V roce 1931 zásadně otřásl matematickým světem objev Kurta Gödela dokládající, že Hilbertův předpoklad je nepravdivý. Dokázal, že v každé části matematiky, bude bez ohledu na to, kolik tvrzení přijmeme za axiomy, vždycky existovat alespoň jedno pravdivé tvrzení, které se z těchto axiomů vyvodit nedá. Jde o Gödelův princip neúplnosti. Platí tedy, že systém axiomů bude vždy neúplný. Axiomy nikdy nebudou stačit k tomu, aby z nich plynula všechna pravdivá tvrzení. V matematice, stejně jako v životě, budou existovat nepolapitelná zrnka pravdy.

Toto tvrzení navázalo na metodu převedení otázek dokazatelnosti na ekvivalentní otázky „vypočitatelnosti“ některých funkcí na přirozených číslech. Gödel vyvinul formální teorii koncepce „vypočitatelné funkce“ a dokázal, že v každém axiomatickém systému bude vždy existovat funkce, kterou nelze v rámci tohoto systému vypočítat.

Na Gödelovu práci navázali další matematikové, kteří začali vyšetřovat „vypočitatelnost“. Jejich cílem bylo zjistit, které funkce jsou vypočitatelné a které nikoli. Jednalo se ale pouze o teoretické studium toho, jak by v principu mohly být výpočty prováděny.

Ohlédneme-li se zpět, spatříme fascinující výsledky Stephena Kleena, Alana Turinga a dalších, kteří konstatovali teoretickou možnost konstrukce programovatelných počítačů (strojů), které mohou být naprogramovány k provádění rozličných výpočtů. Teoretické myšlenky, vypracované v třicátých a počátkem čtyřicátých let dvacátého století měly hlavní roli v raném vývoji počítačů v letech čtyřicátých a padesátých.

Matematici ztratili o toto své duchovní dítě zájem již po dokončení teorií, na jejímž základě ve světe vznikly a byly naprogramovány první počítače. Přestože vývoj počítačových technologií vyžadoval určité matematické schopnosti a využíval matematické značení, tak většina práce již nebyla ve své podstatě matematická. Drtivá většina matematiků pracovala (stejně jako pracují dnes) pouze na problémech, které žádné příliš těžké výpočty nevyžadují. Vznik počítačů je neovlivnil tak, jako třeba fyziky nebo chemiky. Situace se podstatně změnila koncem osmdesátých let minulého století. Sofistikované počítačové systémy byly využity k práci v algebře, matematické analýze a v dalších odvětvích symbolické matematiky.

Přesto se našlo několik matematiků, kteří hledali využití počítačů a jejich technologií na řešení matematických problémů. Díky tomu vznikla řada nových matematických odvětví, jako např. numerická analýza, počítačová teorie čísel a teorie dynamických systémů. Tento raný výzkum ukázal nutnost vzniku teoretické informatiky, jako samostatné vědecké disciplíny. Dále navazoval vznik jejích matematických podoborů, například soubor teorií - programovací jazyky, algoritmy, databáze, umělé inteligence a výpočetní složitosti. V poslední citované disciplíně nacházíme třetí problém tisíciletí. Osobou, která se nejvíce ze všech zasloužila o ustanovení této otázky jako veledůležitého problému teoretické informatiky, byl mladý Američan jménem Stephen Cook [4], [5].

1.2 Stephen Cook

„Stephen Cook se narodil v Buffalu ve státě New York v roce 1939. Po absolvování Michiganské univerzity (obor elektroinženýrství) nastoupil v roce 1961 na Harvard jako doktorand v matematice. Jeho záměrem bylo studium algebry. Zásadní vliv na něho měl logik Hao Wanga, působící na katedře aplikovaných věd na Harvardu. Wang pracoval v novém oboru automatického dokazování vět, podoboru stejně nového odvětví, které John McCarthy ambiciózně nazval umělou inteligencí.

Během studií na Harvardu se Cook seznámil s důležitou prací Michaela Rabina o teorii složitosti. Ta analyzuje výpočetní procesy, aby zjistila, jak efektivně je lze provádět.

V roce 1966 dokončil doktorskou práci. Získal místo na Kalifornské univerzitě v Berkeley a následně v roce 1970 přestoupil na Torontskou univerzitu. O rok později publikoval práci The Complexity of Theorem Proving Procedures[1]. V ní představil novou teoretickou koncepci,

1 Složitost procedur pro dokazování vět

kterou nazval NP úplnost. Na základě svého objevu byl Cook následně zvolen členem Královské kanadské společnosti a americké Národní akademie věd.

Koncepce NP úplnosti vybavila odborníky v teorii složitosti mocným nástrojem pro analýzu výpočetních úloh. Cook ve své práci (1971) dokázal pouze to, že jistý velice umělý a obskurní problém z výrokové logiky je NP úplný a že jej téměř zaručeně není možno vyřešit ani pomocí počítače. Přesto za pár měsíců dokázal Richard Karp z Kalifornské univerzity v Berkeley totéž pro dalších jedenadvacet problémů, z nichž několik vycházelo z praxe a o jejich řešení měl značný zájem průmysl. Následně byl seznam NP úplných problémů rozšířen na několik set, možná tisíc. Téměř všechny výpočetní úlohy mají nějaké aplikace v průmyslu“ [5].

Cook po mnoha letech poznamenal: „NP úplnost se mi jevila jako zajímavý nápad, neuvědomil jsem si ale její možný dopad.“[2]

1.3 Problém P versus NP

Jako problém P versus NP se v teoretické informatice označuje otázka, zda platí rovnost P=NP. Považuje se za nejdůležitější otevřený problém tohoto oboru a je zařazený mezi sedm tzv. problémů tisíciletí.

Problémy tisíciletí[3] je označení pro sedm matematických problémů, které v roce 2000 vyhlásil Clayův matematický institut jako nejdůležitější otevřené problémy soudobé matematiky. Jsou tak jakousi obdobou Hilbertových problémů ze začátku dvacátého století.

1.3.1 Popis tříd P a NP

Třída složitosti P obsahuje všechny úlohy, jejichž řešení lze nalézt deterministickým Turingovým strojem v polynomiálním čase. Polynomiálně řešitelné úlohy jsou úlohy, pro něž existuje algoritmus, které tuto úlohu řeší v polynomiálním čase. Doba takového řešení je pro nejhorší případ omezena shora asymptoticky nějakým polynomem. Pro takový algoritmus pak platí, že doba výpočtu je $O(p(n))$, kde n je velikost vstupních dat a p je nějaký polynom. Takový algoritmus řeší danou úlohu v *polynomiálním* čase nebo je prostě *polynomiální*. Třída úloh P je tvořena polynomiálně řešitelnými rozhodovacími úlohami, kdy za rozhodovací úlohu považujeme takovou, jejíž výsledek je buď „ano“, nebo „ne“. Úlohy tohoto typu jsou většinou

2 CITOVÁNO z [17]

3 Anglicky Millenium Prize Problems

koncipovány tak, že se ptáme, zda existuje přípustné řešení s hodnotou účelové funkce, která je lepší nebo aspoň rovna nějaké definované konstantě. Zvláštním typem rozhodovací verze je pak dotaz, zda vůbec přípustné řešení existuje. Toto se provede tím, že se nějakým vhodným způsobem zvolí nějaká konstanta, např. jako velmi malé (resp. velmi velké) číslo. Následně se ptáme, zda existuje takové řešení, které je větší (resp. menší) než zvolená konstanta. Polynomiálně řešitelné úlohy jsou většinou úlohy, jež lze snadno vyřešit [5], [9], [15].

Pro třídu NP platí totéž s tím rozdílem, že se jedná o nedeterministický Turingův stroj. Jsou to ty problémy, jejichž řešení lze *ověřit* v polynomiálním čase. Nevíme však, zda je lze také v polynomiálním čase *nalézt*.

1.3.2 Nedeterministicky polynomiální

NP (zkratka nedeterministicky polynomiální) je množina problémů, které lze řešit v polynomiálně omezeném čase na nedeterministickém Turingově stroji - na počítači. Ten umožňuje v každém kroku rozvětvit výpočet na *n* větví, v nichž se posléze řešení hledá současně. Ekvivalentně se hovoří o stroji, který na místě rozhodování uhodne správnou cestu výpočtu. Alternativně lze tyto problémy definovat tak, že je to množina problémů, u kterých lze pro dodaný výsledek v polynomiálním čase ověřit jeho správnost (ale obecně nikoliv nalézt řešení v polynomiálním čase). Složitostní třída P je obsažena v NP. NP ale obsahuje velké množství důležitých problémů, zvaných NP-úplné, pro které není znám žádný polynomiální algoritmus. Pravděpodobně nejdůležitějším problémem současné informatiky (patří mezi Problémy milénia) je otázka, zda P = NP. Většina expertů se spíše kloní k názoru, že P je vlastní podmnožinou NP.

Do třídy úloh NP (nedeterministicky polynomiální) patří rozhodovací úlohy, pro které existuje tzv. *ověřovací algoritmus*, který má následující vlastnosti:

- vstupem algoritmu jsou data *d* popisující instanci úlohy a tzv. *certifikát*, který můžeme popsat jako nějakou blíže nespecifikovanou část dat, jejíž velikost je shora omezena určitým pevně daným polynomem vzhledem k délce vstupních dat,
- algoritmus pracuje v polynomiálním čase vzhledem k velikosti vstupních dat *d* a výsledkem je vždy odpověď „ano“ nebo „nevím“,
- pro instanci dané úlohy d, kdy je správná odpověď „ano“, existuje takový certifikát *c*, že ověřovací algoritmus dá odpověď „ano“,

- pro instanci dané úlohy d, kdy je správná odpověď „ne“, existuje takový certifikát *c*, že ověřovací algoritmus dá odpověď „nevím“.

1.3.3 NP – úplnost

V literatuře se lze poměrně často setkat s pojmy a výroky o NP-úplnosti některých úloh, tedy o tzv. NP-úplných[4] úlohách atp. Jistým, avšak poměrně vágním vyjádřením může být prohlášení, že NP-úplné úlohy jsou obtížně řešitelné. Cílem následujících odstavců je objasnit pojem NP-úplné úlohy.

Pro pochopení některých základních vlastností výše uvedených pojmů je třeba rozlišovat typ úlohy a instanci úlohy. *Typem úlohy* rozumíme, jakým způsobem je úloha zadána. To znamená jaká data a v jakém uspořádání budou tvořit zadání úlohy, co má být výsledkem a jaký má být konečný vztah mezi výsledkem a zadáním. Konkrétním příkladem úlohy daného typu je pak *instance úlohy.* Jedním z konkrétních typů úloh jsou tzv. *rozhodovací úlohy*, kdy výsledkem je rozhodnutí buď „ano“ nebo „ne“. Některé rozhodovací úlohy jsou odvozeny z optimalizačních úloh, kdy se k zadání přidá konstanta *K.* Zjišťuje se, zda existuje přípustné řešení s hodnotou účelové funkce, která je lepší nebo rovna *K* [9].

Vstupem do algoritmu, který řeší konkrétní typ úlohy, jsou nějaká data popisující instanci daného typu úlohy. Tato data se nazývají vstupní data. Velikost instance pak měříme jako objem těchto vstupních dat. Obecně se používá velikost dat v bitech, ale např. v grafových úlohách se velikost dat vyjadřuje počtem vrcholů a počtem hran.

NP-úplné problémy jsou takové nedeterministicky polynomiální problémy, na které jsou polynomiálně redukovatelné všechny ostatní problémy z NP. To znamená, že třídu NP-úplných úloh tvoří v jistém smyslu ty nejtěžší úlohy z NP [6]. Pokud by byl nalezen deterministický polynomiální algoritmus pro nějakou NP-úplnou úlohu, znamenalo by to, že všechny nedeterministicky polynomiální problémy jsou řešitelné v polynomiálním čase, tedy že třída NP se „zhroutí“ do třídy P (NP = P). Otázka, zda nějaký takový algoritmus existuje, zatím nebyla rozhodnuta, předpokládá se však, že NP ≠ P.

Hlavní důvod, proč jsou NP-úplné úlohy tak zajímavé, je právě jejich velmi obtížná řešitelnost. Například uplatnění v moderní kryptografii, kódování, kde musíme být schopni rychle ověřovat správnost řešení, ale jeho nalezení musí trvat dlouho. Obtížnost výpočtu ovšem záleží

4 NP-complete, NPC

i na konkrétních datech, pro speciální množinu vstupů může být úloha polynomiální, například řešíme-li obarvení třemi barvami pro jednoduché grafy.

Mezi typické NP-úplné úlohy patří např. problém obchodního cestujícího, tj. hledání (nejkratší) hamiltonovské kružnice, RSA problém, hledání nezávislé množiny, problém kliky (hledání úplného podgrafu), hledání isomorfního podgrafu, 3 barevnost grafu, vrcholové pokrytí, zavazadlový problém (tzv. problém batohu), problém dvou loupežníků atd..

1.3.4 Polynomiální versus exponenciální čas (výpočty trvající déle než stáří vesmíru)

Polynomiální procesy jsou zhruba řečeno ty, které počítač efektivně zvládne. Kdyby totiž byla čísla C a k příliš veliká, například kdyby se k rovnalo několika tisícům, pak by proces mohl vyžadovat tolik základních kroků, že by výpočet trval po celou dobu existence vesmíru. V praxi ale mají polynomiální procesy, které vyvstávají z každodenního života, velmi skromné hodnoty C a k (k bývá obvykle jednociferné) [5], takže je počítače zvládnou opravdu hravě.

Skutečný význam polynomiálních procesů pochopíme při jejich srovnání s těmi, které vyžadují „exponenciální čas“. Vložíme-li do nich vstupní data o velikosti N, tyto procesy spotřebují 2^N nebo více kroků. Například jednoduchá vyhledávácí procedura, řešící problém obchodního cestujícího, vyžaduje k nalezení odpovědi odhadem $N!$ základních kroků, což je ještě podstatně více než 2^N [5].

To, co dělá exponenciální proces skoro neproveditelným, je rychlost, s jakou roste číslo 2^N při zvětšujícím se N. Abychom získali určitou představu tohoto růstu, představme si obyčejnou šachovnici. Řekněme, že očíslujeme všechna políčka na šachovnici, přičemž postupujeme od jedničky v levém horním rohu řadu po řadě dolů až k číslu 64 v pravém dolním rohu.

Představme si, že budeme na šachovnicová políčka pokládat korunové mince. Na políčko označené jedničkou položíme dvě korunové mince (tedy 2^1), na políčko s číslem 2 položíme čtyři mince (tedy 2^2). Na políčko 3 osm mincí (tedy 2^3) a tak dále, takže na každé políčko položíme přesně dvakrát více mincí než jsme položili na políčko předcházející. Na poslední políčko položíme přesně 2^{64} korunových mincí. Jak myslíte, že bude vysoký sloupeček tvořený těmito mincemi? Patnáct metrů? Víc? Ve skutečnosti by dosáhl výšky asi 37 bilionů kilometrů. Trčel by daleko za Měsíc (který se nachází 400 tisíc kilometrů od Země) i za Slunce (od kterého nás dělí pouhých 150 milionů kilometrů). Dosáhl by až k další nejbližší hvězdě, jíž je Proxima Centauri [5].

Sloupce na prvních několika políčkách se nezdají být příliš vysoké. Ale jak se zvětšuje *N*, výšky sloupců dramaticky narůstají. Pro dostatečně malé objemy dat je možné spustit exponenciální proces a získat odpověď. Například problém obchodního cestujícího pro tři města jsme vyřešili ručně. Ale s narůstajícím objemem dat jednou prostě nebude k dispozici dost času k vyřešení úlohy. Protože prakticky všechny exponenciální procesy, které vyplývají z průmyslu a obchodu, by zabraly nejrychlejšímu počítači na světě čas převyšující celkové stáří vesmíru, i kdyby zpracovávaly poměrně skrovné a velice reálné objemy dat. Následující Tabulka 1 porovnává časy, které by slušně rychlý počítač (takový, který umí provést jeden milion základních aritmetických operací za sekundu) potřeboval k provedení procesů s různými funkcemi časové složitosti.

Tabulka 1: Porovnání časů, zdroj [5]

Funkce časové složitosti	Objem dat: *N*				
	10	20	30	40	50
N	0,00001s	0,00002s	0,00003s	0,00004s	0,00005s
N^2	0,0001s	0,0004s	0,0009s	0,0016s	0,0036s
N^3	0,001s	0,008s	0,027 s	0,064s	0,125s
2^N	0,001s	1,0s	17,19min	12,7 dne	35,7let
3^N	0,059 s	58 min	6,5 roku	3855stol.	200mil.stol.

Na prvních třech řádcích vidíme polynomiální procesy. Výpočetní čas sice roste s narůstajícím objemem dat, nárůst je ale umírněný a dokonce ještě i pro data objemu 50 trvá proces jen zlomek sekundy.

Na posledních dvou řádcích najdeme exponenciální procesy. Zde narůstá výpočetní čas dramatickou měrou. U procesů s nezbytným časem 2^N jde o dny, jakmile se objem dat přiblíží ke čtyřiceti, a posune-li se objem dat jen k padesátce, už by nám nalezení odpovědi trvalo přes 35 let. U procesů s nezbytným časem 3^N vyžadují data objemu 40 skoro 4 000 století a data objemu 50 dokonce závratných 200 milionů století.

Tabulka ilustruje propastný rozdíl mezi polynomiálními a exponenciálními procesy. Pochopitelně platí, že je-li exponenciální proces jedinou možností, jak vyřešit daný problém, pak jej prostě budeme schopni vyřešit pouze pro velmi malé objemy dat. Například *N*! je pro všechna *N* kromě velice malých hodnot mnohem větší než 3^N, takže metoda prozkoumání všech možností v problému obchodního cestujícího je předem ztracený případ.

Obrovská propast mezi polynomiálními a exponenciálními procesy ilustruje zřetelnou slabinu této klasifikace: je příliš hrubá. Když si to matematikové uvědomili, začali se pídit

po nějakém pomocném měřítku výpočetní složitosti. Přišli na to, že složitost takových procesů, jako je metoda prostého vyhledávání pro řešení problému obchodního cestujícího nebo problému optimalizace procesu, nespočívá v tom, že by výpočet byl komplikovaný. Naopak, výpočet je extrémně jednoduchý. To, co dělá problém skoro neřešitelným, je prostě jen čirý počet všech možností, které je třeba prozkoumat a pro které je nutno onen extrémně jednoduchý výpočet opakovat znovu a znovu. Protože lidská mysl nebo digitální počítač provádí tyto operace postupně, tedy jednu po druhé, celý proces by trval nemožně dlouho [5].

Pro odlišení takových procesů od těch, které obsahují skutečně komplikované výpočty, byla zavedena ještě třetí kategorie, takzvané nedeterministické polynomiální procesy (zkráceně NP). Běžné počítače jsou deterministické a tudíž pracují zcela předvídatelným způsobem a dodržují předem smluvená pravidla. Proto by měl výraz „nedeterministický“ naznačit, že tento nový koncept je teoretický a nemá nic společného se skutečnými výpočty. Základní myšlenka je následující.

Představme si, že náš počítač může v jisté fázi výpočtu rozhodnout zcela náhodným způsobem mezi několika možnostmi. Například, že při řešení problému obchodního cestujícího může vždy vybrat naprosto náhodně jednu z přípustných tras. Zvolí si tedy jednu trasu a spočítá odpovídající vzdálenost. Pravděpodobnost, že vybraná trasa nebude tou nejkratší, je velmi vysoká. Ale co když má náš počítač neuvěřitelné štěstí a vždy vybere tu nejlepší možnou trasu? Pak by vyřešil problém v polynomiálním čase. Taková dokonalá schopnost šťastné náhody by pomohla obejít potíže s velkým počtem různých možností.

Obecně říkáme, že problém nebo úloha je typu NP, jestliže jej nebo ji lze vyřešit nebo dokončit v polynomiálním čase pomocí nedeterministického počítače. Ten musí být schopen náhodné volby mezi několika možnostmi a nadaný navíc dokonalou schopností šťastné trefy. (Měli bychom si ale uvědomit, že počítač musí ověřit, že se trefil správně. Podstata třídy NP spočívá v tom, že jde pouze o čirý počet možností. Pro problém typu NP musí být možné ověřit optimalitu získaného řešení v polynomiálním čase).

Intuitivně cítíme, že problémy typu NP leží někde mezi polynomiálními problémy (zkráceně problémy typu P) a problémy čistě exponenciálními. Celý koncept NP je čistě teoretický, protože je založen na zcela nereálné myšlence, že by počítač mohl vždy bez výjimky učinit nejlepší možnou náhodnou volbu. Tato schopnost se někdy označuje jako věštírna nebo orákulum. Nicméně jak se ukazuje, je to koncept značného významu. Jedním z důvodů je to, že většina exponenciálních problémů, které vyvstávají v průmyslu a v řízení ekonomiky, jsou typu NP. Jejich řešení jsou obtížná ne proto, že by obsahovala komplikované výpočty, ale proto, že poměrně jednoduchý výpočet musí být opakován pro ohromující záplavu skoro totožných případů.

Když byla klasifikace NP v šedesátých letech dvacátého století zavedena, předpokládali odborníci na informatiku, že třídy P a NP nejsou totožné. Platí, že každý problém typu P je také zároveň automaticky typu NP, ale na druhé straně existují problémy typu NP, které rozhodně nejsou typu P. Zdá se totiž, že neexistuje žádný způsob, jakým by standardní počítač mohl pomocí algoritmu závisejícího polynomiálně na čase pracovat stejně dobře jako imaginární nedeterministický počítač, který dělá dokonalé volby. Odborníci například měli za to, že problém obchodního cestujícího prostě nemůže být vyřešen v polynomiálním čase bez schopnosti dokonalého hádání hypotetickým nedeterministickým počítačem.

Bylo jen otázkou času, než někdo předloží nějaký konkrétní NP problém, který není zároveň P problémem. Dokáže tak, že P a NP jsou rozličné třídy, nebude-li to zrovna problém obchodního cestujícího, najde se nějaký jiný. Nic takového se nestalo. Také však nikdo nedokázal opak, tedy že obě třídy splývají. Tak se zrodil problém P versus NP.

V oné době , tj. koncem šedesátých let dvacátého století, šlo o hodně. O mnoha důležitých problémech z průmyslu a řízení bylo známo, že jsou třídy NP. Důkaz, že P a NP jsou tytéž třídy, by nepochybně odstartoval obrovské úsilí zaměřené na nalezení efektivních procedur, které by vedly k vyřešení těchto důležitých problémů. Zdůrazníme, že důkaz totožnosti tříd P a NP by ještě sám o sobě neznamenal, že se pro specifické problémy takové efektivní procedury najdou. Znamenal by pouze, že každý NP problém může být v principu vyřešen pomocí procedury, která závisí na čase polynomiálně. Nemusel by ovšem přinést návod, jak by taková procedura měla vypadat [5].

V této fázi se na scéně objevil Stephen Cook. Ve svém slavném článku z roku 1971 podstatně pokročil ve studiu problému P versus NP. Zároveň přinesl druhý zásadní důvod jeho důležitosti.

Cook přišel na to, že jeden speciální NP problém má podivuhodnou vlastnost: kdyby jej bylo možné vyřešit v polynomiálním čase, pak by totéž platilo o kterémkoli jiném NP problému! Přesná povaha Cookovy úlohy není pro nás tolik důležitá. Obecně řečeno šlo o to, jaké druhy úloh mohou být řešeny na nedeterministických počítačích. Cook ukázal, že jakýkoli NP problém je možno převést na jeho konkrétní problém. Takže kdyby bylo možné vyřešit v polynomiálním čase jeho problém, totéž by díky přenosu platilo o kterémkoli jiném NP problému. Cook tuto podivuhodnou vlastnost nazval NP úplností. Podle Cooka říkáme, že nějaký NP problém je NP úplný, jestliže by objev polynomiální procedury vedoucí k jeho řešení znamenal, že kterýkoli jiný NP problém může být řešen polynomiálně. Ačkoli šlo o vysoce teoretický problém formální logiky, nedlouho poté Richard Karp a další dokázali, že mnoho jiných

známých NP problémů tuto vlastnost NP úplnosti má. To se také týká problému obchodního cestujícího a problému optimální technologie.

Objevy NP úplnosti a toho, že nejznámější NP problémy jsou NP úplné, znamenaly z jistého hlediska těžkou ránu pro průmysl. Konstrukce efektivních procedur, které by uměly vyřešit úlohy, jakou je problém obchodního cestujícího, by totiž průmyslu přinesla růst zisků a miliardy dolarů. NP úplnost neznamená, že problém v žádném případě nelze vyřešit efektivně. Koneckonců, dokud je problém P versus NP otevřený, pořád se ještě může ukázat, že obě kategorie splývají, v kterémžto případě by každý NP problém bylo možné v principu vyřešit v polynomiálním čase. Jde spíš o to, že jestliže se prokáže, že nějaký problém je NP úplný, pak to naznačuje, jak těžká je to úloha, a tedy že je velice nepravděpodobné, že by se mohla najít polynomiální procedura k jejímu vyřešení. Důvody jsou následující.

Protože úlohy typu problému obchodního cestujícího či problému optimální technologie jsou tak důležité, obětovalo po mnoho let obrovské množství talentovaných mladých výzkumníků mnoho hodin snaze najít efektivní cesty k jejich řešení. Představme si, že dokážeme, že nějaký náš oblíbený NP problém je NP úplný. Pak je tento problém úplně stejně obtížný jako všechny ostatní problémy, které se všichni marně snažili vyřešit. V důsledku toho považují odborníci jakýkoli důkaz, že nějaký problém je NP úplný, za dostatečně pádný důvod k tomu, aby přestali vynakládat čas a úsilí na hledání jeho úplného řešení. Místo toho věnují svou energii na hledání rozumných přibližných řešení nebo jednorázových řešení některých speciálních případů. Takže navzdory své velice umělé povaze pomáhá NP klasifikace manažerům rozhodovat, kam investovat výzkumné úsilí.

A přitom se stále za vším skrývá dosud nevyřešený problém P versus NP. Důkaz, že P a NP jsou totéž, by v principu znamenal, že veškerá práce na NP úplnosti je ztrátou času.

Takový důkaz by měl vážné důsledky také pro bezpečnost internetového provozu. V polovině sedmdesátých let dvacátého století vybudovali matematici a informatici efektivní novou metodu kódování elektronických vzkazů posílaných otevřenou počítačovou sítí. Bezpečnost metody zvané RSA, závisí na tom, že nalezení tajného klíče není úloha z kategorie P. Přestože tento klíč je v principu možné získat, nejrychlejšímu počítači by to trvalo několik let. Prolomení kódu je tedy problém kategorie NP. (Podobně jako u problému obchodního cestujícího spočívá jeho složitost v obrovském počtu různých kombinací.) Kdyby se ukázalo, že prolomení kódu je ve skutečnosti problém z kategorie P, ocitla by se tato metoda nepochybně v ohrožení.

O problému prolomení kódu metody RSA není známo, že by byl NP úplný, takže možná by jeho řešení závisející na čase polynomiálně mohlo být vyvinuto i bez důkazu,

že P splývá s NP. Ale vzato z druhé strany by důkaz totožnosti P a NP okamžitě znamenal, že problém prolomení RSA kódu je řešitelný v polynomiálním čase. Tím by tak zpochybnil celý systém internetového zabezpečení. V současnosti neznáme žádný způsob, jak zaručit bezpečnost komunikace po internetu, který by nezávisel na nemožnosti efektivně vyřešit nějaký NP problém. Současná závislost západních ekonomik na bezpečné elektronické komunikaci přes internet současně ukazuje, jak mnoho je v sázce v souvislosti s problémem P = NP.

2 Vybrané NP-úplné problémy

V této kapitole jsem se zaměřila na 4 problémy, které patří k nejznámějším NP-úplným problémům:

- kombinatorická optimalizace NP-úplných problémů:
  - Problém batohu
  - Problém obchodního cestujícího
  - Barvení grafů
- RSA problém

Důvodem bylo, že tyto problémy mají také široké praktické použití. Na některé příklady jejich praktického využití jsem se zaměřila.

2.1 Problém batohu

Problém batohu spočívá v nalezení optimálního naplnění batohu věcmi tak, aby nebyla překročena jeho nosnost a zároveň aby sumární cena věcí uvnitř byla:

- *nejvyšší* – exaktní řešení
- *co možná nejvyšší* – rychlé řešení

Máme batoh, který má pevně danou nosnost. Máme množinu věcí, které mají svou cenu a váhu. Úkolem je naplnit batoh, aby součet vah do batohu uložených věcí nepřekročil nosnost a přitom součet jejich cen byl co největší.

Tento problém se dá řešit algoritmem, řešící problém hrubou silou, postupně generuje všechny možné kombinace věcí umisťovaných do batohu a testuje obě podmínky (zda zvolené kombinace nepřetíží batoh a zda cenová funkce nabývá maxima).

Negativním příkladem problému batohu může být např. zloděj. Jeho snahou je dát do batohu co nejvíce kořisti (šperky, hodinky). Je však limitován maximální výši hmotnosti K v kilogramech, kterou může dát do batohu. Věci, které odcizil a jsou již v batohu, se skládají z n předmětů o hmotnosti w_1, w_2 …w_n kilogramů o cenách c_1, c_2 ….., c_n Kč. Snaha zloděje je, aby měl co největší kořist v co největší hodnotě [6].

Tato úloha má mnoho poctivějších podob, zejména při sestavování rozvrhu a plánování paralelně probíhajících procesů jak v teoretické, tak praktické rovině [6].

Příkladem může být problém batohu, který je řešen v grafech. Jde o aplikaci rozhodovacího procesu, kdy vrcholy grafu odpovídají stavům rozhodovacího procesu a hrany grafu představují možná rozhodnutí o určitém předmětu. Kombinace poloh nakreslených hran slouží k rozhodování o tom, zda věci o hmotnosti a v ceně „zabalit“ do batohu anebo „nezabalit“. Vrcholy grafu jsou označené souřadnicemi. Jedny ukazují na to, o čem je již rozhodnuto. Druhé ukazují hmotnost předmětů. Šikmé hrany ukazují cenu zabalených předmětů. Rozhodování je limitováno kapacitou batohu. Základem k rozhodnutí jsou tzv. orientované cesty ze souřadnic vrcholu, které mohou např. uvádět součet cen již vybraných / zabalených předmětů. Tyto speciální grafy ukazují také na skutečnost, že v úvahu může přicházet více variant rozhodnutí. Při nich např. nemusí být naplněn požadavek na maximální využití kapacity batohu [6].

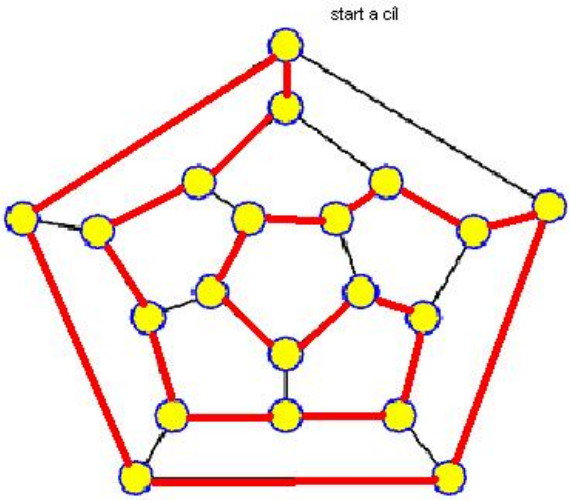
Při řešení problému batohu se však všeobecně doporučuje více využívat známých algoritmů. Mezi ně např. patří tzv. backtracking, který v základu spočívá v prohledávání stromu řešení do hloubky. Dále pak metody větví a mezí pro řešení optimalizačních mezí. Podotýkám, že to již značně přesahuje rámec této bakalářské práce, a proto nebude těmto algoritmům věnována další pozornost.

2.2 Problém obchodního cestujícího

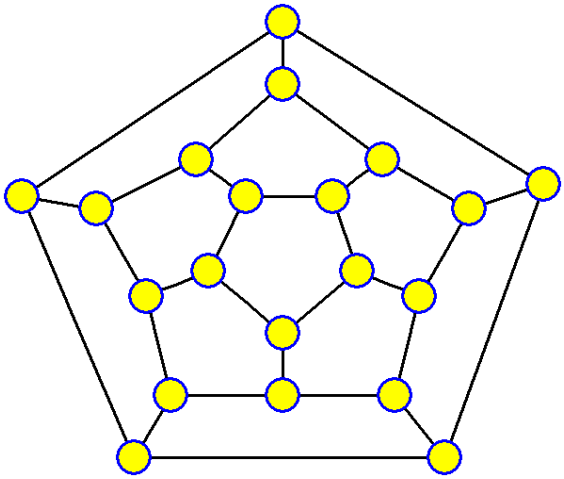
Problém obchodního cestujícího je obtížný diskrétní optimalizační problém, matematicky vyjadřující a zobecňující úlohu nalezení nejkratšího možného průchodu všemi zadanými body na mapě a vrácení zpět do výchozího bodu. Přesněji řečeno, jde o nalezení nejkratší hamiltonovské kružnice v úplném neorientovaném grafu, jehož hrany jsou ohodnoceny délkami.

2.2.1 Hamiltonovská kružnice

Graf, který obsahuje jako svou část kružnici procházející všemi jeho vrcholy, nazýváme *hamiltonovský graf* a příslušnou kružnici *hamiltonovská kružnice* (viz. Obrázky 1 a 2)



Obrázek 1: Hamiltonovská kružnice v Hamiltonovském grafu



Obrázek 2: Hamiltonovský graf - 12-ti stěn

Název grafu se vztahuje ke jménu irského matematika Williama Rowana Hamiltona. Ten vymyslel v roce 1857 hru, skládající se z pravidelného 12-ti stěnu (viz. Obrázek 2) a 20 ti kolíků. Každý kolík byl umístěn v jednom rohu 12-ti stěnu (tj. v každém vrcholu) a byl označen jménem některého z hlavních měst Evropy. Úkolem hráče bylo najít cestu procházející všechna města, každé právě jednou, a vrátit se zpět do města, z kterého vyšel. (Pozn.: Na každém kolíku byl provázek, který hráč spojil vždy s následujícím kolíkem, v pořadí, jakým města procházel).

Každý graf nemusí mít nutně hamiltonovskou kružnici. Nutnými (avšak nikoli postačujícími) podmínkami je, že graf musí být souvislý a každý vrchol musí mít stupeň nejméně rovný dvěma (ke každému vrcholu musí vést alespoň 2 hrany).

Jedním z příkladů NP úlohy je nalezení hamiltonovské kružnice, kdy certifikátem je zde kružnice, ověřovací algoritmus ověřuje, zda se opravdu jedná o kružnici a zda prochází všemi

body. Je velmi důležité jakým způsobem je položena otázka, která se v úloze řeší a to z důvodu požadavku certifikátu a potvrzení pouze pro kladnou odpověď. Pokud otázku obrátíme, můžeme dostat zcela jinou úlohu a ta již do třídy NP patřit nemusí (nebo může být i těžší).

Nyní uvedu několik pravidel pro hledání hamiltonovské kružnice v daném grafu:

- má-li daný graf *n* vrcholů, pak hamiltonovská kružnice má právě *n* hran,
- jestliže vrchol *v* má stupeň *k*, pak hamiltonovská kružnice musí obsahovat právě dvě hrany incidentní s vrcholem *v*,
- při konstrukci hamiltonovské kružnice nemůže být vytvořena kružnice, která by neobsahovala všechny vrcholy daného grafu,
- jakmile hamiltonovská kružnice, kterou konstruujeme, prochází vrcholem *v*, pak ostatní nepoužité hrany incidentní s vrcholem *v* můžeme vyloučit (tyto již v hamiltonovské kružnici ležet nebudou) [6].

2.2.2 Obchodní cestující

Představme si, že jsme obchodními cestujícími ze Strakonic. Naším úkolem je autem navštívit postupně tři města Olomouc, Most, a Náchod, přičemž vyjedeme ze Strakonic a zase se tam vrátíme. Abychom ušetřili čas a benzin, chceme naplánovat svoji cestu tak, aby celková vzdálenost, kterou urazíme, byla co nejkratší. Vytáhneme tedy mapu a zjistíme si nejkratší silniční vzdálenosti mezi každými dvěma městy. Naměřené výsledky[i] představuje Tabulka 2.

Tabulka 2: Silniční vzdálenosti, zdroj [5]

Město	Strakonice	Olomouc	Most	Náchod
Strakonice	0	54	17	79
Olomouc	54	0	49	104
Most	17	49	0	91
Náchod	79	109	91	0

Nyní již jen zbývá uspořádat tři města tak, abychom ujeli co nejkratší vzdálenost. Jak těžká je tato úloha? Spočítat celkový počet procestovaných km pro každé jednotlivé uspořádání daných měst je snadné. Vše, co je k tomu potřeba, je najít tři čísla v tabulce a sečíst je. Provedeme-li to pro každou přípustnou trasu, tedy pro každé možné uspořádání tří měst, pak stačí porovnat

získané výsledky a vybrat ten, který udává nejkratší celkovou trasu. Níže uvedena výsledná Tabulka 3.

Tabulka 3: Výsledná tabulka, zdroj [5]

Trasa	Celková délka
S-O-M-N-S	54 + 49+ 91 + 79 = 273
S-O-N-M-S	54 + 104 + 91 + 17 = 266
S-M-N-O-S	17 + 91 + 109 + 54 = 271
S-M-O-N-S	17 + 49 + 104 + 79 = 249
S-N-O-M-S	79 + 109 + 49 + 17 = 254
S-N-M-O-S	79 + 91 + 49 + 54 = 273

Je zřejmé, že optimální trasa je S-M-O-N-S s celkovou vzdáleností 249km.

Podobně jako většina jiných jednoduchých příkladů není ani tento příliš praktický. Máme-li navštívit jenom tři města, pak mezi jednotlivými trasami nejsou příliš velké rozdíly a nejspíš ani nestojí za to provádět nějaké výpočty. Ale pro obchodního cestujícího, který se chystá do většího počtu měst, se může řada drobných rozdílů poskládat do významné sumy. Na tento příklad se můžeme dívat i z pohledu kvality trasy (zda se pojede po dálnici nebo silnicí II třídy, apod.)

Řekněme, že jsme vlastně chtěli navštívit celkem deset měst. V takovém případě se nejspíš rozhodneme přenechat výpočet počítači. Sestavíme tedy tabulku a napíšeme krátké makro, které ohodnotí délku každé možné trasy a spočítá celkovou ujetou vzdálenost. Pro deset měst totiž činí celkový počet tras 3 628 800.

Abychom zjistili, odkud se vzalo toto číslo, vraťme se k původnímu příkladu se třemi městy. Existují tři různá města, ve kterých můžeme začít. Pak jsou vždy další dvě možnosti města, které navštívíme jako druhé. A pak už zbývá jen jedno město před návratem do Strakonic. Takže celkový počet různých tras je

$$3 \times 2 \times 1 = 6$$

To dává celkem šest různých tras seřazených v tabulce na předcházející stránce.

Pro deset měst máme deset možností, kde zažít, pak devět možných měst, která navštívíme jako druhá, osm možností třetího města a tak dále, takže celkový počet různých tras je:

$$10 \times 9 \times 8x\ 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 3\ 628\ 800.$$

To je veliká spousta tras, které musíme prozkoumat kvůli jednomu výletu. Vskutku, představme si, že spočítáme celkovou vzdálenost pro každou z těchto tras a že nám každý takový výpočet zabere přesně jednu minutu. Budeme-li pracovat osm hodin denně bez přestávky pět dní v týdnu padesát dva týdnů ročně, pak nám tato práce vydrží na něco málo přes dvacet let! Dvacet let kvůli deseti městům.

Přidáme-li jedenácté město, pak se celkový počet různých tras přiblíží čtyřiceti milionům:

$$11 \times 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 39\,916\,800.$$

Pochopitelně nás ani nenapadne provádět takový výpočet na papíře, prostě použijeme počítač. Protože ale faktoriály rostou velice rychle, stačí několik málo měst navíc, a i nejvýkonnější počítač bude udolán. Například číslo 25! je jen kousek od skličujícího numera sestávajícího ze šestnáctky a čtyřiadvaceti nul:

25! = 15 511 210 042 330 985 984 000 000.

Vzhledem k tomu, že okružní cesta po pětadvaceti městech není pro profesionálního obchodního cestujícího nic nemyslitelného, je zřejmé, že tento „problém obchodního cestujícího“, jak se také úloha správně nazývá, před nás klade závažnou výzvu. Pokud se nám zpočátku problém jevil jako přímočarý, bylo to prostě tím, že přímočarý je. Žádná náročná matematika není zapotřebí. Jediné, co potřebujeme, je sčítání dlouhých řad čísel a porovnávání výsledků. To, co dělá tento problém problémem, je pouhá velikost čísla udávajícího počet tras. Změníme počet měst z deseti na jedenáct a výsledný počet různých tras se zjedenáctinásobí. Přidáme další město a počet je rázem třeba vynásobit číslem dvanáct. V dalším kroku, kdy máme učinit okružní cestu pro třinácti městech, počet tras opět poskočí, a to tentokrát na třináctinásobek. Zní to tak nevinně: jedno jediné město navíc. Jenže s každým novým městem je třeba počet tras vynásobit konstantou, která navíc sama také narůstá.

Jestliže nějaká číselná veličina narůstá tímto způsobem, tj. je-li poměr jejího růstu v každém okamžiku přibližně přímo úměrný její aktuální hodnotě, pak říkáme, že jde o faktoriální růst. A je to právě tento růst, který dělá z problému obchodního cestujícího cosi jako časovanou bombu [5], [6], [9].

2.2.3 Problém barvení grafů

Barvení grafu je jednou z disciplín teorie grafů, která se zabývá přiřazováním barev (téměř vždy reprezentovaných přirozenými čísly) různým objektům v grafu – vrcholům, hranám, stěnám atd. Nejčastěji jde o barvení vrcholů, ostatní případy lze na tento jednoduše převést.

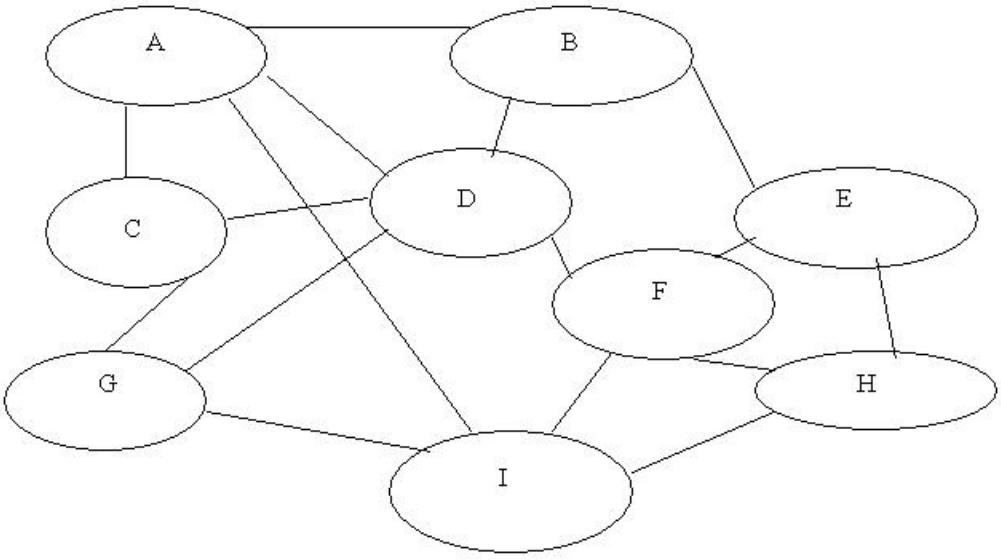
Definice: Obarvení grafu *G* (nebo též obarvení vrcholů grafu) je ohodnocení vrcholů hodnotami z množiny *B* (takzvanými *barvami*), a to takové, že žádné dva sousední vrcholy nejsou ohodnoceny (obarveny) stejnou barvou. V roli barev můžeme brát libovolné prvky z libovolné množiny, často používáme např. čísla [6], [9].

Graf nazýváme *r*-barevným, jestliže existuje jeho obarvení *r* barvami. *Barevnost grafu* (někdy též *chromatické číslo grafu* nebo *vrcholová barevnost*) je nejmenší počet barev, který je potřebný k obarvení grafu. Barevnost grafu *G* značíme $\chi(G)$ [6], [9].

Příklad: Přeprava nebezpečných látek. Mějme *n* nebezpečných látek. Bezpečnostní předpisy zakazují přepravovat některé dvojice látek ve stejném vozidle. Kolik nejméně vozidel nebo kolikrát se musí automobil vrátit k převozu všech *n* látek?

Řešení lze nalézt pomocí barevnosti: Sestrojme graf *G*, jehož vrcholy odpovídají zmíněným látkám a v němž jsou dva vrcholy spojeny hranou, jestliže odpovídající látky nesmí být přepravovány společně. Přípustné umístění látek do automobilu pak odpovídá obarvení grafu *G*. Roli barev zde hrají vozidla nebo kolikrát se automobil vrátí. Potřebujeme nejméně $\chi(G)$ vozidel.

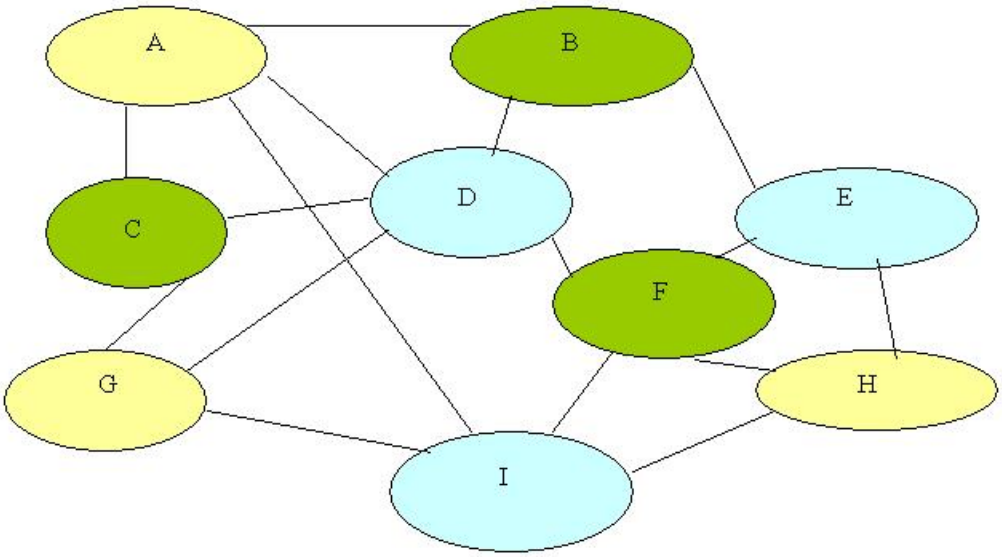
Konkrétní zadání: Mějme 9 nebezpečných látek, označených A, B, C, D, E, F, G, H, I. Bezpečnostní předpisy zakazují přepravovat některé tyto dvojice látek ve stejném automobilu. Tyto látky jsou pospojované hranou (viz. Obrázek 3).



Obrázek 3: Nebezpečné látky, zdroj [vlastní]

- K řešení využijeme barevnost respektive barvení grafu (viz. Obrázek 4)

- Počet barev nám v tomto případě vyznačuje, kolik bude potřeba vozidel, případně kolikrát se musí automobil vrátit pro naložení nebezpečných látek



Obrázek 4: Řešení, zdroj [vlastní]

- Pro obarvení tohoto grafu byly potřebné 3 barvy
- Zjistili jsme, že pro naložení nebezpečných látek budeme potřebovat buď tři automobily nebo automobil pojede tři krát.

2.2.4 RSA problém

K problému RSA se váže aplikace tzv. kryptografie z teorie složitosti. Jedná se o veřejné kódovací klíče. Ty umožňují individuální, bezpečnou a nezávislou komunikaci a kontakt mezi různými účastníky zejména v počítačové síti. Klíčovým problémem je veřejný kódovací klíč, který nemusí být utajen, když není možné zjistit dekódovací klíč. To platí i za podmínky, kdy se zásadně liší operace kódování a dekódování.

V principu můžeme říci, že princip RSA problému je založen na systému veřejného klíče a jeho generování. Jde o systém snadno vyčíslitelných funkcí. Např. *F, G, E, D.* Z funkcí *F* a *G* se pomocí náhodného čísla *r* vypočítají kódovací a dekódovací klíče. Vzniklá zpráva *x* je kódována funkcí *E* a kódovacího klíče *e*. Dekóduje se funkcí *D* a dekódovacím klíčem *d*. Tomu, kdo by chtěl zneužít takové systémy nesmí být dostupný dekódovací klíč *d* a *r* (to se může i zničit) [1], [16].

Kritické místo takového systému je funkce *F*. Není-li možné, aby se z funkční hodnoty *e* dal odvodit argument *r* dle *e=F (r)*. Protivník nesmí vypočítat *d* pomocí *G.* Jedná se o složité funkce, často nazývané jako jednosměrné [16]..

RSA je tedy nejdůležitější ze systému veřejných klíčů. V něm se náhodně volí dvě prvočísla p_1 a p_2 a jejich součin *N* je součástí klíčů.. V principu se dekódovací klíč *d* vypočítává z p_1 a p_2 a kódovacího klíče *e.* Spolehlivost systému je založena na principu velmi komplikovaného rozkladu čísla *N* v součin [16]. Jednosměrné funkce jsou definovány pomocí pojmu polynomiálního času. V samotném důsledku není možné dokázat, že lze rozkládat čísla polynomiálním algoritmem. Z toho v samotném důsledku vyplývá, že jsou pochybnosti o existenci tzv. jednosměrných funkcích. Jde o otevřený problém a zdá se, že ještě těžší než samotný problém P=NP. Přesto veřejné klíče existují. Dosud se jejich bezpečnost opírá o znalost nejlepších algoritmů, kterými se rozkládají čísla a na navazující úlohy pro existenci klíčů. Velmi často se studuje problém prvočísel. Ta jsou základem současné kryptografie a to tak, že pro RSA je nutné náhodně zvolit dvě velká prvočísla. I nadále zůstává otevřenou otázkou neexistence algoritmu, potřebného k rozhodnutí, zda prvočíslo je polynomiálním čase bez použití náhodných čísel. Tedy je-li množina prvočísel v P.

Poměrně závažným, a zatím jen v teoretické rovině, je problém neexistence tzv. kvantového počítače. Přesto, že byl navržen tzv. kvantový Turingův stroj jako standardizovaný zjednodušený matematický model počítače. To s sebou přináší nemožnost vyloučit skutečnost, zda se rozklad čísel dá počítat v polynomiální čase i na obyčejných počítačích. Následně se zatím beznadějně řeší zda „kvantové“ P je různé od klasického P. Podstatné je to, kdyby se podařilo kvantový počítač sestrojit, pak by byla současná kryptografie vážně ohrožena [1], [10], [16].

Turingův stroj je teoretický model počítače popsaný matematikem Alanem Turingem. Skládá se z procesorové jednotky, tvořené konečným automatem, programu ve tvaru pravidel přechodové funkce a potenciálně nekonečné pásky pro zápis mezivýsledků. Využívá se pro modelování algoritmů v teorii vyčíslitelnosti.

Jeden ze způsobů vyjádření Church-Turingovy teze říká, že ke každému algoritmu existuje ekvivalentní Turingův stroj

3 Vybrané algoritmy

Vzhledem ke složitosti NP-úplných optimalizačních problémů se většina vědců soustředí na jinou složku technik, tzv. heuristické optimalizační techniky nebo jednoduše heuristiky. Tyto techniky se snaží nalézt optimální řešení (nebo alespoň body poblíž optimálního řešení) v rozumném výpočetním čase. Někdy se také uvádí, že heuristiky jsou takové metody, které nezaručují nalezení optimálního (nebo ani přípustného) řešení. Zpočátku vývoje operačního výzkumu byly heuristiky brány s jistou skepsí, avšak v současné době si tyto techniky zajistily přední místo mezi optimalizačními metodami a to díky teoretickému výzkumu ve výpočetní složitosti, který signalizoval vrozenou složitost NP-úplných problémů.

Heuristické algoritmy mohou být nazývány též jako stochastické algoritmy:

- *algoritmus* jako heslo = posloupnost konečného počtu elementárních kroků vedoucí k řešení úlohy,
- *heuristika* jako heslo = teorie řešení problémů, neobvyklé řešení,
- s*tochastický* jako heslo = náhodný.

3.1 Definice heuristiky

Informatika jako věda o informacích a jejich zpracování chápe heuristiku jako postup získání řešení problému v krátkém čase. Slouží nejčastěji jako metoda rychle poskytující dostatečné a dosti přesné řešení, které však nelze obecně dokázat. Nejčastější použití heuristického algoritmu nalezneme v případech, kde není možné použít jiného lepšího algoritmu, poskytujícího přesné řešení s obecným důkazem. Heuristika se používá tam, kde úlohy nemají řešení v polynomiálním čase.

Jiný, avšak podobný pohled na heuristiku**:** heuristikou nazýváme postup, který, i když neprobere všechny možnosti, je schopný v některých případech podat výsledek. Výsledek je podáván zpravidla jedním ze dvou možných stavů. Buď jako kladný výsledek (odpověď) nebo jako výrok neurčitosti.

3.2 Použití algoritmu

Heuristické algoritmy se začaly používat s příchodem výpočetní techniky, kde jsou rozvíjeny a často používány pro řešení složitých problémů. Jsou vhodné zejména pro řešení složitých funkcí s mnoha parametry a složitým průběhem s mnoha extrémy.

Například algoritmy deterministické jsou v řešení složitých problémů velmi špatně použitelné, protože jejich náročnost (zejména časová) roste, s lineárně rostoucím rozsahem problému, exponenciálně. Naopak výhodou je oproti heuristickým algoritmům nalezení přesného, optimálního řešení, které lze dokázat.

Velmi často se setkáváme s metodami kombinujícími heuristické algoritmy s deterministickými.

Často používané metody heuristiky:

- *genetický algoritmus* - algoritmus založený na principu přirozené selekce. S úspěchem se využívá jako počítačový heuristický algoritmus schopný nalézt kvalitní řešení i složitého problému v oblasti IT ho využijeme pro řešení mnoha technických a matematických problémů,
- *metoda lokálního hledání* - jedná se o jednu z nejjednodušších metod. Dobře použitelná pro řešení matematických problémů,
- *iterativní metoda* - využívá postupného hledání řešení ve stále se zužující oblasti řešení (postupně se z dobrého řešení dopracovává k ještě lepšímu řešení),
- *optimalizace mravenčí kolonie* – mravenčí kolonie představují poměrně novou metodu diskrétní optimalizace. Inspirace pochází z chování skutečných mravenčích kolonií. Aplikací jednoduchých pravidel, jimiž se řídí jednotliví jedinci, vzniká komplexní chování celku, schopné řešit složité optimalizační úlohy.

Metod je celá řada a každá má své klady a zápory. Mezi klady řadíme možnost nalezení přesného řešení. Mezi zápory dobu, která je k provedení algoritmu potřeba, případně množství upřesňujících parametrů.

Větší jistotu, že nalezený výsledek je přesný, můžeme u některých metod zvýšit opakováním heuristického algoritmu.

Použití heuristických algoritmů je velmi dobře použitelné při řešení tzv. "Problému obchodního cestujícího", který má nejkratší cestou projít města vyznačená na mapě.

Běžně používané algoritmy využívající heuristiku, jejichž služeb využíváme každodenně, jsou algoritmy pro heuristickou analýzu integrované v antivirových softwarech. Dále se heuristické algoritmy využívají pro odhalení e-mailového spamu, ve specializovaném vývojovém a simulačním softwaru určeném pro různé oblasti použití (od strojírenství, přes oblast IT až po lékařství...).

3.2.1 Genetický algoritmus

Genetický algoritmus (genetic algorithms) [8], [13], je heuristický postup, který se snaží aplikací principů evoluční biologie nalézt řešení složitých problémů, pro které neexistuje použitelný exaktní algoritmus. Jejich velkou výhodou je poměrná jednoduchost. Ideovým vzorem pro genetické algoritmy byly principy vývoje, které se uplatňují v přírodě. Zde existují populace jednotlivých živočišných druhů, složených z jedinců různých vlastností. Tyto vlastnosti jsou prvotně zakódovány v jejich genech, které tvoří větší celky, chromozómy.

Ačkoli genetické algoritmy nemají již prakticky s biologií nic společného, udržují si biologickou terminologii. Evolucí jsou míněny postupné změny řešení vedoucí k nalezení extrému funkce. Konkrétní řešení $\boldsymbol{x}$ tvoří jedince (někdy je formálně jedinec nazýván chromozómem). U obchodního cestujícího bude jedincem jedna vygenerovaná hamiltonovská kružnice. Hodnota zdatnosti (*fitness*) jedince odpovídá hodnotě účelové funkce $f(\boldsymbol{x})$ v tomto řešení. Zde není tak důležitý samotný jedinec, jako postupný vývoj, kooperace a fungování populace jako souboru jedinců [11], [12]. Neúspěšní jedinci vymírají, úspěšní přežívají a množí se. Při aplikaci tohoto algoritmu na problémy operačního výzkumu každý jedinec v algoritmu nějakým způsobem kóduje jedno řešení $\boldsymbol{x}$ problému.

Pro manipulace s chromozómy se používají genetické operátory selekce (*selection*), křížení (*crossover*) a mutace (*mutation*). Při selekci se jedná o výběr jedinců z celkové populace, kteří se stanou rodiči. Důležitým hlediskem, jež se přímo či nepřímo uplatňuje při výběru alespoň jednoho z rodičů, je právě jeho zdatnost. Hybnou silou změn jsou křížení (výměna genetické informace mezi jedinci) a mutace (velmi málo pravděpodobné provedení náhodné změny chromozómu, zabraňující zdegenerování populace).

Genetické algoritmy pracují tak, že se nejprve vytvoří počáteční populace o velikosti p jedinců a pak se tato populace mění pomocí genetických operátorů tak dlouho, dokud není splněna nějaká podmínka ukončení.

Kostru genetického algoritmu lze definovat takto z [13]:

```
P:=počáteční_populace_chromozómů(p)
repeat
     R:=selekce(P)
     Q:=křížení(R)
     Q:=mutace(Q)
     P:=vytvořit_novou_populaci(P,Q);
until (podmínka_ukončení);
```

Počáteční populace se většinou získá náhodným generováním.

Selekcí je třeba z populace vybrat zdatné jedince (s dobrou hodnotou *fitness*[5]), kteří se potom stanou rodiči. Rodiče se vybírají pseudonáhodně. Zdatnější jedinci mají větší šanci být vybráni než méně zdatní jedinci. Čím je jedinec zdatnější (čím má lepší hodnotu *fitness*), tím je pravděpodobnější, že bude vybrán ke křízení. Pokud se výběr uskutečňuje v souladu s rozdělením pravděpodobnosti, počítaným jako podíl hodnoty fitness chromozómu k celkové hodnotě fitness populace, označujeme tento způsob výběru jako „ruleta“. Jiným způsobem selekce je například soutěžení (*tournament*). Soutěže se účastní jen část populace a její vítěz, který se stane nejzdatnějším účastníkem, se pak stane rodičem.

Křížení se pro vybranou skupinu rodičovských chromozómů může uskutečňovat několika způsoby. Například pro případ, kdy je jedinec reprezentován sekvencí čísel úloh, představuje nejjednodušší přístup tzv. jednobodové křížení, při němž se zvolí náhodně nějaký bod, dělící oba rodičovské chromozómy na dvě části. Jeden potomek pak zdědí levou část z prvního rodiče a na zbývající pozice se mu doplní chybějící prvky v tom pořadí, v jakém se vyskytují ve druhém rodiči, druhý potomek vznikne analogicky s obráceným pořadím rodičů. Např. jestliže řetězce BACDEF a DCABEF jsou rodičovské chromozómy a dělící bod se nachází za druhou pozicí, pak dostaneme potomky BADCEF a DCBAEF.

Mutace je v obecnosti malá náhodná změna jedné či několika proměnných (prvků chromozómu), která ovlivní řešení, ať už kladně nebo záporně. Pravděpodobnost uskutečnění mutace je nízká. Mutace je nutná k tomu, aby se zamezilo přílišné specializaci – zapadnutí celé populace do jednoho lokálního minima; aby vždy byla možnost vytvoření zásadně nových

[5] V přechodu do nové generace je pro každého jedince spočítána tzv. fitness funkce, jež vyjadřuje kvalitu řešení reprezentovaného tímto jedincem. Podle této kvality jsou vybíráni jedinci, kteří jsou dále modifikování (pomocí mutace a křížení). Tím pak vznikne nová populace.

chromozómů odpovídajících lepšímu řešení. Mutace přinášejí do chromozómů novou genetickou informaci.

Jednou z možností změny *p*-členné populace je vygenerovat pomocí křížení a mutace novou generaci *p* potomků a nahradit jí rodičovskou generaci en bloc. Jiné způsoby umožňují nějaké překrývání populace rodičů a potomků a populaci tedy mění inkrementálně. Např. vygenerovaný potomek nahrazuje pseudonáhodně vybraného slabého příslušníka aktuální populace.

3.2.2 Metoda lokálního hledání

Nejjednodušší, ale nejméně efektivní, heuristickou metodou je tzv. metoda lokálního hledání (*local search method*) [2], [3], někdy také uváděná jako metoda vyhledávání bezprostředního sousedství nebo horolezecký algoritmus. Tato technika je základem mnoha heuristických metod pro kombinatorické optimalizační algoritmy. Metoda určí směr nejprudšího spádu kompletním prohledáním sousedství náhodně vybraného počátečního řešení. Jedná se o jednoduchou iterativní metodu, která má za cíl nalézt alespoň dobré přibližné řešení. Tato metoda má základní nectnost gradientových metod, tj. že skončí v lokálním optimu a nedosáhne globálního optima.

Optimalizační problém je definován jako dvojice (X, f), kde X je množina všech přípustných řešení, tj. řešení splňujících omezení daného problému a f je účelová funkce, která zobrazuje množinu X do množiny reálných čísel. Cílem je pak nalézt takové řešení $\boldsymbol{x}$ v množině X, které optimalizuje účelovou funkci f.

Množina $N(\boldsymbol{x})$ obsahující všechna řešení, která mohou být dosažena z bodu $\boldsymbol{x}$ jednoduchým posunem (aplikací nějakého jednoduchého operátoru na bod $\boldsymbol{x}$), se nazývá *sousední okolí (sousedství)* bodu $\boldsymbol{x}$.

Řešení $\boldsymbol{x}$ se nazývá lokálním minimem funkce f v případě bezprostředního okolí $N(\boldsymbol{x})$, jestliže platí:

$$f(\boldsymbol{x}) \leq f(\boldsymbol{y}), \ \forall \boldsymbol{y} \in N(\boldsymbol{x}). \tag{1}$$

Lokální hledání je pak procedura, která minimalizuje účelovou funkci f počtem úspěšných kroků, kdy je aktuální řešení $\boldsymbol{x}$ (1) nahrazeno takovým řešením, že:

$$f(\boldsymbol{y}) < f(\boldsymbol{x}), \ \boldsymbol{y} \in N(\boldsymbol{x}). \tag{2}$$

Základní vyhledávací algoritmus se spouští v jakémkoliv náhodném řešení a končí v lokálním minimu, kde žádné další zlepšení již není schopen nalézt. Mezi těmito dvěma stavy pak existuje mnoho rozličných způsobů, jak lokální vyhledávání provádět. Například tzv. největší zlepšení lokálního prohledávání se aktuální řešení zamění s takovým řešením, kde je zlepšení největší (resp. kde je nejlepší zlepšení u účelové funkce) až po prohledání celého okolí výchozího bodu. Jiným příkladem je algoritmus tzv. prvního zlepšení, kdy lokální prohledání funguje tak, že je akceptováno lepší řešení v momentu jeho nalezení. Výpočetní složitost lokálního prohledávání záleží na velikosti bezprostředního okolí daného řešení a také na počtu kroků, které jsou potřebné ke zhodnocení konkrétního tahu ve funkčním okolí. Obecně se dá říci, že čím větší okolí bodu, tím více kroků je potřeba k jeho prohledání a tím lépe se dá nalézt lokální minimum.

Velkým problémem lokálního prohledávání je právě lokální minimum. Ačkoliv nalezené řešení může mít dobrou kvalitu, ještě to neznamená, že je nutně optimální. Kromě toho, kdy se lokální prohledávání ocitne v lokálním minimu, neexistuje nějaký zřetelný způsob, jak zajistit nalezení globálního minima. Metody založené na lokálním prohledávání, které se snaží odstranit tento problém, se někdy nazývají metaheuristickými metodami.

3.2.3 Iterativní lokální prohledávání

Jednou z nejjednodušších metaheuristik pro lokální prohledávání je opakované náhodné generování počátečního řešení, ačkoliv to znamená, že se všechny předešlé informace, dosažené v průběhu prohledávání, ztratí. Složitější verze tohoto přístupu využívá informací, které byly zaznamenány v předchozím kroku prohledávacího algoritmu a nazývá se iterativní lokální prohledávání (Iterated Local Search) [18]. Hlavní myšlenkou tohoto přístupu je využití předcházejícího nalezeného lokálního minima a záměny (ve většině případů se využije nejlepší nalezené řešení) nebo modifikace řešení (obvykle se provede určitý počet náhodných přesunů v prostoru) pro vytvoření nového počátečního řešení, čímž se zvyšuje pravděpodobnost navštívení slibnějších oblastí prohledávaného prostoru.

Jednoduchý základní pseudokód pro iterativní lokální prohledávání z [18]:

```
Iterativní_Lokální_Prohledávání
begin
     x = Vygenerované_Počáteční_Řešení
     x = Lokální_Prohledávání N(x)
     do
          x = Modifikace (x, historie)
```

```
            y = Lokální_Prohledávání N(x)
            x = Akceptační_Kritérium (x, y, historie)
        while (ukončovací podmínka)
    end
```

Nejdříve se vygeneruje počáteční řešení (obvykle náhodně). Na tomto řešení se poté aplikuje procedura lokálního prohledávání. Funkce modifikace zajistí změnu řešení *x* za pomoci historie prohledávání (porovnají se výsledky zaznamenané v průběhu chodu algoritmu) a funkce vrátí nové řešení *x*. Toto nové řešení je poté vylepšeno lokálním prohledáváním a pak vrátí řešení *y*, které je novým lokálním minimem. Toto nové řešení je následně porovnáno s řešením *x* a spolu s informacemi, které byly zachyceny v průběhu historie algoritmu je rozhodnuto, zda se toto nové řešení zamění se starým nebo ne. Pokud akceptační kritérium schválí toto nové řešení, pak je prohlášeno za nové řešení *x* a tento proces se opakuje až do té doby než se naplní ukončovací podmínka.

Nevýhodou tohoto přístupu je to, že pokud lokální minimum nebo předešlé nejlepší řešení se nenachází dostatečně blízko (z hlediska minimálního počtu přesunů mezi dvěmi řešeními) ke globálnímu minimu, pak tato metoda nebude pravděpodobně lepší (může být i horší) než lokální prohledávání založené na náhodném výběru řešení.

3.2.4 Optimalizace mravenčí kolonie

Optimalizace mravenčí kolonie (Ant Colony Optimization) [7], [14], je algoritmus založený na chování mravenců v kolonii. Princip je následující. Zdrojem mravenců je jejich mraveniště a cílem je nalezení potravy. Mravenci po určité době naleznou optimální (nejkratší) cestu k cíli a po té se pohybují. Tento efekt je dán tím, že mravenci si cestu značkují feromonem. Intenzita feromonu pak ovlivňuje mravencovo rozhodování, kterým směrem se vydá. Pokud například dojde první mravenec k rozcestí, nejdříve se rozhodne náhodně. Při procházení po cestě zanechá stopu a při návratu touto cestou ji označkuje podruhé, jak se zvyšuje počet projití cestou, zvyšuje se i intenzita feromonu. Pokud prochází kratší cestou, trvá mu to kratší dobu a tím se začne zvyšovat intenzita feromonu. Po určité době je intenzita feromonu tak silná, že se přidávají i ostatní mravenci a zesilování tak probíhá i nadále.

Pro účely implementace optimalizačních algoritmů je vytvářen umělý mravenec, jehož vlastnosti jsou upraveny (oproti skutečným mravencům) tak, aby došlo ke zlepšení výsledků algoritmů, které řeší konkrétní problémy. Podobnosti se skutečnými mravenci jsou především v koloniích spolupracujících mravenců, v použití feromonové stopy, v nepřímé komunikaci

mravenců pomocí feromonové stopy a v pravděpodobnostním rozhodování. U umělých mravenců se vyskytují i rozdíly oproti skutečným. Algoritmus si např. pamatuje vnitřní stavy (jakási osobní paměť, která zaznamenává doposud vykonané akce), umělí mravenci nejsou zcela slepí. Jiným rozdílem je to, že množství zanechaného feromonu je funkcí kvality nalezeného řešení.

Mravenci použití v koloniích fungují jako stochastické procedury vytvářející nová řešení interaktivním přidáváním komponent (feromonová stopa) do částečného řešení. Mravenci při výběru každé další komponenty zvažují informaci o řešeném problému, snaží se využít takovou informaci, která vede ke slibnému řešení problému. Jeden konkrétní mravenec bere v potaz zkušenosti získané v průběhu výpočtu ostatními mravenci; tyto informace jsou reprezentovány feromonovou stopou a neustále se vyvíjí.

Feromon je tedy vyjádřen vahou, která je přiřazena dané cestě vedoucí k cíli. Tato váha umožňuje přidávat další feromony od dalších mravenců. V tomto algoritmu je také zohledněn fakt, že se feromony vypařují, pokud cesta není využívána a tím pádem se i snižuje hodnota vah u jednotlivých spojů. Tento fakt pak zvyšuje robustnost optimalizačního algoritmu pro nalezení globálního extrému.

Pseudokód pro algoritmus mravenčí kolonie z [7]:

```
Mravenčí algoritmus(γ, α1, α2, q, R, S) //např. γ=100, α1=0.1,
α2=0.1, q=0.9, R=n/3
begin
     foreach ant i
          xanti= Vygenerováno_Počáteční_Řešení
          xanti= Lokální_Prohledávání(xanti)

     foreach feromonová dráha j, Tj = 1 / (γ* g(x*))
     zesílení = true
     do
          foreach ant i = 1 to n
               for k = 1 to r
                    s_pravděpodobností (q)
                         //exploatace
                         vyber sousední řešení xanti^k z N(xanti)
                         částečně náhodně, částečně takové, že
                         velikost feromonů je maximalizována
```

```
                    else
                          //explorace
                          vyber sousední řešení x_anti^k z N(x_anti)
                          částečně náhodně a částečně pomocí vah
                          směřující k takovým řešením, které mají
                          vysokou hodnotu feromonů

               end for
               x_anti^(r+1) = Lokální_Prohledávání (x_anti^r)
               if zesílení = true
                    x_anti = best x_anti^k pro k = 1 to r+1
               else
                    x_anti = x_anti^k
          if žádné zlepšení v jakémkoliv x_anti
               zesílení = false
          if existuje takové x_anti, že g(x_anti) < g(x*)
               x* = x_anti
               zesílení = true

          //Vypařování
          foreach feromonová dráha T_j = T_j *(1-α1)
          //Zesilování
          foreach feromonová dráha T_b přítomné v řešení x*
               T_b = T_b + α2/g(x*)
          if proběhlo S iterací aniž by se x* zlepšilo
               //Proveď diverzifikaci
          foreach ant i, x_anti = Vygenerováno_Počáteční_Řešení
               foreach feromonová dráha j, T_j = 1 / (γ * g(x*))
               x_anti = x*

while ukončovací podmínka
end
```

Algoritmus začne tím, že každému mravenci přiřadí náhodné řešení a poté jej vylepší za pomoci lokálního prohledávání. Na základě ceny nejlepšího nalezeného řešení se inicializuje feromonová dráha (tedy matice vah) a pak každý mravenec provede určitý počet kroků a to buď exploatací (ke zlepšení hodnoty aktuálního řešení daného mravence) nebo explorací (pro modifikaci aktuálního řešení daného mravence – částečně náhodně, částečně pomocí vah k řešení, které obsahuje vysoké hodnoty feromonů). Výsledné řešení získané modifikací je pak vylepšeno pomocí lokálního prohledávání. Pokud se algoritmus nachází ve fázích zesilování, pak je nejlepší řešení, které je nalezeno v průběhu modifikace a po provedení lokálního prohledávání označeno jako aktuální řešení daného mravence [19]. Toto se opakuje pro každého mravence. Pokud není provedeno žádné zlepšení, fáze zesilování se vypne (prohledávaný prostor v této fázi nebude příznivým pro nalezení optimálního řešení). Pokud je však nejlepší řešení dále zlepšováno, fáze zesilování se naopak zapne (tedy prostor bude zřejmě slibný – fáze zesilování má analogii ve značkování cesty mravenci pomocí feromonů). Nyní má každý element feromonové dráhy svou hodnotu zredukovánu (dochází k simulaci vypařování feromonů na skutečné dráze). V dalším kroku se těm elementům, které jsou obsaženy v nejlepším nalezeném řešení, zvyšuje hodnota jejich feromonů (což vede k zesilování vhodných argumentů řešení). Pokud po určitém počtu iterací již neexistuje žádné další zlepšení nejlepšího nalezeného řešení, pak se algoritmus diverzifikuje přenastavením datové struktury feromonové dráhy a nastavením všech mravenčích řešení (až na jedno) na nové náhodné počáteční řešení. Zbývající mravenec si drží nejlepší nalezené řešení. Tento proces pokračuje do té doby, než není uspokojena nějaká ukončovací podmínka. Toto je pouze jeden z možných příkladů mravenčího algoritmu.

Závěr

Cíl bakalářské práce, který jsem si stanovila, se mně podařilo naplnit. Zvolila jsem si charakteristiku nedeterministicky polynomiálních úloh. Popsala jsem a na některých praktických příkladech ukázala problém batohu, obchodního cestujícího, barvení grafu a některé vybrané algoritmy.

Při této příležitosti jsem si plně uvědomila, o jak velmi složitý teoretický a současně praktický problém se jedná. Orientace v něm, jeho plné pochopení ale zejména možnost chápat všechny souvislosti a praktické naplnění bylo pro mě velmi složité. Na druhé straně současně velmi inspirující. Měla jsem možnost proniknout do celé řady problémů, teorií a názorů, které jsem buď vůbec neznala anebo byly jaksi jen součásti některých aspektů „běžného života“ anebo byly tzv. v pozadí práce na počítači, při studiu a v mé profesní kariéře. Mám konkrétně na mysli např. využívání bezpečného přenosu dat a informací počítačovou sítí, nebo řešení každodenních pracovních a osobních situací, či odborné práce na počítači. Uvědomila jsem si, že mezi tyto okolnosti bezesporu patří situace blízké problému batohu, obchodního cestujícího, grafů či jisté podoby algoritmu a jejich řešení.

K vlastnímu obsahu mé bakalářské práce, s cílem jistého zobecnění, si dovoluji ještě stručně poznamenat následující. Vyvstává otázka: „Je P totéž co NP, nebo ne?“ O velkém množství problémů bylo prokázáno, že jsou NP úplné. To znamená, že matematici mají k dispozici spoustu způsobů, jak se pokusit o důkaz, že P=NP. Stačilo by najít polynomiální proceduru, která by vyřešila kterýkoli NP úplný problém, a vztah P=NP by byl automaticky dokázán. Například polynomiální procedura, která by řešila problém obchodního cestujícího, by byla důkazem toho, že P = NP.

Bakalářská práce ve mně samozřejmě ještě evokuje další laické dotazy a návaznosti. Mezi nejzásadnější patří skutečnost, kde vlastně leží hranice teoretického lidského poznávání tak, aby výsledky byly ještě prakticky použitelné, popř. moderními informačními technologiemi realizovatelné. Myslím, si že velmi záleží na vysoké integraci postupů a jednotlivých řešení hlavních vědních oborů k definovaným problémům. Ať již máme opět na mysli bezpečnost internetových informací či další „dobývání Vesmíru“, např. plánované výpravy a obývání Měsíce, či možný plánovaný let člověka na Mars.

Seznam literatury a použitých zdrojů

[1] ADÁMEK, J.: *Šifrování pomocí velkých prvočísel*. Vesmír 71, 1992, 615

[2] BENETLEY, J.J. Fast algorithms for geometric traveling salesman problems. In ORSA *Journal on Computing 4*, 1997, s. 387-411.

[3] CENEK, P., KLÍMA, V., JANÁČEK, J. *Optimalizace dopravních a spojových procesů*. Vysoká škola dopravy a spojov v Žilině, Žilina, 1994. ISBN 80-7100-197-X.

[4] COVENE, P.,HIGHFIELD,R.: *Šíp času*. 1 vyd. Ostrava: OLDAG, 1995. ISBN: 80-85954-08-7

[5] DEVLIN, K.: *Problémy pro třetí tisíciletí*. 1 vyd. Praha: Argo, Dokořán, 2005. ISBN: 80-7203-739-0, ISBN: 80-7363-016-8

[6] DEMEL, J.: *Grafy a jejich aplikace*, 1 vyd. Praha Academia, 2002, ISBN: 80-200-0990-6

[7] DORIGO, M., MANIEZZO, V. AND COLORNI, A. The Ant System: Optimization by a colony of cooperating agents. In *IEEE Transactions on Systems and Cybernetics - Part B*, Vol. 26, No. 1, 1996. s. 1-13.

[8] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison – Wesley, 1989. 432 s.ISBN 0-2011-5767-5.

[9] KUČERA, L.: *Kombinatorické algoritmy*. 2 vyd. Praha SNTL, 1989. L11-E1-II-84/12065

[10] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J. a kol.*Umělá inteligence (1).* 1 vyd. Praha: Academia, 1993. ISBN: 80-200-0496-3

[11] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J. a kol. *Umělá inteligence (3).* Praha: Academia, 2001. ISBN 80-200-0472-6

[12] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J. a kol. *Umělá inteligence (4).* Praha: Academia, 2001. ISBN 80-200-1044-0.

[13] MITCHELL, M. *An introduction to genetic algorithms.* Cambridge (Massachusetts): The MIT Press, 1998. 221 s. ISBN 0-262-63185-7.

[14] MOSCATO, P. An introduction to population approaches for optimization and hierarchical objective functions: a discussion on the role of tabu search. In *Annals of Operations Research* 41, 1993. s. 85-121.

[15] PELIKÁN, J.:*Diskrétní modely v operačním výzkumu*. 1 vyd. PROFESSIONAL PUBLISHING, 2001. ISBN: 80-86419-17-7

[16] PUDLÁK, P.: *Staré a nové problémy v matematice*. Vesmír 74, 1995, 305-310

[17] SHASHA, D., LAZARE, C.: *Aut of Teir Minds The Lives and Discoveries of 15 Great Computer Scientists* (Bláhovci: Životy a objevy patnácti velkých informatiků), New York, 1998, p. 148

[18] STÜTZLE T. *Iterated Local Search for the Quadratic Assignment Problem*. Research Report AIDA-99-03, Department of Computer Science, Darmstadt University of Technology, Germany, 1999.

[19] WANG, F.Y., LIU, D. *Advances in Computational Intelligence- theory and applications*. Singapore: WSP, 2006. ISBN 981-256-734-8

Seznam obrázků

Seznam tabulek

[i] Údaje v tabulce nejsou přesné, slouží pouze pro ukázku výpočtu