Univerzita Pardubice

Fakulta ekonomicko-správní

Paralelizace úloh pomocí vybraného optimalizačního algoritmu

Martin Svoboda

Diplomová práce

2012

Univerzita Pardubice
Fakulta ekonomicko-správní
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martin Svoboda**
Osobní číslo:	**E09812**
Studijní program:	**N6209 Systémové inženýrství a informatika**
Studijní obor:	**Informatika ve veřejné správě**
Název tématu:	**Paralelizace úloh pomocí vybraného optimalizačního algoritmu**
Zadávající katedra:	**Ústav systémového inženýrství a informatiky**

Zásady pro vypracování:

Cílem práce je návrh a implementace paralelizace vybraného problému pomocí zvolených stochastických optimalizačních algoritmů.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

AARTS, E., LENSTRA, J. K. *Local Search in Combinatorial Optimization.* **New York: Wiley, 1997. 551 s. ISBN: 0-691-11522-2.**

LAARHOVEN, P.J.M. Van, AARTS, E. *Simulated Annealing: Theory and Applications.* **New York: Springer, 1987. 204 s. ISBN: 90-277-2513-6.**

BURKE, E., KENDALL G. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques.* **New York: Springer. 2005. 620 s. ISBN 978-0387-23460-1.**

MACKEY, A. *Introducing .NET 4.0: with Visual Studio 2010.* **New York: Springer. 2010. 400 s. ISBN: 978-14302-2455-6.**

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

Datum zadání diplomové práce: **3. října 2011**

Termín odevzdání diplomové práce: **30. dubna 2012**

L.S.

doc. Ing. Renáta Myšková, Ph.D.
děkanka

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

V Pardubicích dne 3. října 2011

Prohlašuji:

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

Byl jsem seznámen 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.

V Pardubicích dne 29. 6. 2012

Martin Svoboda

Poděkování

Děkuji Ing. Janu Panušovi, PhD. za hodnotné rady a odborné vedení během mé práce.

ANOTACE

Tato práce se zabývá paralelním zpracováním algoritmů pro úlohy vycházejících z problému obchodního cestujícího. Jsou zde vysvětleny některé heuristické a metaheuristické metody, včetně návrhu jejich paralelizace. Vybrané metody jsou implementovány v prostředí Visual Basic .Net a podrobeny šetření z hlediska kvality nalezených řešení a efektivity paralelizace.

KLÍČOVÁ SLOVA

Problém obchodního cestujícího, heuristika, metaheuristika, genetické algoritmy, paralelizace

TITLE

Paralleled problem by chosen optimization algorithm

ANNOTATION

This work deal with parallel process algorithm for tasks related to the travelling salesman problem. Some heuristic and metaheuristic methods including their parallel design are described. Chosen methods are implemented in Visual Basic .NET environment and research in term of quality solution and effectiveness of parallelization.

KEYWORDS

Traveling Salesman Problem, heuristic, metaheuristic, genetic algorithm, parallelization

Obsah

ÚVOD

Problém obchodního cestujícího je jedním z široce studovaných problémů v kombinatorické optimalizaci. Cílem této diplomové práce je provést návrh algoritmů metod pro paralelní zpracování problému obchodního cestujícího a jejich implementaci v prostředí Visual Basic .NET. Tyto algoritmy budou testovány z hlediska kvality nalezených řešení a porovnány s jejich sekvenčními ekvivalenty co do rychlosti výpočtu.

Pro účely paralelizace byla zvolena knihovna TPL (Task Parallel Library), která usnadňuje programátorovi implementaci simultánního zpracování. TPL má ve své režii kompletní řízení vláken a tím odpadá obtížná práce programování s vlákny.

Po seznámení s variantou TSP a určení způsobu porovnání jednotlivých algoritmů bude představeno několik stochasticky heuristických metod a v některých případech jejich modifikace v kombinaci s deterministickými přístupy.

První část algoritmů se zaměřuje na základní heuristiku nejbližšího souseda (NN), jeho implementaci ve vývojovém prostředí na úrovni simultánního zpracování a následnou modifikaci s hloubkovými metodami prohledávání Sequential Fan a Filter & Fan. Jak byly algoritmy testovány na větších instancích, nabízela se zde možnost datové dekompozice a tím i další způsob použití paralelizace, což dalo za vznik dalšímu algoritmu kombinující shlukovou analýzu a heuristiku 2-opt.

Druhá část je věnována podskupině evolučních algoritmů, tzv. genetických algoritmů (GA). V této části je popsáno několik metod běžně používaných v GA, mezi něž patří například selekce, mutace nebo křížení. Po vytvoření sekvenčního algoritmu je popsáno několik způsobů, jakými lze provést na této metaheuristice paralelizaci a zároveň je pomocí stanovených ukazatelů provedeno testování a porovnání jednotlivých algoritmů.

1 Problém obchodního cestujícího

Cílem problému obchodního cestujícího, anglicky Traveling Salesman Problem (zkratka TSP), je nalézt nejkratší trasu obchodnímu cestujícímu, který začíná ve výchozím místě, navštíví předepsaná města právě jednou a až projde všechna, vrátí se na místo výchozí. [1]

1. 1 Matematická definice

Formální matematická definice TSP vychází ze základní teorie grafů, kde G = (V,E) je graf (orientovaný i neorientovaný), V je množina tzv. vrcholů, E množina dvojic vrcholů V (hran) a F je množina všech Hamiltonovských kružnic (cest) v G. Pro každou hranu e patřící do množiny E existuje hodnota (váha) c_e. Pak je problém obchodního cestujícího určen jako nalezení takové Hamiltonovské kružnice (cesty) v G, aby součet vah hran cesty byl minimální. [2]

1. 2 Varianty TSP

Uvažujeme o dvou případech varianty TSP – symetrické a asymetrické.

Symetrická varianta – v tomto případě se jedná o neorientovaný graf, kde TSP má symetrickou matici nákladů C. Platí zde pravidlo $c_{ij} = c_{ji}$, tudíž nezáleží na tom, jakým směrem se pohybujeme v rámci vrcholů, a zároveň platí, že $c_{ii} = 0$ pro i = 1,n, neexistuje tedy hrana z uzlu do sebe samého. [2]

Asymetrická varianta – pravidlo $c_{ij} \neq c_{ji}$ určuje, že mezi jednotlivými uzly existují různá ohodnocení v rámci dvojic.

Tato práce bude zaměřena na TSP ve variantě symetrické, kde vždy existuje hrana mezi dvojicí měst, resp. zde budeme pracovat s Euklidovskou variantou TSP[1]:

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

pro dvourozměrný prostor, kde c_{ij} je vzdálenost (hrana) mezi vrcholy i a j.

Po volbě varianty můžeme určit matematickou definici jako minimalizaci účelové funkce [2]:

$$\sum_{e \in E} c_e x_e$$

za podmínek:

$$\sum_{e \in \delta(v)} x_e = 2, \quad v \in V \qquad (1)$$

$$x_e = 0 \; nebo \; 1, e \in E$$

$$\{e \in E : x_e = 1\} \qquad (2)$$

kde δ(v) je množina hran vrcholu v z množiny vrcholů G, přičemž podmínka (1) stanovuje, že každá hrana má právě dva uzly. Druhá podmínka zajišťuje, že pro každý vrchol bude použita právě jedna hrana, tzn. pokud je již vrchol zahrnut v Hamiltonovské kružnici, nelze opětovně vybírat z podmnožiny hran daného vrcholu.[2]

1. 3 Náročnost výpočtů

Na TSP se můžeme dívat jako na permutační problém, kdy pro n měst symetrické varianty existuje (n - 1)!/2 řešení. V tabulce 1. je uveden příklad počtu řešení pro daný počet měst.

Tabulka 1. Počet kombinací cest pro n měst

Počet měst n	Počet možných cest
3	1
4	3
5	12
6	60
7	360
8	2 520
9	20 160
10	181 440
11	1 814 400
12	19 958 400
13	239 500 800
14	3 113 510 400
15	43 589 145 600
16	653 837 184 000
17	10 461 394 944 000
18	177 843 714 048 000
19	3 201 186 852 864 000
20	60 822 550 204 416 000

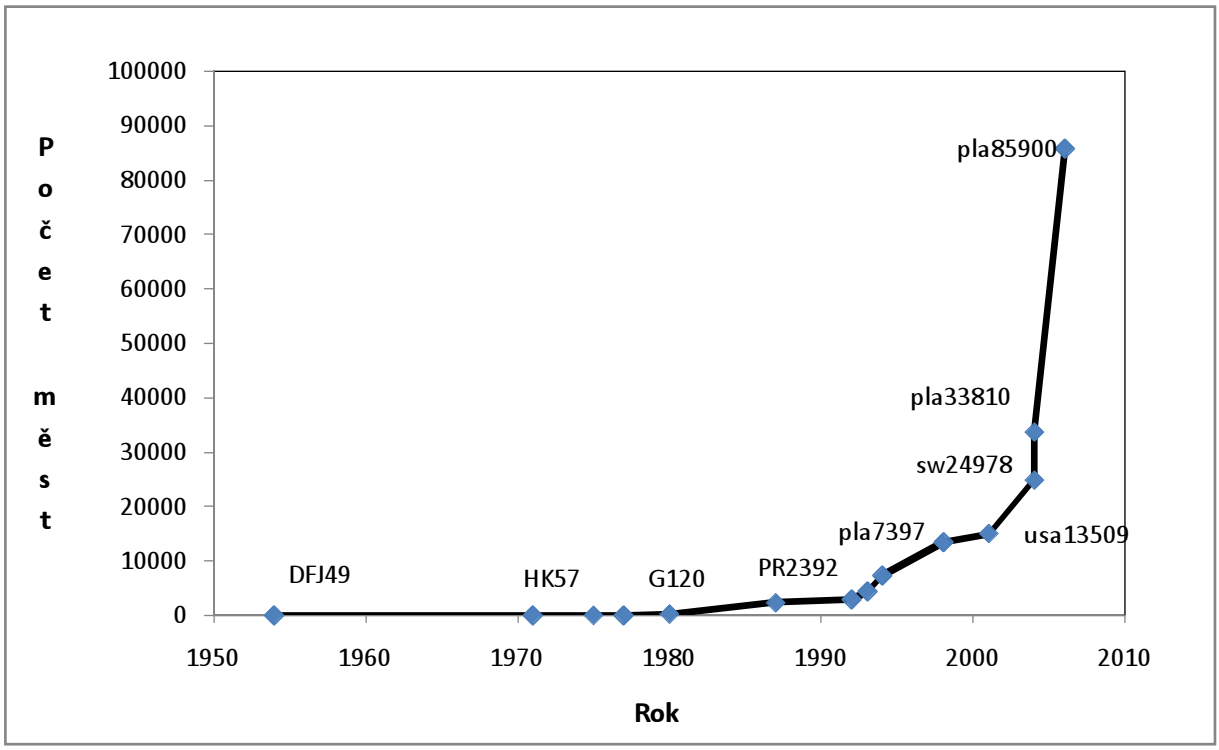
Jak je zřejmé z tabulky 1., při počtu měst vyšším než 12 enormně roste počet možných výsledků a tím i výpočetní čas. Pokud bychom chtěli provést kompletní výpočet variant cest

například pro 20 měst, museli bychom spočítat více než 60 biliard, tzn. 60^{15} kombinací. Proto je pro větší počet měst kompletní výpočet nepoužitelný. [3] TSP tímto spadá do skupiny úloh, u kterých není znám algoritmus řešící úlohu v polynomiálním čase, tzv. NP-úplné.

1. 4 Milníky v TSP

Jedna z prvních literatur souvisejících s TSP byla zveřejněna v roce 1757 Leonhardem Eulerem. Ten ve své práci, odkazujíce se na cesty a okruhy, řeší sekvenci tahů šachového koně tak, aby prošel přes všechna pole a vrátil se na počátek.

Jako s výzkumem tohoto problému pak přišel v roce 1930 Merrill Flood z Princetonské univerzity. Ten se snažil vyřešit problém nejkratší trasy školního autobusu pro 48 zastávek. Průlomový byl rok 1954, kdy G. Dantzig, R. Fulkerson a S. Johnson vyřešili problém se 48mi městy v efektivním čase. Další údaje uvedené v obrázku 1. zaznamenávají vyřešení problému na konkrétním počtu měst v jednotlivých letech. Charakter křivky odpovídá exponenciálnímu průběhu, což je zapříčiněno použitím sofistikovanějších metod v kombinaci s rostoucím výpočetním výkonem. [1]



Obrázek 1. Milníky TSP, převzato z [1]

Jednotlivé hodnoty na křivce grafu určují vyřešení instance pro daný počet měst na konkrétním problému. Nelze tedy tvrdit, že byla nalezena optimální hodnota obecně pro určitý počet vrcholů.

1. 5 Knihovna TSPLIB

Abychom mohli otestovat kvalitu námi vytvořených algoritmů, bude v této práci použita knihovna instancí problémů nazývaná TSPLIB. Tato knihovna je dostupná na adrese http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ [TSPLIB], obsahuje desítky vstupních souborů pro testování, u některých je uvedeno i optimální řešení.

2 Paralelizace úloh

V době psaní této práce jsou už i běžné domácí počítače osazeny procesorem s více jádry. Vícejádrovým procesorem se rozumí čip, který obsahuje dvě nebo více nezávislých CPU (Central Process Unit), které provádějí instrukce programu. Tento způsob architektury umožňuje zpracování k-násobného množství instrukcí za jednotku času, kde *k* značí počet jader.[4]

Program, který je napsán sekvenčním způsobem, bude prováděn řádek po řádku, tedy tak, jak byl napsán. Pokud bychom převedli práci počítače na běžnou situaci ze života, mohli bychom použít kupříkladu vykopání příkopu. Zatímco jeden kopáč vykoná práci za určitý čas, deset kopáčů provede tuto činnost desetkrát rychleji. Tedy za předpokladu, že bude každý vědět, co má dělat. Už samotná dekompozice, zde určení, jakou část má každý pracovník vykonat, zabere určitý čas. Stejně jako na uvedeném příkladu, i výpočetní úlohu lze rozdělit na více částí a tím zmenšit čas běhu. Důležité je určit, zda čas vynaložený na správu (rozdělení, řízení ad.) simultánní činnosti a provedení činnosti není vyšší než její sekvenční zpracování. Dalším příkladem ze života může být vykopání studny. Ač je činnost podobného charakteru, nelze zde zapojit naráz větší množství kopáčů a tím provádět simultánní vypracování. To je důkaz toho, že ne všechny úlohy lze dekomponovat a tím na nich provádět paralelní zpracování. Při vytváření paralelních programů bude tedy nutné dobře určit, zda a jaké části problému jsou vhodné pro samotnou paralelizaci.

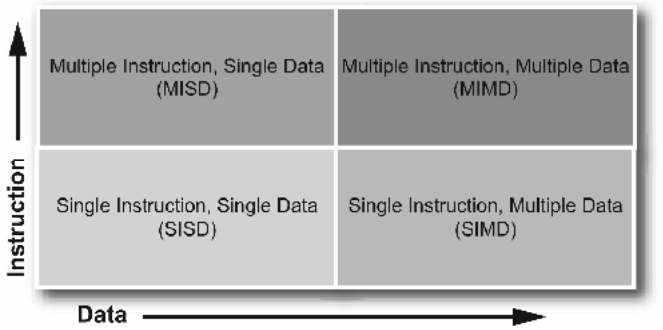
2. 1 Rozdělení architektur

K tomu, abychom mohli provádět paralelní softwarové výpočty, je nutné mít technické vybavení, které podporuje simultánní výpočty na úrovní více vláken. Všeobecně řečeno, architekturu počítače můžeme rozdělit do dvou skupin [5]:

1) počet instrukcí, které je schopen počítač zpracovat v jednom taktu

2) počet proudů dat, zpracovaných v jednom taktu

Tento způsob klasifikace je znám jako Flynnova taxonomie, znázorněná na obrázku 2.

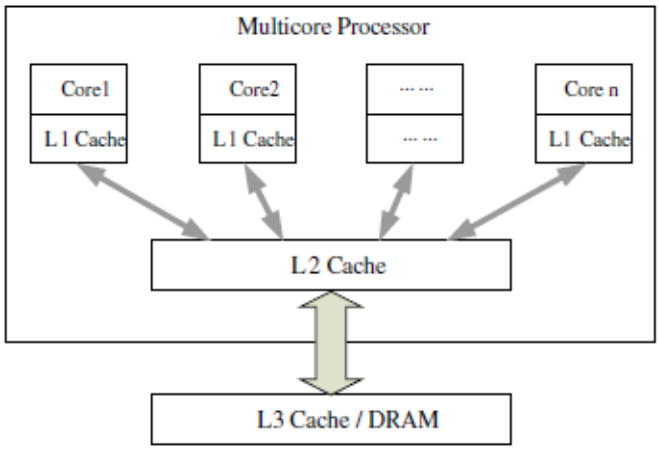


Obrázek 2. Flynnova taxonomie, převzato z [6]

Počítače spadající do kategorie SSID nedokážou na hardwarové úrovni zpracovat simultánně více dat. Proud dat, stejně tak i zpracování instrukcí, zde probíhá sekvenčně. Jednalo se o masově rozšířené 8mi bitové počítače 80.let jako Commodore nebo původní PC IBM[6].

Do kategorie SIMD patří původně superpočítače nebo vektorové procesory. Některé formy SIMD instrukcí se začaly objevovat v běžných počítačích na konci minulého století. Šlo o implementaci instrukcí nazvaných MMX, SSE1-3[6]. Ty dokázaly zpracovat v jednom cyklu dva a více proudů dat, v souvislosti na vývoji dané architektury. Skupina MISD je zde uvedena spíše jako teoretická, než praktická, vzhledem k jeho nasazení do praxe.

Do poslední skupiny spadají i vícejádrové počítače, které zpracovávají v jednom taktu více instrukcí v kombinaci s vícenásobnými daty. V současnosti se jedná o nejběžnější platformu paralelního počítače. Základní architektura vícejádrového systému je uvedena na obrázku 3.

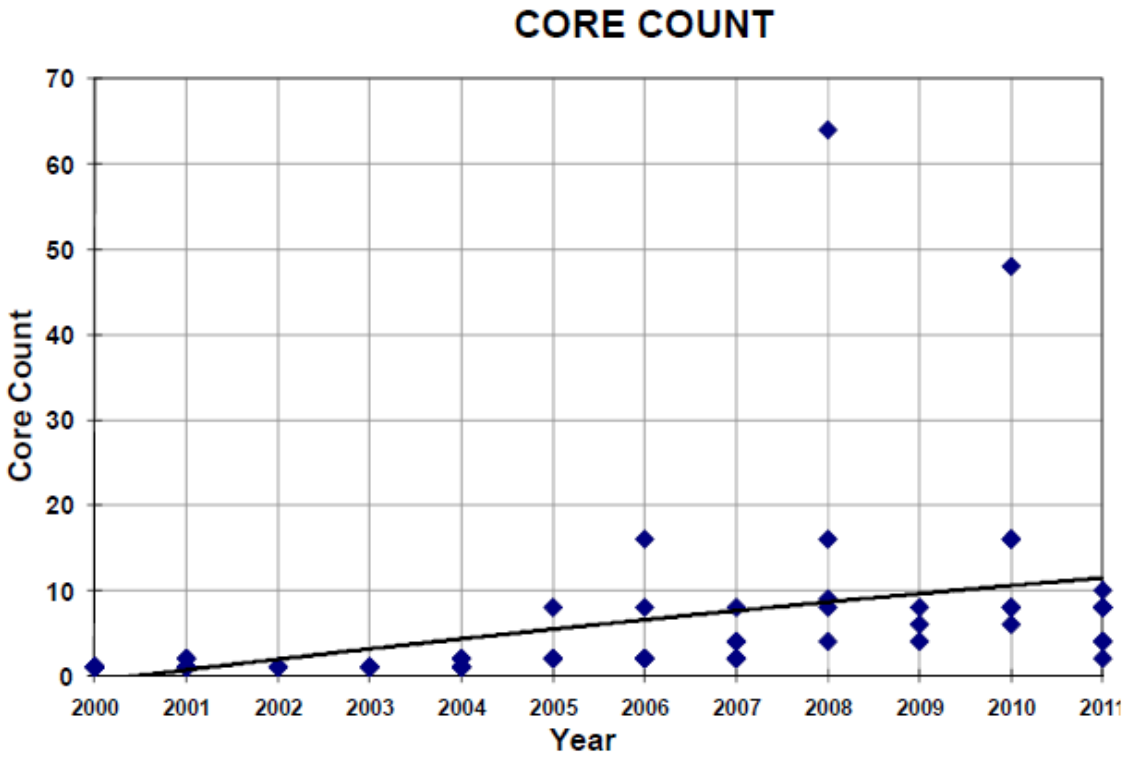


Obrázek 3. Architektura vícejádrového systému, převzato z [7]

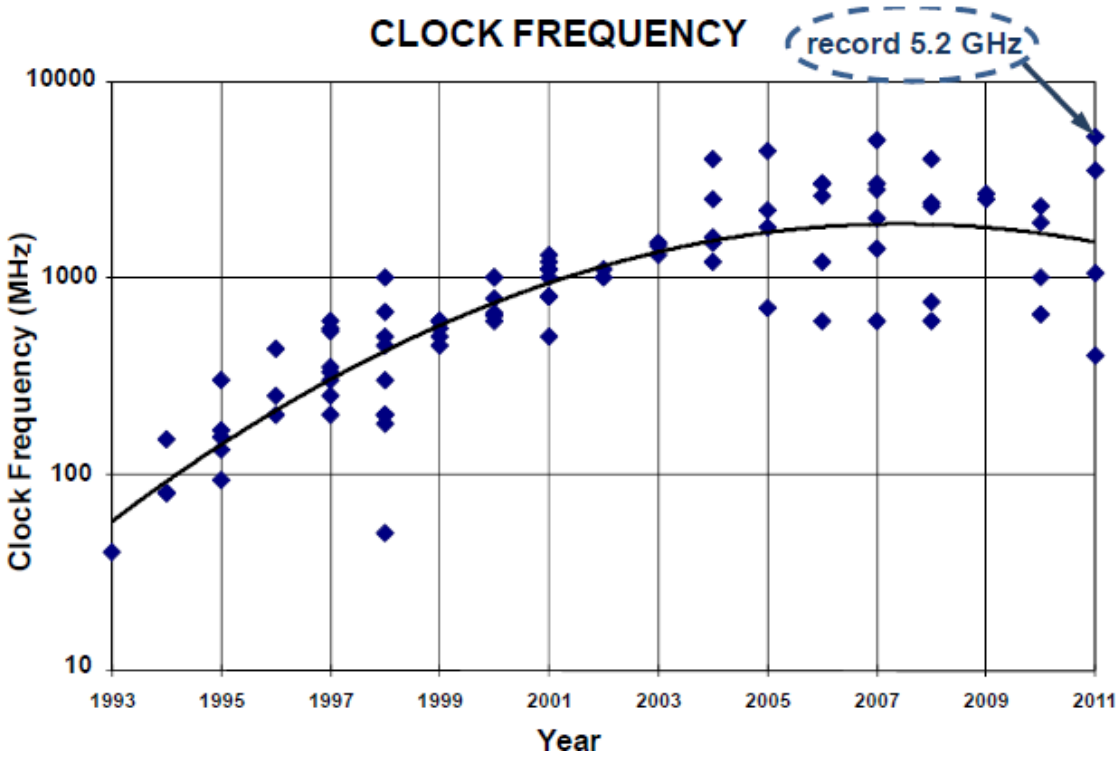
Z obrázku 3. vyplývá, že každé jádro má určenou vlastní L1 vyrovnávací paměť a zbývající úrovně v paměťové hierarchii jsou sdílené pro všechna jádra.

2. 2 Vývoj vícejader

Vícejádrové procesory těží ze základního vztahu mezi elektrickou energií a kmitočtem. Zapojením více jader může každé jádro běžet na nižší frekvenci a rozdělením úlohy mezi ně pak zvyšuje celkový výkon daného procesoru. [8] V minulosti, tedy od 60. let 19. století až do počátku století současného, se vývoj procesorů zaměřoval na zvyšování frekvence samotného procesoru v kombinaci s miniaturizací čipu. I přes zmenšování tranzistorů docházelo v důsledku zvyšování kmitočtu k nárůstu odpadního tepla. To vedlo k omezujícím hranicím nárůstu výkonu jednojádrových procesorů. Vývoj vícejádrových procesorů započal začátkem tisíciletí a počet jader postupně narůstal, jak je zřetelné z obrázku 4. Osa x znázorňuje jednotlivé roky a osa y počet jader vyvinutých v daném roce. Na obrázku 5. je zachycen vývoj frekvence procesoru v jednotlivých letech. Zde je patrné, že frekvence procesoru začala od roku 2007 stagnovat.



Obrázek 4. Vývoj vícejader, převzato z [9]



Obrázek 5. Vývoj frekvence procesoru, převzato z [9]

Implementace vícejader do běžných počítačů započala firma Intel v roce 2006, kdy vydala procesor s dvěma jádry. V době psaní této práce jsou již běžně k dostání na trhu procesory s osmi jádry, které dokáží v jednom taktu zpracovat až 16 vláken – jedná se například o Intel® Xeon® Processor X6550.

2. 3 Hardware pro testování

Pro účely testování budou v této práci využity následující sestavy:

Sestava č. 1:

- Procesor Intel® Pentium® Dual Core T2390 (1,86 GHz), 1MB Cache
- Paměť 2GB DDR2 667 MHz
- 32-bit operační systém Windows XP SP3

Sestava č. 2:

- Procesor AMD Phenom II X6 1100T Black Edition, socket AM3, 3.3 GHz, 3+6MB
- Paměť 8GB DDR3?
- 64-bit operační systém Windows 7 Ultimate SP1

Sestava č. 3:

- Procesor Intel 2600k @3800Mhz
- Paměť 8GB DDR3? 1600MHz
- 64-bit operační systém Windows 7 Ultimate SP1

Zajímavostí procesoru ve třetí sestavě je nová funkce Intelu nazvaná Hyper-threading. Ta dokáže vytvořit v závislosti na počtu jader dvojnásobný počet jader virtuálních. Touto metodou se dle informací uvedených na www.intel.com zvýší výpočetní výkon o cca 30%. Pro účely tohoto šetření, a obzvláště pak pro způsob porovnávání efektivity, jehož výsledek závisí na počtu jader, budou veškeré výpočty prováděny s vypnutým hyper-threadingem.

2. 4 Měření rychlosti paralelního zpracování

V této části práce řešíme problém, jakým způsobem lze měřit nárůst výkonu při paralelním zpracování. Pokud budeme vycházet z teze, že můžeme rozdělit různé úlohy do procesů a ty zpracovat současně v jednom taktu, mělo by dojít k výraznému zvýšení výkonu. V případě úplné nezávislosti úloh je výkonová výhoda jasná. Jednou z metrik, jak určit nárůst výkonu, je srovnávat uplynulou dobu nejlepšího sekvenčního algoritmu s uplynulou dobou výpočtu paralelního zpracování uvedeného ve vztahu[5]:

$$Speedup(SU) = \frac{Time_{best\ sequential\ algorithm}}{Time_{parallel\ implementation}}$$

Tento koeficient je znám jako „zrychlení“ (SU) a charakterizuje, o co je rychlejší program běžící paralelně. Nárůst vychází z počtu fyzických vláken použitých při výpočtu.[5] Výkonnost paralelního programu pak může být[10]:

- sublineární, pokud SU < n
- lineární, pokud SU = n
- superlineární, jestliže SU > n

kde *n* je počet jader procesoru. V následujících kapitolách této práce bude v rámci porovnání sekvenčního a paralelního algoritmu použito parametru *efektivita*, který určíme dle matematického vztahu [10]:

$$e = \frac{SU}{n} * 100\ [\%]$$

kde:

e je efektivita algoritmu uvedená v procentech

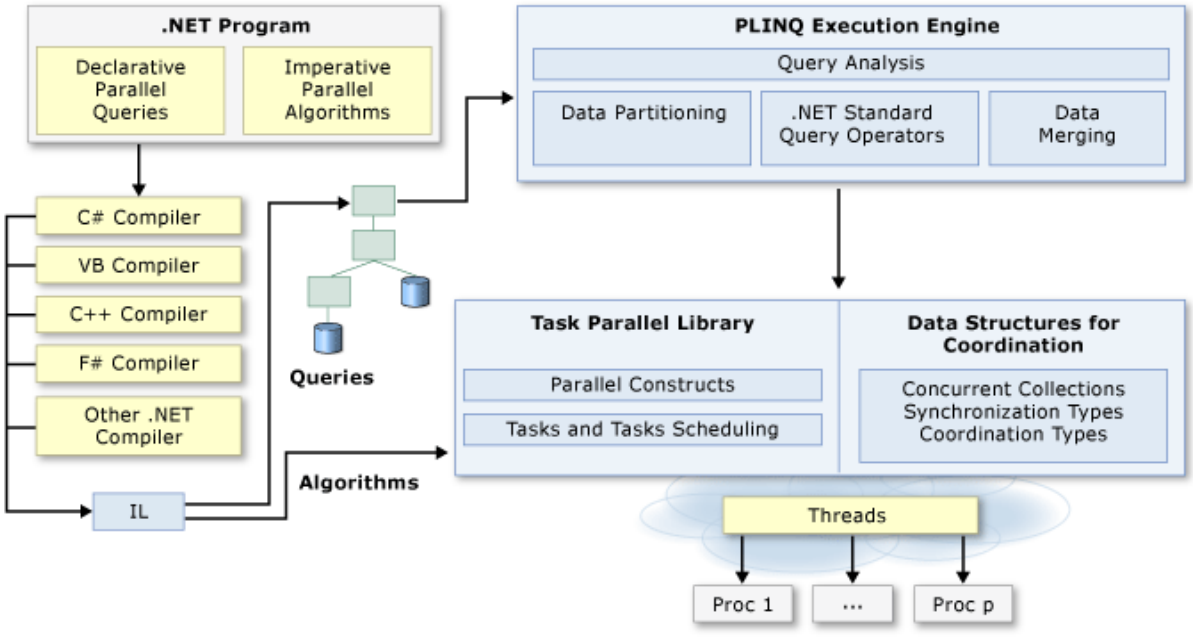
SU nárůst výkonnosti paralelního programu, tzv. „zrychlení“

n je počet jader procesoru

Dosahuje-li efektivita hodnoty 100/*n* procent, je algoritmus stejně rychlý jako jeho sekvenční zpracování. V případě efektivity v rozmezí 100/*n* až 100 % dosahujeme sublineárního zrychlení, přičemž ideální by bylo dosáhnout horní hranice. K superlineárnímu nárůstu dochází zcela výjimečně, kupříkladu z důvodů rychlejšího přístupu k datům díky jejich dekompozici, kdy jsou údaje brány z rychlejší cache paměti.[4]

3 Softwarové prostředí

Ve svém vývojovém prostředí Visual Studio 2010 and the .NET Framework 4 reagovala společnost Microsoft na rozmach více jader rozšířením podpory pro paralelní programování. Zavedla nový runtime, třídy knihoven a diagnostické nástroje. [11] Tyto vlastnosti zjednodušují vývojářům psaní paralelních kódů bez nutnosti přímého psaní aplikací přizpůsobované na vlákna. Celkový přehled architektury paralelního zpracování ilustruje obrázek 6.



Obrázek 6. Architektura .NET Framework 4.0, převzato z [11]

V této práci budeme pracovat s programovacím jazykem Visual Basic .NET. Po spuštění kódu dojde ke kompilaci ve VB, poté dojde k převodu do pomocného jazyka IL, známého jako Microsoft Intermediate Language. Odtud se pak kód nižší úrovně v IL zpracovává v knihovně paralelních úloh (TLP) a následně je vykonán na úrovni vláken.

TPL je sada veřejných typů v třídě System.Threading a System.Threading.Tasks. Účelem této knihovny je zjednodušení psaní procesů pro vývojáře a tím zvyšování efektivity programování. TLP řeší způsob přidělování procesů všem dostupným jádrům, rozdělování vláken a jejich rušení. [11]

3. 1 Paralelní smyčky

Souběžnost dat se odkazuje na situace, ve kterých je stejná operace provedena souběžně na prvcích souboru nebo pole. V paralelním zpracování dat je zdrojový soubor, resp. matice, rozdělen na části, které jsou následně zpracovány vícenásobnými vlákny, podporován třídou System.Threading.Tasks.Parallel. Implementace je dána syntaxí smyček Parallel.For nebo Parallel.ForEach, což je ekvivalent pro běžný zápis použití sekvenčních smyček For a ForEach. Základní zápis bude vypadat následovně:

```
Parallel.For(0, max, Sub(p)
     'provedeni paralelniho kodu
End Sub)
```

Základní syntaxe je tedy stanovena jako provedení funkce, resp. subprocedury, pro všechna *p* od parametru dolní hranice (0) včetně do parametru hranice horní (*max*). Na příkladu bude kód ve smyčce proveden celkem (*max* – 0) – 1 krát. Jelikož Parallel.For pracuje s horní stanovenou hranicí jako otevřeným intervalem. Pro funkčnost paralelního zpracování tedy není třeba přímo vytvářet vlákna, ani řešit způsob zamykání vláken v rámci smyček. Tento způsob simultánních výpočtů bude použit v následujících kapitolách, při konstrukci Hamiltonovské kružnice v uplatnění metod lokálního prohledávání nebo metaheuristik.

4 Počáteční výpočty

4. 1 Načtení souboru

Prvotní výpočty provedeme na příkladu deseti měst. Každé město je určeno souřadnicemi v dvourozměrném poli, které je v našem případě uloženo v souboru. Nyní je třeba hodnoty ze souboru načíst do polí pro další výpočty. Základní algoritmus pro načtení souboru vychází z třídy StreamReader a její metody Readline (načtení řádku). Soubor je třeba mít ve tvaru číslo města, osa x, osa y, přičemž hodnoty budou odděleny středníkem „;“.

Kód pro načtení ve Visual Basicu je následující:

```
Dim souborvstup As New IO.StreamReader("c:\tsp\soubor.txt")
   ReDim polevstup(pocetradku - 1, 1)
   Dim polevypis As String = ""
   For i As Integer = 0 To pocetradku - 1
       Dim text As String = souborvstup.ReadLine()
       Dim rozdeleni() As String = text.Split(";")
       For j As Integer = 0 To 1
           polevstup(i, j) = CDbl(rozdeleni(j + 1))
           polevypis = polevypis & " " & polevstup(i, j)
       Next
   Next
   souborvstup.Close()
```

Výsledkem je načtení dvourozměrného pole o velikosti (2,x), přičemž x je rovno počtu měst, v našem případě je tedy pole o velikosti (2,10) znázorněné v tabulce 2.

Tabulka 2. Matice souřadnic pro 10 měst

Město	1	2	3	4	5	6	7	8	9	10
X	1150	630	40	750	750	1030	1650	1490	790	710
Y	1760	1660	2090	1100	2030	2070	650	1630	2260	1310

4. 2 Matice vzdáleností

Pro určení ohodnocení grafu mezi městy je třeba vytvořit matici, tzv. matici vzdáleností C = (c_{ij}) n*n, kde i,j-tý záznam koresponduje s hodnotou mezi hranami i a j v G. V závislosti na povaze matice ohodnocení je dle [1] TSP rozdělen na dvě třídy: Pokud je C symetrická (tzn. G je neorientovaný), pak se TSP nazývá symetrický problém obchodního cestujícího. Opakem je pak asymetrický TSP, kde je C asymetrická a graf G je ekvivalentně orientovaný. V našem případě budeme řešit úlohu symetrickou.

Vytvoření matice vzdáleností pro n měst je záležitostí dvojitého for cyklu, přičemž provádíme výpočet Euklidovské vzdálenosti. Suma výpočtů vzdáleností je rovna počtu kombinací bez opakování vybráním vždy dvojice pro 10 měst, obecně tedy:

$$C_k(n) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

```
For i As Integer = 0 To rozmer - 1
    maticevzdalenosti(i, i) = 0
    For j As Integer = i + 1 To rozmer - 1
        x1 = polevstup(i, 0)
        x2 = polevstup(j, 0)
        y1 = polevstup(i, 1)
        y2 = polevstup(j, 1)
        vzdalenost = Math.Sqrt(Math.Pow(x1 - x2, 2) +
        Math.Pow(y1 - y2, 2))
        maticevzdalenosti(i, j) = vzdalenost
    Next
Next
```

Po provedení algoritmu dojde k vytvoření matice vzdáleností uvedené v tabulce 3. Ve VB se bude jednat o dvourozměrného pole, jehož indexy mají hodnotu město – 1, neboť ve Visual Basicu je pole indexované od hodnoty 0. [12]

Tabulka 3. Matice vzdáleností pro 10 měst

Město	1	2	3	4	5	6	7	8	9	10
1	0	530	1158	772	483	332	1217	364	616	629
2		0	730	573	389	573	1435	861	621	359
3			0	1218	713	990	2160	1521	769	1028
4				0	930	1010	1006	910	1161	214
5					0	283	1648	841	233	721
6						0	1549	637	306	825
7							0	993	1825	1149
8								0	942	843
9									0	953
10										0

5 Konstruktivní heuristiky

Heuristika je metoda, která hledá velmi dobrá (myšleno blízko optimálních nebo i optimální) řešení v rozumném výpočetním čase. Jelikož heuristika neprochází celý stavový

prostor, nelze garantovat, že nalezené řešení je optimální. Stejně tak nelze zaručit, že bude nalezena optimální hodnota. [3]

Konstruktivní heuristika spočívá ve vytváření cesty od počátku přidáváním vybraných uzlů, dokud nejsou vybrány všechny vrcholy. Tyto algoritmy jsou specifické tím, že se zastaví ve chvíli, kdy je nalezeno řešení a dále se již nesnaží výsledek zlepšit.

5. 1 Metoda nejbližšího souseda

Metoda nejbližšího souseda patří k nejpřirozenějším konstruktivním heuristikám. Vybíráme vždy takový vrchol, který je nejblíže ke zvolenému aktuálnímu vrcholu. Základní problém této metody spočívá v tom, že čím více vrcholů je zahrnuto v konstruované cestě, tím víc roste pravděpodobnost na zahrnutí vrcholu s příliš vysokou cenovou hodnotou. Pseudokód pro vytvoření Hamiltonovské kružnice pomocí NN je následující:

1) Náhodně vyber vrchol
2) Označ ho jako „použitý“
3) Spočítej vzdálenosti k ostatním vrcholům, které nejsou „použity“
4) Vyber vrchol s nejkratší vzdáleností
5) Označ ho jako „použitý“
6) Opakuj od bodu 3, dokud nejsou všechny vrcholy „použity“
7) Spoj poslední vrchol s prvním

Na tomto základním prohledávání, resp. nalezení lokálního minima, nastíníme způsob, jakým budeme v dalších kapitolách používat paralelizaci výpočtů pro konstruktivní heuristiky.

Pro kontrolu, zda je vrchol již použit, bude nezbytné zavést pomocné jednorozměrné pole s hodnotou boolean (pravda/nepravda). Na příkladu, kdy bude jako náhodné město zvoleno číslo 5, bude z tohoto vrcholu nejkratší cesta 7, 10, 4, jejíž vrcholy po čtyřech průchodech budou zaznamenány v pomocné matici typu boolean, reprezentované tabulkou 4.

Tabulka 4. Pomocná matice projitých měst

Město	1	2	3	4	5	6	7	8	9	10
Použito T/F	0	0	0	1	1	0	1	0	0	1

Na náhodně zvoleném vrcholu také závisí hodnota vypočtené celkové cesty. Pokud bychom na začátku stanovili, že základní algoritmus spustíme kupříkladu šestkrát, použili bychom nad tímto algoritmem cyklus s pevným počtem opakování pro 6 hodnot. Na tento

způsob by nám postačilo vytvořit navíc pole, do kterého zaznamenáme celkový výsledek jednotlivých cest, tedy jednorozměrné pole o šesti prvcích, pomocné pole typu boolean by zůstalo zachováno. Pokud ovšem použijeme paralelizaci, bude nutné rozšířit toto pomocné pole na dvojdimenzionální, jelikož výpočty a taktéž kontrola, zda byl vrchol již navštíven, bude probíhat simultánně, resp. nezávisle.

V případě dvoujádrového procesoru, kdy bude náhodně vybrán vrchol 5 a následující vrcholy budou vypočítány jako v odstavci výše pro jádro č. 1 a pro druhé jádro náhodný vrchol 8 s vypočtenými vrcholy 2, 10, 4, bude v programu inicializováno pole o rozměru (9,1), přičemž první řádek koresponduje s hodnotami v tabulce 5, v druhém řádku budou zaznamenány hodnoty pro druhé jádro.

Tabulka 5. Pomocná matice 10ti měst pro paralelní zpracování

Město	1	2	3	4	5	6	7	8	9	10
Použito T/F	0	0	0	1	1	0	1	0	0	1
	0	1	0	1	0	0	0	1	0	1

Algoritmus pro výpočet Hamiltonovské kružnice pro p možností je následující:

```
Parallel.For(0, pocetpokusu, Sub(p)
    Dim temp As Integer = anahoda(p)
    Dim i As Integer = temp
    Dim boolmin As Integer = i
    maticebool(boolmin, p) = True
    maticevysledekindex(0, p) = temp
    For m As Integer = 1 To rozmer - 1
        Dim min As Integer = 1000000
        Dim h As Integer
        For j As Integer = 0 To rozmer - 1
            h = maticevzdalenosti(i, j)
            If h < min Then
                If maticebool(j, p) = False Then
                    min = h
                    boolmin = j
                End If
            End If
        Next
        i = boolmin
        maticebool(boolmin, p) = True
        maticevysledek(m - 1, p) = min
        maticevysledekindex(m, p) = boolmin
    Next
    maticevysledek(rozmer - 1, p) = maticevzdalenosti(temp, i)
End Sub)
```

kde jsou jednotlivé proměnné určeny následovně:

anahoda(p) – pole, jehož velikost je určena počtem pokusů a do kterého se v prvotní fázi zaznamenává náhodné číslo reprezentující výchozí vrchol

boolmin(p) – jednorozměrné pole, jehož hodnota pro každý pokus reprezentuje index výsledku v rámci řádku

maticebool – pomocná matice typu boolean, jejíž prvky indexu určují, zda daný vrchol byl projit

maticevysledek, maticevysledekindex – slouží k zaznamenání údajů (číslo města, délka cesty) při procházení a následné sumarizační výpočty

maticevzdalenosti – zdrojová matice obsahující délku cesty mezi jednotlivými městy

5. 2 Hloubkové metody

Metody nazývané jako „proměnné hloubkové metody“ (Variable depth methods) [2] vychází z prohledávání a vytváření kombinací řešení okolí stavového prostoru. Častou vlastností těchto metod je takzvaný proces „dívání se před sebe“, kdy se vygeneruje poměrně rozsáhlá kombinace tahů. Z těchto kombinací, vedoucích z počátečního do tzv. pokusného řešení (trial solution), je vybrána taková sekvence tahů, jejichž hodnota vynáší nejlepší výsledky.

V oblasti proměnných hloubkových metod se dle [2] staly prominentní dva typy:

1) Spojité konstruktivní, reprezentovány:
   a) Variabilní prohledávání okolí (VNS) od Hansena a Mladenoviče
   b) Sequential Fan od Glovera a Laguny
   c) Filter &Fan metoda od Glovera
2) Nespojité konstruktivní, reprezentovány:
   a) Lin – Kernighan (LK) metody
   b) Řetězové a iterační LK
   c) Metoda ejekce spojení (Ejection Chain)

V následujících kapitolách budou pro účely paralelizace popsány metody Sequential Fan a Filter & Fan. Tyto postupy jsou běžně používány pro efektivní procházení stavového prostoru, zatímco zde bude tento způsob použit pro zahrnování vrcholů v rámci vytváření

cesty samotné. Při konstrukci tedy dochází k prohledávání a vytváření kombinací okolních vrcholů z vybraného vrcholu.

5. 3 Sekvenční vějíř

Technika volně přeložena do češtiny jako sekvenční vějíř (Sequential Fan), vychází z metody „vyhledávacího paprsku“. Při větvení stromu z daného vrcholu na úroveň nižší paprsek určený parametrem omezí větvení jen do velikosti parametru. Parametr se nazývá šířkou paprsku a v popsaných algoritmech je nazýván proměnnou *vetveni*. Tímto způsobem dochází k menšímu větvení, což má za následek také kratší dobu výpočtu. Dalším důležitým parametrem je hloubka čištění, která stanovuje, na jaké úrovni ve stromu započne tzv. ořezávání větví. Znamená to tedy, že počáteční hodnota větvení inicializovaná na první úrovni bude platná až do úrovně, ve které aplikujeme parametr hloubka čištění. Od této úrovně se již bude parametr větvení snižovat. [13]

Sekvenční vějíř využívá daný způsob větvení při vytváření kombinací, resp. zhotovení stromu, úroveň po úrovni, bez zpětného prohledávání, jen k prozkoumání nejslibnějších uzlů v každém stupni hloubky. Varianta nazvaná „filtrování vyhledávacího paprsku“, angl. *Filtered Beam Search*, na každé úrovni vyhodnocuje celkový výsledek jednotlivých vrcholů k tomu, aby byly vybrány ty nejvhodnější. Vybíráme tedy x nejlepších kombinací (výpočet v rámci stromu) a ty pak podrobíme dalšímu zkoumání ve formě výpočtu Hamiltonovské kružnice, resp. procházíme až na poslední úroveň stromu, tentokrát už bez větvení. Výpočet celkové cesty pro x kombinací pak slouží pro výběr $y \in x$ nejlepších.

5.3.1. Větvení

Metoda Branchingu, česky bychom přeložili jako „Metoda větvení“, spočívá v důkladném prohledání prostoru a nalezení nejlepšího řešení v rámci vybrané podmnožiny. Každý krok větvení rozdělí prostor prohledávání do dvou nebo více podmnožin v souvislosti s vytvořením subproblémů, které by měly být díky menšímu počtu kombinací snazší na vyřešení oproti originálnímu problému. [1] V kombinaci s použitím metody NN budeme hledat uzly, jejichž hodnota je nejnižší, přičemž budeme přijímat i horší řešení.

Tím, jak jsou v průběhu prohledávání metodou NN označovány jednotlivé vrcholy jako použité, dostáváme se do situace, kdy jsou zablokovány uzly, které mají v rámci určitého vrcholu nižší hodnotu a v celkové hodnotě subproblému mohou při zahrnutí vykazovat vyšší

hodnotu výsledku. Pro nalezení lepšího řešení v rámci subproblému, který je zde určen jako kombinace vrcholů o velikosti *hloubka*, nám může sloužit právě metoda větvení

5.3.2. Způsob paralelizace

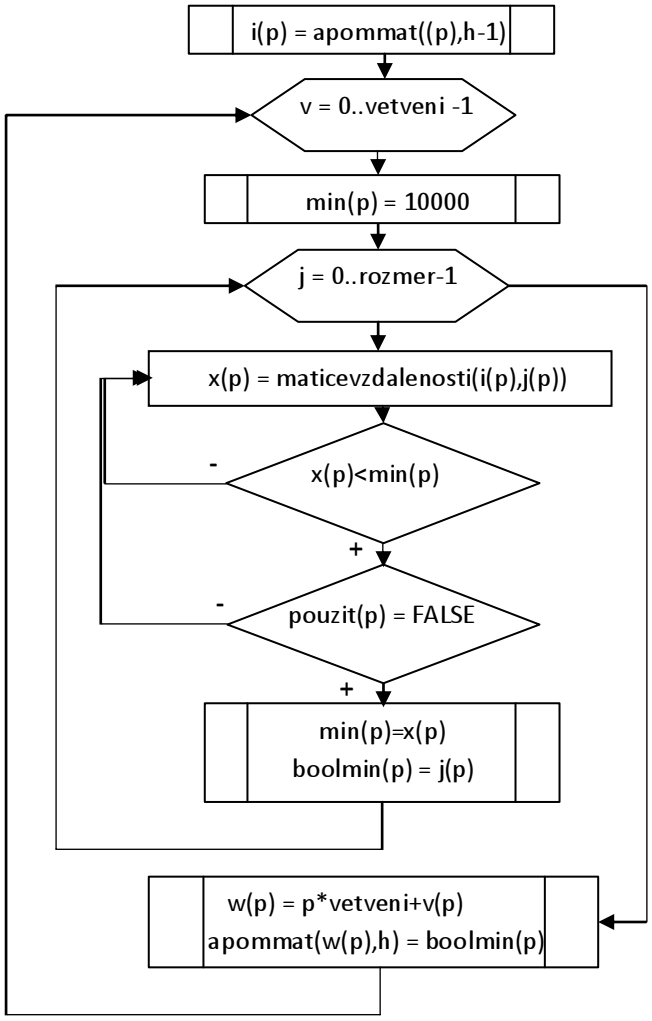
Dle [14] jsou známy dva základní přístupy jak urychlit prohledávání stromu:

- Uzlové, které jsou zaměřeny na to, aby zrychlovaly specifické operace hlavně na úrovni uzlů. Konkrétně se jedná o souběžné výpočty ohodnocení jednotlivých větví, jejich souběžné porovnávání. Obecně jsou nazývány jako nižší úroveň paralelizace, protože žádným způsobem neupravují dráhu prohledávání, ani neovlivňují konstrukci a prohledávání stromu samotného.
- Stromové, zacílené na paralelizované sestavení a výpočet větví stromu.

V této práci bude použit druhý způsob paralelizace, tedy na úrovni stromové, kde využijeme souběžné zpracování pro sestavení a výpočty jednotlivých větví stromu.

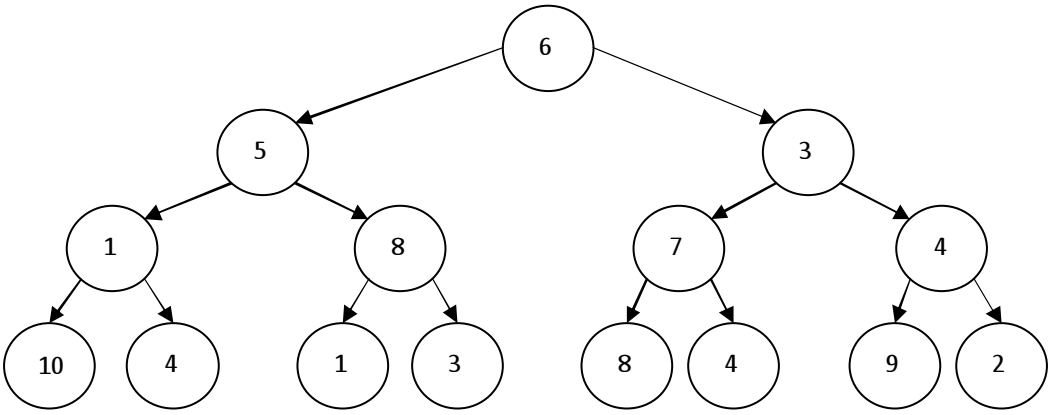
5.3.3. Návrh algoritmu

Tezi z kapitoly předchozí, připouštějící možnost prohledání vrcholů s vyššími hodnotami uzlů než jen minimální, použijeme jako základ pro následující algoritmus. Zavedeme proměnnou větvení, která bude při prohledávání aktuálního vrcholu zahrnovat i vrcholy s vyšší hodnotou uzlu. Při prohledávání vycházíme z původní části algoritmu NN při konstrukci Hamiltonovské kružnice. Tento algoritmus integrujeme do cyklu s pevným počtem opakování určený proměnnou *vetveni*, jak je uvedeno na obrázku 7.



Obrázek 7. Algoritmus větvení

Na začátku algoritmu vybereme příslušný řádek, který reprezentuje poslední prohledané město z předchozí úrovně prohledávání, resp. počáteční město. První for cyklus zajišťuje, že se po prvním průchodu řádku daného města provede vnořený cyklus opětovně a to až do velikosti proměnné vetveni. Zároveň je nastavena minimální hodnota *min(p)* v rámci řádku na velikost vyšší, než je možná hodnota min. Vnořený cyklus s počtem opakování s indexem *i*, pracující do počtu rozmer, je obdobný jako u NN. Na konci tohoto cyklu dojde taktéž k zapsání hodnoty indexu nově nalezeného města, v tomto případě do pole *apommat*. Toto pole je dvourozměrné, navrženo jako pomocné, neboť v rámci každé další úrovně, v souvislosti s hloubkou, je z něj vybírán předchůdce, od kterého se rozvíjí další vrcholy. V tabulce 6. je uveden příklad zápisu hodnot při hodnotách vetveni = 2 a hloubka = 3, který je pro ilustraci znázorněn na obrázku 8.



Obrázek 8. Konstrukce stromu s parametry vetveni = 2, hloubka = 3

Tabulka 6. Hodnoty pomocné tabulky apommat

6							
5	3						
1	8	7	4				
10	4	1	3	8	4	9	2

Způsob zápisu hodnot do pomocné tabulky je uveden na konci vnitřního cyklu v algoritmu obrázku 7. Pomocná hodnota *w(p)*, hodnota daného procesu, je zde vypočítána jako součin čísla procesu a větvení se součtem dané větve procesu. V situaci, kdy proces = 1, vetveni = 2, aktuální větev v = 1 (zde se z důvodů indexace obecného pole od 0 jedná o větev číslo 2) a hloubky 3 (resp. 2 v rámci indexace polí), vypočítaná pomocná w(p)= 1*2+1 = 3, hodnoty indexů pro pomocnou tabulku budou [3,2], v tabulce 6. by byl prvek zapsán na místo čísla 4 ve třetím řádku na konci.

Tento způsob zápisu do pole byl vytvořen z důvodů zjednodušení pro použití paralelizace. Zřejmě by bylo možné vytvořit vícerozměrné pole, jehož počet dimenzí by byl určen hloubkou prohledávání, ovšem z praktického důvodu přepočtu jednotlivých dimenzí a množství proměnných byl použit výše popsaný postup.

Velikost pomocné tabulky je dána parametry hloubka a počtem výsledků, zde uvedených akronymy jako PV(počet výsledků):

```
PV = vetveni sqrt h-1
```

Parametr PV určuje druhý rozměr pole *amaticevysledku*, jehož rozměr je (PV, hloubka), slouží k ukládání prošlých vrcholů a v další fázi na něj bude aplikován paralelní výpočet pro určení hodnot cest jednotlivých výsledků.

Pro zápis údajů do pole uvedeného v předešlém odstavci, amaticevysledku, bude třeba zavést další cyklus s pevným počtem opakování na úrovni vnitřního cyklu. Po prohledání řádku, určení minimální hodnoty na úrovni vrcholu a zapsání vrcholu do pomocné tabulky je nutné zaznamenat vrchol jako použitý pro všechny následující vrcholy, které budou jeho potomky. Zároveň si pro budoucí paralelní výpočty uložíme obdobným způsobem údaj vrcholu do pole *amaticevysledku*. Zápis údajů pomocí for cyklu:

```
aboolmat(p, indexmin(p)) = True
w(p) = (p * vetveni) + v(p)
apommat(w(p), h) = indexmin(p) 'zaznamenani vrcholu do pomocnematice
For wi(p) = ((vetveni * p + v(p)) * PVdel) To
   (((vetveni * p + v(p)) * PVdel) + PVdel - 1)
   maticevysledku(wi(p), h) = indexmin(p)'ulozeni vrcholu do matice
vysledku
Next
```

kde *wi* vychází z čísla procesu a pomocné hodnoty *PVdel*, tj. dělené hodnoty *PV*, která se modifikuje v závislosti na hloubce procházení. Na příkladu, kdy *PV* = 16 (2^5) a aktuální hloubce 3, pak *PVdel* projde dvakrát dělením proměnné vetveni a je 4, výsledná hodnota wi(0) = 0 do 3, wi(1) = 4 do 7. V tabulce 7. je oblast zápisu zvýrazněna zeleně, pro hloubku 4 a procesy 3,4,5 zvýrazněna modře.

Tabulka 7. Zpracování matice výsledků

index/hloubka	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
2	5	5	5	5	5	5	5	5	3	3	3	3	3	3	3	3
3	1	1	1	1	8	8	8	8	7	7	7	7	4	4	4	4
4	10	10	4	4	1	1	3	3	8	8	4	4	9	9	2	2

Cyklus s pevným počtem opakování do úrovně aktuální hloubky zjišťuje, zda se aspirační vrchol (tzn. vrchol, jehož hodnota je menší než hodnota *min* v daném řádku) nenachází v již prošlé cestě:

```
If x(p) < min(p) Then
     pomp(p) = PVdel * p * vetveni
     navstiven(p) = False
     For pomh(p) = 0 To h - 1
        If j(p) = amaticevysledku(pomp(p), pomh(p)) Then
            navstiven(p) = True
        End If
     Next
     If navstiven(p) = False Then And aboolmat(p, j(p)) = False Then
          min(p) = x(p)
          indexmin(p) = j(p)
     End If
End If
```

Pokud není nalezena shoda prvku v daném cyklu a zároveň není prvek zaznamenán jako navštívený v pomocné matici *aboolmat* na úrovni dané hloubky, je zapsán jako prozatímní nejlepší vrchol vycházející z vrcholu předchozího.

Po provedení výpočtů do nastavené úrovně *hloubka* pokračuje algoritmus sekvenčního vějíře dopočítáním kompletní cesty pro všechny propočítané kombinace obdobným způsobem jako při větvení, zde jsou ovšem parametr *vetveni* inicializován na hodnotu 1, tím odpadá nutnost zjišťování použitého vrcholu na úrovni řádku a postačí porovnávání s jeho předchůdci. Tento cyklus je nastaven na pevný počet opakování s počáteční hodnotou proměnné *hloubka* do celkového počtu měst, zde proměnná *rozmer*. Stejně jako při metodě větvení, i zde se výpočet provádí simultánně v závislosti na počtu jader daného počítače.

Nyní je třeba z vytvořených kombinací v *maticevysledku* určit tu s nejmenší hodnotou. Zde opětovně využijeme paralelizace, přičemž na každou kombinaci $k \in PV$ provedeme sumarizaci cest. Cyklem, který porovnává vždy dva vrcholy, získáme z *maticevzdalenosti* pomocí jejich indexů vzdálenost mezi nimi. Po projití celého sloupce je třeba na závěr započítat také cestu mezi výchozím a posledním vrcholem. Tyto výsledky budou zaznamenány do jednorozměrného pole, kde již postačí porovnat entity mezi sebou a vybrat tu nejmenší.

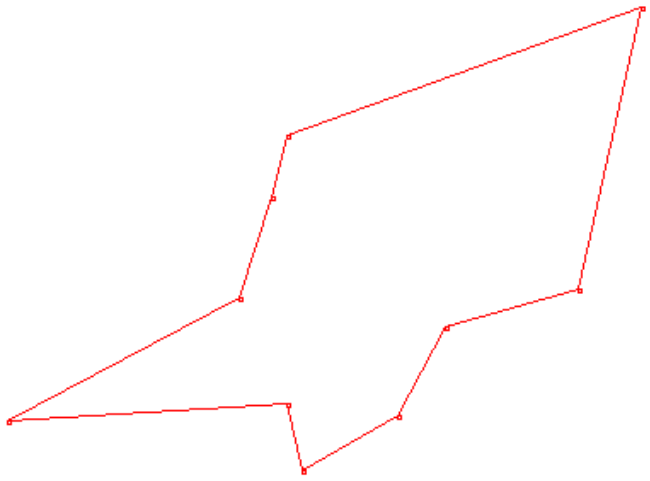
5.3.4. Výsledky algoritmu

V tabulce 8. jsou uvedeny výsledky nejkratších cest pro použitý algoritmus sekvenčního vějíře při různých hodnotách parametrů hloubka a větvení. Na každou dvojici parametrů byl algoritmus spuštěn celkem 10-krát a byla zaznamenána nejnižší nalezená hodnota cesty.

Tabulka 8. Výsledky algoritmu SF

Hloubka	NN	Větvení							
		2		3		4		5	
		výsledek	čas	výsledek	čas	výsledek	čas	výsledek	čas
1	6568	6568	0,00051	6568	0,00056	6568	0,00058	6568	0,00057
2	x	6562	0,00073	6562	0,00087	6371	0,00109	5804	0,00168
3	x	6240	0,00081	6233	0,00116	5283	0,00541	**5250**	0,00597
4	x	6007	0,00082	5445	0,00173	**5250**	0,00496	**5250**	0,01882
5	x	5779	0,00181	**5250**	0,00761	**5250**	0,01843	**5250**	0,04118
6	x	**5250**	0,00693	**5250**	0,01688	**5250**	0,06228	**5250**	0,16313

Pro testování byl použit vzorek 10-ti měst, kde je globálním minimem hodnota 5250, časy uvedené v tabulce 8. jsou pro sestavu číslo 1. Body reprezentující města včetně nejkratší cesty jsou vykresleny na obrázku 9. Metoda NN, spuštěna desetkrát, zaznamenala minimální cestu 6568, údaje v tabulce 8. zvýrazněné tučně jsou hodnoty minima globálního. Pro instanci 10ti měst existuje zhruba 362 tisíc různých kombinací. Z tabulky 8. je patrné, že pro nalezení globálního minima postačilo v kombinaci s heuristikou NN vygenerovat kombinace v počtu 64 až 256.



Obrázek 9. Optimální řešení pro 10 měst

Obdobným způsobem bylo provedeno šetření pro 24 měst. Zde se již nepodařilo dosáhnout globálního minima, při 10ti pokusech bylo dosaženo nejlepší hodnoty 95,7% řešení optimálního, proto bylo nutné daný algoritmus modifikovat.

5. 4 Filter and Fan

Metodu Filter and Fan charakterizuje F. Glower v [15] jako kombinaci sekvenčního vějíře, filtrování a seznamu kandidátů používaném v „zakázaném prohledávání“, angl. Tabu

Search. Sestavu tahů, skládajících se ze sekvence jednotlivých tahů nebo podmnožin tahů, vyhodnocujeme v rámci celé množiny sestav a vybíráme tu nejvhodnější, tj. tu s nejnižší hodnotou délky cest. Z kapitoly 5.3. popisující vytváření kombinací v rámci algoritmu sekvenční vějíř je zřejmé, že při vyšším počtu cest není možné dosáhnout globálního minima, neboť nastavení parametrů větvení a hloubka na vyšší hodnoty vedou k vysokým časům výpočtu. Při zvyšování těchto parametrů nad určitou mez značně roste doba výpočtu, z tohoto důvodu je třeba omezovat množství vytvářených kombinací. Metoda Filter and Fan postupně vybírá nová řešení pro danou úroveň pocházejících ze všech řešení vygenerovaných na dané úrovni. Vytváříme tedy podmnožinu tahů, které mají nejvyšší ohodnocení, v TSP nejnižší hodnotu cesty, z množiny tahů na dané úrovni.

Zatímco v [1] je metoda použita na celé sestavy cest a prohledává stavový prostor, zde algoritmus využívá tento proces na konstrukci cesty výběrem určitých kombinací vrcholů. Počítáme tedy cestu jen pro aktuálně projitou hloubku, resp. cestu ze zahrnutých vrcholů. Na realizaci F&F využijeme část algoritmu pro vytváření kombinací z předchozí kapitoly.

5.4.1. Výběr kandidátů

Na začátku algoritmu je třeba na základě zvoleného parametru *hloubka* určit počet opakování tak, aby došlo k projití všech vrcholů. Vzhledem k tomu, že na začátku vybíráme jedno výchozí město, celkový počet měst, zde uvedený jako *rozmer*, ponížíme o jednotku. Následně počítáme, zda existuje zbytek po dělení mezi počtem měst a hloubkou.

```
If ((rozmer - 1) Mod (hloubka)) = 0 Then
    pocethloubek = (rozmer - 1) / (hloubka)
Else
    pocethloubek = ((rozmer - 1) \ (hloubka)) + 1
End If
```

Pokud je zbytek po dělení větší než nula, je přidán ještě jeden průchod. Na příkladu 29ti měst – jestliže určíme parametr *hloubka* = 7, pak 28 mod 7 = 0 => počet hloubek je roven 4. Když nastavíme hloubku na 6, 28 mod 6 <> 0 => počet hloubek je roven 5. V tomto případě je procházení bodu v dalším cyklu ošetřeno podmínkou

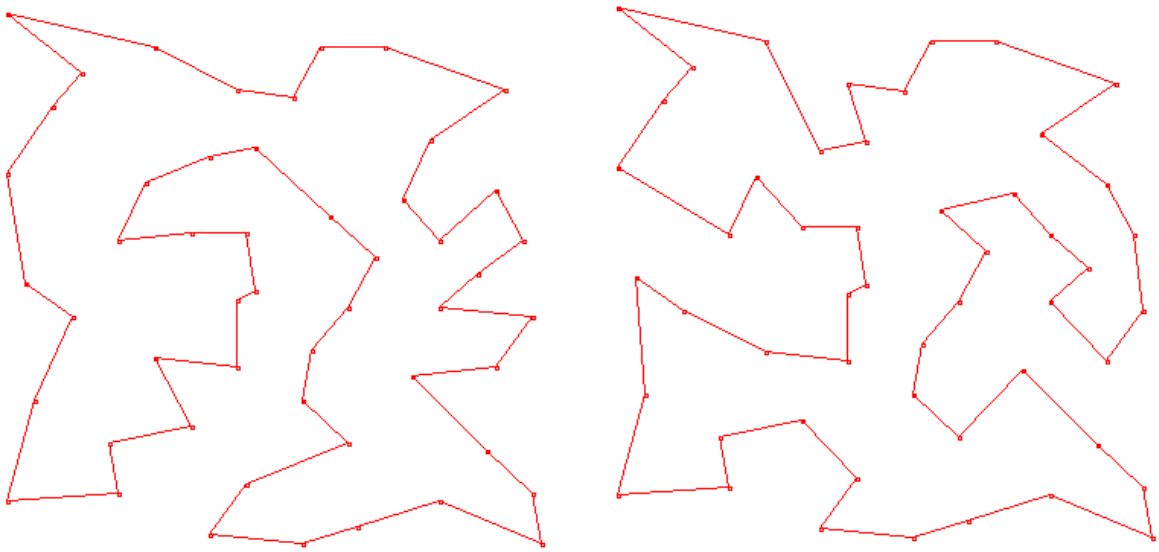
```
If ph = pocethloubek Then hloubkacelk = rozmer
```

kde *ph* představuje právě explorovanou hloubku, ohraničenou parametry 1 do hloubkacelk.

Po vygenerování kombinací pořadí měst pro lokální okolí je nutné vyhodnotit a vybrat nejlepší výsledky. Byly zavedeny dva způsoby vyhodnocování vytvořených tahů. V první variantě se jednalo o spočítání cesty pro aktuální hloubku. Algoritmus však nedosahoval obstojných výsledků, proto bylo implementováno dopočítávání cesty metodou NN bez větvení. Z takto dopočítané cesty byl na základě proměnné *kandidati* vytvářen seznam kandidátů, tj. z pole *avyslednecesty* bylo vybráno *k* výsledků s nejnižší hodnotou cesty. Tyto vybrané kombinace měst jsou následně uloženy do pole *amaticevysledku* a pomocného pole *apommat*.

```
For hl As Integer = 0 To hloubkacelk - 1
    For mb As Integer = 0 To pocetpredchozich - 1
       amaticevysledku(mc * pocetpredchozich + mb, hl) =
       amaticevysledkupom(amaticeindexu(nejindexx), hl)
    Next
Next
apommat(0, hloubkacelk - 1) = amaticevysledku(0, hloubkacelk - 1)
```

Algoritmus byl testován na příkladu s 10ti, 29ti a 51ti městy. Vzhledem k tomu, že vycházíme z metody „sekvenčního vějíře“, nalezení globálního optima na studii 10ti měst nebylo problémem. Pro nalezení nejlepšího řešení na případu 29ti měst bylo již třeba proměnlivě nastavovat parametry *vetveni* a *hloubka*. Po testu, kdy byl algoritmus spuštěn, v souvislosti s měnícími se parametry, celkem 10-krát, při vetveni = 6, hloubka =5 bylo dosáhnuto optimálního řešení pro 29 měst, zatímco pro přiblížení ke globálnímu optimu pro 51 měst, znázorněného na obrázku 10., byly hodnoty nastaveny na 5; 4.



Obrázek 10. Globální minimum vs. získané lokální minimum pro 51 uzlů

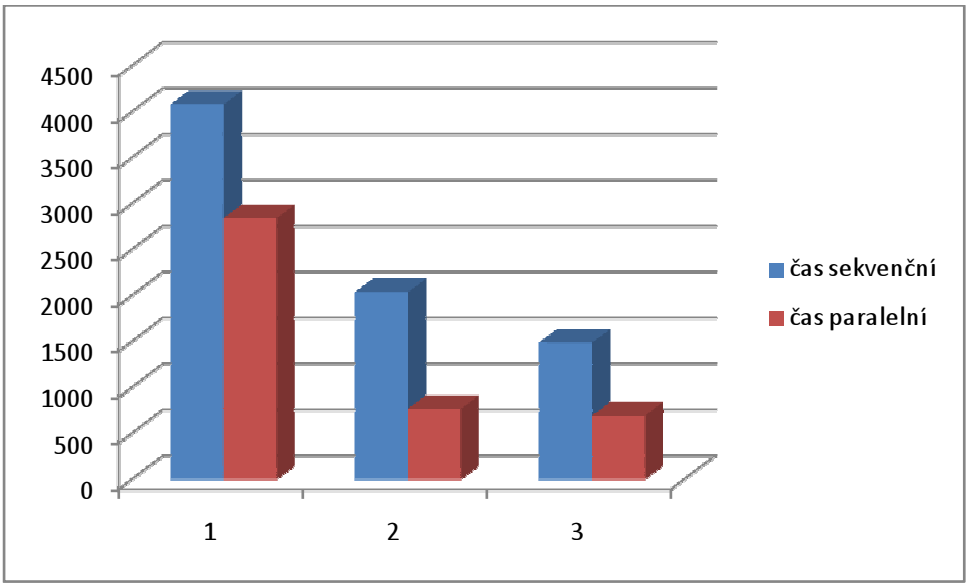
5.4.2. Výsledky algoritmu

Celkové výsledky jsou uvedeny v tabulce 9. V průběhu vytváření výsledků a testování došlo ke změně v podobě proměnlivé hloubky, a to vzhledem k počtu osekání stromu, resp. zachování počtu větví. V situaci, kdy je větvení nastaveno na 5 a po prvním průchodu celkem 7mi hloubkami existuje celkem 5na7 = 78125 kombinací, přičemž při zachování X*vetveni rodičů – parametr *pocetpredchozich*, je nutné snížit hloubku na *hloubka – X*. Při výběru 10ti nejlepších je tedy hloubka ponížena na *hloubka* – 2, jelikož 2 = 10/5 (10/*vetveni)*.

Tabulka 9. Výsledky F&F s parametry v = 3, h= 7

	Kvalita řešení		Čas v sekundách 2.sestava	
Instance	avg	best	sekvenčně	paralelně
Bays29	97,77%	100,00%	0,32	0,12
Eil51	93,24%	95,73%	2,42	0,76
Berlin52	94,57%	98,51%	2,28	0,74
KroA100	90,07%	92,17%	19,66	6,32
Ch150	94,75%	98,54%	83,82	21,33
A280	92,34%	93,99%	2254,22	428,64
Lin318	90,06%	91,03%	1942,50	681,53

Z tabulky 9. je zřejmé, že vytvořený algoritmus dokáže určit v polynomiálním čase pro středně velké instance řešení s průměrnou kvalitou vyšší než 90% a nejlepší dosaženou hodnotou při 10-ti opakováních s kvalitou vyšší než 97%. Časy uvedené v tabulce jsou hodnoty naměřené při sekvenčním a paralelním zpracování na sestavě číslo 2 v jednotkách sekund. V případě souboru Eil51 bylo provedeno se stejnými parametry šetření, kdy za výchozí město byla postupně vybrána všechna města, přičemž bylo dosaženo globálního minima, konkrétně při výchozím městě 23 a 30. Jelikož se jedná o modifikaci metody NN, je výsledná kvalita řešení závislá na volbě počátečního města. Rozdíly výpočetního času sekvenčního a paralelního algoritmu pro testovací počítačové sestavy na instanci 51 měst s parametry v=3, h=7, jsou uvedeny v grafu 1. Osa x reprezentuje čísla testovacích strojů, zatímco osa y stanovuje uplynulý čas potřebný ke zpracování algoritmu v milisekundách.



Graf 1. Výpočetní čas sekvenčního a paralelního algoritmu F&F

5. 5 K-Means

Vzhledem k příliš vysokým časům výpočtu metodou F&F na počtu měst vyšších než 150 došlo k zavedení procedury na dekompozici množiny měst. Pro účely rozdělení byla použita nehierarchická metoda s pevným počtem shluků nazývaná jako K-Means. Podstatou tohoto algoritmu tedy je, že rozdělí množinu dat, v našem případě množinu měst, do *k* shluků. Základní pseudoalgoritmus je určen [16] následovně:

1) Vygeneruj počáteční hodnoty clusteru, urči centroidy každého shluku
2) Přiřaď každý vrchol do nejbližšího clusteru
3) Přepočítej centroidy shluku
4) Jestliže je dosaženo konvergence, ukonči algoritmus, jinak jdi na bod 2.

Pro základní výpočet rozdělní na shluky byla zavedena tato dvourozměrná pole znázorněná v tabulkách 10. a 11.

adata(2, rozmer – 1) sloužící k ukládání souřadnic vrcholů *x*,*y* a informace o tom, do jakého shluku byl vrchol přiřazen

acentroid(1,pocetshluku-1), ve kterém jsou uloženy souřadnice centroidů

Tabulka 10. pole adata pro 10 měst a 2 shluky

Město č.	1	2	3	4	5	6	7	8	9	10
Číslo shluku	1	1	1	0	1	1	0	1	1	0
X	1150	630	40	750	750	1030	1650	1490	790	710
Y	1760	1660	2090	1100	2030	2070	650	1630	2260	1310

Tabulka 11. pole acentroid pro 2 shluky

Centroid č.	1	2
X	1037	840
Y	1020	1929

Po počáteční fázi zápisu souřadnic jednotlivých měst provedeme náhodný zápis souřadnic do pole centroidů. Jedním ze způsobů je například náhodné určení souřadnice X a Y pomocí funkce VB Random, nebo druhý způsob – výběr souřadnic z náhodného vrcholu. Po zapsání údajů do tabulky spustíme proceduru, kde přiřadíme ke každému vrcholu daný shluk a to na základě vzorce [17] klasifikace podle nejbližšího souseda:

$$i = \underset{k=1,\dots,K}{\operatorname{argmin}} \left| x_j - m_k \right|^2$$

tak, abychom dosáhli co nejmenších rozdílů uvnitř shluků [17]:

$$SSE(X, G) = \sum_{i=1}^{K} \sum_{x_j \in C_i} || x_j - m_i ||^2$$

kde x_j je vrchol patřící k danému shluku i, a mi je centroid daného clusteru, vypočítáváme součty čtverců odchylek (Sum of squared error). Nevýhoda metody K-Means je, že ačkoliv konverguje v konečném počtu iterací, může snadno uvíznout v lokálním minimu. Do značné míry mají vliv na konečné řešení počáteční podmínky, zde konkrétně výběr počátečních centrů shluků. [17]

Po přiřazení prvků do shluků provedeme přepočet těžiště, a to Forgyovou metodou, kde spočítáme pomocí nově určených bodů shluku centroidy těžišť. Dále se pak proces přiřazování vrcholů do shluků a přepočítávání těžišť opakuje, dokud nedojde ke stavu, kdy už se prvky ze shluků nepřesouvají, tj. do situace, ve které při posledním přepočítávání již nebyl žádný z prvků přesunut do shluku jiného.

Dalšími deklarovanými poli, určených pro samotný výpočet shluků, jsou:

ashlukvstup – jedná se o třírozměrné pole definované parametry počet shluků, počet bodů ve shluku a hodnotou tři. Do těchto tří hodnot jsou zaznamenány hodnoty X, Y vrcholu

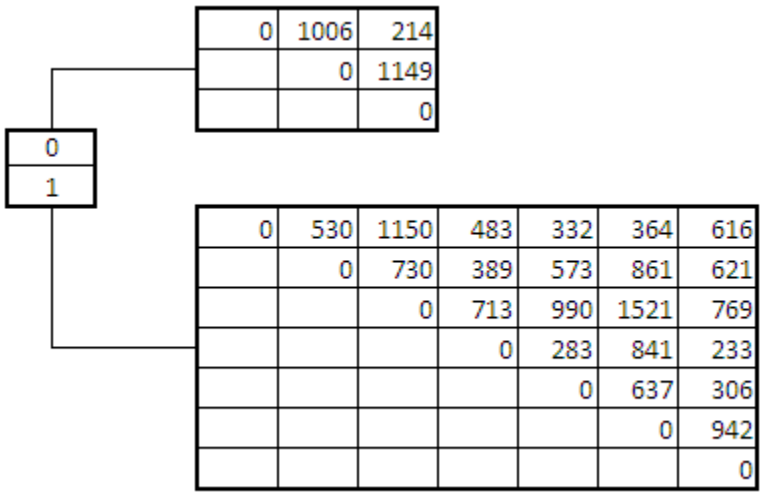
a původní číslo vrcholu, neboť pro účely použití již vytvořených algoritmu budeme pracovat se shlukem stejně jako s celou množinou.

ashlukmaticevzdalenosti – toto pole je identické s původním polem *amaticevzdalenosti*, ovšem přibyl zde třetí rozměr pro zaznamenání čísla shluku. Na obrázku 11 je toto pole uvedeno pro 10 měst a dva shluky s hodnotami použitými z tabulky 10. Před samotným ukládáním hodnot si pole nadeklarujeme:

```
Dim ashlukmaticevzdalenosti(pocetshluku - 1)(,) As Integer
```

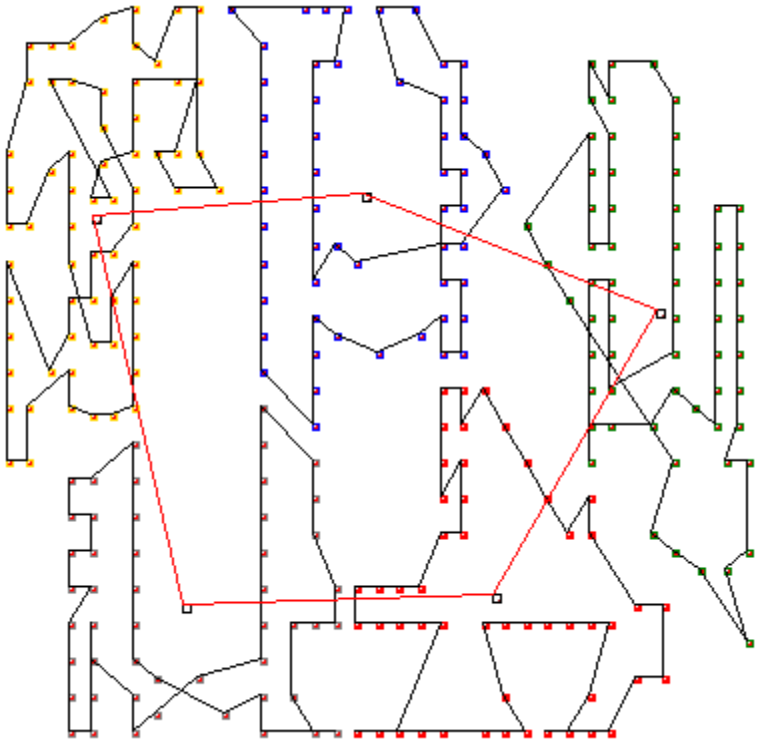
a pro každý shluk, který je ošetřen smyčkou s pevným počtem opakování do počtu shluků a proměnnou *s*, realokujeme zbylé dva rozměry pro každé *s* jako:

```
ReDim ashlukmaticevzdalenosti(s)((pocetboduveshluku(s) - 1),
(pocetboduveshluku(s) - 1))
```



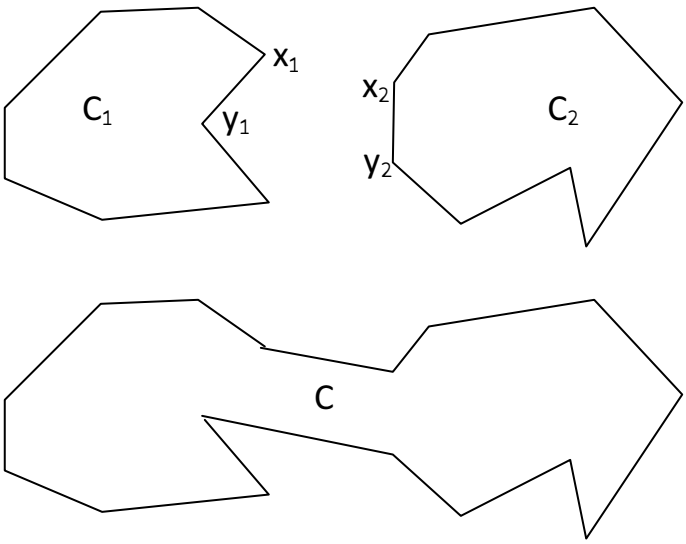
Obrázek 11. pole ashlukmaticevzdalenosti

Na nově vytvořené pole vzdáleností můžeme aplikovat původní procedury branchování F&F pro jednotlivé shluky. Tímto způsobem vznikne *k* Hamiltonovských kružnic vykreslených na obrázku 12, kde *k* reprezentuje počet shluků. Body daných shluků jsou rozlišeny barevně, reprezentovaná cesta shluku černě. Červená barva určuje nejkratší vzdálenost mezi centroidy. Těchto výsledků bylo dosaženo algoritmem F&F při parametrech hloubka = 5 a větvení = 3.



Obrázek 12. Subcesty 280ti měst pro 5 shluků

Červená cesta na obrázku 12, spojující centroidy shluků, bude sloužit jako základ pro další výpočty. Po vytvoření subcest bude třeba zvolit vhodný způsob spojení. Obdobně jako při výpočtu matice vzdáleností pro cesty budou určeny vzdálenosti mezi získanými středy shluků. I zde bude použit F&F algoritmus k získání optimální cesty a sled centroidů bude zapsán do pole *amaticeshluku*. Poté bude pro každou dvojici shluků, vybranou z *amaticeshluku*, určena minimální vzdálenost mezi dvěma body. K těmto dvěma bodům budou vybráni jejich sousedé, zleva a zprava, v rámci vytvořené Hamiltonovské kružnice bude stanovena nejvhodnější kombinace tak, aby hodnota cesty byla minimální. Jestliže budeme vycházet ze dvou Hamiltonovských cyklů C_1, C_2, vyhledáme dvě hrany $e_i = (x_i, y_i)$ pro i = 1,2, o které snížíme hodnotu cesty a vložíme hrany nové $f_1 = (x_1, x_2)$ *a* $f_2 = (y_1, y_2)$ jako přírůstek cesty. Vznikne tím nová kružnice, definovaná jako $C = C_1 + C_2 + f_1 + f_2 – e_1 – e_2$, popsaná na obrázku 13. [2]



Obrázek 13. Spojení subcest, převzato z [2]

Výsledky testovaného algoritmu F&F v kombinaci s nehierachickým shlukováním jsou uvedeny v tabulce 12., kde je porovnávána průměrná kvalita řešení a průměrný čas v sekundách na zpracování paralelním způsobem.

Tabulka 12. Výsledky k-means

Instance	Počet shluků	Kvalita řešení		% zhoršení kvality	Čas paralelně 2.sestava		% zrychlení výpočtů
		avg	best		původní	k-means	
KroA100	2	90,07%	90,51%	-0,44%	6,32	1,31	483%
Ch150	2	94,75%	91,10%	3,65%	21,33	4,41	484%
A280	3	92,14%	88,22%	3,91%	428,64	26,44	1621%
Lin318	5	89,25%	87,73%	1,51%	681,53	9,53	7150%

5. 6 Dílčí závěr

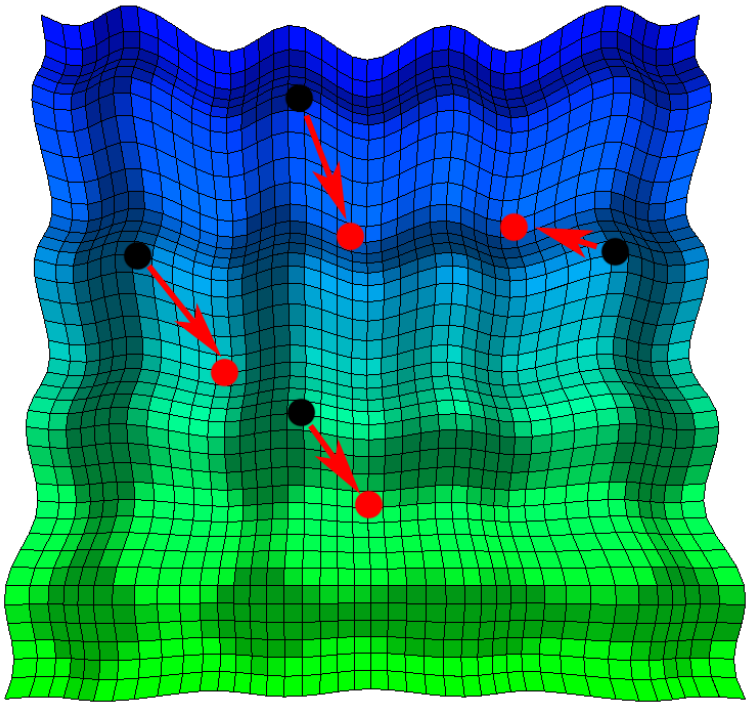
Ze šetření uvedeného v tabulce 8. o rychlosti algoritmu NN vyplývá, že sekvenční zpracování je při menším počtu opakování rychlejší než jeho paralelní ekvivalent. Důvodem je dle [4] tzv. cena za vlákno, jelikož vytváření a rušení vláken je relativně drahá akce. A pokud je na úrovni smyčky prováděn menší počet iterací a uvnitř smyčky se zpracovávají drobné výpočty, nastává situace, kdy je skutečně sekvenční zpracování rychlejší.

Kombinace metody NN s větvením a dopočítáváním cesty vykazuje mnohem lepší výsledky nežli bez dopočítávání. Při menším počtu měst, tzn. kolem 50ti, se algoritmus přibližuje a občas dosahuje minima globálního, při větším je pak průměrná kvalita nalezeného řešení 92% a nejlepší v průměru 94%. Při větší instanci je tedy nalezené řešení vhodné jako výchozí bod pro další metody, které okružní cestu zlepšují.

Efektivita paralelního algoritmu v metodě F&F je závislá na počtu kombinací vytvořených v rámci konstrukce stromu. Hodnota kombinace vychází z nastavených parametrů *hloubka* a *vetveni*. Správná volba těchto proměnných má také vliv na kvalitu nalezeného řešení. Dekompozice technikou K-means zajistila rychlejší zpracování metodou F&F o stovky % času, ovšem za cenu kvality nižší až o 4%.

6 Iterativní heuristiky zlepšování

Heuristiky zlepšování jsou používány na již vytvořené řešení, přičemž se pokouší náhodným prohozením vrcholů nebo uzlů docílit lepšího výsledku. Algoritmus začíná modifikací cesty získané náhodným způsobem nebo například konstrukčními heuristikami, a provádí proces zlepšování do splnění ukončovacího kritéria. Tím může být stanovený počet kroků, uplynutí doby přidělené na provedení algoritmu nebo dosažení stanoveného výsledku. Výsledek zlepšovacích heuristik je závislý na počáteční cestě, proto se zde nabízí možnost provést heuristiku na větším počtu výchozích cest, jak je znázorněno na obrázku 14.



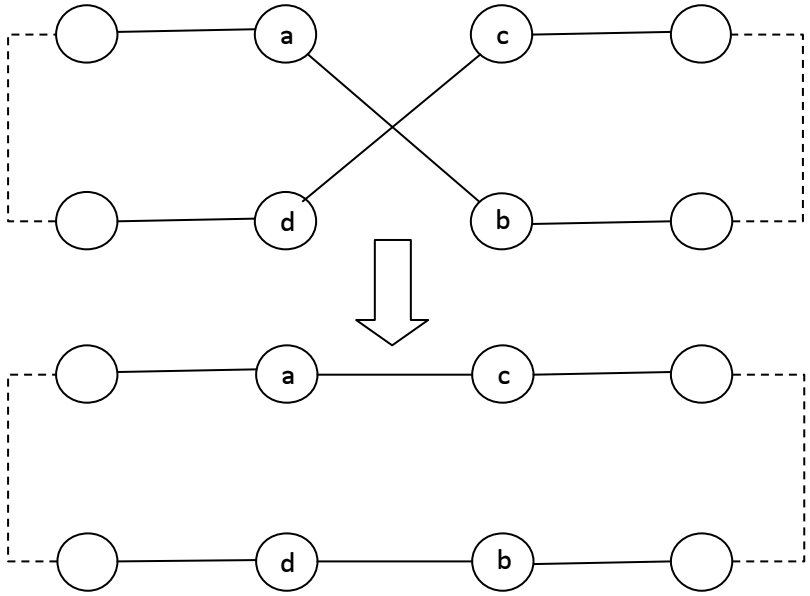
Obrázek 14. Způsob paralelizace zlepšovacích heuristik

Pokud bychom si pro ilustraci představili stavový prostor jako dvourozměrné pole, černé tečky reprezentují počáteční cestu, červené pak lokální minimum získané lokálním prohledáváním.

Mezi nejběžnější metody těchto heuristik patří k-opt, kde *k* určuje sekvenci tahů výměny uzlů. Na obdobném principu funguje i tzv. řetězová výměna, ve které dochází k výměně vrcholů. Tento způsob, stejně jako k-opt, bude použita v kapitole Genetických algoritmů jako součást procesu mutace. V následující kapitole je popsána základní metoda 2-opt společně s vytvořením algoritmu a výsledky šetření.

6. 1 2-opt

Metoda 2-opt je založena na principu výběru dvou hran (u TSP tedy dvou cest) a jejich smazání. Pro ilustraci na obrázku 15. se jedná o hrany ab a cd. Poté dojde k přidání dvou hran – ac a bd, což v praxi pro naše účely v TSP bude znamenat záměnu pořadí prvku. Nová cesta je nyní kratší, protože ab + cd > ac + bd. [18]



Obrázek 15. Metoda 2-opt, převzato z [18]

V algoritmu porovnáváme dvě cesty označené *nova*, *stara* získané z matice vzdálenosti *amvzd* jako

```
nova =(amvzd(alk(i, nah1 - 1), alk(i, nah2 - 1)) + amvzd(alk(i,
nah1), alk(i, nah2)))
stara =(amvzd(alk(i, nah1 - 1), alk(i, nah1)) + amvzd(alk(i, nah2 -
1), alk(i, nah2)))
```

kde *alk* je matice určité cesty *i* pro algoritmus 2-opt a parametry *nah1*, *nah2* udávají hodnotu, kde se objekt překříží. Pokud je nová cesta kratší než původní, dojde k prohození měst a záměně pořadí, jak je uvedeno v následující části kódu

```
If nova < stara Then
    Dim rozmerpommat As Integer = nahoda2 - nahoda
    Dim b As Integer = rozmerpommat
    Dim apommat(rozmerpommat) As Integer
    For a As Integer = nah1 To nah2 - 1
        apommat(b) = alk(i, a)
        b -= 1
    Next
    Dim d As Integer = 1
    For c As Integer = nah1 To nah2 - 1 'forcyklus kop. zpet
        alk(i, c) = apommat(d)
        d += 1
    Next
End If
```

První smyčka s pevným počtem opakování, určena parametry nah1, nah2, ohraničuje interval, ve kterém dojde k inverzi prvků. Z tohoto důvodu je zavedeno pole apommat, do kterého jsou prvky z intervalu inverzně převedeny, a následně je tímto sledem nahrazen interval původní.

Proces zahájíme vygenerováním počáteční cesty, která může vzniknout několika způsoby a to:

- náhodným výběrem ze všech měst, která nejsou prozatím součástí cesty
- náhodným výběrem z bezprostředních sousedů
- použitím vytvořené cesty z výsledků předchozích kapitol, tedy algoritmu F&F, SF nebo NN

Při náhodném výběru ze všech měst zabere doba výpočtu více času, neboť na počátku je cesta, v porovnání s dalšími dvěma způsoby, podstatně delší. Na druhou stranu tímto prvotním způsobem zjistíme, zda je algoritmus efektivní a dochází ke zlepšení. Šetření 2-opt s počátečním náhodným výběrem měst bylo provedeno na stejných instancích jako konstruktivní heuristiky.

Vzhledem k tomu, že se jedná o iterativní proceduru, zavedeme parametr pocetopakovani, který nám udává, kolikrát bude 2-opt proces opakován. Pro zavedení paralelizace byl přidán parametr pocetindividui, čímž byl algoritmus rozšířen o další cyklus s pevným počtem opakování, který lze vytvořit smyčkou parallel.for. Tato paralelizace zajistí, že počítač bude simultánně upravovat n různých cest s odlišnými počátečními stavy a tím zajistí teoreticky vyšší šanci na nalezení cesty optimální. Výsledky šetření pro 2-opt

algoritmus s náhodně vytvořenými počátečními cestami na sestavě 2 jsou uvedeny v tabulce 13.

Tabulka 13. Šetření LK 2-opt RND

Instance	Kvalita řešení		Čas v sekundách 2.sestava		Opakování 2-opt
	avg	best	sekvenčně	paralelně	
Bays29	99,68%	100,00%	0,01	0,02	3000
Eil51	96,69%	98,16%	0,03	0,02	5000
Berlin52	96,21%	99,93%	0,04	0,02	6000
KroA100	94,20%	97,01%	0,11	0,07	18000
Ch150	92,26%	94,94%	0,31	0,18	50000
A280	89,97%	92,07%	1,22	0,63	200000
Lin318	91,65%	92,99%	1,76	0,98	300000

Sloupec Opakování 2-opt udává hodnotu, kolikrát bylo pro danou instanci provedeno pokusů o záměnu uzlů. Celý proces byl z důvodů zavedení paralelizace spuštěn na 40ti nezávislých počátečních cestách.

Pro další šetření bylo zavedeno pomocné pole *asousede*, jehož rozměr je dán počtem měst a parametrem *pocetsousedu.* Tento parametr nám udává počet sousedních měst, které budou uvedeny pro každé město. Z tohoto pole budou vybírání následníci pro aktuální město v rámci vytváření počáteční cesty stejně jako náhodný výběr ve fázi zlepšování cesty 2-opt.

Na počátku je vytvořeno pole a*sousede*, do kterého je uloženo n počet měst, kde *n* = *pocetsousedu*. Poté proběhne vygenerování nahodilých cest. Na začátku je vybráno náhodně město z celého rozsahu a je zapsáno jako prvek cesty. Na základě indexu tohoto města je vybrán náhodný soused z pole *asousede*, zapsán do seznamu a nahrazuje index. Tento proces se opakuje, dokud není cesta naplněna prvky. Aby nedocházelo ke zdvojování použitých měst, je algoritmus ošetřen na základě smyčky s podmínkou na začátku a pomocného pole *asousedebool* deklarovaného jako datový typ boolean, ve kterém jsou projitá města zaznamenána hodnotou TRUE.

Zatímco náhodný výběr hodnot u předchozího typu algoritmu vybíral oba prvky pomocí funkce Random, zde je náhodně vybrán pouze prvek číslo jedna, jak je uvedeno v prvním řádku algoritmu. Druhý prvek je vybrán na základě náhodného výběru z pole *asousede*.

```
Dim nahoda As Integer = rnd.Next(1, rozmer - 2)
Dim pomnahoda2 As Integer = rnd.Next(0, pocetsousedu)
Dim pomnahoda2a As Integer = asousede(nahoda, pomnahoda2)
Dim nahoda2 As Integer
For l As Integer = 0 To rozmer - 1
    If alk(i, l) = pomnahoda2a Then
         nahoda2 = l
    End If
Next
```

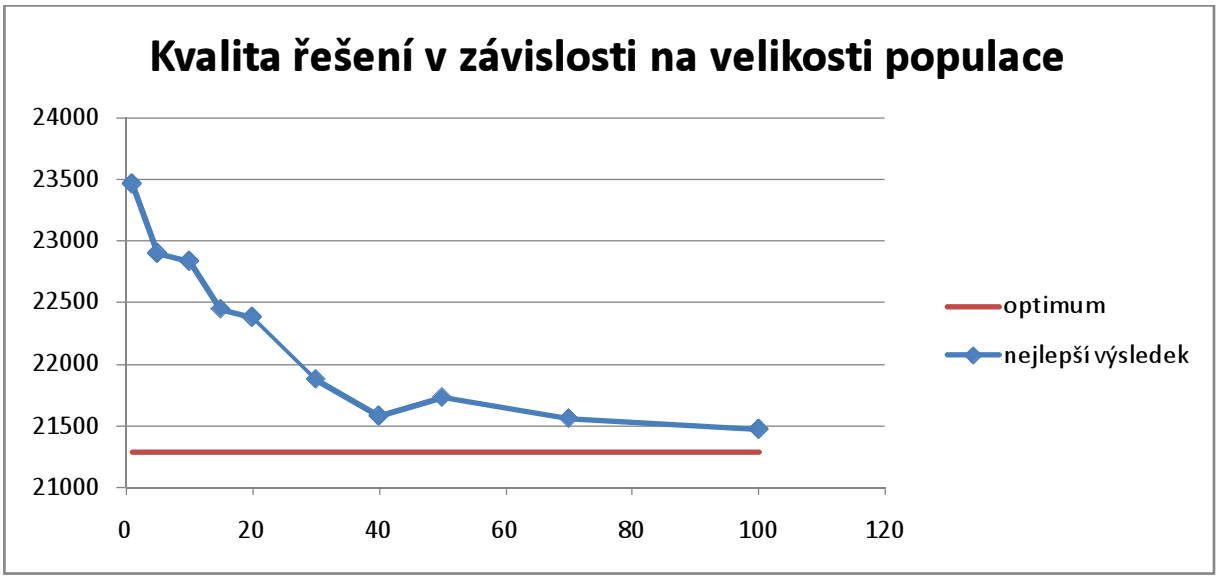
Zde je algoritmus v porovnání s náhodným výběrem obou prvků znatelně pomalejší, neboť je třeba projít celou cestu reprezentovanou polem *alk*. Výsledky získané tímto procesem vykazují s rostoucím počtem měst snižující se kvalitu řešení v porovnání s předchozím způsobem výměny tahů. Dalším rozdílem je poměr rychlosti zpracování paralelně a sekvenčně, která je díky většímu objemu výpočtů uvnitř smyčky vyšší. Tento způsob výměny tahů není nejvhodnější, ale potvrzuje vyšší efektivitu při větším objemu výpočtu uvnitř smyčky.

Tabulka 14. uvádí nejlepší kvalitu řešení pro různé způsoby počátečního vygenerování cest. Na jednotlivé typy byla použita metoda 2-opt s celkovým počtem 10ti opakování.

Tabulka 14. Výsledky 2-opt s počátečními RND-sousedé-NN-F&F

Instance	Kvalita řešení			
	2-opt rnd	2-opt sousedé	NN	F&F
Bays29	100,00%	100,00%	99,77%	100,00%
Eil51	99,53%	98,16%	97,48%	99,07%
Berlin52	100,00%	94,56%	99,29%	100,00%
KroA100	98,12%	99,04%	96,82%	98,72%
Ch150	95,05%	93,67%	98,48%	98,95%
A280	92,17%	88,59%	94,09%	96,56%
Lin318	93,44%	91,56%	93,30%	94,39%
Průměr	96,90%	95,08%	97,03%	98,24%

Technika 2-opt pro počáteční náhodné generování a vytvoření z okolních sousedů vycházela z výběru nejlepšího řešení populace o velikosti 40ti entit. Na této hodnotě závisí do jisté míry kvalita, resp. se zvyšující se velikostí populace roste pravděpodobnost nalezení lepšího řešení, jak znázorňuje graf 2 pro instanci KroA100. Osa x v grafu 2. reprezentuje velikost populace, zatímco osa y kvalitu nalezeného řešení.



Graf 2. kvalita řešení v závislosti na počtu jedinců

6. 2 Dílčí závěr

Paralelní zpracování techniky 2-opt vykazuje ve výsledcích testování na sestavě č.1. vyšší čas pro zpracování oproti jejímu sekvenčnímu ekvivalentu. Důvodem je podobně jako u metody NN příliš malý počet operací uvnitř paralelní smyčky. Ke zvýšení efektivity by jistě přispěla modifikace algoritmu na 3-opt nebo vyšší. Důkazem jsou výsledky upravené metody, ve které jsou uvnitř paralelní smyčky navíc zakomponovány další výpočty v souvislosti s procházením nejbližších sousedů vybraného uzlu.

Rychlost konvergence s různými počátečními cestami byla podle očekávání značně rozdílná. Zatímco u náhodného vygenerování byla rychlost konvergence a čas vysoké, v případě použití výsledků z metody F&F nebylo téměř co upravovat. Rozdílná byla kvalita nalezeného řešení při použití daných typů počátečních cest. Nejlepšího průměru dosáhl algoritmus při využití cest z konstruktivních heuristik, a to v průměru přes 98% kvality řešení. U náhodně vytvořené skupiny bylo po dokončení zlepšování na těchto instancích dosaženo nižších průměrných hodnot.

Heuristika 2-opt je velice rychlá a efektivní metoda a v kapitole GA bude použita jako jeden z operátorů mutace.

7 Metaheuristiky

Nevýhodou heuristických metod je, že se snadno zachytí v lokálních minimech stavového prostoru. Metaheuristiky jsou metody, které řídí a upravují klasické heuristiky k tomu, aby produkovaly i taková řešení, která jsou mimo lokální prohledávání. [3] Snaží se tedy o diverzifikaci procházení stavového prostoru (připouštějí i horší řešení) a zintenzivnění prohledávání ve slibných oblastech (použití heuristiky).

Mezi nejznámější metaheuristiky dle [3] patří:

Tabu Search (TS) – metoda zakázaného prohledávání, která používá například heuristickou techniku k-opt a v základní verzi zavádí tzv. zakázaný seznam. V tomto seznamu jsou uloženy tahy (získané při modifikaci cesty), které nemohou být v rámci dalších modifikací použity, čímž je zaručen únik z lokálního minima.

Simulované žíhání (SA) – stejně jako u metody TS, i zde lze využít iterativní heuristiku k-opt k procházení stavového prostoru. SA připouští horší řešení s určitou pravděpodobností, jež je závislá na snížené vhodnosti (o kolik je nové řešení horší než aktuální) a na parametru teploty. Jestliže je aktuální řešení blízko lokálního optima, pak následná sekvence tahů připuštěná jako horší zajistí posun do jiné oblasti. Pokud je teplota příliš nízká nebo je snižována příliš rychle, únik z lokálního minima je obtížnější. [19]

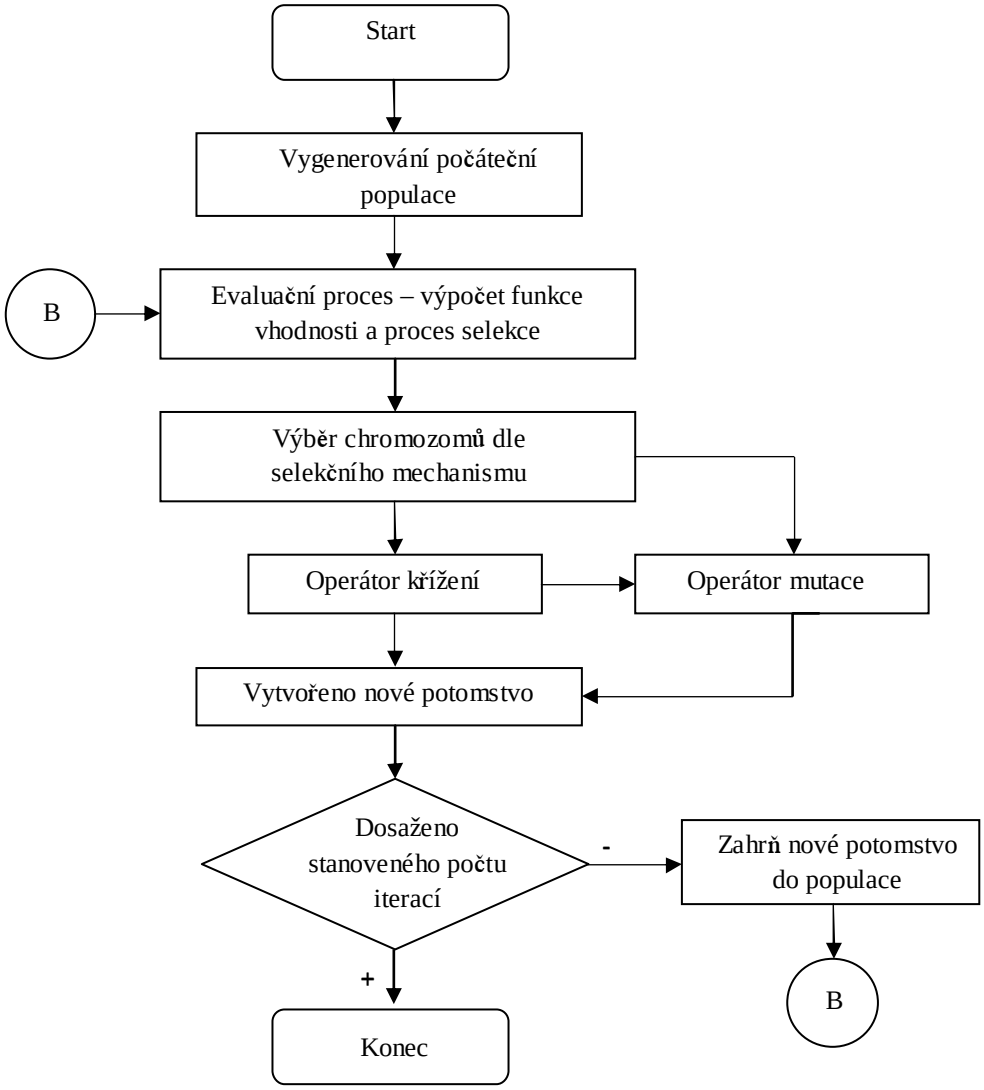
Genetické algoritmy (GA) – založeny na populační simulaci, podrobněji popsány v následující kapitole.

Mravenčí kolonie (ACO) – stejně jako genetické algoritmy je založena na populační simulaci, zde na skupině mravenců. Každý mravenčí jedinec prochází soubor měst, zpočátku náhodně, a přitom zanechává pachovou stopu, tzv. feromon. Při dalších průchodech se rozhoduje pro další uzel na základě pravděpodobnosti náhodně nebo na množství virtuálních feromonových stop, kterou zanechali ostatní mravenci z populace.

7. 1 Genetické algoritmy

Genetické algoritmy patří společně s technikou simulovaného žíhání do skupiny tzv. Řízených náhodných technik prohledávání. Jedná se o evoluční procesy, které vychází z principů přirozeného výběru, inspirované biologickými procesy zejména ve formě vývoje generací nebo zlepšování rysů daných jedinců [20]

Základní myšlenka těchto algoritmů je vytvářet nové sady řešení s použitím určitých operací na počáteční, resp. starší sadě. Z nově vytvořených řešení jsou vybrána ta nejlepší, která jsou součástí nové sady, přičemž tento proces je opakován, dokud není splněno některé z ukončovacích kritérií. Tím může být například čas, počet iterací nebo dosažení určité úrovně populace potažmo jedince. Tyto operace, jako jsou křížení, mutace nebo evaluace, jsou zachyceny na obrázku 16.



Obrázek 16. Vývojový diagram GA, převzato z [20]

7.1.1. Počáteční populace

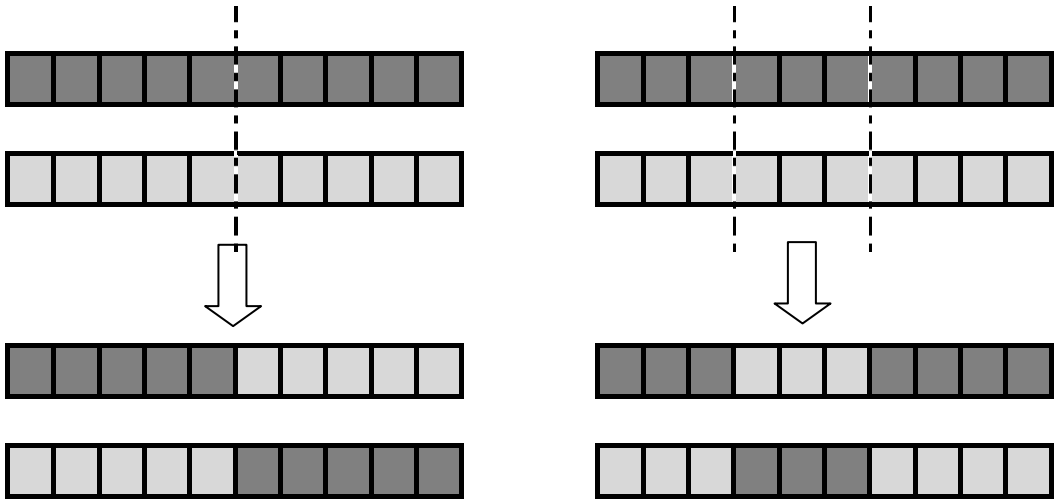
Na tom, jak rychle bude algoritmus konvergovat ke kýženému výsledku, závisí vytvoření počáteční populace. Pro náš příklad budeme experimentovat s několika metodami vytváření výchozí populace a to:

- náhodný výběr bodů
- náhodný výběr bodů, které jsou si blízkými sousedy
- konstruktivní heuristikou NN

V našem příkladě pro genetické algoritmy je proměnná pro velikost počáteční populace nazvaná *Velikost populace*. Při vysoké hodnotě tohoto parametru je třeba delší čas k nalezení dobrého výsledku, zatímco nevýhoda nižší hodnoty spočívá ve větší šanci, že si cesty budou navzájem podobné. Nastává další fáze genetických operátorů, mezi něž patří křížení a mutace.

7.1.2. Křížení

Genetické algoritmy mají mezi všemi evolučními algoritmy díky své fázi křížení nejblíže k přírodnímu vzoru. Princip křížení je založen na rozdělení chromozómu v určitém bodě a nahrazení částí chromozomu jiného. Na obrázku 17. je znázorněno tzv. jedno a dvoubodového křížení [21].



Obrázek 17. jedno a dvoubodové křížení, převzato z [21]

Pokud použijeme jednobodového křížení, oba chromozomy budou rozděleny v jednom určeném bodě. Následně jsou dva noví potomci vytvořeni připojením rozdílných částí

rodičovských chromozomů. V případě dvoubodového křížení jsou genotypy rozděleny ve dvou bodech, přičemž nový chromozom je vytvořen použitím části 1. a 3. z prvního chromozomu a 2. částí chromozomu druhého a opačně.

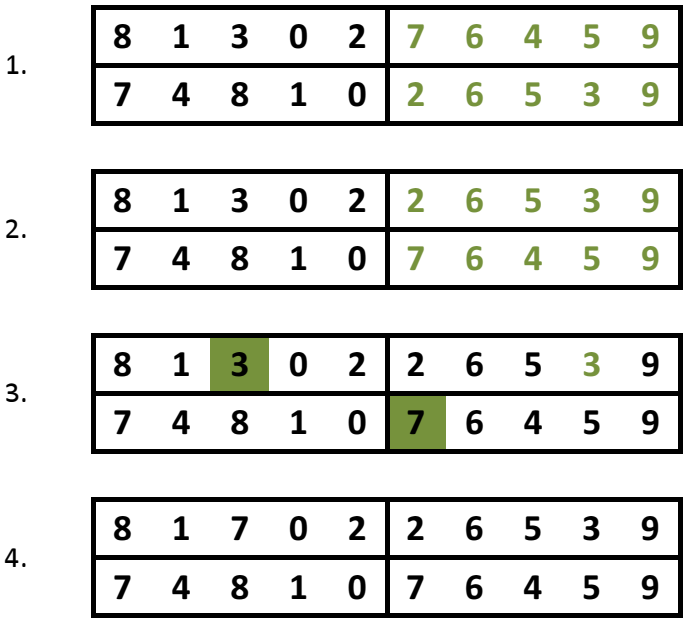
Pro TSP existuje dle [22] několik druhů křížení:

7.1.2.1. Náhodné křížení

V daném genotypu je vybrán náhodně počet prvků, které zůstanou v rámci určitého chromozomu zachovány. Ostatní geny pak budou nahrazeny genem druhého genotypu, a to v případě, že hodnota již není v genotypu obsažena, v opačném případě dojde k použití genu následujícího.

7.1.2.2. Částečně párové křížení

V originále Partial matched crossover, ve zkratce PMX, pracuje v kombinaci s dvoubodovým křížením. V chromozomu jsou náhodně vybrány dvě pozice, např. a, b, kde platí, že a <b a zároveň b<délka chromozomu. V intervalu a až b jsou geny nahrazeny z druhého chromozomu a zbytek původního genotypu je doplněn podle vzorce, kde je v případě zdvojené hodnoty použita hodnota z „oříznuté“ části původního genotypu. Celý tento proces je znázorněn na obrázku 18. [22]



Obrázek 18. PMX křížení, převzato z [22]

Algoritmus vypočítá na základě počtu měst a zadané hodnoty *Procento křížení* body, v nichž se bude genotyp dělit. V následujícím odstavci je část kódu pro krok 2 z obrázku 18., kde dochází k záměně alel jednotlivých chromozomů:

```
Dim achromnew(1, rozmer - 1) As Integer
  For i = 0 To rozmer - 1
      If i >= zlom1 And i < zlom2 Then 'interval od do zlom1 - zlom2
          achromnew(0, i) = achrom(1, i) ' v intervalu krizim
          achromnew(1, i) = achrom(0, i)
      Else
          achromnew(0, i) = achrom(0, i)
          achromnew(1, i) = achrom(1, i)
      End If
  Next
```

Platí tedy, že pokud jsme na pozici genu, která spadá do intervalu tzv. křížení, dojde ke vzájemné záměně v rámci dvou chromozomů. V kroku 3 pokračuje algoritmus ošetřením hodnot v genotypech, které by se mohly vyskytovat dvakrát.

```
For i = 0 To zlom1 - 1
    For j As Integer = zlom1 To zlom2 - 1
        If achromnew(ch, i) = achromnew(ch, j) Then
            achromnew(ch, i) = achromnew(1 - ch, k)
            k += 1
            j = zlom1 - 1
        End If
    Next
Next
```

Zde procházíme tu část pole chromozomu, kde nedošlo ke křížení a jednotlivé body, resp. geny, porovnáváme s „vyměněnou“ částí. Pokud se hodnoty pole shodují, pak je vyšetřovaná hodnota nahrazena z „vyměněné“ části původního chromozomu a proces kontroly se pro vyšetřovaný prvek opakuje. Na obrázku 18. se v prvním kroku jedná o alelu 3, která bude nahrazena alelou 7, dojde k opětovné kontrole duplicity. Pokud by se alela 7 vyskytovala duplicitně, byla by nahrazena následující, a to hodnotou 6.

Chromozomy podrobené křížení jsou vybrány z populace náhodným způsobem. V tomto případě budou kříženy takové genotypy, jejichž hodnota, tedy délka cesty, je minimální. Pro účely křížení zavedeme proměnnou *Velikost skupiny*, která nám udává počet genotypů, z nichž určíme dva nejlepší.

7.1.2.3. Pořadové křížení

Dalším operátorem křížení, které je podobné PMX, se nazývá pořadové křížení, v originále Order Crossover (OX). Proces tohoto křížení je znázorněn na obrázku 19.

1.

8	1	3	0	2	7	6	4	5	9
7	4	8	1	0	2	6	5	3	9

2.

8	1	-	0	-	7	-	4	-	-
-	-	8	1	0	2	-	-	3	-

3.

8	1	0	7	4	-	-	-	-	-
8	1	0	2	3	-	-	-	-	-

4.

8	1	0	7	4	2	6	5	3	9
8	1	0	2	3	7	6	4	5	9

Obrázek 19. OX operátor, převzato z [22]

V prvním kroku je opět zvolen interval ohraničený dvěma hodnotami, přičemž v druhém kroku jsou hodnoty z intervalu prvního chromozomu vyškrtnuty z chromozomu druhého a opačně. Každý chromozom je pak znovu uspořádán takovým způsobem, že jsou hodnoty zapsány mimo původní interval označený indexy. [22]

Zde lze problém s dvojitými hodnotami řešit jiným způsobem, tedy ještě před zapsáním všech prvků genotypu:

```
Dim k As Integer = 0
For i = 0 To rozmer - 1
    Dim zapis As Boolean = True
    For j As Integer = zlom To rozmer - 1
        If achrom(ch, i) = achrom(1 - ch, j) Then
            zapis = False
        End If
    Next
    If zapis = True Then
        achromnew(ch, k) = achrom(ch, i)
        k += 1
    End If
Next
```

Touto částí cyklu procházíme všechny hodnoty prvního genotypu a porovnáváme s prvky v ohraničeném intervalu genotypu druhého. Pokud se alely neshodují (hodnota *zapis* typu boolean je rovna TRUE), prvek je zapsán do nového chromozomu (proměnná *achromnew*). Tímto způsobem je zapsáno celkem n-k prvků, kde *n* je délka chromozomu a *k* délka intervalu indexy ohraničená. Zbylá část genotypu je doplněna standardně překřížením původních dvou chromozomů v kroku 4:

```
For i As Integer = zlom To rozmer - 1
   achromnew(ch, i) = achrom(1 - ch, i)
Next
```

7.1.2.4. Křížení s rekombinací hran ERX

Tento typ operátoru je v současnosti považován za jeden z nejúspěšnějších pro řešení úlohy obchodního cestujícího. Na rozdíl od dvou předchozích metod výše popsaných je výsledkem tohoto křížení ze dvou rodičovských genotypů pouze jeden potomek. [23]

Základní pseudoalgoritmus ERX je dle [24] definován následovně:

1) vytvoř hranovou tabulku pro každý vrchol V_i, kde každá množina S_i vrcholu V_i obsahuje propojené sousední město z mateřských chromozomů
2) vyber náhodně město, označ ho jako „aktuální“ a vlož do nového genotypu
3) odstraň „aktuální město“ ze všech podmnožin S_i
4) jestliže „aktuální město“ má už prázdnou podmnožinu S_i, pak přejdi na krok 6
5) vyber vrchol, který má nejmenší počet prvků v podmnožině S_i. Pokud existuje více takových vrcholů, vyber náhodný. Přidej město jako „aktuální“ a vlož do nového genotypu a přejdi na bod 3.
6) V případě, že jsou již všechny vrcholy použity, ukonči algoritmus, jinak přejdi na krok 2.

V prvním kroku pro ERX definujeme hranovou tabulku

```
Dim aerxsousede(rozmer - 1)() As Integer
```

kde druhý rozměr bude redefinován v rámci smyčky s pevným počtem opakování stanovených na hodnotu počtu měst. Pro každý gen obou chromozomů provedeme šetření, kde zapisujeme prvky do tohoto pole:

```
For pomj As Integer = 1 To rozmer - 2
  Dim j As Integer = achrom(ch, pomj)
  zapis = True
  For t As Integer = 0 To aerxsousede(j).Length - 1
     If achrom(ch, pomj - 1) = aerxsousede(j)(t) Then
         zapis = False
     End If
 Next
 If zapis = True Then
  ReDim Preserve aerxsousede(j)(aerxsousede(j).Length)
  aerxsousede(j)((aerxsousede(j).Length - 1)) = achrom(ch, pomj - 1)
  aerxsousedepocty(j) += 1
  End If
Next
```

Tímto způsobem vznikne tzv. jagged array, volně přeloženo jako „zubatá pole“. Ta mají oproti klasicky definovaným polí rozdílnou hodnotu určitého rozměru, v našem případě je tento rozměr reprezentován počtem sousedů určitého vrcholu. Základní smyčkou procházíme všechny vrcholy, přičemž pro krajní města v genotypu (první a poslední) je třeba pro prozkoumání sousedních genů upravit parametry. V druhém řádku algoritmu dochází k deklaraci proměnné *j*, do které je uloženo číslo města. Následujícím forcyklem procházíme množinu sousedů vybraného města a pokud se soused zprava (*pomj + 1)*, resp. zleva (*pomj - 1)*, nenachází v seznamu, dojde k rozšíření pole a zapsání prvku. Pole *aerxsousedepocty* je definováno počtem vrcholů a obsahuje informaci o počtu jedinečných sousedů.

```
Dim studna3 As New ArrayList
Dim minimum As Integer = aerxsousedepocty.Min
For i As Integer = 0 To rozmer - 1
    If minimum = aerxsousedepocty(i) Then
       studna3.Add(i)
    End If
Next
```

Základní funkcí pro pole pak zjistíme minimální hodnotu a indexy s těmito hodnotami zapíšeme do nově definovaného pole sloužícího k náhodnému výběru vrcholu. Poté nastává fáze zápisu vybraného města do nového chromozomu a vymazání z množiny sousedů.

Operátor křížení slouží v tomto algoritmu k diverzifikovanému prohledávání stavového prostoru. Jednotlivé průběhy pro instanci Eil51, tedy kvalita řešení jednotlivých operátorů, jsou uvedeny v příloze.

7.1.3. Mutace

Operátor mutace náhodně vybírá pozici v chromozomu a mění pozice genů, čímž dochází k modifikaci informací. A právě prováděním nahodilých nepravidelných změn v chromozomu zajistí, že je probádáno takových částí stavového prostoru, kterých by za pouhé selekce a křížení dosaženo nebylo. [25]

Proměnná *Procento mutace* udává, kolik % chromozomů z populace bude náhodně vybráno. Proměnnou *pocetmutovanych* určíme jako:

```
Dim pocetmutovanych As Integer = velikostpopulace /
procentomutovanych
```

Následně vytvoříme cyklus s pevným počtem opakovaní, tedy do hodnoty *pocetmutovanych*, kde provedeme náhodný výběr chromozomů z populace a na tuto nově vzniklou skupinu bude aplikován operátor mutace. Parametr *pocetopakovani_mutace* udává, kolikrát dojde v rámci jednoho genotypu k pokusu o záměnu. Jedná se tedy o kritérium ukončení, které může být stejně tak nastaveno například na maximální počet iterací, při kterých již nedošlo ke zlepšení.

7.1.3.1. Mutace výměny

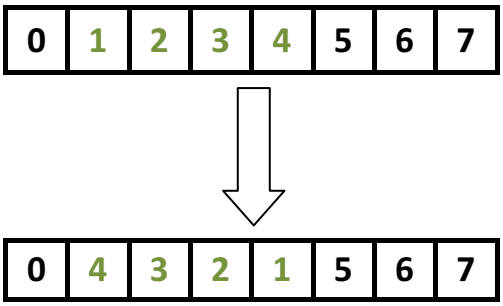
Tento operátor mutace, v originále Exchange Mutation (EM), pracuje na principu náhodného výběru dvou genů v chromozomu a jejich vzájemné záměny, jak je uvedeno níže. [26]

```
For k As Integer = 0 To pocetopakovani_mutace
    Dim nahoda As Integer = rnd.Next(0, rozmer)
    Dim nahoda2 As Integer = rnd.Next(0, rozmer)
    Dim pomocny As Integer = amutace(i, nahoda)
    amutace(i, nahoda) = amutace(i, nahoda2)
    amutace(i, nahoda2) = pomocny
Next
```

Parametry *nahoda* a *nahoda2* jsou vygenerovány jako náhodné číslo na základě taktu procesoru a udávají, jaké dva geny budou zaměněny.

7.1.3.2. Mutace převrácení

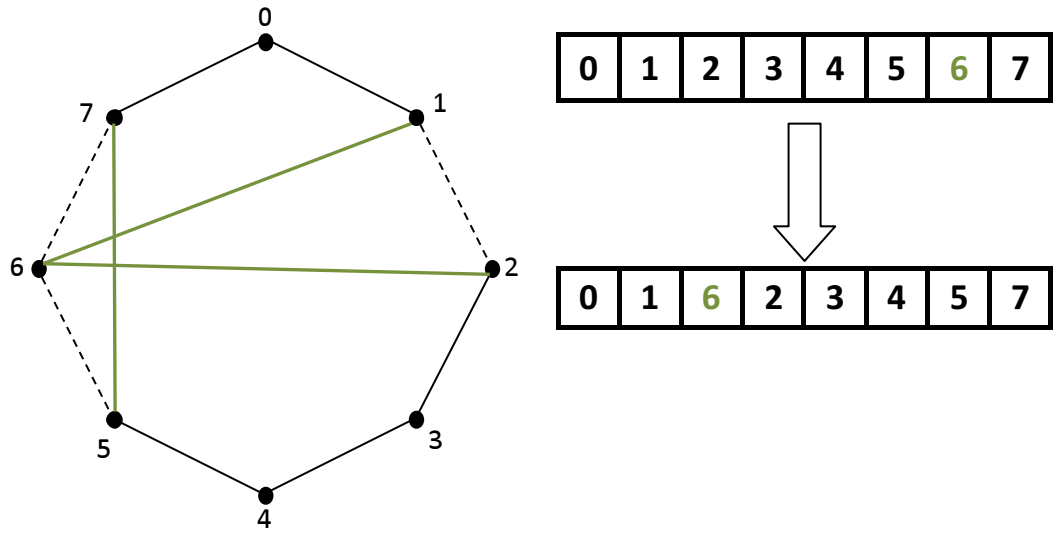
Tento způsob mutace je znám jako 2-opt, popsaný v kapitole Iterativních heuristik. Opět jsou vybrány dva náhodné body, v tomto případě určující interval, ve kterém dojde k inverzi genů, znázorněných na obrázku 20.



Obrázek 20. Mutace inverzí, převzato z [26]

7.1.3.3. Mutace vložení

Operátor vložení vybírá namátkově město a vkládá ho na náhodnou pozici. Obrázek 21. popisuje tento operátor při náhodně zvolených bodech 7, 3, kde hodnota 7 znamená pozici genu, který bude přesunut, číslo 3 uvádí jeho umístění.



-

Obrázek 21. Mutace vložení, převzato z [2]

Před samotnou realizací mutace spočítáme, stejně jako u předchozích dvou typů mutací, zda nová cesta bude kratší než původní. Získáním součtu vzdáleností mezi městy z pole *maticevzdalenosti* pro dvojice (6,7);(6,5);(1,2) jako část cesty původní a porovnání součtu třech dvojic (6,1);(6,2);(5,7) určíme, zda dojde k záměně genů.

Pro všechny vypsané typy mutací tedy platí porovnání původní cesty s nově navrhovanou. Pokud je nová cesta kratší než původní, dojde k zápisu chromozomu, algoritmus pokračuje dále. Úlohou mutace v procesu GA tohoto algoritmu je konvergence

k suboptimálnímu řešení pro určitou podmnožinu stavového prostoru. Výsledky konvergence jednotlivých metod mutace pro instanci Eil51 jsou uvedeny v příloze.

7.1.4. Selekce

Po provedení operátorů křížení je použita metoda selekce, sloužící ke stanovení další generace populace. V této práci jsou použity dva druhy selekce – tzv. ROULETTE WHEEL a výběr nejlepších.

7.1.4.1. Roulette Wheel

V GA je běžné vybírat takové chromozomy, jejichž funkce vhodnosti (z anglického fitness function) je vysoká. Pro tuto proceduru poslouží metoda často používaná, nazývaná Roulette Wheel. V tabulce 15. je uvedeno pět genotypů s popisem reprezentující jejich cestu a délku cesty. Hodnotu fitness je třeba vyhodnotit inverzně, tj. pro nejmenší hodnotu délky cesty určit jako nejvyšší fitness.

Tabulka 15. Roulette wheel

Číslo	Chromozom	Délka cesty	Fitness	RW %
1	2-4-3-7-5-6-1-9-8-10	540	1,08	5,72%
2	4-9-3-2-6-5-7-1-10-8	100	5,48	29,04%
3	10-3-2-1-7-4-9-6-5-8	221	4,27	22,63%
4	8-1-7-5-6-10-2-9-3-4	302	3,46	18,34%
5	10-9-4-8-6-7-5-2-1-3	190	4,58	24,27%
			18,87	100,00%

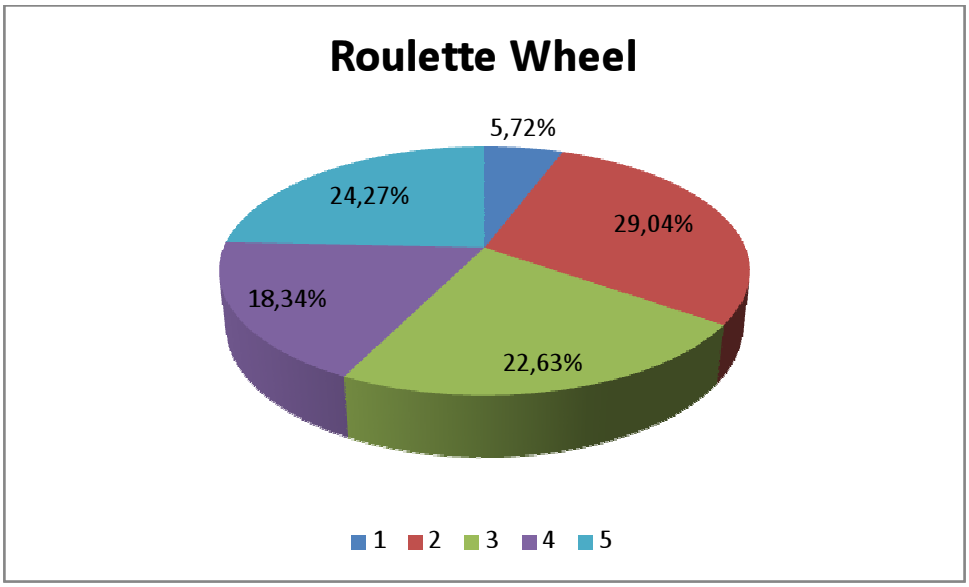
V tomto případě je hodnota Fitness určena jako

$$F(i) = \left[\frac{\max\big(D(i)\big) - D(i)}{100}\right]$$

kde je $\max\big(D(i)\big)$ stanovena jako maximální hodnota z množiny cest vynásobena koeficientem, který je vyšší než 1, a to z důvodů nenulovosti hodnoty Fitness(i) pro nejvyšší D(i). Sloupec RW % je pak definován jako

$$P(i) = \frac{F(i)}{\sum F(i)}$$

Rozložení vhodnosti jednotlivých chromozómů pro konkrétní příklad z tabulky je zakresleno v grafu 3.



Graf 3. Rozložení vhodnosti chromozomů, převzato z [28]

Pro účely selekce byla zavedena proměnná *Velikost skupiny*, jejíž hodnota určuje počet chromozómu, které budou náhodně vybrány pro účely mutace. Ze skupiny o počtu X genotypů jsou vybrány 2 s nejvyšší fitness function a ty jsou použity pro proces mutace. Po provedení operátoru dochází k vyhodnocení nově vytvořených chromozomů a výběru pěti dle metody Roulette Wheel (RW). Pseudoalgoritmus pro tuto metodu lze zaspat takto:

1) vypočti délku cest nově vytvořených chromozomů z křížení
2) vypočti fitness pro všechny chromozomy ve skupině
3) urči procenta RW jednotlivých chromozomů
4) spouštěj ruletu, dokud nebude vybráno stanovené množství jedinců

7.1.4.2. Výběr nejlepších

I v tomto případě nejdříve dopočítáme cesty nově vzniklých chromozomů. Každou nově získanou cestu pak porovnáváme s množinou cest použitých při prvotním náhodném výběru do skupiny. Pokud je pak nová cesta kratší než nejhorší ze skupiny, dojde k jejímu přepsání včetně chromozomu v populaci.

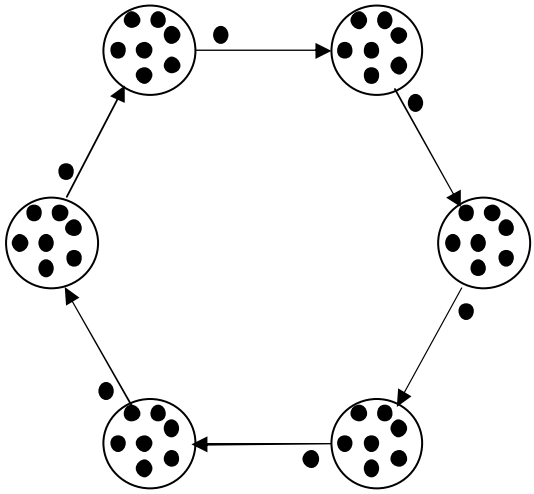
7.1.5. Použití paralelizace pro GA

V běžné populaci, vyskytující se v přírodě, jsou miliony jedinců nebo skupin, kteří existují a vyvíjejí se souběžně. Tento způsob souběžnosti inspiroval také k paralelnímu zpracování genetických algoritmů. Základní metody simultánního zpracování rozdělujeme dle [27] na dva typy:

- ostrovní model
- buněčný algoritmus

7.1.5.1. Ostrovní model

Tento typ, v originále Island Model, patří do skupiny distribučních evolučních algoritmů (EA). Zde se distribucí myslí přenášení chromozómů mezi jednotlivými populacemi, které vznikají rozložením populace celkové. [27] Celý proces ostrovního modelu je znázorněn na obrázku 22., kde kruhy představují subpopulace a šipky směr migrace chromozomů.



Obrázek 22. Ostrovní model, převzato z [28]

Parametry určující počet a způsob přenosu mezi subpopulacemi mohou být různé. Pro naše testování byl zvolen jednoduchý postup, kdy vybíráme ze subpopulace vždy jeden chromozom s nejlepším ohodnocením a předáváme subpopulaci následující, resp. poslední předává první. Chromozom, který bude nahrazen, má v dané subpopulaci nejnižší ohodnocení, tedy nejdelší cestu. V našem případě použijeme kompletní algoritmus, kde vybíráme mezi dvěma typy vytvoření populace a třemi možnostmi pro mutaci a křížení. Zavedeme dva nové parametry:

Pocetsubpopulaci, jehož hodnota udává počet populací, se kterými budeme pracovat

Pocetopakovani - do hodnoty velikosti této proměnné bude pracovat hlavní smyčka

Na základě první proměnné rozšíříme z původního algoritmu pole *apopulace* o další, tedy třetí rozměr, do něhož budeme zapisovat číslo populace. Po prvním provedení smyčky zahrnující náhodné vytvoření populace, ohodnocení, křížení a mutaci, provedeme výpočet

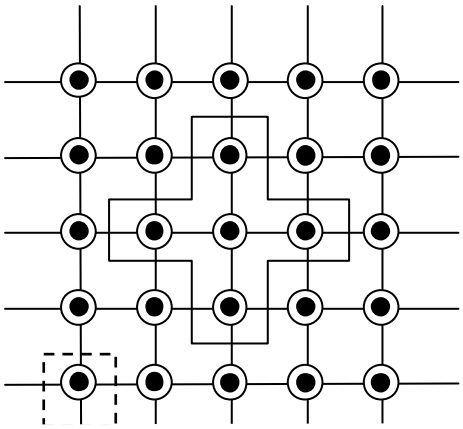
nejlepší a nejhorší cesty v rámci každé subpopulace. Poté dojde k nahrazení nejhoršího chromozomu ze subpopulace j + 1 chromozomem nejlepším ze subpopulace j, jak je uvedeno níže:

```
 For j As Integer = 0 To pocetpop - 2
    prvnirozmerworst = worstcestaglobindex(j + 1)
    prvnirozmerbest = bestcestaglobindex(j)
    For i As Integer = 0 To rozmer - 1
       apopulaceglob(prvnirozmerworst,i, j + 1)
         =apopulaceglob(prvnirozmerbest, i, j)
    Next
Next
```

Na konci každého opakování smyčky dojde k vyhodnocení nejlepší cesty a zapsání do vítězné matice, přičemž při dalším průchodu je nová nejlepší délka porovnána s nejlepší dosaženou a dojde k eventuelnímu novému zápisu do matice vítězné.

7.1.5.2. Buněčný algoritmus

Metoda buněčného algoritmu spočívá ve zpracování určité části subpopulace, založené na principu výběru jedince a jeho nejbližšího okolí. Nejběžnější topologie pro tento algoritmus je tzv. toroidní mřížka, znázorněná na obrázku 23, která je v tomto příkladu dvojrozměrná a hodnota sousedství stanovena na 5. [28]



Obrázek 23. Toroidní mřížka buněčného algoritmu, převzato z [28]

V buněčném algoritmu jsou na vybraného jedince a jedinců v jeho bezprostředním okolí aplikovány vybrané operátory GA. I zde by bylo možné, podobně jako u ostrovního modelu, zavést paralelní model a to právě na principu výběru rozdílných, nesousedních jedinců z množiny populace a jejich následné zpracování.

7.1.6. Výsledky GA

V tabulce 16. jsou uvedeny výsledky ostrovního modelu pro instanci Eil51 na jednotlivých operátorech křížení a mutace s náhodně vytvořenými populacemi a selekcí RW.

Tabulka 16. Výsledky operátorů křížení a mutace na Eil51

Křížení / Mutace		PMX	OX	ERX	RND
Exchange	QSbest	94,25%	92,81%	89,12%	89,12%
	QSavg	87,46%	88,66%	84,17%	86,15%
Inversion	QSbest	98,61%	98,38%	98,61%	98,84%
	QSavg	97,57%	97,51%	96,82%	98,07%
Insertion	QSbest	97,71%	97,93%	97,93%	99,07%
	QSavg	96,36%	96,27%	95,11%	96,88%
RND	QSbest	99,07%	99,30%	99,07%	99,30%
	QSavg	97,93%	97,95%	97,42%	98,11%
Best	QSbest	99,07%	99,30%	99,07%	99,30%
	QSavg	94,83%	95,10%	93,38%	94,80%

QSbest je nejlepší kvalita řešení, QSavg pak průměrná. Hodnoty jsou získány z 10ti nezávislých spuštění. Nastavení dalších parametrů je uvedeno v tabulce 17.

Tabulka 17. Nastavené parametry pro Eil51

Parametr	Hodnota
Velikost populace	100
Velikost skupiny	5
Počet opakování	200
Procento mutace	40
Opakování mutace	10
Počet subpopulací	5
Počet opakování subpop	20

Z tabulky 16. je patrné, že nejlepších výsledků bylo dosaženo při kombinaci všech tří typů mutací s operátorem křížení OX. Výsledky pro další instance měst s tímto typem křížení a náhodnými mutacemi jsou uvedeny v tabulce 18,, kombinace parametrů pro výpočty v tabulce 19.

Tabulka 18. Výsledky GA pro jednotlivé instance

Instance	Kvalita řešení		Průměrný čas v s 2.sestava		Kombinace parametrů
	avg	best	sekvenčně	paralelně	
Bays29	100,00%	100,00%	2,62	1,72	1
Eil51	99,44%	100,00%	7,85	3,67	2
Berlin52	99,71%	100,00%	7,18	3,72	1
KroA100	99,51%	100,00%	134,95	61,23	3
Ch150	99,66%	99,74%	153,94	376,90	4
A280	99,30%	99,85%	331,16	161,60	5
Lin318	97,72%	98,22%	1749,03	786,83	6
Průměr	99,33%	99,69%			

Tabulka 19. Kombinace parametrů pro jednotlivé instance

Parametr/kombinace	1	2	3	4	5	6
Velikost populace	100	100	100	100	100	100
Velikost skupiny	5	5	5	5	5	5
Počet opakování	200	200	200	200	100	200
Procento mutace	40	40	40	40	40	40
Opakování mutace	10	10	20	30	180	140
Počet subpopulací	5	5	5	5	7	7
Počet opakování subpop	20	40	200	350	140	350
Počáteční populace	RND	RND	RND	NN	NN	NN

Z šetření vyplývá:

- při větším počtu subpopulací klesá pravděpodobnost uvíznutí v lokálním minimu, roste čas výpočtu při zachování původní konvergence
- vyšší hodnota počtu opakování mutací způsobuje uvíznutí v LM, čas výpočtu se snižuje, proto je u větších instancí výhodnější vytvořit počáteční populaci z NN a zavést nízký počet opakování operátorů
- u instance a280 je, vzhledem ke svému charakteru vzdáleností mezi městy, nejvýhodnější výchozí populace metodou NN a následné použití vysokého počtu operátorů mutací. K dosažení globálního minima by bylo účinné zavést jako operátor mutace vyšší úroveň výměny tahů k-opt.

7.1.7. Dílčí závěr

Z počátku vytváření algoritmu GA byly pokusy o paralelizaci úlohy na úrovni jednotlivých operátorů. Ve většině případů ovšem efektivita dosahovala obdobných hodnot jako u 2-opt nebo NN. Z tohoto důvodu byla zavedena paralelizace na úrovni jednotlivých

subpopulací, čímž se díky předávání jednotlivců zvýšila také efektivita v nalezení kvalitnějšího řešení. Algoritmus by bylo možné v několika částech značně modifikovat a tím změnit efektivitu v konvergenci nebo diverzifikaci stavového prostoru, resp. změnit rychlost výpočtu nebo dosáhnout lepší kvality nalezených řešení. Pro dosažení nebo alespoň přiblížení se globálnímu optimu je zcela zásadní nastavení parametrů operátorů, a to v závislosti na dané velikosti instance.

8 Závěr

Cílem této práce bylo provést paralelizaci metod používaných pro problém obchodního cestujícího. U vybraných metod byla provedena šetření, zaměřena především na časový rozdíl, respektive efektivitu paralelního a sekvenčního zpracování. Důležitým ukazatelem při ladění algoritmů a optimálního nastavování parametrů byla také hodnota kvality dosahovaných výsledků. Test z obou hledisek probíhal na vybraných instancích s velikostí 29ti až 318ti městy.

Pro splnění cíle byly vybrány metody z různých skupin, a to z důvodů odlišného použití paralelizace. V první části konstruktivních heuristik se jedná o čistou dekompozici úlohy zaměřenou částečně deterministickým přístupem. Shlukováním vzniká dekompozice na úrovni datové, kde se otevírají další možnosti použití paralelizace. U konstruktivní metody F&F lze uvést, že pro menší instance je algoritmus schopen nalézt v polynomiálním čase řešení blížící se globálnímu optimu, přičemž s rostoucím počtem měst kvalita klesá, až po úroveň cca 90% při instanci 318 měst. F&F, patřící do skupiny tzv. hloubkových metod, by byla použitelná také jako zlepšovací heuristika. Koeficient efektivity dosahoval u všech třech sestav v průměru hodnoty 0,6, což je vzhledem ke spodní hranici efektivity průměru 0,3 ucházející výsledek.

Další kapitola, zabývající se zlepšováním řešení, konkrétně 2-opt, řeší paralelizaci na úrovni simultánního procházení stavového prostoru. Zde je efektivita paralelního algoritmu velmi nízká a to z důvodu nižších výpočtů uvnitř smyček, stejně jako u původní metody NN. 2-opt iterativní technika v kombinaci s výchozími cestami vygenerovanými F&F dosahuje na skupině testovaných instancí měst kvality řešení v průměru 98%.

Metaheuristika GA užívá paralelní zpracování na principu několika nezávislých populací, které si mezi sebou vyměňují stanovený počet jedinců. Zde je efektivita paralelního algoritmu obstojná, stejně jako u F&F, a dosahuje v průměru hodnoty 0,48. Obdobným způsobem jako v GA lze provést paralelizace rovněž u ACO, jelikož také vychází z populační simulace. U metaheuristik TS a SA by bylo možné vytvořit obdobné simultánní zpracování jako u heuristiky 2-opt, tedy na úrovni vícenásobného procházení stavového prostoru.

Seznam použité literatury

[1] APPLEGATE, David L., Robert E. BIXBY, Vašek CHVÁTAL a William J. COOK. *The traveling salesman problem*: *a computational study*. Princeton: Princeton University Press, 2006, ix, 593 s. ISBN 978-0-691-12993-8.

[2] GUTIN, Gregory a Abraham P PUNNEN. *The traveling salesman problem and its variations*. New York: Springer, c2007, xviii, 830 p. ISBN 03-064-8213-4.

[3] BURKE, Edmund a Graham KENDALL. *Search methodologies*: *introductory tutorials in optimization and decision support techniques*. New York: Springer, 2005, vi, 620 p. ISBN 03-872-8356-0.

[4] TOUB, Stephen. A. *Patterns of parallel programming* [online]. 2010 [cit. 2012-04-12]. Dostupné z: http://download.microsoft.com/download/3/4/D/34D13993-2132-4E04-AE48-53D3150057BD/Patterns_of_Parallel_Programming_VisualBasic.pdf

[5] AKHTER, Shameem a Jason ROBERTS. *Multi-core programming*: *increasing performance through software multi-threading* [online]. 1. vyd. New York: Intel Press, 2006, 336 s. [cit. 2012-04-15]. ISBN 09-764-8324-6. Dostupné z: http://www.intel.com/intelpress/samples/mcp_samplech01.pdf

[6] EL-REWINI, Hesham. *Advanced computer architecture and parallel processing*. Hoboken: Wiley, c2005, s. 1-18. ISBN 0-471-46740-5.

[7] SUN, Xian-He a Yong CHEN. Reevaluating Amdahls law in the multicore era. *Journal of Parallel and Distributed Computing* [online]. 2010, roč. 70, č. 2, s. 183-188 [cit. 2012-04-20]. ISSN 07437315. DOI: 10.1016/j.jpdc.2009.05.002. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0743731509000884

[8] RAMANATHAN, R.M. 4. *Intel Multi-Core Processors* [online]. [cit. 2012-04-22]. Dostupné z: http://www.intel.com/technology/architecture/downloads/quad-core-06.pdf

[9] *ISSCC 2011 TRENDS REPORT* [online]. 2011 [cit. 2012-04-23]. Dostupné z: http://isscc.org/doc/2011/2011_Trends.pdf

[10] HANÁK, Ján. *Praktické paralelné programovanie v jazykoch C# 4.0 a C*. Vyd. 1. Brno: Artax, 2009, 132 s. ISBN 978-80-87017-06-7.

[11] Parallel Programming in the .NET Framework. [online]. [cit. 2012-04-30]. Dostupné z: http://msdn.microsoft.com/en-us/library/dd460693.aspx

[12] MACKEY, Alex. *Introducing .NET 4.0 with Visual studio 2010*. New York: Distributed to the book trade worldwide by Springer Verlag, c2010, xxxi, 471 p. Expert's voice in .NET. ISBN 9781430224563.

[13] MCMULLEN, P.R. a Peter TARASEWICH. A beam search heuristic method for mixed-model scheduling with setups. *International Journal of Production Economics* [online]. 2005, roč. 96, č. 2, s. 273-283 [cit. 2012-05-11]. ISSN 09255273. DOI: 10.1016/j.ijpe.2003.12.010. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0925527304002166

[14] TALBI, El-Ghazali. *Parallel combinatorial optimization*. Hoboken, N.J.: Wiley-Interscience, c2006, xvi, 330 p. ISBN 978-047-1721-017.

[15] GLOVER, Fred. 6. *A Template For Scatter Search And Path Relinking* [online]. 1998 [cit. 2012-05-15]. Dostupné z: http://leeds-faculty.colorado.edu/glover/SS-PR%20Template.pdf

[16] HRUSCHKA, E. R., L. N. DE CASTRO a R. J. G. B. CAMPELLO. *Hybrid evolutionary algorithms*: *Clustering Gene-Expression Data: A Hybrid Approach that Iterates Be tween k-Means and Evolutionary Search* [online]. Berlin: Springer, 2007, s. 313-335 [cit. 2012-05-25]. ISSN 978-3-540-73297-6. DOI: 10.1007/978-3-540-73297-6_12.

[17] Using genetic algorithm for selection of initial cluster centers for the K-means method. RUTKOWSKI, Leszek. *Artificial intelligence and soft computing*: *10th international conference, ICAISC 2010, Zakopane, Poland, June 13-17, 2010*. New York: Springer, c2010, s. 165-172. Lecture notes in computer science, 6113. ISBN 9783642132315.

[18] AARTS, Emile H. L. a J. K. LENSTRA. *Local search in combinatorial optimization*. Princeton: Princeton University Press, 2003, xii, 512 s. ISBN 06-911-1522-2.

[19] LAARHOVEN, P. a E. AARTS. *Simulated annealing*: *theory and applications*. Dordrecht: Kluwer Academic Publishers, 1987, xi, 186 s. ISBN 90-277-2513-6.

[20] ABU-TAIEH, Evon M, Asim A EL-SHEIKH a Jeihan ABU-TAYEH. *Utilizing information technology systems across disciplines*: *advancements in the application of computer science*. Hershey, PA: Information Science Reference, c2009, xix, 343 p. ISBN 16-056-6617-3.

[21] WEISE, Thomas. *Global Optimization Algorithms – Theory and Application* [online]. 2009 [cit. 2012-06-10]. Dostupné z: http://www.it-weise.de/projects/book.pdf

[22] JOHANNES J. SCHNEIDER, Johannes J.Scott Kirkpatrick. *Stochastic Optimization*: *Application of Genetic Algorithms to TSP*. Berlin [etc.]: Springer, 2006, s. 415-422. ISBN 9783540345602.

[23] HYNEK, Josef. *Genetické algoritmy a genetické programování*. 1. vyd. Praha: Grada, 2008, 182 s. ISBN 978-80-247-2695-3.

[24] ICAN, Özgür Saygın, Fatih SEMIZ a Çağlar SEYLAN. New crossover technique for genetic algorithm for solving traveling salesman problem. [online]. [cit. 2012-06-17].

Dostupné z:
http://metu.academia.edu/FatihSemiz/Papers/423400/NEW_CROSSOVER_TECHNIQUE_FOR_GENETIC_ALGORITHM_GA_FOR_SOLVING_TRAVELLING_SALESMAN_PROBLEM_TSP_

[25] DEEP, Kusum a Hadush MEBRAHTU. Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman problem. *International journal of combinatorial optimization problems and informatics IJCOPI* [online]. 2011, s. 2-24 [cit. 2012-06-20]. ISSN 2007-1558. Dostupné z: http://ijcopi.org/ojs/index.php?journal=ijcopi&page=article&op=download&path%5B%5D=66&path%5B%5D=105

[26] BUCKLAND, Mat. *AI techniques for game programming*. Cincinnati, Ohio: Premier Press, c2002, xxxii, 448 p. ISBN 19-318-4108-X.

[27] ALBA, Enrique a Bernabé DORRONSORO. IEEE Transactions on Evolutionary Computation: The Exploration/Exploitation Tradeoff in Dynamic Cellular Genetic Algorithms. [online]. 2005 [cit. 2012-06-23]. DOI: 1089-778X. Dostupné z: http://150.214.190.154/docencia/sf1/Enrique-Alba.pdf

[28] ALBA, Enrique. *Cellular genetic algorithms*. 1st ed. New York: Springer, 2008, p. cm. ISBN 03-877-7610-9.

Seznam zkratek

TSP	Traveling Salesman Problem
CPU	Central Process Unit
GA	Genetic Algorithm
SA	Simulated Annealing
SU	Speed Up
VB	Visual Basic
NN	Nearest Neighborhood
LK	Lin-Keringhan
SF	Sequential Fan
F&F	Filter and Fan
PMX	Parially matched crossover
OX	Order crossover
ERX	Edge Recombination crossover
RW	Roulette Wheel
EA	Evolution Algorithm
ACO	Ant Colony
TS	Tabu Search
RND	Random

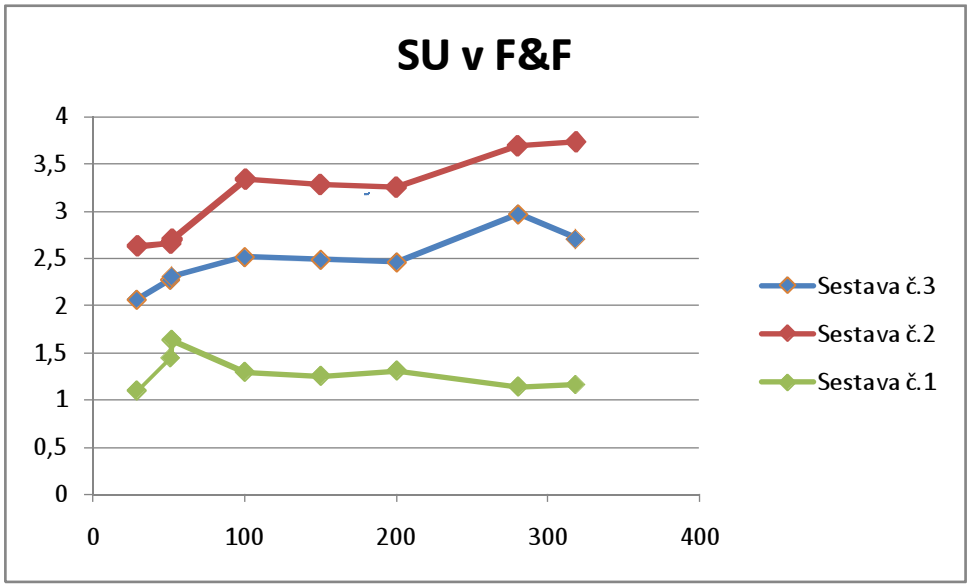
Seznam obrázků

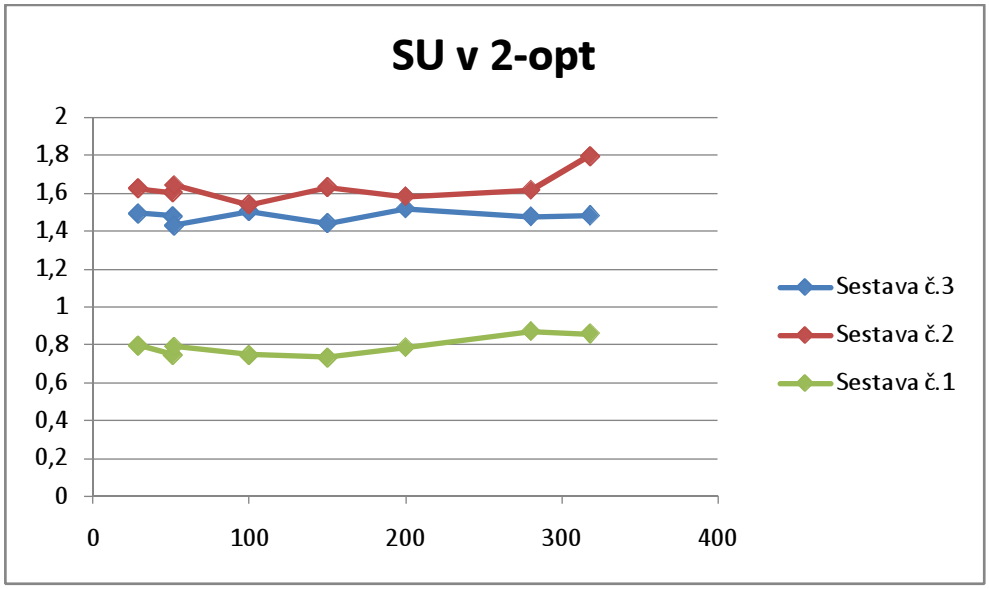
Seznam tabulek

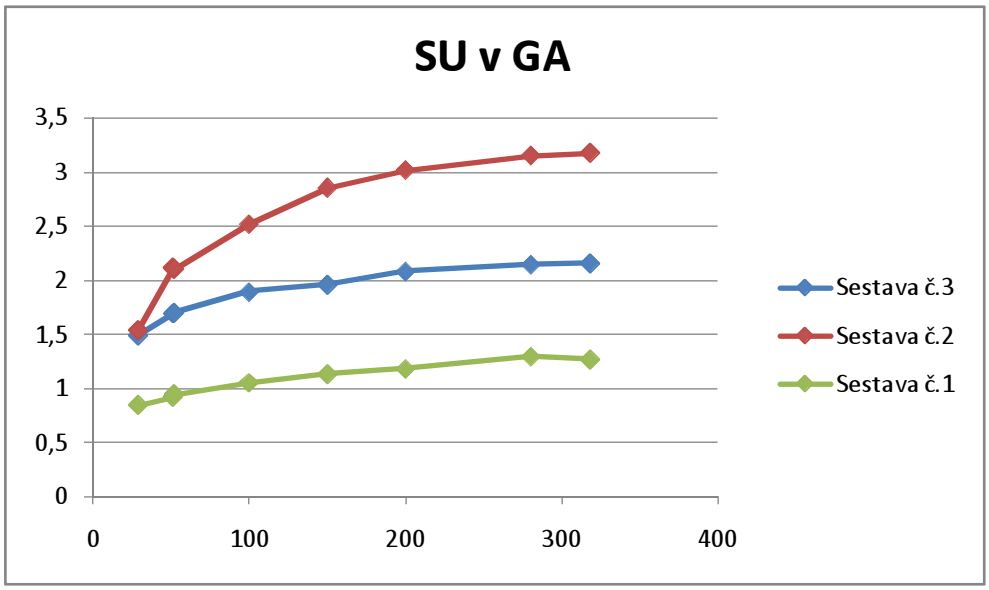
Seznam grafů

Seznam příloh

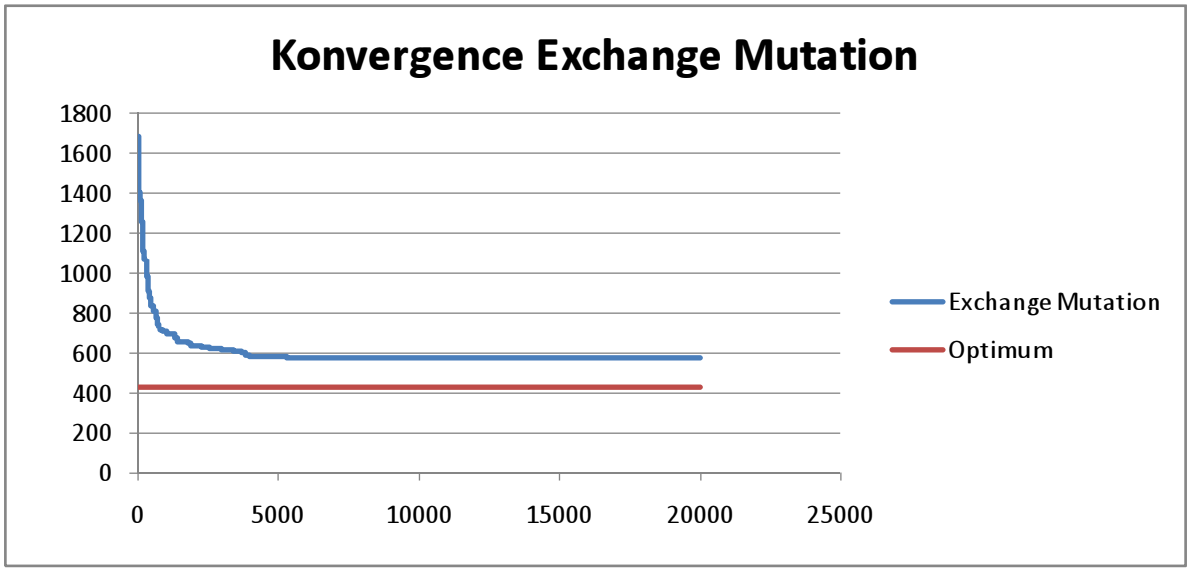
Příloha 1. **Grafy SU jednotlivých metod**

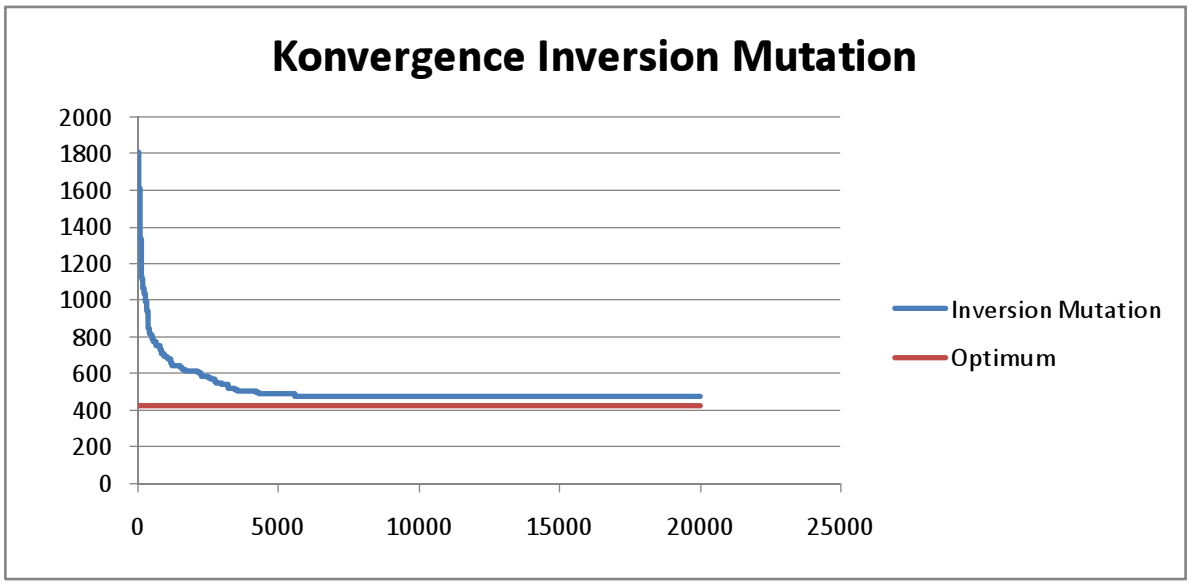


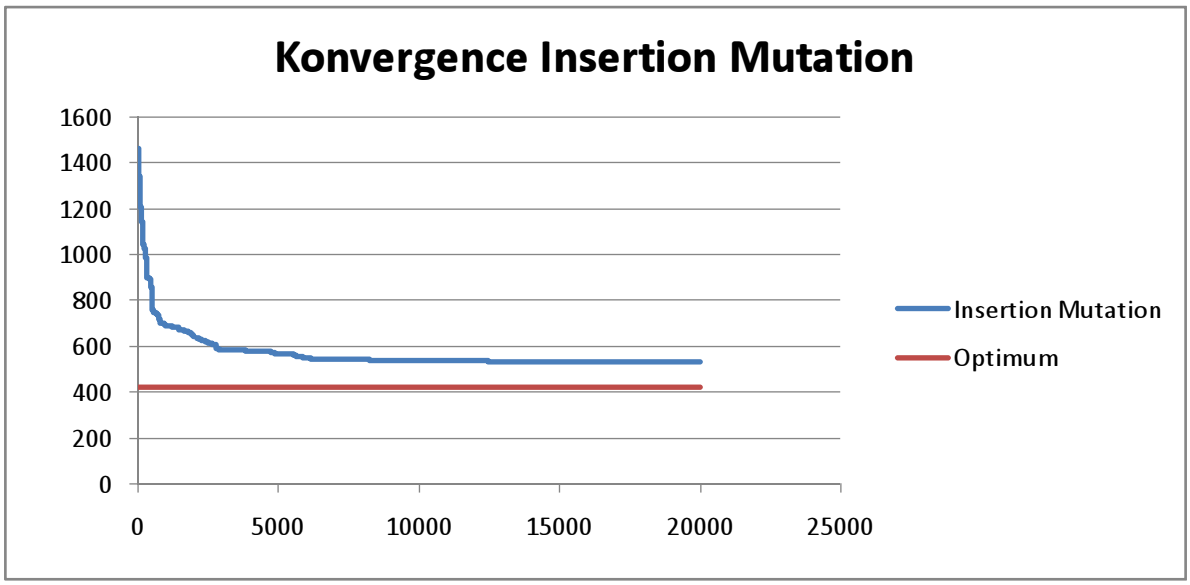




Příloha 2. Konvergence mutací







Příloha 3. Průběh křížení