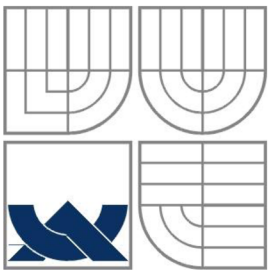
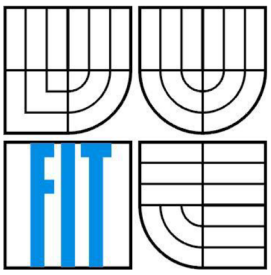


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ÚLOHA OBCHODNÍHO CESTUJÍCÍHO

TRAVELLING SALESMAN PROBLEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Adam Kolář

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. František V. Zbořil CSc.

BRNO 2011

Abstrakt

Cílem této bakalářské práce je navrhnout prostředí testující problém obchodního cestujícího a porovnat efektivitu jednotlivých přístupů k řešení.

V první části jsou diskutovány možnosti genetických algoritmů v závislosti na nastavení křížení, mutací a velikosti populace.

V druhé části jsou na stejný problém použity dva druhy neuronových sítí. Za zástupce samoučící varianty byla zvolena Kohonenova neuronová síť. Hopfieldova neuronová síť reprezentuje metodu minimalizace energetické funkce s pevným nastavením koeficientů. U obou neuronových sítí byly popsány možné výhody a nevýhody aplikace. V závěru byly všechny zjištěné poznatky interpretovány ve společném kontextu.

Abstract

The aim of this bachelor's thesis is to design a testing environment for the traveling salesman problem and compare the effectiveness of different approaches to the solution.

The first part discussed the possibility of genetic algorithms, depending on the setting of a crossover, mutations and population size.

In the second part, there is the same problem using two types of neural networks. The representative of the self-learning net was chosen Kohonen neural network. Hopfield neural network represents a method of minimizing the energy function with fixed coefficients. At both neural networks, there were described possible advantages and disadvantages. In the end, all the findings were interpreted in a global context.

Klíčová slova

úplný graf, Hamiltonovská kružnice, problém obchodního cestujícího, NP-úplný problém, Darwinova evoluční teorie, genetický drift, genetické algoritmy, mutace, křížení, fitness funkce, neuronové sítě, Kohonenova neuronová síť, Hopfieldova neuronová síť, Lyapunova funkce

Keywords

entire graph, Hamiltonian cycle, travelling salesman problem, NP-complete problem, Darwin evolution theory, genetic drift, genetic algorithms, mutation, crossover, fitness function, neural network, Kohonen neural network, Hopfield neural network, Lyapunov function

Úloha obchodního cestujícího

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. Ing. Františka V. Zbořila CSc. .

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adam Kolář
3.3.2011

Poděkování

Na tomto místě bych rád poděkoval panu doc. Ing. František V. Zbořil CSc. za odborné vedení a cenné rady při řešení mé práce. Dále pak panu Ing. Jaroslavu Rozmanovi za konzultaci příspěvku do soutěže EEICT, který reflektoval závěry této práce. Rád bych také poděkoval slečně bc. Janě Koplové za podnětné návrhy při úpravách vzhledu aplikace, jež je jedním z výstupů mého výzkumu.

Obsah

1 Úvod do oblasti umělé inteligence

První zmínky o umělé inteligenci se začaly objevovat ve 40 letech 20. Století. Nejprve s definicí umělého neuronu, poté s pravidly určování umělých neuronových sítí. V roce 1950 Alan Turing ve svém článku *Computing Machinery and Intelligence* definoval další principy učení, posilovaného učení, genetických algoritmů a také navrhl tzv. Turingův test.

Tento test zjednodušeně tvrdil, že stroj můžeme prohlásit inteligentním poté, co rozhodčí nerozezná, se zadanou pravděpodobností, jeho lingvistický výstup od promluvy lidské bytosti. Jako protiargument tomuto testu byl stavěn argument čínského pokoje, který tvrdil, že schopnost smysluplně odpovídat na zadané otázky ještě nemusí být důkazem umělé inteligence. Ukazoval to na příkladu člověka v knihovně, která obsahovala všechny možné čínské texty. Pokud by pak člověk, který tomuto jazyku vůbec nerozumí, dostával otázky, mohl by pouhým hledáním zadaného textu získat potřebné informace k odpovědi, aniž by přitom vůbec tušil, o čem je řeč. Jeho činnost by tedy mohl zastat nemyslící stroj. Jedná se pouze o hypotetický, nicméně platný argument.

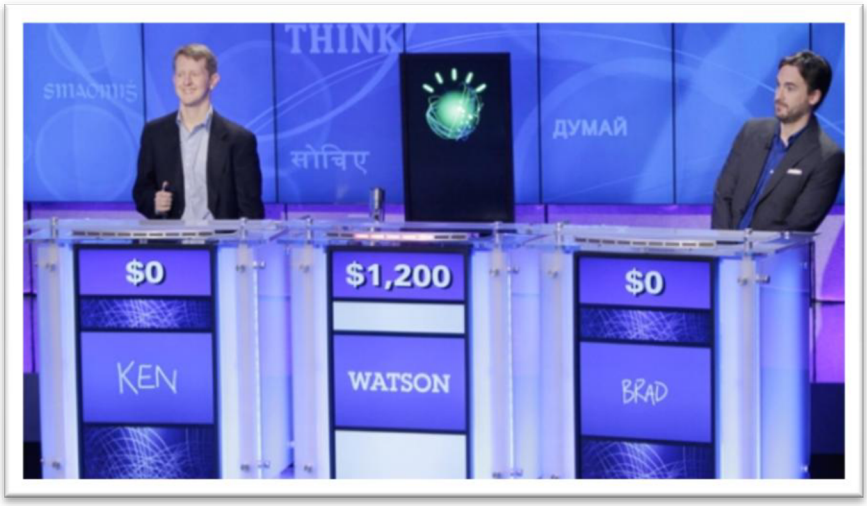
Definic inteligence a to nejen inteligence stroje, se objevilo několik. Obecně můžeme tvrdit, že se jedná o schopnost řešit nově vzniklé problematické situace, učit se a poučit se ze zkušeností a rozpoznávat důležité skutečnosti a vztahy.

Samotný pojem umělá inteligence se začal používat až po konferenci v roce 1956 v Dartmouth College v Hanoveru, během níž účastnící nabyli přesvědčení, že umělá inteligence bude založena především na dedukci, kterou v následujících letech zvládnou výkonné počítače a budou schopny nahradit lidskou inteligenci. Tento předpoklad se v dalších letech ukázal jako přehnaný.

Vývoj umělé inteligence (dále AI) by se dal následně rozdělit do tří charakteristických období. Do konce 60. let se objevily první programy pro zpracování přirozeného jazyka, pro hraní her (dáma) a řešení obecných úloh. Byly prvně použity algoritmy neuronových sítí, metody A^* a jí podobné. V roce 1958 byl vynalez jazyk *LISP*. Nicméně se jednalo také o období vystřízlivění z původních předpokladů roku 1956. Jedním z neúspěchů byl například pokus o překlad jazyka s výlučným využitím slovníků. Začalo se ukazovat, že kvalita překladu souvisí do značné míry s kontextem a znalostmi, což zapříčiňovalo expanzi možných stavů k prohledání. Z toho poté plynula nutnost nalézt správné heuristiky prohledávání takových stavových prostorů. I z toho důvodu je druhá etapa, odehrávající se v 70. letech, nazývána obdobím vystřízlivění. V této době se objevují první expertní systémy, například pro identifikaci sloučenin či infekčních onemocnění. Během třetí etapy je již více projektů AI komerčních. Samotná AI se stává uznávaným vědním oborem. Nový zájem a komercializace přináší peníze a další vývoj, v jehož důsledku se objevuje optimalizovaný hardware, určený problémům řešeným metodami umělé

inteligence. Jsou realizovány rozsáhlé projekty a prohlubuje se výzkum neuronových sítí. Poslední etapa probíhá až do současnosti. Je charakterizována kromě dalšího výzkumu v oblasti softcomputingu (práce s neurčitostí a neúplnými informacemi) také hromadným nasazováním produktů umělé inteligence do oblasti praktického využití (Zbořil, 2004).

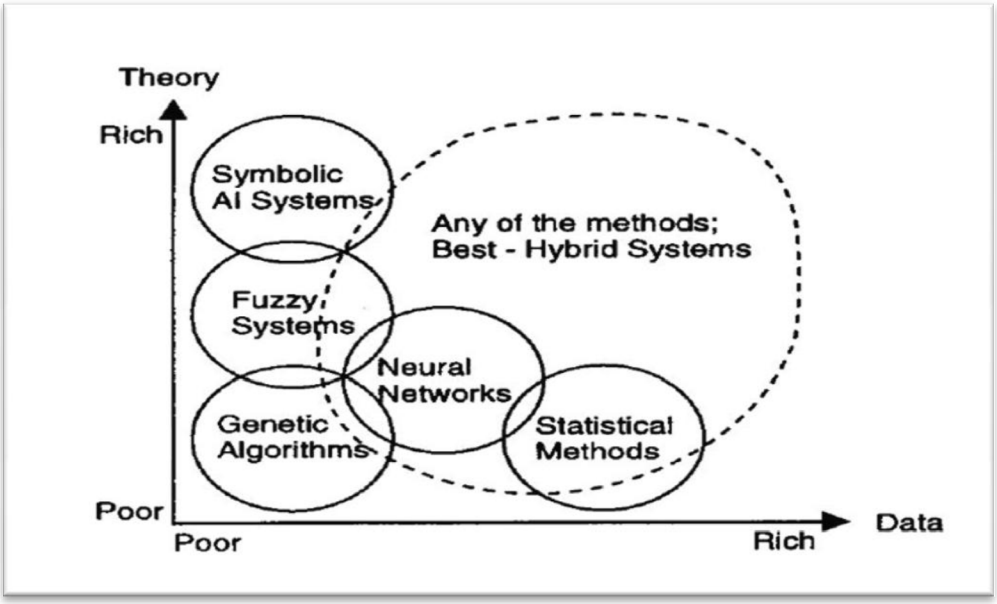
Za příklady můžou posloužit internetové vyhledávače, efektivně zpracovávající a vybírající důležité informace. Mikrovlnná trouba *Sharp*, která pomocí neuronových sítí umí určit dobu vaření. Klimatizace *Videocon* zase pomocí neuro-fuzzy systému nastaví správnou teplotu v místnosti. Podobný systém pracuje i v pračkách, které umí analyzovat tkaninu a podle jejího charakteru nastavit parametry praní. Důležitými odvětvími jsou i robotika a automobilový průmysl, který AI využívá v bezpečnostních prvcích a systémech detekujících osoby u krajnice. Příkladem mohou být i převodovky. *Mercedes* takto instaloval automatickou převodovku, která řadí podle individuálního stylu řidičovy jízdy (Jirsík, 2007). Známý příklad umělé inteligence v kombinaci se superpočítačem je systém *Watson* vyvinutý společností *IBM*. *Watson* v sobě obsahuje 15000 operačních jader a 15TB operační paměti. Je zkonstruován tak, že je schopen odpovědět téměř na jakoukoliv otázku, kdy pomocí své inteligence nalezne v krátkém čase odpovědi a je dokonce sto se poučit ze svých chyb. Je také prvním strojem, který vyhrál kvízovou show proti dalším dvěma lidským účastníkům. Ukázka z průběhu soutěže je na obrázku 1.1 (Ibm.com, 2010).



Obrázek 1.1 Superpočítač *IBM Watson*

Je zřejmé, že za posledních 50 let došlo k enormnímu vývoji. Bohužel se zatím nepodařilo vyvinout systém, který by plně prošel Turingovým testem. Nicméně samotný obor umělá inteligence se od dob svého počátku velmi diferencoval a zahrnuje v sobě spoustu metod řešení různých problémů, které jsou pro danou oblast efektivní (ukázka na obrázku 1.2 (Jirsík, 2007)). Mezi nejznámější problematiky AI patří strojové učení a prohledávání stavového prostoru. To je vhodné pro řešení problémů s množinou možných stavů. Algoritmus pak generuje například vítěznou strategii. Do oblasti AI spadá i dolování dat a práce s expertními systémy.

Poslední dvě metodologie, včetně definice cílů, kterých s nimi chci dosáhnout, podstupuji hlubší analýze v kapitole 3. Jsou to genetické algoritmy (kap. 3.1) a neuronové sítě (kap. 3.2). V kapitole 2 definuji problém obchodního cestujícího, který se stane základním předmětem experimentů s vybranými algoritmy. Získané informace budou sloužit k rozboru chování zvolených postupů, včetně návrhu možných úprav.



Obrázek 1.2 Aplikační oblast AI

2 Problém obchodního cestujícího

2.1 Formalizace zadání

Problém obchodního cestujícího můžeme neformálně popsat takto. Existuje množina měst, kterou chce náš cestující navštívit. Přitom se nakonec vrátí do města, ve kterém celou cestu započal. Kromě města, do kterého se vrátí, uvidí každé město právě jednou a trvá na tom, že při své pouti urazí nejkratší možnou vzdálenost (Algoritmy.net, 2010).

Nyní popišme celý problém rigorózně. Mějme úplný neorientovaný graf

$$G = (V, E) \qquad (1)$$

s množnou všech vrcholů V a množinou všech hran E. Hrany grafu mají ohodnocení d

$$d: E \to \mathbb{N} \qquad (2)$$

Tento graf reprezentuje topologii propojení měst, kdy je každé jedno město přímo spojeno se všemi ostatními. Dále definujme cyklus C

$$C = (V, E') \qquad (3)$$

který prochází všemi vrcholy grafu G a přitom k tomu využije množinu hran E' takovou, že $E' \subseteq E$ bez opakování. Dále platí

$$d(E') = \sum_{e \in E'} d(e) \qquad (4)$$

$$d(C) = d(E') \qquad (5)$$

Problém obchodního cestujícího tedy zní takto: mějme úplný neorientovaný graf $G = (V, E)$ a najděme pro tento graf cyklus $C = (V, E')$ pro nějž platí $d(C) = L$, kde L je délka nejkratší možné trasy. Jedná se tedy o hledání nejkratší Hamiltonovské kružnice v grafu. Úplný graf o více jak 2 vrcholech je Hamiltonovský, jelikož splňuje alespoň jednu z dostačujících podmínek [1] Hamiltnovských grafů (Šalát, 1981) a je souvislý s vrcholy stupně alespoň dva.

Zmiňme, že existují varianty hledání optimálně nejrychlejší cesty, kdy jsou do řešení zahrnuty i další faktory, nejenom vzdálenost, a proto nejlepší řešení nemusí být zároveň to nejkratší (letecká doprava). Travelling salesman problem (dále TSP) je NP-těžká úloha (non-polynomial), což znamená, že náročnost roste exponenciálně s velikostí problému a tudíž nelze popsat polynomiálním rozvojem. Rozhodovací varianta TSP je NP-úplná, tedy existuje nedeterministický Turingův stroj, který maximálně po polynomiálním počtu kroků dá odpověď ano či ne, vztahující se k nejkratší možné trase, typicky (pomocí ternárního výrazu)

$$d(C) \leq L ? ANO : NE$$

[1] Existují 3 dostačující podmínky, aby byl graf Hamiltonovský a bylo možné v něm najít Hamiltonovskou kružnici. Jsou jimi Diracova, Pósova a Oreho.

2.2 Metody řešení

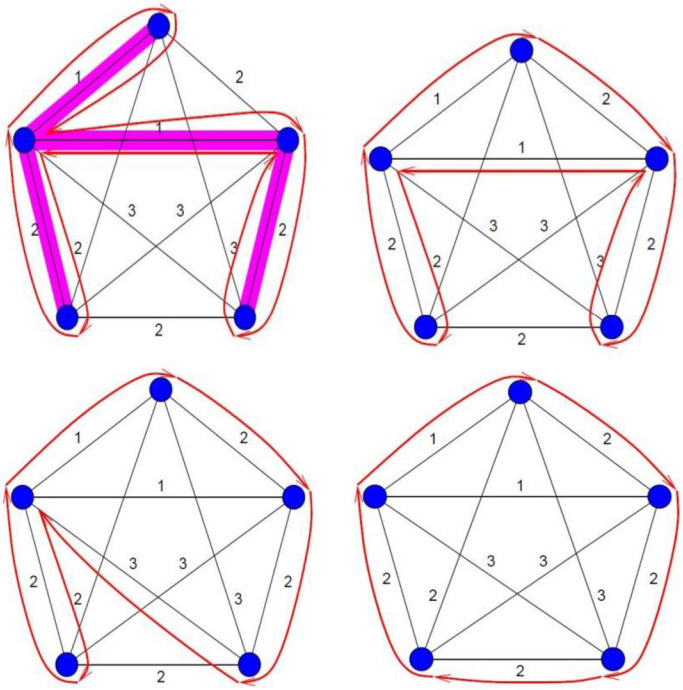
Nejjednodušším řešením TSP je otestovat všechny možné varianty a na základě výsledků určit nejkratší cestu. Jelikož počet variant pro n měst odpovídá $n!$ možnostem, při větším počtu měst tento postup není vhodný. Další zajímavé algoritmy jsou používané v rámci lineárního programování (optimalizace problému pomocí soustav rovnic), tzv. Cutting-plane algoritmus.

Existují také polynomiální řešení grafů, kde je nejkratší vzdálenost mezi body definována trojúhelníkovou nerovností. U těchto řešení lze dokázat, že nalezený cyklus nejhorší varianty můžeme popsat (Christofidesův algoritmus)

$$d(C) \leq 2d(C^*) \tag{6}$$

$$d(C) \leq \frac{3}{2}d(C^*) \tag{7}$$

v závislosti na typu řešení (C^* odpovídá nejkratšímu možnému cyklu). To vychází z úprav minimální kostry grafu (Algoritmus.net, 2011). Postup řešení z oblasti teorie grafu je uveden na obrázku 2.1.



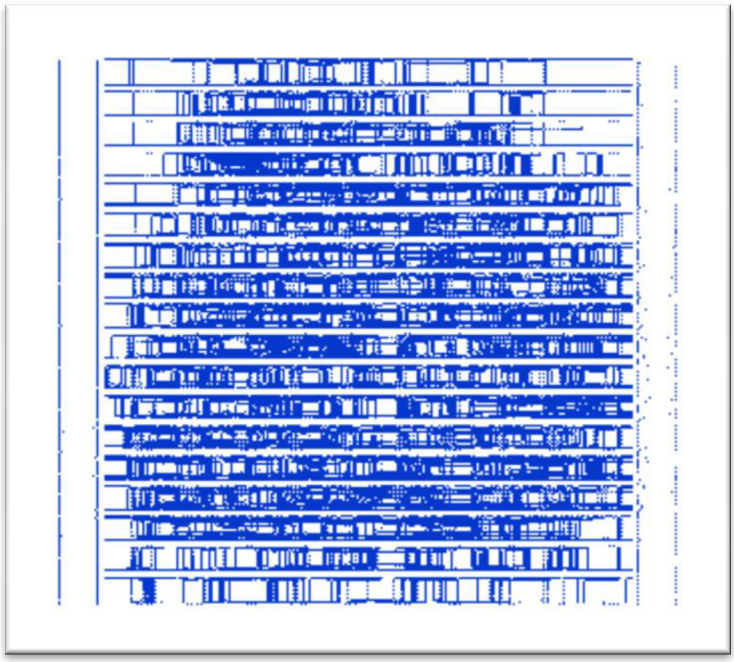
Obrázek 2.1 Úprava minimální kostry grafu Hamiltonovské kružnice

Obvyklá řešení této úlohy nespočívají v nalezení přesné hodnoty správného výsledku, nýbrž v aplikování metod heuristických, které v dosažitelném čase produkují, dle námi zvolených pravidel, dostatečně přesné řešeni, které ovšem nelze dokázat. Mezi takové metody řadíme algoritmy využívající neuronové sítě či genetické algoritmy. Jimi se budeme podrobně zabývat. Další heuristickou metodou je například metoda simulovaného žíhání, která spadá do kategorie metod řešících problém pomocí lokálního prohledávání. Existuje několik jejích variant, u všech je nutné definovat funkci pro změnu teploty. Teplota určuje pravděpodobnost žíhání. Obvykle platí, že pravděpodobnost žíhání roste úměrně vzdálenosti a snižující se teplotě. Vyberme náhodně dvě města. Pak ona vzdálenost u TSP může být například rozdíl součtu

vzdáleností vybraných měst od měst následujících se součtem vzájemných vzdáleností vybraných měst a vzájemných vzdáleností následujících měst. Žíháním můžeme rozumět například algoritmus měnící pořadí měst mezi nejkrajnějšími městy vybranými pro zjištění měnící se vzdálenosti. Takový algoritmus může po dostatečném množství průchodů generovat dostatečně přesná řešení. Obecná podmínka zastavení takového algoritmu odpovídá například porovnáním explicitně zadané minimální odchylky dvou po sobě jdoucích řešení s aktuálním rozdílem po sobě jdoucích řešení, popřípadě neměnností nejlepšího řešení v průběhu n kroků. Podrobně se budu zabývat optimalizací těchto parametrů u mnou vybraných metod, proto je dále nebudu na tomto místě rozvádět. Metodu podobnou myšlence simulovaného žíhání pro řešení TSP lze vidět v přírodě u mravenců, kteří jsou na základě intenzity jimi vylučovaných feromonů schopni nalézt nejkratší dostupnou cestu k potravě a zpět do mraveniště. Pozorováním jejich chování byla Marcem Dorigem také navržena heuristika popsána v práci *Ant Colony Optimization*. Optimalizací heuristik je nesčetné množství, například iterační či stochastické optimalizace pracující s markovskými řetězci.

2.3 Historie řešení TSP

Prvně byl TSP definován počátkem 19. století W.R.Hamiltonem a Thomasem Kirkmanem. Většímu zájmu se tento problém začal těšit až ve 20. století, kdy byl popisován a zkoumán na několika univerzitách. Nárůst zájmu nastal s příchodem počítačů v 50. letech, kdy se otevřely možnosti jak pro broot force metody, tak pro optimalizační a heuristické algoritmy řešení. Roku 1972 dokázal Richard Karp, že se jedná o NP-těžký problém. Další zkoumání posouvalo počet optimálně spojených míst z desítek až k 34000 díky programu *Concorde* v roce 2005 a 85900 v roce 2006. Výsledek spojování několika tisíc bodů je na obrázku 2.2 (Tsp.gatech.edu, 2007).



Obrázek 2.2 85900 bodů spojených na VLSI

Postupy řešení TSP jsou využity v oblasti mikročipů, práce s DNA, logistiky a plánování.

3 Stochastické optimalizační algoritmy

TSP je NP-těžký problém a proto počet možných variant, kterých nabývá stavový prostor, nelze popsat polynomiálně. Počet možností odpovídá permutaci všech měst bez opakování. Pokud bychom pro TSP požadovali přesné řešení, bylo by nutné všechny tyto stavy otestovat, což by reálně nebylo možné z důvodu přílišné časové náročnosti. V takovém případě je vhodné použít například stochastickou metodu se správnou heuristikou. Ta, ačkoliv pracuje nedeterministicky, může s určitou přesností v dosažitelném čase uspokojitelně aproximovat správné řešení. Je nutné zmínit, že rychlost řešení ovlivňuje i množina aplikovaných dat. Jak bylo řečeno, důležitým faktorem u každé stochastické metody je volba heuristiky. Jelikož se AI jako vědní obor úzce dotýká jak informatiky, tak biologie a v některých případech i filosoficky-etických otázek (evoluce, člověk vs. robot), budou heuristiky aplikovány právě z oblasti biologie. Ukazuje se totiž, že sama příroda využívá ve spoustě oblastí efektivní a pozoruhodně jednoduché algoritmy, které je možné aplikovat i v IT. Jedná se pak o jakousi symbiózu biologie a informatiky, kdy algoritmické řešení může potvrdit správné předpoklady biologů a dokonce se může ukázat vskutku efektivním. Do této oblasti spadají i metody genetických algoritmů a neuronových sítí.

Mým cílem proto bylo vybrané metody z obou oblastí otestovat a srovnat jako možnou variantu řešení TSP. Zaměřil jsem se na vhodnou optimalizaci parametrů vzhledem k rychlosti a paměťové náročnosti výpočtů. Za účelem optimalizace a následného srovnávání jsem provedl experimenty, z nichž jsem se pokusil vyvodit závěry o vhodnosti výběru algoritmů, včetně diskuse nad klady a řešením záporů jednotlivých metod.

3.1 Genetické algoritmy

3.1.1 Teoretické východisko genetických algoritmů

Genetické algoritmy (dále GA) a jejich počátek jakožto vědní disciplíny jsou spjaty se jménem John Holland, který v 70 letech minulého století započal jejich zkoumání. Jedná se o typ evolučního algoritmu. Genetické algoritmy patří mezi optimalizační metody numerické matematiky, které se hodí pro hledání extrému mezi extrémy lokálními složitějších matematických problémů, a to použitím Darwinovy teorie a pojmů jako jsou generace, křížení a mutace jedinců atd..

Nejprve tedy uveďme důležité poznatky evoluční teorie, které budeme dále aplikovat. Evolučních teorií je několik. Poslední zcela ucelenou, avšak pro další bádání příliš obecnou teorií je Darwinova. Ta postupem času přešla v neodarwinismus, který reflektoval další objevy v genetice, molekulární biologii a cytologii a syntetizoval je spolu s původní Darwinovou teorií. Té se pro vysvětlení dalších principů přidržíme. Pro zajímavost uveďme, že optimalizační algoritmy nemusí čerpat myšlenky pouze z evoluce biologické. Například fyzikální termodynamická evoluce stavu je aplikována v metodě simulovaného žíhaní a chování některých neuronových sítí reflektuje poznatky o magnetických doménách.

Darwinova teorie vychází z několika faktů. Zmiňuji zde pro mé zkoumání a další výklad jen podstatné:

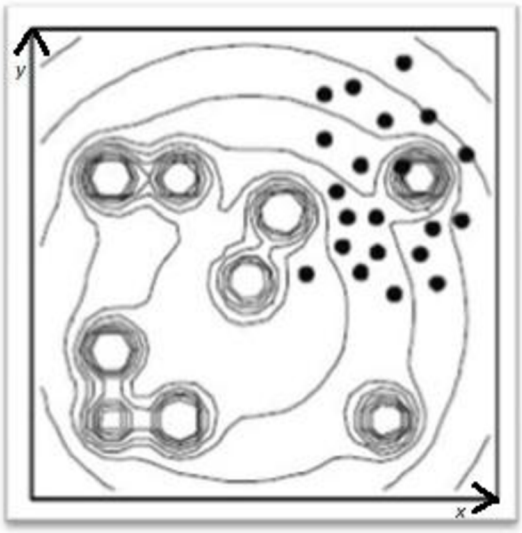
1. **Jedinci nejsou stejní, nejsou neměnní, mohou dědit vlastnosti svých rodičů**
   Tento fakt bude rozebrán v následujících kapitolách. Dotkne se křížení jedinců a způsobu kombinace jejich genetického materiálu.

2. **Přirozený výběr**
   Jedinci s lepší genetickou výbavou, silnější a zdatnější, vstupují do reprodukce častěji než ostatní, což by měl být jeden z faktorů umožňující lepší rozvoj následující generace.

Další body, které se projevují v rámci implementace genetických algoritmů:

3. **Genetický drift**
   Jedná se o náhodné události u jedinců, ovlivňující spíše menší populace. Patří mezi ně například mutace.

Jak již bylo zmíněno, jedinci se rozmnožují, přičemž svým potomkům předávají genetickou informaci. Při využití algoritmů budeme uvažovat jedince jako soubor genů obsažených v DNA chromozomů. Během křížení dochází ke kombinaci stávajících genetických informací a k mutacím, rozumějme změnám v aktuální konfiguraci DNA jedince. Křížení jedinců se odvine od jejich síly v generaci, což bude odpovídat fitness funkci, tedy fenotypu jedince. Fenotyp znamená soubor všech pozorovatelných vlastností organismu. Tvorbu generací a jejich postupný vývoj lze popsat také pomocí fitness funkce (obrázek 3.1)



Obrázek 3.1 Vrstevnicový pohled na trojrozměrnou fitness funkci s populací jedinců

Na obrázku je schematicky znázorněn vrstevnicový model fitness funkce s jejími lokálními maximy. To jsou oblasti s koncentrací kružnic představující místa s největším gradientem. Černé tečky jsou jedinci v populaci. Jejich cílem je dostat se do oblasti s největším gradientem. Pokud dochází k mutacím, jejich posun je nedeterministický, pokud ke křížení, pohybují se směrem největšího růstu.

3.1.2 Základní pojmy genetických algoritmů

3.1.2.1 Jedinec, genotyp, chromozom

Postupně formalizujme problém TSP v rámci GA. Nejprve je nutné definovat genetický materiál, genotyp, se kterým budeme dále pracovat. Genetický materiál bude neměnný a bude tvořen množinou genů. Geny chápejme jak jednotlivá města. $G = \{g_1, g_2, g_3 \ldots . g_n\}; n \in \mathbb{N}$ je množina genů g, jejichž počet n odpovídá počtu aktuálně zpracovávaných měst. Jedinec je kompozitem takového materiálu. Vektor reprezentující genovou sadou bude obsahovat celý genetický materiál G bez opakování. Tato podmínka je spjata s konkrétním zadáním TSP, jak je popsáno v kapitole věnující se formálnímu popisu. Jedince proto bude reprezentovat chromozom, jakožto permutace všech genů genotypu, proto $\alpha = permutace(G); |\alpha| = n$. Kardinalita chromozomu odpovídá celkovému počtu měst. Podotkněme, že obecně bereme chromozom jako binární vektor (usnadňuje to manipulaci v kódu) odpovídající možné variaci genů.

3.1.2.2 Populace

Populace v TSP je množinou m jedinců, tedy $P = \{\alpha_1, \alpha_2 \ldots \alpha_m\}; m \in \mathbb{N}$. Lze ji chápat jako podmnožinu variací binárních vektorů genotypu, tedy

$$P_{obecna} = \{\alpha_1, \alpha_2, \ldots . , \alpha_o\} \subseteq G^n \qquad (8)$$

Generace v TSP je podmnožinou obecného pojmu generace $P \subseteq P_{obecna}$.

3.1.2.3 Fitness funkce

Pro další výpočty je důležitá definice takzvané fitness funkce a její normalizované podoby. V podstatě se jedná o fenotyp jedince, tedy soubor vlastností a znaků spolu s genotypem. Ten ovlivňuje přežití jedince v populaci, jeho schopnost podílet se na reprodukci a šíření kombinace svého genetického materiálu dál. V přeneseném slova smyslu se jedná o evaluační funkci, která definuje pravděpodobnost vybrání jedince pro křížení. Existuje přímá úměra mezi velikostí fitness funkce a výskytem jedince při reprodukci.

Definujme fitness funkci $F, F: \alpha_n \rightarrow R^+$, potom její normalizovaný tvar odpovídá výrazu

$$F_i' = \frac{F_i}{\sum_{j=1}^{m} F_j} \tag{9}$$

kde m je počet jedinců populace. Proto platí

$$F_i' \in < 0; 1 > \tag{10}$$

$$\sum_{j=1}^{m} F_i' = 1 \tag{11}$$

Diskrétní hodnoty normalizované fitness funkce kopírují výsledky funkce hustoty pravděpodobnosti výskytu jedinců v procesu křížení.

Hodnotícím kritériem u TSP je délka trasy, kterou cestující urazil. Čím kratší trasa přes všechna města je, tím více by se měla objevovat tato kombinace v důležitých operacích. Pravděpodobnost výskytu bude proto přímo úměrná velikosti rozdílu délky trasy reprezentované určitým chromozomem od maximální délky z celé populace. Zapsáno vztahem

$$f_i = D_{max} - D_i \tag{12}$$

kde D je délka trasy. Normalizace je provedena dle uvedeného vztahu (9) (Farah Al-Dulaimi, 2008).

3.1.3 Algoritmy výběru jedince pomocí fitness funkce

Normalizovaná fitness funkce (9) se využívá pro výběr jedinců z populace při mutacích a křížení. Využitelná je jak přímá hodnota fitness funkce, tak hodnoty z této funkce vycházející. Implementačně jsem ověřil dva postupy. Bude se jednat o algoritmy rulety a deterministického vzorkování (deterministic sampling).

3.1.3.1 Deterministic sampling

Tento algoritmus přesně deterministicky určí, kolikrát se bude jedinec účastnit procesu křížení. Jedná se tedy o celočíselnou hodnotu S pro kterou platí $S \leq m$, kde m je počet jedinců populace. Hodnotu S můžeme vyjádřit pomoci normalizované fitness funkce takto (funkce $round()$ provede zaokrouhlení)

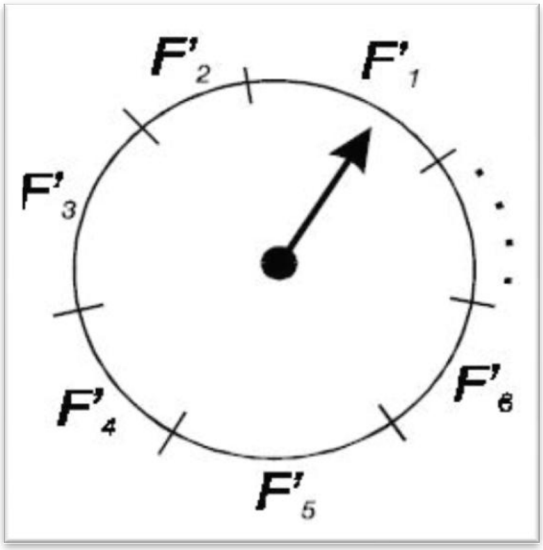
$$S_i = round\left(F_i' m\right) + 1 \tag{13}$$

Algoritmus poté pracuje tak, že množiny jedinců s nenulovou hodnotou S náhodně podstupuje výběru, vybraným hodnotu S snižuje a v případě nulové hodnoty S je vyloučí z množiny vhodné pro další operace. Pro výběr používá

náhodně generovaný index s hodnotou odpovídající rovnoměrnému rozložení pravděpodobnosti pro interval $i \in < 0, modGen - 1 >$, kde $modGen$ je modulo počtu prvků s použitelnou hodnotou S (Farah Al-Dulaimi, 2008).

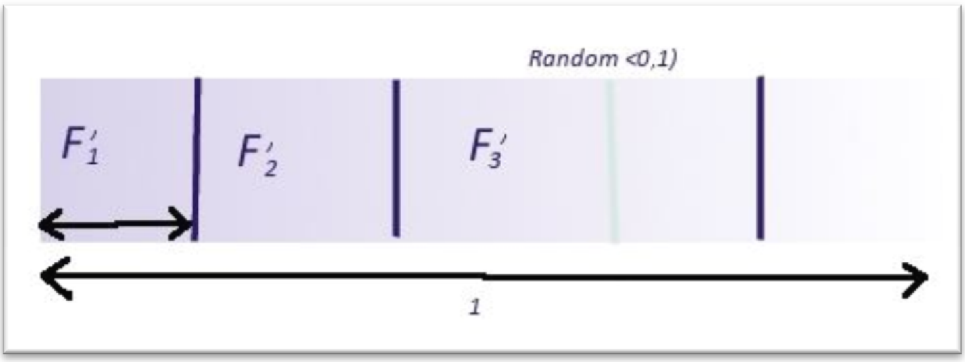
3.1.3.2 Ruletový algoritmus

Ruletový algoritmus funguje jako skutečná ruleta. Představme si kotouč, na kterém jsou vymezeny úseky (obrázek 3.2). Kotouč roztočíme a vhodíme na něj kuličku. Pravděpodobnost, že kulička zůstane v požadovaném výseku, odpovídá velikosti plochy výsekem zabrané. Pokud si problematiku rulety převedeme na problém TSP, tak výseky rulety jednotkové plochy odpovídají hodnotám normalizované fitness funkce každého jedince. Čím je hodnota fitness funkce větší, tím je při rovnoměrném generování náhodného čísla větší pravděpodobnost, že při jejich vzájemném porovnání bude náhodně vygenerované číslo menší, než hodnota fitness funkce aktuálně zvoleného jedince. V takovém případě bude vybrán právě jedinec s porovnávanou fitness funkcí (Hynek, 2007).



Obrázek 3.2 Ruleta ohraničená fitness funkcí jedinců

Z hlediska implementace existuje několik možností, jak problém řešit. Jedním z nejefektivnějších algoritmů, především pak pro velkou množiny dat, je binární strom. Ten s logaritmickou složitostí vybere jedince, jehož hodnota fitness funkce je větší než náhodě vygenerované číslo. V implementaci jsem použil převzatý rekurentní algoritmus a svůj algoritmus náhodného výběru vycházející z myšlenky rulety. Oba algoritmy slovně popíši s pomocí schematických obrázků

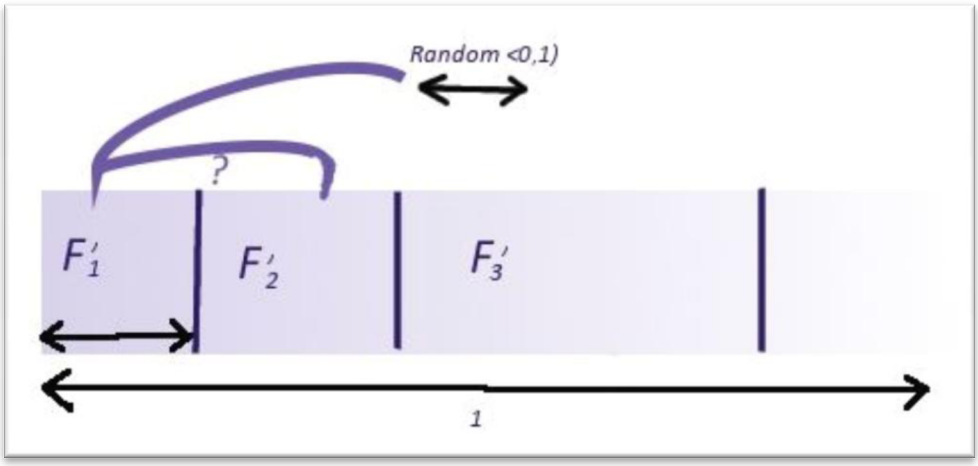


Obrázek 3.3 Schéma rulety pro první příklad pseudokódu

1. Inicializujeme první interval od 0 do velikosti F_1'.
2. Inicializujeme index-ukazatel.
3. Nastavíme predikát nalezení intervalu na false.
4. Dokud není nalezen odpovídající interval, tedy predikát je false nebo dokud jsme neprošli všechny prvky populace.
5. Do index-ukazatele vložíme index na prvek, kterému odpovídá fitness F_n' naposledy přičtená k hornímu ohraničujícímu intervalu.
6. Otestujeme, zda výsledek funkce $Random()$ (generuje náhodná čísla v rozsahu od 0 do 1 včetně) leží v aktuálním intervalu a podle toho nastavíme predikát výskytu.
7. Pokud se nejednalo o poslední prvek populace, do nové spodní hranice přiřadím starou horní hranici testovacího intervalu a do nové horní hranice testovacího intervalu vložím starou horní hranici intervalu, navýšenou o hodnotu normalizované fitness funkce následujícího prvku v populaci.
8. Návrat na bod 4.
9. Po ukončení je v index-registru vybraný prvek.

Celou situaci popisuje i obrázek, ze kterého plyne, že čím větší plochu jedinec zabere, tím má větší šanci být vybrán.

Druhým implementovaný algoritmus pracuje na podobném principu.



Obrázek 3.4 Schéma rulety pro druhý příklad pseudokódu

Generátor vygeneruje náhodnou hodnotu $i \in < 0, \max(F))$, tedy od 0 do hodnoty odpovídající maximální hodnotě normalizované fitness funkce. Poté iterativně prochází jedince populace. Pokud skončí u posledního jedince, začne opět od začátku. Maximálně celý cyklus projde přes tolik prvků, kolik je v generaci jedinců. Optimálně se zastaví u prvního jedince, jehož hodnota relativní normalizované funkce je větší, než námi vygenerované číslo. Z toho plyne, že jedinci s vyšší hodnotou fitness funkce mají při rovnoměrném rozložení generovaných hodnot větší šanci na úspěch. Při dalším generování se začíná od následujícího jedince.

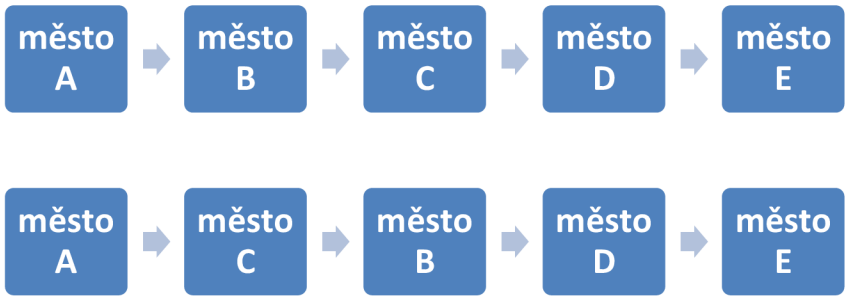
3.1.4 Mutace

Operace mutace je pro vývoj populace velmi důležitá. Jedná se o implementaci dalšího prvku náhodnosti do celkového vývoje. Ten může být v určitých fázích velmi vhodný, jelikož vnese do generace abnormální jedince, kteří akcelerují vývoj. Není ovšem zaručeno zlepšení. Jak již bylo zmiňováno, jedná se sice o posun populace v rámci trojrozměrné fitness funkce, nicméně to nemusí být nutně směrem největšího růstu. Všechny tři uvedené druhy mutací patří k všeobecně známým metodám.

3.1.4.1 SMX-Specialized mutation operator

Mutace pomocí SMX pracuje velmi jednoduše. Náhodně se vyberou dva geny genotypu a ty se mezi sebou v chromozomu prohodí.

Jedinci, který bude mutován, byla náhodně přiřazena dvojice hodnot *(2,3)*. To znamená, že se vymění geny na pozicích 2 a 3.



Obrázek 3.5 Původní a mutovaný jedinec

V navržené aplikaci, použitých tabulkách a grafech je tato mutace reprezentovaná zkratkou SOM (switch ones mutation)

3.1.4.2 Záměna dvou úseků genů v chromozomu

Algoritmus zachovává úseky genů, jen dva z úseků mezi sebou vymění. Velikost a pozice úseku se generuje náhodně. Při překročení absolutní délky chromozomu se uvažují prvky od začátku chromozomu. V navržené aplikaci, použitých tabulkách a grafech je tato mutace reprezentovaná zkratkou SPM (switch parts mutation)

3.1.4.3 Inverze úseku

Algoritmus náhodně invertuje úsek chromozomu. Pokud délka přesahuje pro počáteční bod inverze hranice počtu bodů, pokračuje opět z počátku. V navržené aplikaci, použitých tabulkách a grafech je tato mutace reprezentovaná zkratkou ILM (inverse length mutation)

3.1.4.4 Kombinovaná mutace

Použity byly dva způsoby kombinace mutací. První z nich vykoná několik mutací po sobě na stejném jedinci (dále jej budu nazývat join mutace). U druhého způsobu se náhodně vybere mutace z množiny povolených mutací a ta se aplikuje (dále jej budu nazývat split mutace).

3.1.4.5 Statická mutace

Princip funkce spočívá v tom, že se předem nastaví procento mutací vůči křížení a v daném poměru budou probíhat mutace do dosažení koncové podmínky.

3.1.4.6 Dynamické mutace

Takové mutace nebudou mít konstantní pravděpodobnost výskytu. Frekvence mutace poroste s počtem stavů, které se od sebe odlišují méně, než je dáno maximální hodnotou odchylky pro splnění ukončující podmínky. Jinými slovy, výskyt mutace se bude zvyšovat v případě, že v rámci zadání konečné podmínky bude jedinec neměnný, případně změny budou dostatečně malé. Dynamika se chová spojitě a v mé implementaci sleduje celkem 4 stupně

intenzity. Pokud je provádění algoritmu neohraničeno ukončujícími podmínkami, vždy existuje hodnota počtu iterací řídící dynamiku.

3.1.4.7 Adaptivní mutace

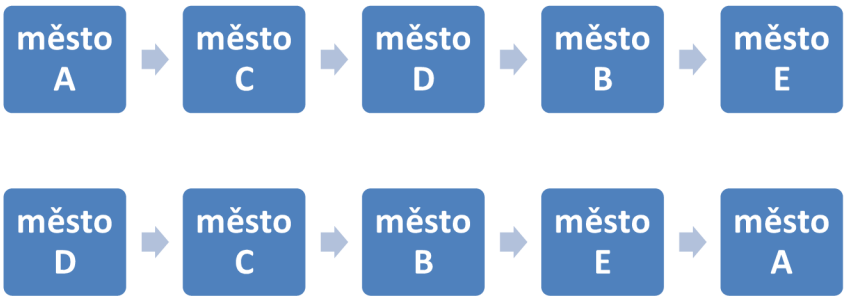
Adaptivní mutace jsou dalším způsobem, jak urychlit řešení v jeho závěrečné fázi. Při jejich aktivaci jsou generovány náhodně jednotlivé různé typy mutací, čímž se může jinak neměně generovaná množina rozrůznit a křížením získat lepší výsledky. Podrobně je toto téma rozebráno v kapitole 3.1.8

3.1.5 Křížení

U jedinců, kteří jsou ohodnoceni fitness funkcí, již známe jejich sílu, resp. možnosti, s jakými jsou schopni zasáhnout to procesu křížení. Samotné křížení představuje tvorbu nové generace, je to analogie k rozmnožování jedinců v přírodě. Po vybrání dvou vhodných kandidátů dojde k jejich křížení, kdy si vymění a zkombinují svůj genetický materiál. V TSP to znamená vhodné zkombinování úseků cest obou jedinců za vzniku jedince nového. Následuje rozbor použitých operátorů křížení (Yu, 2010). U operátorů, které nejsou převzaty (viz. předchozí odkaz) a jsou mým návrhem, je to výslovně uvedeno.

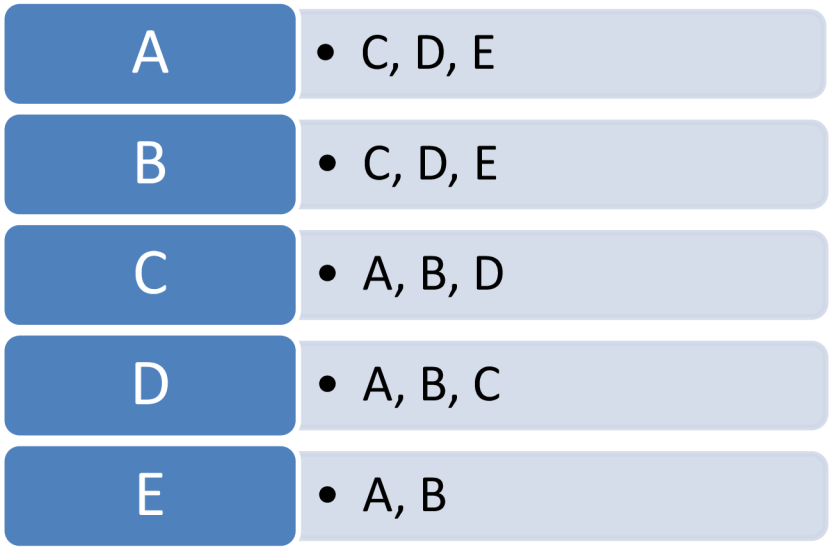
3.1.5.1 ERX – Edge recombination crossover

Mějme dva jedince. Pro oba společně vytvořme tabulku, ve které budou obsažena pro každé město města s ním sousedící, a to od obou jedinců společně. Z takto vytvořené tabulky postupně vybírejme města s nejmenším počtem sousedů. Po každém výběru město odstraníme ze všech zaznamenaných vazeb. V případě shodnosti počtu sousedů vybíráme nedeterministicky. Výsledný jedince bude obsahovat města postupně seřazena podle četnosti spojení s dalšími městy a to od méně k více četným. Demonstrujme na příkladu.



Obrázek 3.6 Rodiče

Výsledná tabulka bude pro tento příklad vypadat následovně



Obrázek 3.7 Tabulka sousednosti ERX

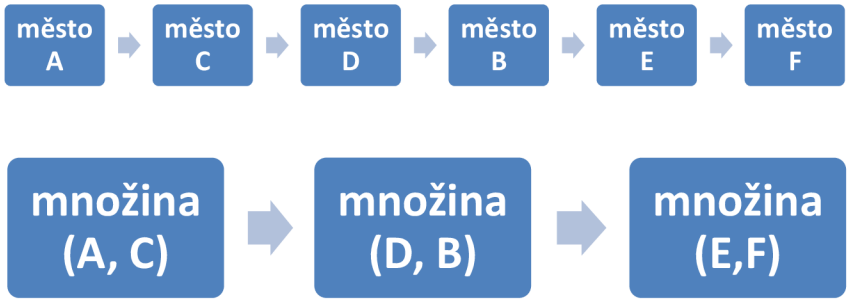
Z tabulky je jasně vidět, že potomek bude začínat městem E. Po skončení křížení bude výsledný potomek vypadat například takto:

Obrázek 3.8 Potomek

Toto je pouze jedna z možností. Vzhledem k náhodnosti vnesené do algoritmu v případě stejného počtu sousedů lze vygenerovat i potomka jiného.

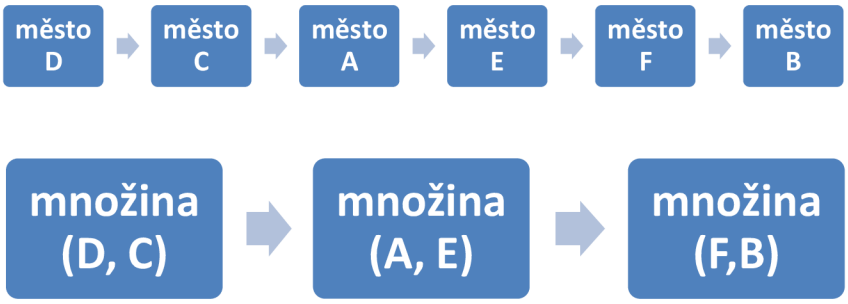
3.1.5.2 OX – Order crossover

Tento typ křížení se snaží zachovávat logické posloupnosti genů rodičů v potomkovi. Nejprve vygenerujeme dvě náhodná čísla, která vytvoří ohraničení chromozomu a rozdělí jej na tři disjunktivní podmnožiny. Toto ohraničení je stejné pro oba rodiče. Uvažujme například ohraničení $O = (2,4)$ při číslování indexů měst od 1. Pak pro příklad níže vzniknou následující podmnožiny.



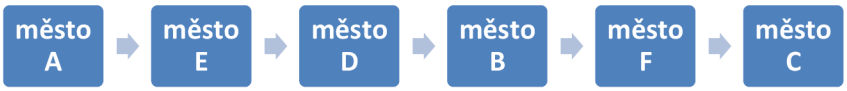
Obrázek 3.9 Rodič 1

Stejnou operaci provedeme i s druhým rodičem



Obrázek 3.10 Rodič 2

Potomka bude tvořit prostřední množina prvního rodiče v zachovaném pořadí přenesená v potomku do stejné pozice. Zbylé geny doplní rodič 2. Způsob doplnění je následující. Začínáme na pozici druhého dělícího bodu v rodiči i potomku. Postupně doplňujeme ještě nevyskytující se geny rodiče do potomka. Pokud dorazíme na pomyslný konec chromozomu (potomka či rodiče), modulárně se vracíme na začátek a procházíme 1. a 2. oddíl chromozomu. Výsledek pro uvedený příklad bude následující.



Obrázek 3.11 Potomek

Je vidět, že na rozdíl od ERX je v tomto algoritmu náhodnost dána právě body dělení. Zbylé provádění je čistě deterministické. U ERX algoritmu se může náhoda při správné konstelaci objevit u přenosu všech genů.

3.1.5.3 OX (optimalizovaná verze)

U optimalizovaného algoritmu OX jsem vyšel z několika předpokladů.

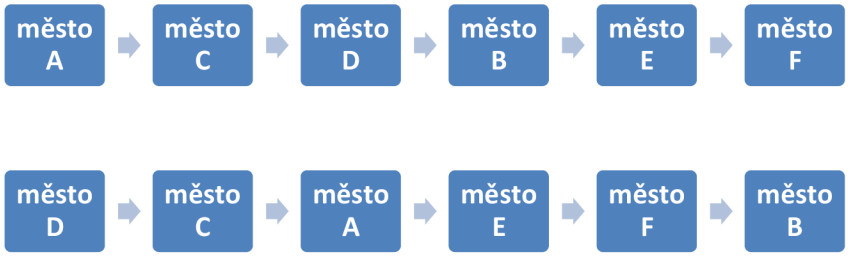
- Rodič s vyšší fitness funkcí předává prostřední neměnnou část posloupnosti genů. Ta je brána jako nositel hlavní informace. Je to ekvivalent toho, kdy se potomek podobá více jednomu z rodičů.
- Posloupnost měst uvažujme jako řetěz. Při jeho zatížení se přeruší s větší pravděpodobností mezi vzdálenějšími body.

Místo, ve kterém dojde k dělení, se vybere vždy ze dvou po sobě jdoucích úseků. Počáteční bod dvojice vybereme nedeterministicky. Cílem je odstranit dlouhé spojnice a křížení.

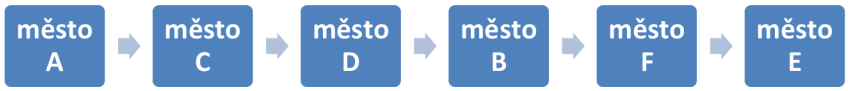
3.1.5.4 PMX-Partially matched crossover

PMX algoritmus je obdobou OX, který ovšem nezachovává pořadí v takové míře a i u jedince, který doplňuje první a třetí úsek, nemusí zachovat logiku pořadí doplňovaných genů. Opět se vybírají dva význačné body, které chromozomy rozdělí na tři podmnožiny. Prostřední množina prvního rodiče se přenese na potomka do stejné pozice v jeho chromozomu. Zbylé geny doplňuje

druhý rodič. V první fázi přidáme do chromozomu potomka na odpovídající místa geny z první a třetí množiny druhého rodiče. Uvažujeme jen ty, které již nejsou v prostřední, tedy druhé množině. V druhé fázi se do chromozomu dodají opět geny druhého rodiče z jeho prostřední množiny. Přidávají se postupně od první do poslední pozice vždy na první volné místo. Příklad pro $O = (2,4)$



Obrázek 3.12 Rodiče



Obrázek 3.13 Potomek

3.1.5.5 PCH – Pair challange

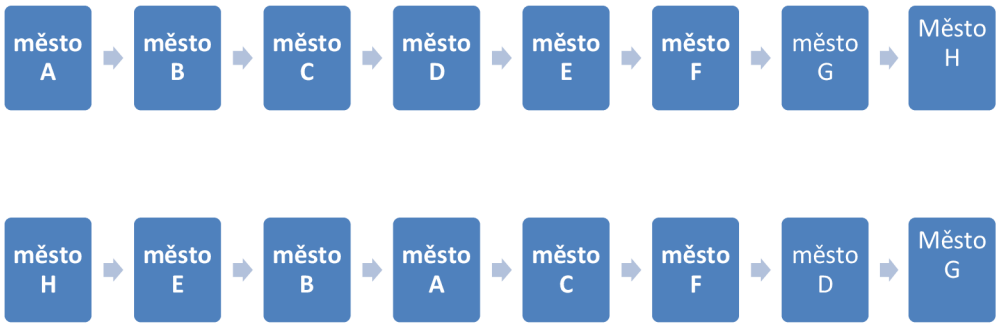
Tato metoda byla mnou navržena. PCH neboli porovnávání párů, jak již název napovídá, bude srovnávat geny na základě zvolené heuristiky. Průběh algoritmu je následovný. Nejprve inicializujeme výsledný chromozom náhodně vybraným počátečním genem.

1. Postupujeme gen po genu obou rodičů.
2. Vždy spolu porovnáváme geny na stejném indexu.
3. Kritériem srovnání je vzdálenost genu rodiče od posledního genu vloženého do chromozomu potomka.
4. Ten gen, resp. to město, jehož vzdálenost je menší a ještě není v potomkově chromozomu, se vloží do výsledku jako první.
5. Následně se vloží druhé město z porovnávaného páru.
6. Vždy se ovšem vkládá tak, aby nevznikly duplicity.

Metoda vytváří na základě klíče vzdálenosti uspořádané trojice.

3.1.5.6 CX – Cycle crossover

U CX nejprve vybereme bod počátku u prvního rodiče. Tímto bodem nahradíme bod na stejném indexu u druhého rodiče. Nahrazený bod rodiče 2 přemístíme na místo, kde se nachází tento bod v prvním rodiči. S nahrazeným bodem v rodiči 2 opakujeme proceduru do chvíle, než se dostaneme k bodu, který jsme nahrazovali jako první. V případě, že nejsou nahrazeny všechny geny druhého rodiče, nenahrazené geny se ponechají ve stejné konstelaci.



Obrázek 3.14 Rodiče

Obrázek 3.15 Potomek

Počáteční bodem záměny byl náhodně zvolen první gen. Následoval řetězec AHGD, který se vložil do chromozomu rodiče 2. Zbylé geny mu zůstaly zachovány v původním pořadí. U této metody může docházet k častým úplným kopiím.

3.1.5.7 OLC – Optimal length crossover

OCL, které je mým vlastním návrhem, se snaží najít nejkratšího možného potomka dvou rodičů. Generuje ho na základě náhodně zvoleného počátečního genu, který se stane prvním genem tohoto potomka. Jelikož chromozom uvažujeme jako posloupnost genů, má náhodně zvolený gen u obou rodičů v rámci řetězce své pořadí $P = \{poradi_{r1}, poradi_{r2}\}$. Počínaje tímto genem je vždy následující takový, že jeho spojnice s předchozím genem je ze všech až 4 (neuvažujeme již zařazené geny) možných spojnic nejkratší. Následníka tedy volíme z množiny indexů ukazujících do chromozomu rodiče $r1$ a rodiče $r2$ $I = \{poradi_{r1} - 1, poradi_{r2} - 1, porad_{r1} + 1, poradi_{r2} + 1\}$. Pro případ indexu ukazujícího mimo množinu genů použijeme modulární přístup. Takto postupujeme pro každý následující gen, dokud lze vygenerovat množina I. Nevýhodou může být fakt, že dva rodiče budou mít s velkou pravděpodobností velmi podobné potomky. Jedná se spolu s ERX také v porovnání s ostatními algoritmy o časově náročnější metodu.

3.1.5.8 Kombinované křížení

Se stejnou mírou pravděpodobnosti vybíráme různé druhy křížení. Tím se mohou odstranit vzájemné nedostatky jednotlivých metod.

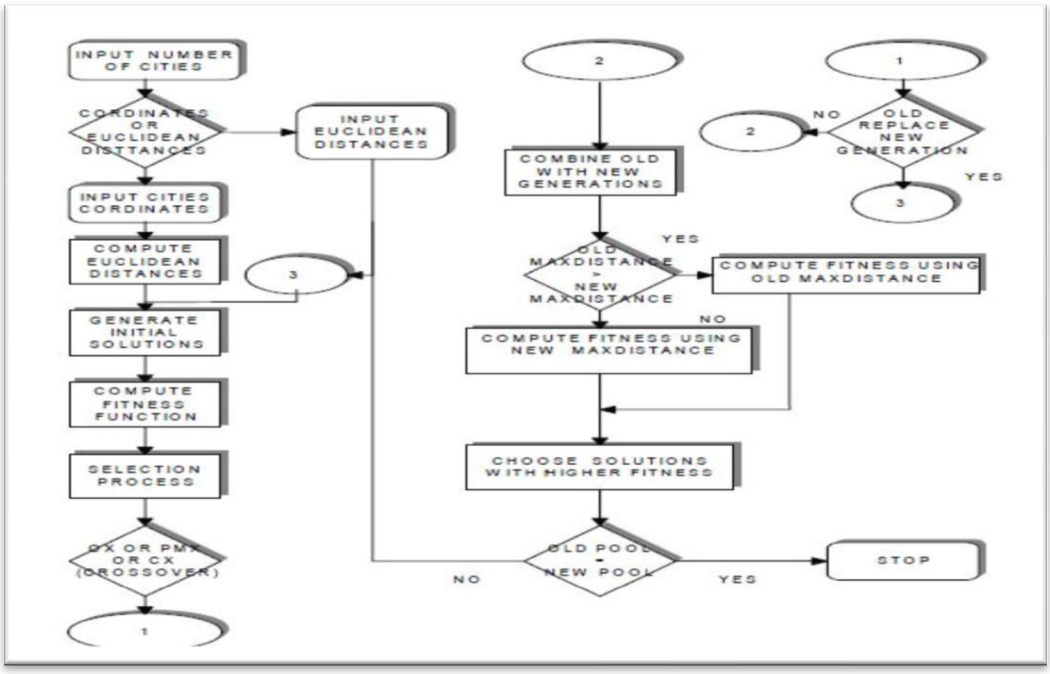
3.1.6 Podmínky ukončení algoritmu

Jelikož optimalizační algoritmus není ohraničen explicitně místem, kdy má skončit, využívají se tzv. ukončující podmínky. Klasickou podmínkou ukončení může být u TSP například délka trasy, pod kterou se dostane výsledek aktuálně nejlepšího jedince. Já jsem zvolil jiný přístup. Jelikož se jedná o iterační algoritmus, uvažujme nejlepší hodnotu jedince v každé iteraci. Algoritmus končí, jestliže se po n iteracích průběžné nejlepší řešení nezlepší o více než $i\%$.

3.1.7 Aplikace teoretického základu do algoritmické podoby

Popíši obecně použitý algoritmus, jehož detaily obsáhne aplikační specifikace.
Schematicky je znázorněn na obrázku 3.16 (Farah-al-Dulaimi, 2008).

1. Vytvoř počáteční populaci.
2. Nastav pravděpodobnost mutace ke křížení.
3. Prováděj, dokud není dosažena ukončující podmínka.
    - Spočítej eukleidovské vzdálenosti a fitness funkce u jednotlivých jedinců.
    - Vlož do nové generace nejsilnější jedince.
    - Dále proveď křížení s mutacemi jedinců a vygeneruj novou generaci.
    - Zjisti výsledky a případně uprav poměr mutací a křížení.
    - Ověř ukončující podmínku.



Obrázek 3.16 Schematická reprezentace genetického algoritmu

3.1.8 Experimenty s aplikací

Veškeré následující experimenty byly provedeny s procesorem *Intel(R) Core(TM)2 Duo CPU T9300 2.5GHz* na platformě *Windows (32bit)*. Vektorovou formu grafů naleznete v příloze C. Křivky v grafech jsou synchronizovány stejným časovým zdrojem. Ten určoval zaznamenání naměřené hodnoty pro aktuální generaci.

3.1.8.1 Experimenty s křížením

Prozkoumejme chování všech metod křížení bez použití mutace a formulujme cíle, na které bychom se chtěli v experimentech zaměřit.

- Jak je řešení závislé na počtu jedinců v populaci?
- Jak ovlivní řešení počet přeživších jedinců do další generace?
- Je řešení nějak závislé na počtu měst určených k propojení?

Jako referenční nastavení pro zkoumání chování křížení si zvolím 50 náhodně vygenerovaných měst. Z generace přežijí právě 3 jedinci. Velikost populace budu měnit a to na 4 různé hodnoty (30, 60, 240, 500 jedinců). Parametry nastavení křížení uvádím ve zkráceném formátu[2].

ERX

ERX pracuje tak, že udržuje konzistenci často se vyskytujících skupin měst, resp. genů, v chromozomu. Chápejme to tak, že pokud se v jedincích často

[2] Formát je následovný: [název metody křížní] [počet jedinců v generaci] [počet přeživších jedinců]. První a poslední parametr je volitelný. Příklad: ERX 240 10 znamená křížení ERX s populací 240 jedinců, z toho 10 přeživších do další generace.

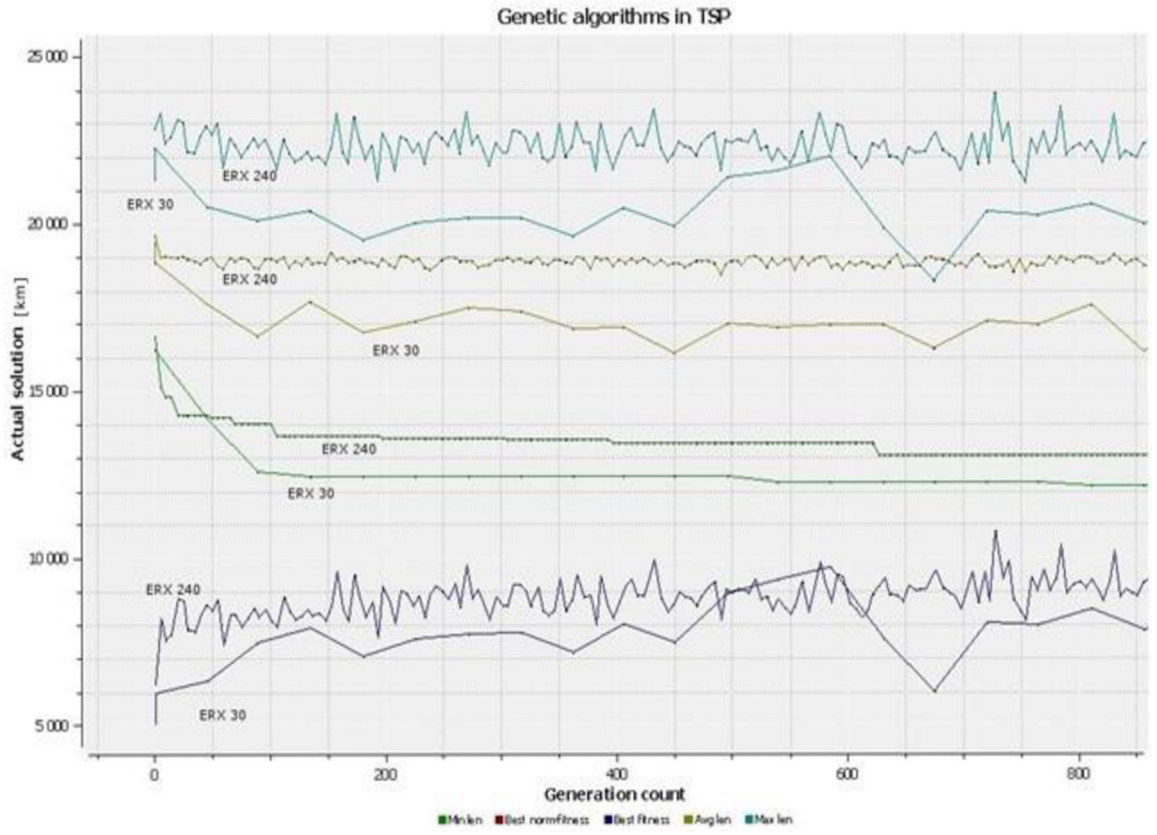
objevují stejné posloupnosti genů, ERX umožní nedeterministicky měnit jejich pořadí v silné skupině, ale obsah množiny se snaží zachovat.

ERX křížení dosahovalo přibližně stejných výsledků (se zanedbatelnými odchylkami) délky nejlepších okruhů po stejném počtu generací. Objevil se ovšem zajímavý jev, který byl potvrzen ve větší či menší míře u všech křížení, kromě těch se silnou heuristikou (viz. popis OLC). Pokud porovnáme populaci s malým počtem jedinců z intervalu $I \in \langle 20; 50 \rangle$, větší populace dosahuje dobrých výsledků rychleji. Křivky nejkratších cest jsou u porovnávaných testovacích množin jedinců velmi podobné. Pokud se populace znatelně zvětší (240 jedinců), akcelerace zlepšení nejkratší trasy bude sice oproti menším množinám opět lepší, ale dojde také k rychlejšímu přechodu do stacionární fáze výpočtu, což je chvíle, kdy se cyklicky začínají generovat podobná řešení a již v populaci nedochází k takové progresi. V trojrozměrném grafu fitness funkce to přirovnejme k uváznutí v lokálním minimu.

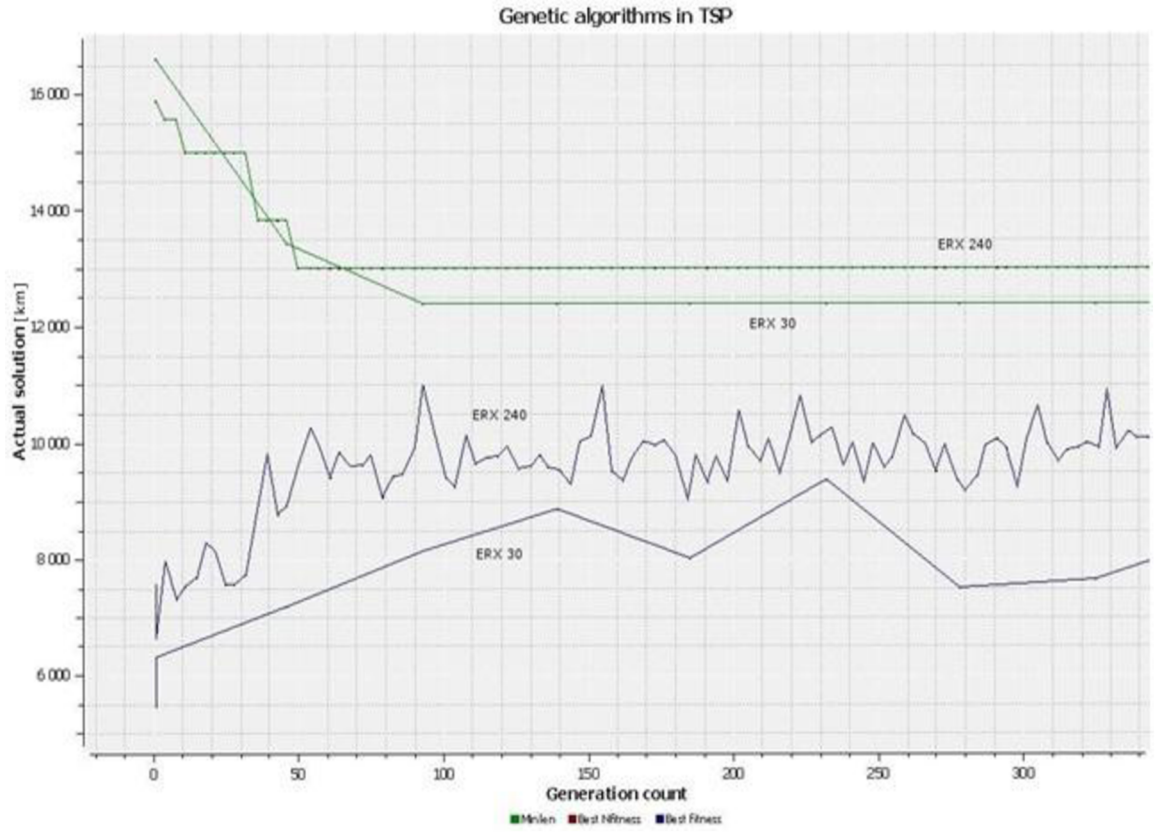
Důvod tohoto chování si vysvětluji následovně. Při inicializaci populace se nepoužila žádná speciální metoda. První generace jedinců byla vygenerována zcela náhodně, a proto se v ní objevovali především průměrní jedinci. Jelikož jsme mohli vybírat z většího počtu vzorků, počáteční nejlepší řešení je s větší pravděpodobností lepší než nejlepší řešení dané menší populací. V dalším průběhu dochází ke křížení. Následující situaci popisuje obrázek 3.17. V malé populaci jsou rozdíly délek okruhů mezi silnějšími a slabšími menší (modrá křivka), a celková populace je silnější (porovnání dvou žlutých křivek průměrné hodnoty délky). Nejrozdílnější fází vývoje jsou iterace do přibližně 100. generace. Je vidět, že průměrná délka velmi výrazně reflektuje zlepšení nejsilnějších jedinců (zelená křivka), což znamená, že v menší populaci se rychleji šíří vlastnosti kvalitního genového materiálu. Všimněme si také křivky délek okruhu nejslabších jedinců. U menší populace dochází k větším odchylkám. To lze přisoudit schopnosti ERX kvalitně rozrůzňovat populaci. Je také vidět, že i menší populace je schopna produkovat kvalitní řešení a to rychleji. Každá metoda však má svá minima, což ukáži u algoritmu OX.

U větší populace nenastává viditelná změna průměrné délky, která by indikovala dynamické zlepšování celé generace. Naopak zůstává přibližně ve stejné hodnotě. I změny u křivky nejlepší (minimální) vzdálenosti jsou více skokové, což spolu s neměnou průměrnou hodnotu ukazuje na populaci s několika silnými jedinci, kteří jen pomalu prosazují svou kvalitní kombinaci genů do celku. Ve velké populaci proto dochází k částečnému potlačení vlivu elitních jedinců.

Rozdíl hodnot se postupně maže při použití cílených mutací (takových, které cíleně zlepšují určité vlastnosti) případně inicializací první velké generace nikoliv náhodnými, ale předem deterministicky vytvořenými jedinci, kteří budou představovat hned na počátku lepší řešení.



Obrázek 3.17 Průběh křížní ERX s 30 a 240 jedinci



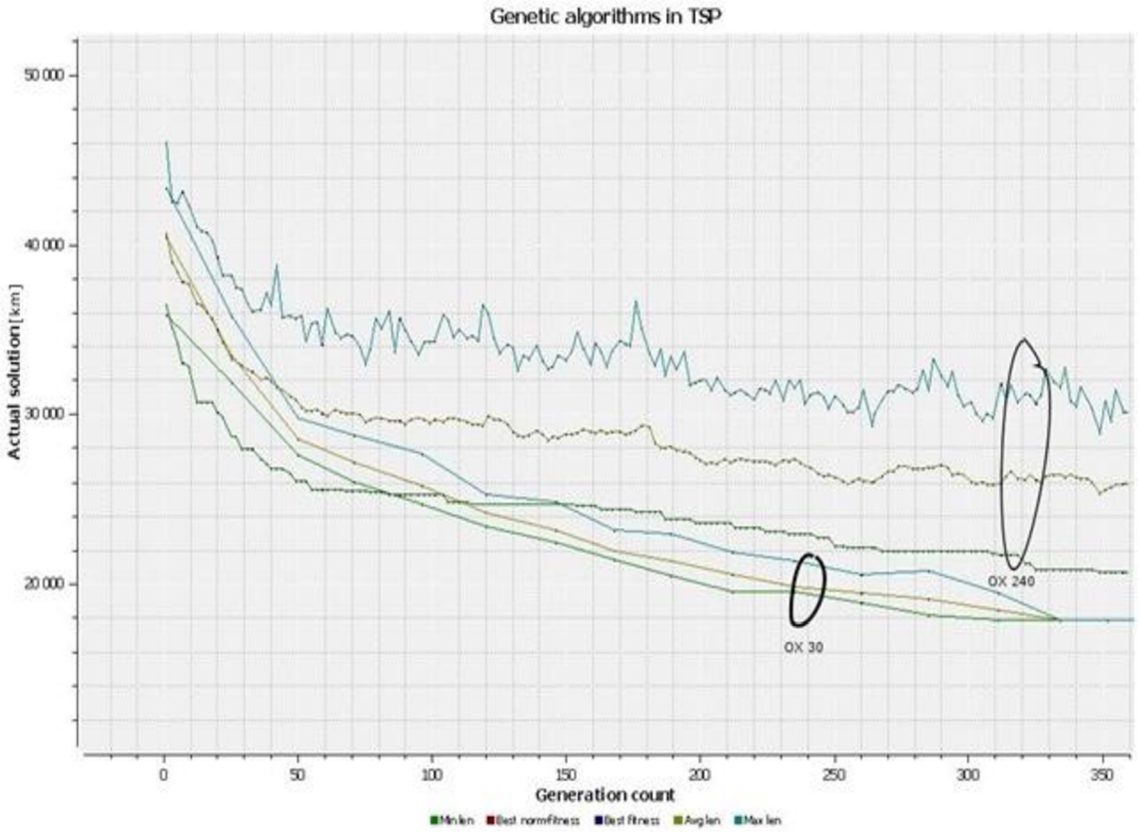
Obrázek 3.18 průběh křížení ERX 30 a 240 jedinci

Diference minimální a maximální délky u algoritmu ERX nasvědčuje ještě jedné skutečnosti. I po ustálení se generují sice podobná, ale různorodá řešení. Tento stav není samozřejmý a u metody OX bude rozebrána situace, kdy a proč takový stav nenastane.

Anomálie zpomalování rychlosti řešení je také řízena počtem předávaných nejsilnějších jedinců. Každá metoda křížení reaguje jinak pro různé hodnoty přeživších. Experimentálně jsem ověřil, že nejsou vhodná žádná extrémní řešení. V případě mnou testované aplikace s populacemi do 500 jedinců jsem empiricky zvolil střední nejefektivnější hodnotu na 10 přeživších jedinců. Pokud je toto číslo větší, neúměrně se omezí vliv například slabších mutantů, v opačném případě se zpomalí prosazení nejsilnějších kombinací genotypu.

OX

Nejprve uveďme graf dvou výpočtů s OX ukazující průběh minimální, průměrné a maximální délky řešení z populace 30 a 240 jedinců.



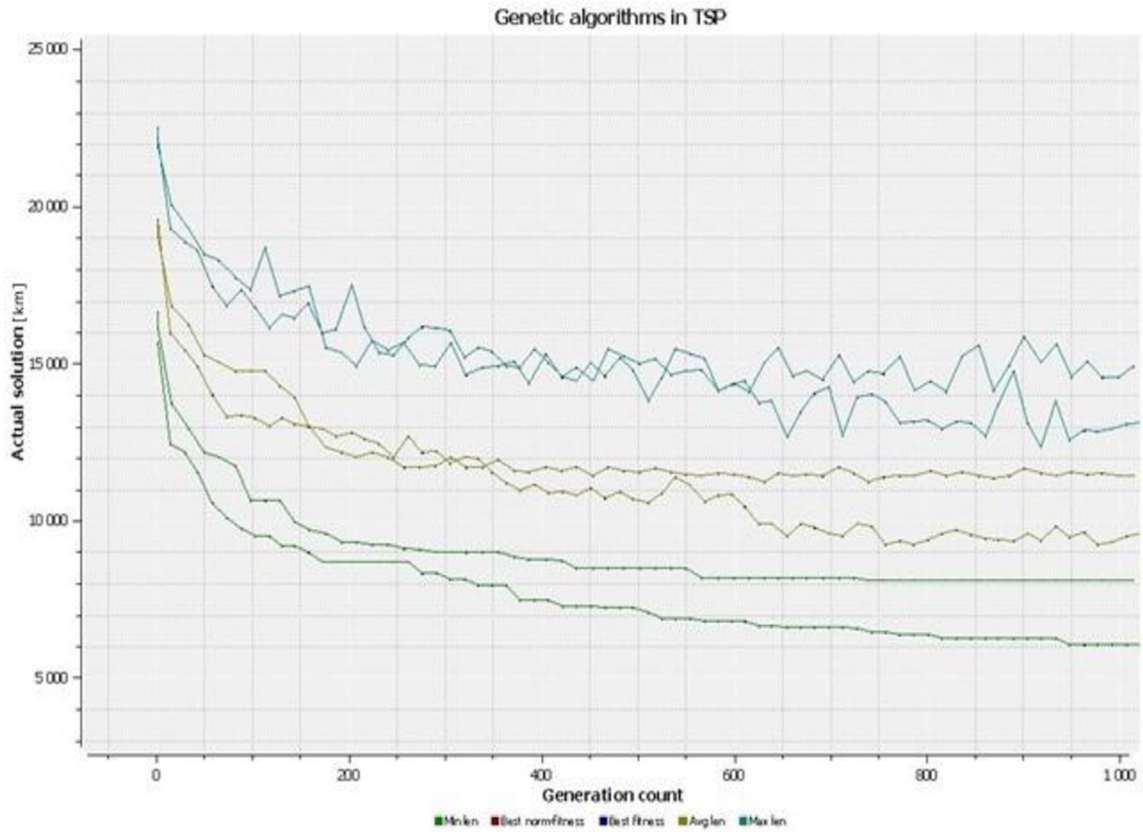
Obrázek 3.19 Průběh křížení OX s 30 a 240 jedinci

Metoda křížení OX 240 3 generuje popisované zpomalení šíření nejsilnějších kombinací genotypu. V tomto případě dochází k rovnoměrnému, ale nepříliš extenzivnímu, distribuování genů kvalitních jedinců (proto je zelená trajektorie nejkratšího jedince u OX 240 po 300 generacích výše než u OX 30).

Algoritmus OX 30 (stejně jako je to vidět na ukázce PMX) také reprezentuje úskalí použití příliš malých populací a náhodného ruletového výběru založeného na popsané fitness funkci. Ten v ruletě nedává reálnou

možnost nejslabším jedincům účastnit se na reprodukci. Je to z důvodu vyjádření fitness funkce jako podílu stávajícího a nejhoršího dosaženého řešení. V bodě spojení tří křivek v grafu došlo díky uniformitě jedinců ke spojení velikosti minimální a maximální délky řešení, tedy v populaci je pouze jediný originální výsledek. K podobným řešením jsem se dostal bez použití mutací s nevhodným počtem přeživších jedinců algoritmy PMX, CX, OX i OLC. OLC bylo ale schopno generovat i z uniformního jednoho jedince rozdílné potomky.

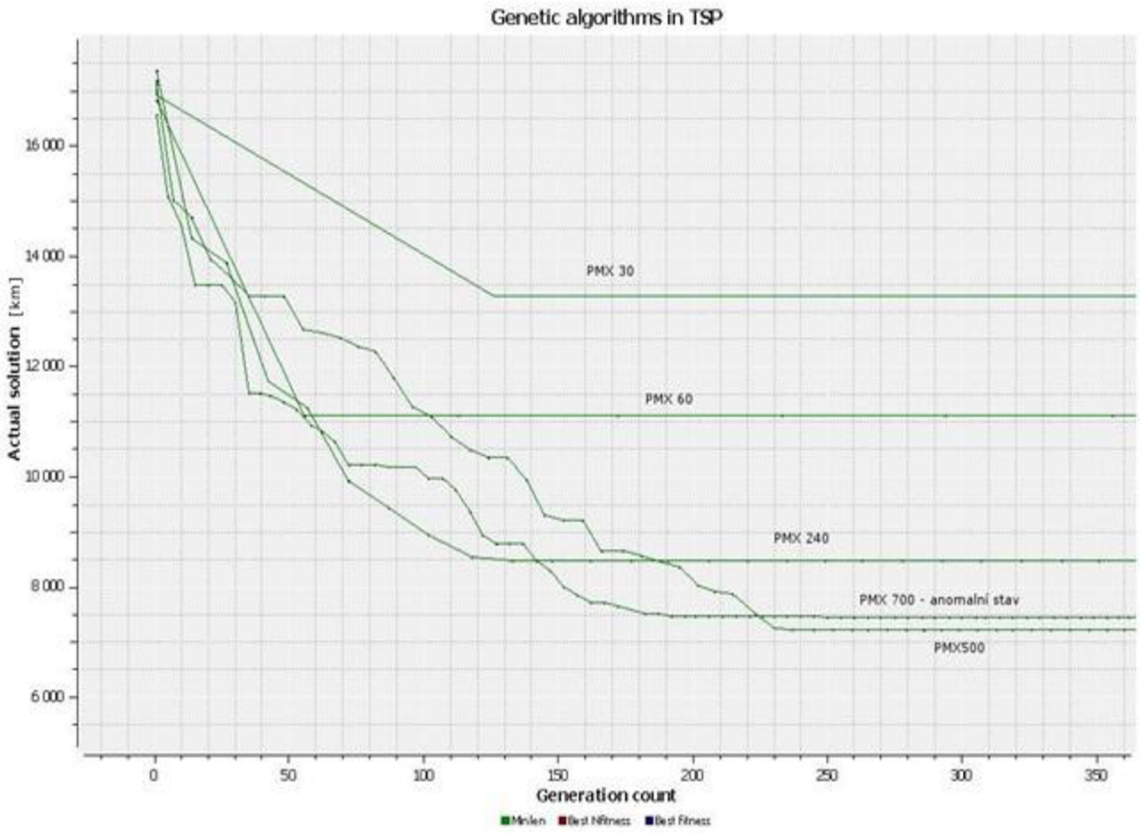
Kdybychom změnili počet přeživších z 3 na 40, žlutá křivka průměrné hodnoty by se posunula blíže maximální hodnotě a celé řešení by bylo v rámci svých nejlepších řešení ještě horší, protože takto velký počet neměnných jedinců opět brzdí distribuci lepšího výsledného řešení (viz. graf na obrázku 3.20).



Obrázek 3.20 Křížení OX 240 s 3 a 40 přeživšími jedinci

PMX, CX

Tyto druhy křížení tvoří speciální množiny, u kterých se až po hranici zpomalení distribuce vlivem počtu jedinců objevuje pro rostoucí počet jedinců v populaci vždy lepší konečný výsledek. Po bližším zkoumání jsem zjistil, že obě křížení mají společnou vlastnost. Potomek zachovává maximum genů rodičů na naprosto stejných pozicích, nedochází ani ve větší míře k promíchání některým z náhodných přístupů. Proto zlepšení odráží možnost potomka čerpat z větší základny variant párů rodičů, které docílíme jedině vetší populací, potažmo mutacemi.



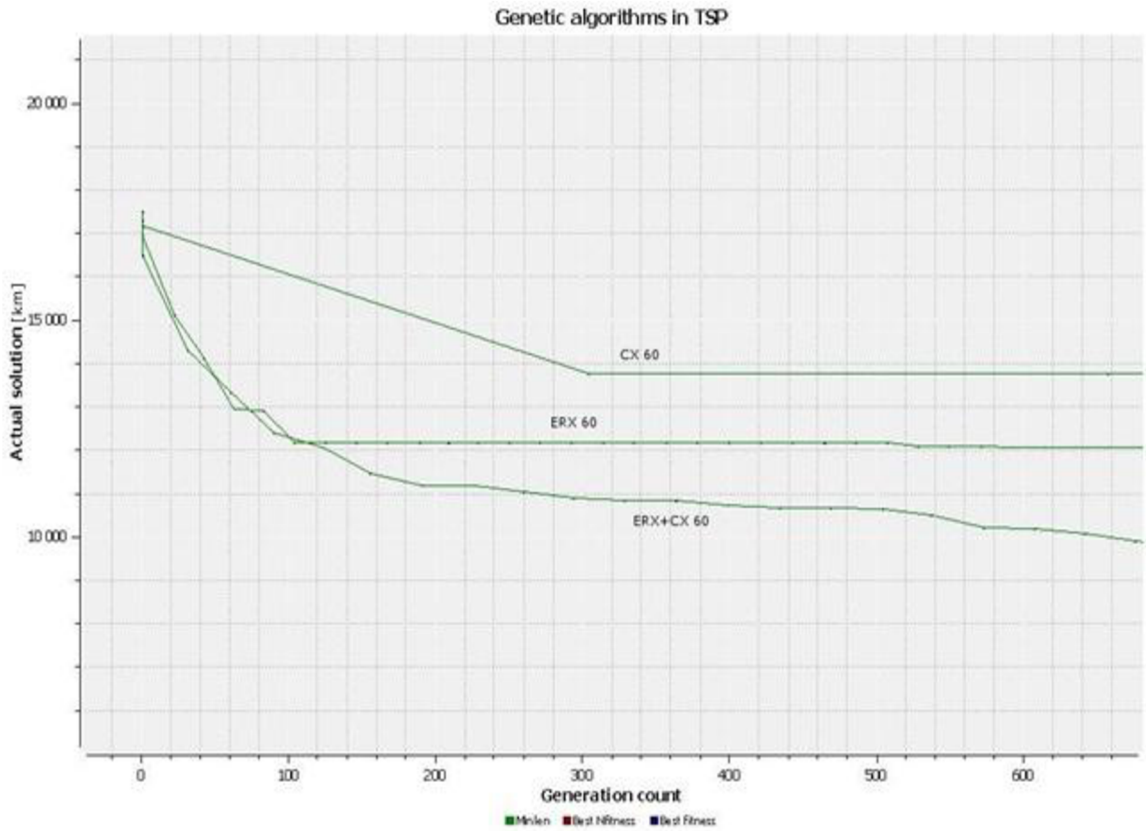
Obrázek 3.21 Průběh křížení PMX

OLC

Heuristické metody s menší mírou nedeterminismu jsou díky systematičtějšímu přístupu ke generování jedinců nepříliš závislé na počtu jedinců a dosahují velmi rychle dobrého, nicméně dále již neměnného, řešení s malými rozdíly mezi nejlepšími a nejhoršími. Dalším problémem je časová náročnost metod, které jim neumožňují řešit Hamiltonovské kružnice o velkém počtu bodů.

Kombinované křížení

Zastoupení zvolených křížení je rovnoměrné. Generují průměr řešení zvolených křížení, přičemž se výsledná křivka přibližuje hodnotě výsledku nejlepší ze zvolených metod. Pokud je v kombinaci metoda se silnou heuristikou (např. řadící geny striktně podle vzájemné vzdálenosti), výsledná křivka ji vždy téměř aproximuje. Častým jevem je, že kombinované řešení dosáhne výsledku nejlepšího, protože dvě křížení na sebe působí podobným efektem jako mutace a celkovou populaci si navzájem divergují, resp. jedna metoda křížení může vnést do množiny jedinců pro druhou metodu nestandardní výsledky a naopak.



Obrázek 3.22 Porovnání průběhu dílčích a kombinované metody

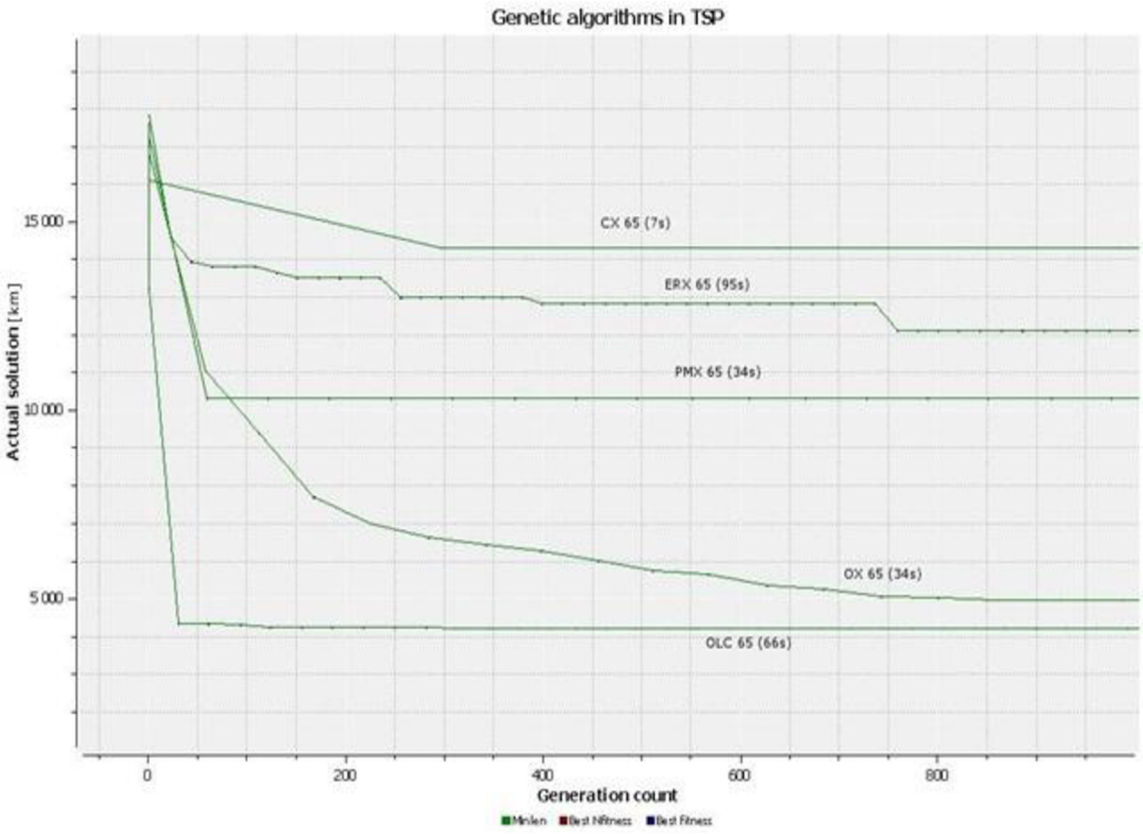
Pro další testování je nutné určit nejvhodnější velikost populace a počet jedinců, které zachováme. V rámci populací od 50 do 500 jedinců jsem dospěl experimentálně k hodnotě 10 nejsilnějších jedinců. Vliv přeživších jedinců na celkové řešení byl demonstrován u křížení OX a je obecně platný.

Za optimální velikost populace byla zvolena experimentálně hodnota 65 jedinců s 10, kteří přežijí do další generace. S tímto nastavením jsem testoval generativní schopnost metod křížení na bázi 50 měst. Výsledky pro všechny zkoumané metody tvořily křivky podobající se exponenciále. Bez mutace se s velkou mírou pravděpodobnosti každé (kromě těch se silnou heuristikou) křížení dostává do stavu ustálení ještě před nalezením optimální délky. Křivky lze přibližně popsat tímto vztahem

$$f(x) = Z^{x-n} + y \qquad (14)$$

Z ... *základ leží v intervalu* $Z \in (0;1)$
x ... *pořadí aktuální generace*
n ... *parametr ovlivňující míru poklesu křivky, hodnotu aktuálního směrového vektoru*
y ... *parametr specifikace konkrétního řešení*

Ilustruje to následujíc graf 3.23 se zaznamenanými časy trvání výpočtu.



Obrázek 3.23 Porovnání průběhu všech metod křížení

Nejmarkantnější pokles křivky nejkratšího okruhu je u metody OLC, která je založená na silné heuristice s malým počtem nedeterminismů. I s vlivem mutací bude generovat podobná řešení. Její schopnost rozrůznit populaci není velká, jelikož nalézá po malém počtu generací podobná kvalitní řešení. Proto bude OLC vhodná do kombinované metody, kde poslouží jako akcelerátor řešení a bude tvořit prvotní kvalitní generace.

Ačkoliv jsou metody PMX a OX na první pohled velmi podobné, cyklické doplňování měst do prvního a třetího úseku chromozomu potomka dává ve výsledcích lepší hodnoty, nežli metoda udržující původní pozice genů obou rodičů. Toto je typická ukázka a důkaz toho, že větší náhodnost se zachováním jen některých původních části chromozomu je velmi efektivní.

V této části testování jsem se zaměřil na optimalizaci počtu jedinců pro další experimenty. Dospěl jsem k hodnotě 65 10. Při výpočtech s menší populací se řešení dosahuje vlivem implementace rychleji, nicméně se ustaluje téměř v konstantní hodnotě a může dojít i k uniformizaci jedinců. Pro populaci 65 jedinců dostáváme kvalitní výsledky v dosažitelném čase s tím, že i po ustálení dochází k velmi pozvolnému zlepšování. U populací větších než 65 jedinců trvá výpočet u některých metod příliš dlouho, v podstatě se stejnými výsledky. Pro efektivní nasazení by bylo potřeba použít mechanizmů urychlující distribuci změn v celé populaci, jelikož u početných populací (ERX,OX 160,PMX,CX 700) může docházet k výraznému zpomalení aproximace nejlepšího výsledku. Přidáním mutací se vliv velikosti do jisté míry vytrácí. Při srovnávacích testech se nejlépe chovala metoda OX. Dokázala nejdéle generovat výrazně se zlepšující

řešení. Díky tomu může optimálně podporovat mutace, jelikož i v rámci malých změn je schopna generovat nová, lepší řešení ovlivňující celou populaci. Dobré vlastnosti distribuce potvrdil i fakt, že v populaci 240 jedinců byla průměrná hodnota délky okruhu přesně mezi limitními hodnotami maxima a minima, což svědčí o dobré distribuci požadovaných genů. Chování můžeme ovlivnit i nastavením přeživších jedinců, kteří svým vlivem mohou posunovat křivku průměru v grafu blíže minimální nebo maximální hodnotě. Počet měst chování algoritmů křížení neovlivňoval.

3.1.8.2 Experimenty s mutacemi

Nastolme otázky, jež budeme diskutovat.

- Mají rozdílné mutace různý vliv na jednotlivé metody křížení?
- Které mutace dosahují nejlepších výsledků?
- Jaké procentuelní nastavení výskytu mutací zvolit pro optimální řešení?

Po provedení experimentů s jednotlivými druhy křížení jsem zjistil, že vybrané metody křížení nijak neovlivňují vliv mutací při jejich vzájemném porovnávání. Proto jsem se rozhodl pro ilustraci uvést křížení ERX, které považuji za dostatečně reprezentativní vzhledem k ostatním postupům.

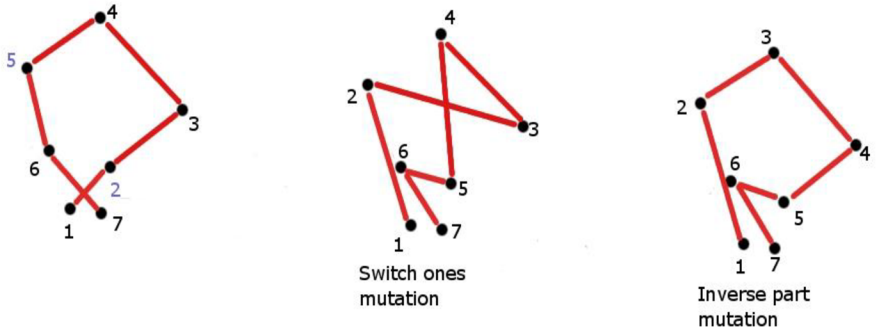
Bylo použito nastavení 65 10 nad bází 50 náhodně vygenerovaných měst s 5% výskytu mutací. Výpočet byl limitován časem 600 s. Získané informace pro nejlepší dosažené hodnoty z 20 testovacích cyklů jsem zaznamenal do tabulky 3.1.

Mutace	SOM	SPM	ILM	None
Nejlepší dosažená vzdálenost [km]	7448.5	7486.8	6901.2	12837.4
Počet generací	6527	6773	7011	7200
Čas[s]	560	582	592	590

Tabulka 3.1 Reprezentativní hodnoty pro uvedené mutace

Experimentováním jsem ověřil, že nejlepších vlastností dosahovala metoda inverze. Tato metoda převrací pořadí genů ve zvoleném úseku. Pokud trajektorii tvoří jednoduchá smyčka a ohraničení invertovaného úseku vnitřek smyčky úplně pokryje, výsledná trasa tuto smyčku obsahovat nebude. Pokud bude pokrytí částečné, je rozpletení smyčky také možné. V tomto případě může ale dojít ke vzniku smyček nových, které obecně celou trasu prodlužují.

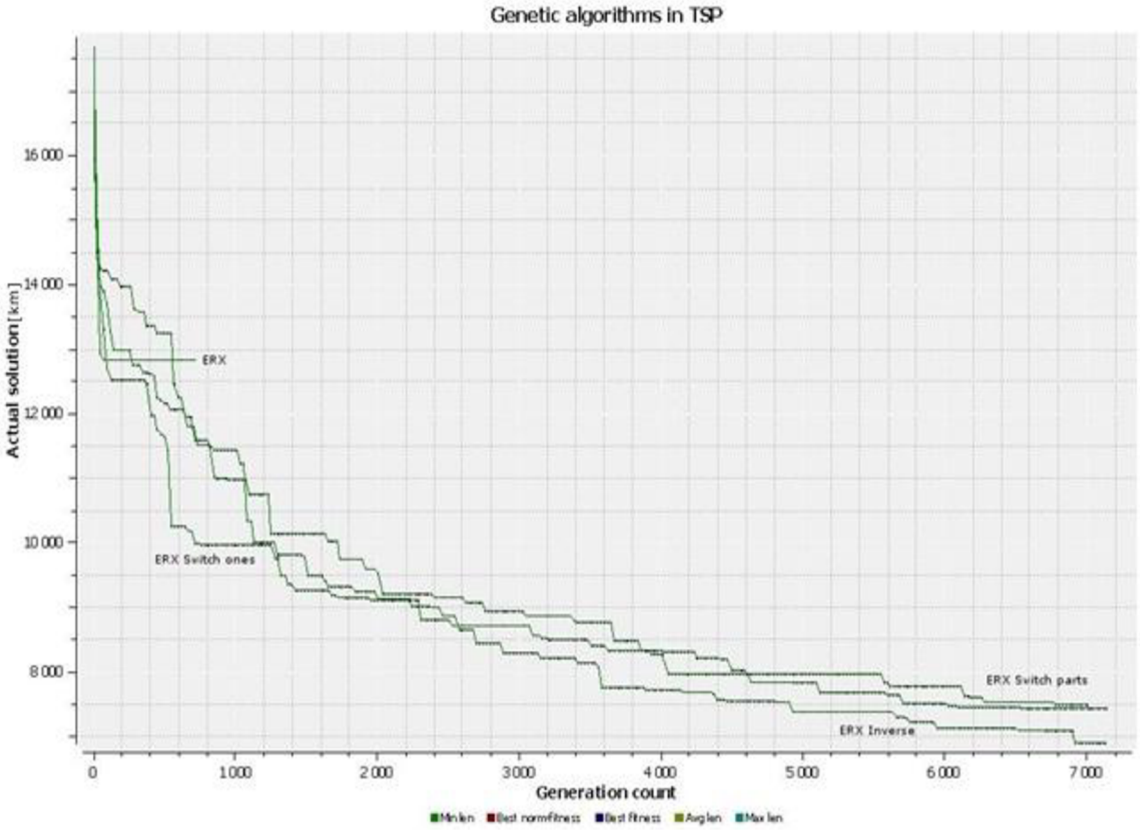
Mutace měnící dva body v chromozomu je podobná inverzní metodě. Na rozdíl od inverze může generovat aditivní smyčky tam, kde inverze pouze celou křivku převrátí. Typicky lze demonstrovat rozdíl takových metod na 4 prvcích při výběru dvou krajních uvnitř smyčky (obrázek 3.24).



Obrázek 3.24 Porovnání Switch ones mutation a Inverse mutation

Výměna dvou úseků dosahovala výsledků nejhorších. Nesleduje žádnou zřejmou myšlenku a je aplikovatelná především jako mechanizmus rozrůznění populace jedinci netypickými pro zvolenou metodu křížení s uchování vybraných posloupností chromozomu.

Následující graf demonstruje vliv jednotlivých mutací na zlepšování řešení, jak jsem popsal výše.



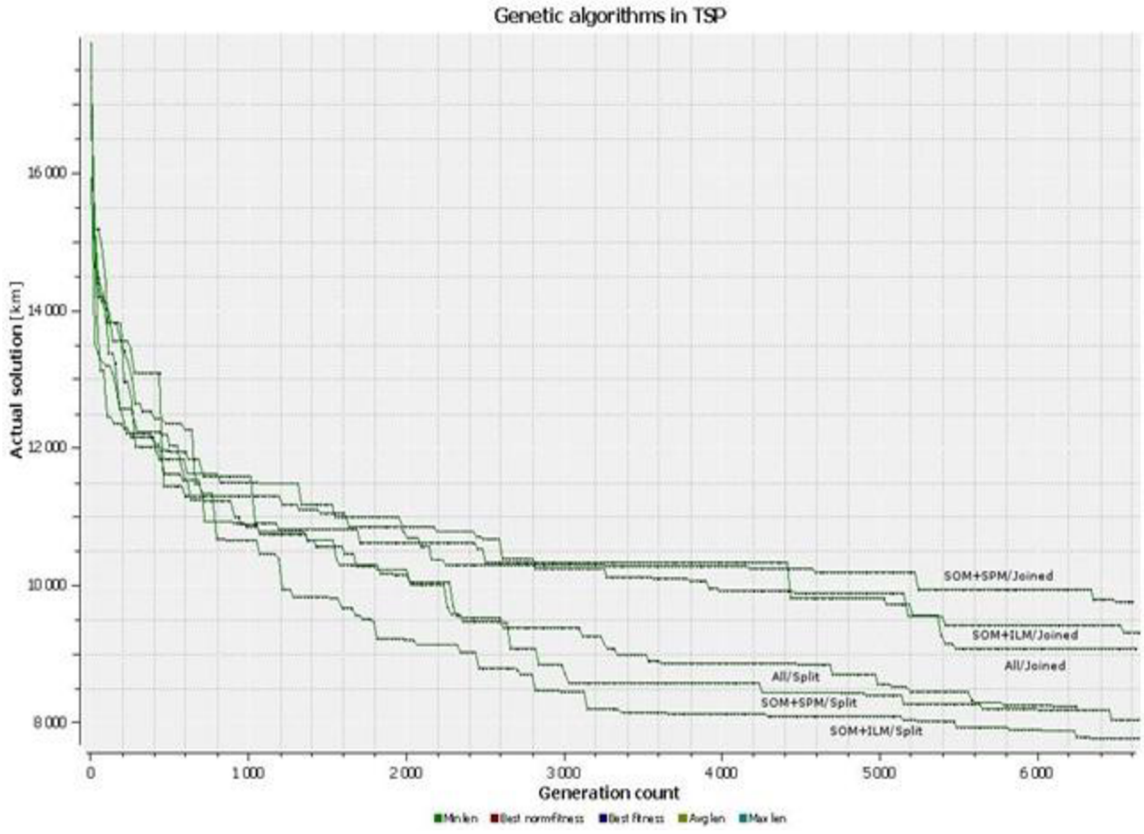
Obrázek 3.25 Zlepšování řešení vlivem mutací

Další variantu, kterou jsem testoval, byl vliv kombinovaných mutací, a to spojených či oddělených. Spojená mutace (joined) provede na jednom jedinci obě mutace po sobě. Mutace oddělené (split) provedou se stejnou pravděpodobností na jednom potomku pouze jednu ze zvolených mutací.

Pro své testování jsem zvolil tři kombinace se stejným nastavením jako v předchozím případě. Vytvořil jsem dvojice, dle výsledků, dvou nejlepších mutací, dvou nejhorších mutací a kombinaci všech tří mutací. Výsledky reprezentuje tabulka 3.2 a obrázek 3.26.

Mutace	SOM/SPM	SOM/ILM	ALL
Nejlepší dosažená vzdálenost (SPLIT) [km]	8044.3	7774.6	8247.5
Nejlepší dosažená vzdálenost (JOINED) [km]	9764.5	9326.9	9080
Počet iterací (SPLIT)	6472	6332	6099
Počet iterací (JOINED)	5488	6544	5473
Čas (SPLIT)[s]	556	544	590
Čas (JOINED)[s]	590	574	500

Tabulka 3.2 Vliv kombinovaných mutací na řešení

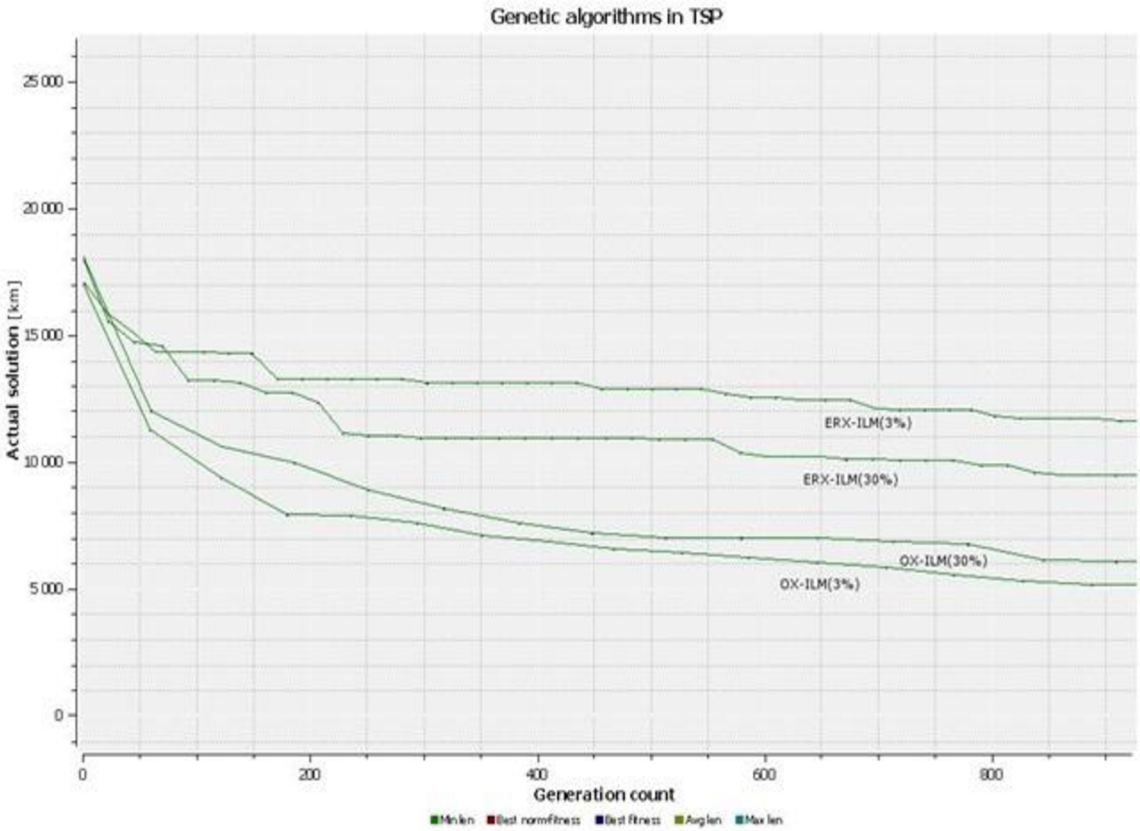


Obrázek 3.26 Porovnání split a joined kombinovaných mutací

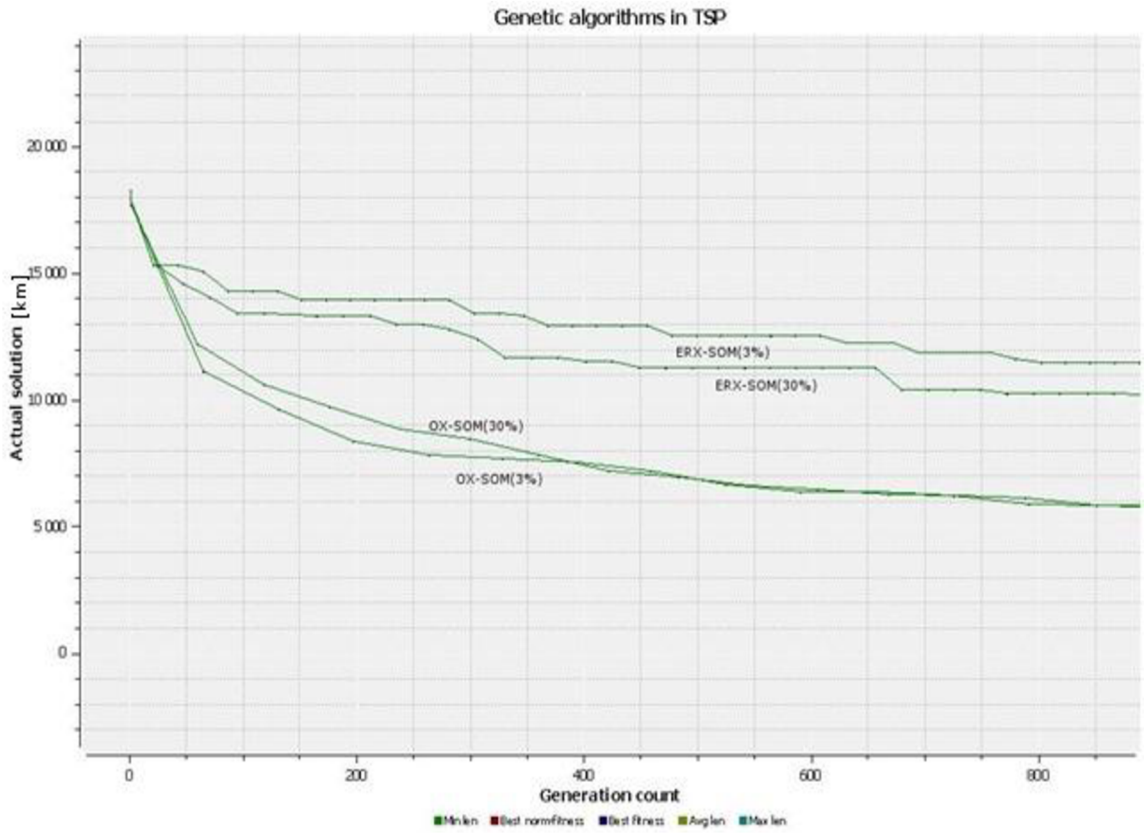
Z experimentů a přiložených materiálů vyplývá, že při oddělené kombinaci mutací se výsledek nezlepšuje. Ještě horších výsledných délek jsem dosahoval při spojení mutací, jelikož mohlo docházet k potlačení původní myšlenky využití mutace. Dobře je tento fakt vidět u dvou heuristických metod určených pro rozpletení jednoduchých smyček, kde je rozdíl mezi metodou typu split a joined největší.

V této části textu diskutuji vliv procentuelního výskytu mutace na efektivnost řešení. Pro experimentování zachovám nastavení z předchozích testů a budu porovnávat 3% a 30% mutace. Jako ukázku jsem zvolil dvě metody křížení, které v předchozích testech vykazovali různou schopnost rychle generovat dobrá řešení. Jsou jimi metody ERX a OX.

V prvním grafu (3.27) je použita mutace ILM (inverze), která prokazovala nejlepší vlastnosti. Ve druhém (3.28) SOM (záměnu dvou genů), která měla menší vliv při získávání lepších řešení.



Obrázek 3.27 Porovnání ERX a OX s mutací ILM (3% a 30% výskyt)



Obrázek 3.28 Porovnání ERX a OX s mutací SOM (3% a 30% výskyt)

Z prvního grafu vyvozuji následující. Pokud má metoda křížení dobré vlastnosti a dokáže sama generovat zlepšující se řešení (OX), je lepší použít mutace v menším zastoupení (optimální výsledky jsem dosahoval s 3% mutací). Větší zastoupení fungovalo kontraproduktivně, jelikož narušovalo kontinuitu rekurentního výpočtu následujících generací.

Naopak u metody ERX, která má menší schopnosti generovat zlepšující se řešení, zvyšující se procento mutací řešení zlepšovalo. Tato skutečnost mě vedla k použití dynamických mutací. Myšlenka vychází z předchozí hypotézy, kdy v prvních fázích výpočtu použijeme 3% mutaci. Dynamiku bude řídit počet iterací, kdy není porušena ukončující podmínka. V závislosti na růstu iterací splňující podmínku bude zvyšováno procento mutace, jelikož celé řešení vykazuje podobné vlastnosti jako ERX v prvním grafu (tedy neschopnost generovat dále se zlepšující řešení). Je ovšem nutné správně zvolit dynamiku vzhledem k bodu, kdy pro zvolené nastavení dochází ustálení zlepšování řešení.

Druhý graf reprezentuje skutečnost, že mutace s menším vlivem na progresi řešení mají menší schopnost odlišit dvě řešení lišící se procentuálním výskytem těchto mutací. Čím je metoda křížení efektivnější a čím menší schopnost prosadit se do řešení metoda mutací má, tím menší rozdíl mezi křivkami bude. Na základě výsledků bude testována metoda adaptivních mutací, které se spouští vždy až v poslední fázi intervalu splnění ukončující podmínky. Pokud bude správně nastavena dynamika, mělo by se jednat o místo, kdy ani zvolená mutace nebude příznivě ovlivňovat řešení, a proto se začnou náhodně střídat různé druhy mutací pro vygenerování odlišných potomků.

Ověřil jsem vliv mutací. Nejlepší vlastnosti prokazovala ILM. Kombinované mutace dosahovaly globálně horších výsledků. Jako nejvhodnější nastavení pro počty měst do 1000 jsem zvolil 3% mutaci. Nastínil jsem možnost užití adaptivních a dynamických mutací, kterou ověřím v dalších částech řešení.

3.1.8.3 Experimenty s optimálním nastavením

S nastavením 65-10 jsem provedl testy pro vybrané metody křížení. Na základě předchozího zkoumání jsem zvolil inverzní mutaci s výskytem 3%. Porovnával jsem vliv adaptivních dynamických mutací na rychlost aproximace nejlepšího možného řešení. Ustanovil jsem konstantní koeficient dynamiky na 350 neměnných iterací, který řídil zvyšování procentuálního zastoupení mutace. Testoval jsem také kombinovanou metodu. Zvolil jsem algoritmus OX s OLC jako akcelerátorem se silnou heuristikou.

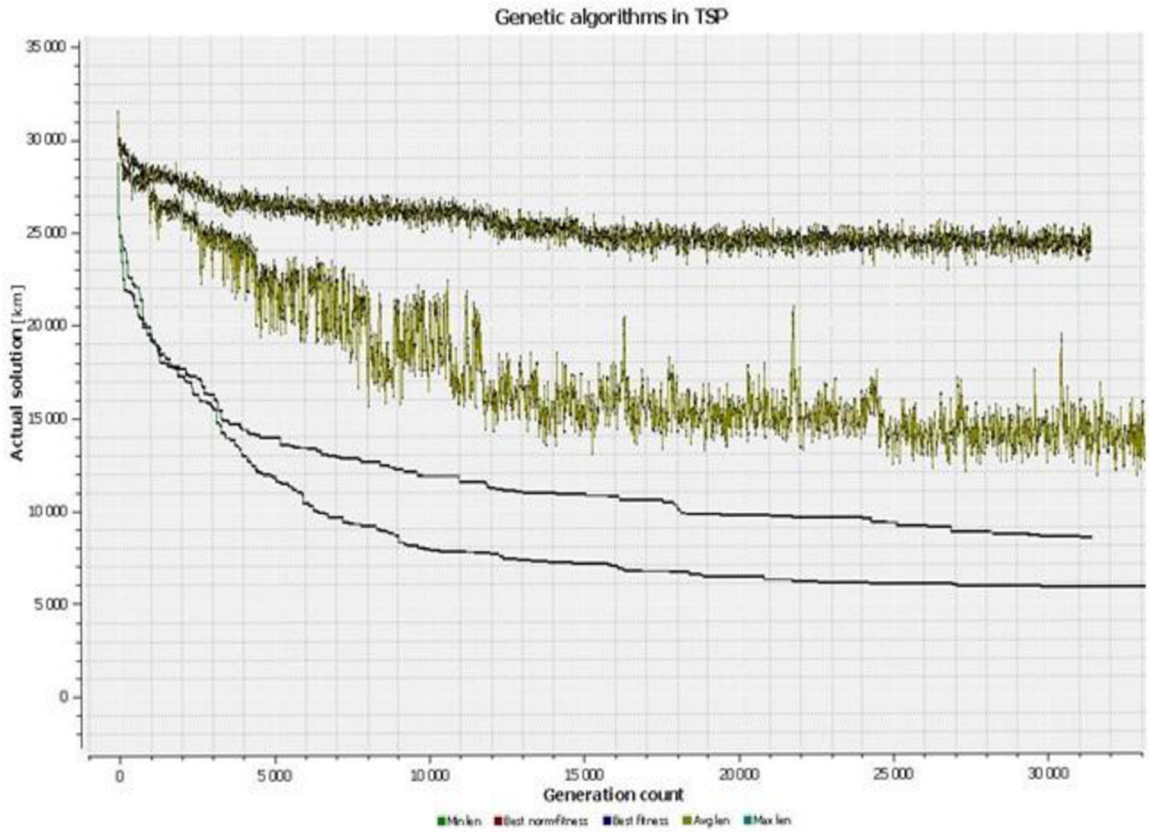
Ukončující podmínkou výpočtu byl logický součin predikátů času výpočtu překračujícího 2 hodiny a počtu 8000 iterací, během nichž nedošlo ke změně nejlepší hodnoty. Výsledky odpovídají nejlepším hodnotám z 10 testovacích běhů s množinou 80 měst.

Křížení	ERX	OX	OX-Opti	PMX	OLC+OX
Nejkratší délka [km]	8497	5605	5600	5732	5749
Nejkratší délka s optimalizací [km]	5778	5512	5771	5790	5500
Čas dosažení[s]	7232	1388	1580	926	2584
Čas dosažení s optimalizací[s]	5640	1276	1410	2404	438
Počet iterací	31109	15327	17991	10518	19434
Počet iterací s optimalizací	37900	22533	24885	45792	4884

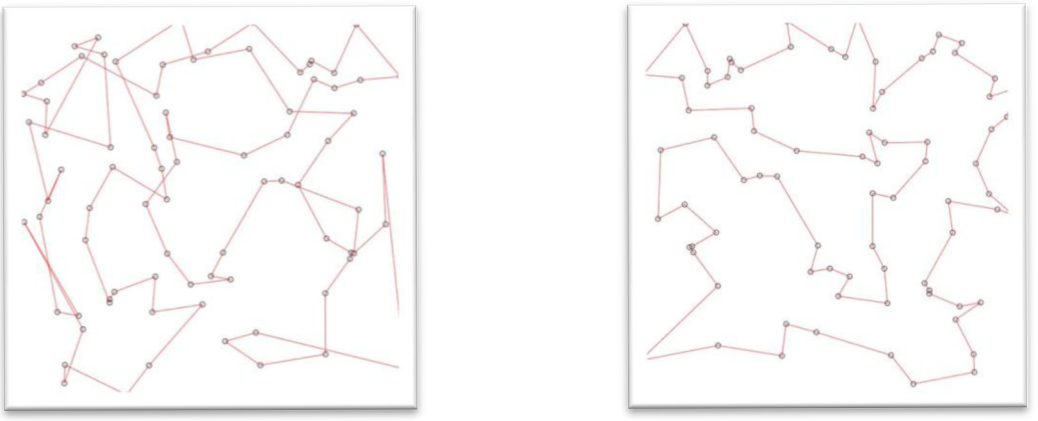
Tabulka 3.3 Přehledové výsledky

ERX

U metody ERX (obrázek 3.29) byl nejmarkantněji vidět vliv dynamické optimalizace. Všimněme si také křivky průměrné hodnoty a její oscilaci (spodní žlutá křivka) oproti stacionárnímu výpočtu (horní žlutá křivka).



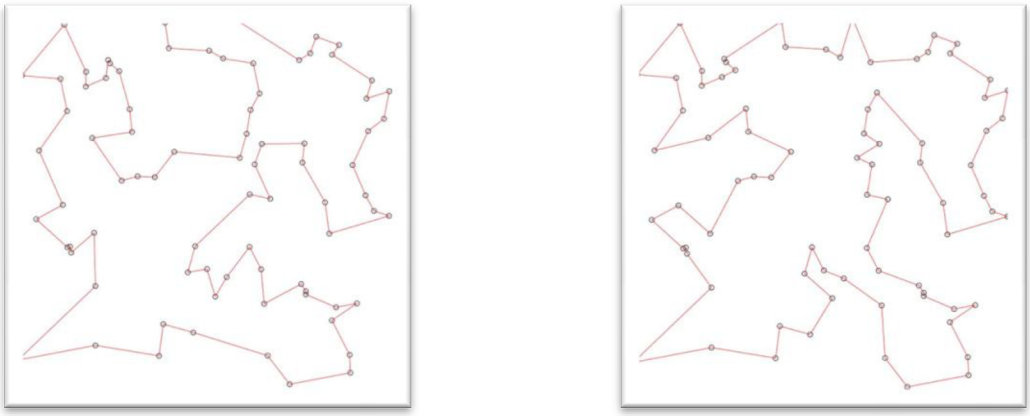
Obrázek 3.29 ERX s a bez dynamické mutace



Obrázek 3.30 ERX s normální a dynamickou mutací

OX

OX crossover dosahoval výborných výsledků již při předchozích testech. S optimalizací vygeneroval druhou nejlepší trasu.



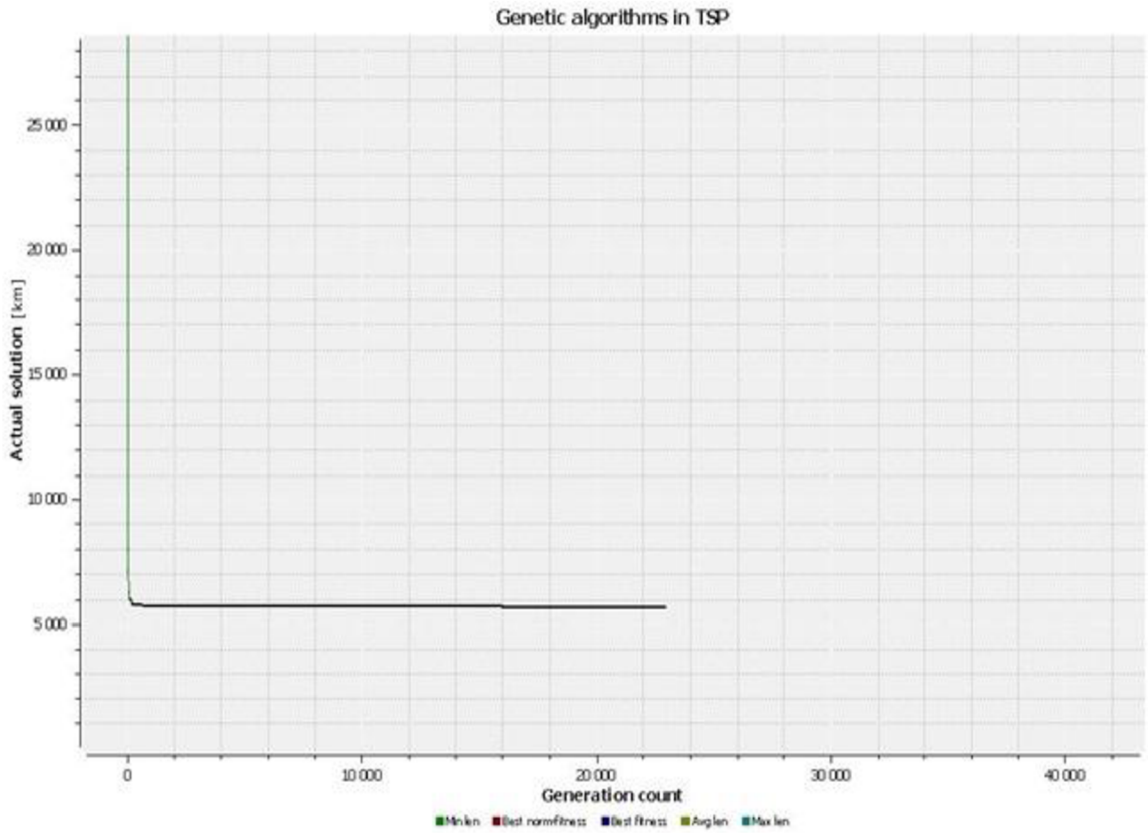
Obrázek 3.31 OX s normální a dynamickou mutací

OX (optimalizované)

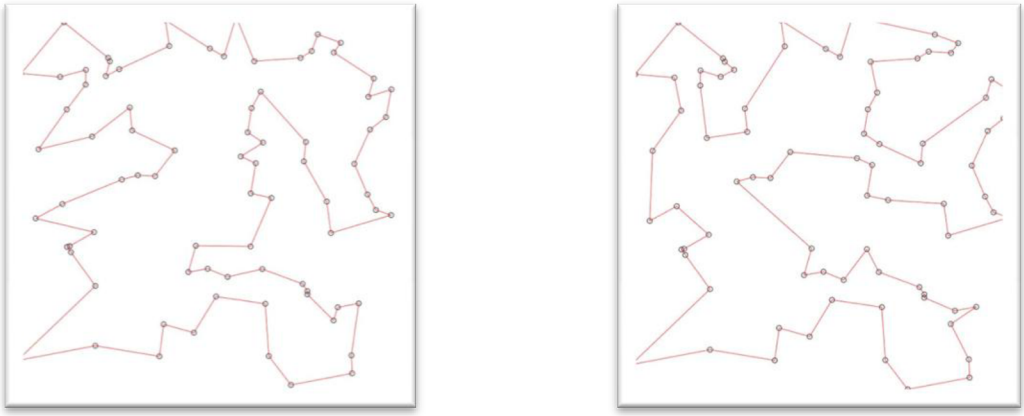
Pokus s použitím optimalizace algoritmu OX, která oslabila stochastickou složku výpočtu, ukázal, že dosahované řešení bylo průměrně horší než u klasického OX. Výpočty s použitím dynamické mutace se výrazně nelišily od těch s mutací statickou. Podobné výsledků jsem zaznamenal s metodou PMX. Obě totiž dosahovali velmi rychle stavů lokálních minim, které již další mutace nepříliš ovlivňovala.

OLC+OX

Testovaná kombinovaná metoda prokázala, jak efektivní je kombinace deterministicky založeného přístupu silné heuristiky a OX křížení. Výsledné řešení bylo dosahováno nejrychleji ze všech metod. Kvalitu aproximace ukazuje i strmě klesající křivka nejkratší dosažené vzdálenosti.



Obrázek 3.32 Průběh kombinovaného křížení



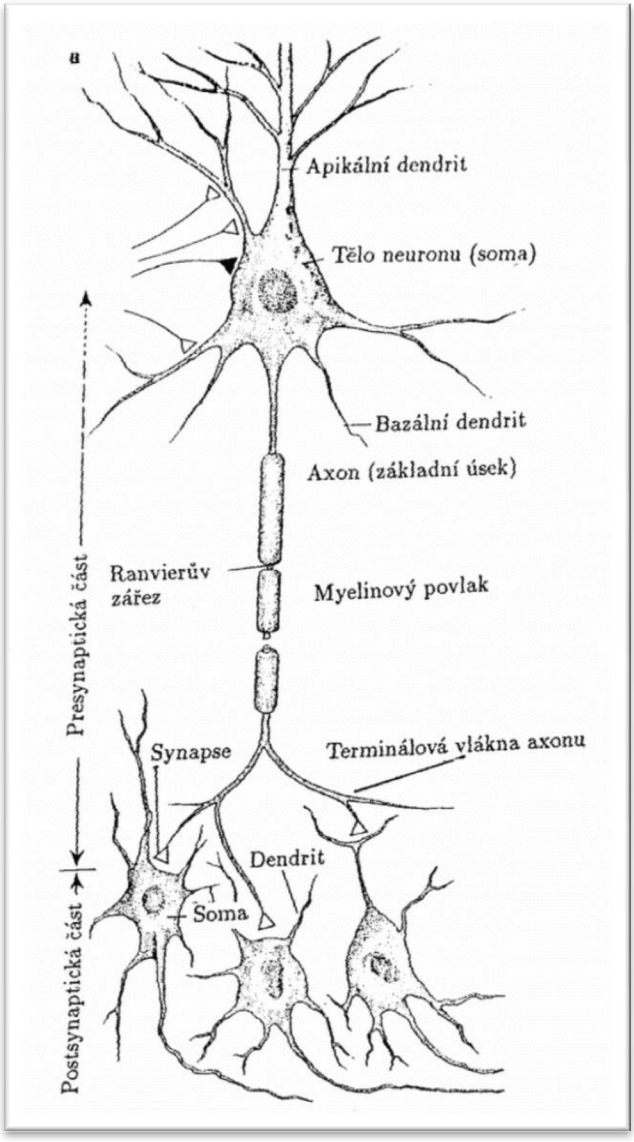
Obrázek 3.33 Kombinované křížení s a bez dynamické mutace

Jednotlivé metody byly otestovány s množinou 80 měst. U většiny metod se potvrdil pozitivní vliv dynamických a adaptivních přístupů. Nejrychleji a nejlépe generovalo řešení kombinované křížení. To také ukázalo výhodnost zkvalitnění počáteční generace cíleným přístupem.

3.2 Neuronové sítě

3.2.1 Princip činnosti

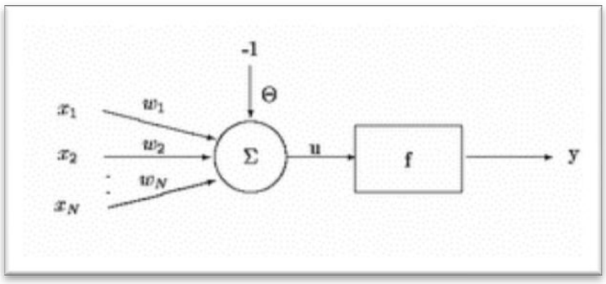
Jak již bylo řečeno v úvodu, neuronové sítě byly od počátku vývoje AI jedním z hlavních témat. Jedná se o výpočetní model, který pracuje na podobném principu jako skutečný mozek. Ten se skládá z neuronů, které mezi sebou vytváří topologii pomocí spojení axonů a dendritů (obrázek 3.34 (Veselovský, [200-])). Při učení se v našem mozku posilují konkrétní spojení mezi skupinami neuronů a tím se upevňuje uchování získané informace. Čím víc informaci studujeme, tím silnější vazby vznikají.



Obrázek 3.34 Spojení neuronů v mozku

Stejnou myšlenka je použita i při implementaci neuronové sítě. Vytváří se síť vzájemně propojených modelů neuronů. Tyto spoje jsou ohodnoceny váhami podle jejich důležitosti (obrázek 3.35 (Veselovský, [200-])). Rozhodující je, zda je neuron v síti aktivní či ne. To zjistíme pomocí evaluačních funkcí, které převedou

sumu součinu vah jdoucích z bodů přistupujících k neuronu s jejich aktivitou na hodnotu, která je porovnána s aktivační energií neuronu. Pokud ji energie vstupující do neuronu přesáhne, neuron se aktivuje.



Obrázek 3.35 AI model neuronu

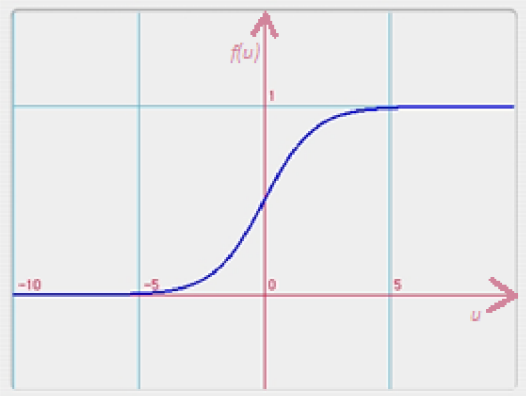
Aktivita neuronu y je dána převodní funkcí

$$y_j = f\left(\left[\sum_{i=0}^{N} w_{ij} x_i\right] - \theta\right) \tag{15}$$

x_i ... *aktivita neuronu i (hodnota 0 nebo 1)*
w_{ij} ... *váha spojení mezi neurony x_i a y_j*
y_j ... *výstupní hodnota neuronu y*
θ ... *prahový potenciál neuronu (aktivační)*

Aktivačních funkcí je několik. Mezi významné patří na příklad sigmoidální aktivační funkce

$$f(u) = \frac{1}{1+e^{-u}} \tag{16}$$



Obrázek 3.36 Sigmoida

Další aktivační funkce jsou například hyperbolický tangens nebo Heavisideova funkce (Šíma, 1996).

3.2.2 Rozdělení neuronových sítí

Hlavním kritériem, podle kterého se neuronové sítě dělí, je způsob, jakým se nastavují váhy mezi neurony:

- **Učení s učitelem**
  Je nutné použít množinu trénovacích vstupů, u nichž známe požadované výsledky. Síť zpracuje pomocí přednastavených vah vstup

a generuje výstup. Ten je učitelem srovnán s ideálním výstupem a podle výsledku dojde k upravení vah sítě (například pomocí backpropagation), aby generovala výsledky lepší. Je velmi důležité použití kvalitní testovací množiny, kterou můžeme optimalizovat například pomocí genetických algoritmů. Po fázi učení je neuronová síť připravena a můžeme ji použít k výpočtu.

- **Učení bez učitele**
  Síť se optimalizuje sama během výpočtu. Uvedeme příklad Kohonenovi neuronové sítě.
- **Nastavení pevných vah**
  Příkladem bude Hopfieldova neuronová síť. Zde rozebereme způsoby, jak pevné váhy popsat energetickými funkcemi.

Neuronové sítě můžeme také rozdělit pomocí topologie na jednovrstevné a vícevrstevné, které obsahují několik vnitřních vrstev mezi vrstvou vstupní a výstupní.

3.2.3 Kohonenova neuronová síť pro řešení TSP

Kohonenova síť představuje samoučící se variantu neuronové sítě. Obvykle se používá například pro rozpoznávání útvarů a při řešení TSP není tak obvyklá a ve své základní variantě nedosahuje příliš dobrých výsledků.

3.2.3.1 Princip funkčnosti

Kohonenova síť může mít topologii tvaru matice s body ohraničenými osmiokolím, nebo tvaru lineárního řetězce. Tuto variantu, kde mají jednotlivé neurony aktivní dvouokolí, použiji při řešení.

Váhy jednotlivých spojení jsou mapovány přímo na neurony a představují hodnotu dvourozměrného vektoru. Každý neuron je svým vektorem na počátku inicializován a během učení je vektor upravován.

Pro řešení TSP si představme vektory neuronů jako náhodně vygenerované pozice (jedna pro každé město) spadající do intervalu ohraničeného najkrajnějšími městy výpočetní mapy. V krocích se prochází množina propojovaných bodů a pro každý z nich se hledá vítězný neuron. Ten je v kontextu eukleidovské vzdálenosti nejblíže právě řešenému městu. Pro tento neuron provedeme úpravu vah.

Úprava vah proběhne podle Hebbova pravidla učení

$$w_{i1}(t+1) = w_{i1}(t) + \alpha h(I,i)(x_{i1} - w_{i1}(t)) \qquad (16)$$

$$w_{i2}(t+1) = w_{i2}(t) + \alpha h(I,i)(x_{i2} - w_{i2}(t)) \qquad (17)$$

w_{ij} ... *váha i. neuronu*
α ... *koeficient učení sítě, řídící velikost změn*
x_{ij} ... *souřadnice pozic měst*

$h(I, i)$ ... funkce predikátu *určujícího dosah změn (zda se v dané Manhattan vzdálenosti budou distribuovat změny provedené na indexu I)*

Funkce $h(I, i)$ nabývá dvou diskrétních hodnot, 0 nebo 1 a určuje, zda váha na indexu i bude upravována. Pokud vítězná váha leží o $|I - i|$ indexů dál od aktuálně řešeného vektoru a tolerance je menší nebo rovna této hodnotě, dojde ještě k jeho úpravě. Vzdálenost Manhattan reprezentuje v tomto případě počet neuronů mezi indexy I a i v lineární topologii Kohonenovy sítě se zavedeným dvouokolím (Šíma, 1996).
Koeficient α lze během propočtů snižovat tak, jako Manhattan vzdálenost určující dosah změn.

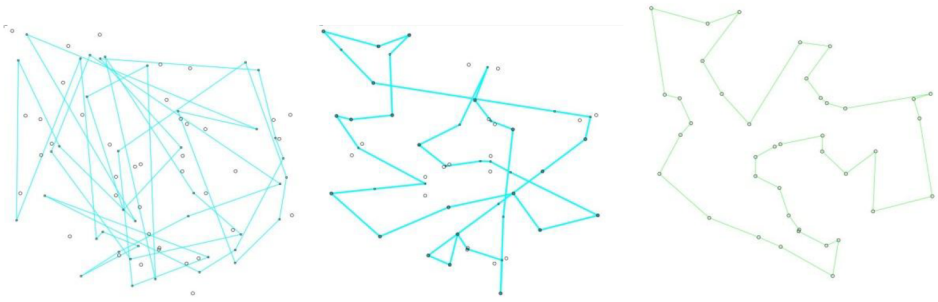
Výpočet sítě může být ukončen po přednastaveném počtu iterací bez rozdílu výsledku. V testovací aplikaci jsou ale nastaveny iterační 100 cyklové kroky, po nichž se výpočet pozastavuje. Výsledek přitom pouze aproximuje spojnice mezi body a skutečná trasa přes ně nemusí procházet. Také vygenerované neurony nemusí odpovídat v poměru 1:1 jednotlivým městům, ale mohou vznikat shluky.

3.2.3.2 Experimenty

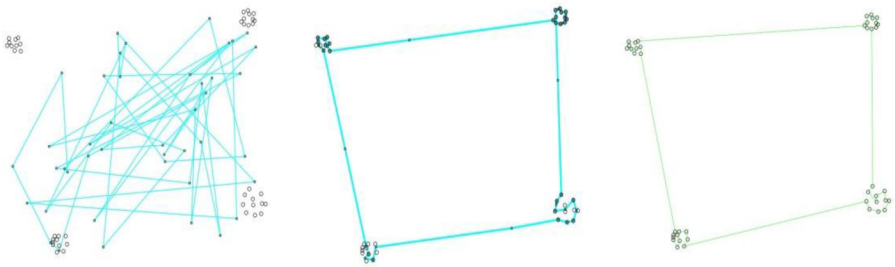
Experimenty potvrdily nepříliš velkou vhodnost Kohonenovy neuronové sítě v základní variantě jako nástroje řešení TSP. Síť se rychle ustalovala v rovnovážné poloze a její efektivnost s narůstajícím počtem měst značně klesala. Typický byl výskyt shluků měst a posun sítě do těžiště. Rozložení pomyslné hmoty odpovídá hustotě výskytu měst v inicializovaném úseku plátna.

Jediný faktor, ve kterém vykazovala neuronová síť vzhledem k způsobu dosahování řešení dobré výsledky, byl spojování konurbací měst (ukázka na obrázku 3.38). Jednalo se ovšem pouze o přibližné spojení, jelikož Kohonenova neuronová síť nemusí procházet přímo zvolenými městy. Existují situace, kdy je přibližné spojení dostačující a v tomto konkrétním případě by i tato síť mohla najít uplatnění.

Vždy je úspěch celého řešení závislý na počáteční inicializační množině.



Obrázek 3.37 Inicializace neuronové sítě, řešení a referenční řešení



Obrázek 3.38 Inicializace neuronové sítě se shluky, řešení a referenční řešení

Vzhledem k relativně rychlému ustálení by po menší úpravě mohla Kohonenova neuronová síť sloužit jako inicializační metoda evolučních algoritmů. Zvolená úprava by mohla například po ustálení sítě její jednotlivé prvky umisťovat do středů nejbližších ještě neobsazených měst, čímž bychom zajistili nutnou podmínku tvaru topologie chromozomu jedinců probíraných genetických algoritmů. Dodejme, že existují jednoduché postupy vycházející z popsaných metod, které velmi efektivně generují trasy až pro několik tisíc měst.

3.2.4 Hopfieldova neuronová síť pro řešení TSP

Hopfieldova neuronová síť reprezentuje rekurentní neuronovou síť s nastavením pevných vah schopnou učení. Její chování koresponduje s fyzikální teorií o magnetických materiálech. My budeme rozebírat spojitou adaptivní variantu založenou na minimalizaci energie (s diskretizací časových jednotek).

3.2.4.1 Princip funkčnosti

Obecně je síť definována jako propojení neuronů, které jsou zároveň vstupními i výstupními jednotkami a jsou mezi sebou navzájem propojeny tak, že každý neuron je spojen se všemi ostatními. Spojení sama na sebe může být také uvažováno.

Neurony jsou propojeny synaptickými vahami a vyskytují se v určitém stavu aktivity. Obvykle se jedná o binární či bipolární stav. Během výpočtu dochází ke vzájemnému ovlivňování jednotlivých prvků, dokud nenastane rovnovážný stav a síť se tzv. „nezrelaxuje“.

Nyní popišme energetickou funkci. Jedná se o nerostoucí, zdola ohraničenou hodnotu, která popisuje stav sítě (Volná, 2002).

Diskrétní Hopfieldova síť o n prvcích byla popsána energetickou funkcí

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} v_i v_j w_{ij} + \sum_{i=1}^{n} v_i \theta_i \qquad (18)$$

v_i ... *aktivita neuronu i*

w_{ij} ... *váha spojení mezi neurony* v_i *a* v_j

θ_i ... *práh aktivity neuronu (aktivní v případě* $\sum_{j=1}^{n} v_j w_{ij} > \theta_i$*)*

Ukažme, že změna aktivity neuronu koresponduje pouze s úbytkem energie celého systému. Energie i neuronu je na základě (18) dána vztahem

$$E_i = -\sum_{j=1}^{n} v_i v_j w_{ij} + v_i \theta_i \qquad (19)$$

Pokud je neuron aktivován, pak hodnota nové energie k němu vztažená je $E_i = -\sum_{j=1}^{n} v_j w_{ij} + \theta_i$, pokud je neuron deaktivován, je nová hodnota energie nulová. Změna energie je v obou případech dána vztahem

$$\Delta E_i = -\Delta v_i \left(\sum_{j=1}^{n} v_j w_{ij} - \theta_i\right) \qquad (20)$$

V případě kladné změny aktivity neuronu (z 0 do 1) je druhý činitel výrazu (20) kladný a celkově se jedná o úbytek energie. Stejně je tomu i v opačném případě (blíže o tomto problému pojednává Lyapunova funkce stability systému). Podotkněme, že v případě záporných vah dochází k rozkmitání dvojice touto vahou spojených, protože ta způsobí rozdílné stavy těchto neuronů (naopak kladná hodnota vah podporuje udržení souhlasných stavů)

Nastavení vah může být jak na kladné, tak záporné, popřípadě nedefinované hodnoty. Toto se odvíjí od charakteru řešeného problému.

Spojité systémy pracují ve spojitém čase s aktivitami definovanými spojitou funkcí potenciálu. Nejedná se tedy přímo o binární či bipolární hodnotu, ale o reflexi vývoje hyperbolického tangentu.

Energetickou funkci obecně definuje Hopfield-Tank model sítě (Potvin, 1993)

$$E = -0{,}5 \sum_i \sum_j v_i v_j w_{ij} - \sum_i v_i I_i + \sum_i \int_0^{v_i} g^{-1}(x)dx \qquad (21)$$

v_i ... *hodnota aktivity i neuronu*

w_{ij} ... *váha mezi i a j neuronem*

$g(x)$ ... *funkce definující aktivitu neuronu z jeho potenciálu, inverze pak z aktivity vyjadřuje potenciál, ve výrazu způsobuje posun lokálních minim ke středu hyperkrychle (Mańdziuk, 1996).*

I_i ... *externí vstup do sítě (například podmínka vzdálenosti měst)*

Pokud je funkce g monotónně rostoucí a matice neuronů je čtvercová, lze dokázat, že E je Lyapunova funkce. Derivace podle času (vyjádřená za pomoci parciálních derivací dle potenciálu a aktivity neuronu (Berg, 2007)) je menší nebo rovna nule, funkce je klesající.

Změnu potenciálu neuronu, ze kterého vychází jeho aktivita, popisuje diferenciální rovnice

$$\frac{du_i}{dt} = -\frac{\partial E}{\partial v_i} \qquad (22)$$
$$\frac{du_i}{dt} = \sum_j v_j w_{ij} + I_i - u_i$$

Rozeberme chování systému popsaného jako parciální derivaci energie dle aktivity neuronu. Můžou nastat 4 základní stavy:

- **Úbytek energie systému, snížení aktivity neuronu**
  Úbytek potenciálu neuronu.
- **Úbytek energie systému, zvýšení aktivity neuronu**
  Přírůstek potenciálu neuronu.
- **Přírůstek energie systému, snížení aktivity neuronu**
  Přírůstek potenciálu neuronu.

- **Přírůstek energie systému, zvýšení aktivity neuronu**
  Úbytek potenciálu neuronu.

Z rozboru je vidět, že systém posiluje změny vedoucí ke snížení energie a naopak oslabuje změny (změnou potenciálu) vedoucí k jejímu zvýšení.

Na základě (22) pro spojitou síť je vyvozen vztah pro potenciál i neuronu ve stavu nulové směrnice, tedy bodu ustálení, jako

$$u_i = \sum_{j=1}^{n} v_j w_{ij} + I_i \qquad (23)$$

Z potenciálu neuronu lze převodním vztahem odvodit jeho aktivitu. Pro její odvození se používají sigmoidální funkce. Použijme hyperbolický tangens, jehož limita vstupuje v nekonečnu do hodnoty 1 resp. -1. Aktivita pak odpovídá výrazu

$$v_i(u) = 0{,}5\{1 + \tanh(\propto u)\} = 0{,}5\left\{1 + \frac{e^{2\propto u}-1}{e^{2\propto u}+1}\right\} \qquad (24)$$

$v_i(u)$ ... *aktivita i. neuronu s aktuálním potenciálem u*
α ... *koeficient, který ovlivňuje velikost směrového vektoru tečny, resp. rychlost změn aktivity*

Hopfieldova síť implementovaná jako řešení TSP tvoří mřížku o n^2 prvcích. Ty vytváří vertikální vrstvy, kdy v každé vrstvě jsou neurony reprezentující jednotlivá města a jsou vzájemně propojeny se všemi neurony sousedních dvou vrstev. Pořadí města v trase odpovídá pořadí vrstvy v síti.

Spojení neuronů mezi sebou nebudeme popisovat vahami, ale pokusíme se odvodit energetickou funkci soustavy, pomocí níž budeme schopni vyjádřit potenciál každého neuronu. Jako zdroj energie využijeme soustavu omezujících podmínek. Indexy u neuronů v pořadí značí konkrétní město a pořadí v trase.

- V řádcích neuronové sítě, které reprezentují jednotlivá města, musí být aktivován vždy právě jeden neuron. Počet stejných neuronů aktivovaných ve stejném řádku, tedy počet opakujících se měst, je dán sumou.
  $$E_1 = 0{,}5 \sum_x \sum_{i=0}^{n} \sum_{j=1, j\neq i}^{n} v_{xi} v_{xj} \qquad (25)$$
  Celá suma se dělí kvůli symetrickému započítávání duplicitních měst.
- Ve sloupcích sítě, které odpovídají pořadí města v trase, musí být aktivován opět právě jeden neuron.
  $$E_2 = 0{,}5 \sum_i \sum_{x=1}^{n} \sum_{y=1, y\neq x}^{n} v_{xi} v_{yi} \qquad (26)$$
- Musí být navštíveny všechna města.
  $$E_3 = 0{,}5\left(\sum_x \sum_i v_{xi} - n\right)^2 \qquad (27)$$
- Poslední podmínka se vztahuje k samotné vzdálenosti. Výpočet je součtem dílčích vzdáleností mezi sousedními městy.
  $$E_4 = 0{,}5 \sum_x \sum_{y, x\neq y} \sum_{i=1}^{n} d_{xy} v_{xi}\left(v_{y,i-1} + v_{y,i+1}\right) \qquad (28)$$

d ... *vzdálenost mezi body x , y*

Každá energetická funkce se projevuje v určité míře. Od toho je pak odvislý její vliv na celkové řešení Tyto míry vyjadřují koeficienty *A, B, C, D*. Celkovou energii soustavy proto uvažujeme jako

$$E = AE_1 + BE_2 + CE_3 + DE_4 \tag{29}$$

Restrikce tvořené energetickými funkcemi byly upravovány různými optimalizacemi, které kladli důležitost na specifické faktory. My se budeme zabývat verzí uvedenou v (29).

Nyní se vraťme ke vzorci (22). Ve spojitém systému, který neuvažuje čtvercovou topologii, je velikost rychlosti změny potenciálu neuronu dána vztahem (pokud bychom čtvercovou topologii uvažovali, museli bychom přidat sumaci přes index $y \neq x$, kde x je index v rámci počítaného potenciálu)

$$\frac{u_i(t+\Delta t)-u_i(t)}{\Delta t} = \sum_j v_j(t) w_{ji}(t) + I_i - u_i(t) \tag{30}$$

Ve spojité simulaci do diferenciální rovnice zakomponujeme (22) a (29) a uvažujeme čtvercovou topologii na řešení TSP.

$$\frac{du_{xi}}{dt} = -E - \frac{u_{xi}}{\tau}; \tau = 0$$

$$\frac{du_{xi}}{dt} = -A \sum_{j=1, j\neq i}^{n} v_{xj} - B \sum_{y=1, y\neq x}^{n} v_{yi} - C\left(\sum_x \sum_i v_{xi} - n\right)$$
$$-D\left\{\sum_{y\neq x} d_{xy}\left(v_{y,i-1} + v_{y,i+1}\right)\right\} - \frac{u_{xi}}{\tau}; \tau \neq 0 \tag{31}$$

Proto hodnota nového potenciálu neuronu je po infinitezimálním časovém okamžiku

$$u_{xi}(t+\Delta t) = \left(-A \sum_{j=1, j\neq i}^{n} v_{xj} - B \sum_{y=1, y\neq x}^{n} v_{yi} - C\left(\sum_x \sum_i v_{xi} - (n+\varrho)\right)\right.$$
$$\left. - D\left\{\sum_{y\neq x} d_{xy}\left(v_{y,i-1} + v_{y,i+1}\right)\right\} - \frac{u_{xi}}{\tau}\right)\Delta t + u_{xi}(t); \tau \neq 0 \tag{32}$$

Výsledek (32) představuje rekurentní vztah závislý na předchozí hodnotě potenciálu. Infinitezimální velikost časové změny Δt se bude pohybovat v rozmezí 10^{-5} až 10^{-6} *[s]*. Menší časová změna by měla dosahovat lepších výsledků s pomalejší konvergencí.

Můžeme porovnat hodnotu účelové energetické funkce a energetické funkce Hopfieldovy neuronové sítě ((29), (21) bez posledního členu). Použijeme proto tzv. Kroneckerovo delta. Platí $(\delta_{ij} = 1 \wedge i = j) \vee (\delta_{ij} = 0 \wedge i \neq j)$

$$w_{xiyj} = -A\delta_{XY}\left(1 - \delta_{ij}\right) - B\delta_{ij}(1 - \delta_{XY}) - C - Dd_{XY}\left(\delta_{i,j-1} + \delta_{i,j+1}\right)$$
$$I_{Xi} = CN_e \tag{33}$$

N_e ... *konstanta je většinou mírně větší než počet řešeních měst a reprezentuje pozitivní vstupní signál*

Cílem řešení je nalézt minimum energetické funkce, nejlépe to globální. Proto se snažíme upravit potenciál v (32) na hodnotu co nejbližší 0. V tomto ohledu budou hrát důležitou roli již zmiňované pevné váhy *A, B, C a D* (Potvin, 1993).

3.2.4.2 Metody úniku z lokálního minima

Do Hopfield-Tankova modelu zaveďme model pravděpodobnosti přijmutí nového energetického stavu. Využijeme stejnou množinu energetických funkcí v kombinaci s pravděpodobností p příjmu nové hodnoty potenciálu. Tato pravděpodobnost bude upřednostňovat změny vedoucí k minimalizaci energetické funkce.

$$p(u_{xi}) = \frac{1}{1+e^{\frac{\Delta E(u_{xi})}{T}}} \tag{34}$$

$\Delta E(u_{xi})$ ... *změna energie konkrétního neuronu při novém potenciálu* u

T ... *aktuální teplota, při jejíž vyšší hodnotě jsou přijímána i horší řešení*

p ... *pravděpodobnost příjmu nového stavu*

Po zvoleném neměnném počtu iterací se dostane síť do stavu rovnováhy a pravděpodobnost změny stavu bude vycházet pouze z energetických funkcí systému a právě zpracovávaného neuronu (Potvin, 1993).

$$p(u_{xi}) = \frac{e^{-\frac{E(u_{xi})}{T}}}{\sum_{w} e^{-\frac{E(w)}{T}}} \tag{35}$$

Pro implementační část je použita metoda žíhání, kde pravděpodobnost vychází ze vztahu

$$p(u_{xi}) = e^{\frac{E_{uxiOld} - E_{uxiNew}}{T}} \tag{36}$$

$E_{uxiOld/New}$ ... *energetická hodnota neuronu ve stavu před a po zavedení nového potenciálu*

T ... *teplota, při jejíž vyšší hodnotě jsou přijímána i horší řešení*

p ... *pravděpodobnost příjmu nového stavu*

Je nutné správně určit teplotu a průběh ochlazování tak, aby se systém choval přirozeně s tím, že stav vedoucí k minimalizaci energie bude přijat vždy.

3.2.4.3 Postup algoritmu

1 Náhodně nastavíme aktivitu neuronů na hodnotu $\frac{1}{n} \pm < 0, \beta >$; $\beta = 0{,}03$.

2 Inicializujeme pole vzájemných vzdáleností jednotlivých měst, normalizujeme jej a inicializujeme počáteční energii systému.

3 Je vybrán náhodně jeden z neuronů, u kterého nastavíme novou hodnotu potenciálu a aktivity (můžeme zakomponovat metodu úniku z lokálního minima). Neuron je aktivní, přesahuje-li jeho hodnota koeficientu v nastavenou aktivační hladinu (doporučené hodnoty bývají přibližně 0,9) a je

deaktivován při poklesu aktivity pod hodnotu deaktivační hladiny (doporučené hodnoty bývají přibližně 0,1).

4 Bod 3 opakujeme $5n^2$, kde n^2 představuje interní cykly, po $5n^2$ cyklech proběhne jeden externí, během nějž se přepočítá energie systému.

5 Pokud nová hodnota energie splňuje podmínku ukončení, skončíme, jinak se vracíme zpět k bodu 3. V aplikovaném algoritmu ukončující podmínka není použita a aplikaci ukončuje uživatel sám.

3.2.4.4 Optimalizace popsaných postupů

Hopfield-Tank model je optimalizován velkým množstvím způsobů, které upravují vztahy v energetické funkci tak, aby zdůraznily jednotlivé faktory ovlivňující řešení. V mé aplikaci jsem použil kromě výše popsaného postupu dva upravené.

První upravená metoda nepracuje rekurentně s potenciálem předchozího stavu neuronu, respektive místo infinitezimálního časového okamžiku jako doby mezi následujícími stavy volíme jednotkový skok, kterým se nutnost použití předchozího potenciálu eliminujeme.

Druhý postup opět vychází z původní myšlenky energetické funkce a Lyapunova funkce předpokladu minimalizace. Empiricky se zde ovšem nastavují koeficienty vzájemných vztahů a je upravena transformační sigmoidální funkce.

Rozebereme aktivační funkci pro jeden zvolený neuron v mřížce Hopfieldovy sítě.

Cílem první energetické funkce je aktivace konkrétního města pouze jedenkrát pro zvolenou trasu. V celkové sumě se uvažuje odečet aktivity samotného neuronu na indexech *i* a *j*.

$$E_{1ij} = \sum_{jj=0}^{n} v_{i,jj} \tag{37}$$

Druhá energetická funkce postihuje více různých aktivovaných neuronů pro zvolenou pozici.

$$E_{2ij} = \sum_{ii=0}^{n} v_{ii,j} \tag{38}$$

Energetická funkce definující podmínku co nejkratší trasy zní

$$E_{3ij} = \sum_{ii=0}^{n} dist_{ii,i} \cdot \left(v_{ii,j-1} + v_{ii,j+1}\right) \tag{39}$$

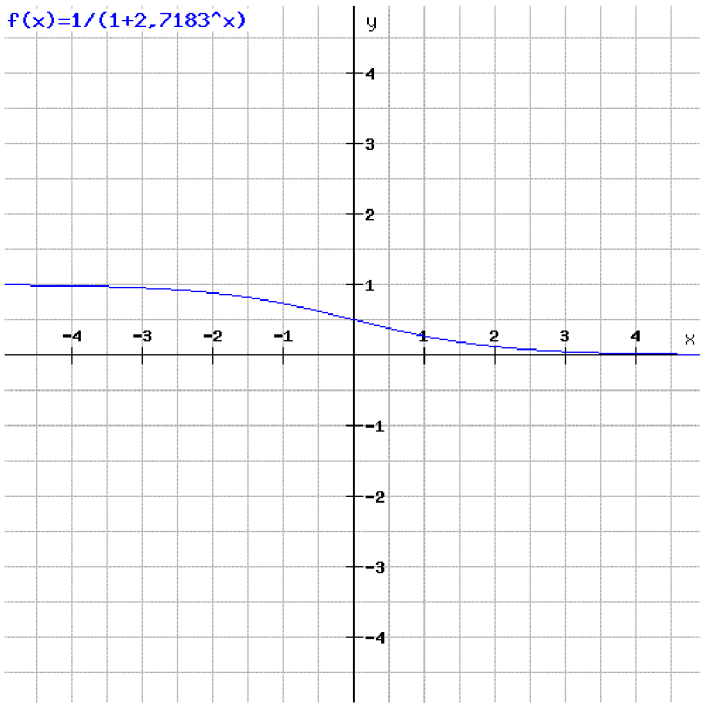
Výsledek ovlivňuje nejenom předpis výsledné sumy, ale i tvar sigmoidy, která definuje aktivitu 1 ve svém druhém kvadrantu. Vzhledem k použitým koeficientům se pro stavy se správným rozložením potenciálu vývoj sítě ustálí.

$$E_{ij} = 5\left(E_{1ij} + E_{2ij} - 2v_{i,j}\right) + 10E_{3ij} + 50\left(E_{1ij} + E_{2ij} - 2\right) - 15 \tag{40}$$

Pak hodnota nové aktivity neuronu bude

$$v_{i,j} = v_{i,j} + 0{,}02\left(\frac{1}{1+e^{E_{ij}}} - v_{i,j}\right) \tag{41}$$

Převodní vztah pro stávající energetický stav odpovídá obrázku 3.39



3.39 Sigmoidální funkce upraveného Hopfield-Tank modelu

3.2.4.5 Zhodnocení metod

V aplikaci jsem použil tři možné režimy aktivace neuronů mřížky. První z nich byl založen striktně na hladině aktivity a prahu aktivace a deaktivace. Zbylé dva módy kladly důraz na unikátnost a výskyt všech měst, resp. na unikátnost pořadí a typu města v závislosti na pozici města v mřížce. V obou dvou postupech nefigurovala přímo deaktivační hladina, nýbrž prvek s větší aktivitou odstranil prvek kolizní, pokud jeho aktivita byla nižší.

U testovaných metod jsem použil mód striktní aktivační a deaktivační hladiny nastavené na deaktivaci *0,1* a hranici aktivace v rozmezí *0,9 -1*.

Ani jedna z prvních dvou metod založených na parametrické energetické funkci nedosahovala dobrých výsledků. U obou hrálo důležitou roli empirické testování stávající konfigurace, se změnami jednotlivých váhových koeficientů *A, B, C a D*.

- **Metoda energetické funkce (rovnice 32)**

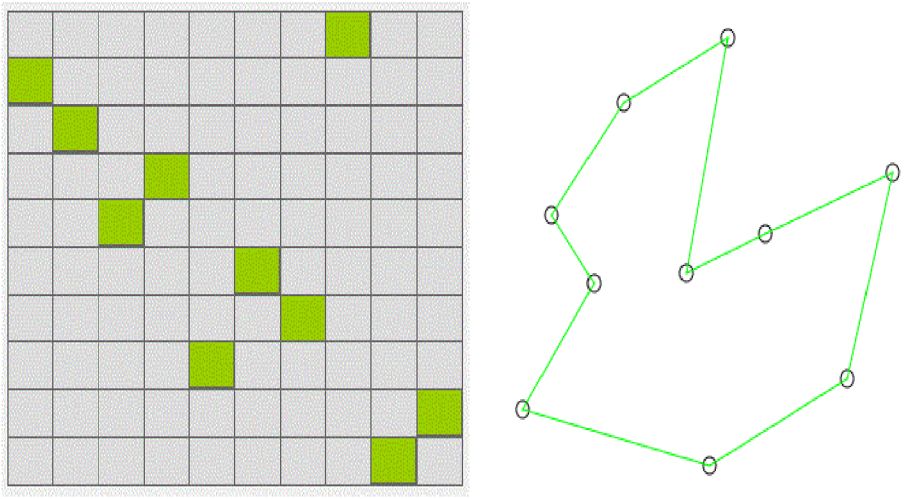
  Rekurentní vývoj stavu sítě se ukázal jako ne příliš efektivní, především pak s malým krokem $h = 10^{-5}$ *[s]*. Čím menší krok je nastaven, tím by měl vývoj řešení trvat déle, při přepokládané větší přesnosti. První předpoklad se potvrdil, druhý nikoliv. Časová konvergence do ustáleného stavu byla pro množinu 10 měst v řádu stovek milisekund až jednotek sekund. S narůstajícím stavovým prostorem se čas značně prodlužuje. Často se stávalo, že skupiny měst nebyly vůbec díky vývoji sítě aktivovány.

  Z tohoto pohledu se mnohem lépe jevil krok jednotkový, který v mé aplikaci představoval samostatnou variantu pro testování. Na základě závěrů z (Mańdziuk, 1996) jsem použil nastavení proměnné ϱ z (32) na hodnotu *1,1*, která působí jako zdůraznění restrikce přesného počtu měst. Takové

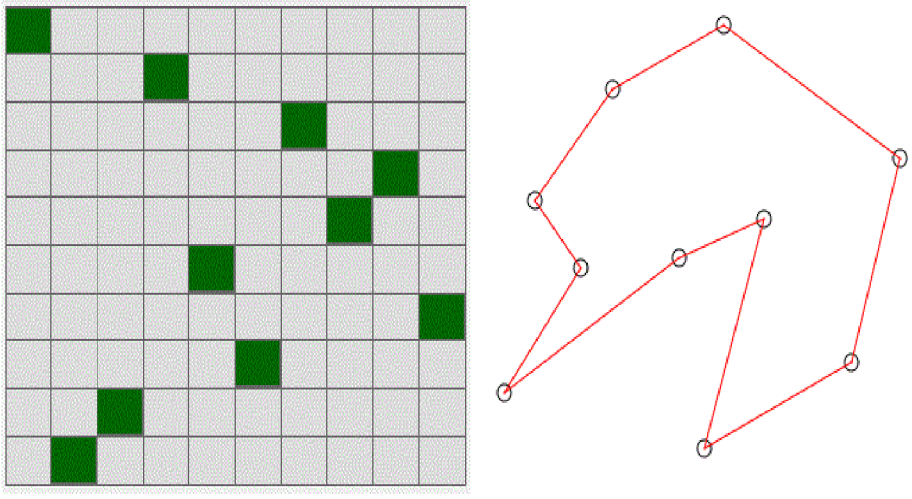
nastavení je optimalizováno pro řetězec 10 měst. Vzhledem k jednotkovému kroku dochází k rychlému vývoji celé sítě. Při vzrůstajícím počtu měst se systém může rozkmitat vlivem skokových změn stavu některých neuronů, které nedovolí adaptaci celé sítě. U menšího kroku k ustálení dochází s větší pravděpodobností. Ani u jednoho z kroků se mi nepodařilo prokázat pozitivní vliv simulovaného žíhání, jakožto metody bránící uváznutí v lokálním minimu. Vykládám to především špatným nastavením rychlosti ochlazování celého systému.

- **Metoda upravené energetické funkce (rovnice 42)**

Tato převzatá metoda klade větší důraz na unikátnost aktivace jediného neuronu v rámci řádku a sloupce. Dokázala konvergovat a ustálit se i pro větší počty měst (až 50 měst) při aplikaci striktní aktivace neuronů. Při takovémto stavovém prostoru generovala průměrně 4 krát delší trasy, než-li byl ideální nejkratší okruh. Dosahovala nejlepších průměrných výsledků u testování nad množinou 10 měst (viz. tabulka 3.4).



Obrázek 3.40 Ukázka nejlepšího dosažitelného řešení s maticí Hopfieldovy sítě



Obrázek 3.41 Ukázka průměrného řešení s mapou aktivovaných neuronů

	Odchylka průměrného řešení od absolutního minima	**Odchylka dosaženého minima od absolutního minima**
Metoda energetické funkce (32) s infinitezimálním krokem	(+)48%	(+)21%
Metoda energetické funkce (32) s jednotkovým krokem	(+)29%	0%
Optimalizovaná metoda (42)	(+)26%	(+)14%

Tabulka 3.4 Výsledky testování 20 vzorky nad okruhem s 10 městy

Empiricky jsem dospěl u prvních dvou metod k nastavení koeficientů *A, B, C, D* na hodnoty (500, 500, 200, 300) resp. (100, 100, 90, 110), které potvrzují závěry z (Potvin, 1993). Je vidět, že u všech metod je kladena stejná váha na první dva koeficienty unikátnosti výskytu v řádku a sloupci. Je také patrný podobný poměr rozložení vah, kvantitativně upravených potřebě kroku. Velmi důležitou složkou je koeficient D, který představuje ideu co nejkratší vzdálenosti. Proto, abych odstranil nutnost velkých změn nastavení pro jiné konfigurace mapy, jsem všechny vzdálenosti normalizoval do intervalu $\langle 0; 1\rangle$

Při testování uvedených hodnot parametrů se objevovaly potíže s malým počtem měst (do 5 měst). Při takovém nastavení funkce v dosažitelném čase nenabývala minimální aktivity nutné k aktivaci neuronu. Řešení spočívá například ve variabilním přenastavení vah energetické funkce.

S nastavením části energetické funkce zabývající se co nejkratší celkovou vzdáleností vyplývá ještě jeden problém. Ten vzniká, pokud propojujeme shluk měst, která mají k sobě podobnou vzdálenost v určitém intervalu a jedno osamocené město, které je vzhledem k maximálním vzdálenostem ve shluku několikanásobně dál (bráno opět jako průměrná vzdálenost města ke shluku). Ukázka viz. obrázek 3.42.

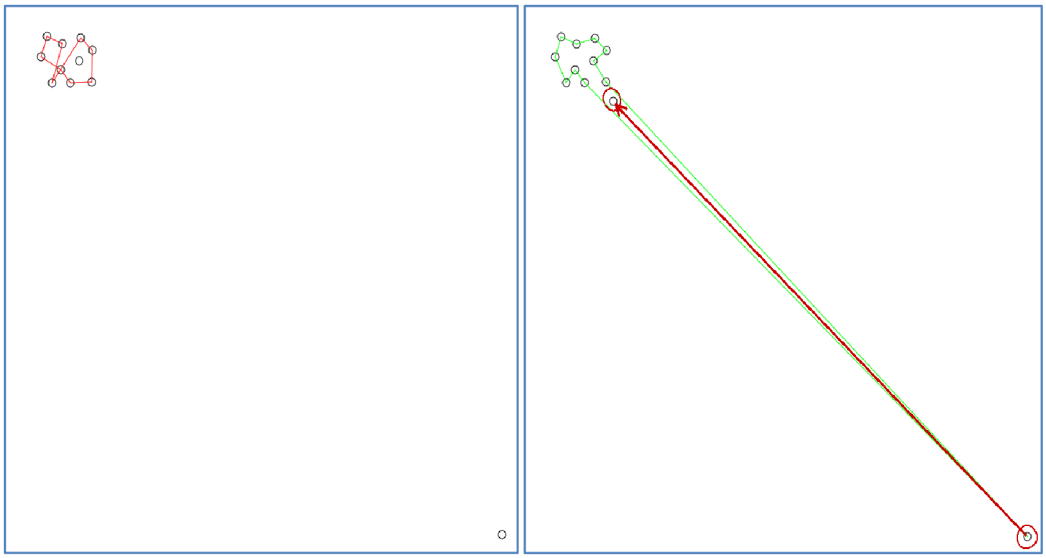
3.42 Ukázka problémového nastavení mapy

Při nastavení vysokého prahu aktivity pak toto město nemusí být vůbec uvažováno v cestě. Je to způsobeno tím, že má z celé množiny největší hodnotu energetického prvku s koeficientem *D*, což bude zvyšovat jeho energetickou hodnotu a v závislosti na tom, v kontextu Lyaponovy funkce, bude klesat jeho aktivita. Celý systém se tak vychýlí z ideální rovnovážné pozice. Z hlediska energetické funkce se může jevit neuronové síti v takových případech výhodnější dané město neuvažovat a tím dosáhnout lepších energetických podmínek celého systému, což je ovšem opět odvislé od globálního nastavení.

Prvním možným řešením je úprava koeficientů, což je poměrně náročná činnost vyžadující množství experimentů. Můžeme snížit práh aktivity například s kombinací metody výběru unikátního neuronu z řádku a sloupce. Tímto krokem lze zajistit aktivaci všech neuronů, nicméně řešení již není ideální. Z tohoto titulu se zdá jako nejlepší varianta snížení hodnoty vzdálenosti inkriminovaného města vůči ostatním v množině všech předpočítaných vzdáleností (aplikace na počátku výpočtu inicializuje vektor, kde jsou po celou dobu uloženy vzájemné vzdálenosti mezi městy, abychom je nemuseli přepočítávat v průběhu). Bylo by ovšem nutné zachovat vzájemné poměrné pořadí vzdáleností seřazených dle velikosti. To je ve složitějších topologiích problematické. Přitom by město fyzicky zůstávalo na stejných souřadnicích. Variantou k tomuto postupu by mohlo být rozdělení množiny měst na několik disjunktních podmnožin, u kterých by se provedla výše zmíněná úprava vzdáleností a systémy by se řešily separátně. Až následně by došlo k syntéze podmnožin v jedno ucelené řešení.

V první části obrázku 3.43 vidíme ustálený příklad nepropojení měst vlivem jednoho příliš vzdáleného od množiny zbylých (mohou nastat i oscilující neustálené varianty) tak, jak je popsáno výše. Na druhé části obrázku je pak zelenou křivkou vykreslena správná trasa. Červenou šipkou je naznačena pozice reálného města tak, jak ji chápe systém po jeho interním přepočítání ve vektoru

vzájemných vzdáleností. Jedná se pouze o demonstrativní příklad, další řešení tohoto problému ponechávám v této práci otevřené.



Obrázek 3.43 Neúplné a úplné propojení měst s ukázkou, kde by bylo město z pohledu předpočítaných vzdáleností.

Využití neuronových sítí diskutovaných v mé bakalářské práci jako nástroje k řešení TSP je ryze experimentální. Dosahované výsledky založené na empirii nastavení koeficientů jsou náchylné na specifické tvary topologie měst a řešení je již pro řády desítek měst nespolehlivé, velmi nepřesné, s pomalou konvergencí. Hopfieldova neuronová síť řešící TSP existuje ovšem ve velké škále variant, které používají různě upravené energetické funkce a ty mohou dosahovat lepších výsledků.

3.3 Významné body implementace

Aplikace je implementována v objektovém prostředí *C++* frameworku *Qt v. 4.7* (Summerfield, 2011). Veškeré výpočty jsou spouštěny v samostatných vláknech, které zajišťují plynulý chod aplikace. V případě vícejádrových procesorů se může zátěž rozložit na několik jader.

Nyní postupně popíši nejdůležitější třídy využité v aplikaci. Podrobnější reference jsou v příloze, kde je vygenerovaná celá *doxygen* dokumentace.

Propojení signálů všech modulů s ovládacími prvky řídícího panelu a vykreslovacího plátna zastává třída `MainWindow`. Odtud jsou jednotlivým modulů předávány inicializační struktury pro zahájení výpočtů.

Hlavní výpočetní vlákno genetických algoritmů tvoří třída `Population`, která v sobě zapouzdřuje jedince `Genotype`, dále pak objekty sloužící ke křížení a mutacím. Všechny třídy křížení dědí stejné rozhraní z abstraktní třídy `Crossover`, která implementuje ryze virtuální metody. Ty jsou dále potomky přetěžovány.

Samostatná vlákna představují i chod neuronových sítí. Konkrétně Kohonenova neuronová síť je reprezentovaná třídou `KohonenNeuralNet` a Hopfieldova neuronová síť třídou `HopfieldNeuralNet`. Obě veřejně dědí a přetěžují metody třídy `QThread`. Obě obsahují i privátní instance třídy `Random`, která zprostředkovává kongruentní generátory náhodných čísel. Všechna vlákna se synchronizují také vzájemně a představují návrhový vzor *strategie* s potlačením analytické paralýzy (Virius, 2011).

Ukončení výpočtů a dynamiku chodu řeší třída `EndCondition`.

Kolizní úseky kódu jsou zabezpečeny pomocí zámků a volatilních proměnných. Zpřístupnění aktuálních řešení řídí časovače. Ty mají svou speciální třídu dědící rozhraní `QTimer`. Podle nich jsou emitovány signály s výsledky nezávislé na smyčce zpracování událostí.

Konkrétní výsledky jsou interpretované několika způsoby. Jednak jde o samotnou vizualizaci skutečného řešení na plátně, jakožto instance třídy `Scene`. Možnosti zoomu, vkládání prvků a dalších uživatelských možností řeší třída `View`. Do scény je možné importovat několik ukázkových map, ke kterým jsou generována reálná řešení s přepočty měřítek. Tuto možnost zajišťuje třída `mapLoader`, která funguje také jako samostatné vlákno, což je s výhodou využito především u velkých map.

Zobrazování informací v podobě textu zajišťuje singleton `StatGen`. Grafy jsou akcelerovány toolkitem `Qwt`, který využívá pro svou činnost třída `PlotGen`. Speciální grafický modul komunikuje s Hopfieldovou sítí. `VisualHopNet` třída zobrazuje aktuálně aktivované neurony v síti.

Několik tříd je také věnováno zobrazení nápovědy a informačních panelů.

Veškeré funkce, které byly implementovány za účelem řešení TSP jsou ve speciálním jmenném prostoru tak, aby mohly být lehce přenositelné do jiných projektů.

4 Závěr

V této bakalářské práci jsem formálně popsal problém obchodního cestujícího. Navrhl jsem a experimentálně ověřil několik variant řešení založených na evoluční teorii a neuronových sítích.

Kapitola 3.1 byla věnována genetickým algoritmům. Jsou zde popsaná jak teoretická východiska problému, tak experimenty provedené s navrženou aplikací. V nich jsem specifikoval pro zvolený interval propojovaných bodů výchozí nejvhodnější nastavení populace ve tvaru 65 10. Popsal jsem způsob zlepšování řešení v rozdílně velkých populacích pro různé typy křížení včetně úskalí, které se s některými z nich pojí. Učinil jsem rozbor mnou navržených heuristik, vytvářejících uspořádané posloupnosti měst podle vzájemných vzdáleností. Ověřil jsem, že takovéto heuristiky v kombinovaných metodách urychlují řešení. V práci jsou věnované kapitoly také technikám mutací a popisu jejich konkrétním vlivům na průběh řešení. Na základě dosažených zjištění jsou uvedeny experimenty s variabilní dynamikou mutací a variantami vzájemného spojování a kombinací mutačních operátorů. Nejlepších výsledků jsem dosahoval s inverzními dynamickými mutacemi, což potvrdilo teoreticky vyvozené předpoklady.

Kapitola 3.2 se zabývá neuronovými sítěmi. Byly použity dva typy. První, asociativní Kohonenova neuronová síť v základní variantě, která není pro tento typ řešení běžná. Druhá Hopfieldova, u níž byly popsány 3 přístupy řešení energetickou funkcí s pevnými koeficienty. U obou metod jsem provedl podrobné zhodnocení s návrhy na vylepšení zjištěných nedostatků. Hlavním nedostatkem Kohonenovy sítě byla nepřesná aproximace, kdy trasy nemusely z principu návrhu procházet vyznačenými městy. Hopfieldova síť reagovala špatně na specifické topologie měst a obtížně se nastavovaly koeficienty energetických funkcí. Konstatoval jsem, že v rámci mé aplikace dosahovaly genetické algoritmy znatelně lepších výsledků.

Vývoj projektu a jeho další směřování shledávám v několika bodech.

Do programu by bylo vhodné dodat další nové metody řešení TSP. Slabiny shledávám v samotné vizualizaci, která by mohla být propracovanější a především by mohla zobrazovat více informací. S tímto faktem už samotný návrh počítá a prostor pro rozšiřování této oblasti, stejně jako implementaci nových metod, je nachystán.

V aplikaci je možné pracovat i s mapami. V aktuální verzi jsou zpracovány pouze 3. Proto by bylo vhodné vytvořit univerzální rozhranní pro import map, například s možností porovnání informací o dosažených vzdálenostech s výsledky webových serverů zabývajících se plánovači tras.

Další ze směrů, kterým by se mohla aplikace ubírat, je propojení funkcionalit neuronových sítí a genetických algoritmů. Genetické algoritmy by mohly například optimalizovat topologii neuronové sítě a zvýšit její výkonnost.

5 Literatura

Algoritmus.net [online]. c2011 [cit. 27-3-2011]. Problém obchodního cestujícího. Dostupný z WWW: <http://www.algoritmy.net/article/5407/Obchodni-cestujici>.

BERG, J., BIOCH, J. C. *Constrained Optimalization with the Hopfield-Lagrage Model* [online]. Rotterdam: Erasmus University Rotterdam, 1993. 25 s. Oborová práce. Erasmus University Rotterdam. Dostupné z WWW: <http://publishing.eur.nl/ir/repub/asset/1462/eur-few-cs-93-10.pdf>.

FARAH AL-DULAIMI, B., ALI, A. H. *Enhanced Traveling Salesman Problem Solving by Genetic Algorithm Technique (TSPGA)*. Engeneering and Technology 38: World Academy of Science, 2008.

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

Ibm.com [online]. 2010 [cit. 2011-04-08]. IBM Watson. Dostupné z WWW: <http://www.ibm.com/innovation/us/watson/index.html>.

JIRSÍK, V. *Umělá inteligence* [online]. c2007, poslední revize 25.listopadu 2007 [cit. 27-3-2011]. Dostupné z WWW: <http://jaja.kn.vutbr.cz/~belovic/MUIN/01_UMI_definice.pdf>.

MAŃDZIUK, J. Solving the Travelling Salesman Problem with a Hopfield – type neural network. *Demonstration Mathematica,* 1996, 29, 1, s. 219-231. Dostupný také z WWW: <http://www.mini.pw.edu.pl/~mandziuk/PRACE/TSP_DM.pdf>. ISSN 0420-1213.

POTVIN, J. Y. The Traveling Salesman Problem: A Neural Network Perspective. *INFORMS Journal on Computin,* 1993, 5, 4, s. 328-348. Dostupný také z WWW: <http://ce.sharif.ir/courses/8485/2/ce667/resources/root/Seminar_no_7/paper_potvin_nn_tsp.pdf>. ISSN 1091-9856.

PRATA, S. Mistrovství v C++. 3. vyd. Praha: Computer Press, 2007. 1120 s. ISBN 978-80-251-1749-1.

SUMMERFIELD, M. *Advanced Qt Programming*. 1. vyd. Boston: Pearson Education, 2011. 554 s. ISBN 978-0-321-63590-7.

ŠALÁT, T. a kol. *Malá encyklopédie matematiky*. 1. vyd. Bratislava: Vydavatelstvo Obzor, 1981. 814 s.

ŠÍMA, J., NERUDA, R. *Teoretické otázky neuronových sítí*. Praha: Vydavatelství MFF UK, 1996. *ISBN* 80-85863-18-9.

Tsp.gatech.edu [online]. c2007, poslední revize 1.1.2007 [cit. 28-3-2011]. Optimal tours. Dostupné z WWW: <http://www.tsp.gatech.edu/optimal/index.html>.

VESELOVSKÝ, M. *Avari.cz/uir* [online]. [200-] [cit. 28-3-2010]. Neuronové sítě. Dostupné z WWW: <http://avari.cz/uir/index.php?pg=vsechno>.

VIRIUS, M. *1001 Tipů a triků pro C++*. 1. vyd. Praha: Computer Press, 2011. 450s. ISBN 978-80-251-2941-8.

VOLNÁ, E. *Neuronové sítě 1*. Ostrava: Vydavatelství Ostravské univerzity, 2002. Spojitá Hopfieldova síť, s. 85.

YU, X., MITSUO, G. *Introduction to Evolution Algorithms*. 1. vyd. New York: Springer, 2010. 418 s. ISBN 978-1-84996-128-8.

ZBOŘIL, F. V. *Základy umělé intelligence* [online]. c2004, poslední revise 3.2006 [cit. 27-3-2011]. Dostupné z WWW: <https://www.fit.vutbr.cz/study/courses/IZU/private/oporaizu-esf-4.pdf>.

Přílohy

A. Obsah přiloženého CD

Uvádím adresářovou strukturu s popisem

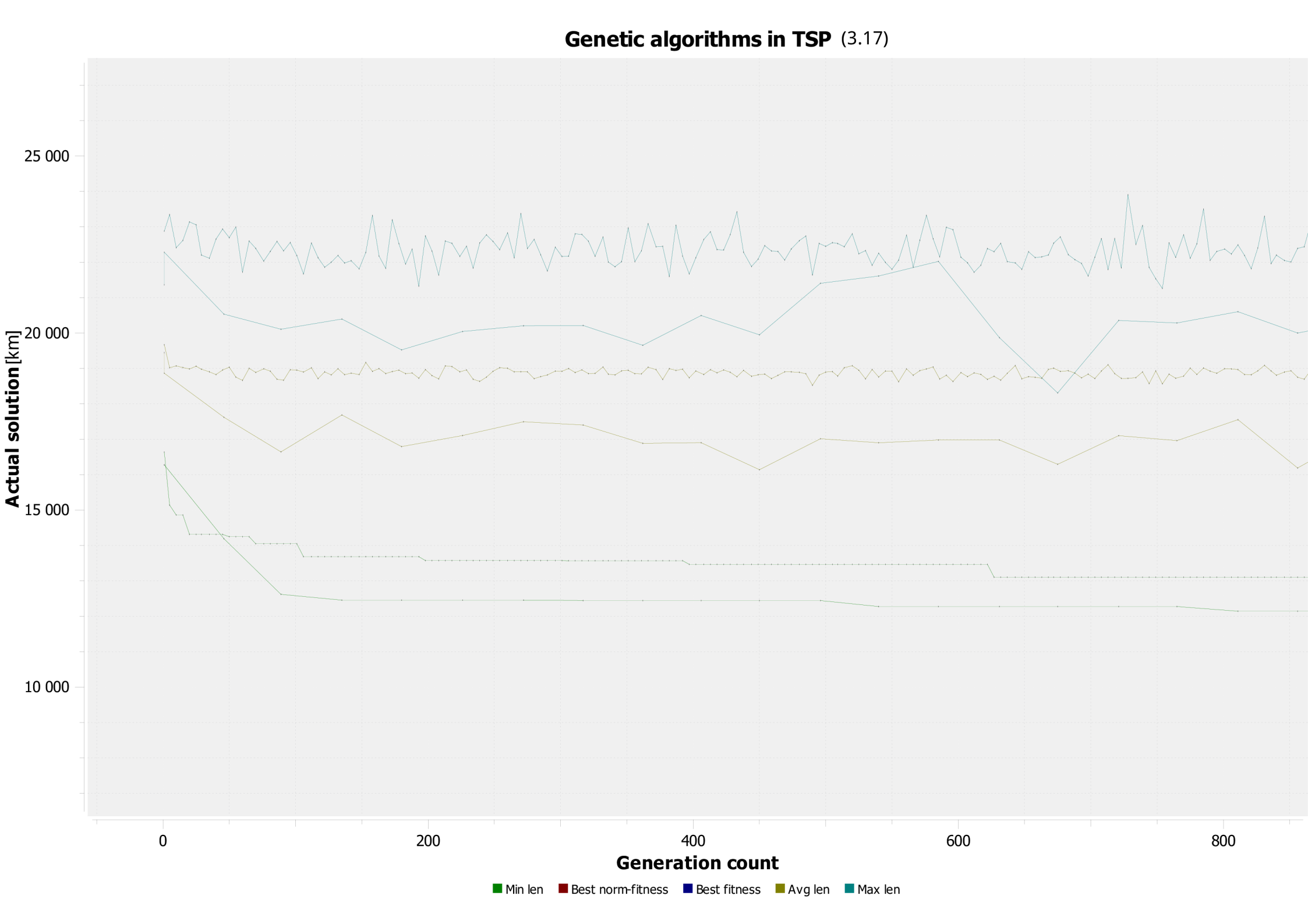
`doc`	Textový výstup práce
`*.pdf`	Texty
`*.doc`	
`graphSupplement`	Příloha s grafy ve vektorové podobě
`doxygen`	Aplikace pro generování dokumentace
`html`	Výsledek generovní dokumentace pomocí make
`index.html`	Vstupní bod dokumentace
`src`	Soubor zdrojových kódů
`declarations`	Deklarace
`definitions`	Definice
`make.bat`	Makefile aplikace
`src_qwt`	Zdrojové kódy Qwt grafů
`doxylog.txt`	Konfigurační soubor dokumentace
`README.txt`	Readme text s návodem k ovládání makefilu

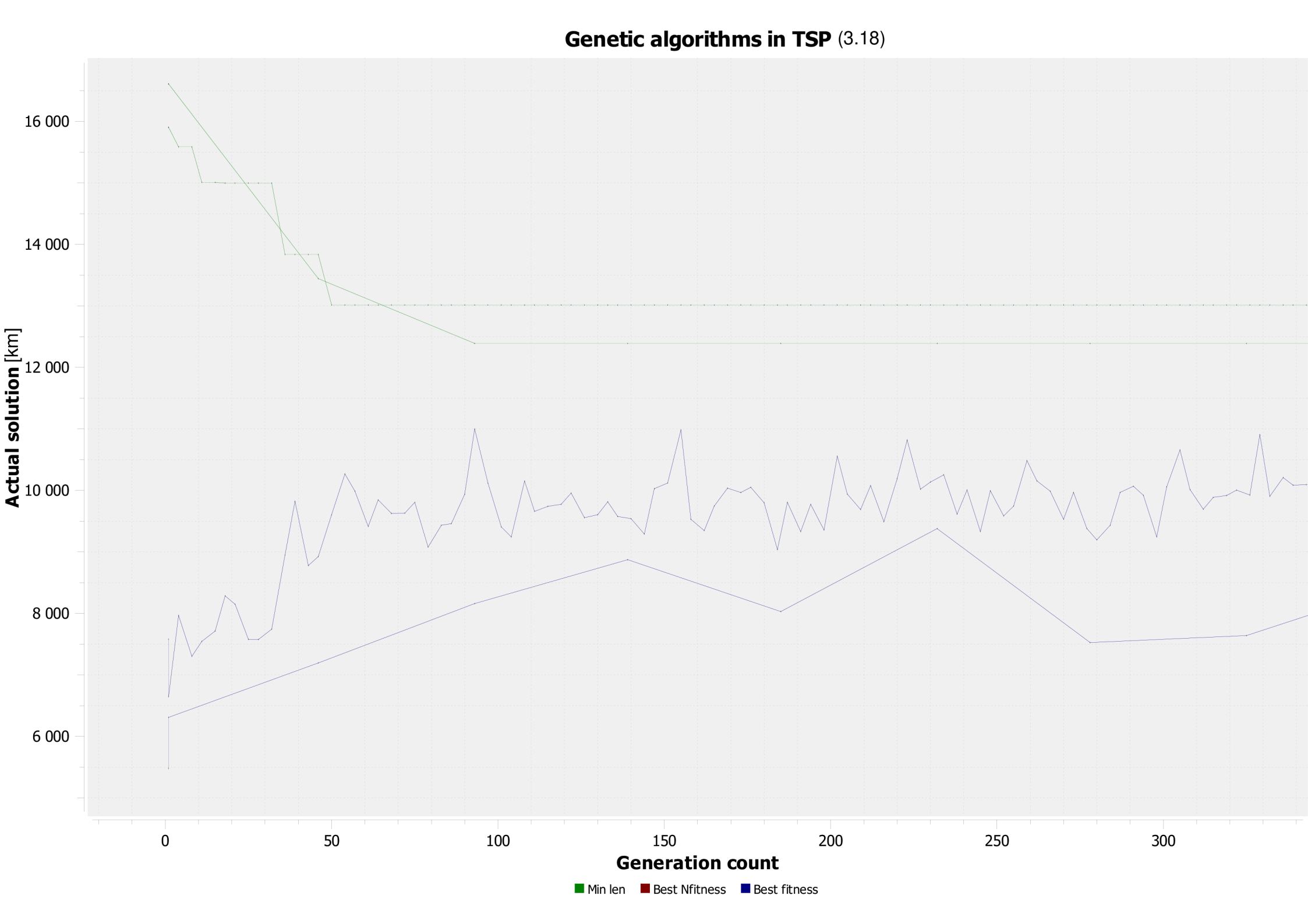
B. Seznam zkratek

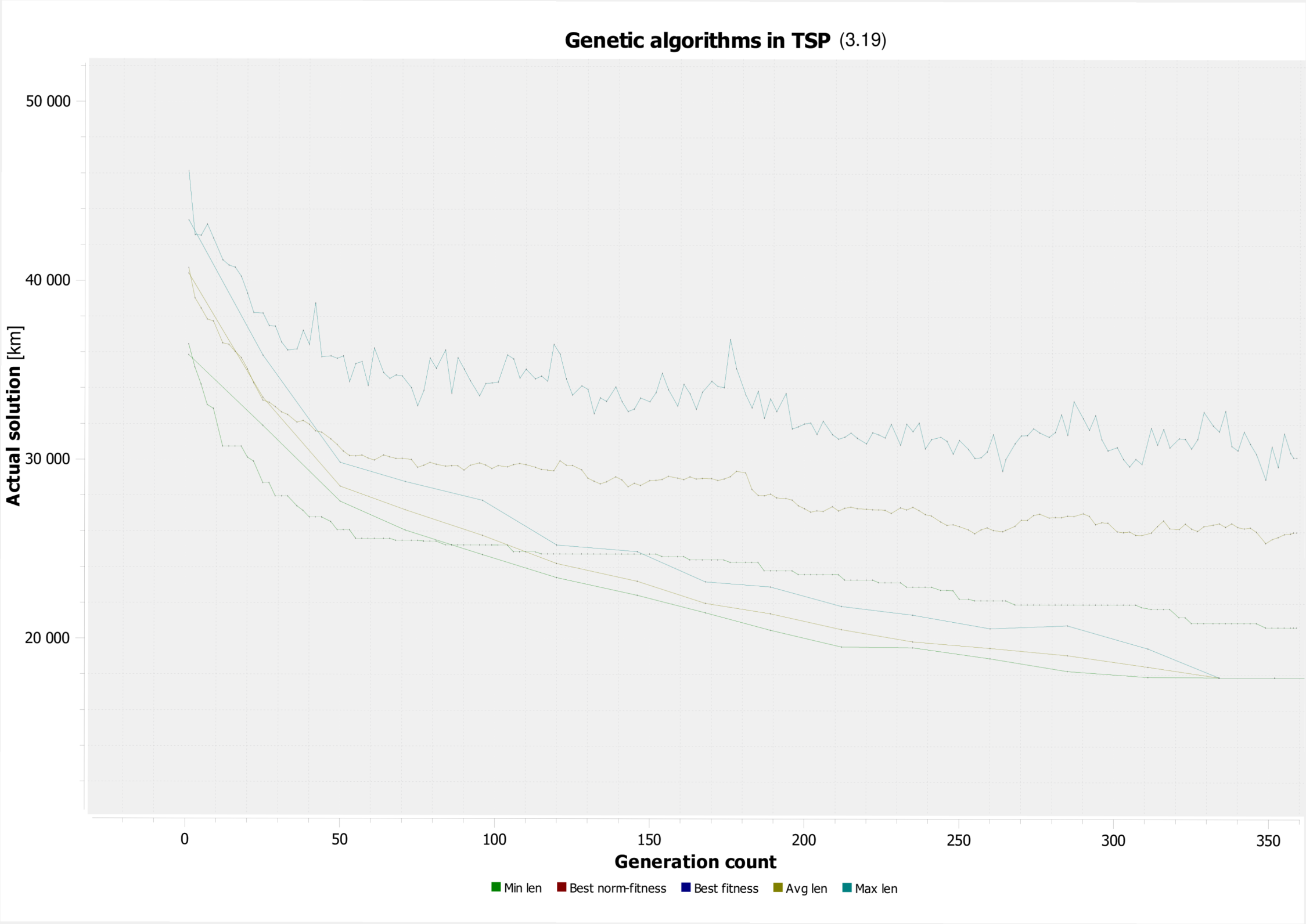
AI	artificial intelligence
TSP	travelling salesman problem
NP	non-polynomial
GA	genetické algoritmy
SMX	specialized mutation operator (uváděný též jako SOM)
SOM	switch ones mutation
SPM	switch parts mutation
ILM	inverse length station
ERX	edge recombination crossover
OX	order crossover
PMX	partially matched crossover
PCH	pair challenge
OLC	optimal length crossover

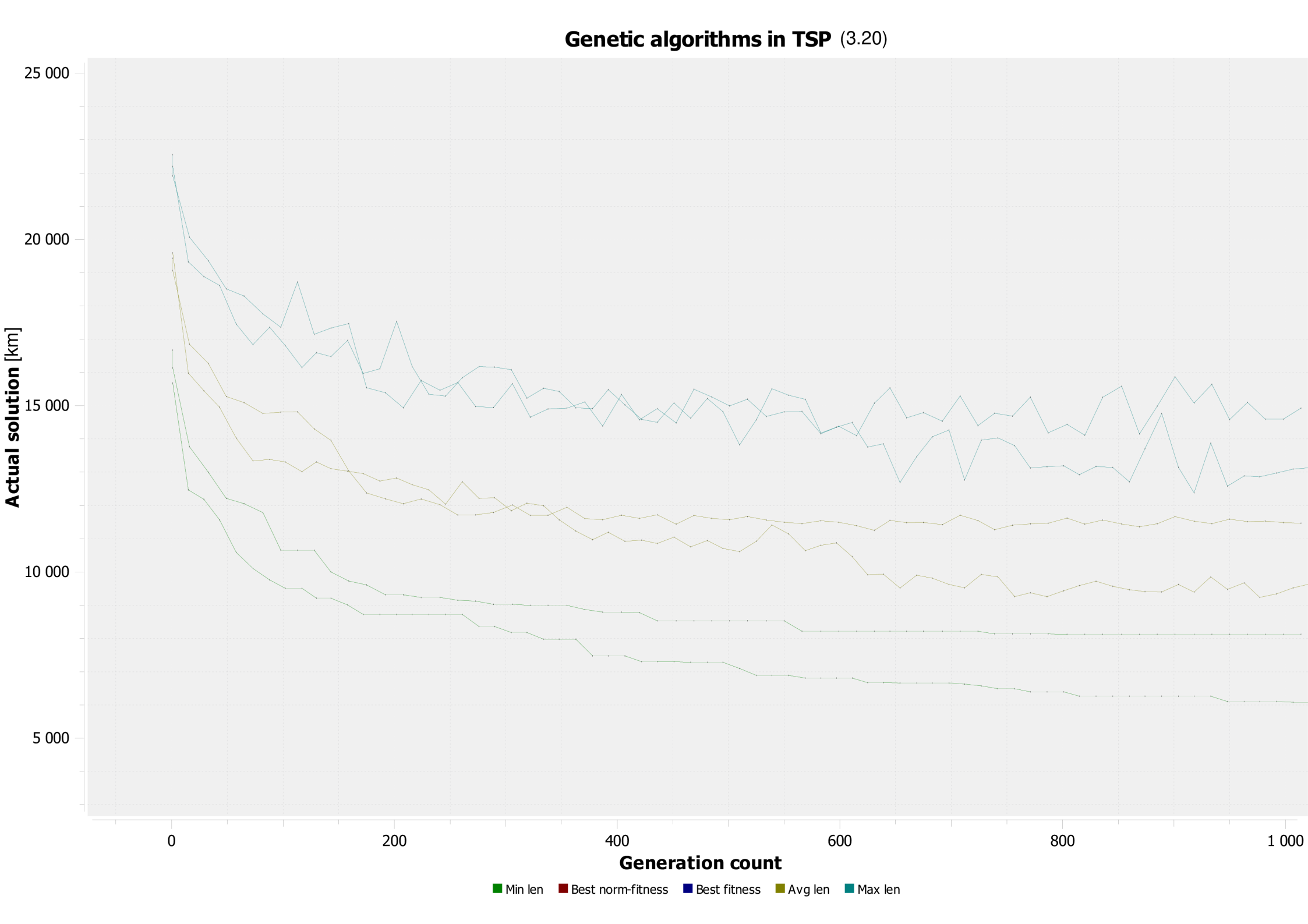
C. Grafy k experimentům ve vektorové podobě

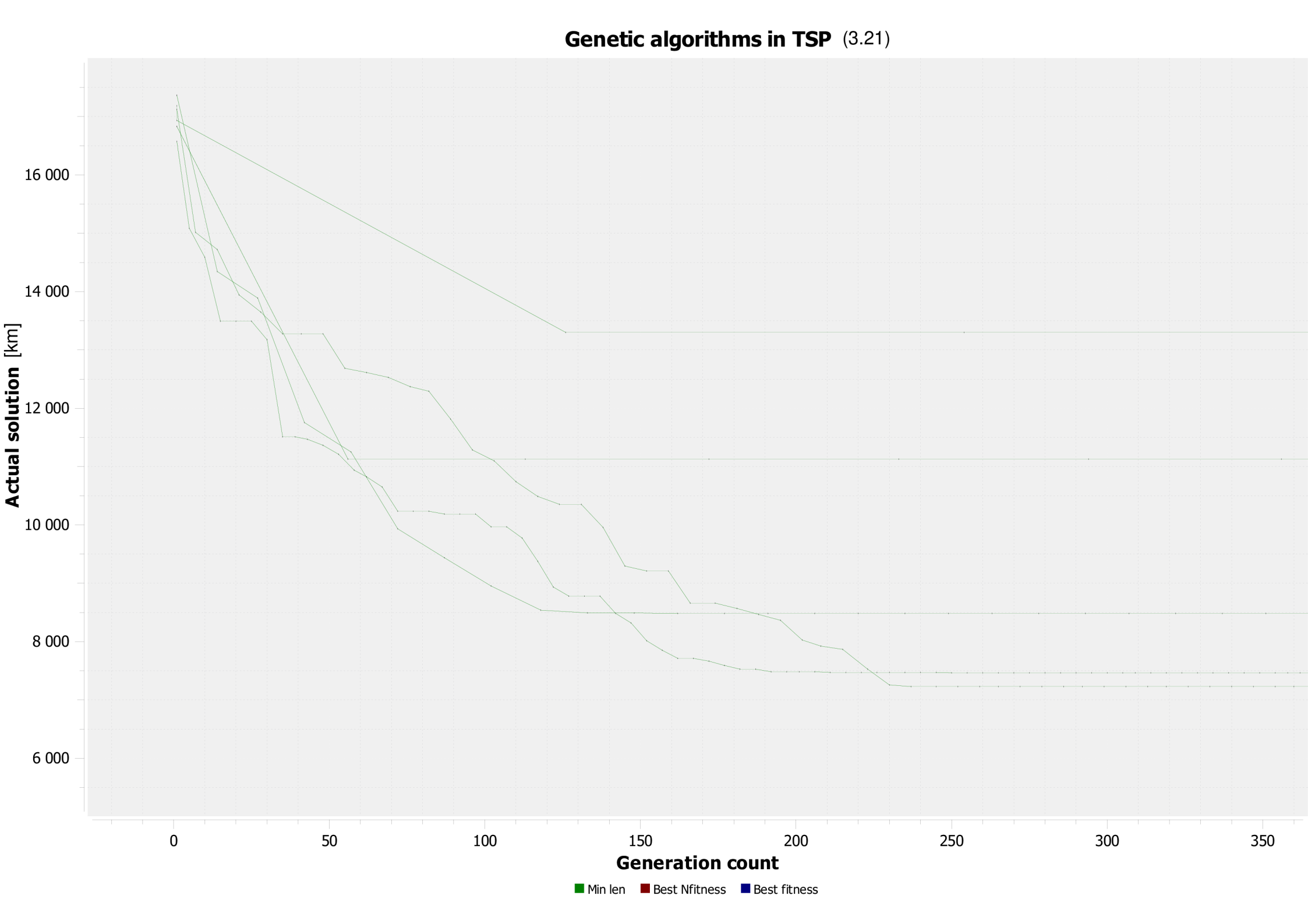
(samostatně ve složce `doc\graphSupplement`)

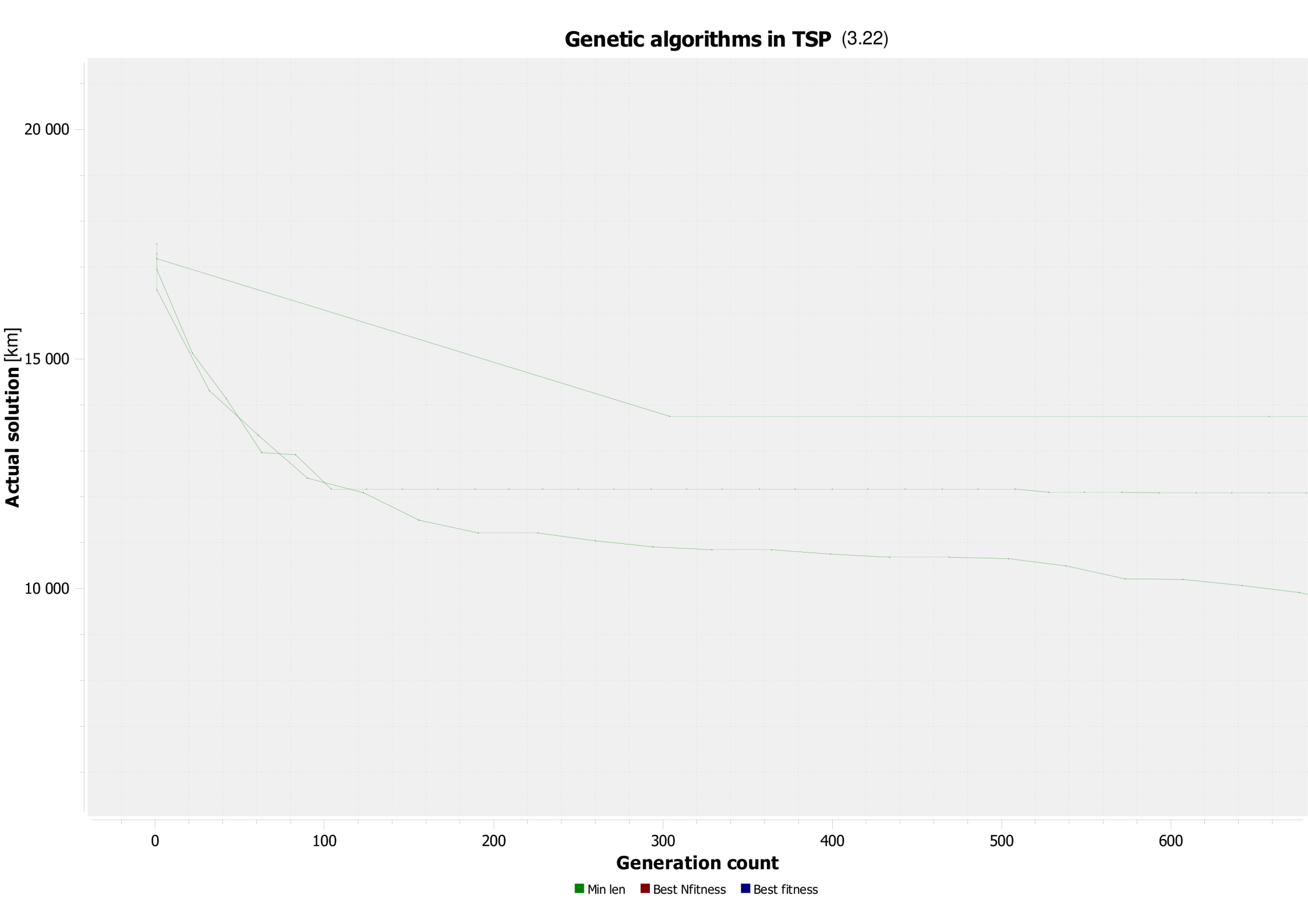

Genetic algorithms in TSP (3.17)
25 000
20 000
15 000
10 000
Actual solution [km]
0
200
400
600
800
Generation count
Min len
Best norm-fitness
Best fitness
Avg len
Max len

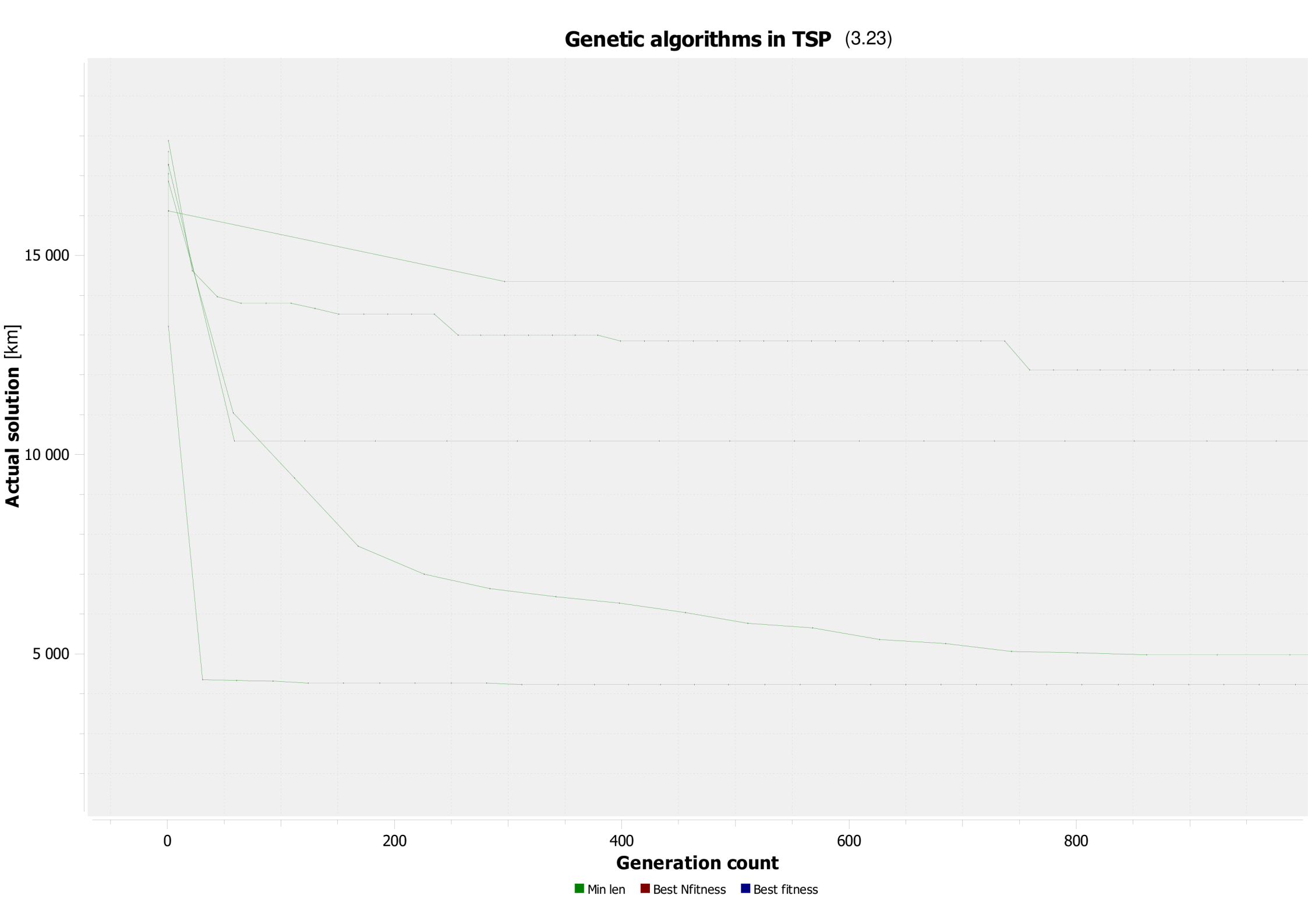

Genetic algorithms in TSP (3.18)
16 000
14 000
12 000
10 000
8 000
6 000
Actual solution [km]
0
50
100
150
200
250
300
Generation count
Min len
Best Nfitness
Best fitness

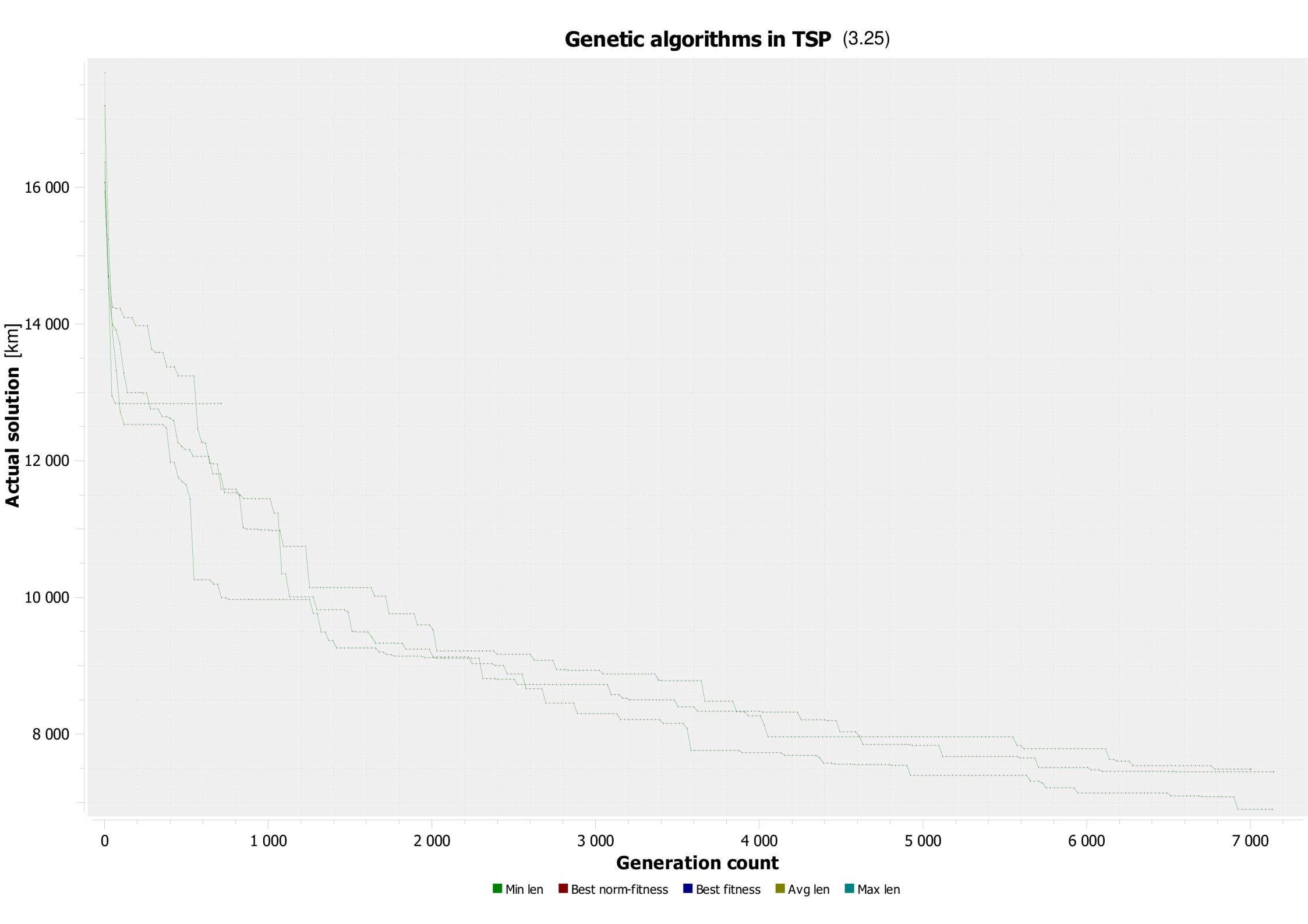

Genetic algorithms in TSP (3.19)
50 000
40 000
30 000
20 000
Actual solution [km]
0
50
100
150
200
250
300
350
Generation count
Min len
Best norm-fitness
Best fitness
Avg len
Max len

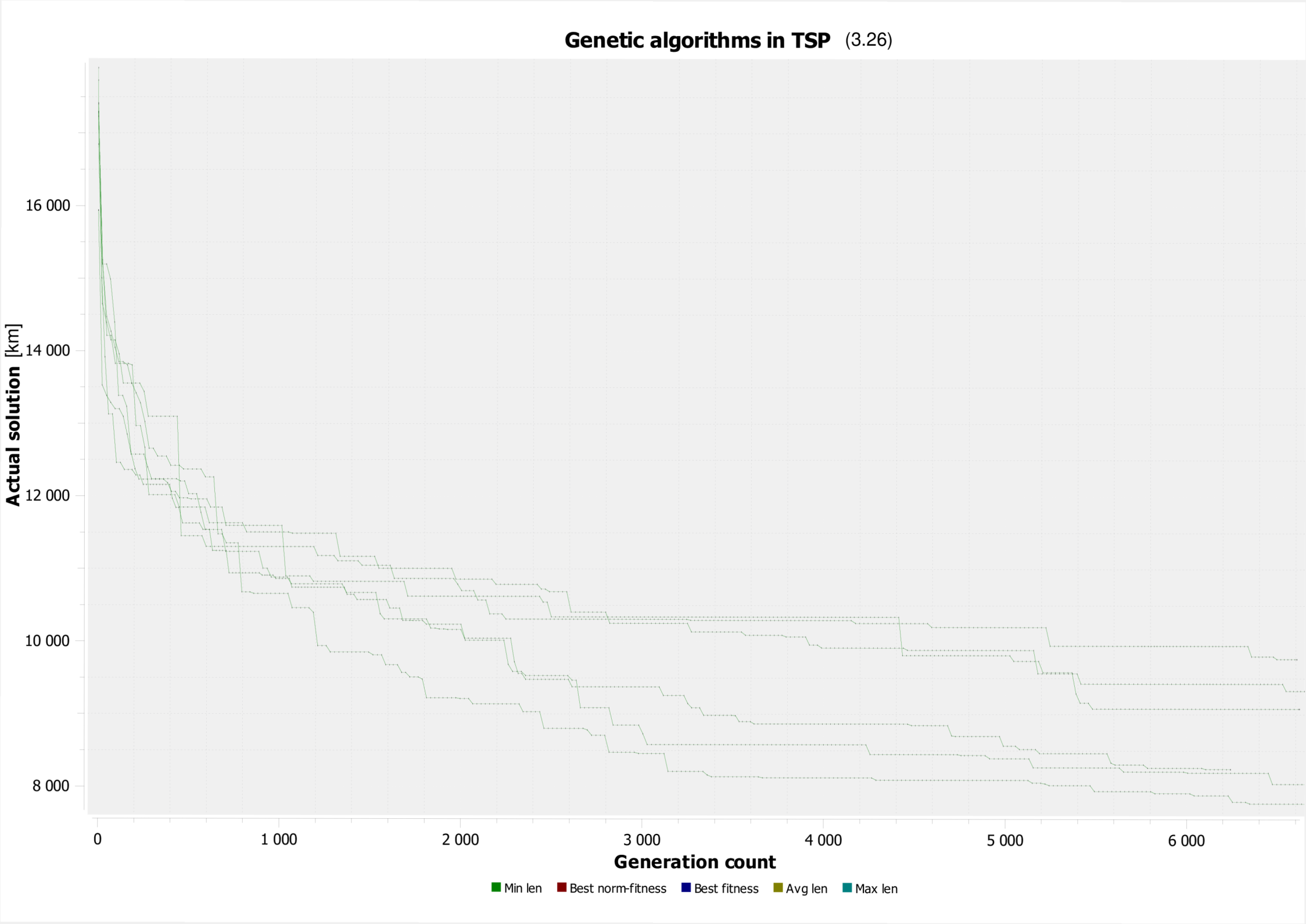

Genetic algorithms in TSP (3.20)
25 000
20 000
15 000
10 000
5 000
Actual solution [km]
0
200
400
600
800
1 000
Generation count
Min len
Best norm-fitness
Best fitness
Avg len
Max len

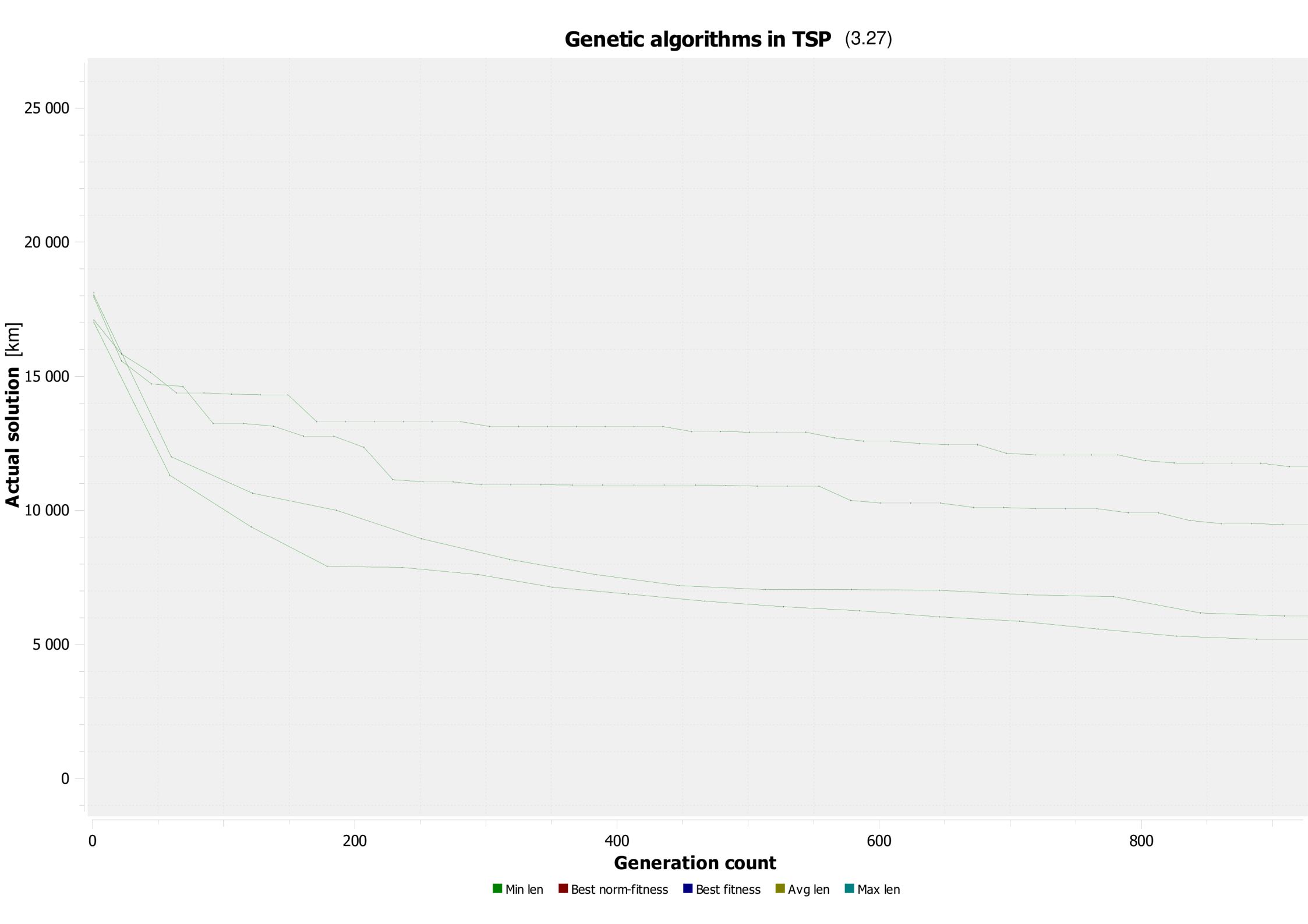

Genetic algorithms in TSP (3.21)
Actual solution [km]
16 000
14 000
12 000
10 000
8 000
6 000
0
50
100
150
200
250
300
350
Generation count
Min len
Best Nfitness
Best fitness

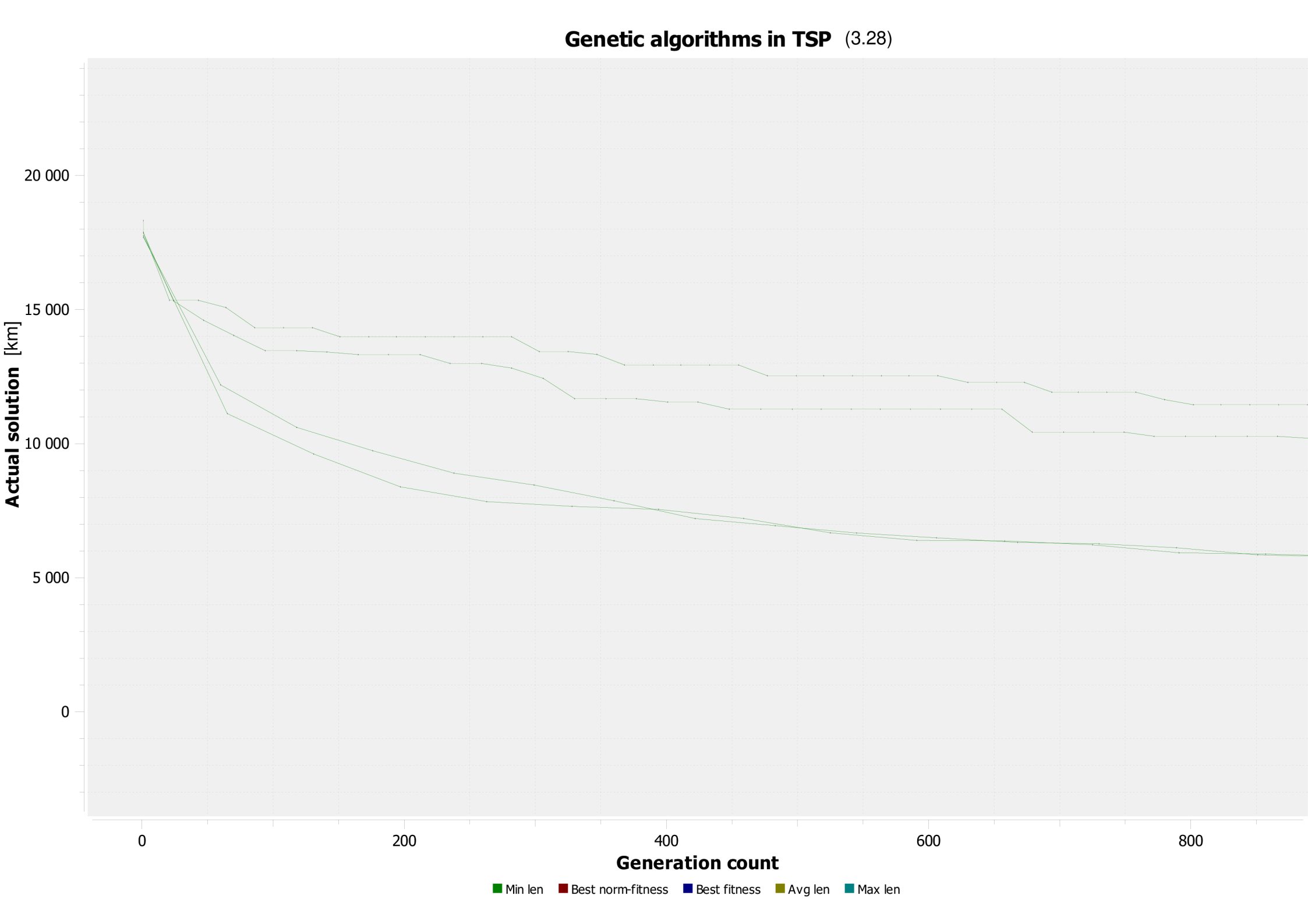

Genetic algorithms in TSP (3.22)
20 000
15 000
10 000
Actual solution [km]
0
100
200
300
400
500
600
Generation count
Min len
Best Nfitness
Best fitness

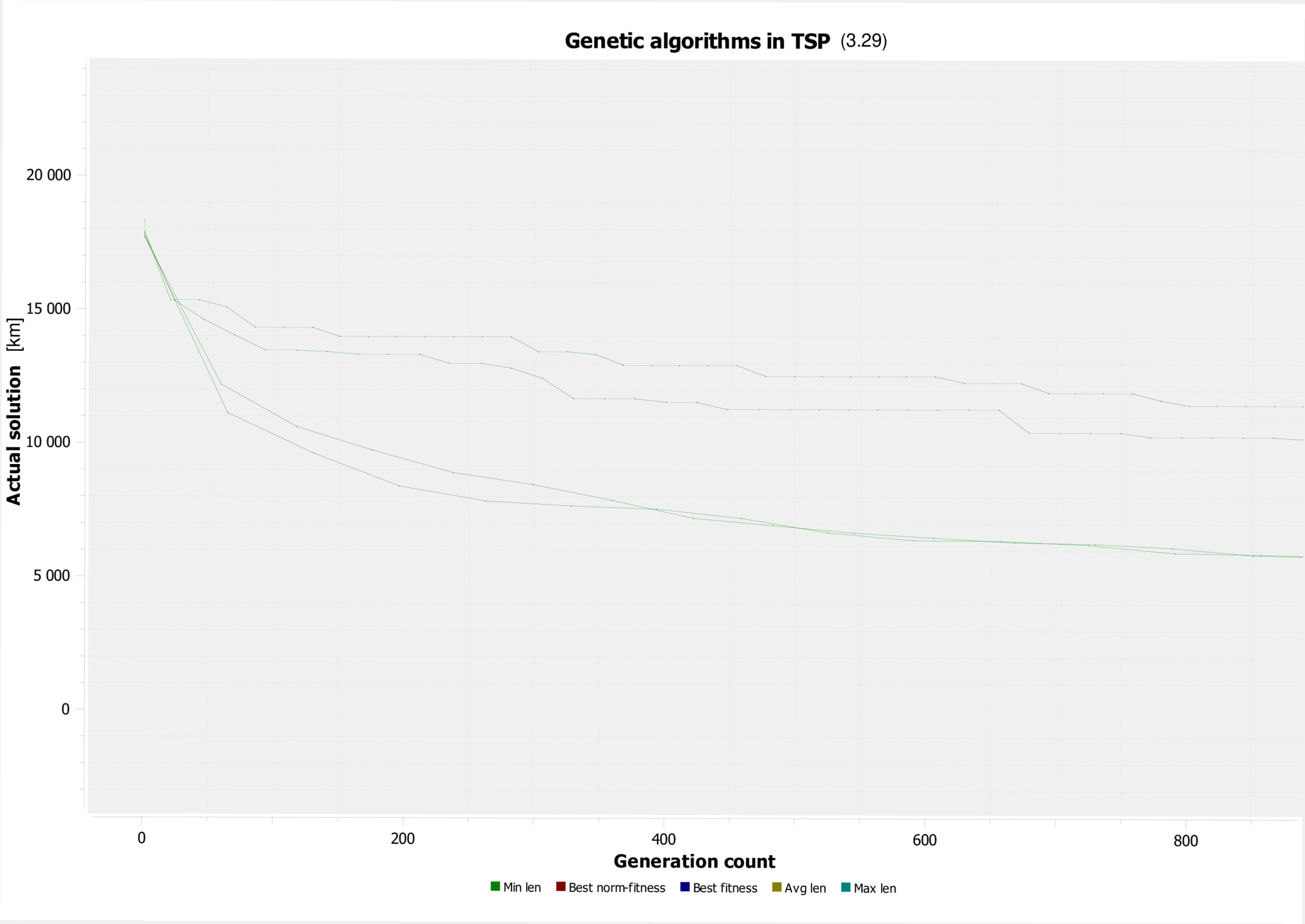

Genetic algorithms in TSP (3.23)
Actual solution [km]
15 000
10 000
5 000
0
200
400
600
800
Generation count
Min len
Best Nfitness
Best fitness

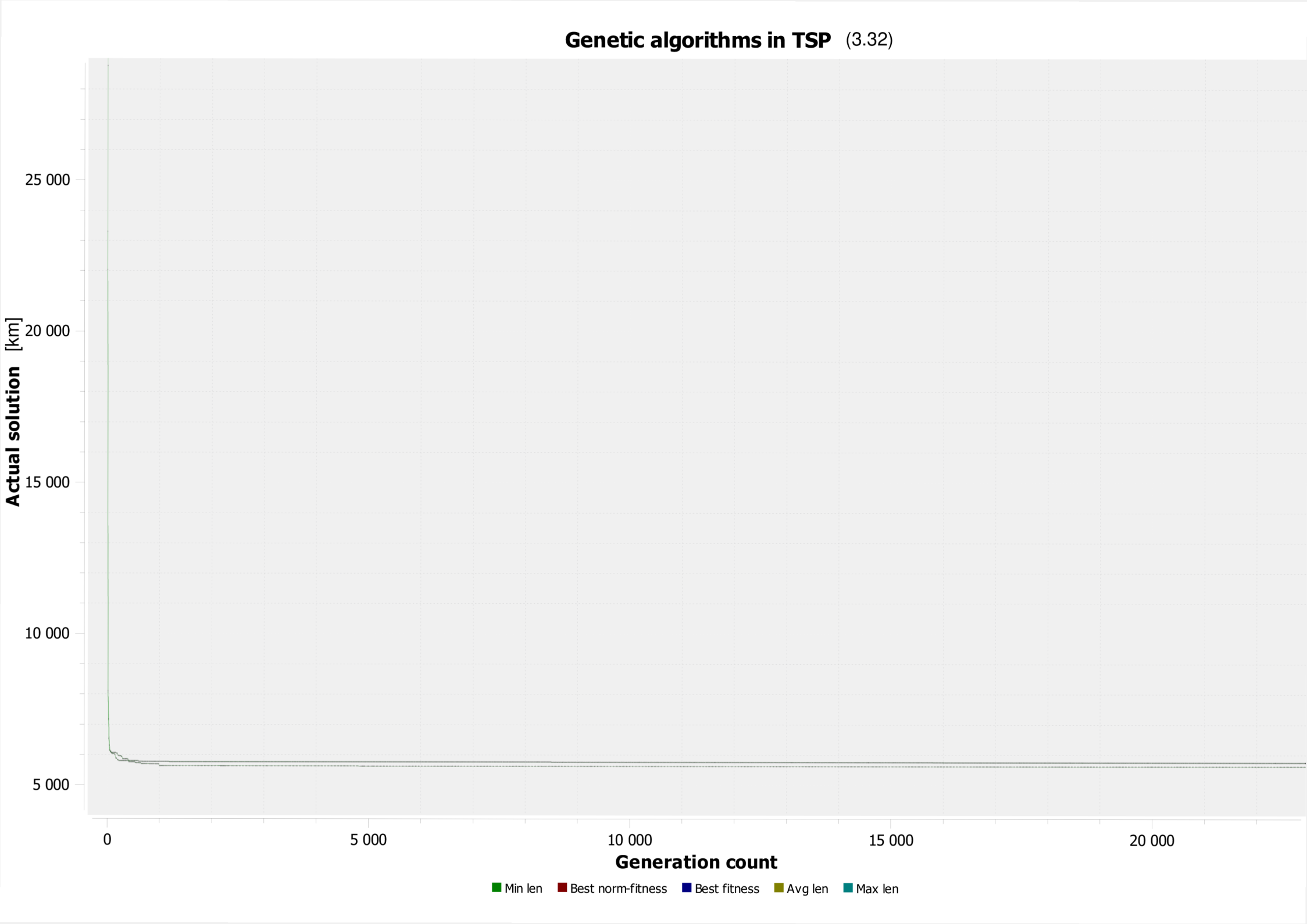

Genetic algorithms in TSP (3.25)
Actual solution [km]
16 000
14 000
12 000
10 000
8 000
0
1 000
2 000
3 000
4 000
5 000
6 000
7 000
Generation count
Min len
Best norm-fitness
Best fitness
Avg len
Max len


Genetic algorithms in TSP (3.26)
Actual solution [km]
16 000
14 000
12 000
10 000
8 000
0
1 000
2 000
3 000
4 000
5 000
6 000
Generation count
Min len
Best norm-fitness
Best fitness
Avg len
Max len


Genetic algorithms in TSP (3.27)
25 000
20 000
15 000
10 000
5 000
0
Actual solution [km]
0
200
400
600
800
Generation count
Min len
Best norm-fitness
Best fitness
Avg len
Max len


Genetic algorithms in TSP (3.28)
Actual solution [km]
20 000
15 000
10 000
5 000
0
0
200
400
600
800
Generation count
Min len
Best norm-fitness
Best fitness
Avg len
Max len


Genetic algorithms in TSP (3.29)
20 000
15 000
10 000
5 000
0
Actual solution [km]
0
200
400
600
800
Generation count
Min len
Best norm-fitness
Best fitness
Avg len
Max len


Genetic algorithms in TSP (3.32)
Actual solution [km]
25 000
20 000
15 000
10 000
5 000
0
5 000
10 000
15 000
20 000
Generation count
Min len
Best norm-fitness
Best fitness
Avg len
Max len