VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

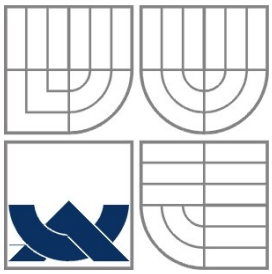
ZJIŠŤOVÁNÍ TOPOLOGIE BEZDRÁTOVÉ SENZOROVÉ SÍTĚ GENETICKÝMI ALGORITMY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

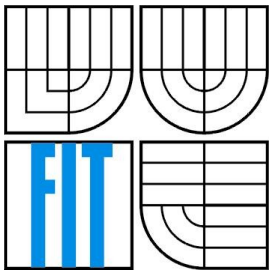
AUTOR PRÁCE
AUTHOR

ŠTĚPÁN DALECKÝ

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ZJIŠŤOVÁNÍ TOPOLOGIE BEZDRÁTOVÉ SENZOROVÉ SÍTĚ GENETICKÝMI ALGORITMY

DISCOVERY OF WIRELESS SENSOR NETWORK TOPOLOGY USING GENETIC ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ŠTĚPÁN DALECKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. František V. Zbořil, CSc.

BRNO 2012

Abstrakt

Cílem této bakalářské práce je navrhnout genetický algoritmus, který bude schopen určit polohu senzorů bezdrátové senzorové sítě na základě síly signálu mezi jednotlivými senzory.

Nejprve se práce zabývá teorií genetických algoritmů a okrajově popisem bezdrátové senzorové sítě. Následně je na základě této teorie navržen genetický algoritmus, který slouží k zjištění topologie bezdrátové senzorové sítě. Práce také popisuje důležité rysy implementace tohoto algoritmu. Závěrem jsou zhodnoceny dosažené výsledky.

Abstract

The thesis deals with a design of the genetic algorithm that is able to discover the wireless sensor network topology using signal strength among particular sensors.

At first, the thesis describes the theory of genetic algorithm and wireless sensor network. Subsequently, on the basis of this theory, the genetic algorithm serving for the wireless sensor network topology discovery has been designed. The thesis also describes important features of the algorithm implementation. In conclusion, the outcomes have been reviewed.

Klíčová slova

umělá inteligence, optimalizace, genetický algoritmus, chromozóm, fitness funkce, genetické operátory, křížení, mutace, bezdrátové sítě, senzorové sítě, topologie sítě, graf

Keywords

artificial intelligence, optimization, genetic algorithm, chromosome, fitness function, genetic operators, crossover, mutation, wireless network, sensor network, network topology, graph

Citace

Štěpán Dalecký: Zjišťování topologie bezdrátové senzorové sítě genetickými algoritmy, bakalářská práce, Brno, FIT VUT v Brně, 2012

Zjišťování topologie bezdrátové senzorové sítě genetickými algoritmy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. Ing. Františka V. Zbořila, CSc.
Další informace mi poskytl Ing. Jan Horáček.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

…..……
Štěpán Dalecký
14. 5. 2012

Poděkování

Rád bych poděkoval doc. Ing. Františku V. Zbořilovi, CSc. za konzultace, vedení této práce a cenné informace o genetických algoritmech. Také bych chtěl poděkovat Ing. Janu Horáčkovi za praktické rady z oblasti bezdrátových senzorových sítí a za občasné konzultace. Oběma pak za pomoc při zpracování příspěvku do soutěže EEICT.

Velmi rád bych poděkoval i rodičům za podporu ve studiu.

Obsah

1 Úvod

Mezi člověkem a počítačem je velký rozdíl především v tom, jak si každý z nich dokáže poradit s různými problémy. I když mají oba dva řešit totéž, každý z nich to bude dělat jiným způsobem. Zatímco počítač zvládne velice rychle zpracovat velké množství informací pomocí elementárních operací, člověk zpracuje menší množství informací, zato ale komplexně. Člověk si také nedokáže zapamatovat mnoho třeba i nesmyslných a nesouvisejících informací, jelikož k zapamatování potřebuje právě souvislosti. Pro počítač není problém do paměti uložit a poté z ní přečíst téměř jakákoliv data.

Počítače vymyslel člověk a tím už jsou jejich schopnosti limitovány. Člověk řešení problému musí zvládnout sám nebo musí být schopen počítači sdělit, jakým způsobem má postupovat, aby dosáhl požadovaného výsledku. Každý postup je výhodný za jiných podmínek. V této práci se zaměřím především na to, jak k řešení problému využít počítač právě takovým způsobem, aby jeho výsledky vypadaly, jako by je vytvořil přímo člověk. Tuto myšlenku lze považovat za jeden ze stavebních kamenů umělé inteligence. Od umělé inteligence je pak již jen krůček ke genetickým algoritmům.

Cílem práce je navrhnout a implementovat genetický algoritmus, který určí topologii bezdrátové senzorové sítě na základě síly signálu mezi jednotlivými senzory. Bezdrátovou senzorovou síť si můžeme představit jako několik zařízení, která jsou rozmístěna například v lese a sledují pohyby osob. Zařízení mohou vzájemně komunikovat. Bližší zařízení komunikují přímo, vzdálenější zařízení pak komunikují pomocí těch bližších. Jedním z hlavních důvodů, proč zjišťujeme topologii sítě, je využití této znalosti pro efektivní směrování dat mezi zařízeními.

Nejprve se budu zabývat umělou inteligencí a optimalizací a poté vysvětlím, co je to bezdrátová senzorová síť. Největší část textu však věnuji teorii genetických algoritmů a návrhu genetického algoritmu pro zjišťování topologie sítě. Také popíši důležité rysy návrhu a implementace algoritmu. V závěru práce potom zhodnotím dosažené výsledky.

2 Umělá inteligence a evoluční algoritmy

V této kapitole popíši, k čemu se hodí umělá inteligence, co je to optimalizace a jaká je základní myšlenka evolučních algoritmů.

2.1 Umělá inteligence

Umělá inteligence je obor informatiky, který se zabývá vytvářením systémů chovajících se inteligentně. Posouzení toho, zda se daný systém chová inteligentně, se většinou dělá tak, že se porovnají výsledky, kterých při stejné činnosti dosáhne systém a člověk. Systém, který danou činnost provádí obdobně jako člověk, lze považovat za inteligentní. Spousta inspirace a myšlenek se čerpá z biologie.

Co je tedy vhodné řešit pomocí umělé inteligence a co ne? Úlohy vzhledem k jejich řešení můžeme rozdělit do následujících kategorií:

- postup řešení daného problému je znám a je vhodný a výhodný,
- postup řešení daného problému je znám, ale není vhodný – například by byl časově tak náročný, že už by byl výsledek k ničemu (např. při řízení letu rakety) nebo by výpočet trval déle, než si můžeme dovolit čekat (např. týdny, měsíce, roky, ale i řádově déle),
- postup řešení daného problému není znám, ale známe alespoň to, jak má výsledek vypadat nebo jaké má splňovat podmínky.

Pro řešení posledních dvou kategorií problémů se hodí umělá inteligence. Na první kategorii bychom ji použít mohli, ale nebyl by to nejvhodnější způsob. Obzvláště výhodná je umělá inteligence tam, kde algoritmus řešení úlohy není vůbec znám.

2.1.1 Optimalizace

Optimalizace [5] je matematická úloha, která má za cíl najít v množině přípustných řešení takové řešení, pro něhož účelová funkce dosahuje svého maxima, případně minima. Množina přípustných řešení je taková množina, která obsahuje všechna možná řešení dané úlohy. Účelová funkce je taková funkce, která každému řešení z množiny přípustných řešení přiřadí hodnotu, která udává kvalitu daného řešení.

Pro optimalizaci hlavně složitějších účelových funkcí se často užívá umělé inteligence, jelikož konvenční matematické metody (např. matematické programování, simulované žíhání) nedosahují tak dobrých výsledků.

Matematicky lze optimalizaci definovat takto: hledáme takové $x_0 \in X$, aby $\forall x \in X : f(x_0) \geq f(x)$ pro maximalizaci a nebo $\forall x \in X : f(x_0) \leq f(x)$ pro minimalizaci.

X	Množina přípustných řešení
$f: X \rightarrow R$	Účelová funkce

2.1.2 Spojení s biologií

Biologie je velikou inspirací v mnoha vědních oborech, a to i technických. Například ptáci byli jistě vzorem prvních letadel. Tento trend neminul ani počítače a umělou inteligenci.

Umělá inteligence dokonce přebírá celou řadu různých postupů a myšlenek založených na biologii a na poznání toho, jak příroda funguje. Příkladem mohou být neuronové sítě, mravenčí kolonie, kolonie ptáků a nebo evoluční algoritmy. V dalších kapitolách se zaměřím především na evoluční algoritmy, zejména pak na genetické algoritmy.

2.2 Evoluční algoritmy

Základní myšlenka evolučních algoritmů [4] vychází z Darwinovy teorie evoluce a přirozeného výběru. V přírodě mezi sebou jednotliví jedinci populace soupeří o teritorium, o jídlo, jednoduše o přežití. Také musí zvítězit nad jinými, aby se mohli rozmnožovat. Většinou přežívají ti silnější (lépe adaptovaní na aktuální podmínky) a slabší jedinci postupně vymírají. Zdatnější jedinci tudíž budou mít více potomků než ti slabší. Za několik generací bude druh čím dál tím lépe přizpůsoben životu v daném prostředí. Podobného principu využívají evoluční algoritmy. Jedná se v podstatě o simulace evolučního vývoje na počítači.

V evolučních algoritmech představuje každý jedinec jedno potenciální řešení úlohy. Stejně jako v přírodě pracujeme zároveň s více jedinci – s jejich populací. Princip hledání řešení spočívá v simulaci vývoje této populace a v hledání takového jedince, který má nejlepší vlastnosti – je nejlépe adaptovaný.

Kvalita každého jedince (jeho schopnost přežít a mít potomky) je ohodnocena fitness funkcí. Řešení úlohy představuje jedinec, jenž má po vývoji několika generací nejlepší ohodnocení fitness funkcí.

2.2.1 Genetický algoritmus

Genetický algoritmus ke kombinaci dvou jedinců využívá genetické operátory. Základními genetickými operátory jsou křížení a mutace. Křížení kombinuje dva nebo více jedinců takovým způsobem, aby byla jistá pravděpodobnost, že vznikne potomek, který je lepší než jeho rodiče. Mutace je náhodná změna jednoho jedince, která zabrání algoritmu, aby uvázl v lokálním minimu.

2.2.2 Genetické programování

V genetickém programování je jedinec reprezentován datovou strukturou strom, která představuje určitý algoritmus. Postupným vývojem tohoto stromu vzniká algoritmus, který má na dané vstupy požadované výstupy.

3 Genetický algoritmus

Za otce genetických algoritmů [1] považujeme Johna Hollanda z univerzity v Michiganu, který svoji práci započal v šedesátých letech minulého století. V roce 1975 publikoval knihu s názvem „Adaption in Natural and Artificial System“, kde poprvé popsal genetický algoritmus. Kniha měla dva základní cíle: prohloubit znalosti o evolučním vývoji a na základě těchto znalostí navrhnout podobný umělý systém. Budu se dále věnovat spíše stránce technické, která se týká umělé inteligence, než biologickým procesům.

Z pohledu umělé inteligence genetický algoritmus patří do skupiny algoritmů evolučních. Je tedy založen na Darwinově teorii evoluce a teorii přirozeného výběru. Tyto principy jsou abstrahovány a ve zjednodušené podobě použity jako výpočetní model. Z matematického hlediska se algoritmus snaží o optimalizaci fitness funkce.

Genetický algoritmus [3] je heuristická metoda pro řešení úlohy. Úkolem je najít optimální řešení ze všech možných řešení daného problému. Všechna přípustná řešení se nazývají prohledávaný prostor. Pokud je prohledávaný prostor malý, je možné prozkoumat všechna možná řešení, ale s tím, jak prostor roste, se prozkoumávání všech řešení stává obtížnější a pomalejší. Proto potřebujeme nějakou metodu, která nebude prozkoumávat celý prostor a dostatečně rychle najde optimální nebo suboptimální (téměř optimální) řešení. Jednou z takových metod je genetický algoritmus.

Genetický algoritmus pracuje s populací řešení, čímž se liší od běžných optimalizačních technik, protože ty využívají jen jedno jediné řešení. Každé řešení úlohy je reprezentováno jedním chromozómem. Jednou z prvních věcí, kterou musíme při použití genetického algoritmu navrhnout, je reprezentace jednotlivých chromozómů – způsob kódování řešení. Dále musíme určit genetické operátory, které provádí křížení a mutaci chromozómů. Genetickým operátorům a návrhu způsobu kódování řešení se vyplatí věnovat značnou pozornost, jelikož genetický algoritmus je na tom velmi závislý a není vždy snadné najít vhodné kódování a operátory reflektující řešený problém.

Obdobně jako si příroda vybírá nejlepší jedince, tak i my musíme být schopni posoudit, který chromozóm je lepší a který horší. K tomu slouží fitness funkce, která každému chromozómu v populaci přiřadí číslo odpovídající jeho kvalitě. Cílem genetického algoritmu je maximalizace této funkce, a proto potřebujeme fitness funkci zvolit tak, aby mělo optimální řešení maximální ohodnocení. Pokud je fitness funkcí například chybová funkce, pak musíme její hodnotu buď invertovat, nebo upravit algoritmus takovým způsobem, aby za lepší považoval chromozómy s menším ohodnocením. Potom se jedná o minimalizaci této fitness funkce a optimální řešení se nachází v jejím minimu.

Jakmile určíme kódování řešení, genetické operátory a fitness funkci, můžeme nechat populaci vyvíjet. To se děje pořád podle stejného algoritmu. Nejdříve vytvoříme počáteční populaci, která by měla být co nejrozmanitější, a tudíž obsahovat co nejvíce z prohledávaného prostoru. Obvykle se počáteční populace vytváří náhodně.

Genetický algoritmus pracuje následujícím způsobem [1] [2]:

1. **Inicializace** – vytvoření chromozómů počáteční populace (obvykle náhodně).
2. **Ohodnocení** každého chromozómu populace pomocí fitness funkce.
3. **Reprodukce** – opakuj pro vytvoření nové populace:
    - **Selekce** – vyber dva rodičovské chromozómy z populace v závislosti na jejich ohodnocení.
    - **Křížení** – s pravděpodobností křížení aplikuj genetický operátor křížení na rodiče a tím vytvoř potomka. Pokud křížení neproběhne, potomek je kopií rodiče.
    - **Mutace** – s pravděpodobností mutace aplikuj genetický operátor mutace a tím proveď náhodnou změnu potomka.
    - **Vložení** chromozómu do nové populace.
4. **Nahrazení** staré populace novou.
5. Pokud není splněna **ukončovací podmínka**, přejdi na bod 2.
6. **Výsledkem** je chromozóm s nejlepším ohodnocením.

Abychom věděli, kdy výše uvedený algoritmus můžeme ukončit a kdy můžeme prohlásit dosud nejlepší nalezený chromozóm za výstup algoritmu, a tudíž za řešení úlohy, je nutné ještě stanovit ukončovací podmínku. Určení této podmínky není snadné, jelikož nesmí algoritmus zastavit předčasně, protože by nenašel dostatečně dobré řešení, ale ani ho nesmí nechat běžet zbytečně dlouho, abychom neplýtvali časem a výpočetními prostředky.

Při návrhu genetického algoritmu tedy musíme definovat:

- kódování řešení do chromozómů,
- fitness funkci pro hodnocení chromozómů,
- genetické operátory (křížení, mutace),
- různé parametry (např. velikost populace, pravděpodobnost křížení a mutace),
- způsob vytvoření počáteční populace,
- ukončovací podmínku.

Většinou máme mnoho možností, jak genetický algoritmus navrhnout. Správné a nebo špatné rozhodnutí může mít zásadní vliv na kvalitu řešení a rychlost algoritmu. Při rozhodování nám může pomoci odborná literatura, praxe a v neposlední řadě také experimentování.

Výhody a nevýhody genetických algoritmů oproti řešení problémů konvenčním způsobem nastíním v závěru této kapitoly.

3.1 Chromozóm

Chromozóm je složen z řetězce genů, z nichž každý vyjadřuje část informace o řešení. Alela představuje hodnotu, kterou gen může nabývat. Způsobu, jakým jsou data uložena, se říká genotyp. Význam genotypu se nazývá fenotyp.

Například můžeme mít chromozóm, který bude reprezentovat barvu. Standardně se barva na počítači ukládá ve ve formátu RGB. Tudíž chromozóm by byl složen z řetězce 3 genů, a to R, G a B. Každá ze tří barevných složek může nabývat hodnoty 0 až 255, což znamená, že odpovídající gen má 256 alel. Chromozóm s genotypem (255, 165, 0) má fenotyp „oranžová“.

Každý chromozóm reprezentuje jedno řešení v genetickém algoritmu. V jeho genetické informaci musí být toto řešení nějakým způsobem kódováno. Existuje mnoho možností, jak řešení reprezentovat chromozómem, a každá možnost je vhodná pro jiný typ úlohy. Obvyklé typy kódování jsou binární kódování (případně oktalové nebo hexadecimální), kódování pomocí permutací, kódování pomocí výčtu hodnot nebo stromem. Kódování je silně závislé na typu úlohy a mělo by umožnit reprezentovat všechna možná řešení úlohy.

3.1.1 Binární kódování

Binární kódování se používá nejčastěji, ale není vždy vhodné. Dobře dokáže reprezentovat například celá čísla nebo pravdivostní hodnoty. Při využití binárního kódování pro kódování celých čísel je však třeba dávat pozor na reprezentaci záporných hodnot a zvolit vhodný počet bitů, jelikož délka kódovaného řetězce má vliv na přesnost.

Tabulka 1: Binární kódování

Chromozóm	0 1 1 1 0 1 0 0 1 1 1 1 1 0 0 1

3.1.2 Kódování permutacemi

Při využití kódování pomocí permutací je chromozóm reprezentován sadou čísel. Jednotlivé chromozómy se od sebe liší pouze uspořádáním. Toto kódování je vhodné pro problémy, kde hledáme optimální pořadí, například problém obchodního cestujícího.

Tabulka 2: Kódování permutacemi

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

3.1.3 Kódování výčtem hodnot

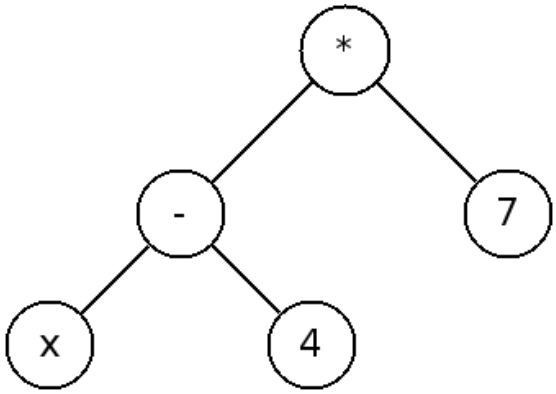
V případě, že použijeme kódování výčtem hodnot, každý gen v chromozómu může nabývat jedné hodnoty z předem určeného výčtu hodnot. Hodnoty výčtu mohou být libovolné hodnoty vztahující se k řešenému problému.

Tabulka 3: Kódování výčtem hodnot

Chromozóm 1	(nahoru), (dolu), (doprava), (doleva)
Chromozóm 2	(modrá), (bílá), (žlutá), (červená)

3.1.4 Kódování stromem

Reprezentace chromozómu jako stromu lze využít například pro evoluční vývoj funkčního předpisu.



Obrázek 1: Kódování stromem

Obrázek 1 představuje funkci $f(x)=(x-4)*7$.

3.2 Populace

Populace je soubor chromozómů. Způsob vytvoření počáteční populace a velikost populace má vliv na genetický algoritmus.

Vytvoření počáteční populace se obvykle dělá náhodně s ohledem na to, aby byla co nejrozmanitější a aby pokud možno pokrývala celý prohledávaný prostor. Měla by tedy obsahovat všechny možné alely všech genů. Můžeme také použít heuristiku při jejím vytváření a nebo využít již nějakého známého a dostatečně dobrého řešení, čímž urychlíme hledání optimálního řešení. Musíme však dbát na to, aby byla populace stále dost rozmanitá, jinak algoritmus prozkoumá jen malou část prohledávaného prostoru a nenalezne globální optimum.

Velikost populace je jeden ze základních faktorů, který má vliv na rychlost algoritmu a úspěšnost nalezení řešení. Malá populace nemusí být dostatečně rozmanitá, a tudíž řešení nemusí být kvalitní. Zbytečně velká populace jen zabírá paměť a prodlužuje výpočetní čas. Je proto důležité zvolit vhodný kompromis. Obecně platí, že čím je větší populace, tím jsou lepší výsledky a delší výpočetní čas.

3.3 Fitness funkce

Fitness funkce slouží k ohodnocení kvality chromozómů. Každému chromozómu přiřazuje reálné číslo, které určuje, jak dobré řešení daný chromozóm představuje. Vypovídá i o tom, jak moc se řešení představované chromozómem blíží řešení optimálnímu.

Fitness funkce je v průběhu algoritmu vyhodnocovaná velice často, a proto se vyplatí optimalizovat její výpočet, jelikož je obvykle časově náročný, a tudíž se stává úzkým hrdlem algoritmu.

3.4 Reprodukce

Reprodukce je nejvýznamnější částí genetického algoritmu. Je to proces, při kterém křížením rodičů vznikají noví a pokud možno lépe adaptovaní potomci. Reprodukce se skládá ze 4 kroků:

- **Selekce** – výběr rodičů,
- **Křížení** – vznik potomků kombinací rodičů,
- **Mutace** – náhodná změna potomka,
- **Náhrada** – odstranění rodičů z generace a vložení potomků.

3.4.1 Selekce

Pro vývoj populace je třeba určit, z kterých chromozómů a s jakou pravděpodobností se stanou rodiče. Tato volba by měla probíhat náhodně, ale s ohledem na hodnotu fitness funkce – čím má chromozóm lepší ohodnocení, tím má větší pravděpodobnost, že bude vybrán.

Význam, který při výběru chromozómů přikládáme jejich ohodnocení, se nazývá selekční tlak. Čím je vyšší selekční tlak, tím pravděpodobněji vybereme chromozóm s vyšším ohodnocením. Selekční tlak je častým parametrem selekčních metod a má vliv na rychlost algoritmu a na kvalitu nalezeného řešení. Při nízkém selekčním tlaku může výpočet trvat zbytečně dlouho, naopak při vysokém selekčním tlaku můžeme dávat přednost výhradně kvalitním jedincům. Mohli bychom tak přijít o cenný genetický materiál a algoritmus by mohl uváznout v lokálním extrému.

Výběr elity

Výběr elity je způsob omezení výběru rodičů. Rodiče nebudeme vybírat z celé populace, ale populaci seřadíme podle hodnoty fitness funkce a rodiče vybereme jen z nějakého procenta nejlépe ohodnocených chromozómů. Výhoda metody tkví ve výběru lepších rodičů, čímž zvýšíme pravděpodobnost vzniku kvalitních potomků. Nevýhodou může být to, že zanedbáním horší části populace přijdeme o genetický materiál a populace může rychleji degenerovat.

Souboj

Náhodně vybereme dva chromozómy a podle fitness funkce vezmeme lepší z nich. Stejným způsobem zvolíme ještě jeden chromozóm a oba vybrané pak použijeme jako rodiče.

Ruleta

Výběr rodiče pomocí rulety probíhá obdobně, jako by byly chromozómy výseče v kruhu a vhozená kulička jednu výseč vybrala. Rozdíl je v tom, že velikost výseče příslušející danému chromozómu je úměrná hodnotě jeho fitness funkce. Pravděpodobnost výběru i-tého chromozómu p_i se vypočte podle vzorce (1), kde f_i je hodnota fitness funkce pro i-tý chromozóm.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{1}$$

Tento způsob výběru je nevýhodný, když jeden nebo jen velmi málo chromozómů má vysokou hodnotu fitness funkce a ostatní malou. Výběr rodičů se tak omezuje jen na velmi malou skupinku nejlepších a populace tím může degenerovat. V porovnání s výběrem soubojem nebo náhodným výběrem je tato metoda výpočetně náročnější.

Náhodný výběr

Z populace vybereme rodiče zcela náhodně bez ohledu na hodnotu fitness funkce.

3.4.2 Křížení

Křížení dvou chromozómů je operace, která z rodičů vybraných selekcí vytvoří potomky. Operátor křížení by měl být vytvořen takovým způsobem, aby bylo pravděpodobné, že křížením vznikne potomek lepší, než jsou jeho rodiče.

Existuje několik druhů křížení. Nejjednodušší z nich je jednobodové křížení. Způsob, jak kombinovat rodiče, abychom dostali potomky, je výrazně závislý na řešeném problému a musí brát ohled na specifika úlohy.

U křížení je významná i pravděpodobnost, s jakou se tento operátor aplikuje. Příliš nízká hodnota může genetický algoritmus svést téměř k náhodnému prohledávání a naopak vysoká hodnota pravděpodobnosti může snižovat kvalitu populace tím, že se bude neustále většina chromozómů měnit. Obvyklé hodnoty pravděpodobnosti křížení jsou 70 % až 90 %. Pokud křížení neaplikujeme, potomkem se stává kopie rodiče.

Jednobodové křížení

Jednou z nejpoužívanějších metod křížení je jednobodové křížení. Požaduje dva rodiče a vytvoří z nich dva potomky. Proces křížení probíhá takovým způsobem, že náhodně zvolíme bod v řetězci genů. První potomek v genetické informaci nese část řetězce prvního rodiče od začátku do zvoleného bodu a druhá část je doplněna z druhého rodiče od náhodně zvoleného bodu do konce. Druhý potomek vznikne stejným způsobem, jen je vyměněno pořadí rodičů.

Tabulka 4: Jednobodové křížení

Rodič 1	**0 1 0 0** 1 1 0 0
Rodič 2	1 1 1 0 **0 1 1 1**
Potomek 1	**0 1 0 0 0 1 1 1**
Potomek 2	1 1 1 0 1 1 0 0

Dvoubodové křížení

Dvoubodové křížení probíhá podobně jako jednobodové. Rozdíl je v tom, že náhodně zvolené body jsou dva. Potomci vznikají z rodičů kopií a poté výměnou řetězce genů mezi těmito náhodně zvolenými body.

Tabulka 5: Dvoubodové křížení

Rodič 1	**0 1** 0 0 1 **1 0 0**
Rodič 2	1 1 **1 0 0** 1 1 1
Potomek 1	**0 1 1 0 0 1 0 0**
Potomek 2	1 1 0 0 1 1 1 1

Uniformní křížení

Při uniformním křížení se potomci vytváří z rodičů na základě náhodně vygenerované masky. Pokud je bit v masce roven 0, pak se do potomka zkopíruje gen z prvního rodiče, pokud je roven 1, pak se zkopíruje gen z druhého rodiče. Pro druhého potomka se použije stejný postup, ale invertovaná maska.

Tabulka 6: Uniformní křížení

Rodič 1	0 1 0 0 1 1 0 0
Rodič 2	1 1 1 0 0 1 1 1
Maska	**0 1 0 1 0 0 1 1**
Potomek 1	0 1 0 0 1 1 1 1
Potomek 2	1 1 1 0 0 1 0 0

Aritmetické křížení

Aritmetické křížení je vhodné tehdy, když máme uloženu informaci jako reálné číslo. Potomky P_1 a P_2 rodičů A a B získáme podle vzorce (2). Parametr r je pro každé křížení náhodně zvolené číslo z intervalu <0;1>.

$$\begin{aligned} P_1 &= rA + (1-r)B \\ P_2 &= (1-r)A + rB \end{aligned} \tag{2}$$

3.4.3 Mutace

Mutace představuje náhodnou změnu v chromozómu. Křížení pouze kombinuje stávající genetický materiál, zatímco mutace nám přináší materiál nový, čímž zabraňuje degenerování populace. Mutace dokáže zabráni algoritmu v uváznutí v lokálním extrému fitness funkce.

Podobně jako křížení, tak i mutace se řídí pravděpodobností aplikace. V případě, že bychom zvolili pravděpodobnost mutace příliš velkou, tak se z genetického algoritmu stane slepé a náhodné prohledávání. Při malé pravděpodobnosti mutace naopak hrozí, že algoritmus uvázne v lokálním extrému fitness funkce a nenajde globální optimum. Obvyklá pravděpodobnost mutace je v jednotkách procent.

Převrácení hodnoty bitu

Při užití převrácení hodnoty bitu náhodně vygenerujeme masku. Pokud odpovídající bit masky je 1, hodnotu bitu chromozómu invertujeme.

Tabulka 7: Mutace převrácením hodnoty bitu

Před mutací	0 **1** 0 **0** 1 1 1 **0**
Maska	0 **1** 0 **1** 0 0 0 **1**
Po Mutaci	0 **0** 0 **1** 1 1 1 **1**

Záměna bitů

Mutace záměnou bitů probíhá tak, že z chromozómu vybereme náhodně dvě pozice a že bity, které jim odpovídají, se mezi sebou vymění.

Tabulka 8: Mutace záměnou bitů

Před mutací	0 **1** 0 0 1 1 1 **0**
Po mutaci	0 **0** 0 0 1 1 1 **1**

3.5 Nahrazení

Nahrazení je poslední částí reprodukce. Definuje způsob, jakým budeme nahrazovat v populaci staré chromozómy novými. Existuje několik běžně používaných modelů:

- **Generační** – celá původní populace je nahrazena populací novou,
- **Inkrementační** – jediný chromozóm populace je nahrazen novým,
- **Překrytí generací** – v populaci se nahradí pouze některé chromozómy novými.

3.5.1 Elitářství

Elitářství se využívá v případě, že použijeme generační model, ve kterém se stará populace zcela nahradí populací novou. Pokud ve staré populaci bylo nějaké dobré řešení, tak o něj můžeme přijít, protože se řešení nemusí vůbec stát rodičem a nebo se aplikováním genetických operátorů

zhorší. Tuto situaci řeší právě elitářství. Nejlepší chromozóm (případně několik nejlepších chromozómů) beze změny vložíme do nové populace, což nám zaručí, že nikdy nepřijdeme o nejlepší dosud nalezené řešení.

3.6 Ukončovací podmínka

Ukončovací podmínka slouží k zastavení genetického algoritmu. Chromozóm s nejvyšší hodnotou fitness funkce se ukončení stává nejlepším nalezeným řešením úlohy. Existuje několik metod, jak algoritmus ukončit:

- **Počet generací** – určíme maximální počet generací, po který necháme populaci vyvíjet. Poté bez ohledu na kvalitu nalezeného řešení algoritmus ukončíme.
- **Časový limit** – jakmile algoritmus překročí nastavenou dobu, je ukončen bez ohledu na kvalitu řešení.
- **Nejlepší chromozóm** – skončíme, když ohodnocení nejlepšího chromozómu bude lepší než stanovená hranice.
- **Nalezení řešení** – v případě, že nalezneme požadované řešení (známe hodnotu fitness funkce tohoto řešení), algoritmus ukončíme.
- **Průměrná hodnota fitness** – průběžně počítáme průměrnou hodnotu fitness funkce celé populace, jakmile tato hodnota dosáhne požadované hranice, algoritmus ukončíme.
- **Nezlepšování nejlepšího chromozómu** – vývoj populace ukončíme v případě, že se po stanovený počet generací nepodařilo najít lepší řešení.
- **Nezlepšování průměru fitness funkce** – skončíme v případě, že se po daný počet generací nezlepšilo průměrné ohodnocení populace (průměrná hodnota fitness funkce).

Obdobně jako u jiných částí genetického algoritmu je možné zvolit ukončovací podmínku na míru konkrétní aplikaci a tím algoritmus zefektivnit. Běžně se využívá kombinace podmínek a algoritmus je ukončen, jakmile alespoň jedna podmínka platí.

3.7 Srovnání s konvenčními metodami optimalizace

Výstupem genetického algoritmu ve většině případů nebývá optimální řešení, ale jen řešení suboptimální a doba potřebná k výpočtu je delší než u konvenčních metod. Proto by měl být algoritmus používán jen při takových úlohách, kde nemůžeme volit jiný postup. Přesto má algoritmus i svoje klady. Největší výhodou je, že ho můžeme použít právě tam, kde postup řešení není znám. Některé další výhody a nevýhody genetického algoritmu ve srovnání s konvenčními metodami obsahuje tabulka 9.

Tabulka 9: Výhody a nevýhody použití genetického algoritmu

Výhody	**Nevýhody**
Nemusíme znát postup řešení problému	Pomalejší oproti konvenčním řešením
Snadno paralelizovatelný	Obtížné hledání vhodné fitness funkce
Aplikovatelný na mnoho problémů	Potřeba vhodně určit množství parametrů
Odolný na uváznutí v lokálním extrému	Obtížné určení ukončovací podmínky
Nemusíme znát průběh optimalizované funkce	Těžko se aplikují konkrétní znalosti problému
Poradí si s velkým prohledávaným prostorem	Často najde jen suboptimální řešení
Lehko přizpůsobitelný změně podmínek	Požaduje mnohokrát výpočet fitness funkce

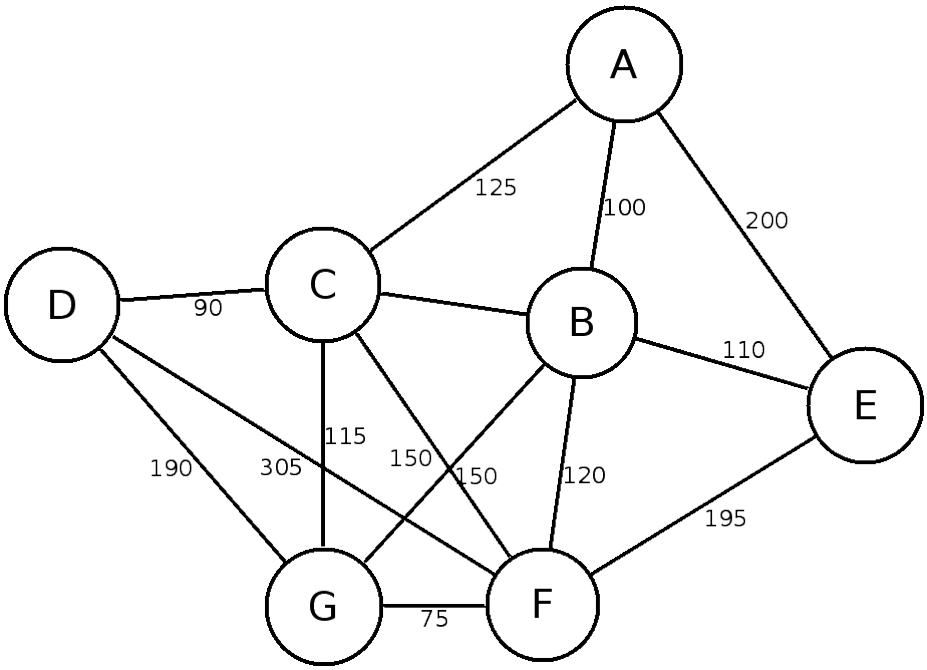
4 Bezdrátová senzorová síť

Bezdrátovou senzorovou síť tvoří jednotlivá zařízení, která spolu mohou komunikovat. Bližší zařízení komunikují přímo, vzdálenější zařízení pak komunikují pomocí těch bližších. Pro vzájemnou komunikaci všech zařízení mezi sebou je důležitá znalost přibližné topologie sítě, aby bylo možné efektivně směrovat data mezi jednotlivými senzory bezdrátové sítě.

Jednotlivé senzory poskytují různé informace. Pro zjišťování jejich topologie využijeme jedinou informaci, a to sílu signálu, kterou k nim vysílá senzor, s nímž zrovna komunikují.

4.1 Senzorová síť jako graf

Pro genetický algoritmus potřebujeme formálnější a abstraktnější popis senzorové sítě. Na síť lze také nahlížet jako na ohodnocený graf, jak ukazuje obrázek 2. Senzory jsou uzly grafu a hrany spojují ty uzly, mezi nimiž byla změřena síla signálu. Ohodnocení hrany odpovídá síle signálu.



Obrázek 2: Senzorová síť jako graf

4.2 Měření síly signálu

Síla signálu se mezi každou dvojicí senzorů měří několikrát. Průměr těchto měření se poté využije jako ohodnocení hrany grafu. Vícenásobné měření je nutné, jelikož síla signálu velmi kolísá a přesnost měření má zásadní vliv na úspěšné zjištění přibližné topologie sítě.

Na sílu signálu má ještě vliv nastavený vysílací výkon každého senzoru. Pro další postup práce předpokládám, že všechny senzory mají nastaven stejný vysílací výkon.

4.3 Převod síly signálu na vzdálenost

Abychom byli schopni určit polohy senzorů, je vhodnější převést sílu signálu na vzdálenost. Nejprve změříme sílu signálu a jí odpovídající vzdálenost. Poté určíme závislost vzdálenosti na síle signálu a tuto závislost vyjádříme tabulkou. Jelikož lze měnit vysílací výkon senzorů, je třeba sestavit pro každý vysílací výkon vlastní převodní tabulku.

Přesnost měření síly signálu klesá s rostoucí vzdáleností mezi senzory. Proto je vhodné upřednostnit měření mezi bližšími senzory s vyšší silou signálu a potlačit měření mezi vzdálenějšími senzory s menší silou signálu. Od jisté vzdálenosti již síla signálu nemá žádnou vypovídací hodnotu o vzájemné poloze senzorů. Pokud je síla signálu menší než tato experimentálně stanovená hranice, tak z tohoto měření nelze odvodit vzdálenost senzorů, a proto měření při sestavení grafu ignorujeme.

4.4 Shrnutí

Ze senzorové sítě vytvoříme ohodnocený graf. Ohodnocením hran se stane vzdálenost uzlů, které hrana spojuje. Tento graf bude vstupem genetického algoritmu, jenž bude mít za úkol určit souřadnice jednotlivých uzlů.

5 Využití genetického algoritmu při zjišťování topologie bezdrátové senzorové sítě

Při návrhu genetického algoritmu pro řešení konkrétního problému jde o to, jakým způsobem aplikovat teorii v praxi. Genetický algoritmus obsahuje mnoho parametrů a proměnných, které je třeba vhodně zvolit. Postupně se budu snažit popsat kompletní řešení. Vysvětlím, jak jsem se u jednotlivých aspektů genetického algoritmu rozhodl a proč jsem se tak rozhodl. Popíši zde i problémy, které nastaly při užití algoritmu a které změny bylo kvůli nim nutné provést.

Vstupem algoritmu je ohodnocený graf, kde uzly odpovídají jednotlivým senzorům a ohodnocení hran udává vzdálenost mezi senzory. Výstupem algoritmu budou souřadnice každého senzoru.

5.1 Schéma

Pro aplikaci jsem musel modifikovat schéma genetického algoritmu uvedené v kapitole 3, navíc jsem přidal operaci normalizace. Nové a detailnější schéma algoritmu je následující:

1. **Inicializace** – vytvoření chromozómů počáteční populace.
2. **Normalizace** – operace provedená pro každý chromozóm populace.
3. **Ohodnocení** každého chromozómu populace pomocí fitness funkce.
4. **Reprodukce** – opakuj pro vytvoření nové populace.
    - **Selekce** – turnajem vyber dva rodičovské chromozómy z populace v závislosti na jejich ohodnocení.
    - **Křížení** – s pravděpodobností 95 % aplikuj genetický operátor křížení na rodiče a tím vytvoř potomka. Pokud křížení neproběhne, potomek je kopií rodiče.
    - **Mutace** – s pravděpodobností 10 % aplikuj genetický operátor mutace a tím proveď náhodnou změnu potomka.
    - **Vložení** chromozómu do nové populace.
5. **Nahrazení** staré populace novou – inkrementační model.
6. Pokud není splněna **ukončovací podmínka**, přejdi na bod 2.
7. **Výsledkem** je chromozóm s nejlepším ohodnocením.

5.2 Chromozóm

Chromozómem musí být možno reprezentovat jakékoliv rozestavení libovolného počtu senzorů. Proto je vhodné každý senzor kódovat jako bod v rovině – dvojicí souřadnic x a y. Celý chromozóm potom bude obsahovat tolik bodů, kolik je senzorů. Souřadnice jsou uloženy jako celá čísla. Tabulka 10 znázorňuje strukturu chromozómu.

Tabulka 10: Struktura chromozómu

Senzor	**x**	**y**
A	x_a	y_a
B	x_b	y_b
...	...	...
Z	x_z	y_z

5.2.1 Odhad chyby bodu

U každého bodu je navíc uložena informace o odhadu jeho chyby. Odhad chyby bodu využívám jako druhý ukazatel kvality řešení a pro úpravu pravděpodobnosti mutace bodu. Výhoda je v tom, že mám informaci o přesnosti každého bodu zvlášť, kdežto fitness funkce hodnotí chromozóm jako celek.

Odhad chyby počítám jako maximum z chyb, které v poloze bodu vytvoří každá hrana, která z něj anebo do něj vede. Jelikož nelze žádným jednoduchým způsobem přesně určit, jakou chybu hrana způsobí, tak porovnám délku hrany v chromozómu s odpovídající hranou ve vstupním grafu a chyba hrany je rovna rozdílu těchto délek. Abych chybu hrany přepočetl na chybu uzlu, dělím ji dvěma, což zaručí, že se chyba rovnoměrně rozdělí mezi oba body.

5.3 Populace

Velikost populace jsem stanovil jako čtyřicetinásobek počtu senzorů. Při menší populaci se často stávalo, že algoritmus nenašel řešení vůbec. Větší populace jen prodlužuje čas potřebný k nalezení řešení.

5.4 Fitness funkce

Fitness funkce je chybová funkce, která každému chromozómu přiřazuje reálné číslo, které určuje, jak moc se topologie reprezentovaná hodnoceným chromozómem liší od topologie určené vstupním grafem. Cílem tedy je tuto funkci minimalizovat. Fitness funkce je popsána rovnicí (3).

$$f(x)=\sum_{e\in E}\left(\frac{(h(e)-length(e))^2}{4}+penalization(e)\right) \tag{3}$$

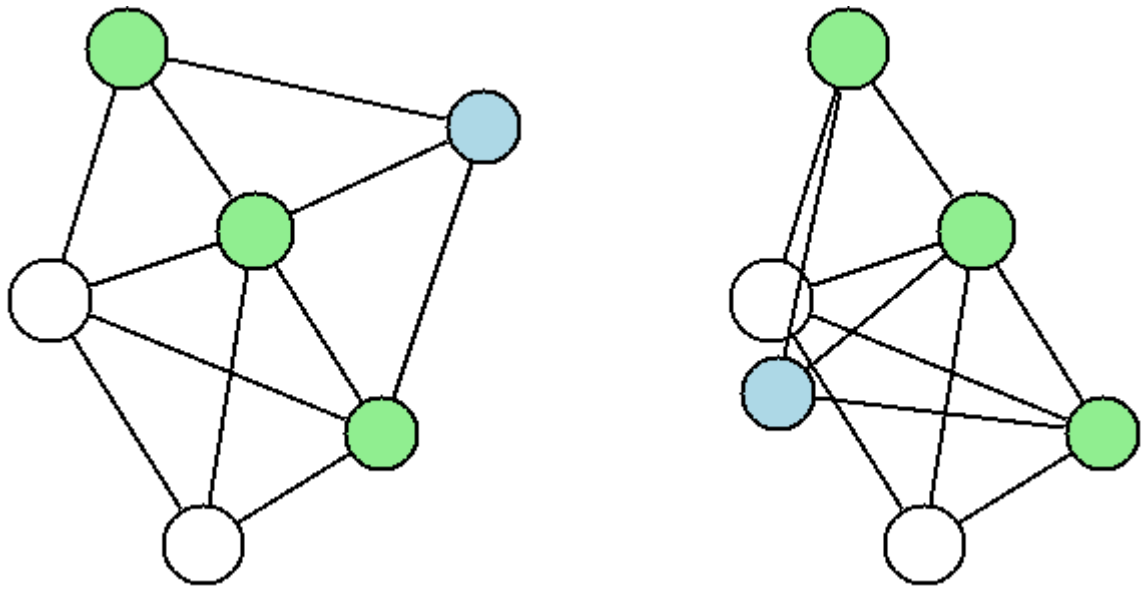
$$penalization(e)=\begin{cases}(min(e)-length(e))^4 & if\ length(e)<min(e)\\ 0 & else\end{cases} \tag{4}$$

E	množina hran
e	aktuální hrana
h(e)	vzdálenost (ohodnocení hrany vstupního grafu) vypočtená ze síly signálu
length(e)	vzdálenost senzorů v hodnoceného chromozómu
penalization(e)	penalizace za hranu, která je kratší než nejkratší hrana z počátečního nebo koncového uzlu e, vypočtená podle rovnice (4)
min(e)	nejmenší ohodnocení hrany, která vede z počátečního nebo koncového uzlu e

Použití penalizace, respektive minimální vzdálenosti v jejím výpočtu, předpokládá, že každý senzor bude mít signál k fyzicky nejbližšímu senzoru. Pokud by tato podmínka nebyla splněna, pak by hodnocený senzor, který nemá signál k nejbližšímu, byl penalizován.

5.4.1 Otočení bodů podle osy

Ze začátku jsem používal fitness funkci bez penalizace. Některé body se však neustále přibližovaly k ostatním a síť se vůbec nechtěla rozprostřít do plochy. Je to způsobeno tím, že fitness funkce bez penalizace nedokáže postihnout situaci, kdy se část grafu jako by překlopí na druhou stranu podle nějaké osy, jak je naznačeno na obrázku 3. Osa odpovídá zeleným senzorům, pro modrý senzor pak existují dvě ekvivalentní polohy, kde hodnocení obou chromozómů na obrázku je stejné, případně se liší jen minimálně. Přidání penalizace vytvoří mezi těmito polohami požadovaný rozdíl v ohodnocení, a algoritmus se tudíž snaží senzory umisťovat do správné pozice.



Obrázek 3: Otočení bodů podle osy

5.5 Inicializace

Počáteční populaci generuji náhodně. Souřadnice každého bodu musí být kladné kvůli použití jednobodového křížení jejich binární reprezentace, protože vyjádření záporných čísel v doplňkovém kódu způsobuje při jednobodovém křížení problémy.

5.5.1 Omezení souřadnic

Souřadnice sice generuji náhodně, ale i tak je třeba určit rozsah čísel, v jakých se náhodné hodnoty budou pohybovat. Toto omezení zlepší efektivitu algoritmu, jelikož se omezením prostoru, do kterého se náhodně body umisťují, zvýší pravděpodobnost správného rozmístění bodů.

Většinou platí, že jsou senzory rozmístěny zhruba ve čtvercové ploše. Zbývá tedy určit rozměry tohoto čtverce a všechny body generovat tak, aby ležely uvnitř. Lze předpokládat, že hustota senzorů v ploše je rovnoměrná, a proto počet senzorů odpovídající hraně čtverce bude odmocnina z celkového počtu senzorů. Pro určení délky hrany čtverce potřebujeme vědět nejen počet senzorů, ale i to, jaká je mezi senzory vzdálenost. Tu můžeme získat jako průměrnou vzdálenost mezi všemi senzory. Proto souřadnice náhodně generuji z intervalu <0; max>, kde max vypočítám podle vzorce (5).

$$max = \sqrt{|V|}\frac{1}{|E|}\sum_{e \in E} e \tag{5}$$

V	množina uzlů
E	množina hran
e	ohodnocení (délka) příslušné hrany

5.6 Selekce

Výběr dvou rodičů, z kterých vznikne potomek, provádím turnajem. Tato metoda se jeví jako výhodná, jelikož rozdíly v ohodnocení jednotlivých chromozómů jsou značné, a proto by například metodu výběru pomocí rulety nebylo možné použít.

5.7 Křížení

V chromozómu se vzájemně kříží sobě si odpovídající body. S pravděpodobností:

- 90 % se provede jednobodové křížení binární reprezentace pro každou souřadnici každého bodu, jak naznačuje tabulka 11,
- 10 % se provede aritmetické křížení podle rovnice (6).

První typ křížení slouží k základnímu rozestavení bodů, druhý pak k jemnějšímu upřesnění pozice a zmenšení chyby. Pravděpodobnost křížení chromozómu je 95 %.

Tabulka 11: Křížení chromozómu

Rodič A	x_1	y_1	x_2	y_2	...	x_n	y_n
	↕	↕	↕	↕	...	↕	↕
Rodič B	x_1	y_1	x_2	y_2	...	x_n	y_n

$$\begin{aligned} x &= r x_a + (1-r)\, x_b \\ y &= r y_a + (1-r)\, y_b \end{aligned} \tag{6}$$

x, y	souřadnice bodu potomka
x_a, y_a	souřadnice bodu z prvního rodiče
x_b, y_b	souřadnice bodu z druhého rodiče
r	náhodně zvolená konstanta v intervalu <0; 1>

5.8 Mutace

Operátor mutace posune bod o náhodně zvolenou vzdálenost v každé ose. Interval, z jakého se náhodně vybírá vzdálenost pro posun, je <0; max>, kde jako max beru délku nejdelší hrany ze vstupního grafu.

Pravděpodobnost posunu každého bodu v chromozómu se vypočte podle vzorce (7). Jak vyplývá ze vzorce, mutace častěji pohybuje s body, které mají velkou chybu, což má kladný vliv na rychlost algoritmu.

$$p_i = \frac{\frac{1}{N} + \frac{error_i}{\sum_{i=0}^{N} error_i}}{2} \tag{7}$$

N	počet bodů
$error_i$	odhad chyby i-tého bodu

Pravděpodobnost mutace chromozómu je 10 %.

5.9 Nahrazení

Pro nahrazení staré populace novou jsem zvolil model překrytí generací, takže se nahradí pouze část populace novými chromozómy. Konkrétně se vytvoří takový počet potomků, aby odpovídal 10 % aktuální populace a sloučí se se stávající populací. Takto vzniklá nová populace se seřadí podle hodnoty fitness funkce. Z nové populace se odstraní nejhorší chromozómy. Počet odstraněných chromozómů je stejný jako počet přidaných potomků, takže velikost populace se v průběhu výpočtu nemění.

5.10 Normalizace

Ke zlepšení efektivity algoritmu jsem navrhl operaci normalizace. Je to speciální operace vytvořená jen pro tuto aplikaci genetického algoritmu. Provádí se před užitím genetických operátorů u všech chromozómů populace. Spočívá v průchodu všech bodů chromozómu a nalezení nejmenší hodnoty pro osu x a y. Tato hodnota se od všech bodů odečte. Tím dostaneme všechna řešení do jednoho místa a poté lépe fungují genetické operátory, zejména křížení.

5.11 Ukončovací podmínka

Pro ukončení výpočtu genetického algoritmu využívám dvou podmínek současně. První podmínka se váže ke změně ohodnocení nejlepšího chromozómu, druhou podmínkou je překročení časového limitu. Genetický algoritmus se zastaví, jakmile je splněna jakákoliv z podmínek a řešením se stává nejlépe ohodnocený chromozóm.

5.11.1 Změna ohodnocení nejlepšího chromozómu

Pokud po nějakou dobu nedojde ke zlepšení řešení, je vhodné algoritmus ukončit, jelikož lze předpokládat, že se řešení nebude zlepšovat ani nadále, a tudíž pokračování ve výpočtu je zbytečné.

K určení, kdy se řešení přestane zlepšovat, využiji znalost toho, jaká je závislost ohodnocení nejlepšího chromozómu na počtu generací. Protože využívám model překrytí generací, a nejlepší řešení mi tudíž vždy zůstane a nebo je nahrazeno v následující generaci řešením ještě lepším (s nižším ohodnocením), lze říci, že funkce závislosti ohodnocení nejlepšího chromozómu na počtu generací je nerostoucí. Když úseky této funkce aproximuji přímkou, mohu pak její směrnici brát jako rychlost vylepšování řešení. Jakmile se směrnice takto vytvořené přímky bude blížit nule, rychlost zlepšování řešení se také bude blížit nule a výpočet se může zastavit. Směrnici vypočítám pomocí metody nejmenších čtverců [6] podle vzorce (8).

Potřebuji tedy zvolit dva parametry – počet hodnot intervalu, v kterém budu hledat přímku aproximující body, a sklon této přímky, při kterém ukončím výpočet. Obě hodnoty jsem stanovil experimentálně. Jako počet hodnot jsem zvolil padesátinásobek počtu senzorů a algoritmus ukončím, když směrnice přímky aproximující tyto hodnoty bude větší než -10^{-8}.

$$k = \frac{n\sum_{i=1}^{n}(x_i y_i) - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n}(x_i^2) - (\sum_{i=1}^{n} x_i)^2} \tag{8}$$

k	sklon (směrnice regresní přímky)
n	počet hodnot intervalu
x_i	generace
y_i	ohodnocení nejlepšího chromozómu i-té generace

6 Důležité rysy návrhu a implementace

Aplikaci jsem rozdělil na tři základní části, kterými jsou server, klient a síťová knihovna pro komunikaci klienta se serverem. Za nejdůležitější části považuji server a síťovou knihovnu. Předpokládám jejich budoucí rozšíření a vylepšení, proto jsem věnoval značné úsilí kvalitnímu objektovému návrhu. Klient slouží jen k testování a ukázání funkčnosti aplikace.

Všechny části jsem implementoval v jazyce C++ a jeho standardní knihovně STL [8] [9]. V ukázkovém klientovi jsem na vizualizaci použil knihovnu Qt [10].

6.1 Architektura klient – server

Vzhledem k tomu, že je aplikace určená především jako pomocná a bude využívána jinými programy, je třeba zvolit vhodný způsob propojení těchto programů a komunikace mezi nimi. Jako nejvhodnější se mi jeví architektura klient – server s textovým komunikačním protokolem.

Obrázek 4: Architektura klient – server

Nezvolil jsem konkurentní variantu serveru, jelikož genetický algoritmus je výpočetně velmi náročný a nemělo by smysl zpracovávat více požadavků zároveň. Místo toho se požadavky řadí do fronty a zpracovávají se postupně.

6.2 Komunikační protokol

Pro komunikaci mezi klientem a serverem jsem zvolil textový komunikační protokol. Protokol musí být schopen přenést vyjádření grafu a potom výsledek čili reprezentaci bodů v rovině. Pro přenos dodatečných informací jsem před data připojil hlavičky.

Strukturu komunikačního protokolu ukazuje tabulka 12. V této i ve všech dalších tabulkách popisujících části protokolu mají řádky tabulky význam řádku protokolu, což znamená, že je za nimi znak konce řádku. Mezi sloupci protokol očekává alespoň jeden bílý znak kromě konce řádku. Posledním bajtem v datech musí být znak s hodnotou nula. V tabulce 12 je to znázorněno pomocí \n pro nový řádek a \0 pro ukončení dat.

Tabulka 12: Struktura komunikačního protokolu

Hlavička 1: hodnota 1 **\n**
Hlavička 2: hodnota 1 **\n**
… **\n**
Hlavička N: hodnota N **\n**
\n
Data **\0**

6.2.1 Hlavičky protokolu

Hlavičky, kterým rozumí server jsou pak popsány v tabulce 13 a hlavičky, jenž odesílá server klientovi vysvětluje tabulka 14.

Tabulka 13: Hlavičky, které přijímá server

Název	**Datový typ**	**Popis**
Debug	bool	působí odesílaní průběžných výsledků a délek hran
Timeout	int	maximální doba běhu genetického algoritmu v sekundách
Power	int	vysílací výkon senzorů

Tabulka 14: Hlavičky, které posílá server

Název	**Datový typ**	**Popis**
Final	bool	indikuje poslední odesílané řešení, využití hlavně při Debug=true
Epoch	int	počet generací, které uplynuly od začátku výpočtu
Fitness	double	hodnota fitness funkce nejlepšího řešení
AvgFitness	double	průměrná hodnota fitness celé populace
AvgError	double	průměr odhadu chyby všech senzorů v nejlepším řešení
Time	int	uplynulý čas v sekundách od počátku výpočtu

Popis datových typů hlaviček se nachází v tabulce 15. U datových typů jsou uvedené jen informace, co si pod ním představit, obecně lze využít jakýkoliv textový řetězec, který je schopna zkonvertovat třída `std::stringstream` na požadovaný datový typ.

Tabulka 15: Datové typy protokolu

Datový typ	**Popis**
bool	true, false
int	celé číslo
double	desetinné číslo

6.2.2 Data protokolu

Serveru se jako data posílá reprezentace vstupního grafu. Způsob převodu grafu na data protokolu ukazuje tabulka 16. Po výpočtu, případně v průběhu výpočtu, server odpoví zasláním nejlepšího řešení. Kódování nejlepšího řešení do dat protokolu popisuje tabulka 17.

Tabulka 16: Graf jako data protokolu

$from_1$	to_1	$signal_1$
$from_2$	to_2	$signal_2$
...	...	...
$from_N$	to_N	$signal_N$

Tabulka 17: Řešení jako data protokolu

$point_1$	x_1	y_1	$error_1$
$point_2$	x_2	y_2	$error_2$
...	...	...	...
$point_{1N}$	x_N	y_N	$error_N$

6.3 Objektový návrh

Návrh celé aplikace jsem rozdělil do tří na sobě nezávislých částí, které navzájem spolupracují. UML diagram tříd se nachází na přiloženém CD a poskytuje detailnější pohled na návrh. Zeleně jsou na něm označeny třídy, které jsou součástí síťové knihovny, která je využívána serverem i klientem. Růžovou barvu mají třídy patřící ukázkovému klientovi. Modrá barva odpovídá třídám serveru.

6.3.1 Síťová knihovna

Knihovna je určena k abstrakci a zjednodušení komunikace mezi serverem a klientem. Také usnadňuje implementaci nového klienta. Skládá se z tříd `Socket`, `ProtocolData` a `Protocol`. Všechny třídy jsou nezávislé na genetickém algoritmu a lze je využít i v jiné aplikaci.

`Socket` je abstrakce nad BSD schránkami a zajišťuje jejich zapouzdření. Je to jediná třída, která se stará o komunikaci, všechny ostatní jsou jen její nadstavby a využívají ji.

Zapouzdření práce s komunikačním protokolem, který vychází z tabulky 12, poskytuje třída `ProtocolData`. Obsahuje metody, které pracují s hlavičkami a daty.

Třída `Protocol` je poděděná z třídy `Socket` a místo odesílaní surových dat umožňuje odesílat a přijímat přímo instance `ProtocolData`.

6.3.2 Server

Server je nejsložitější částí této aplikace. Obstarává komunikaci s klientem a hlavně provádí samotný výpočet pozice pomocí genetického algoritmu. Návrh serveru lze rozdělit do tří modulů. První modul se zabývá síťovou komunikací a je postaven na síťové knihovně.

Druhý modul tvoří malá knihovna pro práci s genetickými algoritmy. Odděluje logiku algoritmu od jeho konkrétní aplikace. Obsahuje dvě abstraktní třídy – `AbstractGeneticAlgorithm` a `AbstractChromosome`. Připravil jsem je pro co nejjednodušší a nejrychlejší aplikaci genetického algoritmu. Pro základní funkčnost tedy stačí implementovat minimum dalších metod.

Poslední a největší část představuje využití genetického algoritmu při zjištění topologie bezdrátové senzorové sítě. Použil jsem knihovnu pro práci s genetickými algoritmy popsanou v minulém odstavci. Z odpovídajících abstraktních tříd knihovny jsem podědil třídy `GeneticAlgorithm` a `Chromosome` a doplnil jsem jim čistě virtuální i další metody. Přidal jsem také třídy potřebné k reprezentaci grafu. Důležitou roli na serveru hraje ještě převod síly signálu na vzdálenost. Pro tento účel jsem vytvořil tabulky, pomocí kterých se převod realizuje.

Převodní tabulka

Převodní tabulka je textový soubor obsahující na každém řádku dvojici síla signálu a k tomu odpovídající vzdálenost, jak je uvedeno v tabulce 18. Pokud nějaká dvojice signál – vzdálenost není nalezena, pak se využije aproximace přímkou a chybějící hodnota se dopočítá pomocí dvou nejbližších sousedních bodů.

Tabulka 18: Převodní tabulka

$signál_1$	$vzdálenost_1$
$signál_2$	$vzdálenost_2$
...	...
$signál_n$	$vzdálenost_n$

Pro každý vysílací výkon je třeba vytvořit novou tabulku. Podle požadavku z hlavičky komunikačního protokolu vyhledá server správnou tabulku, pomocí které uskuteční převod. Pokud požadovaná tabulka není nalezena, použije se implicitní (`default.sdt`). Soubory s tabulkami jsou uloženy v adresáři `tables` u serveru. Jméno souboru s tabulkou se tvoří z vysílacího výkonu a přípony sdt. Například tedy pro výkon 20 se hledá tabulka s názvem souboru `20.sdt`.

6.3.3 Klient

Ukázkový klient využívá síťovou knihovnu. Pro grafické uživatelské rozhraní jsem použil knihovnu Qt. Klient umožňuje posílat dotazy na server a vykreslit nalezenou topologii. Můžeme v něm zvolit i některé parametry. Například maximální dobu výpočtu, zapnout debug mód a nebo nastavit, s jakým vysílacím výkonem senzorů má genetický algoritmus počítat. Také lze zvolit, k jakému serveru a na jaký port se má klient připojit. Mimo základní nastavení a vykreslení ještě klient poskytuje statistiky o provedených pokusech o zjištění topologie. Ukládá počet generací, hodnotu fitness funkce řešení, průměrnou chybu senzoru a čas potřebný pro nalezení řešení.

7 Testování a experimenty

Pro testování algoritmu a experimenty s ním jsem vybral několik různých topologií. Fungování navrženého genetického algoritmu jsem zkoušel na testovacích topologiích, ve kterých jsem nepočítal se silou signálu a převodem na vzdálenost. Poté jsem provedl měření a stanovil jsem závislost vzdálenosti na síle signálu. Pro otestování praktického nasazení jsem provedl i několik experimentů s reálnými topologiemi. Zaměřil jsem se i na stanovení úspěšnosti nalezení přibližné topologie sítě. Závěrem jsem popsal problémy, které jsem při testování a experimentování objevil, a jejich případné řešení.

7.1 Vysvětlení obrázků

Senzory jsou na obrázcích vyznačeny jako zelené kruhy (uzly) s číslem. Bílé číslo odpovídá číslu uzlu ze vstupního grafu. Kružnice kolem uzlu vypovídá o přesnosti polohy a ohraničuje plochu, ve které se senzor může nacházet. Hrany znázorňují, mezi jakými senzory byl naměřen signál. Ohodnocení hrany udává vzdálenost mezi odpovídajícími senzory.

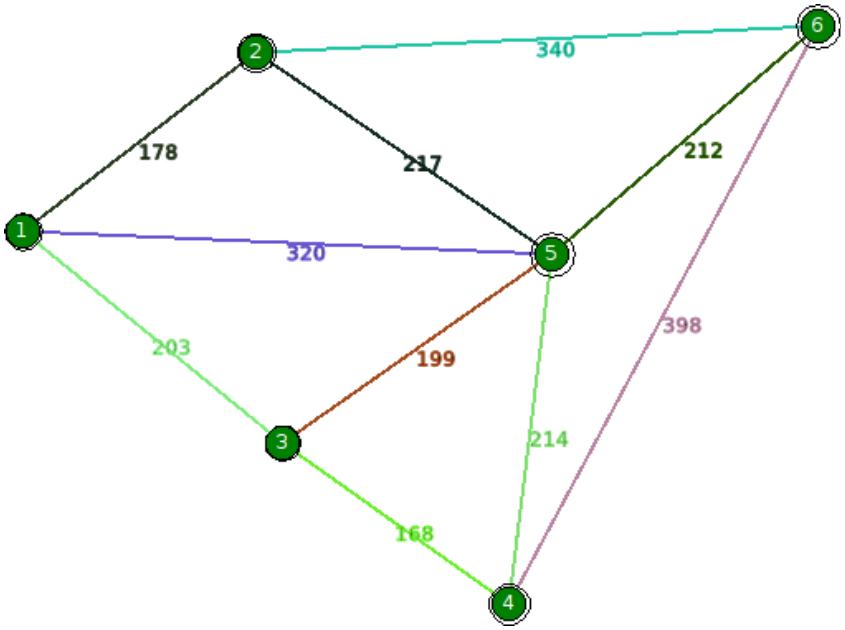
7.2 Testovací topologie

Testovací topologie mají za úkol ověřit pouze funkčnost algoritmu. Nepočítám přitom s přepočtem síly signálu na vzdálenost. Vstupem algoritmu je graf, jehož ohodnocení hran přímo odpovídá změřené vzdálenosti mezi uzly.

Během prvních spuštění programu nebylo téměř možné dospět k alespoň přibližné topologii, jelikož fitness funkce neobsahovala penalizaci. Problém je podrobně popsán v kapitole 5.4. Úprava algoritmu problém vyřešila.

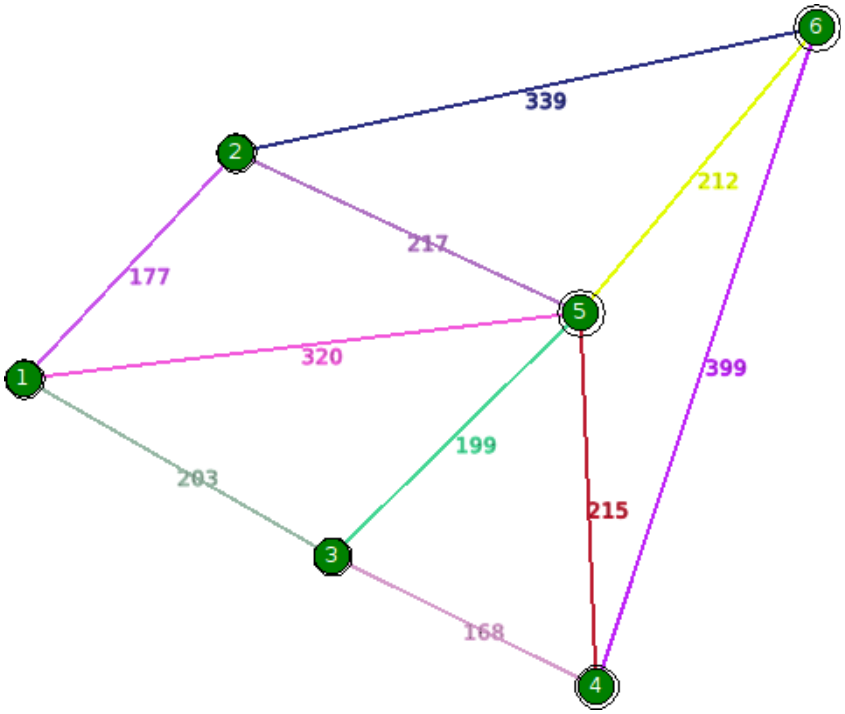
7.2.1 Topologie s malým počtem uzlů

Zadání topologie skládající se z 6 uzlů můžeme vidět na obrázku 5.



Obrázek 5: Testovací topologie s malým počtem uzlů – zadání

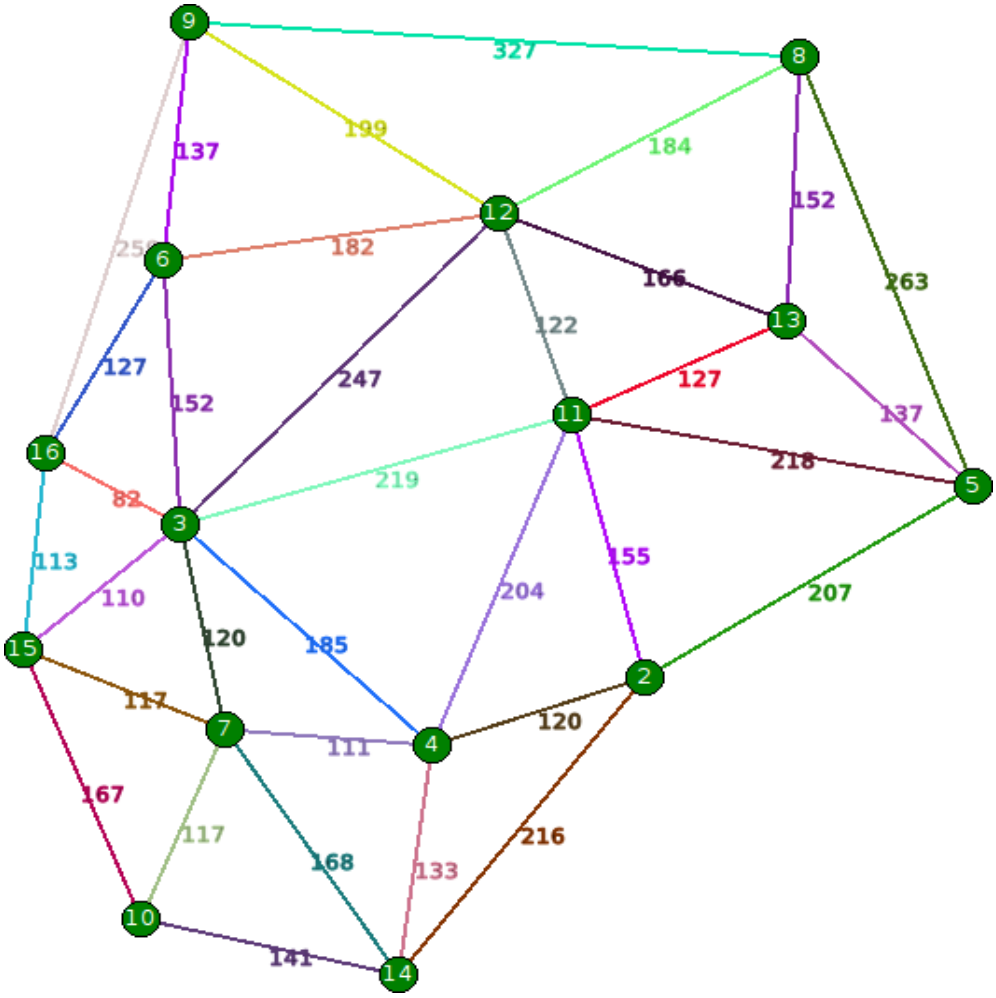
Nalezenou topologii ukazuje obrázek 6. Při srovnání řešení a zadání je vidět, že se polohy jednotlivých senzorů ani vzdálenosti hran téměř neliší. Rozdíl byl jen u některých hran, a to maximálně jednu jednotku délky.



Obrázek 6: Testovací topologie s malým počtem uzlů – řešení

7.2.2 Topologie s velkým počtem uzlů

Testovací topologie měla 15 uzlů. Jak přesně vypadá a jaké jsou reálné vzdálenosti mezi senzory, ukazuje obrázek 7. Algoritmus nalezl tuto topologii velice přesně a od topologie na obrázku 7 se téměř nelišila.



Obrázek 7: Testovací topologie s velkým počtem uzlů – zadání/řešení

7.3 Měření síly signálu a její převod na vzdálenost

Měření síly signálu mezi jednotlivými uzly probíhá takovým způsobem, že se každý uzel snaží komunikovat s uzly ve svém okolí. S každým sousedním uzlem si při této komunikaci vymění 5 zpráv a tím se 5× změří síla signálu k tomuto uzlu. Z těchto 5 hodnot se poté spočte průměrná síla signálu. Stejné měření proběhne z druhé strany a také se 5 naměřených hodnot zprůměruje. Síla signálu mezi dvěma senzory použitá pro algoritmus je průměr měření z obou stran.

Nejdříve je třeba určit, z jakých měření vytvořím tabulku pro přepočet síly signálu na vzdálenost. Zkusil jsem dvě možnosti, a to měření jen mezi dvěma uzly a poté měření topologie s více uzly. Z obou měření jsem sestavil tabulku závislosti síly signálu na vzdálenosti.

7.3.1 Měření mezi dvěma uzly

Nejdříve jsem změřil sílu signálu a vzdálenost jen mezi dvěma uzly a na základě tohoto měření jsem sestavil převodní tabulku [7]. Takto vytvořená tabulka nebyla využitelná pro převod síly signálu na vzdálenost v rozsáhlejších topologiích, protože při větším počtu senzorů je síla signálu ovlivňována i okolními senzory.

7.3.2 Měření mezi více uzly

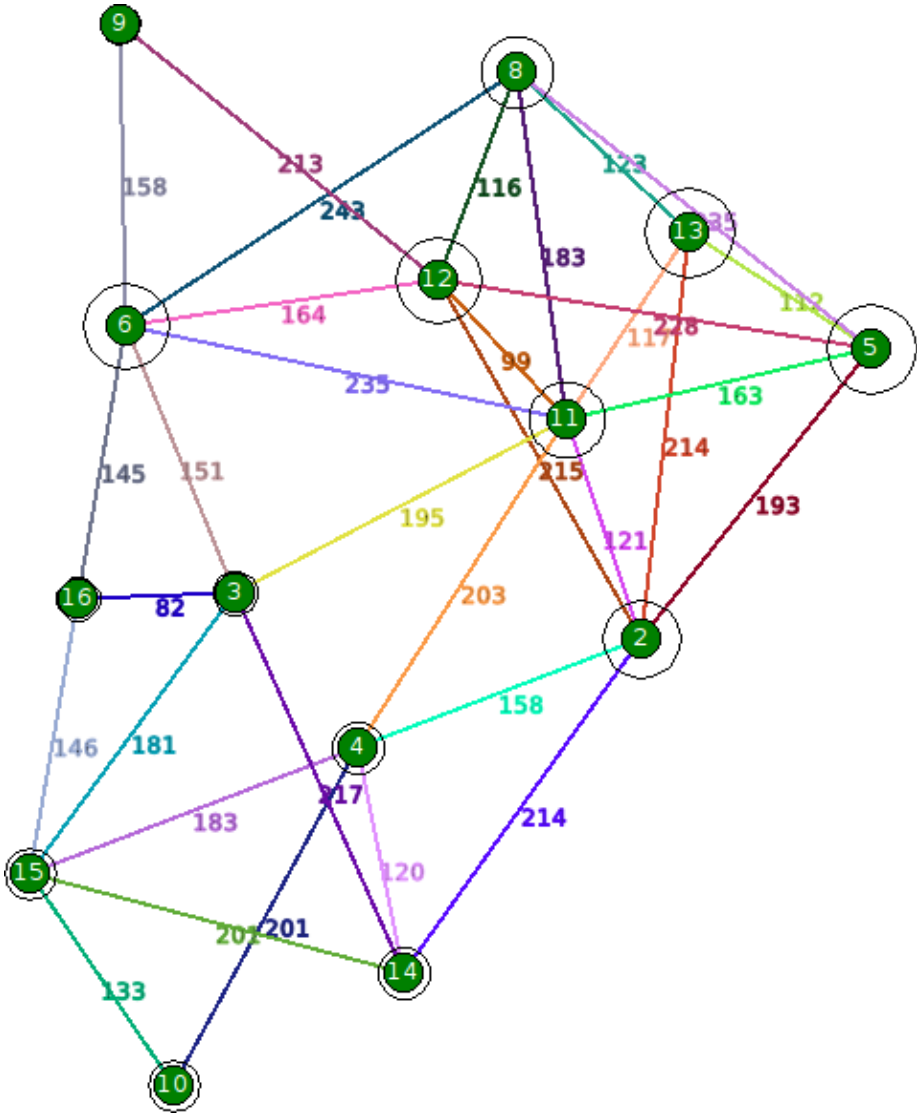
Protože se měření jen mezi dvěma uzly ukázalo jako nepoužitelné, tak jsem zkusil pro měření sestavit větší topologii s 15 uzly a změřit všechny vzdálenosti, u kterých znám sílu signálu. Ze síly signálu a změřené vzdáleností jsem sestavil převodní tabulku. Takto sestavená tabulka je již vhodná i pro topologie s více senzory.

7.4 Experimenty s reálnou topologií

Pro ověření funkčnosti v reálných podmínkách jsem provedl několik experimentů. Využil jsem testovací topologii z úvodu této kapitoly i dvě nové topologie, které se mezi sebou lišily vzdáleností mezi jednotlivými senzory. K určení těchto topologií je již využita síla signálu přepočtená na vzdálenost a ne pouze změřená vzdálenost jako u testovacích topologií.

7.4.1 Původní testovací topologie s velkým počtem uzlů

Nalezenou testovací topologii, jejíž zadání je na obrázku 7, můžeme vidět na obrázku 8. Chybí v ní senzor číslo 7, jelikož v něm nebyly dostatečně nabité baterie, a tudíž nebylo možné provést měření síly signálu. Při srovnání nalezeného rozmístění senzorů s jejich reálnou polohou můžeme vidět, že sice přesně neodpovídají vzdálenosti mezi senzory, ale pro orientaci je nalezené řešení dostatečně dobré. Nalezená poloha každého senzoru se nijak významně neliší od reality.



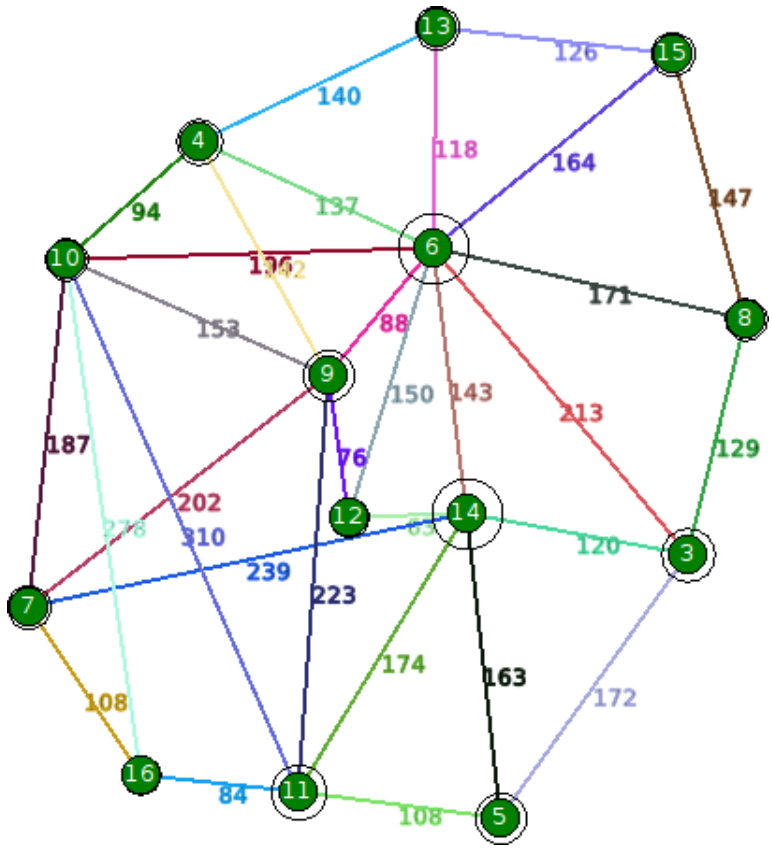
Obrázek 8: Nalezená původní testovací topologie s velkým počtem uzlů

7.4.2 Velké vzdálenosti mezi uzly

V této topologii byly vzdálenosti mezi uzly příliš velké, a měření tudíž nepřesné, a proto genetický algoritmus nebyl vůbec schopen nalézt ani přibližnou topologii sítě. Jedná se o topologii, ve které už se výrazným způsobem projevila nedostatečná přesnost měření, jež je závislá právě na vzdálenosti senzorů.

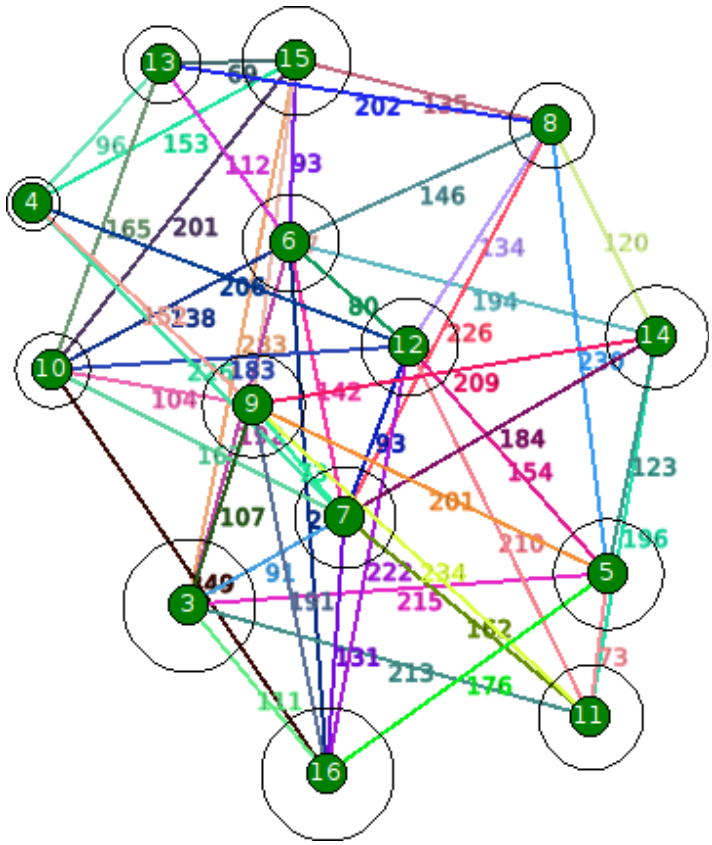
7.4.3 Malé vzdálenosti mezi uzly

Síť se 14 senzory a menšími vzdálenostmi mezi nimi ukazuje obrázek 9.



Obrázek 9: Reálná topologie s malými vzdálenostmi mezi uzly

Použitím genetického algoritmu jsem dostal výslednou topologii, kterou ukazuje obrázek 10. Můžeme vidět, že uzly číslo 3, 7 a 14 mají odlišnou polohu od reality. Poloha ostatních uzlů zhruba odpovídá jejich skutečné poloze.



Obrázek 10: Nalezená reálná topologie s malými vzdálenostmi mezi uzly

7.5 Měření úspěšnosti nalezení přibližné topologie

Úspěšnost nalezení přibližné topologie počítám jako poměr počtu úspěšného nalezení topologie ku celkovému počtu pokusů. Měření jsem prováděl odděleně na testovacích a reálných topologiích, protože jsem očekával, že vlivem nepřesnosti měření síly signálu se hodnoty budou výrazně lišit. S každou topologií jsem provedl 25 měření. Tabulku jednotlivých měření obsahuje příloha.

7.5.1 Testovací topologie

K určení úspěšnosti pro testovací topologie jsem využil testovací síť s malým i velkým počtem senzorů. Za úspěšné považuji výsledky, které mají průměrnou chybu na senzor do 5 jednotek pro topologii s malým počtem uzlů a do 10 jednotek pro topologii s velkým počtem uzlů. Úspěšnost nalezení přibližné topologie pro testovací sítě je 78 %.

7.5.2 Reálná topologie

Pro stanovení úspěšnosti reálné topologie jsem využil původní testovací topologii s velkým počtem uzlů a reálnou topologii s malými vzdálenostmi mezi uzly. Topologii s velkými vzdálenostmi mezi uzly jsem ignoroval, jelikož se ji algoritmu nikdy nepodařilo ani přibližně určit.

Za úspěšné považuji výsledky, které mají průměrnou chybu na senzor do 20 jednotek pro původní testovací topologii a do 54 jednotek pro topologii s malými vzdálenostmi mezi uzly. Úspěšnost nalezení přibližné topologie pro reálné sítě je 55 %.

7.6 Odhalené problémy

Během testování a experimentování jsem odhalil několik problémů. Hlavní dvě příčiny vzniku těchto problémů jsou nedostatek informací a nepřesné informace.

7.6.1 Nedostatek informací

Vzhledem k tomu, že vstupem algoritmu je jen síla signálu mezi jednotlivými uzly, ale ne směr, není možné řešit problémy týkající se orientace celé topologie, respektive jejího natočení či převrácení. Nedostatek informací způsobuje tyto problémy:

- **Zrcadlové převrácení** – nalezená přibližná topologie může být zrcadlově převrácená,
- **Otočení** – natočení celé topologie neodpovídá realitě. Může být otočena o libovolný úhel.

Jedním z možných řešení by mohlo být zpracování dalších informací, z kterých půjde odvodit směr. Pokud by například alespoň tři senzory znaly svoji GPS polohu a tato poloha byla využita v genetickém algoritmu, stačilo by to k odstranění obou dvou nedostatků.

7.6.2 Nepřesnost informací

Do kategorie nepřesných informací řadím především chyby měření a chyby převodu síly signálu na vzdálenost. Chyba, kterou do výsledku přidá samotné určení polohy genetickým algoritmem v případě úspěšného nalezení přibližné topologie, je oproti chybám zmíněným dříve spíše zanedbatelná.

Chyby měření jsou závislé především na hardwaru. Jak se ukázalo, je to zdroj největších chyb. Síla signálu mezi uzly kolísá. Vliv na ni má například prostředí, okolní uzly a nastavený vysílací výkon. Jedním z řešení bylo měřit sílu signálu vícekrát a naměřené hodnoty průměrovat.

Převod síly signálu na vzdálenost do algoritmu přináší další typ chyb. Jejich původ je ale opět v měření, jelikož převodní tabulku sestavuji na základě měření síly signálu a reálné vzdálenosti mezi senzory. Pokud je tedy chybné nebo nepřesné měření síly signálu, tak je potom nepřesná či chybná i převodní tabulka.

7.6.3 Chyba genetického algoritmu

Jeden z problémů, které přinesl genetický algoritmus respektive ohodnocení jednotlivých chromozómů fitness funkcí, lze demonstrovat například na trojúhelníku. Každé tři body, mezi nimiž je vzájemně změřena síla signálu, tvoří trojúhelník a po přepočtu síly signálu na vzdálenost mezi uzly lze říci, že známe délky stran tohoto trojúhelníka. Může se ale stát, že díky různým chybám délky těchto stran nesplňují trojúhelníkovou nerovnost. Z toho plyne zřejmě nejzásadnější

chyba, s kterou se genetický algoritmus musí dokázat vyrovnat. Tato chyba roste s tím, kolik informací máme, a promítá se do hodnocení chromozómu. Obecně lze říci, že měření nám poskytuje více informací, než potřebujeme, a některé z nich jsou vzájemně v rozporu. Tento rozpor spíše než chybné rozmístění uzlů způsobuje to, že genetický algoritmus není schopen najít ani přibližnou topologii senzorové sítě.

7.7 Zhodnocení

Algoritmus jsem vyzkoušel na testovacích i reálných datech. Pro testovací topologie algoritmus funguje velice dobře. V reálných topologiích má vliv na úspěšné nalezení topologie především přesnost měření a převod síly signálu na vzdálenost. V případě úspěšného nalezení přibližné topologie algoritmus velkou nepřesnost nepřináší, jelikož jde hlavně o poměry vzdáleností a vzájemnou polohu senzorů.

8 Závěr

Práce se zabývá určením přibližné topologie bezdrátové senzorové sítě na základě síly signálu mezi jednotlivými senzory. K určení této topologie využívám genetický algoritmus. Výstupem algoritmu jsou souřadnice jednotlivých senzorů.

Pro určení přibližné topologie bezdrátové senzorové sítě je nejprve třeba změřit síly signálů mezi jednotlivými senzory sítě a tyto síly signálu převést na vzdálenost. Na základě těchto měření se vytvoří ohodnocený graf reprezentující tuto síť – uzly odpovídají jednotlivým senzorům a ohodnocení hran je vzdálenost vypočtená ze síly signálu mezi senzory. Takto vytvořený graf je vstupem genetického algoritmu. Optimalizace pomocí genetického algoritmu spočívá v nalezení takové reprezentace vstupního grafu, u které jsou známy souřadnice jednotlivých uzlů a rozdíl v ohodnocení hran vstupního grafu od grafu řešení je minimální. Ohodnocení hran grafu řešení se počítá jako vzdálenost uzlů, které hrana spojuje. Souřadnice uzlů v nalezeném grafu jsou hledané souřadnice příslušných senzorů.

Aplikaci jsem nejprve využil pro určení souřadnic testovací topologie, tj. takové topologie, která nebere v úvahu měření síly signálu, ale jen změřené vzdálenosti mezi uzly. Na takových topologiích dosahuje program velmi dobrých výsledků. Přesnost určení přibližné topologie je dostatečná a úspěšnost nalezení topologie testovací sítě je 78 %.

Při experimentech s reálnými topologiemi, kde již polohu určuji na základě síly signálu, jsem zjistil, že program funguje lépe pro topologie s menšími vzdálenostmi mezi senzory, jelikož s rostoucí vzdáleností klesá přesnost měření. Přesnost měření má zásadní vliv na určení správné polohy senzorů a úspěšnost nalezení přibližné topologie sítě. Menší vliv na úspěšnost algoritmu a na kvalitu řešení má pak přesnost převodu síly signálu na vzdálenost. Úspěšnost nalezení přibližné topologie reálné sítě je 55 %.

Celou aplikaci jsem rozdělil na tři základní části – server, síťovou knihovnu a klienta. Nejdůležitější částí je server, jenž implementuje genetický algoritmus. Síťová knihovna slouží pro usnadnění komunikace mezi serverem a klientem, případně pro zjednodušení vývoje nového klienta. Klient vytvořený v rámci této práce je určen jen k demonstraci funkčnosti aplikace a genetického algoritmu.

Prostor pro další výzkum vidím například v použití systému GPS u vybraných senzorů. Využitím informací o poloze z GPS by se dal vyřešit problém orientace topologie sítě. A kdyby se navíc podařilo zmenšit vliv přesnosti měření síly signálu na kvalitu a úspěšnost řešení, dosáhli bychom daleko lepších výsledků i v praktickém použití.

Literatura

[1] SIVANANDAM, S. N. a S. N. DEEPA. *Introduction to genetic algorithms*. Berlin: Springer, © 2008, 442 s. ISBN 978-3-540-73189-4.

[2] ZBOŘIL, František Vítězslav. *Soft Computing*: *Genetické Algoritmy*. Brno, 2012. Prezentace. FIT VUT v Brně.

[3] OBITKO, Marek. *Introduction to Genetic Algorithms* [online]. © 1998 [cit. 2012-05-01]. Dostupné z:http://www.obitko.com/tutorials/genetic-algorithms/

[4] KALÁTOVÁ, Eva a Jaroslav DOBIÁŠ. Evoluční algoritmy. *Západočeská univerzita v Plzni*: *Katedra informatiky a výpočetní techniky* [online]. Duben 2000 [cit. 2012-05-04]. Dostupné z:http://www.kiv.zcu.cz/studies/predmety/uir/gen_alg2/E_alg.htm

[5] DEMEL, Jiří. *Grafy a jejich aplikace*. Vyd. 1. Praha: Academia, 2002, 257 s. ISBN 80-200-0990-6.

[6] BUDÍKOVÁ, Marie, Maria KRÁLOVÁ a Bohumil MAROŠ. *Průvodce základními statistickými metodami*. 1. vyd. Praha: Grada, 2010, 272 s. ISBN 978-80-247-3243-5.

[7] ŽÍDEK, Petr. *Použití inteligentních agentů v bezdrátových senzorových sítích*. Brno, 2010. Diplomová práce. FIT VUT v Brně.

[8] *C++ Reference* [online]. © 2000-2012 [cit. 2012-05-01]. Dostupné z:http://www.cplusplus.com/reference/

[9] *C++ Reference* [online]. 2012 [cit. 2012-05-01]. Dostupné z: http://en.cppreference.com/w/cpp

[10] *Qt library 4.8* [online]. © 2011 [cit. 2012-05-01]. Dostupné z: http://qt-project.org/doc/qt-4.8/

Seznam příloh

A. Obsah přiloženého CD

Adresářová struktura

`src`	Zdrojové kódy
`network`	Síťová knihovna
`server`	
`client`	
`tables`	Tabulky převodu signálu na vzdálenost
`default.sdt`	Tabulka použitá v testech a experimentech
`100.sdt`	Tabulka pro převod v poměru 1:1
`topologies`	Testovací a experimentální topologie
`testing_small.txt`	
`testing_big.txt`	
`experimental_testing.txt`	
`experimental_short_disance.hex`	
`doc`	Dokumentace
`thesis.pdf`	Technická zpráva ve formátu pdf
`thesis.odt`	Technická zpráva ve formátu odt
`uml_class_diagram.svg`	UML diagram tříd
`Doxyfile`	Nastavení pro generování dokumentace
`Makefile`	

B. Manuál k aplikaci

Závislosti

Potřebné nástroje ke kompilaci aplikace:

- Operační systém s podporou BSD schránek – pro síťovou knihovnu,
- GCC s podporou C++11 nebo kompatibilní překladač,
- make,
- qmake,
- Qt 4.7 nebo vyšší – jen pro překlad klienta.

Překlad aplikace ze zdrojových textů

Nejprve je nutné zkopírovat adresářovou strukturu přiloženého CD do nového adresáře. Následně lze aplikaci přeložit pomocí příkazu `make`. Tím se vytvoří adresář `release`, kde bude přeložená aplikace a vše potřebné ke spuštění a testování.

```
$ make
```

Vygenerování dokumentace

Dokumentaci k aplikaci lze vygenerovat pomocí programu doxygen. Vygenerovaná dokumentace se bude nacházet v adresáři `doc/html`.

```
$ make doxygen
```

Server

Serveru lze při spuštění nastavit port, na kterém bude očekávat nová spojení. Implicitní port je 1234.

```
$ ./server [port]
```

Klient

Veškeré nastavení klienta se provádí až po jeho spuštění v GUI. Lze nastavit server (implicitně localhost) a port (implicitně 1234) a další volby týkající se genetického algoritmu.

C. Tabulka k měření úspěšnosti nalezení přibližné topologie

Číslo měření	Průměrná chyba senzoru			
	Testovací topologie		Reálná topologie	
	Malý počet uzlů	Velký počet uzlů	Původní testovací	Malé vzdálenosti
1	4,74	0,70	18,56	57,50
2	4,50	39,94	17,60	60,67
3	4,63	1,69	18,79	62,77
4	4,63	0,85	18,06	53,76
5	4,46	0,71	18,13	52,86
6	4,78	18,16	18,30	54,19
7	4,68	19,97	31,85	50,99
8	4,68	0,92	18,25	66,50
9	4,55	19,27	25,67	56,16
10	4,55	2,72	22,10	65,10
11	4,55	0,76	18,17	51,74
12	4,55	0,83	24,35	52,02
13	4,66	18,21	22,43	54,61
14	4,66	0,67	17,40	53,79
15	4,66	25,02	32,12	58,17
16	4,66	1,23	28,56	53,98
17	4,58	1,08	18,31	55,80
18	4,58	0,82	17,90	54,07
19	4,58	0,74	22,41	53,06
20	4,58	21,57	17,73	53,14
21	4,58	18,59	21,85	55,74
22	4,63	15,80	18,46	52,46
23	4,63	0,62	18,30	50,97
24	4,63	16,19	30,52	51,29
25	9,90	1,14	18,64	54,17
Úspěšnost v %	96 %	60 %	60 %	48 %
Celkem v %	78 %		55 %	