FAKULTA INFORMATIKY A MANAGEMENTU UHK

Hledání optimálního rozmístění vysílačů

- genetický algoritmus -

semestrální projekt z předmětu ZT4

Vypracoval:	**Petr Voborník**
Obor:	im(5)
Forma:	prezenční
Ročník:	4.
Datum:	2.5.2005
E-mail:	vobornik@mikmik.cz

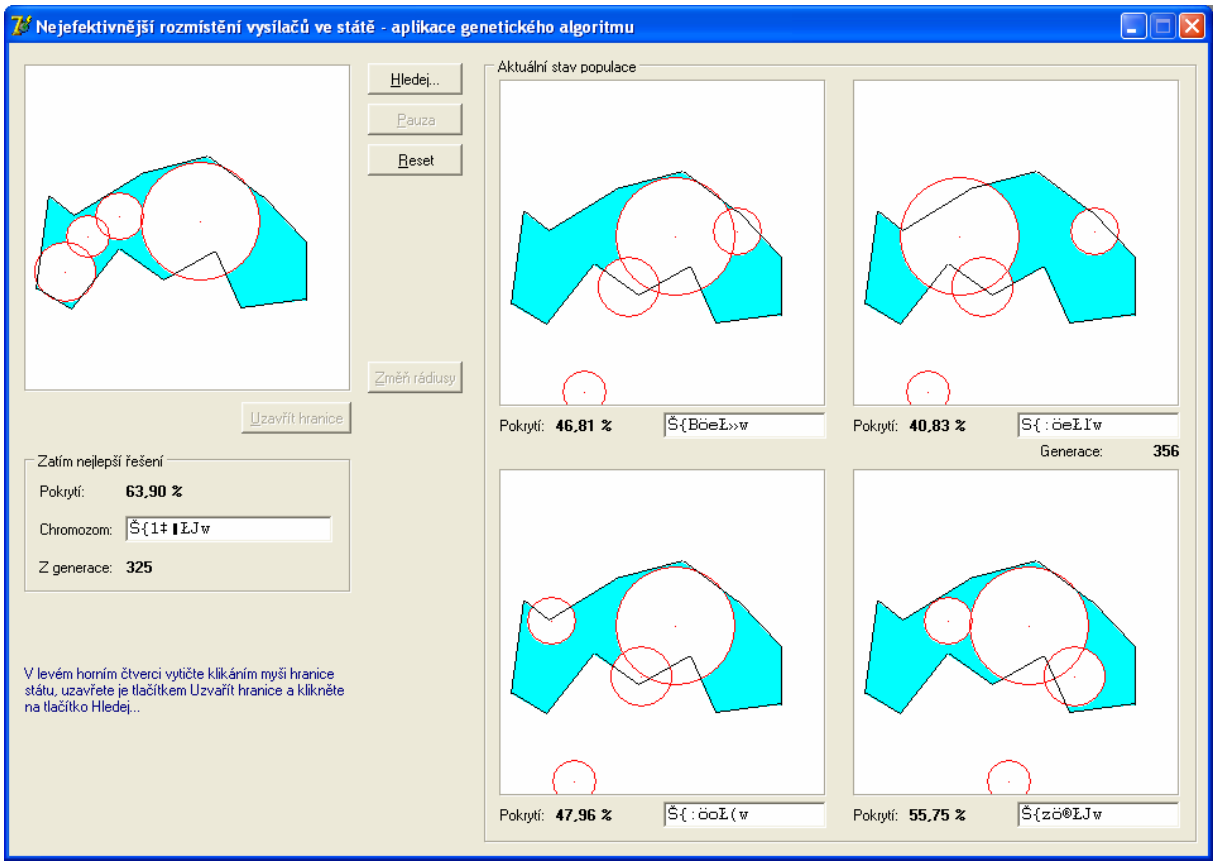
Obsah

1. Úvod

Pro aplikaci genetického algoritmu jsem si zvolil problematiku optimálního rozmístění vysílačů signálu (ať již televizního či pro mobilní telefony) na dané ploše tak, aby pokryly co možná největší území státu definovaného hranicemi. Vzhledem k obrovskému počtu možností i ve velmi omezené a zjednodušené verzi příkladu považuji tuto úlohu za typickou pro řešení pomocí genetického algoritmu. Rozhodl jsem se proto vytvořit program, který by názorně zobrazoval průběh tohoto hledání a pokud možno dosahoval i nějakých dobrých výsledků.

2. Uživatelská dokumentace

V této části je nastíněn způsob ovládání a fungování programu.



2.1. Zadání

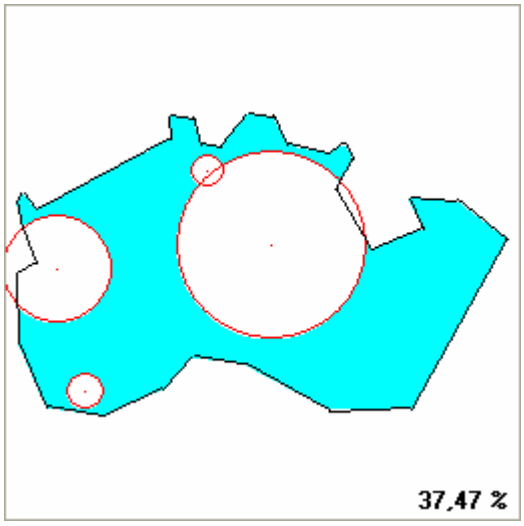
Účelem programu je pomocí genetického algoritmu hledat a nalézt co nejefektivnější rozmístění vysílačů na dané ploše tak, aby svým signálem pokryly co možná největší procento plochy státu vytyčeného hranicemi.

2.2. Ovládání

Po spuštění programu je na první pohled jediné aktivní tlačítko Změň rádiusy, které sice funguje, ale jeho výsledky nejsou zatím užitečné. První, co je totiž potřeba udělat je definovat hranice „státu“, jehož plochu se bude program snažit v co největší míře pokrýt.

Definování hranic státu se provádí ve čtverci situovaném do levého horního rohu okna. Prvním kliknutím do prostoru tohoto čtverce se vytvoří první bod hranic, po druhém kliknutí se již tyto dva body spojí a tak dále se bude pokračovat spojením úsečkou vždy místa kliknutí a místa kliknutí předchozího (konce poslední úsečky).

Poslední úsečku – spojnici mezi koncem a začátkem propojení všech úseček – a tím i uzavření hranic a vymezení prostoru státu je možné provést pouze kliknutím na tlačítko Uzavřít hranice.



Poté se okamžitě vygeneruje a vykreslí výchozí náhodná populace do čtyř čtverců v pravé části okna. Vlevo nahoře se vyobrazí nejlepší z nich. Pod každým čtvercem s vyobrazením rozmístění vysílačů je uvedeno procentuální pokrytí státu. Hranice státu jsou vyznačeny černými úsečkami, umístění vysílačů červenými body, jejich rádius (dosah) červenými kružnicemi a signálem nepokrytá plocha státu světle modře. Vše ostatní zůstává bílé.

Od tohoto okamžiku je také zpřístupněno tlačítko Reset, kterým můžete změnit toto počáteční náhodné rozmístění na jiné náhodně vygenerované. Také je již i srozumitelný efekt tlačítka Změň rádiusy, které opět náhodně změní rádiusy jednotlivých vysílačů. Všechny tyto změny jsou okamžitě vyobrazeny do vykreslené populace i do nejlepšího z nich.

Kliknutím na tlačítko Hledej... je spuštěn proces samotného vyhledávání nejefektivnějšího rozmístění vysílačů, aby pokryly co největší plochu státu.

2.3. Průběh hledání

Průběh hledání probíhá přímo před očima uživatele. Jak již bylo řečeno, v pravých čtyřech čtvercích se vyobrazuje aktuální testovaná populace, v levém horním čtverci je pak vždy vyobrazeno doposud nejlepší nalezené řešení. Pod každým takovýmto vyobrazením je hodnota procentuálního pokrytí plochy státu při aktuálním rozmístění vysílačů. Také je zde „chromozom“, neboli popis daného rozmístění zakódovaný do textového řetězce.

U dosud nejlepšího rozmístění je také uvedeno, z které generace toto řešení pochází, u aktuální populace je zase vpravo uprostřed číslo generace, která je právě testována.

Vyhledávání je možné pozastavit tlačítkem Pauza. Poté jej lze opět znovu spustit v místě kde přestal tlačítkem Hledej..., nebo tento stav vyresetovat do náhodného počátečního rozmístění tlačítkem Reset. Je-li stav vyresetován, je možné opět spustit vyhledávání od nulté generace, změnit rádiusy vysílačů, nebo vytyčit nové hranice státu postupem uvedeným v předchozí kapitole.

Průběh vyhledávání automaticky končí dosažením stoprocentního pokrytí (je-li takového stavu vůbec možné dosáhnout), o čemž uživatele informuje dialog.

3. Programátorská dokumentace

V této kapitole je podrobněji rozebrán způsob a technologie řešení celého problému, spolu s podrobným vysvětlením celého zdrojového kódu a možnostmi jeho modifikace. Program byl vytvořen v programovacím jazyku Borland Delphi 7.

3.1. Zadání

Úkolem bylo aplikovat technologii genetického algoritmu na daný případ - hledání a nalezení nejefektivnějšího rozmístění vysílačů na ploše.

Příklad byl pro tento ukázkový program zjednodušen na tyto podmínky:

- plocha pro řešení má rozměr 256x256 pixelů (pro snadné zakódování souřadnic)
- výměr státu je dán uzavřenou soustavou úseček
- vysílače mohou stát i mimo území státu
- pokrytí vysílače přesně vymezuje jeho rádius
- abstrahuje se od třetího rozměru (neexistují kopce, údolí, stromy ani nic, co by mohlo rušit či posilovat signál vysílače)

3.2. Struktura prvků

Prvky programu jsou unity, použité objekty a použité funkce.

3.2.1. Unity

Program se skládá celkem ze tří unit (souborů zdrojového kódu) a jsou to:

- frm_Main.pas - Hlavní unit programu, obsahující obslužné funkce jediného okna programu. Stará se o definici hranic, obsluhu událostí a tlačítek a vykreslování stavů.
- unt_GenAlg.pas - Obsahuje určení typů jednotlivých objektů vstupujících do algoritmu, konstanty s nastavením parametrů a funkce algoritmu.
- unt_Pokryti.pas - Obsahuje funkce pro výpočet a podporu vykreslování plochy státu nepokryté signálem (funkce hodnotící dané řešení).

3.2.2. Objekty

Jednotlivé objekty algoritmem používané jsou brány jako pole typů (recods). Jejich struktura je následující:

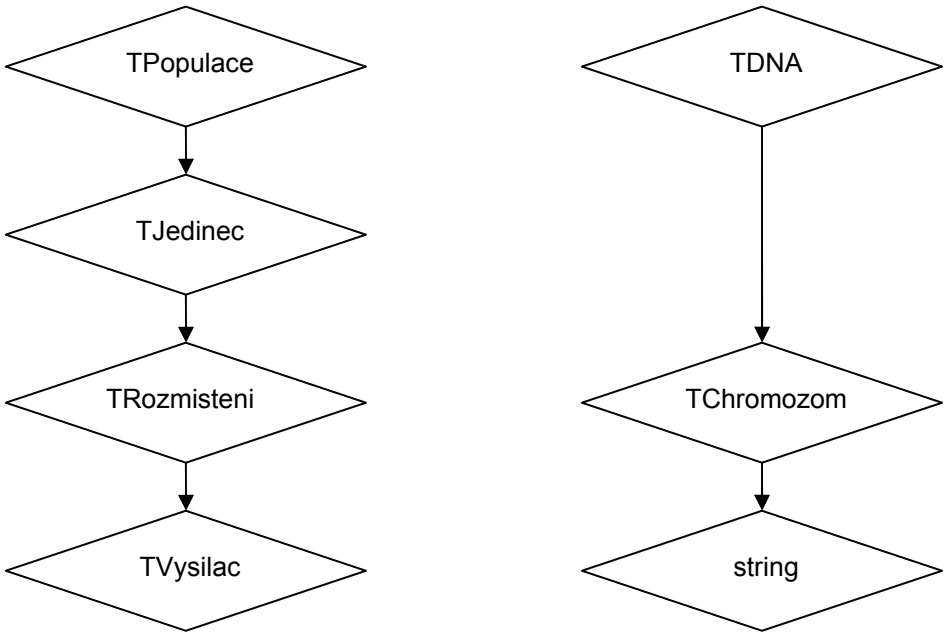


Schéma postupuje od nejvyššího řádu (populace / DNA) k nejnižšímu (TVysilac / string). Popišme si tedy význam těchto jednotlivých typů:

TVysilac obsahuje pouze informaci o poloze vysílače (X, Y). Jeho rádius je uložen zvlášť v proměnné **Radiusy** s indexem jeho pořadí.

TRozmisteni obsahuje pole objektů typu TVysilac, tedy rozmístění všech vysílačů. Navíc je v něm uložen chromozom (**TChromozom**), který tento stav kóduje a pokrytí, což je počet pixelů, které z prostoru státu toto rozmístění pokrývá.

TJedinec k samotnému rozmístění přidává pomocnou proměnnou „Šance na přežití“, která určuje šanci vstupu do reprodukce v dalším cyklu algoritmu.

TPopulace je pak polem těchto jedinců a určuje stav dané generace. Stav populace lze zakódovat do pole chromozomů, kteréžto má typ **TDNA**.

3.2.3. Funkce

Funkce, které jsou zajímavé pro algoritmus se nacházejí pouze v unitách unt_GenAlg.pas a unt_Pokryti.pas. Jedná se o tyto funkce:

Unit unt_GenAlg.pas

- **function ChromozomToRozmisteni(const Chromozom: TChromozom; const Index: integer = -1) : TRozmisteni;**

  Rozkóduje chromozom do typu **TRozmisteni** a pomocí funkce **SpocitejPokryti** nastaví i jeho pokrytí.

- **function DNAToPopulace(const DNA: TDNA) : TPopulace;**

  Převede populační DNA řetězec na **TPopulace** a každému jednotlivci vypočte jeho šanci pro přežití.

- **function UzByl(const Chromozom: TChromozom) : boolean;**

  Vrátí informaci o tom, zda se již daný chromozom někdy v minulosti vyskytl.

- **function IndexVyvoleneho(const Verditk: real; Populace: TPopulace) : integer;**

  Ze zadaného „verdiktu“ (číslo od 0 do 1) projde danou populaci a vrátí index toho jedince, který byl tímto verdiktem „vylosován“.

- **procedure Mutace(var Chromozom: TChromozom);**

  Zmutuje chromozom (viz. kapitola 3.5.1).

- **procedure Krizeni(var Chromozom1, Chromozom2: TChromozom);**

  Zkříží dva chromozomy (viz. kapitola 0).

- **function Reproduction(const DNA: TDNA) : TDNA;**

  Provede reprodukci DNA řetězce do nové generace (viz. kapitola 3.3).

- **function NahodnyChromosom: TChromozom;**

  Vygeneruje náhodný chromozom.

- **function NahodneDNA: TDNA;**

  Vygeneruje náhodný DNA řetězec (TDNA – pole chromozomů).

- **procedure Reset;**

  Nastaví výchozí podmínky algoritmu: vygeneruje náhodné výchozí DNA, vynuluje nejlepší nalezené rozmístění, vynuluje číslo generace a vymaže historii.

- **procedure Search;**

  Zahájí vyhledávání. V jeho cyklu probíhá vyhledávání dokud není nalezeno stoprocentní pokrytí nebo není algoritmus pozastaven uživatelem.

Unit unt_Pokryti.pas

- **procedure UseckyNaHranice(SeznamUsecek: TUsecky; PocetUsecek: integer);**

  Převede zadané pole úseček na hranice státu a toto nastavení uloží.

- **function SpocitejPokryti(const Rozmisteni: TRozmisteni; const Vykreslit: boolean = false; const IndexPlochy: integer = -1; const BezSnimacu: boolean = false) : integer;**

  Spočítá pokrytí daného rozmístění (viz. kapitola 3.6). Navíc podle vstupních parametrů umožňuje toto pokrytí vykreslit.

3.3. Reprodukce

Reprodukce (funkce **Reproduction**) přemění jeden řetězec DNA na jiný – na jeho novou generaci. Postup reprodukce je následující:

DNA řetězec je nejprve rozkódován a převeden na typ **TPopulace**. Z této populace jsou náhodně vybráni jedinci (každý má šanci na výběr podle toho, jak je jeho rozmístění úspěšné – kolik procent plochy státu pokrývá), kteří vytvoří novou populaci. Tato nová populace je rovnou zapsána do nového DNA řetězce.

Nový DNA řetězec nyní projde samotnou reprodukcí. Vždy dva sousední chromozomy (1. s 2., 3 se 4....) jsou zkříženy. Pokud byly do tohoto páru vybráni dva stejné chromozomy, je jeden z nich zmutován. V případě lichého počtu je poslední co zbyl zmutován.

Nyní se ještě prověří jednotlivé chromozomy celého řetězce, nebyl-li již některý z nich použit v předchozí generaci, pokud ano, je tento zmutován. Následně jsou všechny chromozomy zaznamenány do „kroniky“ (řetězce, který zaznamenává jejich existenci a tím zajišťuje, aby se již příště nevyskytly ve stejné podobě).

3.4. Zakódování chromozomu

Chromozomy jsou díky velikosti pole (256x256) zakódovány v textovém řetězci (string). Souřadnice vysílače jsou zde pak zaznamenány jako dva znaky tohoto řetězce. Jeden znak totiž odpovídá hodnotě mezi 0 a 255, tedy určuje jednu souřadnici jednoho vysílače. Souřadnice všech vysílačů jsou pak v tomto řetězci seřazeny od prvního po poslední, vždy po dvou znacích.

Ě	Ć	b	i	¤	x	'	)
X1	Y1	X2	Y2	X3	Y3	X4	Y4

DNA je pak polem (množinou) několika těchto chromozómů a kóduje v sobě celou populaci.

Ě	ž	¸	\	i	l	q	-
R	/	b	\	9	l	q	A
d	/	š		i	l	?	-
d	/	¸		i	□	q	-

3.5. Operace s chromozomy

Během reprodukce populace dochází k transformaci chromozomů jednotlivých jedinců. Tyto transformace jsou dvojího typu: mutace a křížení.

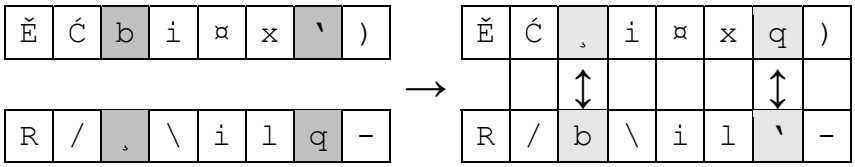
3.5.1. Mutace

Mutace je realizována náhodným vybráním určitých (defaultně jednoho) znaků řetězce (tedy jedné souřadnice jednoho vysílače), který je nahrazen novým, náhodně vybraným znakem.

Ě	Ć	b	i	¤	x	'	)
↓							
Ě	Ć	3	i	¤	x	'	)

3.5.2. Křížení

Křížení probíhá náhodným vybráním několika (defaultně dvou) znaků chromozomu, kteréžto jsou prohozeny se stejným „genem“ (znakem) chromozomu druhého.



3.6. Ohodnocení rozmístění

Funkce **SpocitejPokryti**, z daného rozmístění vysílačů spočítá pixely, které na ploše státu pokrývá signál vysílačů. Implementace této funkce byla na tvorbě celého programu tou nejobtížnější.

Kvůli zvýšení rychlosti algoritmu neprobíhá testování každého pixelu plochy zvlášť, ale je zde aplikován scan-line algoritmus, vylepšený o řešení problematiky kružnic a možnost vícenásobného překrytí signálu – dělení pásem. Algoritmus pro zjednodušení postupuje obráceně – spočítá plochu státu, která pokryta není a tuto hodnotu pak odečte od celkové plochy státu.

3.7. Vylepšení

Nejvýznamnější vylepšení běžného genetického algoritmu spočívá v možnosti návratu nejlepšího dosud nalezeného jedince do populace. Ta probíhá následujícím způsobem:

Při dekódování DNA řetězce do populace je z ní náhodně vybrán jeden jedinec. Pokud je hodnota jeho pokrytí výrazně nižší (defaultně 120% jeho hodnoty) než hodnota doposud nejlepšího jedince, dostává tento nejlepší šanci (defaultně 10%) zaujmout místo tohoto slabého jedince a vnést tak silné geny pro reprodukci nové generace.

3.8. Možnosti nastavení

Většina nastavení je možná pouze na úrovni zdrojového kódu a týká se nastavovaných konstant v jednotlivých unitách. Jejich přehled a význam je následující:

Unit frm_Main.pas

- **ciMaxRaius** – maximální možná hodnota, jakou lze náhodně vygenerovat jako rádius vysílače. Defaultně 80 pixelů. Minimum je 5.

Unit unt_GenAlg.pas

- **ciVysilacu** – Počet rozmísťovaných vysílačů. Defaultně 4.
- **ciPopulace** – Počet jedinců v populaci. Defaultně 4 (při vyšším počtu nebudou zbylí vykreslováni).
- **ciMutovatGenu** – Počet genů (bytů), které budou v chromozomu změněni při procesu mutace. Defaultně 1.
- **ciKrizitGenu** – Počet genů (bytů), které budou prohozeny dvěma chromozomům při procesu křížení. Defaultně 2.
- **ciSilaDanaGeneraci** – Ke kolika procentnímu navýšení hodnoty pokrytí dojde u jedince, jehož síla je porovnávána se silou zatím nejlepšího jedince. Defaultně 1,2.
- **ciSanceNaNavratChampiona** – Je-li doposud nejlepší jedinec výrazně silnější (ve smyslu pokrytí), než-li jedinec ze současné generace, který je s tímto porovnáván, jaká je šance, že bude tento slabý jedinec nahrazen tímto doposud nejsilnějším? Defaultně 0,1.

Unit unt_Pokryti.pas

- **ciPauzaMeziGeneracemi** – Pauza (v milisekundách), po kterou si uživatel může prohlédnout vykreslenou populaci, než dojde k vytvoření nové populace. Defaultně 100 (záleží na rychlosti PC a na tom, jak moc podrobně chce uživatel proces studovat).
- **cbVykreslovatPopulaci** – Má se vůbec populace vykreslovat? Pokud ne a se současným nastavením výše uvedené čekací pauzy na 0 lze výrazně urychlit průběh hledání a omezit tak svou pozornost pouze na jeho výsledky. Defaultně true.

4. Poznatky

Během tvorby programu a při jeho následném ladění došlo ke zjištění několika poznatků o genetických algoritmech.

Předně nelze nikdy s jistotou tvrdit, že algoritmem nalezené řešení je tím nejlepším. To lze ověřit jedině vyzkoušením všech existujících možností, ovšem, je-li toto v přijatelném čase možné, není třeba aplikace genetického algoritmu.

Nalezený výsledek by měl vždy ještě zkontrolovat člověk (odborník), protože někdy je již na první pohled patrné, že ač nalezené řešení je velmi dobré, posunutím vysílače o pixel či dva na určitou stranu bude dosaženo výsledku ještě lepšího.

Nastavení parametrů algoritmu je třeba vždy dobře promyslet a odzkoušet, protože to může případ od případu výrazně zvýšit či snížit celkovou efektivnost.

Použití heuristik by mohlo značně zvýšit efektivnost algoritmu.

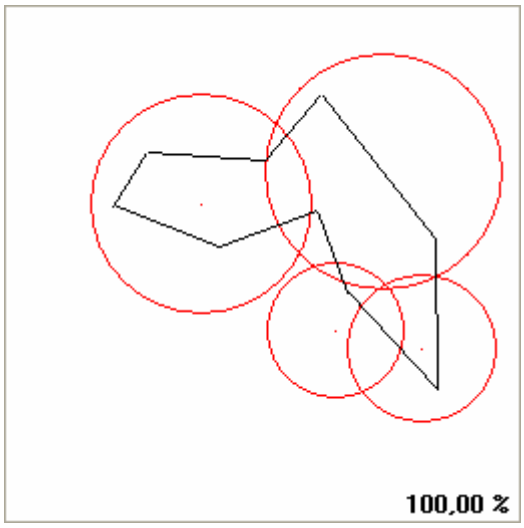
Většina řešení nalezená v první třicítce generací se již v budoucnu zlepší pouze nepatrně.

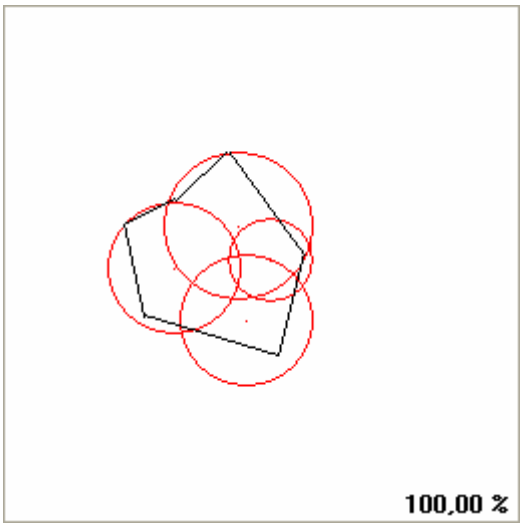
4.1. Konečná řešení

Existuje několik typů „konečného“ řešení algoritmu. Tyto jsou probrány níže.

4.1.1. Stoprocentní úspěch

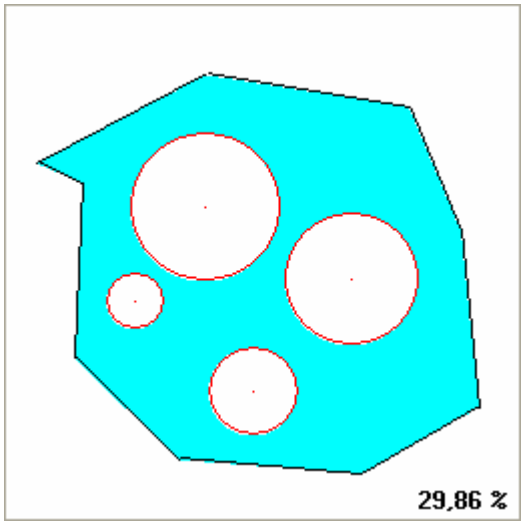
Vysílače se podařilo rozmístit tak, že je pokrytí plochy státu stoprocentní. Řešení se stejným výsledkem existuje (pokud vůbec existuje) většinou více, možná i s použitím méně vysílačů, algoritmus se však zastaví na prvním takovém, neboť není schopen nějak dále porovnávat jejich efektivnost.

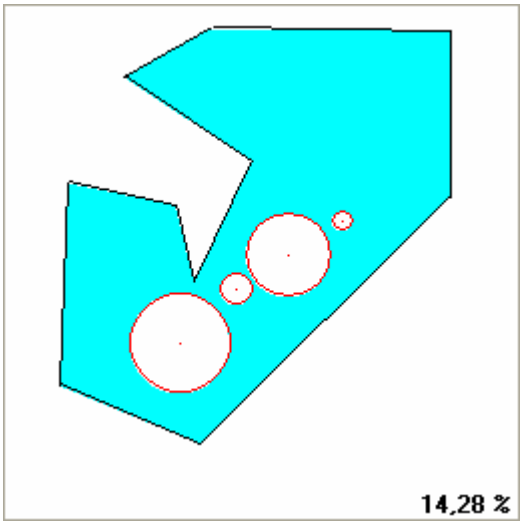




4.1.2. Lepší už to nebude

Vysílače byly rozmístěn tak, že každý má celý svůj rádius uvnitř státu a vzájemně se neprolínají. Lepšího pokrytí již na první pohled dosáhnout nelze, i když řešení se stejným výsledkem existuje jistě spousta. Nicméně algoritmus v tomto případě nepozná, že dosáhl globálního maxima a pokračuje v hledání dál. Vzhledem k tomu, že je použita pouze neostrá nerovnost v porovnávání výsledků, bude algoritmus dále nacházet a vykreslovat další možná rozmístění vysílačů, nicméně už pouze vždy se stejným procentem pokrytí jako je to v současnosti nejlepší.



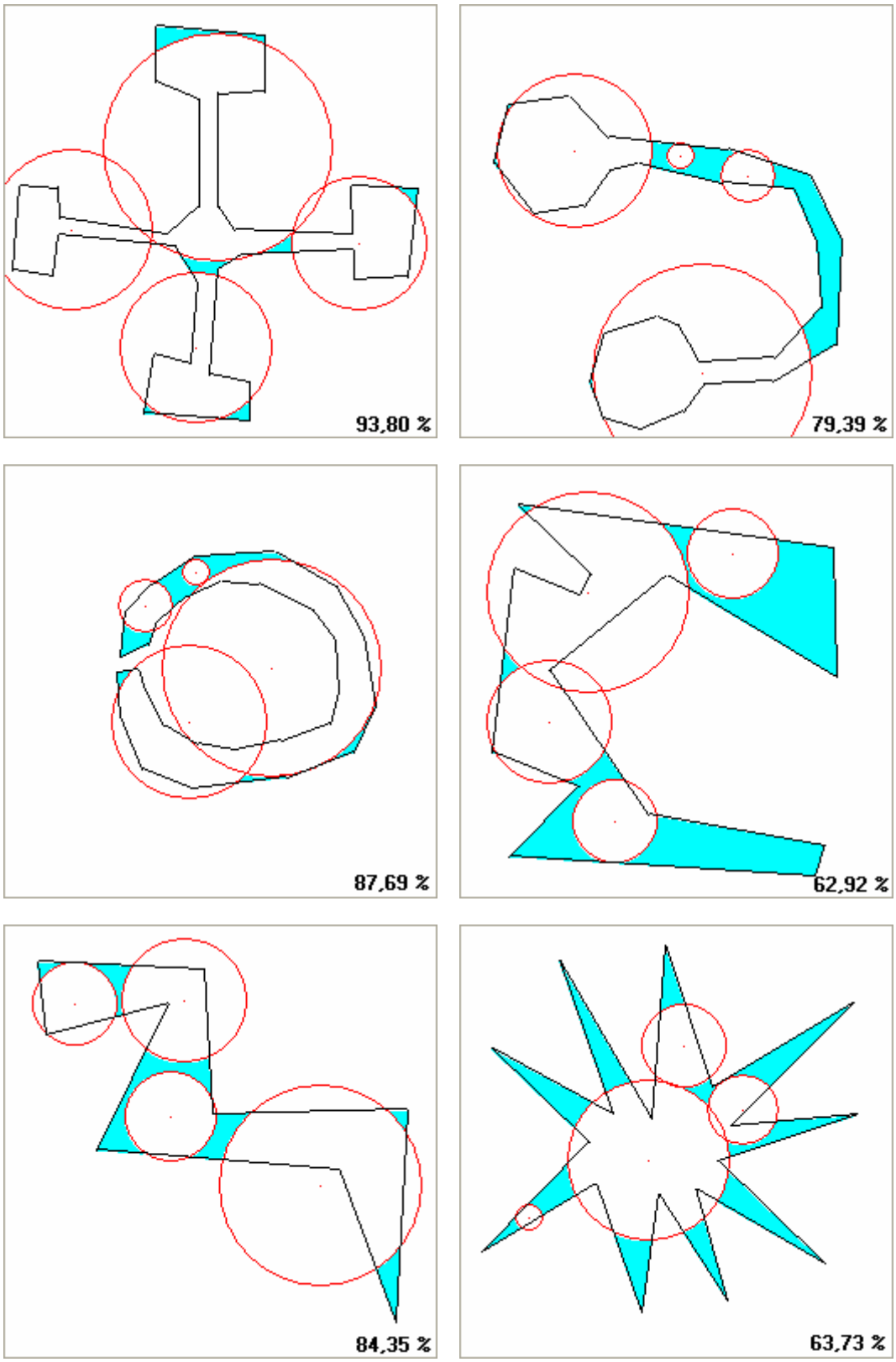


4.1.3. Je toto skutečně nejlepší?

V některých případech nelze dosáhnout předchozího stavu, protože třeba nejméně jeden z vysílačů má v jakékoli poloze rádius až za hranice státu, přičemž však stoprocentního pokrytí dosáhnout nelze (alespoň tedy ne na „první“ pohled). Zde teprve genetický algoritmus nalezne své uplatnění, protože nejlepší řešení není pro člověka na první pohled patrné. U takto zjednodušené úlohy je sice možné dojít k nejlepšímu možnému řešení výpočtem, nicméně i tento by byl již tak dost složitý a v reálných podmínkách zřejmě téměř nemožný, či pracnější než-li samotný genetický algoritmus. Zkoušení řešení hrubou silou je zřejmě v celku marná snaha, neboť již u takto zjednodušeného příkladu existuje (256^2)^4, tedy 18446744073709551616 možných řešení.

Nalezené řešení sice nelze prohlásit za nejlepší možné, ovšem může posloužit jako řešení konečné (pro realizaci), nebo být inspirací pro další modifikaci výsledků či podmínek řešení (přidat další vysílač, posílit signál, oželet pokrytí v neobydlených oblastech...).

5. Ukázky



6. Závěr

Aplikace genetického algoritmu na zvolenou problematiku se podařila a vzniklý program celkem bez problémů funguje a dosahuje i dobrých výsledků, které by dalšími modifikacemi a vylepšeními šlo jistě ještě o něco zvýšit.

6.1. Možné vylepšení algoritmu

Pro přiblížení k reálnému stavu by bylo možné vylepšit algoritmus tak, aby počítal například s třetím rozměrem (kopce, údolí, stromy, budovy, překážky...), umožňoval nastavovat výšku vysílačů a tím zvyšovat jejich dosah v dané lokalitě. S tím by souviselo i zahrnutí ceny (ceny vysílače daného dosahu, výšky síly, ceny za pronájem daných lokalit, možnost využití existujících komínů, kopců, stožárů...) do výpočtu ohodnocení atd.

6.2. Heuristiky

Mezi možná vylepšení efektivnosti algoritmu patří také využití heuristik. Například mutovat přednostně geny vysílače s nejmenším pokrytím, zkoušet jemnější „pošupování“ místo hrubé mutace, zařadit výpočet středu větších ucelených ploch, či jinak pomoci náhodě a uspíšit tak vývoj k nejlepšímu výsledku.