MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Genetické algoritmy

LOKÁLNE PREHĽADÁVANIE A OPTIMALIZAČNÉ PROBLÉMY

Matúš Goljer

Brno, 2010

Obsah

1 Optimalizačné problémy

Optimalizačné problémy označujú triedu problémov ktorých cieľom je nájsť najlepšie riešenie z množiny prípustných riešení. Nad každým stavom[1] je definovaná *účelová funkcia* $c(s) : \mathbb{D} \to \mathbb{R}$, ktorá stavu z domény priradí jeho cenu (hodnotu). Úlohou optimalizačného algoritmu je potom nájst riešenie a maximalizovať (príp. minimalizovať) jeho cenu.

1.1 Prehľadávanie hrubou silou

Najjednoduchšia a priamočiara metóda na hľadanie optimálneho riešenia problému je systematicky prehľadať celý stavový priestor. Prehľadávací algoritmus postupne generuje všetky prípustné riešenia, ktorých cenu porovnáva s doposiaľ najlepším nájdeným riešením. Táto metóda nám zaručuje úplnosť, to znamená, že vždy nájde najlepšie možné riešenie. Algoritmus 1 ukazuje kostru prehľadávania hrubou silou. Funkcia NEXT určuje konkrétnu stratégiu prehľadávania stavového priestoru (BFS, DFS...). Funkcia VALID kontroluje či vygenerovaný stav skutočne reprezentuje riešenie. Funkcia COMPARE porovná dva stavy pomocou účelovej funkcie COST.

Hlavnou nevýhodou tohoto prístupu je, že v prevažnej väčšine reálnych problémov existuje obrovské množstvo riešení, prípadne stavov ktoré treba skontrolovať. Keby sme napríklad chceli hrubou silou prelomiť heslo zložené z troch znakov obsahujúce len čísla 0 až 9, museli by sme v najhoršiom prípade otestovať všetkých 10^3 možností. Pridaním ďalších troch znakov by počet možností narástol na 10^6. Každým ďalším znakom sa celkový počet možností rádovo zväčsuje. Tento fenomén sa nazýva kombinatorická explózia, a prakticky obmedzuje použitie triviálneho prehľadávania na veľmi malé problémy.

```
output: Najlepšie riešenie, prípadne NIL ak riešenie neexistuje
best ← ⊥;
while s ← Next(𝔻) do
    if Valid(s) then
        if Compare(s,best,Cost) then best ← s;
    end
end
return best;
```

Algoritmus 1: Kostra algoritmu prehľadávania hrubou silou

1.2 Lokálne prehľadávanie

Lokálne prehľadávanie je metaheuristika na riešenie náročných optimalizačných problémov[1]. Nad stavovým priestorom optimalizovaného problému musí byť definovaná relácia *byť susedom*, kde dva stavy sú susedné, ak sa líšia iba malou lokálnou zmenou. Ak si stavový priestor predstavíme ako n-rozmerný priestor, kde osi tvoria premenné problému, susedné stavy sa potom budú

[1] Stav označuje nejaké priradenie premenných v probléme. Nemusí sa nutne jednať o riešenie alebo platné priradenie.

líšit hodnotou premennej na jednej ose[2]. Aby bol algoritmus efektívny, musia byť tieto susedné riešenia jednoducho generovatelné.

Narozdiel od úplného prehľadávania lokálne prehľadávanie začína s nejakým dopredu zvoleným stavom a postupne prehľadáva jeho susedné stavy. Ak je nový stav vyhovujúci, pokračujeme ďalej z tohoto nového stavu. Prehľadávanie sa typicky ukončí po danom časovom limite alebo v prípade, že po určitý počet krokov nebolo nájdené žiadne zlepšujúce riešenie. Lokálne prehľadávanie nie je úplné, pretože môže skončit aj v prípade, že najlepšie nájdené riešenie nie je optimálne. To môže nastať v prípade, ak optimálne riešenie leží ďaleko od prehladávanej časti stavového priestoru. V praxi sa však často akceptuje riešenie splňujúce istú spodnú hranicu, preto bezpodmienečné nájdenie globálneho optima nie je nevyhnutné.

```
input  : Iniciálne riešenie a počet krokov, po ktorých prehľadávanie skončí
output: Najlepšie nájdené riešenie, prípadne počiatocný stav ak riešenie neexistuje
steps ← 0;
best ← init;
while steps < limit do
    s ← NextNeighbour(best);
    if Accept(s,best,Cost) then
        best ← s;
    end
    steps ← steps +1;
end
return best;
```

Algoritmus 2: Kostra algoritmu lokálneho prehľadávania

Konkrétnu stratégiu opäť určujú vyznačené funkcie. Najtriviálnejšia implementácia je tzv. Hill-climbing, kde funkcia ACCEPT akceptuje vzdy riešenie s lepšou cenou ako má aktuálne najlepšie riešenie. To znamená, že sa vždy snaží stúpať do kopca[3]. Akonáhle však dosiahne lokálne optimum, všetky okolné stavy sú horšie, a prehľadávanie tu uviazne (viď Obr. 1). Táto heuristika sa preto hodí na hľadanie zlepšujúcich stavov, ktoré neležia ďaleko od počiatočného stavu. Výber počiatočného stavu teda môže mať pomerne veľký dopad na kvalitu nájdeného riešenia. Hill-climbing je preto výhodné kombinovať s ďalším prehľadávaním, kde sa pre každý stav nájdený "vonkajším" cyklom spustí limitované lokálne prehľadávanie.

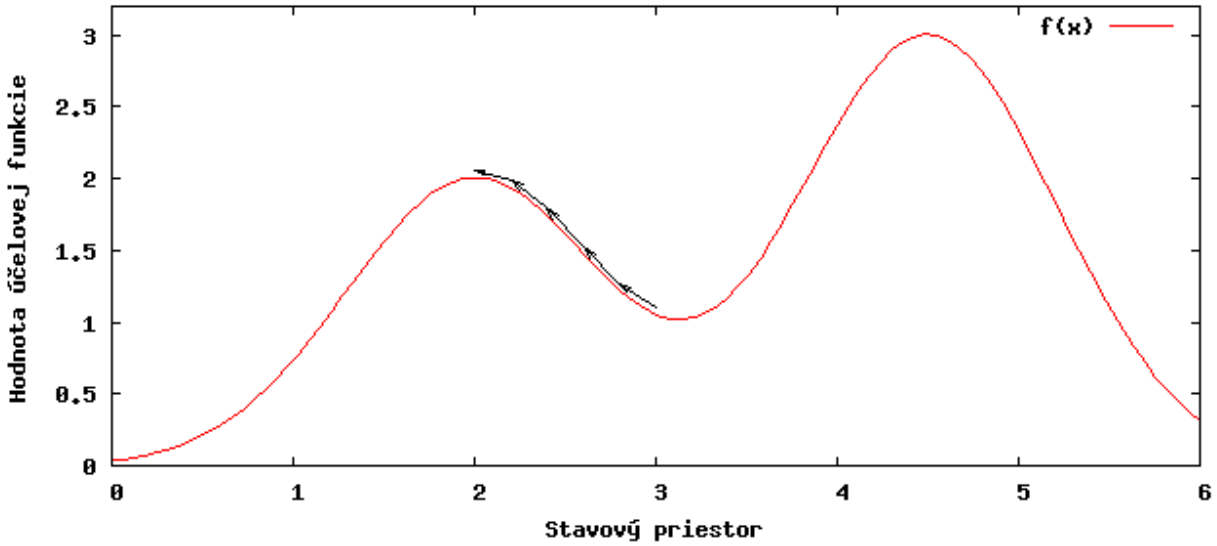
Problém uviaznutia riešia rôzne ďalšie metódy ako simulované žíhanie[4], tabu prehľadávanie[5] alebo *genetické algoritmy*, ktoré popisuje nasledujúca kapitola.

[2] V prípade, že neuvažujeme diskrétny problém, zmena na ose môže byť definovaná konštantou c

[3] V prípade, že funkciu minimalizujeme z kopca zostupujeme.

[4] http://en.wikipedia.org/wiki/Simulated_annealing

[5] http://en.wikipedia.org/wiki/Tabu_search



Obr. 1: Konvergencia riešenia k lokálnemu maximu.

2 Genetické algoritmy

Genetické algoritmy generujú riešenia zložitých problémov aplikáciou evolučných princípov ako dedičnosť, rekombinácia, mutácia a prirodzený výber. [4][2]

2.1 Úvod do genetiky

Každý organizmus obsahuje sadu pravidiel ktoré kódujú spôsob akým je tento organizmus zostavený. Tieto pravidlá sú zakódované v *génoch*, ktoré sú spolu prepojené do dlhých reťazcov nazývaných *chromozómy*. Každý gén reprezentuje špecifickú vlastnosť daného organizmu, napríklad farbu očí alebo vlasov, a má viacero rôznych hodnôt. Gény a ich hodnoty sú označované ako *genotyp*. Fyzické pozorovatelné vlastnosti jednotlivého individua (teda výsledok "aplikácie" genotypu) sa nazývajú *fenotyp*.

Ked sa dva organizmy spária, výsledný potomok môže obsahovať časť génov od jedného rodiča, a časť od druhého. Tento proces sa nazýva rekombinácia. Občas sa tiež stane, že určitý gén zmutuje. Normálne tieto mutácie nemajú na vývoj fenotypu žiaden pozorovatelný vplyv, ale niekedy sa môžu prejaviť ako zcela nové rysy.

2.2 Popis algoritmu

V genetických algoritmoch sa populácia reťazcov (chromozómov), ktoré kódujú možné riešenia (fenotypy), *vyvíja* smerom k lepším riešeniam. Tradične sú chromozómy reprezentované binárnym reťazcom núl a jedničiek, avšak existujú aj iné prípustné reprezentácie závislé od typu problému (vektory, matice, stromy). Evolúcia začína s populáciou náhodne vygenerovaných jedincov a prebieha v generáciách. V každej generácii je u každého jedinca z populácie spočítaná *fitness* funk-

cia, ktorá udáva úspešnosť daného riešenia, inak povedané, šance dožiť sa reprodukčného veku a predať svoje gény potomkom. Na základe hodnoty fitness funkcie sú z populácie náhodne vybraný jedinci, pričom jedinci s vyššou hodnotou majú obecne väčšiu šancu na výber. Vybraný jedinci sú ďalej upravený (rekombinovaný a mutovaný) a tvoria nasledujúcu generáciu, ktorá je vstupom ďalšej iterácie algoritmu. Algoritmus končí, ked bol dosiahnutý určitý počet generácií, alebo bolo nájdené dostatočne vyhovujúce riešenie.

```
gen ← 0;
population ← GenerateInitialPop ();
while gen < limit do
    selectedPopulation ← Select (population,Fitness);
    selectedPopulation ← Crossover (selectedPopulation);
    selectedPopulation ← Mutation (selectedPopulation);
    population ← selectedPopulation;
    gen ← gen +1;
end
return Best (population,Fitness);
```

Algoritmus 3: Kostra genetického algoritmu

2.3 Inicializácia

Iniciálna populácia je tvorená náhodne vygenerovanými riešeniami. Veľkosť populácie je silne závislá na doméne problému, obecne ale obsahuje stovky jedincov. Vygenerované riešenia zvyčajne pokrývajú veľkú časť stavového priestoru, a tak znižujú šancu uviaznutia v lokálnom minime. Niekedy však môžu byť do populácie nasadený jedinci, ktorý by mohli smerovať k optimálnemu riešeniu a urýchliť tak proces evolúcie (podobne ako tvrdí teória inteligentného dizajnu).

2.4 Selekcia

V každej generácii je časť populácie vyčlenená k rozmnožovaniu a tvorbe novej generácie. Jednotlivý jedinci sú vybraný pomocou fitness funkcie, kde lepšie riešenia (v závislosti na fitness funkcii) sú typicky preferované. Niektoré metódy neohodnocujú celú populáciu, ale len náhodne vybranú časť, pretože výpočet fitness funkcie môže byť časovo náročná operácia. Väčšina selekčných algoritmov je stochastická, a navrhnutá tak, že spolu s najlepšími jedincami je vybraná aj určitá časť slabších jedincov, aby sa zachovávala genetická diverzita, ktorá zabranuje predčasnej konvergencii k lokálne optimálnym riešeniam.

2.4.1 Výber úmerný vhodnosti

Výber úmerný hodnote fitness funkcie, tiež známy ako roulette-wheel selection, priraďuje jedincom pravdepodobnosť výberu úmernú ich vhodnosťi. Ak f_i je vhodnosť i-teho jedinca v populácii, jeho pravdepodobnosť výberu je rovná $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$, kde N je počet jedincov v populácii.

Tento proces môže byť jednoducho modelovaný ruletovým kolesom v kasíne (odkiaľ plynie aj názov). Každý jedinec dostane priradený výsek úmerný pravdepodobnosti výberu p_i. Následne je vygenerované náhodné číslo ktoré určí rotáciu kolesa a tým aj vybraného jedinca. Napriek tomu, že šanca výberu nevhodného jedinca je malá, tento typ výberu túto možnosť narozdiel od niektorých jednoduchších výberov pripúšťa. Aj napriek tomu, že ďalšia populácia môže dočasne obsahovať menej vhodné jedince[6], ich výber môže byť výhodný pre ďalšiu rekombináciu a udržanie genovej diverzity.

2.4.2 Turnajový výber

Turnajový výber spúšťa niekoľko turnajov medzi k náhodne zvolenými jedincami z populácie. Výherca každého turnaja (jedinec s najlepšou hodnotou fitness funkcie) je vybraný pre ďalšiu generáciu, prípadne rekombináciu. Selekčný tlak môže byť jednoducho kontrolovaný veľkosťou vybranej vzorky. V prípade, že je vzorka veľká, slabší jedinci majú nižšiu šancu výberu. Ak zoradíme jedincov podľa vhodnosti od najlepšieho po najhoršieho, pravdepodobnosť výberu i-tého jedinca je daná vzťahom $p(1-p)^i$, kde p je pravdepodobnosť výberu najlepšieho jedinca. Turnajový výber bude deterministický (vždy vyberie najlepšieho jedinca zo skupiny) ak nastavíme pravdepodobnost p na 1. Ak je veľkosť turnaja rovná jednej, tento výber sa rovná náhodnému výberu.

2.4.3 Elitizmus

Elitizmus nie je priamo selekčná stratégia, ale heuristika, ktorá pred vlastnou selekciou skopíruje najlepšieho jedinca (alebo niekoľko najlepších) priamo do novej generácie. Toto môže rapídne urýchlit účinnosť genetického algoritmu, pretože zabráni strate najlepšieho doposiaľ nájdeného riešenia.

2.5 Rekombinácia

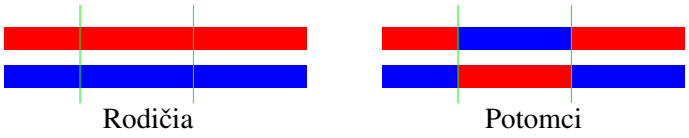
Rekombinácia je genetický operátor používaný na zmenu chromozómu jedinca z generácie na generáciu. Je analogický k biologickej rekombinácii, kde potomok zdedí časť chromozómu od každého z oboch rodičov. Rekombinácia v genetických algoritmoch však nie je limitovaná na dvoch rodičov, a obecne môže používať ľubovolný počet predkov a potomkov. V nasledujúcom texte budeme používat model s dvoma rodičmi a dvoma potomkami, ostatné modely sú analogické.

Najjednoduchšia varianta rekombinácie je jednobodová rekombinácia (vid Obr.2). Pred samotným procesom je zvolená náhodná konštanta $0 < i < L$ (L je dĺžka celého chromozómu), hraničný bod. Vsetky gény za týmto bodom sú u oboch rodičov vzájomne vymenené. Takto vytvorené nové chromozómy tvoria potomkov. Ďalšia varianta je dvojbodová rekombinácia (vid Obr.3), kde sa zvolia dva hraničné body, a gény medzi nimi sa vymenia medzi rodičmi.

Tieto jednoduché varianty fungujú len v prípade, že funkcie génov oboch rodičov si pozične odpovedajú. V prípade, že ako reprezentáciu chromozómu používame stromy alebo permutácie je potreba použiť zložitejšie rekombinačné metódy. V prípade permutácií by priama výmena častí chromozómu mohla spôsobiť duplikáciu niektorých génov. Rekombinácia teda prebieha tak, že

[6] Časti genómu môžu reprezentovať časť výborného riešenia, ktoré by inak bolo stratené.

Obr. 2: Jednobodová rekombinácia



Obr. 3: Dvojbodová rekombinácia

časť pred hraničným bodom je skopírovaná do potomka, a doplnená chromozómom druhého rodiča bez prvkov, ktoré už potomok obsahuje. Napríklad pre dvoch rodičov ABCDEFGHI a IGAHFDBEC a hraničný bod 4 budú potomci ABCDIGHFE and IGAHBCDEF. V prípade stromov si rodičia vymieňajú podstromy.

2.6 Mutácia

Hlavnou úlohou operátoru mutácie je udržovanie genetickej diverzity z jednej generácie chromozómov na ďalšiu. Mutácie by mali zabrániť genetickému algoritmu pred uviaznutím v lokálnych optimách a pred situáciou, kde je väčšina populácie príliž podobná (ich vzdialenosť v stavovom priestore je príliž malá). V takom prípade by totiž rekombinácia viedla k skoro rovnakým jedincom a evolúcia by sa prakticky zastavila.

Klasický operátor mutácie na bitových sekvenciách priraďuje každému génu (bitu) istú malú pravdepodobnosť zmeny jeho hodnoty na opacnú. Pre každý gén sa potom vygeneruje náhodné číslo z rozsahu nula až jedna, a v prípade, že je toto číslo menšie ako daná pravdepodobnosť je daný gén zmenený. Tento typ mutácie sa nazýva bodová mutácia. Ďalšie typy sú inverzia, kde je časť chromozómu prevrátená. U chromozómov kódovaných ako permutácie mutácie zahŕňajú výmeny dvoch členov postupnosti alebo zamiešanie viacerých prvkov. Pri stromoch to sú zmeny hodnôt v jednotlivých uzloch.

3 Zhrnutie

Optimalizačné problémy sú spravidla výpočetne zložité problémy, často NP-úplné, s obrovským stavovým priestorom. Úplné prehľadávanie je preto nevyhovujúce, často môže presahovať časové horizonty v rádoch tisícov rokov. Lokálne prehľadávanie sa snažia vyriešiť tento problém používaním heuristických metód. Jednou z nich je evolučné programovanie, kde počítačovo simulujeme biologickú evolúciu nad populáciou riešení. Z Darwinovho princípu prirodzeného výberu plynie, že lepšie najriešenia by mali pretrvávať v nasledujúcich generáciách. Táto metóda však nie je úplná a nezaručuje nám nález globálne optimálneho riešenia. Príkladom problémov, ktoré

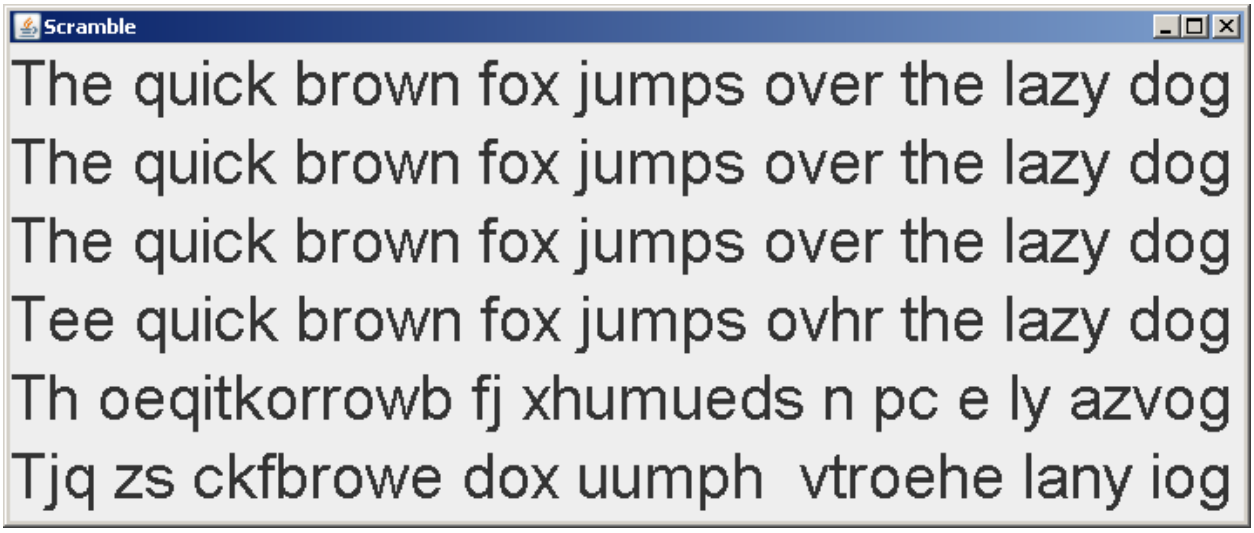
sú dobre modelovatelné genetickými algoritmami sú Problém obchodného cestujúceho (Traveling salesman problem - TSP), rozvrhovanie úloh, zarovnávanie sekvencií proteínov, návrh topológií bezdrôtových sietí alebo tréning neurónových sietí.

4 Aplikácie

4.1 Scramble

Scramble[7] je jednoduchá ukážka demonštrujúca silu genetických algoritmov a evolúcie pri hľadaní riešenia. Problém sa skladá z usporiadania "rozhádzaných" písmen do originálneho poradia. Samozrejme, v tomto prípade existuje triviálny zoraďovací algoritmus, avšak ten je silne problémovo závislý. V prípade geneticého algoritmu program o probléme nemá žiadnu predchádzajúcu informáciu a pracuje čiste na optimalizácii účelovej funkcie. Je to ukážka jednoduchosti používania frameworku evolúcie. Stačí len zvoliť správnu reprezentáciu chromozómu a definovať účelovú funkciu.

V prípade Scramble je chromozóm reprezentovaný ako permutácia znakov. Používa sa klasická rekombinácia pre permutácie tak, ako je popísaná v kapitole 2.5. Mutácie sú reprezentované prehodením dvoch znakov.



Obr. 4: Ukážka programu Scramble. V prvom riadku je zobrazený cielový reťazec. V riadkoch 2 až 6 sú zobrazený členovia populácie (zoradený podľa hodnoty fitness funkcie). Vidíme, že genetický algoritmus bol schopný nájsť riešenie po zhruba 30000 generáciách, zatiaľ čo všetkých možných usporiadaní je 43!.

[7] http://www.fi.muni.cz/~xgoljer/ai/scramble/

4.2 Problém obchodného cestujúceho

Problém obchodného cestujúceho je jeden z klasických a intenzívne študovaných optimalizačných problémov. Tento problém ma široké aplikácie od plánovania a logistiky po výrobu mikročipov alebo sekvencovanie DNA. Tento problém patrí do triedy NP-ťažkých problémov, čo mimo iné znamená, že neexistuje známe riešenie v polynomiálnom čase. Genetické algoritmy a iné heuristiky sú teda jediná možnosť ako tento problém riešiť.

Úlohou riešiteľa je nájsť najkratšiu okružnú cestu cez všetky zadané mestá. V reči grafov, hľadáme Hamiltonovskú kružnicu s čo najmenšiou váhou v ohodnotenom grafe. V predstavenom genetickom algoritme[8] je cesta reprezentovaná ako permutácia vrcholov grafu, kde poradie presne udáva poradie v akom cestujúci navštevuje mestá. Pri hľadaní riešenia sú použité rovnaké operátory ako pri predchádzajúcom príklade. Existuje však veľké množstvo ďalších operátorov, ktoré pracujú či už s permutáciou alebo priamo nad grafovou reprezentáciou[3]

4.3 Problém batohu

Problém batohu je ďalší z rady optimalizačných problémov. Máme zadanú množinu predmetov, u každého predmetu je známa jeho váha a cena. Úlohou je vybrat podmnožinu predmetov tak, aby sa zmestili do batohu (batoh uniesie len určitú hmotnostť) a aby sme maximalizovali cenu vybraných predmetov. Táto varianta je takzvaný 0-1 problém, jeho generalizácia potom pripúšťa výber nula až n predmetov z každého druhu (prípadne neobmedzený počet).

Týmto problémom sa dá modelovať veľké množstvo iných problémov, najmä z oblasti prideľovania zdrojov. Predstavme si napríklad firmu, ktorá dostane množinu zakázok. Každá zakázka vyžaduje určitú veľkosť tímu, a prinesie určitý zisk. Veľkosť tímu môžeme chápať ako váhu predmetu (s tým, že počet zamestnancov je celková nosnosť batohu), a zisk zo zakázky je cena. Snažíme sa prideliť zdroje (zamestnancov) do tímov tak, aby sme maximalizovali zisk.

V prezentovanej aplikácii[9] riešime 0-1 problém. Informácia o tom, či sme daný predmet vybrali je kódovaná ako binárny reťazec, kde 1 a 0 odpovedá výberu, resp. ponechaniu predmetu. Fitness funkcia je priamo spočítaná ako $\sum_{i=1}^{n} v_i x_i$, kde $x \in \{0, 1\}$ značí výber predmetu a v značí cenu predmetu. V prípade, že celková váha $\sum_{i=1}^{n} w_i x_i$ presiahne maximum, fitness funkcia vracia 0. Mutácia je implementovaná ako prepnutie bitu z 0 na 1, príp. obrátene. Rekombinácia je implementovaná jednobodovo.

4.4 Evolúcia auta

Táto[10] aplikácia ukazuje použitie genetických algoritmov na evolúciu "reálnych" strojov. V simulovanom prostredí s gravitáciou a trením sa populácia "áut" vyvíja k optimálnemu dizajnu na prekonanie zložitého terénu. Podobné aplikácie sa používaju v reálnom priemysle na návrh a dizajn tvaru lietadiel[5] alebo karosérií[6].

Táto ukážka okrem iného ukazuje, že kódovaná vlastnosti nemusia vždy reprezentovať informácie jedného typu. U obchodného cestujúceho to boli mestá, u problému batohu prislúchajúci

[8] http://www.obitko.com/tutorials/genetic-algorithms/tsp-example.php
[9] http://www.fi.muni.cz/~xgoljer/ai/knapsack/
[10] http://www.qubit.devisland.net/ga/

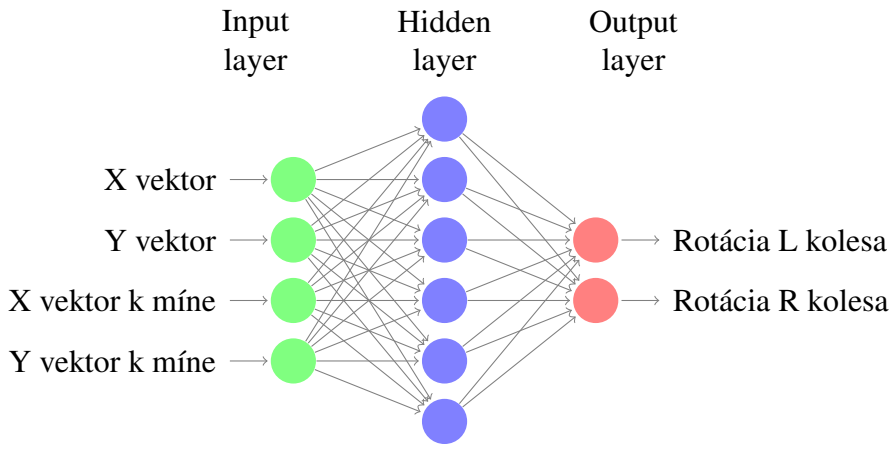
výber. V tomto prípade sa chromozóm simulovaného dvojrozmerného auta skladá z pozícií kolies a závaží, z ich priemeru a zo sily a dĺžky pružín. Je teda možné kódovať značne rozdielnu informáciu. Chromozóm je zakódovaný ako reťazec reálnych čísel. Namiesto klasickej rekombinácie sa používa náhodný vážený priemer, kde sa náhodne vygeneruje váha w z intervalu $(0, 1)$. Následne sa spočíta hodnota génu potomka ako $wg_1 + (1 - w)g_2$, g_1 a g_2 reprezentujú hodnoty rodičovských génov. Mutácia náhodne zmení hodnotu vybraného génu.

Ďalšia ukážka[11] simulovanej evolúcie "organizmov" odmeňuje jedincov na základe dopredu zvolenej úlohy ako uraziť čo najvačšiu vzdialenosť, sledovať objekt v simulovanom prostredí alebo obsadiť strategické zdroje.

4.5 Minesweeper

Minesweeper[12] je aplikácia kde sa v simulovanom prostredí odmínovacie roboty snažia pozbierať čo najvačší počet mín. Robot je riadený neurónovou sieťou (Obr. 5). Sieť má 4 vstupy, ktoré reprezentujú x a y zložky vektorov určujúcich smer robota a smer k najbližšej míne. Sieť ma dva výstupy ktoré ovládajú pásy robota (podobne ako na tanku). Váhy neurónovej siete sú na začiatku inicializované náhodne. Úlohou sieťe je počítať funkciu, ktorá upravuje smerový vektor tak, aby bol zhodný s smerom k najbližšej míne. V prípade, že sa dva vektory zhodujú, jednoducho pokračuje zvoleným smerom. Je zrejmé, že v tomto prípade sa jedná o pomerne jednoduchú fukciu, ktorú by bolo možné naprogramovať priamo. Pri iných zložitejších problémoch, napríklad (vzpriamený) pohyb humanoidného robota, kde je množstvo senzorov a parametrov, je nájsť správne váhy zložitý problém.

Sieť je reprezentovaná ako vektor reálnych čísel, ktorý je možné jednoznačne namapovať na neurónovú sieť (topológia sa s evolúciou nemení). Sú použité štandardné operátory mutácie a rekombinácie.



Obr. 5: Neurónová sieť ovládajúca správanie robota

[11] http://www.youtube.com/watch?v=oCXzcPNsqGA
[12] http://www.fi.muni.cz/~xgoljer/ai/minesweeper/

Literatúra

[1] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

[2] Introduction to genetic algorithms. `http://www.obitko.com/tutorials/genetic-algorithms/` (9.12. 2010), 1998.

[3] P. Larranaga, C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13:129–170, 1999. 10.1023/A:1006529012972.

[4] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[5] J. F. Wang, J. Periaux, and M. Sefrioui. Parallel evolutionary algorithms for optimization problems in aerospace engineering. *Journal of Computational and Applied Mathematics*, 149(1):155 – 169, 2002.

[6] K. Wloch and P. J. Bentley. Optimising the performance of a formula one car using a genetic algorithm. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. Tino, A. Kabán, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 702–711. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-30217-9_71.