Význam, uplatnění a návrhy na vylepšení smečkových algoritmů

Daniel Valenta, Petr Sosík

Slezská univerzita v Opavě, Filozoficko-přírodovědecká fakulta, Ústav informatiky
Bezručovo náměstí 13, 746 01 Opava 1
F180337@fpf.slu.cz, petr.sosik@fpf.slu.cz

Abstrakt

Článek se zabývá problematikou heuristických optimalizačních smečkových algoritmů inspirovaných především společenským soužitím vlků obecných a jejich schopností ve smečkách dynamicky vytvářet vlastní hierarchii, ve které má každý člen jasně stanovenou roli. Díky tomuto principu vlci projevují při plnění denních úkolů vyšší míru inteligence, než jaké by byl schopen kterýkoliv z nich dosáhnout samostatně.
Toto chování lze matematicky popsat, a tudíž simulovat v multiagentovém systému. Simulované chování se můžeme následně snažit optimalizovat podle předlohy reálných vlků.
Smečkové algoritmy mají využití zejména při řešení optimalizačních problémů. Již v současné době našly uplatnění v celé řadě odvětví, jako jsou například neuronové sítě, inženýrství, počítačové sítě, ekonomie, matematika, medicína, počítačová grafika, aj.
Článek je revizí již existujících publikací. Jeho cílem je vytvořit stručný přehled o dosavadních výsledcích v tomto oboru a opravit chyby, které původní publikace obsahují. Součástí článku jsou také nastíněny návrhy na vylepšení či rozšíření existujícího algoritmu a možnosti budoucího výzkumu.

1 Vlk obecný

Vlci jsou pozoruhodná, společenská šelmovitá zvířata. Jsou na vrcholu potravinového řetězce a nemají přirozené nepřátelé. Živí se savci menšího i většího vzrůstu, výjimečně loví i ptáky, ryby, plazy, či hmyz. Vlčí smečka má obvykle 5 až 7 členů (až 30) a platí v ní přísná hierarchie, ve které má každý jedinec pevně stanovenou roli [16]. Vlci vytvářejí hierarchii sami mezi sebou. Hierarchie se mění v průběhu času (vlci stárnou a rodí se nové generace) a vlci v ní mohou mít různou roli při různých činnostech. Rozlišujeme vlky:

- *Alpha* – Alpha samec a samice, tzv. dominantní pár, jejich příkazy následují ostatní vlci, rozhodují při lovu, volbě místa ke spaní, aj. Alpha samice má jako jediná samice v hierarchii rozmnožovací úlohu. Nejdůležitějšími vlastnostmi Alpha páru jsou organizační schopnosti a disciplína (nikoliv síla).
- *Beta* – podporují a respektují Alpha pár při rozhodovacích schopnostech, poskytují mu zpětnou vazbu, často jsou kandidáty na nové Alpha vlky, pověřují úkoly vlky umístěné níže v hierarchii.
- *Delta* – podřizují se Alpha a Beta vlkům, plní jejich rozkazy, ovládají Omega vlky, často jde o starší vlky, či naopak mladší s potenciálem posunout se v hierarchii výše. Dělíme je na:
    - *Skauty* – sledují okolí a varují smečku v případě nebezpečí.
    - *Strážce* – pomáhají při lovu a v případě nebezpečí chrání smečku, svou přítomností zajišťují pocit bezpečí ve smečce.
    - *Ošetřovatelé* – pomáhají starým a nemocným vlkům.
- *Omega vlci* – podřizují se všem ostatním vlkům, mají právo jíst až jako poslední, pomáhají filtrovat násilí a frustraci ve smečce tím, že si na nich ostatní vlci mohou "vybít zlost". Ztráta omega vlka může být příčinou rozepří ve smečce a následného narušení disciplíny, hierarchie, i rozpadu smečky [16].

Techniku lovu vlčí smečky lze popsat s pěti krocích:

A. *Pátrání po kořisti* – vlci se snaží nalézt co nejvydatnější kořist s ohledem na potřebné úsilí k ulovení kořisti.
B. *Obtěžování kořisti* – snaha vlků v popředí upoutat na sebe pozornost a případně oddělit vyhlídnutou kořist od stáda.
C. *Obklíčení kořisti* – snaha dostat kořist do patové situace.
D. *Kořist je obklíčena* – kořist nemá kam utéct.
E. *Útok* – vlci útočí na slabá místa kořisti (břicho, nohy, čumák) dokud nepodlehne únavě, poté ji strhnou na zem a zadáví [16].

Obr. 1: Technika lovu [10]

2 Smečkový algoritmus

V této kapitole využijeme poznatků ze života vlků, analogicky je uplatníme při návrhu algoritmu aplikovatelného k řešení například optimalizačních problémů. Pro návrh a implementaci algoritmu jsou použity poznatky ze zdrojů [10][12][14].
Cílem algoritmu je nalezení globálního extrému (minima) nějaké zadané funkce (*fitness* funkce) na jejím definičním oboru, kterým je omezený n-rozměrný prostor (*prostředí*). V ilustracích budeme v článku využívat 1D, 2D a 3D prostor. Jednotliví vlci jsou plně popsáni svými pozicemi (n-rozměrnými vektory) v tomto prostoru. Hodnota *fitness* na pozici vlka představuje kvalitu jeho řešení – jeho *fitness* (čím menší hodnota funkce, tím lepší fitness).
V průběhu algoritmu si vlci určitým způsobem vyměňují informace o své poloze a na jejich základě upravují své polohy v dalším kroku algoritmu, s cílem co nejvíce se přiblížit požadovanému extrému fitness funkce. Hierarchie vlků v každém kroku algoritmu se odvíjí od kvality jejich aktuálního řešení, a vlci výše postavení v hierarchii více ovlivňují příští pozice ostatních vlků. Populace vlků je konstantní, v průběhu algoritmu se mění pouze jejich postavení v hierarchii a jejich aktuální pozice.
Algoritmus postupně plynule přechází mezi dvěma fázemi:

1. Fáze hledání kořisti – smečka díky vysokému podílu náhodných pohybů vlků extenzívně prohledává prostředí, aby zamezila uváznutí algoritmu v lokálním minimu.
2. Fáze obklíčení kořisti – vliv náhodných pohybů je postupně omezován a členové smečky se postupně stahují kolem nalezeného extrému (minima) fitness funkce.

Notace matematických rovnic v dalším textu není standardní a veškerá násobení vektorů se ve vzorcích počítají po komponentách následovně:

$$(x_{1,1}, x_{1,2}, \dots, x_{1,n}) * (x_{2,1}, x_{2,2}, \dots, x_{2,n}) =$$

$$(x_{1,1} * x_{2,1}, x_{1,2} * x_{2,2}, \dots, x_{1,n} * x_{2,n}),$$

kde $x_{i,j}$ označuje komponentu i na osové souřadnici dimenze j. Tedy například:

$$(x_1, y_1) * (x_2, y_2) = (x_1 * x_2,\ y_1 * y_2),$$

kde x_i označuje komponentu i na osové souřadnici x a y_i označuje komponentu i na osové souřadnici y.
S. Mirjalili, autor který navrhl tento algoritmus [10], používá ve svých publikacích právě tuto notaci. Jeho vzorce jsou převzaty také v jiných dostupných publikacích, jenž citují Mirjaliliho práci. Rozhodl jsem se proto tuto notaci zachovat.

2.1 Prostředí a vlk

Cílem vlka v prostředí je najít a „ulovit kořist," tj. optimální řešení problému (souřadnice, na nichž fitness funkce nabývá stanovaného kritéria, v našem případě globálního minima). Metaforicky budeme nazývat hodnotu fitness funkce na aktuální pozici vlka dosud nejlepší nalezenou kvalitou kořisti.
Pohyb vůdčích vlků ve 3D prostředí směrem ke odhadované pozici kořisti je znázorněn na obr. 2:

Obr. 2: Pohyb vlků ve 3D prostředí v jednom kroku algoritmu: (X,Y,Z) – pozice vlka, (X^*,Y^*,Z^*) – kořist.

Fialová šipka na obr. 2 znázorňuje aktuální pozici vlka. Medvěd představuje pozici odhadnuté kořisti. Ostatní šipky znázorňují potencionální nové pozice jedince, jenž jsou blíže (nebo stejně vzdálené) ke kořisti. Pozice stejně vzdálené od kořisti na ose X jsou vypočteny jako:

$$X^* - (X - X^*) = 2X^* - X\ ;\ X^* - (X^* - X) = X,$$

kde X^* je pozice kořisti a X je pozice vlka. Konkrétní nová pozice jedince (nemusí být nutně blíže ke kořisti) závisí na dalších parametrech, o kterých se dočteme v dalším textu.

2.2 Sociální hierarchie

V každé iteraci algoritmu zařazujeme podle fitness funkce každého z jedinců do vlčí hierarchie. Jedinec s nejlepší (v našem případě nejnižší) fitness hodnotou je zařazen do kategorie Alpha, s druhou nejlepší fitness hodnotou do kategorie Beta, s třetí nejlepší fitness hodnotou do kategorie Delta a všichni ostatní jedinci jsou zařazeni do kategorie Omega.
Při aktualizaci sociální hierarchie algoritmus bere v potaz také fitness hodnoty vlků Alpha, Beta a Delta z předchozích iterací a zachovává tak dosud nejlepší nalezené řešení problému.

2.3 Hledání a pronásledování kořisti

Aktualizace pozic vlků v prostředí probíhá na základě odhadu pravděpodobné pozice kořisti podle vlků Alpha, Beta a Delta (více v podkapitole *Obklíčení kořisti*). Abychom zajistili divergenci mezi hledáním kořisti a lovem, přiřadíme každému jedinci vektor:

$\vec{A}$ se složkami $rand(-1,1) * a$,

kde funkce $rand(-1,1)$ generuje náhodné číslo v rozmezí -1 až 1 a kde:

$$a = 2 - \frac{2i}{i_{max}},$$

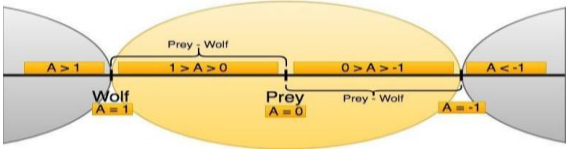
přičemž i je aktuální iterace algoritmu, imax je maximální počet iterací algoritmu. Přitom a se chová tak, že s přibývajícími iteracemi algoritmu lineárně klesá jeho hodnota od 2 k 0, a zvyšuje se tedy postupně (v pokračujících iteracích algoritmu) pravděpodobnost, že také $\|\vec{A}\|$ (velikost $\vec{A}$) bude klesat. Vektor $\vec{A}$ má tedy hodnotu v rozmezí (-a, a) a ovlivňuje směr pohybu jedinců v prostoru (viz. podkapitola Hledání a pronásledování kořisti). Přitom velikost vektoru:

$$\|\vec{A}\| > 1$$

ve zhruba první polovině celkového počtu iterací algoritmu přinutí jedince odchylovat se od doposud nejlepší nalezené kořisti s vidinou, že mohou nalézt kořist lepší (tedy že dosud nejlepší nalezená kořist je nejlepší pouze lokálně), zatímco:

$$\|\vec{A}\| < 1$$

ve zhruba druhé polovině iterací algoritmu přinutí jedince naopak přibližovat se k dosud nejlepší nalezené kořisti a tu pronásledovat a lovit. Každý jedinec má přiřazen vlastní vektor $\vec{A}$ a rozhoduje proto každý sám za sebe, kdy bude kořist hledat a kdy lovit.

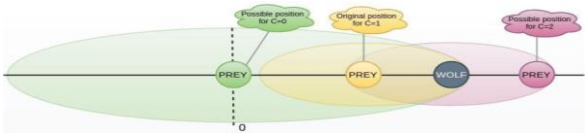


Obr. 3: Vliv vektoru $\vec{A}$, pro přehlednost v 1D prostředí

Žlutý kruh na obrázku 3 vyznačuje část prostředí, v němž je každá pozice blíže kořisti než současná pozice jedince. Je-li $\|\vec{A}\| < 1$, pak nová pozice kořisti bude s vysokou pravděpodobností právě uvnitř tohoto kruhu.
Další složka algoritmu podporující fázi hledání, či pronásledování kořisti, je vektor:

$\vec{C}$ se složkami $rand(0,2)$,

který je náhodným číslem v rozmezí 0 až 2 (jiným, než generuje funkce rand v $\vec{A}$) a slouží k tomu, aby náhodně zvýšil, nebo naopak snížil jedincem nalezenou kořist. Na rozdíl od vektoru $\vec{A}$ není lineárně snižován v průběhu přibývajících iterací algoritmu. Vektor $\vec{C}$ napomáhá vlkům chovat se více přirozeně. Analogicky se v přírodě vyskytují v loveckých cestách vlků různé překážky zabraňující pohodlnému přiblížení ke kořisti. Vektor $\vec{C}$ tedy napomáhá vlkům upřednostňovat průzkum prostředí, či lov kořisti, vyhýbat se tak lokálnímu optimu, a to nezávisle na iteraci algoritmu.



Obr. 4: Vliv vektoru $\vec{C}$, pro přehlednost v 1D prostředí

Jak můžeme vidět na obrázku 6, složky vektoru $\vec{C}$ mohou být i nulové, čímž mohou v konkrétní dimenzi výrazně měnit vliv pozice kořisti na novou pozici vlka. Tato pozice přitom může být i velmi vzdálená od pozice vlka i původně odhadnuté pozice kořisti a vliv vektoru $\vec{C}$ je proto silně náhodný.

2.4 Obklíčení kořisti

Aktualizace vektorů pozic jednotlivých jedinců $\vec{X}_j$, kde j je index jedince, probíhá podle předpisu:

$$\vec{X}_j(i+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3},$$

kde i je aktuální iterace algoritmu a $\vec{X}_1, \vec{X}_2, \vec{X}_3$ jsou potencionální nové pozice vlků Alpha, Beta, Delta, kteří reprezentují pozice v prostředí nejbližší optimu. Aktualizovaná pozice jedince $\vec{X}_j(i+1)$ je tedy průměrem potencionálních pozic $\vec{X}_1, \vec{X}_2, \vec{X}_3$ a nachází se mezi nimi. Přitom:

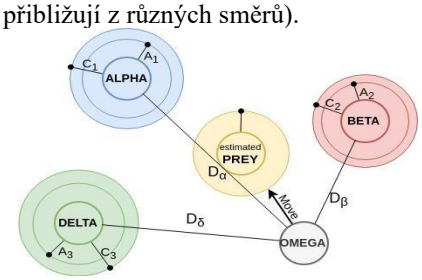
$$\vec{X}_1 = \vec{X}_\alpha(i) - \vec{A}_1 * \vec{D}_\alpha,$$
$$\vec{X}_2 = \vec{X}_\beta(i) - \vec{A}_2 * \vec{D}_\beta,$$
$$\vec{X}_3 = \vec{X}_\delta(i) - \vec{A}_3 * \vec{D}_\delta\ ,$$

kde $\vec{X}_\alpha(i)$, $\vec{X}_\beta(i)$, $\vec{X}_\delta(i)$ jsou pozice vlků Alpha, Beta, Delta reprezentující pozice s dosud nejlepší nalezenou kořistí v iteraci i, vektor $\vec{A}$ je definován v podkapitole Hledání a pronásledování kořisti a je vypočten 3x (indexy 1, 2, 3), $\vec{D}_\alpha$, $\vec{D}_\beta$, $\vec{D}_\delta$ definují vzdálenost mezi pozicí jedince X_j od kořisti následovně:

$$\vec{D}_\alpha = |\,\vec{C}_1 * \vec{X}_\alpha(i) - \vec{X}_j(i)\,|,$$
$$\vec{D}_\beta = |\,\vec{C}_2 * \vec{X}_\beta(i) - \vec{X}_j(i)\,|,$$
$$\vec{D}_\delta = |\,\vec{C}_3 * \vec{X}_\delta(i) - \vec{X}_j(i)\,|,$$

kde $|\vec{X}|$ je vektor, jehož složky jsou absolutní hodnoty složek vektoru $\vec{X}$. Vektor $\vec{C}$ je definován v podkapitole Hledání a pronásledování kořisti, je vypočten stejně jako vektor $\vec{A}$ 3x (indexy 1, 2, 3) a ovlivňuje váhu odhadované pozice kořisti $\vec{X}_\alpha(i)$, $\vec{X}_\beta(i)$, $\vec{X}_\delta(i)$ (zesiluje ji, nebo naopak zeslabuje).
Vzdálenosti $\vec{D}_\alpha$, $\vec{D}_\beta$, $\vec{D}_\delta$ tedy v pozicích $\vec{X}_1$, $\vec{X}_2$, $\vec{X}_3$ určují okolí odhadu pozice kořisti $\vec{X}_\alpha(i)$, $\vec{X}_\beta(i)$, $\vec{X}_\delta(i)$

ovlivněné váhou $\vec{C}$, v němž se nachází všechny možné pozice, které jsou kořisti blíže, než současná pozice jedince $\vec{X}_j$(i). Konkrétní pozice je závislá na vektoru $\vec{A}$, přitom je-li $\|\vec{A}\| < 1$, pak se potencionální pozice jedince $\vec{X}_1, \vec{X}_2$ nebo $\vec{X}_3$ nachází právě uvnitř tohoto okolí (čím je $\|\vec{A}\|$ blíže nule, tím blíže je kořisti) a naopak.
Omega vlci se nepodílejí na odhadu kořisti a jejich pozice se aktualizují pouze na základě odhadu pozice kořisti vlky Alpha, Beta, nebo Delta. Vlci Alpha, Beta a Delta jsou si v tého základní implementaci algoritmu rovnocenní a při aktualizaci svých pozic zohledňují také své vlastní aktuální pozice – nemají tak tendenci měnit příliš svou pozici. Pokud mají vlci tendenci přibližovat se ke kořisti, dochází k jevu, kdy kořist postupně obklíčí (vlci se přibližují z různých směrů).



Obr. 5: Aktualizace pozic $\vec{X}_j(i+1)$ jedinců Omega podle vzorce na začátku kapitoly 2.4 [10]

Analogie pro napadení kořisti je pak v algoritmu stav, kdy dojde ke splnění ukončovací podmínky algoritmu, tedy pokud vyčerpáme všechny iterace algoritmu (řešením je pozice Alpha vlka), nebo když se Alpha přesune na takovou pozici v prostředí, jejíž hodnota odpovídá optimu.

2.5 Popis algoritmu

Vstupy algoritmu jsou:

- dimenze prostředí problému,
- hranice prostředí problému,
- fitness funkce charakterizující problém,
- velikost smečky (počet vlků),
- počet iterací algoritmu,
- ukončovací podmínka,
- kritérium pro fitness funkci.

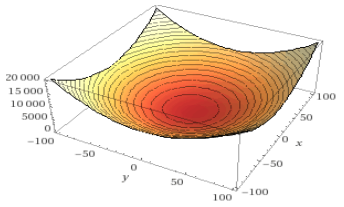
Pseudokód algoritmu:

1. Inicializace jedinců (vlků):
   - náhodné umístění vlků do prostoru.
2. V každé iteraci *i*:
   a. výpočet fitness funkce jedinců a určení sociální hierarchie:
      i. Alpha = jedinec s nejlepší (nejnižší) fitness funkcí,
      ii. Beta = jedinec s druhou nejlepší fitness funkcí,
      iii. Delta = jedinec s třetí nejlepší fitness funkcí,
      iv. Omega = ostatní jedinci,
   b. výpočet dosud nejlepšího řešení $\vec{X}_\alpha(i)$, $\vec{X}_\beta(i)$, $\vec{X}_\delta(i)$ (přiřazení pozic vlků Alpha, Beta, Delta),
   c. aktualizace pozic všech jedinců $\vec{X}_j(i+1)$, přičemž pro každého z nich aktualizujeme vektory $\vec{A}$, $\vec{C}$,
   d. kontrola ukončovací podmínky.

3 Příklad implementace

V této kapitole si předvedeme funkčnost navrženého algoritmu na příkladu optimalizace problému hledání minima jednoduché fitness funkce F ve 2D prostoru:

$$F = x^2 + y^2, kde\ x, y \in (-100, +100).$$



Obr. 6: Graf jednoduché fitness funkce F [18]

Pro použití v *n* dimenzích funkci *F* zobecníme:

$$F = (x_1)^2 + (x_2)^2 + \ldots + (x_n)^2\ ,$$

kde x_j je složka n-dimenzionálního prostoru, $1 \leq j \leq n$.
Dále stanovíme velikost smečky (počet vlků) na hodnotu 7 (inspirace přírodou) a počet iterací algoritmu například na hodnotu 50. Ukončovací podmínku definovat nebudeme (algoritmus vždy projde všechny iterace). Je-li algoritmus v první iteraci, pak:

- vypočti fitness hodnotu pro každého z jedinců tak, že dosadíme do funkce *F* pozice jedinců $\vec{X}_j(1)$,
- srovnej fitness hodnoty *F* jedinců:
  - vlk s nejnižší fitness hodnotou se stává vlkem Alpha,
  - vlk s druhou nejnižší fitness hodnotou se stává vlkem Beta,
  - vlk s třetí nejnižší fitness hodnotou se stává vlkem Delta,
  - ostatní vlci se stávají vlky Omega.

V dalších iteracích proveď pro každého z jedinců:

- vypočti fitness hodnotu *F* pro daného jedince,
- zařaď jedince do stávající sociální hierarchie:
  - jedinec se stává novým Alpha vlkem je-li výsledná hodnota *F* nižší než

fitness hodnota dosavadního Alpha vlka,
- jedinec se stává novým Beta vlkem je-li výsledná hodnota *F* vyšší než fitness hodnota dosavadního Alpha vlka, ale nižší než fitness hodnota dosavadního Beta vlka,
- jedinec se stává novým Delta vlkem, je-li výsledná hodnota *F* vyšší, než fitness hodnota dosavadního Alpha a Beta vlka, ale nižší než fitness hodnota dosavadního Delta vlka,
- jedinec se stává Omega vlkem, je-li výsledná hodnota *F* vyšší než fitness hodnota dosavadního Alpha, Beta i Delta vlka.

Činnost programu, tedy pohyb vlků v prostředí v jednotlivých iteracích, probíhá podle popisu algoritmu v předchozí kapitole. Algoritmus při testování konvergoval již po 20 krocích k hodnotě 0 s přesností na více než 3 desetinná místa a po padesáti krocích konvergoval až k hodnotě: 4.886810777574884e-07.

4 Současné uplatnění algoritmu

Obecně uplatníme tento algoritmus při řešení vyhledávacích a klasifikačních problémů pomocí optimalizace.
V současné době nalezneme uplatnění algoritmu v celé řadě odvětví. V inženýrství například při návrhu integrovaných obvodů a kontrolérů, při plánování cesty robotů, nebo při minimalizaci nákladů a spotřeby při dodávání, nebo odběru elektřiny (ať už při návrhu sítě elektráren dodávajících proud koncovým zákazníkům, nebo při návrhu energeticky úspornějších hardwarových i softwarových komponent počítače). Další uplatnění nalezneme v oblasti návrhu počítačových (či jiných) sítí. V oblastech, jako jsou architektura a ekonomie, se uplatní při minimalizaci nákladů, naložených kamionů či potřebného materiálu.
V neuronových sítích byl algoritmus efektivně použitý pro nalezení optimálních vstupních vah u neuronové sítě typu Backpropagation, dále k samotnému učení vícevrstvého perceptronu.
V matematice má algoritmus uplatnění při výběru optimální podmnožiny (například v oblastech, jako jsou statistika, fuzzy logika, aj.) a v počítačové grafice při segmentaci obrazu (například prahování). Segmentace obrazu se využívá v oblasti medicíny při identifikaci částí snímků pořízenými rentgenem, nebo CT.
Ve všech těchto oblastech, které se mnohdy vzájemně překrývají, prokazuje použití smečkového algoritmu pozitivní výsledky (alespoň v konkrétních případech) z hlediska času zpracování a přesnosti (kvality) v porovnání s výsledky podobných [3][4][5][6][9][15].

5 Srovnání s podobnými algoritmy

Algoritmy prohledávání do šířky či hloubky poskytují přesná řešení celé řady problémů. Neposkytují ale dostatečně rychlé řešení při řešení složitých (či neřešitelných) problémů, a právě tehdy se nabízí využít optimalizační algoritmus. V tomto případě můžeme sáhnout po některém z algoritmů zmíněném v tabulce níže. Vzhledem k šíři problematiky nejsou popsané činnosti těchto algoritmů, avšak zájemce si o nich může přečíst více informací v odkazované literatuře.

Algoritmus	Stručné srovnání
Hejna částic [17][1]	V rámci práce [1] byly oba (i smečkový) algoritmy srovnány na příkladu optimalizace nákladů na výrobu pohonných hmot. Hejna částic zpočátku konvergují rychleji k optimální hodnotě, avšak smečkový algoritmus optimální hodnoty dosáhl dříve. Algoritmus kombinující vlastnosti obou algoritmů na stejném příkladu prokázal pozitivní výsledek v porovnání s oběma z nich.
Genetický algoritmus [8][17][5]	Algoritmus byl použít v práci [5] při řešení problému toku energie v elektrické síti. Během 100 iterací algoritmu nebyl schopen konvergovat k optimální hodnotě, resp. možná konvergoval předčasně. Jistě bychom ale našli i příklady, kdy algoritmus funguje efektivně ve srovnání s tímto případem.
SOMA algoritmus [17] [2]	Algoritmus byl použít v práci [2], ve které se osvědčil při řešení problému obchodního cestujícího například v porovnání s algoritmem hejna částic (PSO). Přímé srovnání se smečkovým algoritmem se mi nepodařilo nalézt.
Mravenčí kolonie [7][13]	V práci [13] byl algoritmus použít při řešení problému obchodního cestujícího a byl srovnán s genetickým algoritmem. Přímé srovnání se smečkovým algoritmem se mi nepodařilo nalézt.

6 Další zkoumání

V blízké době plánuji především zkoumat chování popsaného algoritmu s různě nastavenými parametry a dále plánuji zkusit nahradit vektor $\vec{C}$ přiřazující kořisti váhu méně náhodným způsobem.

Algoritmus dosud pracuje jen s malou podmnožinou vlastností života reálných vlků. Skutečné vlky však dokážeme rozlišit pomocí celé řady parametrů – síla, rychlost, výdrž, stáří, aj. V další fázi výzkumu proto plánujeme zahrnout do algoritmu některé z těchto vlastností a využít je například k volbě délky kroku při přesunu pozice vlka směrem ke kořisti. Implementované vlastnosti vlků by se rovněž mohly promítnout při tvorbě společenské hierarchie podobně, jako u genetických algoritmů. Vlci by rovněž mohli tvořit v průběhu iterací nové generace s vlastnostmi optimalizovanými pro konkrétní typ úlohy a nahrazovat generace původní. Z matematického hlediska by to znamenalo definovat fitness funkci komplexněji než pouze pomocí aktuální kvality řešení jedince.
Některé vlastnosti, jako jsou únava (zrychlení / zpomalení kroku pohybu ke kořisti v závislosti na předchozích reakci vlka), hlad (možný efektivnější přístup k divergenci mezi hledáním kořisti a lovem), potřeba odpočinku (důkladnější prohledání okolí, pokud se vlkům nedaří nalézt ideální kořist), aj. plánuji implementovat pomocí gramatik typu kolonie.
Bude přinejmenším zajímavé sledovat chování modifikovaného algoritmu.

Literatúra

[1] Chopra, N., Kumar, G., & Mehta, S. (2016). Hybrid GWO-PSO Algorithm for Solving Convex Economic Load Dispatch Problem, 4.

[2] Dokania, S., Bagga, S., & Sharma, R. (2017). Opportunistic Self Organizing Migrating Algorithm for real-time Dynamic Traveling Salesman Problem. In *2017 51st Annual Conference on Information Sciences and Systems* (CISS) (pp. 1-6). IEEE. https://doi.org/10.1109/CISS.2017.7926065.

[3] Faris, H., Aljarah, I., Al-Betar, M. A., & Mirjalili, S. (2018). Grey wolf optimizer: a review of recent variants and applications. In *Neural Computing and Applications* (Vol. 30, pp. 413-435). IEEE. https://doi.org/10.1007/s00521-017-3272-5.

[4] Kayvan, S. (2016). *An application of Grey Wolf Optimization algorithm for fuzzy portfolio selection problem* [Online].

[5] Khaled, A., El-Rafei, A., & Badra, N. Solution of Optimal Power Flow using Evolutionary-Based Algorithms. *International Journal Of Engineering: Science And Technology*, *9*(55-68). https://doi.org/9. 55-68. 10.4314/ijest.v9i1.5.

[6] Khelifi, A., Saliha, C., & Bentouati, B. A new hybrid algorithm of particle swarm optimizer with grey wolves' optimizer for solving optimal power flow problem. *Leonardo Electronic Journal Of Practices And Technologies*, *17*(32).

[7] KRÖMER, P. (2012). O*ptimalizace pomocí mravenčích kolonií: Inspirace, definice, aplikace a perspektivy* [Online]. Retrieved from https://homel.vsb.cz/~kro080/mravenci.pdf.

[8] Mitchell, M. (c1996). *An introduction to genetic algorithms*. Cambridge, Mass.: MIT Press.

[9] Mirjalili, S. (2015). How effective is the Grey Wolf optimizer in training multi-layer perceptrons. *Applied Intelligence*, 43(1), 150-161. https://doi.org/10.1007/s10489-014-0645-7.

[10] Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey Wolf Optimizer. *Advances In Engineering Software*, 69(1), 46-61. https://doi.org/10.1016/j.advengsoft.2013.12.007.

[11] Radinger, E. H. (2018). *Moudrost vlků: jak myslí, jak vnímají a pečují o sebe*. Praha: Mladá fronta.

[12] Rajesh, K. Downloads - presentations and source codes [Online]. Retrieved March 24, 2019, from.

[13] RANDALL, P. Coded and Testing [Online]. Retrieved March 24, 2019, from https://reflectionsonor.wordpress.com/2012/08/01/ant-colony-optimization-coded-and-testing/.

[14] SISdevelop: SwarmPackagePy [Online]. Retrieved March 24, 2019, from https://pypi.org/project/SwarmPackagePy/

[15] Jiang, T., & Zhang, C. (2018). Application of Grey Wolf Optimization for Solving Combinatorial Problems: Job Shop and Flexible Job Shop Scheduling Cases. *Ieee Access*, *6*, 26231-26240. https://doi.org/10.1109/ACCESS.2018.2833552.

[16] Velké šelmy: VLK (CANIS LUPUS) [Online]. (2010). Retrieved March 24, 2019, from http://selmy.ursus.cz/vlk/V-obecne.html.

[17] Volná, E. (2012). *Evoluční algoritmy a neuronové sítě* [Online]. Ostrava. Retrieved from http://www1.osu.cz/~volna/Evolucni_algoritmy_a_neuronove_site.pdf.

[18] Wolfram|Alpha [Online]. (2018). Retrieved March 24, 2019, from https://www.wolframalpha.com/.