VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A ŘÍZENÍ

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PLÁNOVÁNÍ CESTY ROBOTA POMOCÍ ROJOVÉ INTELIGENCE

ROBOT PATH PLANNING BY MEANS OF SWARM INTELLIGENCE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. ALEŠ SCHIMITZEK
AUTHOR

VEDOUCÍ PRÁCE — RNDr. JIŘÍ DVOŘÁK, CSc.
SUPERVISOR

BRNO 2013

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2012/2013

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Aleš Schimitzek

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Plánování cesty robotu pomocí rojové inteligence

v anglickém jazyce:

Robot path planning by means of swarm intelligence

Stručná charakteristika problematiky úkolu:

Úkolem systému pro plánování cesty robotu je najít cestu z výchozího do cílového bodu tak, aby se robot nedostal do kolize se známými překážkami, aby nebyla porušena případná omezení kladená na pohyb robotu a aby byla optimalizována nějaká kriteriální funkce. K možným přístupům k řešení tohoto problému patří také metody rojové inteligence (např. metody mravencích systému nebo ptačích hejn).

Cíle diplomové práce:

1. Analyzovat přístupy k plánování cesty robotu.
2. Popsat metody rojové inteligence a jejich aplikace na plánování cesty.
3. Navrhnout a implementovat systém pro plánování cesty robotu pomocí rojové inteligence.
4. Provést ověřovací a srovnávací experimenty.

Seznam odborné literatury:

MOHEMMED, A. W.; SAHOO, N. C.; GEOK, T. K. Solving shortest path problem using particle swarm optimization. Applied Soft Computing, vol. 8, 2008, pp. 1643-1653.
TAN, G.-Z.; HE, H.; SLOMAN A. Ant Colony system algorithm for real-time globally optimal path planning of mobile robots. Acta Automatica Sinica, vol. 33, 2007, pp. 279-285.
ZHANG, Q.-Z.; LIU, X.-H.; GE, X.-S. Non-holonomic motion planning with PSO and spline approximation. In Proceedings of the 2007 IEEE International Conference on Control and Automation, Guangzhou, China, 2007, pp. 2600-2604.
ZHU, Q.; HU, J.; CAI, W.; HENSCHEN, L. A new robot navigation algorithm for dynamic unknown environments based on dynamic path re-computation and an improved scout ant algorithm. Applied Soft Computing, vol. 11, 2011, pp. 4667-4676.

Vedoucí diplomové práce:RNDr. Jiří Dvořák, CSc.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2012/13.

V Brně, dne 20.11.2012

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA STROJNÍHO INŽENÝRSTVÍ DĚKANÁT

Ing. Jan Roupec, Ph.D.
Ředitel ústavu

prof. RNDr. Miroslav Doupovec, CSc., dr. h. c.
Děkan

ABSTRAKT

Tato diplomová práce se zabývá plánováním cesty robota pomocí rojové inteligence. V teoretické části jsou popsány nejznámější metody rojové inteligence (optimalizace mravenčí kolonií, optimalizace včelím rojem, optimalizace rojem světlušek a optimalizace hejnem částic) a jejich aplikace pro plánování cesty. V praktické části je zvolena optimalizace hejnem částic pro návrh a implementaci plánování cesty v programu C#.

ABSTRACT

This diploma thesis deals with the path planning by swarm intelligence. In the theoretical part it describes the best known methods of swarm intelligence (Ant Colony Optimization, Bee Swarm Optimization, Firefly Swarm Optimization and Particle Swarm Optimization) and their application for path planning. In the practical part particle swarm optimization is selected for the design and implementation of path planning in the C#.

KLÍČOVÁ SLOVA

Plánování cesty, rojová inteligence, optimalizace mravenčí kolonií, optimalizace včelím rojem, optimalizace rojem světlušek, optimalizace hejnem částic, mobilní robot.

KEYWORDS

Path planning, swarm intelligence, Ant Colony Optimization, Bee Swarm Optimization, Firefly Swarm Optimization, Particle Swarm Optimization, mobile robot

Prohlášení o originalitě

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, pod vedením vedoucího diplomové práce pana RNDr. Jiřího Dvořáka, CSc. a s použitím uvedené literatury.

V Brně dne 23.5.2012 Podpis: ……..….

Bibliografická citace mé práce:

SCHIMITZEK, A. Plánování cesty robotu pomocí rojové inteligence. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 75 s. Vedoucí diplomové práce RNDr. Jiří Dvořák, CSc..

Poděkování:

Rád bych tímto poděkoval RNDr. Jiřímu Dvořákovi, CSc. za odbornou pomoc, cenné rady a vedení při vypracování diplomové práce.

Obsah:

1. Úvod

V prvopočátku se plánování cesty využívalo hlavně pro řešení problému obchodního cestujícího, aby se ušetřil čas a hlavně náklady na dopravu. V dnešní době je plánování cesty spjato s roboty, kdy člověk vyžaduje po robotu určitou samostatnost a tím i jeho lepší využití. První výsledky takových plánování lze už dnes vidět hlavně ve vojenském využití, kdy roboti sami dorazí na určené místo a podle předem definované funkce vykonávají na místě svojí úlohu. Zajímavé jsou různé školní projekty, kdy roboti sami hrají fotbal, projíždí různě složitá bludiště, pohybují se mezi lidmi a nabízejí informace atd.

I když plánování cesty robota může z prvopočátku vypadat jednoduše, skýtá mnoho problémů a omezení, které je třeba vyřešit. Tyto problémy mohou být způsobeny různými příčinami, počínaje velikosti robota přes jeho poloměr otáčení až po problém kolize robota s překážkou.

Plánování cesty robota pomocí rojové inteligence zažilo největší rozmach v posledních dvaceti letech. Rojové inteligence spadají mezi genetické algoritmy, neuronové sítě a jiné biologicky inspirované metody umělé inteligence.

Tato diplomová práce je rozdělena na čtyři okruhy. V teoretické části jsou popsány různé metody pro plánování cesty a čtyři nejznámější případy rojové inteligence, které byly inspirovány právě chováním a výměnou informací reálného roje, hejna nebo kolonie, které hledají potravu, hnízdiště nebo se páří. Tyto nejznámější rojové inteligence jsou optimalizace mravenčí kolonií, optimalizace včelím rojem, optimalizace rojem světlušek a optimalizace hejnem částic. Pro implementaci jsem zvolil algoritmus optimalizace hejnem částic, kterému je věnována další část práce. Inspirací pro tuto část byly články [26 - 29]. Třetí část je věnovaná popisu programu. Poslední část diplomové práce obsahuje výsledky experimentů a simulací vytvořeného programu a jeho porovnání s potenciálovým polem.

2. Plánování cesty robota

Podle [1] je plánování cesty součástí navigace, kterou lze definovat jako určení pozice objektu, který se bude pohybovat k cíli, a nalezení vhodné cesty, která vede k cíli.

Plánování může být globální, lokální nebo může jít o kombinaci globálního a lokálního plánování. Globální plánování se provede ještě před prvním pohybem robota a je nutné nastavit následující podmínky. První podmínka je, aby prostředí, ve kterém se robot pohybuje, bylo předem známé. Druhá podmínka je, aby prostředí od zahájení výpočtu cesty až po dosažení cíle samotným robotem bylo neměnné. Oproti tomu lokální plánovaní se provádí během pohybu robota v prostředí a odpadají problémy známosti a statičnosti prostředí. Posledním případem plánování je kombinace globálního a lokálního plánovaní. Nejprve provede globální plánování, a pokud se v prostředí objeví změna, tak se plánuje lokálně. U některých případů plánování nelze na první pohled rozeznat rozdíl mezi lokálním a globálním plánováním [1][2].

2.1 Metody plánování cesty

Pro plánování cesty robota existuje velká škála metod. Další metody přibývají nebo se stávající algoritmy pro plánování cesty modifikují. Podle práce [1] mají většinou tyto metody dvě části:

1. Předzpracování prostoru
2. Plánování cesty (dotazovací část)

Podrobněji o těchto částech v pracích [2][3][4].

Z důvodu velkého množství metod předzpracování prostoru jsou zde uvedeny jen tři základní metody.

1. Metody rozkladu do buněk
2. Mapy cest
3. Potenciálové pole

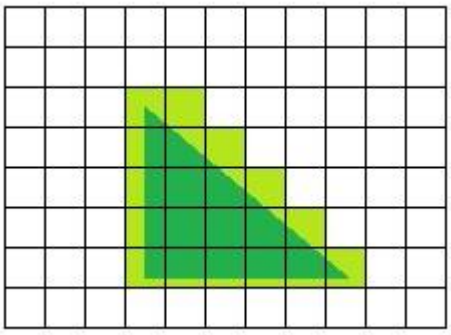
Poté co se vytvoří vhodná grafová struktura, aplikujeme vhodný algoritmus pro vlastní plánování cesty. Těchto algoritmů je velké množství, počínaje Dijkstrovým algoritmem přes algoritmus A* až po mravenčí algoritmus.

2.1.1 Metody rozkladu do buněk

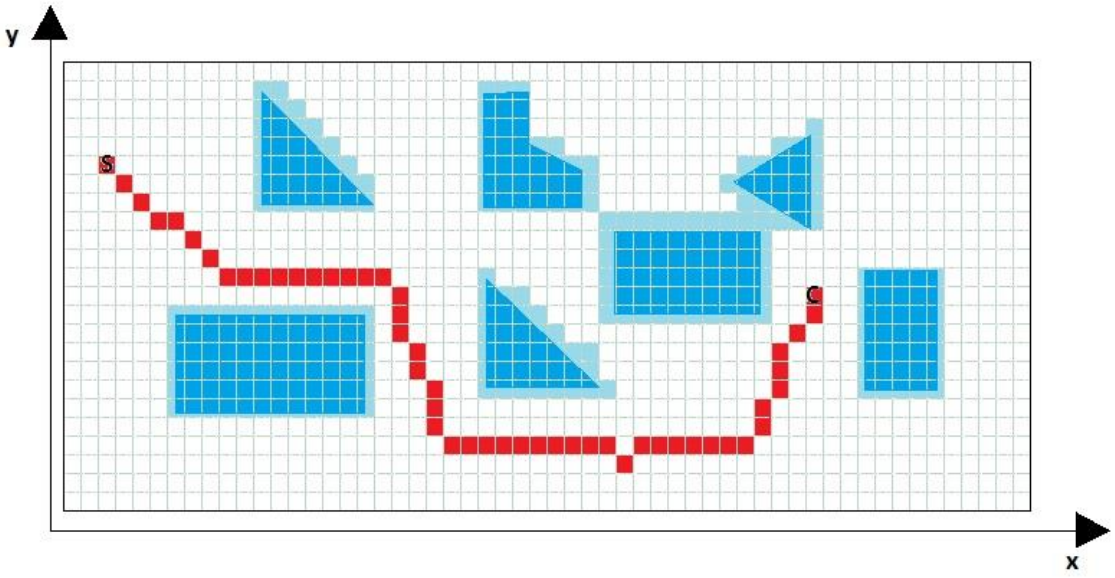
Tyto metody spočívají v tom, že daný prostor rozdělíme do buněk. Formy buněk mohou nabývat různých tvarů a velikosti. V dané buňce se zkoumá, zda obsahuje překážku, nebo jestli je zde volný prostor. Podle těchto poznatků se vytvoří graf, kde vrcholy jsou buňky neobsahující překážku a hrany grafu jsou spojnice mezi buňkami. Základními metodami rozkladu buněk jsou exaktní a aproximativní rozklad.

Aproximativní rozklad do buněk

Celý prostor se rozloží na buňky stejné velikosti (nejčastěji čtvercového tvaru). Pokud do dané buňky zasahuje překážka, tak se vyplní (obr. 1) jakoby obsahovala celou překážku a nezahrne se do grafové struktury. Výpočetní nároky a přesnost jsou dané velikostí buněk. V práci [1] je zmínka, že velikost buněk může být proměnlivá.



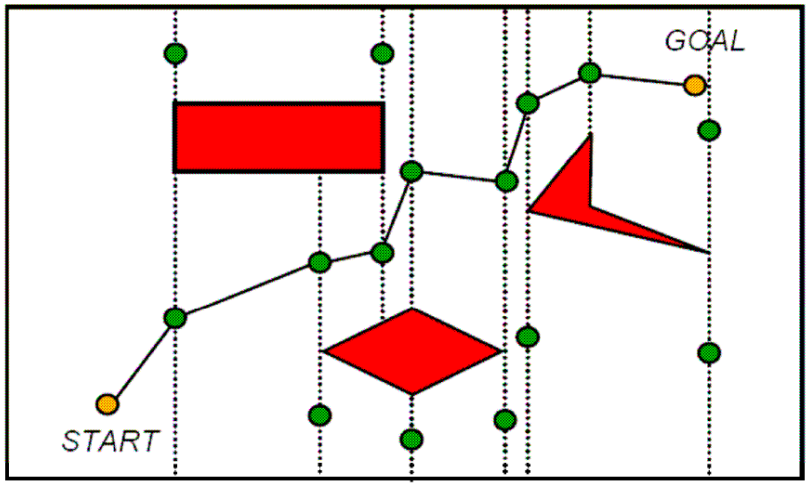
Obr. 1 Ukázka zaplnění buněk, do kterých zasahuje překážka.



Obr. 2 Pracovní prostor rozdělený aproximativní metodou do buněk o stejné velikosti

Exaktní rozklad do buněk

Buňky jsou vytvořeny nejčastěji tak, že se vytvoří kolmice k ose x a ty protínají hrany překážek, a tím vzniknou buňky (obr. 3). Vrcholy grafu jsou vždy na kolmici umístěny tak, že jsou v polovině mezi překážkou a jinou překážkou nebo hranicí pracovního prostoru. Hrany grafu tvoří spojnice mezi těmito vrcholy.



Obr. 3 Pracovní prostor rozdělen podle exaktního rozkladu buněk [2].

2.1.2 Mapy cest

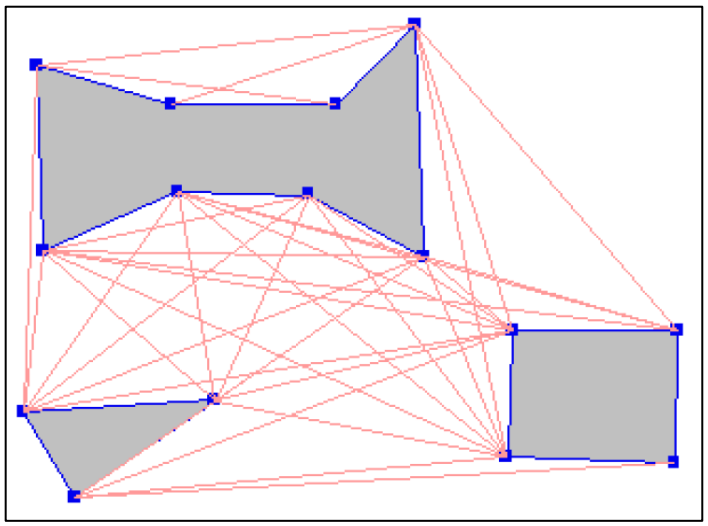
Mapy cest lze rozdělit na deterministické a pravděpodobnostní.

Deterministické mapy cest

Mezi deterministické mapy cest patří graf viditelnosti a Voronoiovův diagram.

Graf viditelnosti

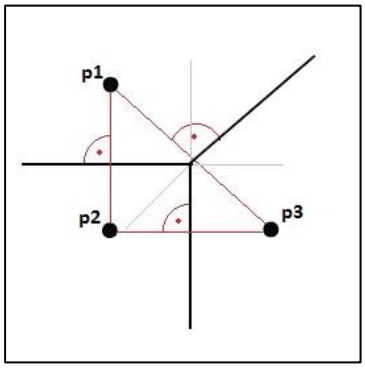
Graf viditelnosti je tvořen uzly, které jsou umístěny na hranách překážek a startovním a cílovým uzlem. Hrany grafu se utvoří tak, že se propojí jednotlivé uzly s tou podmínkou, že hrana grafu nesmí protnout překážku. Tato metoda je omezena pouze na polygonální překážky. Problém u této metody je, že čím více máme hran překážek tím je náročnější na výpočetní výkon.



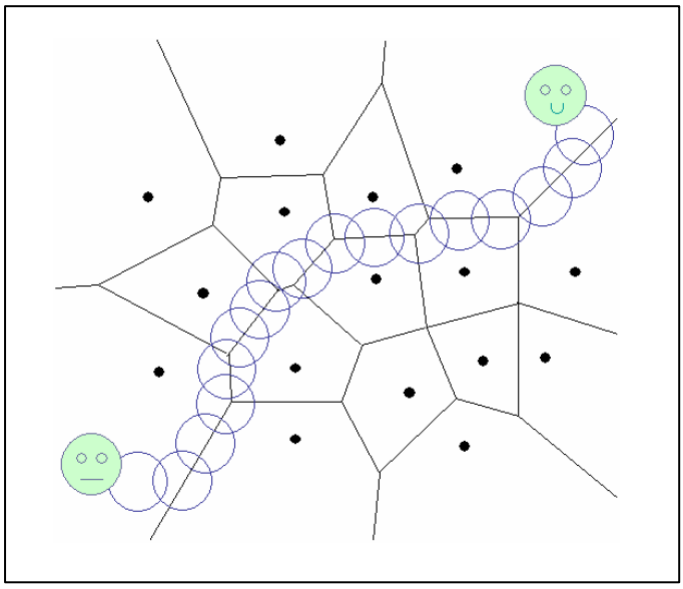
Obr. 4 Graf viditelnosti [1].

Voronoiovův diagram

Konstrukce diagramu se provádí tak, že vezmeme například tři body označené p1, p2, p3. Tyto body značí překážky. Tyto body propojíme (na obr. 5 fialové úsečky). K těmto úsečkám se ve středu vytvoří kolmice (na obr. 5 částečně černé a částečně šedé přímky). V místě, kde se protnou, vytvoří uzel. Poté se vymažou části přímky, které jsou na obr. 5 šedé. Tak nám vznikne Voronoiovův diagram, kdy uzly v diagramu jsou body styku úseček resp. polopřímek a hrany diagramu jsou samotné úsečky resp. polopřímky. Voronoiovův diagram lze aplikovat i na nebodové překážky [5].



Obr. 5 Konstrukce Voronoiova diagramu



Obr. 6 Voronoiovův diagram [5].

Pravděpodobnostní metody map cest

Pravděpodobnostní metody map cest mají dvě fáze. První fází je, že se vytvoří grafová struktura. Tvorba této grafové struktury probíhá tak, že se náhodně generují uzly náležící do prostoru C_{free} (tento prostor je tvořen body, které nejsou v kolizi

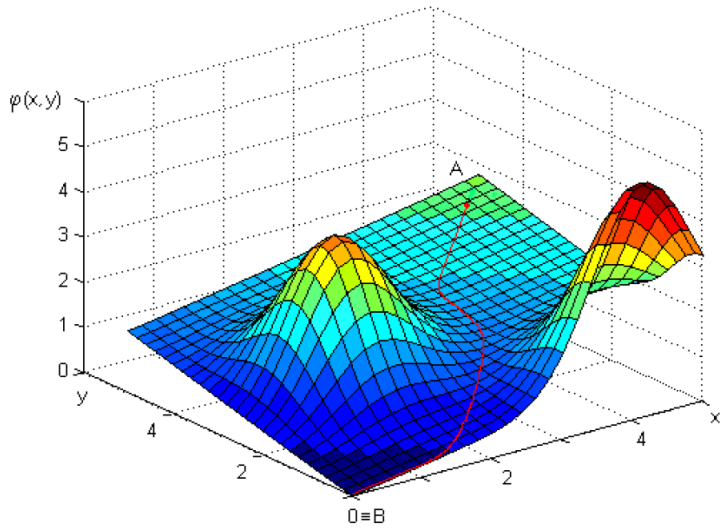
s překážkou). Poté se propojí tyto uzly hranami. Tato fáze probíhá tak dlouho, dokud není dostatečně popsán prostor C_{free}[2].

V druhé fázi (tj. zpracování dotazu na cestu) se připojí start a cíl stejně jako náhodně vygenerované uzly v první fázi. Poté začne hledání cesty[2].

Více o této problematice nalezneme v práci [2][6].

2.1.3 Potenciálové pole

Potenciálové pole se vytváří tak, že se celý pracovní prostor ohodnotí potenciálovou funkcí $\varphi(x,y)$. V potenciálovém poli má pak start vyšší hodnotu potenciálové funkce než cíl. Překážky v potenciálovém poli mají o mnoho větší potenciál než start. Pokud se podíváme na potenciálové pole jako celek, tak připomíná sjezdovou dráhu, kdy start je na vrcholu kopce, překážky jsou terénní vyvýšeniny a cíl je pod kopcem [2].



Obr. 7 Potenciálové pole [2].

3. Rojová inteligence

Rojová inteligence je umělá inteligence založena na chování živočichů určitého druhu v roji (hejnu, skupině). Název rojová inteligence zavedli Gerardo Beni a Jing Wang roku 1989 (v kontextu celulárních robotických systémů)[7].

Optimalizace mravenčí kolonií (Ant Colony Optimization), optimalizace včelím rojem, optimalizace hejnem světlušek (Firefly Swarm Optimization) a optimalizace hejnem částic (Particle Swarm Optimization) jsou algoritmy založeny právě na rojové inteligenci.

3.1 Optimalizace mravenčí kolonií

Optimalizace mravenčí kolonie (dále jako ACO) patří mezi metody rojové inteligence. Tento algoritmus byl inspirován chováním některých druhů mravenců. Jedná se o metaheuristickou optimalizaci.

Chování hmyzu pomocí feromonu (tento způsob je nazván stigmergy, podle článku [8]) zdokumentoval už ve čtyřicátých a padesátých letech dvacátého století francouzský entomolog Pierre-Paul Grassé. Další vědci poté následovali a vytvářeli modely chování hmyzů na základě feromonu. Marco Dorigo v roce 1992 ve své disertační práci jako první implementoval chování mravenců na hledání optimální cesty grafem. V dnešní době existuje mnoho modifikací tohoto algoritmu.

3.1.1 Reálná mravenčí kolonie

Některé druhy mravenců jsou schopny najít nejkratší cestu k potravě, i přes svůj hendikep, nedostatečný zrak. Mravenec jako jedinec je neefektivní. Z tohoto důvodu se využívá kolonie jako koordinovaného systému. Tito mravenci využívají systém stigmergy [8]. Je to systém založený např. na použití feromonů (chemického značení). Zajímavostí je, že každá kolonie mravenců má svůj vlastní feromon, kterým se rozeznají od jiného mravenčí kolonie stejného druhu. Feromony tak slouží ke komunikaci mezi mravenci.

3.1.2 Postup hledání cesty

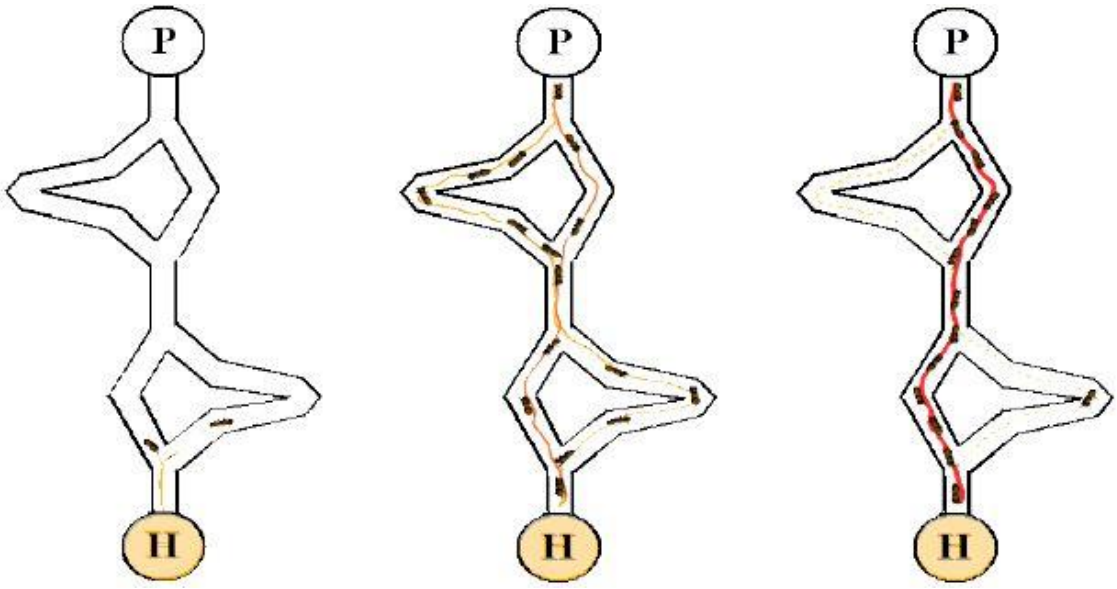
Na obr. 8 je dáno mraveniště (H) a potrava (P) pro mravence. Nalezení nejkratší cesty podle literatury [1], [8] a [10] ve skutečné kolonii probíhá tak, že se nejprve mravenci (hledači) náhodně pohybují kolem mraveniště, hledajíc potravu. U rozcestí vyberou náhodně cestu podle vzorce:

$$P_i(t) = \frac{(S_i(t)+k)^h}{\sum_{j=1}^{n}(S_j(t)+k)^h} \qquad (1)$$

kde $S_i(t)$ je intenzita feromonové stopy i-té cesty, k a h jsou konstanty, n je počet možných pokračování v cestě.

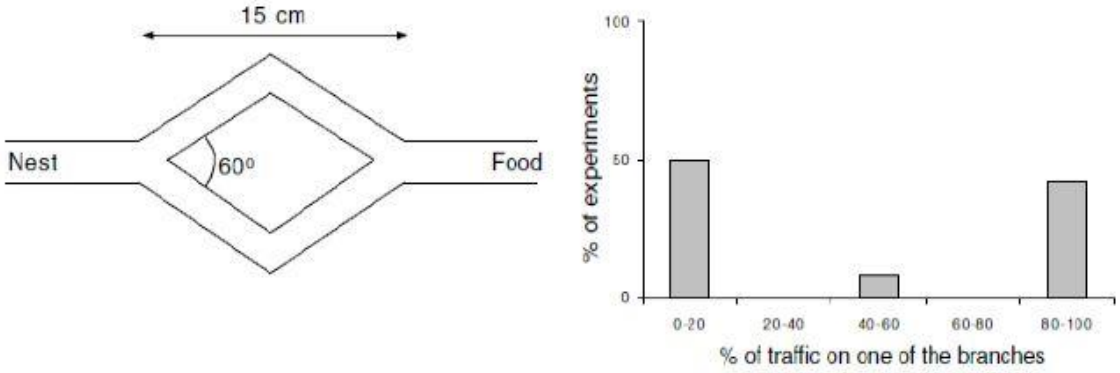
Až mravenec najde potravu, vrací se stejnou cestou zpět k mraveništi a za sebou zanechává feromonovou stopu. Ostatní mravenci buďto posílí tuto cestu svým feromonem, nebo najdou jinou. Síla feromonové stopy určuje atraktivnost cesty.

Feromonové stopy mají ještě tu vlastnost, že se časem vypařují a tím se ještě snižuje jejich atraktivnost. Po takto vyhledaných cestách nakonec zbude jen ta nejkratší, která má nejsilnější feromonovou stopu.



Obr. 8 Nalezení nejkratší cesty mezi mraveništěm a jídlem [1].

Pokud existují dvě stejné cesty, tak podle literatury [8] a [9] se cesta po čase ustálí na jedné cestě.



Obr. 9 Nalezení nejkratší cesty pro dvě stejně dlouhé cesty [1].

3.1.3 Ant System (AS)

Podle literatury [8] je Ant System navržený tak, že při každé iteraci jsou všem *m* mravencům aktualizovány feromonové hodnoty podle vzorce:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \qquad (2)$$

kde ρ je rychlost odpařování, m je počet mravenců a $\Delta\tau_{ij}^{k}$ je množství feromonu položené na hraně (i, j) mravencem k, kdy platí:

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k & \text{pokud mravenec } k \text{ použil hranu } (i,j) \text{ ve své cestě} \\ 0 & \text{jinak} \end{cases} \tag{3}$$

kde Q je konstanta a L_k je délka cesty konstruované mravencem k.

Pokud mravenec k je v uzlu i, zde máme dílčí řešení s^p. Pravděpodobnost, že mravenec přejde do uzlu j je dána vztahem:

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{c_{il}\in N(s^p)}\tau_{il}^{\alpha}\eta_{il}^{\beta}} & \text{jestliže } c_{il} \in \boldsymbol{N}(s^p) \\ 0 & \text{jinak} \end{cases} \tag{4}$$

kde $\boldsymbol{N}(s^p)$ je množina uzlů, které jsou spojeny hranami (i, l), l je uzel, který již mravenec k navštívil. Parametry α a β značí relativní význam feromonu proti heuristické informaci η_{ij}, která je dána vzorcem:

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{5}$$

kde d_{ij} je vzdálenost mezi uzly i a j.

3.1.4 Max-Min Ant Systém (MMAS)

Podle [8], tento algoritmus je modifikací původního algoritmu AS. Jeho hlavními charakterizujícími prvky jsou ty, že jen nejlepší mravenec aktualizuje feromonovou stezku a že hodnota feromonu je vázaná. Aktualizace feromonu je realizována následujícím vzorcem:

$$\tau_{ij} = \left[(1-\rho)\tau_{ij} + \sum_{k=1}^{m}\Delta\tau_{ij}^{best}\right]_{\tau_{min}}^{\tau_{max}} \tag{6}$$

kde τ_{max} a τ_{min} jsou horní a dolní mez uloženého feromonu, kdy operátor $[x]_b^a$ je definován takto:

$$[x]_b^a = \begin{cases} a & x > a \\ b & x < b \\ x & \text{jinak} \end{cases} \tag{7}$$

a $\Delta\tau_{ij}^{best}$ je:

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/L_{best} & \text{jestliže } (i,j) \text{ patří k nejlepší cestě} \\ 0 & \text{jinak} \end{cases} \tag{8}$$

kde L_{best} je délka cesty nejlepšího mravence. To může být buď nejlepší cesta nalezená v aktuální iteraci (L_{ib}) nebo dosud nejlepší nalezené řešení (L_{bs}).

Pokud jde o spodní a horní hranice τ_{max} a τ_{min} jsou obvykle získávány empiricky a následně upraveny na daný problém. V některých případech je dobré definovat τ_{max} a τ_{min} na základě analytických úvah.

3.1.5 Ant Colony System (ACS)

Další modifikace podle práce [8] je ACS. Tento ACS staví na základě MMAS. Aktualizace feromonu se provádí na konci každé iterace jenom nejlepším mravencem a aktualizace se provádí dle následujícího vzorce:

$$\tau_{ij} = \begin{cases} (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij} & \text{jestliže } (i,j) \text{patří k nejlepší cestě} \\ \tau_{ij} & \text{jinak} \end{cases} \quad (9)$$

kde stejně jako u MMAS je $\Delta\tau_{ij}$ spočteno podle vzorce (8), kde L_{best} může nabývat hodnot L_{ib} nebo L_{bs} jako u MMAS.

Poté ještě za běhu algoritmu probíhá lokální aktualizace feromonu, která je dána rovnicí:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \varphi\tau_0 \quad (10)$$

kde φ náleží do intervalu $(0,1\rangle$

Dalším důležitým rozdílem dle [8] mezi ACS a AS, je rozhodovací pravidlo, které dle pravděpodobnosti rozhoduje o pohybu mravence z uzlu *i* do uzlu *j*. V ACS se toto pravidlo nazývá *„pseudorandom proportional“* a závisí na náhodné proměnné *q*, která je v intervalu $\langle 0,1\rangle$ a parametru q_0, takže platí:

$$j = \begin{cases} \arg max_{c_{il} \in \boldsymbol{N}(s^p)} \left\{ \tau_{il}^{\alpha} \eta_{il}^{\beta} \right\} & \text{pokud } q \leq q_0 \\ \text{podle vzorce(3)} & \text{jinak} \end{cases} \quad (11)$$

Další modifikace jsou popsány v práci [1].

Pseudokód ACO

Pseudokód podle článku [10]:

1. Vygeneruje se počáteční matice feromonů **P**
2. Iterace = 0
3. **while** dokud nejsou splněny ukončující podmínky **do**
4. Náhodně rozmístění *m* mravenců na uzly grafu, kdy množství potravy C^k=0
5. **foreach** *k* **do**
6. Jdi vpřed podle rovnice (4)
7. Vyhodnocení kvality nalezeného řešení mravencem
8. Aktualizace feromonů podle vzorce (2)
9. **end**
10. **++iterace**
11. **end**

Jedna z možných aplikací ACO na plánování cesty je detailně popsaná v práci [1].

3.2 Optimalizace včelím rojem

Pro optimalizaci včelím rojem existuji dvě podskupiny. První je metoda napodobující pářicí chování včel a druhá je inspirovaná sháněním potravy.

Zásadní roli zde hraje přímá výměna informací oproti ACO, kde informace jsou vyměňovány pomocí feromonů.

3.2.1 Metody napodobující pářicí chování

Honeybee Matin Optimisation Algorithm (HBMO) podle článku [13] vytvořil v roce 2001 Abbass. Jedná se o metaheuristickou metodu. Původně byla tato metoda určena k řešení splnitelnosti booleovských formulí. Dnes lze použít na spoustu dalších problémů.

Podle článků [13-15] se u HBMO nejprve nastaví dělnice a počáteční populace (náhodně vygenerovaná). Nejlepší členové z počáteční populace včel jsou vybrány jako královny úlu, všichni ostatní jsou trubci. Dále se musí definovat kapacita královnina semenného váčku (spermatéky – spermatheca). Tato čísla odpovídají maximálnímu počtu páření v jednom letu párování.

Podle článku [13] na začátku svatebního letu, kdy svatební let je reprezentován jako sled pohybů stavovým prostorem, královny si zcela náhodně nastaví energii a rychlost.

Po zahájení svatebního letu královen se v každé iteraci změní rychlost letu královen (11), energie královen (12) a podle pravděpodobnosti úspěšnosti páření. Podle vzorce (13) se královna spáří s trubci a tím dojde k přidání informací (genotyp). Jedná se o informaci aktuální polohy královny ve stavovém prostoru, do semenného váčku. Svatební let královny končí, když semenný váček je plný nebo energie královny je na prahové hodnotě.

$$S(t+1) = \alpha * S(t) \tag{11}$$

kde $S(t)$ je rychlost královny, α náleží do intervalu $\langle 0,1 \rangle$.

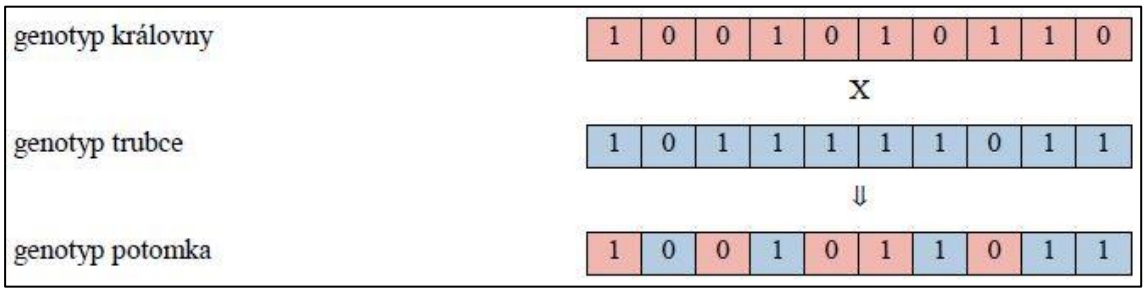
$$E(t+1) = E(t) - \gamma \tag{12}$$

kde $E(t)$ je energie královny a γ je hodnota snížení energie po každé iteraci.

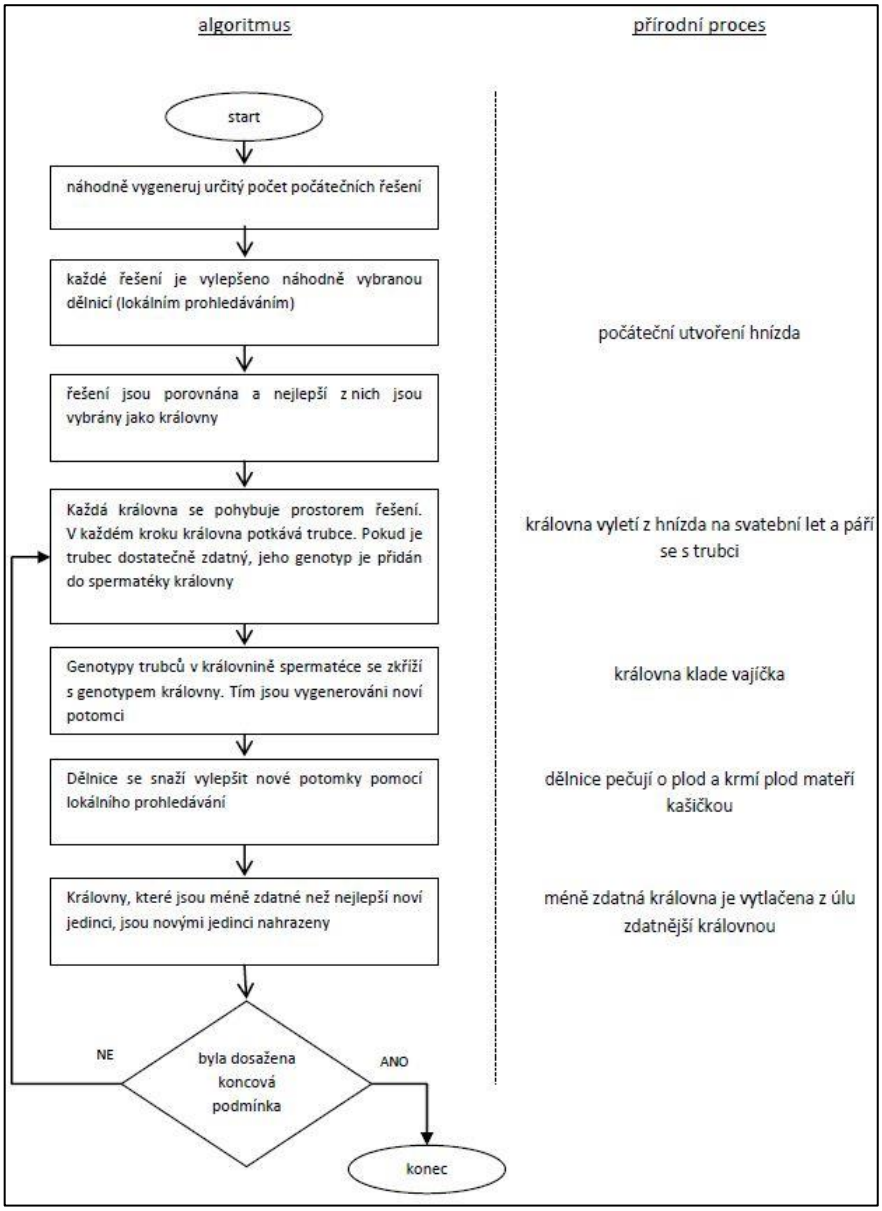
$$Prob(Q,D) = e^{-\frac{\Delta(f)}{S(t)}} \tag{13}$$

kde *Prod(Q,D)* je pravděpodobnost úspěšnosti páření mezi královnou *Q* a trubcem *D*, *Δ(f)* je absolutní rozdíl vhodnosti mezi trubcem *D* a královnou *Q*, *S(t)* je rychlost královny.

Po svatebním letu dojde ke zkřížení genotypů trubců ze semenného váčku a genotypu královny a královna naklade vajíčka (řešení). Zde začnou působit dělnice, jedná se heuristický algoritmus, který vylepšuje řešení (larvy) pomocí lokálního prohledávání. Pokud nová řešení (larvy) jsou lepší než královna, tak ji nahradí.



Obr. 10 Princip jednoduchého operátoru křížení pro binární reprezentaci [16].



Obr. 11 Diagram algoritmu HBMO [16].

3.2.2 Metody inspirované sháněním potravy

Umělý včelí roj byl navržen roku 2005 Karabogou, kdy ABC byl založen na chování roje včely medonosné. Podle článků [11] a [12] ABC je navržen pro multi-variabilní a multi-modální spojité funkce optimalizace.

Základní popis umělého včelstva – volný optimalizační problém

Podle článků [11] a [12] je včelstvo rozděleno do tří skupin a to, včely dělnice, diváci a zvědi. Roj je v počátku rozložen na dvě poloviny. První polovinu tvoří včely dělnice a druhou polovinu včely diváci. Na každém zdroji potravy je jedna včela dělnice. Včely dělnice, jejichž zdroj potravy byl odmítnut, se stávají zvědy.

V algoritmu ABC poloha zdroje potravy představuje možné řešení a množství nektaru zdroje odpovídá kvalitě fitness funkce. Počet včel dělnic nebo včel diváků se rovná počtu řešení v populaci. V prvním kroku algoritmus ABC vygeneruje náhodně počáteční populaci $P(G = 0)$ z *SN* řešení, kdy *SN* je velikost populace. Každé řešení x_i $(i = 1,2, \dots, SN)$ je *D*-rozměrný vektor. Po inicializaci se seznam pozic podrobí opakovanému cyklu $C = 1,2, \dots, MCN$. Včely dělnice mění polohu a testují množství nektaru (fitness hodnotu), zdroje potravy. Za předpokladu, že nektar (fitness hodnota) je lepší než předchozí, tak včela dělnice si zapamatuje polohu nového zdroje a starou zapomene. Pokud nektar (fitness hodnota) není lepší než předchozí, tak si v paměti uchovává předchozí polohu. Poté, co všechny včely dělnice ukončí hledací proces, předají informace o nektaru a jeho poloze včelám divákům na tzv. tančící ploše. Včely diváci vyhodnotí předané informace od všech včel dělnic a zvolí zdroj potravy s pravděpodobností podle vztahu (14). Stejně jako v případě včel dělnic si pamatují poslední nejlepší hodnotu nektaru (fitness hodnotu) a jeho polohu.

Včely diváci zvolí zdroj potravy v závislosti na pravděpodobnosti spjaté s daným zdrojem p_i podle rovnice:

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \tag{14}$$

kde fit_i je fitness hodnota v poloze *i* a *SN* je počet zdrojů potravy, který je roven počtu včel dělnic.

Aby bylo možné vytvořit kandidáta na novou pozici zdroje, ABC používá následující rovnici:

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \tag{15}$$

kde $k \in \{1,2, \dots, SN\}$ a $j \in \{1,2, \dots, D\}$ jsou indexy (když je *k* určen, tak *i* musí být odlišný od *k*), ϕ_{ij} je náhodné číslo mezi $\langle -1,1 \rangle$. To řídí produkci v sousedství zdrojů kolem x_{ij} a představuje srovnání dvou poloh zdrojů.

Překročí-li v_{ij} předem nastavené omezení, může být hodnota v_{ij} nastavena na přijatelnou hodnotu. Většinou je to řešeno tak, že tato hodnota se nastaví na omezující hodnotu.

Zdroje opuštěné včelami jsou nahrazovány novými zdroji poskytnutými zvědy. V ABC pokud už danou pozici není možné zlepšit pomocí předem daných cyklů, je zdroj potravy opuštěn. Hodnota daných cyklů je v algoritmu ABC důležitým kontrolním

parametrem, který se nazývá „limit“ (mez) pro opuštění. Předpokládejme, že opustíme zdroj x_i a $j \in \{1,2,\dots,D\}$, pak zvěd objevuje nový zdroj potravy, který nahrazuje x_i. Tato operace může být definovaná takto:

$$x_i^j = x_{min}^j + rand(0,1)(x_{max}^j - x_{min}^j) \qquad (16)$$

Poté se každý kandidát na pozici zdroje v_{ij} vyhodnotí, jestli není lepší než předešlí kandidáti v paměti. V případě, že nový zdroj potravy má stejnou nebo lepší hodnotu než původní zdroj, je starý nahrazen novým. V opačném případě je starý zachován v paměti. Jinými slovy je hladový (greedy) mechanismus výběru použit jako výběrová operace mezi starým a novým kandidátem.

Jak vyplývá z výše uvedených vysvětlení, existují čtyři ovládací parametry:
Počet zdrojů potravy

1. Počet včel dělnic a diváků (*SN*)
2. Hodnota „limit“
3. Maximální počet cyklů (*MCN*)

Pseudokód algoritmu ABC pro volný optimalizační problém

Pseudokód algoritmu ABC pro volný optimalizační problém podle článku [12]:

1. Inicializuje se populace řešení $x_{ij}, i = 1,\dots,SN, j = 1,\dots,D$
2. Vyhodnotí se populace
3. cyklus=1
4. **repeat**
5. Vytvoří se nová řešení v_{ij} pro včely dělnice pomocí (15) a vyhodnotí se
6. Aplikuje se hladový výběrový mechanismus
7. Vypočítá se pravděpodobnosti P_{ij} pro řešení x_{ij} podle (14)
8. Vytvoří se nova řešení v_{ij} pro včely diváky z řešení x_{ij} vybraných podle P_{ij} a vyhodnotí se
9. Aplikuje se hladový výběrový mechanismus
10. Určí se opuštěné řešení pro zvěda (pokud existuje) a nahradí se novým náhodně vytvořeným řešením x_{ij} pomocí (16)
11. Zapamatuje se nejlepší dosud dosažené řešení
12. cyklus = cyklus +1
13. **until** cyklus=MCN

Základní popis umělého včelstva – vázaný optimalizační problém

Podle článku [12], aby se algoritmus ABC přizpůsobil pro vázaný optimalizační problém, byla přijata Debiho vazebná metoda na místo chamtivý mechanismu výběru. Debiho metoda se skládá ze tří velmi jednoduchých heuristických pravidel:

1. Každému proveditelnému řešení se dává přednost před jakýkoliv neproveditelným řešením.

2. Mezi dvěma přípustnými řešeními, z nichž jedno má lepší účelovou funkci, se dává přednost tomu, které má lepší účelovou funkci.
3. Mezi dvěma nepřípustnými řešeními se dává přednost tomu, které má menší narušení mezí.

Na základě těchto pravidel bylo možné vytvořit kandidáta použitím následující rovnice:

$$v_j = \begin{cases} x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) & R_j < MR \\ x_{ij} & \text{jinak} \end{cases} \quad (17)$$

kde $k \in \{1,2,\dots,SN\}$ je index, (jestliže index k je určen, tak i musí být odlišný od k), R_j je náhodně vybrané číslo v rozsahu $\langle 0,1 \rangle$ a $j \in \{1,2,\dots,D\}$. MR (změna vazby) je řídicí parametr, který určuje, zda parametr x_{ij} bude změněn.

V algoritmu ABC jsou zvědi vytvářeni na předem stanovené období cyklů pro objevení nových zdrojů potravy v náhodném pořadí. Toto období (SPP) je dalším řídicím parametrem. V každém SPP cyklu je kontrolováno, zda je opuštěn zdroj potravy. Jeli tomu tak, zvěd provádí vyhledávací proces.

Pseudokód algoritmu ABC pro vázaný optimalizační problém

Pseudokód algoritmu ABC pro vázaný optimalizační problém podle článku [12]:

1. Inicializuje se populace řešení $x_{ij}, i = 1,\dots,SN, j = 1,\dots,D$
2. Vyhodnotí se populace
3. cyklus=1
4. **repeat**
5. Vytvoří se nová řešení v_{ij} pro včely dělnice pomocí (17) a vyhodnotí se
6. Použije se výběrový proces založený na Debiho metodě
7. Vypočítá se pravděpodobnosti P_{ij} pro řešení x_{ij} podle (14)
8. Vytvoří se nova řešení v_{ij} pro včely diváky z řešení x_{ij} vybraných podle P_{ij} a vyhodnotí se
9. Použije se výběrový proces založený na Debiho metodě
14. Určí se opuštěné řešení pro zvěda (pokud existuje) a nahradí se novým náhodně vytvořeným řešením x_{ij} pomocí (16)
15. Zapamatuje se nejlepší dosud dosažené řešení
16. cyklus = cyklus +1
17. **until** cyklus=MCN

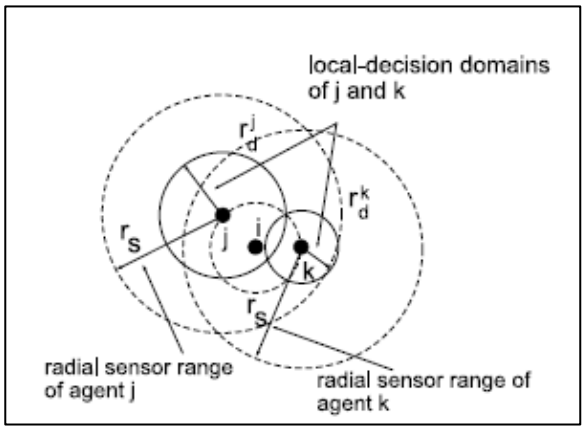
Jedna z možných aplikací pro plánování cesty pomocí včelího roje je zmíněna v článku [14].

3.3 Optimalizace roje světlušek (FSO)

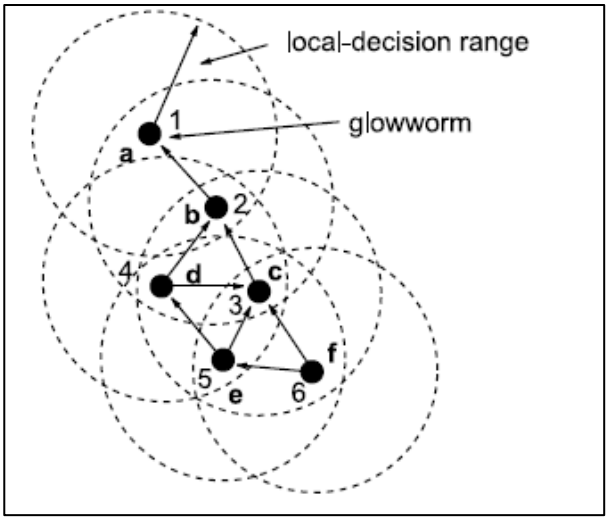
Tato technika je založena na biologickém chování světlušek. Byla poprvé publikována roku 2006 Krishnanandem a Ghosem.

Podle článku [17] ve FSO se hejno světlušek inicializuje náhodně do vyhledávajícího prostoru. V souladu s tím, že mají určité luminiscenční množství

s názvem „luciferin“, tak vyzařují ve tmě světlo, jehož intenzita je úměrná množství „luciferinu“, kterým světlušky komunikují s ostatními světluškami v proměnlivém okolí. Okolí je definováno jako lokální rozhodovací doména, která má proměnlivý rozsah r_d^i omezený radiálním snímačem r_s $(0 < r_d^i \leq r_s)$. Světluška *i* uvažuje ostatní *j*-té světlušky jako své sousedy. Jestliže světluška *j* je v dosahu sousedství světlušky *i* a světluška *j* má vyšší svítivost (větší množství luciferinu), tak světluška *i* je přitahována k *j*-té světlušce. Rozhodování světlušek ve FSO závisí pouze na informacích dostupných z jejich okolí. Například na obr. 12 máme světlušku *k* a světlušku *j*, které jsou stejně vzdáleny od světlušky *i*. Nicméně světluška *j* a světluška *k* mají různé velikosti lokálních rozhodovacích domén. Díky tomu světluška *j* používá informace od světlušky *i*. obr. 13 znázorňuje orientovaný graf založený na relativní úrovni luciferinu a dostupnosti pouze lokálních informací každé světlušky. Každá světluška vybere pomocí pravděpodobnostního mechanismu souseda, který má hodnotu luciferinu vyšší, než její vlastní a pohybuje se směrem k němu. Tyto posuny, které jsou založené na místních informacích a selektivních sousedských interacích umožní roji světlušek se rozdělit do disjunktivních podskupin, které směřují a setkají se ve více optimech dané multimodální funkce.



Obr. 12 Ukázka lokální rozhodovací domény [17].



Obr. 13 Orientovaný graf založený na relativní úrovni luciferinu a dostupnosti pouze lokálních informací každé světlušky [17].

3.3.1 Popis FSO

Podle článku [17] algoritmus FSO začíná tak, že náhodně umístí populaci *n* světlušek ve tmě do vyhledávacího prostoru, tak aby byla dobře rozmístěna. Zpočátku, všechny světlušky obsahují stejné množství luciferinu l_0. Každá iterace se skládá z aktualizace luciferinu, následuje pohyb na základě přechodového pravidla.

Aktualizace luciferinu je dána vztahem:

$$l_i(t+1) = (1-\rho)l_i(t) + \gamma J(x_i(t+1)) \tag{18}$$

kde $l_i(t)$ představuje hodnotu luciferinu spojeného se světluškou v čase *t*, ρ je útlumová konstanta luciferinu $(0 < \rho < 1)$, γ je posilovací konstanta luciferinu a $J(x_i(t+1))$ představuje hodnotu fitness funkce v místě světlušky v čase *t*.

Během pohybu každá světluška použije pravděpodobnostní mechanismus, který rozhodne o pohybu směrem k sousedovi, který má hodnotu luciferinu vyšší než její. Obr. 13 ukazuje orientovaný graf mezi souborem šesti světlušek na základě jejich hodnotě luciferinu a na místních informacích. Pokud vezmeme obr. 12 a vezmeme světlušky *a*, *b*, *c* a *d*, které mají vyšší hodnotu luciferinu než hodnota luciferinu světlušky *e*, tak světluška *e* je v dosahu oblasti C a D (kde C je lokální rozhodovací doména světlušky *c* a D je lokální rozhodovací oblast světlušky *d*). Takže světluška má na výběr dva možné pohyby. V takovém případě světluška *i* se pravděpodobnostně rozhodne, že se bude pohybovat směrem k sousedovi *j* podle vztahu:

$$p_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)} \tag{19}$$

kde $j \in N_i(t)$, $N_i(t) = \{j : d_{ij}(t) < r_d^i(t); l_i(t) < l_j(t)\}$ je množina sousedů světlušky *i* v čase *t*, $d_{ij}(t)$ představuje Euklidovskou vzdálenost mezi světluškou *i* a světluškou *j* v čase *t*. Z toho plyne, že světluška *i* vybere světlušku $j \in N_i$ s $p_{ji}(t)$ danou vzorcem (19), a tak může provést diskrétní pohyb světlušky podle vzorce:

$$x_i(t+1) = x_i(t) + s\left(\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}\right) \tag{20}$$

kde $x_i(t) \in R_m$ je umístění světlušky *i* v čase v *m*-rozměrném reálném prostoru R_m, $\|.\|$ představuje Euklidovský normový operátor a $s > 0$ je délka kroku.

Dále se musí se aktualizovat rozsah okolí r_d^i a to pomocí vzorce:

$$r_d^i(t+1) = min\left\{r_s, max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\right\} \tag{21}$$

kde β je konstantní parametr a n_t je parametr sloužící k řízení počtu sousedů.

Možné aplikace pro plánování cesty pomocí FSO jsou zmíněny v článcích [24] a [30].

Algoritmus optimalizace rojem světlušek

Algoritmus optimalizace rojem světlušek podle článku [17]:

Nastaví se počet dimenzí = *m*
Nastaví se počet světlušek = *n*
Nechť *s* je velikost kroku
Nechť $x_i(t)$ je umístění světlušky *i* v čase *t*
Náhodně se vygenerují světlušky
for *i* = 1 *n* do $l_i(0) = l_0$
$r_d^i(0) = r_0$
Nastaví se maximální počet iterací = *iter_max*
Nastaví se *t* = 1
while ($t \leq$ *iter_max*) do
{
pro každou světlušku *i* proveď (fáze aktualizace luciferinu)

$l_i(t)$ vypočteme pomocí vztahu (18)

pro každou světlušku *i* proveď (pohybová fáze)
{
$$N_i(t) = \{j: d_{ij}(t) < r_d^i(t); l_i(t) < l_j(t)\}$$

pro každou světlušku $j \in N_i(t)$ proveď

$p_{ij}(t)$ vypočteme pomocí vztahu (19)

j = *vyber_svetlusku*($\vec{p}$)

$x_i(t+1)$ vypočteme pomocí vztahu (20)

$r_d^i(t+1)$ vypočítáme pomocí vztahu (21)
}
$t \leftarrow t + 1$
}

3.4 Optimalizace hejnem částic (Particle Swarm Optimalization - PSO)

Tuto metodu jsem se po dohodě s vedoucím diplomové práce RNDr. Jiřím Dvořákem, CSc. rozhodl implementovat v programu.

Optimalizace hejnem částic je stochastická nelineární metoda, kterou publikovali v článku v roce 1995 R. C. Eberhart (Purdue School of Engineering and Technology, Indianapolis, Indiana) a J. Kennedy (Bureau of Labor Statistics , Washington, DC) [19].

Tento rojový algoritmus byl založen na chování hejna ptáků a ryb, kdy základní pravidla definoval už zoolog Heppner, který právě studoval chování a pohyby hejna ptáků. Spolu s Grenanderem vytvořili model chování ptáků [20].

Model simuloval pohyb ptáků, kteří směřovali k hnízdišti. A zde byly použity právě Heppnerova dvě pravidla. První pravidlo bylo, že pták se snaží být co nejblíže sousedním ptákům a přitom do nich nenarazit. Druhé pravidlo je spjato s hnízdištěm –

pokud pták najde hnízdiště, tak mu instinkt napovídá raději přistát, než zůstat s hejnem. Hejno ptáků následuje tohoto jedince a přistává v hnízdišti [21].

Základní popis PSO

Každá částice má k dispozici dva druhy informací. První informace je vlastní zkušenost – znalost, jaký stav byl nejlepší, a jak dobrý byl. Druhá informace je sociální znalost o chování, sousedních částic [22].

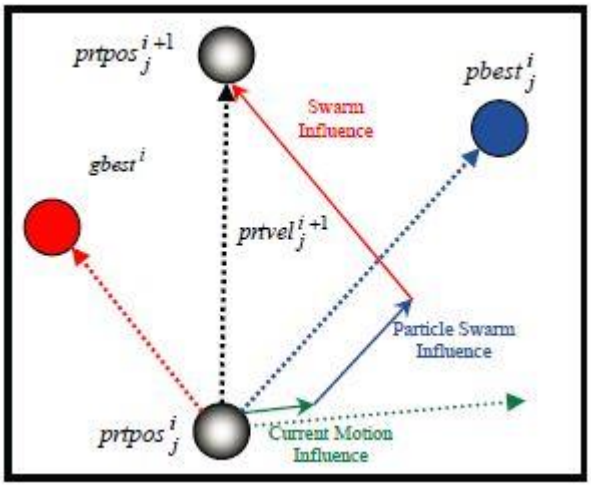
Každá částice je reprezentována jako *n*-dimenzionální vektor pozice $x_i(t)$ a k ní odpovídající okamžitý vektor rychlosti $v_i(t)$. Mimo jiné si pamatuje nejlepší hodnotu fitness funkce a postavení p_i. p_g je nejlepší pozice celého roje. Během každé iterace je aktualizovaný vektor rychlosti (22) a polohy (23) [22].

$$\vec{v}_i(t+1) = \vec{v}_i(t) + c_1 * r_1 * \left(\vec{p}_i(t) - \vec{x_i}(t)\right) + c_2 * r_2 * \left(\vec{p}_g(t) - \vec{x_i}(t)\right) \quad (22)$$

$$\vec{x}_i(t+1) = x_i(t) + \vec{v}_i(t+1) \quad (23)$$

kde $i=1,2,3,..,P$ (P je počet částic v roji), $t=1,2,3,..,T$ (T je počet cyklů), c_1 a c_2 jsou parametry učení, které nabývají hodnot v intervalu $\langle 0,2 \rangle$, r_1 a r_2 jsou náhodná čísla z intervalu $\langle 0,1 \rangle$.

Pokud některá rychlost $\vec{v}_i(t)$ je menší jak $-V_{max}$ nebo větší jak V_{max}, je hodnota rychlosti nahrazena hodnotou $-V_{max}$ nebo V_{max}, podle toho kterou překročily. V_{max} je maximální rychlost částice. Během každé iterace se aktualizuje rychlost (22) a poloha (23) každé částice. Spolu s aktualizací rychlosti a polohy se rovněž aktualizuje p_i a p_g. Algoritmus se zastaví, až nastane maximální počet iterací nebo je splněno jiné kritérium.



Obr. 14 Aktualizace částice [25].

3.4.1 Modifikace se setrvačnou váhou

Modifikace proběhla u výpočtu vektoru rychlosti podle vzorce (24). Vektor rychlosti $\vec{v_i}(t)$ se vynásobí $w(t)$, což je setrvačná váha, která se při každé iteraci

aktualizuje podle vzorce (25) nebo podle vzorce (26) anebo se také nastavuje jako konstanta, což nás vrací zpět k původnímu algoritmu.

$$\vec{v}_i(t+1) = w * \vec{v}_i(t) + c_1 * r_1 * \left(\vec{p}_i(t) - \overrightarrow{x_i}(t)\right) + c_2 * r_2 * \left(\vec{p}_g(t) - \overrightarrow{x_i}(t)\right) \quad (24)$$

$$w(t) = w_{min} - \frac{w_{max} - w_{min}}{T} * t \quad (25)$$

kde w_{min} je počáteční váha a w_{max} je koncová váha.

$$w(t) = \frac{1}{1 + exp\left(0.015\left(t - \frac{T}{3}\right)\right)} \quad (26)$$

3.4.2 Modifikace s koeficientem zúžení

Podle článku [23] bylo nutné zavést nějakou formu útlumu pro rychlost částice mimo V_{max}, jelikož, když je bez omezení, tak rychlost částice rychle vzroste na nepřijatelné hodnoty. Z tohoto důvodu byl zaveden koeficient zúžení (constriction coefficient) χ. Pak původní rovnice (22) má podobu:

$$\vec{v}_i(t+1) = \chi\left[\vec{v}_i(t) + c_1 * r_1 * \left(\vec{p}_i(t) - \overrightarrow{x_i}(t)\right) + c_2 * r_2 * \left(\vec{p}_g(t) - \overrightarrow{x_i}(t)\right)\right] \quad (27)$$

kde χ je určen takto:

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \quad (28)$$

kde $\phi = c_1 + c_2 > 4$,

Je-li použita Clercova metoda zúžení a ϕ je roven 4.1, kdy $c_1 = c_2$, tak koeficient zúžení χ má přibližnou hodnotu 0,7298.

3.4.3 Ukázka použití PSO na minimalizaci funkce

Uvažujeme funkci, která je zadaná takto:

$$f = (x+3)^2 + (y+2)^2 \quad (29)$$

kdy *x* a *y* jsou souřadnice bodů v 2D prostoru.

Hledáme minimum funkce (29). Hledání začneme tak, že nastavíme parametry pro výpočet vektoru rychlosti (c_1, c_2 případně $w(t)$ jako koeficient), poté zvolíme také maximální počet iterací *max_iteration* a velikost populace *P*.

Algoritmus nejprve vygenerujeme populaci *P* v 2D prostoru s příslušnými rychlostmi. Poté se spočítá hodnota funkce (29) pro každou částici a nastaví se nejlepší poloha $p_i(t)$ částice *i*. Následně se určí nejlepší globální poloha roje $p_g(t)$.

Poté následuje smyčka, ve které se vypočítá pro každou částici nová rychlost a poloha dané částice. Například pokud máme částici s polohou $x = 150, y = 200$ a

rychlost $v_x = -5, v_y = -2$, koeficienty $c_1 = c_2 = 1.4$, $w(t) = 0.7298$, $p_i(t) = [145,195]$ a $p_g(t) = [145,180]$, tak nový vektor podle (24) bude:

$$\vec{v}_i(t+1) = w * \vec{v}_i(t) + c_1 * r_1 * \left(\vec{p}_i(t) - \overrightarrow{x_i}(t)\right) + c_2 * r_2 * \left(\vec{p}_g(t) - \overrightarrow{x_i}(t)\right)$$
$$= 0.7298 * [-5,-2] + 1.4 * 0.8 * ([145,195] - [150,200]) + 1.4$$
$$* 0.5 * ([140,180] - [150,200]) \approx [-16,-21]$$

A nová poloha podle (23) bude:

$$\vec{x}_i(t+1) = x_i(t) + \vec{v}_i(t+1) = [150,200] + [-16,-21] \approx [134,179]$$

Poté se spočítá hodnota funkce (29) pro každou částici a nastaví se nejlepší poloha $p_i(t)$ částice *i*. Následně se určí nejlepší globální poloha roje $p_g(t)$.

Zkontrolují se ukončující podmínky např. *iteration=max_iteration*. Pokud nejsou splněny, pokračuje se v cyklu. Pokud jsou ukončující podmínky splněny, pak se ukončí cyklus a vypíše se výsledek (správný výsledek minimalizace funkce (29) je $x = -3, y = -2$).

Algoritmus PSO

Algoritmus PSO podle článku [23]:

1. Inicializuje se populace částic s náhodnými pozicemi a rychlostmi v *D*-dimenzionálním prohledávácím prostoru.
2. **loop**
3. Pro každou částici se vyhodnotí optimalizační fitness funkce *D* proměnných.
4. Porovná se hodnota fitness funkce částice *i* s $pbest_i$. Pokud je aktuální hodnota lepší než $pbest_i$, pak se $pbest_i$ nastaví na aktuální hodnotu a p_i se položí rovné aktuální pozici částice x_i.
5. Určí se částice s nejlepší hodnotou fitness funkce a její index se vloží do proměnné *g*.
6. Provede aktualizaci poloh a rychlostí částic podle rovnic (22) a (23).
7. Je-li ukončující kritérium splněno (obvykle maximální počet iterací nebo dostatečně dobrá hodnota fitness funkce) vyskoč ze smyčky
8. **end loop**

3.4.4 PSO s Fergusonovými spliny

Jedna zajímavá aplikace PSO pro plánování cesty je v článku [22], kdy na PSO je použit fitness funkce založena na Fergusonových splinech.

Fergusonův spline

Podle článku [22] kubické Fergusonovy spliny jsou vhodné po plánování cesty robota. Spliny napodobují přírodní pohyb, který je plynule propojen. Fergusonův spline je definovaný vztahem:

$$k: X(t) = P_0F_1(t) + P_1F_2(t) + P_0'F_3(t) + P_1'F_4(t) \quad (30)$$

kde $t \in <0,1>$ je parametr, P_i a P_i' jsou vektory, které definují spline a F_i jsou Fergusonovy multinomiály popsány takto:

$$F_1(t) = 2t^3 - 3t^2 + 1 \quad (31)$$

$$F_2(t) = -2t^3 + 3t^2 \quad (32)$$

$$F_3(t) = t^3 - 2t^2 + t \quad (33)$$

$$F_4(t) = t^3 - t^2 \quad (34)$$

Pokud do vztahu dosadíme počáteční bod $X(0)$, pak spline se rovná vektoru P_0, a koncový bod $X(1)$ se rovná vektoru P_1. Hodnoty bodů $P_i', i = \{1,2\}$ jednoduše získáme derivací Fergusonových multinomiálů:

$$F_1'(t) = 6t^2 - 6t \quad (35)$$

$$F_2'(t) = -6t^2 + 6t \quad (36)$$

$$F_3'(t) = 3t^2 - 4t + 1 \quad (36)$$

$$F_4'(t) = 3t^2 - 2t \quad (37)$$

P_0' je vektor tečny v počátečním bodě P_0 a P_1' je vektor tečny v koncovém bodě P_1. Tato implementace kubických funkcí umožňuje připojit dva spliny pomocí jednoduchého pravidla. Pokud máme další spline:

$$\bar{k}: \bar{X}(t) = \bar{P}_0\bar{F}_1(t) + \bar{P}_1\bar{F}_2(t) + \bar{P}_0'\bar{F}_3(t) + \bar{P}_1'\bar{F}_4(t) \quad (38)$$

Spojitost první derivace ve spojení splinů k a $\bar{k}$ je zaručena následujícími identitami:

$$P_1 \equiv \bar{P}_0, P_1' \equiv \bar{P}_0' \quad (40)$$

Spojitost první derivace je nezbytná pro plynulý pohyb robota.

Popis částic a vyhodnocení

Podle článku [22] plánování cesty pro mobilního robota může být realizováno prostřednictvím vyhledávání v prostoru funkcí. Redukujeme takový prostor do subprostoru, který obsahuje pouze řetězce kubických splinů. Matematický zápis splinu k je dán rovnicí (30) ve 2D prostoru může být popsána takto:

$$f(t) = a_x t^3 - b_x t^2 + c_x t + d_x$$
$$g(t) = a_y t^3 - b_y t^2 + c_y t + d_y \quad (41)$$

kde *a*, *b*, *c* a *d* jsou konstanty definované rovnicemi:

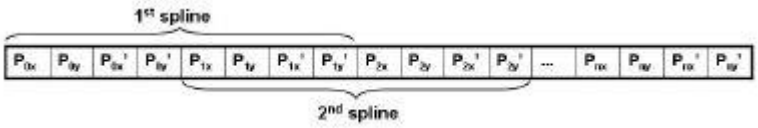
$$a = 2P_0 - 2P_1 + P_0' + P_1' \quad (42)$$

$$b = -3P_0 + 32P_1 - 2P_0' - P_1' \quad (43)$$

$$c = P_0' \quad (44)$$

$$d = P_1' \quad (45)$$

Každý spline je definován pouze body P_0, P_1 a vektory P_0' a P_1'. Podle identit (40), každý následný splin v řetězci sdílí koncový bod a k němu odpovídající vektor. Celkový počet proměnných, které definují celou trajektorii ve 2D prostoru je roven *6n*, kde *n* je počet splinů v řetězci. Struktura každé částice, která se používá, pro optimalizaci je znázorněna na obr. 15.



Obr. 15 Struktura každé částice [22].

Jedním z nejdůležitějších částí PSO algoritmu je najít dobré ohodnocení částic. Globální minimum takové funkce odpovídá plynulé trajektorii, která je bezpečná (tj. dostatečně vzdálená od překážky), ale nesmí být příliš dlouhá. Optimalizační funkce by měla nadhodnocovat trajektorie, které způsobí kolizi s překážkou.
Pro algoritmus PSO je použita fitness funkce, která má podobu:

$$f = \frac{l}{l_{min}} + \left(\frac{\alpha}{d}\right)^2 \quad (46)$$

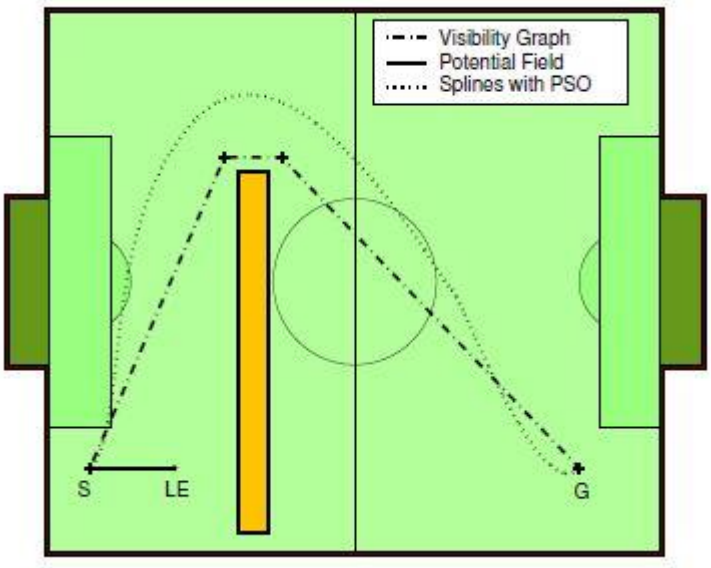
kde l_{min} je Euklidovská vzdálenost mezi částicí a požadovaným cílem, α je konstanta, která určuje vliv překážky, *l* a *d* popisují vlastnosti každé částice. Konkrétně *l* je délka trajektorie vypočtena podle rovnice:

$$l = \int_0^1 \sqrt{(f'(t))^2 + (g'(t))^2}\, dt \quad (47)$$

d je minimální vzdálenost mezi trajektorií a nejbližší překážkou definovaný vztahem:

$$d = \min_{o \epsilon O} \min_{t \epsilon <0,1>} \sqrt{(f(t) - o_x)^2 + (g(t) - o_y)^2} \quad (48)$$

kde f a g jsou funkce definované rovnicemi (41) a *O* je množina všech překážek v prostoru robota.



Obr. 16 Plánování cesty pomocí PSO s Fergusonovy spliny [22].

4. Plánování cesty robota pomocí PSO

Tato část je inspirovaná články [26],[27], [28] a [29].

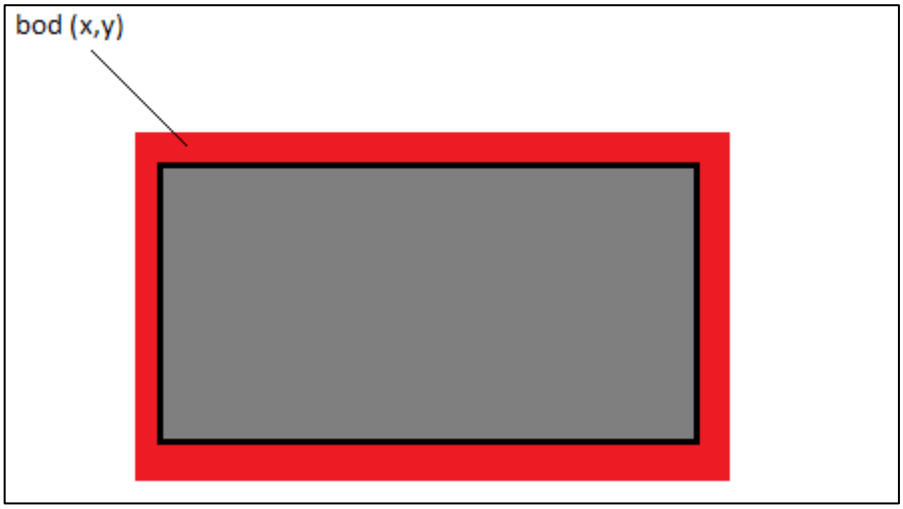
4.1 Potenciálové pole

Tato metoda slouží v programu jako metoda porovnávací. Potenciálové pole v programu funguje tak, že se nejprve ohodnotí pole, které je diskrétním pracovním prostorem. Ohodnocení potenciálu jednotlivých bodů probíhá pomocí vztahu:

$$\varphi(x,y) = \sqrt{(x_p - x_g)^2 + (y_p - y_g)^2} \tag{49}$$

kde x_p a y_p jsou souřadnice aktuálně hodnoceného bodu a x_g a y_g jsou souřadnice cílového bodu.

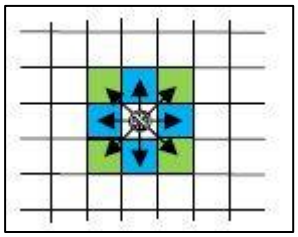
Pokud vytvoříme překážku a ohodnocujeme potenciál bodu (x, y), který je na obr. 17 vyznačen v nedovolené blízkosti překážky (na obr. 17 vyznačeno červenou barvou), tak vypočet potenciálu takového bodu bude proveden podle rovnice (49) a k ní se přičte kladné nekonečno. V důsledku to znamená, že pro daný bod je potenciál roven kladnému nekonečnu.



Obr. 17 Bod v nedovolené blízkosti překážky.

Musíme rovněž ohodnotit potenciál překážky, proto body, které tvoří překážku, ohodnotíme kladným nekonečnem.

Pokud máme vytvořené potenciálové pole, tak dojde k vytvoření cesty. To v programu probíhá tak, že se posuneme vždy o jedno pole (obr. 18) od bodu s větším potenciálem do bodu s menším potenciálem.

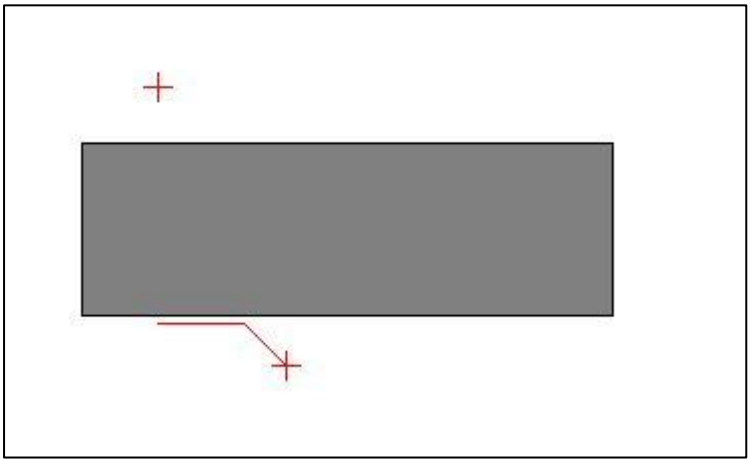


Obr. 18 Pohyb robota o jedno pole [18].

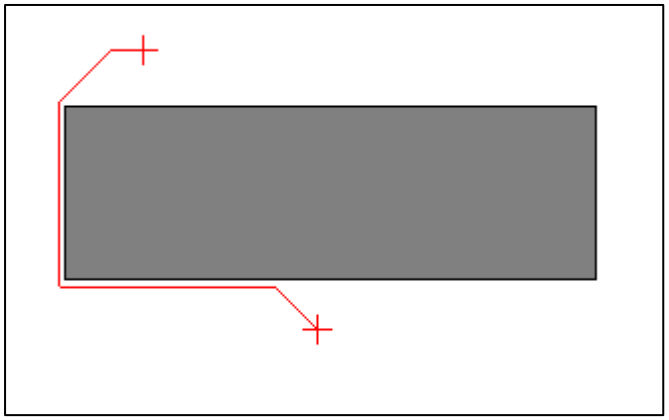
Vytváření cesty končí splněním jedné ze dvou podmínek:

1. Dosažením cíle
2. Maximálním počtem pohybů.

V programu se objevil ten problém, že pokud algoritmus narazí na lokální minimum (obr. 19), tak cesta uvázne v lokálním minimu, přičemž algoritmus neustále pracuje (dokud nenastane maximální počet pohybů). Jedno z řešení, které je implementováno v programu je, že daný potenciál lokálního minima zvýšíme o konstantu a ještě zvýšíme potenciál v okolí bodu podle toho, jak je postaveno lokání minimum vůči startu a cíli. Důsledek řešení je vidět na obr. 20.



Obr. 19 Nalezení lokálního minima.



Obr. 20 Řešení problému s lokálním minimem

4.2 Původní přístup PSO

Algoritmus optimalizace hejnem částic v diskrétním pracovním prostoru, kdy aktualizace polohy provádím pomocí vzorce (23) a aktualizaci rychlosti vztahem (24), kdy $w(t)$ je konstanta nebo se určí výpočtem pomocí vzorce (25). Pro daný PSO jsem aplikoval fitness funkce, která má tvar:

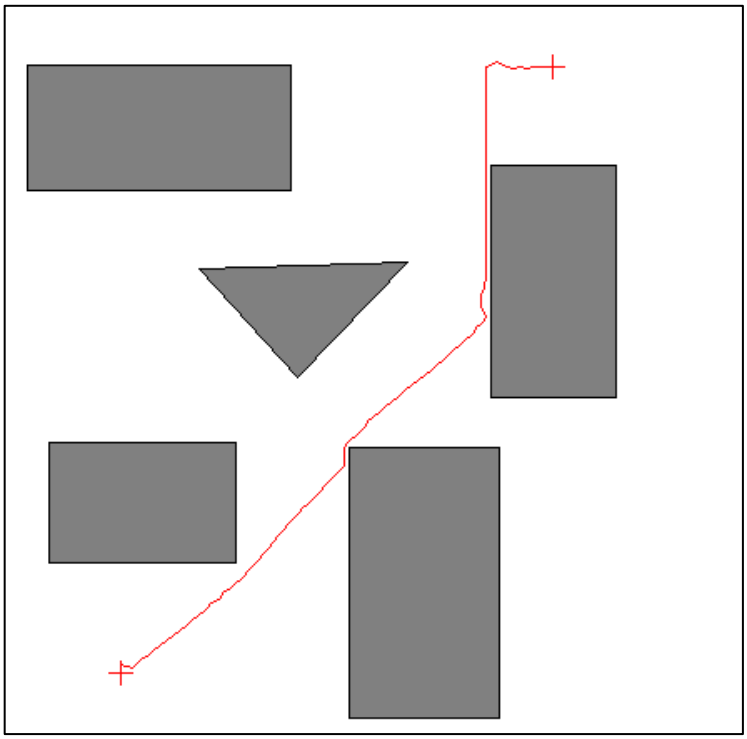
$$f(x,y) = \sqrt{(x_p - x_g)^2 + (y_p - y_g)^2} + s \qquad (50)$$

kde x_p a y_p jsou souřadnice p částice $p\epsilon\{1,\dots,P\}$, kdy P je velikost roje, x_g a y_g jsou souřadnice cílového bodu a s nabývá hodnot nula nebo kladné nekonečno. Zda proměnná s bude kladné nekonečno nebo nula rozhone jak daleko od překážky je p částice. Například pokud je bod (x, y), který je na obr. 17 vyznačen v nedovolené blízkosti překážky (na obr. 17 vyznačeno červenou barvou), tak s nabývá hodnoty kladné nekonečno. Pokud p částice nebude v nedovolené blízkosti překážky, tak hodnota proměnné s je nula.

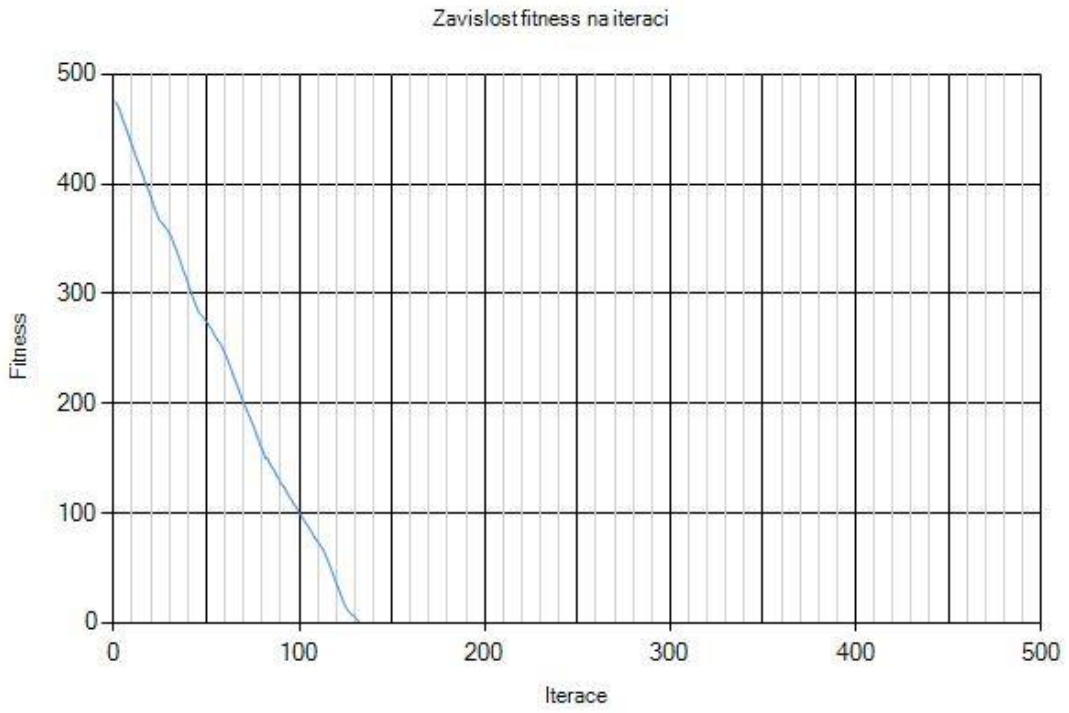
Algoritmu funguje, jak je popsáno v odstavci 3.4.3., kdy hledá minimu fitness funkce (50).

Algoritmus ukončuji splněním jedné ze dvou podmínek, a to vyčerpáním maximálního počtu iterací nebo dosažením cíle.

Problém u tohoto algoritmu je, že se po aktualizaci polohy částice může ocitnout v překážce. V programu jsem vybral řešení, že pokud částice po aktualizaci polohy se ocitne v překážce, tak ji vrátím zpět na původní polohu a vygeneruji nový vektor rychlosti.



Obr. 21 Ukázka funkčnosti PSO.



Obr. 22 Průběh fitness funkce pro ukázkový příklad PSO.

4.3 Prvá modifikace PSO

Následný algoritmus optimalizace hejnem částic, kdy pro aktualizaci polohy a rychlosti opět používám vztahy (23) a (24), a $w(t)$ je konstanta nebo se určí výpočtem pomocí vzorce (25). Ale pro daný PSO algoritmus využívám jinou fitness funkci. Tato fitness funkce má tvar:

$$f(p_i) = w_1 \frac{1}{\min_{j=1,..,N} |p_i - o_j|} + w_2 |p_i - p_g| \tag{51}$$

kde $|p_i - o_j|$ je vzdálenost mezi částicí p_i a překážkou o_j, kterou lze definovat takto:

$$|p_i - o_j| = \sqrt{(x_{pi} - x_{oj})^2 + (y_{pi} - y_{oj})^2} \tag{52}$$

$|p_i - p_g|$ je vzdálenost částice p_i od cíle p_g definovaná vztahem:

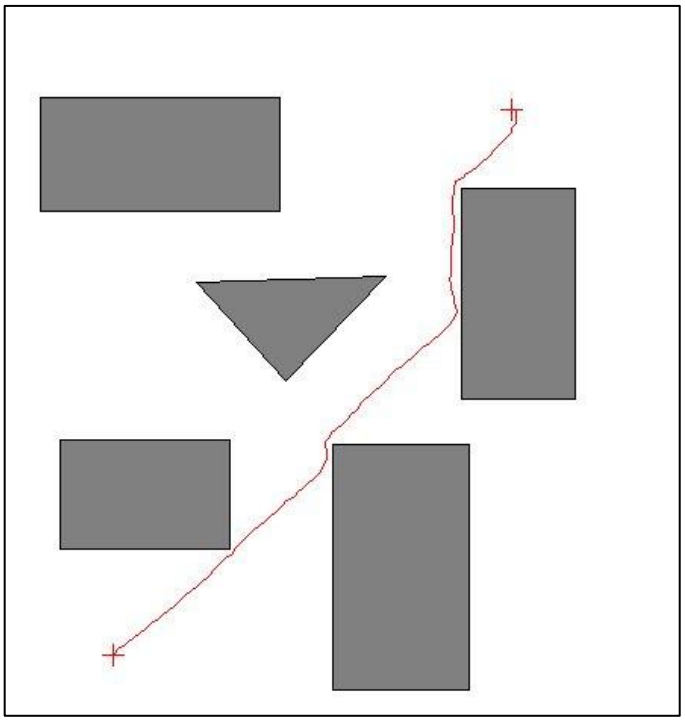
$$|p_i - o_j| = \sqrt{(x_{pi} - x_{oj})^2 + (y_{pi} - y_{oj})^2} \tag{53}$$

a w_1 a w_2 jsou koeficienty.

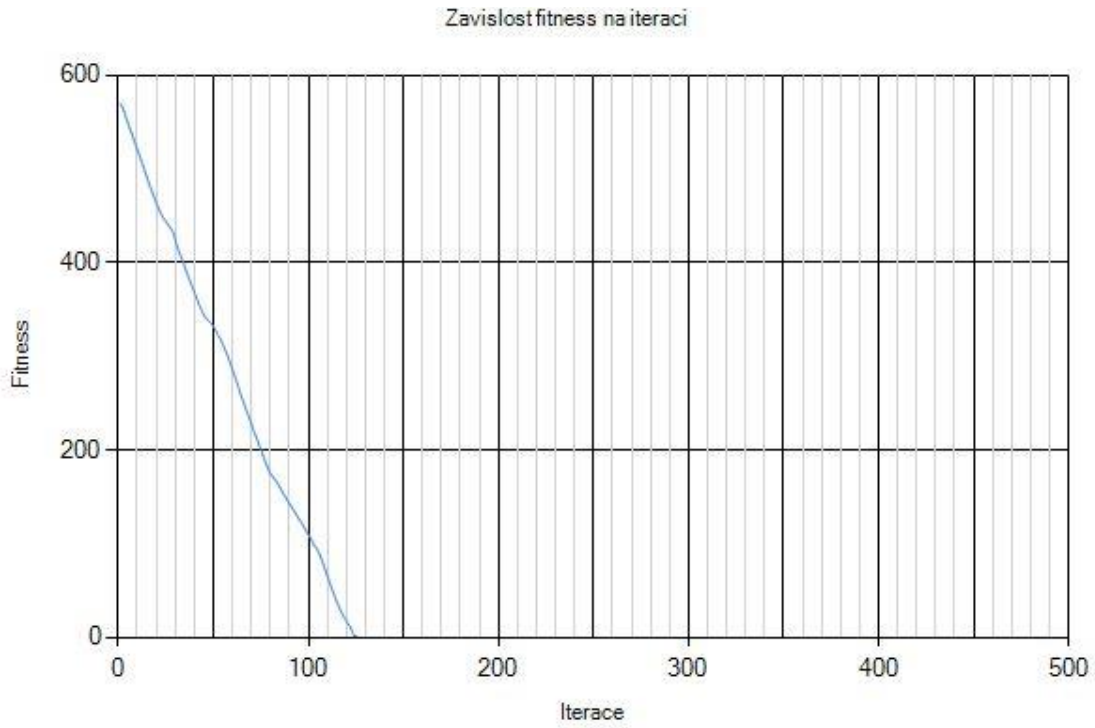
Algoritmu funguje, jak je popsáno v odstavci 3.4.3., kdy hledá minimu fitness funkce (51).

Algoritmus ukončuji splněním jedné ze dvou podmínek, a to vyčerpáním maximálního počtu iterací nebo dosažením cíle.

Pokud se po aktualizaci polohy ocitne částice v překážce, tak tento problém je řešen stejně jako v předešlém algoritmu.



Obr. 23 Ukázka funkčnosti PSO1.



Obr. 24 Průběh fitness funkce pro ukázkový příklad PSO.

4.4 Druhá modifikace PSO

Optimalizace hejnem částic (v programu označena jako PSO2), kdy pro aktualizaci polohy a rychlosti opět používám vztahy (23) a (24), kde $w(t)$ je konstanta nebo se určí výpočtem pomocí vzorce (25). Pro daný PSO algoritmus využívám fitness funkci, která byla původně určena pro potenciálové pole. Vztah pro fitness funkci v 2D prostoru upravený na hledání minima vypadá takto:

$$F(q) = F_{att}(q) + F_{rep}(q) \tag{54}$$

kde $F_{att}(q)$ je definovaná jako atraktivní síla. Pro 2D pracovní prostor rozdělíme $F_{att}(q)$ na *x* a *y* složku. V tom případě $F_{att}(q)$ má podobu:

$$\begin{aligned} F_{att-x}(q) &= k\left|x - x_g\right| \\ F_{att-y}(q) &= k\left|y - y_g\right| \end{aligned} \tag{55}$$

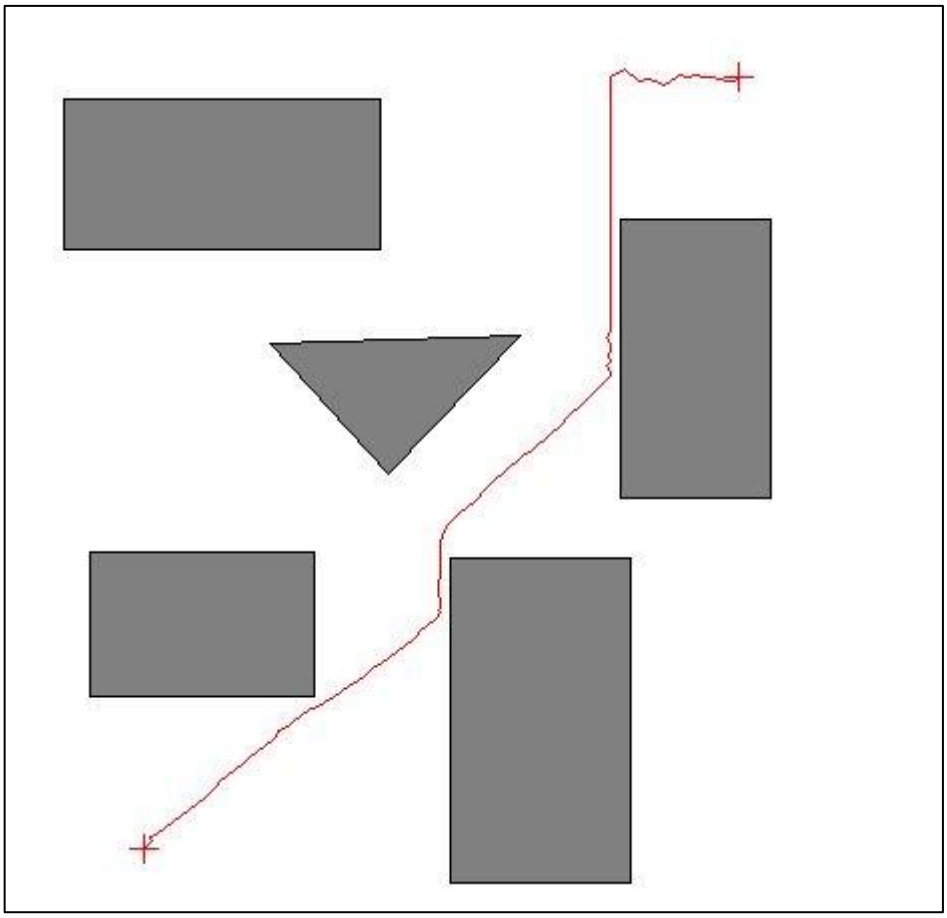
kdy *x* a *y* jsou souřadnice polohy dané částice a x_g a y_g jsou souřadnice polohy cíle, *k* je konstanta.
$F_{rep}(q)$ je odporová síla. V 2D pracovním prostoru ji rozdělíme na *x* a *y* složku:

$$\begin{aligned} F_{rep-x} &= \begin{cases} 0 & \rho(q) \geq \rho_0 \\ \eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\left(\frac{1}{\rho^2(q)}\right)\frac{|x-x_c|}{\|q-q_c\|} & \rho(q) < \rho_0 \end{cases} \\ F_{rep-y} &= \begin{cases} 0 & \rho(q) \geq \rho_0 \\ \eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\left(\frac{1}{\rho^2(q)}\right)\frac{|y-y_c|}{\|q-q_c\|} & \rho(q) < \rho_0 \end{cases} \end{aligned} \tag{56}$$

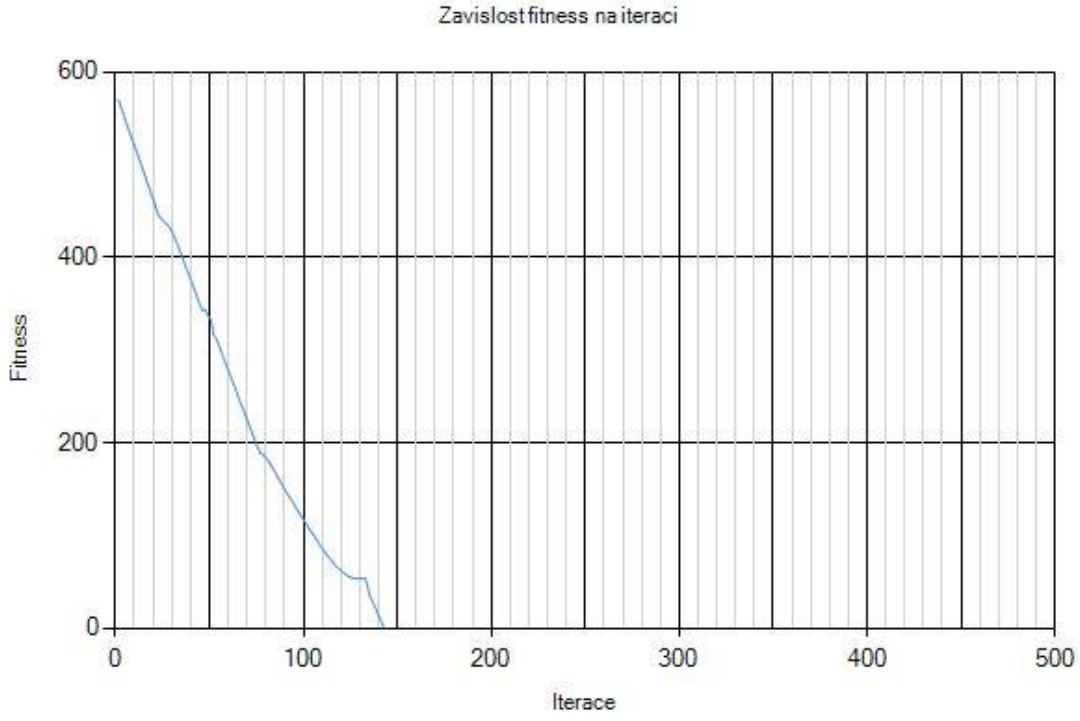
kde ρ_0 je konstanta, která určuje nejmenší vzdálenost od překážky, $\|q - q_c\|$ je nejkratší vzdálenost částice od překážky, η je konstanta, $\rho(q)$ je vzdálenost částice od překážky. Poté vektory sečteme a dosáhneme výslednou sílu *F(q)*. V tomto případě hledáme minimum síly *F(q)*.

Algoritmu funguje, jak je popsáno v odstavci 3.4.3., kdy hledá minimum fitness funkce (54).
Algoritmus ukončuji splněním jedné ze dvou podmínek, a to vyčerpáním maximálního počtu iterací nebo dosažením cíle.

Pokud se po aktualizaci polohy ocitne částice v překážce, je tento problém řešen stejně jako v předešlém algoritmu.



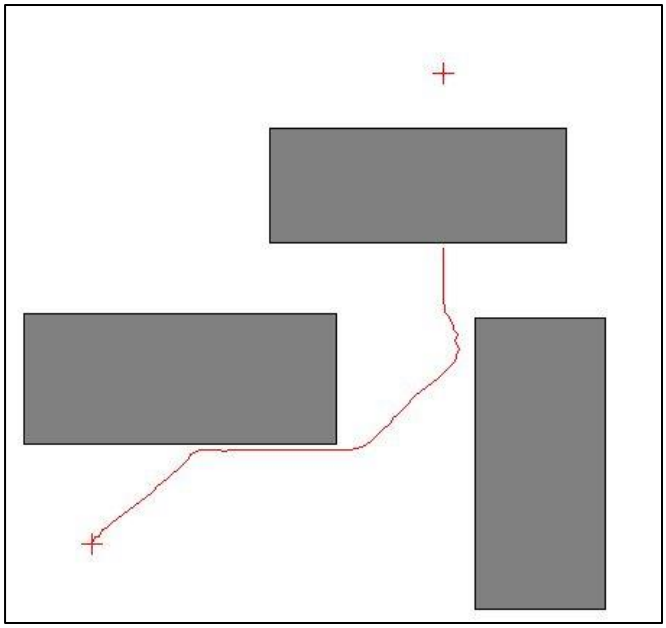
Obr. 25 Ukázka funkčnosti PSO2.



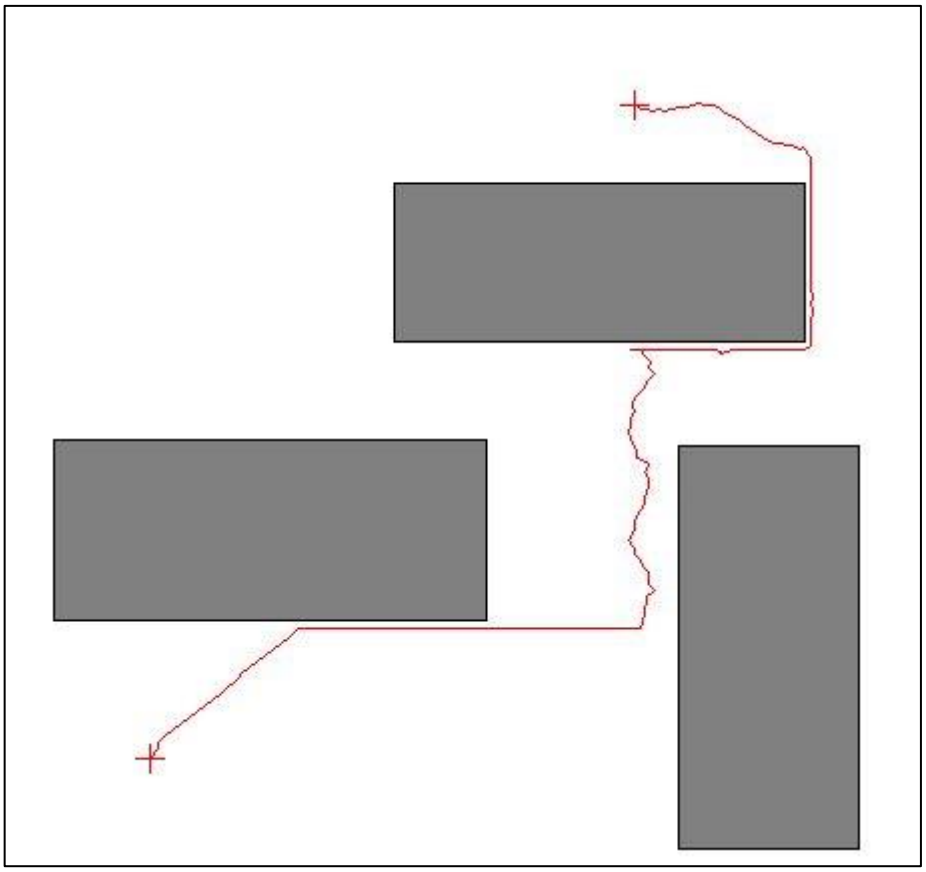
Obr. 26 Průběh fitness funkce pro ukázkový příklad PSO.

4.5 Problémy s lokálním minimem

U všech optimalizací hejnem částic, které jsem použil v programu, se objevil problém s lokálním minimem (obr. 27), které je dáno právě fitness funkcemi PSO. Pro řešení takového problému jsem se nechal inspirovat lidským chováním. Pokud se narazí na nějaký problém, tak se stanoví subcíl, což znamená vyřešit daný problém a poté dosáhnout požadovaného cíle. To samé jsem použil u daného problému, kdy pokud algoritmus narazí na lokální minimum, tak se podle postavení cíle, startu a p_g určí subcíl. K tomuto subcíli si zapamatuji vzdálenost cíle a p_g, kterou zmenším o konstantu. Po stanovení subcíle znovu vygeneruji populaci roje v místě, kde se lokální minimum nacházelo. Pokud roj dosáhne vzdálenosti cíle a p_g, kterou jsme zmenšili o konstantu, odebereme subcíl, vygenerujeme znovu populaci kolem tohoto místa a necháme algoritmus dosáhnout cíle.



Obr. 27 Lokální minimum u optimalizace hejnem částic.

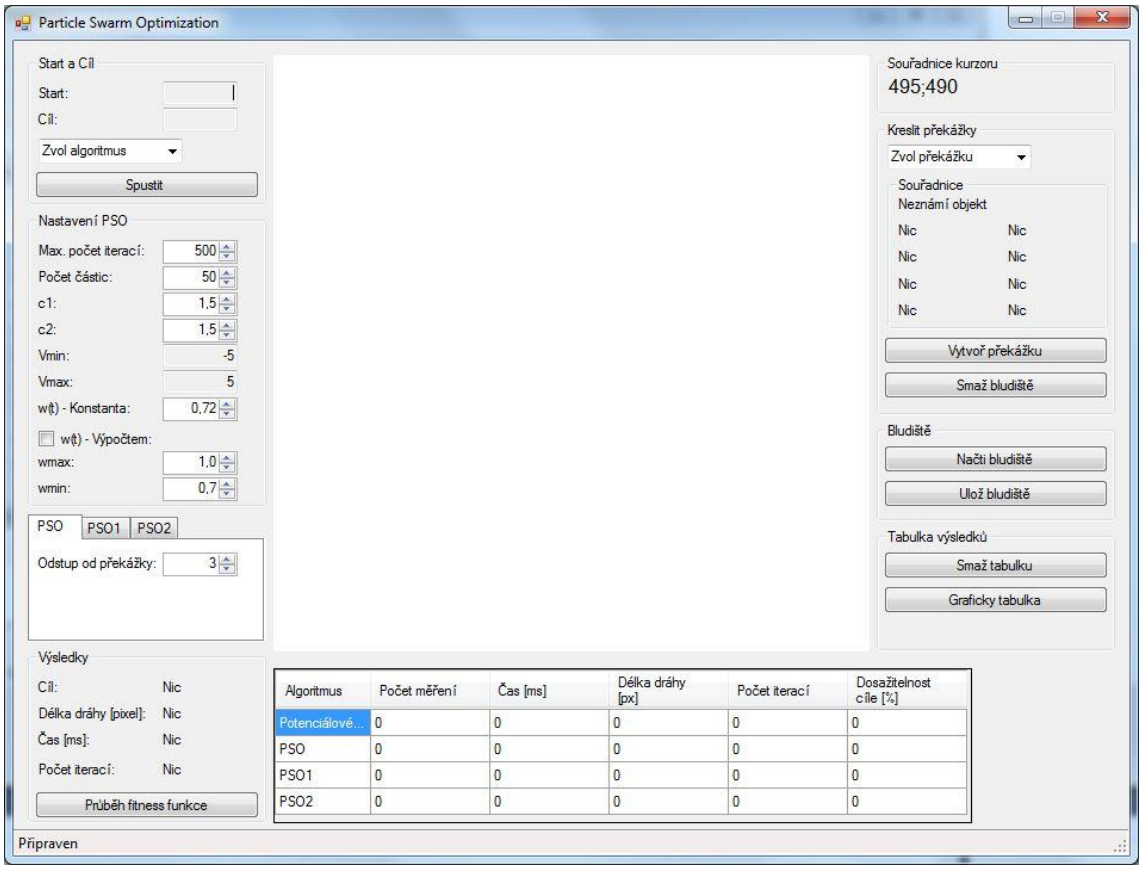


Obr. 28 Řešení lokálního minima u optimalizace hejnem částic.

Obr. 29 Průběh fitness funkce pro řešení lokálního minima u PSO.

5. Popis programu

Program byl vytvářen v programovém prostředí Microsoft Visual C#. V programu bylo využito objektového programování. Cílem tohoto programu je plánování cesty mobilního robota za požití zvolených algoritmů v programu, což jsou jmenovitě potenciálové pole a optimalizace hejnem částic a jejich modifikace. Rovněž bylo cílem, aby program byl uživatelsky příjemný a jednoduchý na ovládání.



Obr. 30 Uživatelské prostředí programu.

5.1 Uživatelské prostředí

Uživatelské prostředí je rozděleno do čtyř hlavních částí.

První částí je pracovní plocha, která má rozměry 500x500 bodů. Nachází se ve středu uživatelského prostředí.

Další část se nachází v levé části uživatelského prostředí. Tato část se rozkládá do čtyř sekcí. Horní sekce zobrazuje zvolené souřadnice cíle a startu. Volíme zde algoritmus, který chceme použít pro plánování cesty mobilního robota. V této sekci se nachází i tlačítko pro spuštění plánování cesty. Pod touto sekcí se nachází sekce, ve které zadáváme parametry pro optimalizaci hejnem částic (maximální počet iterací, velikost hejna koeficienty, atd.) Pod touto nastavovací sekcí parametrů optimalizace hejnem částic nalezneme sekci, která je zaměřena na nastavování fitness funkcí, které jsou použity v algoritmech PSO. Poslední sekcí nazvanou výsledky se zobrazují naměřené hodnoty použitého algoritmu.

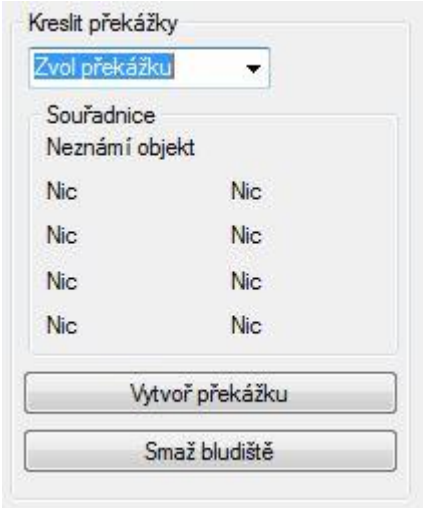
Třetí část se nachází ve spodní části uživatelského prostředí. V této části je pouze tabulka, která slouží k porovnání jednotlivých algoritmů.

Poslední část se nachází v pravé části uživatelského prostředí. Rovněž tato část se rozkládá do čtyř sekcí, jako druhá část. Horní sekce má pouze informativní účel, a to informovat uživatele o poloze kurzoru na pracovní ploše. Pod touto sekcí se nachází sekce pro tvorbu překážek. Volíme zde druh překážky, kterou chceme na pracovní ploše. Pod volbou překážky se zobrazují informativně souřadnice vrcholů zvolených obrazců, které zadáváme kurzorem na pracovní ploše. Rovněž nesmí chybět tlačítka pro vytvoření překážky a smazání překážek. Další sekce je zaměřena pro ukládání již vytvořeného bludiště a jejich nahrávání. Poslední sekce je zaměřena na ovládání tabulky, která se nachází pod pracovním prostorem. A to jmenovitě smazání tabulky a její grafický přehled.

Pod uživatelským prostředím se nachází informační lišta, která má za úkol informovat uživatele o tom, co se s programem děje.

5.2 Tvorba překážek

Překážky vytváříme pomocí sekce „Kreslit překážky“ v pravé části uživatelského prostředí, kdy můžeme zvolit mezi překážkou trojúhelníkového tvaru, obdélníkového tvaru a polygonem čtvrtého až osmého stupně. Souřadnice zvolené překážky zadáváme pomocí kurzoru na pracovní ploše. Překážky se nevytváří hned po udání poslední souřadnice vrcholu překážky, ale až po stisknutí tlačítka „Vytvořit překážku“. Celé bludiště můžeme smazat stisknutí tlačítka „Smaž bludiště“.



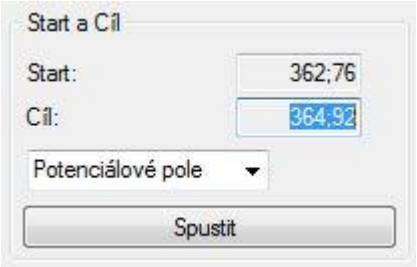
Obr. 31 Sekce Kresli překážky.

5.3 Uložení a nahrání bludiště

Z důvodu toho, aby si uživatel mohl vytvářet příklady bludišť a uchovávat je pro pozdější experimenty, byla vytvořena právě sekce s názvem „Bludiště“. Bludiště jsou ukládána jako bitmapy, takže velikost daného souboru zabírá úložnou paměť v rozsahu dvou až šesti megabajtů. Pokud by se zvolil vhodný archivační program, tak velikost souboru nepřesáhne dva megabajty.

5.4 Start a cíl

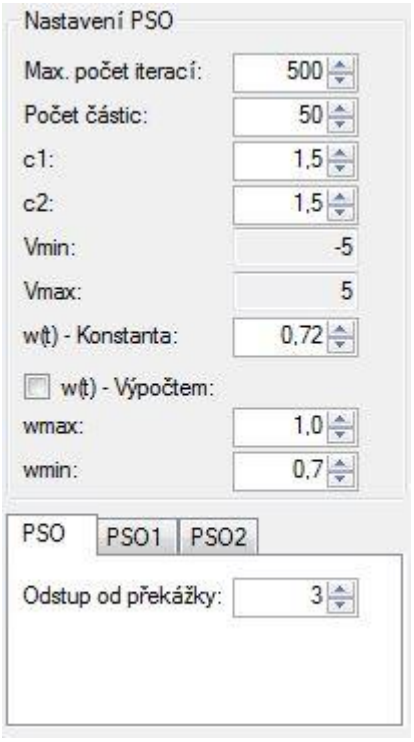
Souřadnice startu a cíle se vkládají pomocí kurzoru na pracovní ploše, kdy pokud chceme zadat pozici startu nebo cíle, tak klikneme kurzorem myši na příslušný *TextBox* a následně volíme pozici v pracovní ploše pomocí kurzoru a potvrzení pravého nebo levého tlačítka myší. V této sekci je rovněž volba algoritmů, kdy volíme mezi metodou potenciálového pole (podkapitola 4.1), PSO (podkapitola 4.2), PSO1 (podkapitola 4.3) a PSO2 (podkapitola 4.4).



Obr. 32 Sekce Start a Cíl.

5.5 Nastavení PSO a nastavení parametrů fitness funkcí

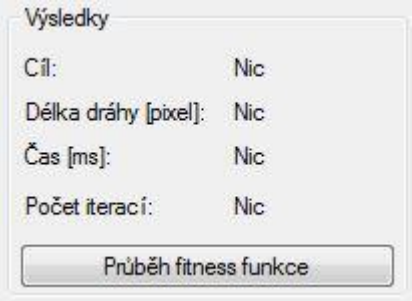
Pro nastavování parametrů algoritmů a fitness funkcí je vytvořena nastavovací část. Na vkládání parametrů algoritmů a fitness funkcí jsou převážně použity tzv. *numericUpDown* nástroje. Je to z důvodu toho, aby uživatel nezadával nesmyslné hodnoty a tak nezapříčinil pád programu. Rovněž jsou zde stanoveny meze nastavení hodnot.



Obr. 33 Nastavení PSO a parametrů fitness funkcí.

5.6 Výsledky, tabulka a práce s tabulkou

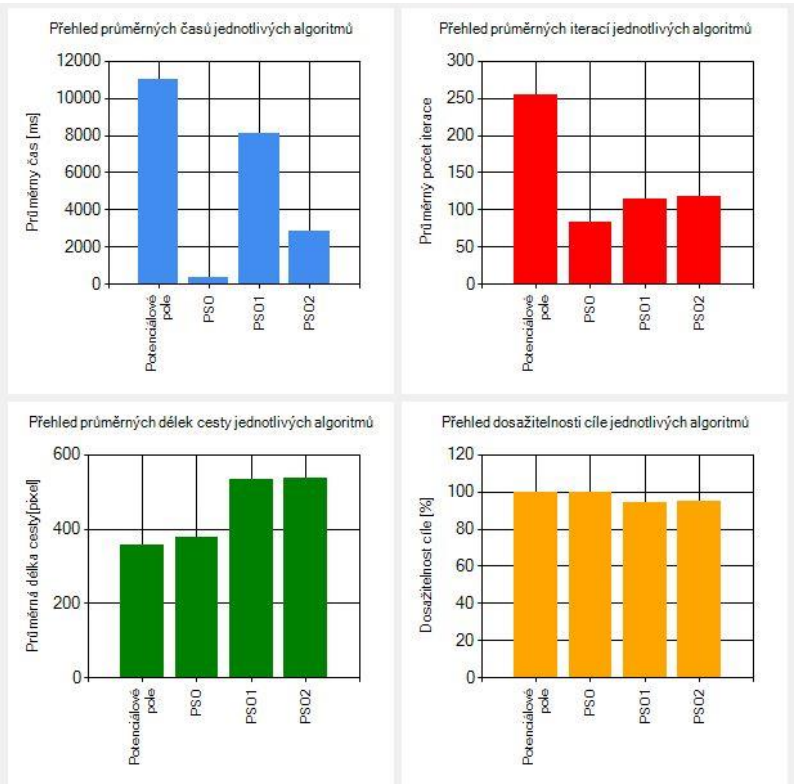
Ve výsledkové sekci se vypisují naměřené hodnoty právě dokončeného algoritmu. Jmenovitě čas běhu programu v milisekundách, počet iterací, délka nalezené dráhy v pixelech a v poslední řadě zda zvolený cíl byl dosažen. V této sekci se nachází i tlačítko pro vykreslení průběh fitness funkce algoritmů PSO. Tento průběh fitness funkce se vykreslí v novém okně.



Obr. 34 Sekce výsledky.

Tabulka slouží k porovnání jednotlivých algoritmů, kdy v tabulce máme počet měření, průměrný čas běhu algoritmu v milisekundách, průměrnou délku dráhy robota v pixelech, průměrný počet iterací a procentuální úspěšnost programu.

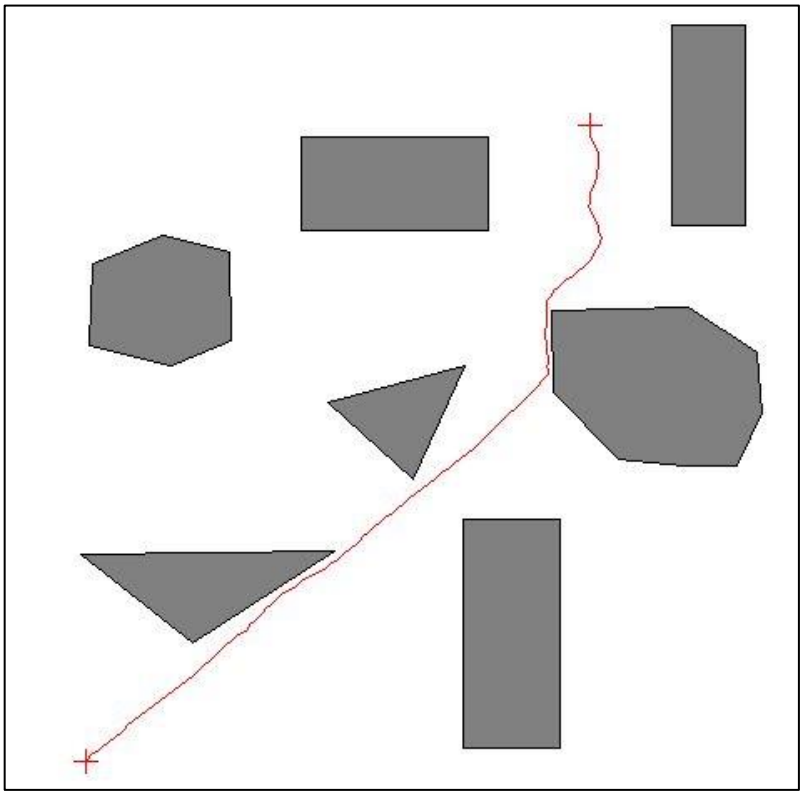
Dále s tabulkou můžeme pracovat tak, že můžeme buď smazat hodnoty v tabulce, nebo tuto tabulku graficky zobrazit. Grafické zobrazení jsem zvolil hlavně z toho důvodu, že grafické zobrazení je pro člověka přirozenější.



Obr. 35 Grafické znázornění tabulky v programu.

5.7 Pracovní plocha

Tato pracovní plocha složí především jako vizualizace nalezené cesty pro mobilního robota. Volíme zde souřadnice startu, cíle a body pro překážky. Robot je v programu zobrazován jako jeden bod (pixel).



Obr. 36 Pracovní plocha programu.

6. Experimenty a simulace

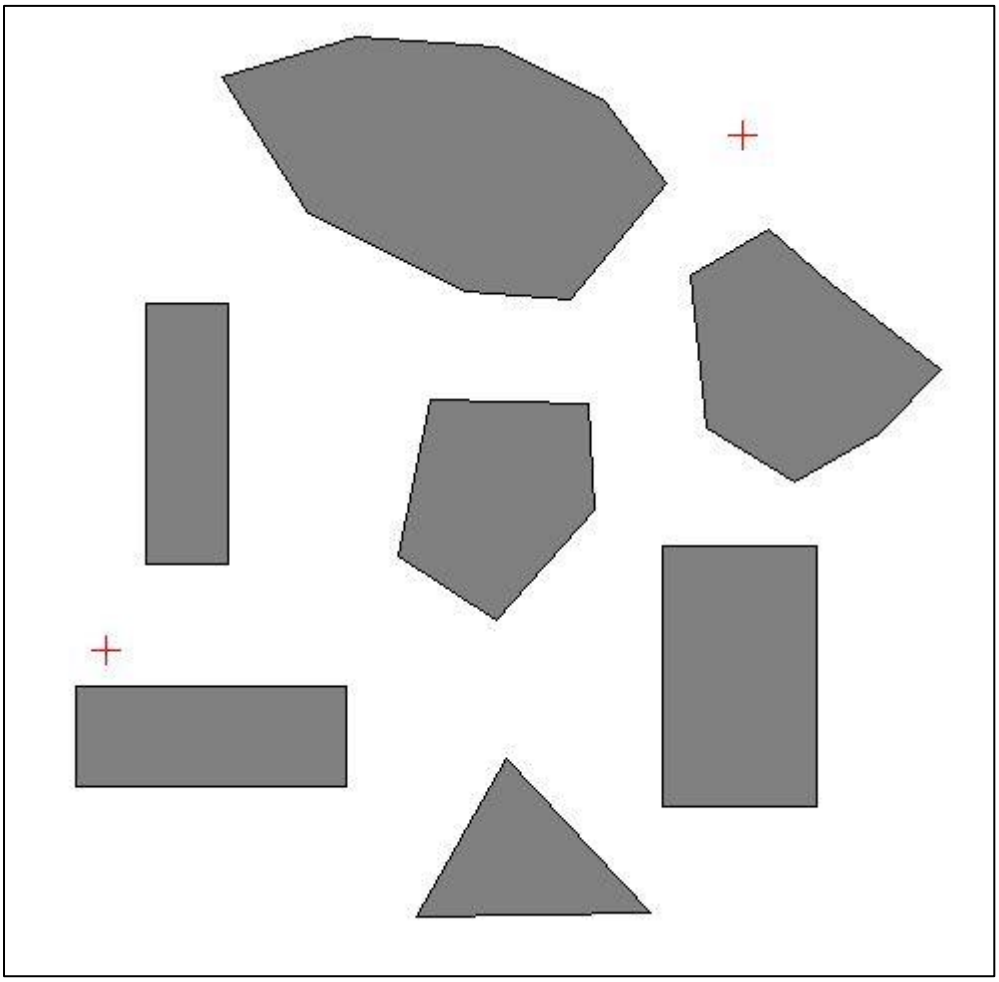
Všechny experimenty byly provedeny v programu, který byl napsán v jazyce C#. Program byl testován na dvoujádrovém procesoru Intel Core i3-2328 s frekvencí 2.20Ghz na každém jádře, s RAM pamětí 4GB a s integrovanou grafickou kartou Intel Graphics 3000.

6.1 První scéna

V prvním scéně jsem nastavil pracovní prostředí, které je zobrazena na obr. 37. Poté jsem nastavil hodnoty parametrů (tab. 1 a tab. 3), jak PSO algoritmů, tak všech fitness funkcí. Nastavené parametry jsou odvozené z literatury a následně doladěny úvodními experimenty. V prvním experimentu jsem nastavil $w(t)$ jako konstantu a v druhém experimentu $w(t)$ byla určena výpočtem pomocí vzorce (25). Všechny algoritmy byly spuštěny 100 krát a byly počítány průměrné časy výpočtů jednotlivých algoritmů, průměrné délky jednotlivých drah, průměrný počet iterací a procentuální úspěšnost dosažení cíle.

Nastavení PSO:	
Maximální počet iterací:	500
Velikost hejna:	50
c_1:	1,5
c_2:	1,5
Setrvačná váha $w(t)$:	0,72
Nastavení fitness funkce pro PSO:	
Odstup od překážky:	3
Nastavení fitness funkce pro PSO1:	
w_1:	20
w_2:	1,2
Nastavení fitness funkce pro PSO2:	
k:	0,3
η:	20
ρ_0:	5

Tab. 1 Nastavení PSO algoritmů a jednotlivých fitness funkcí.

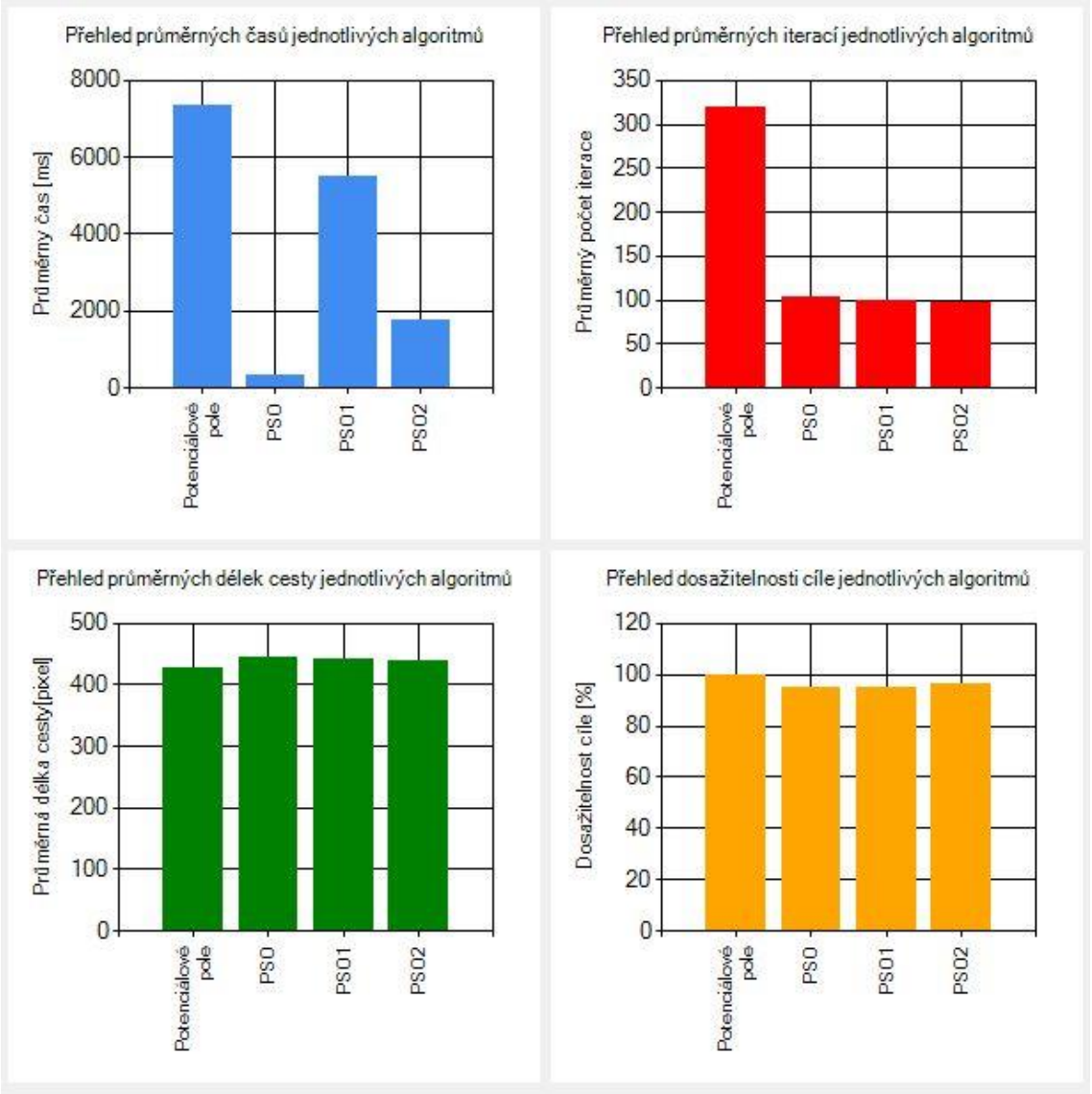


Obr. 37 Zvolený pracovní prostor pro první scénu.

6.1.1 Setrvačná váha jako konstanta

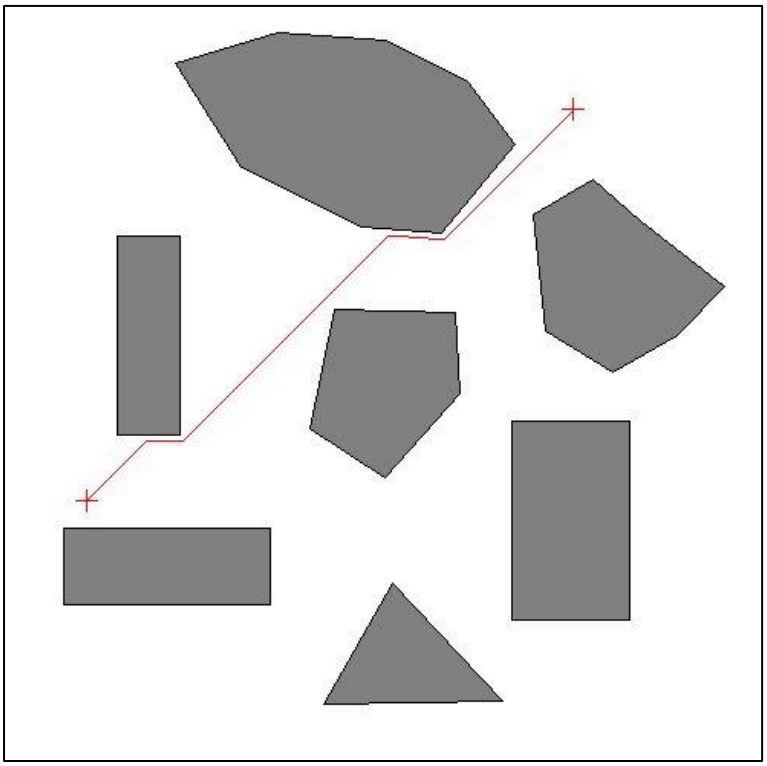
Algoritmus	Počet měření	Průměrný čas [ms]	Průměrná délka dráhy [pixel]	Průměrný počet iterací	Dosažitelnost cíle [%]
Potenciálové pole	100	7330,79	426,28	320,00	100,00
PSO	100	329,27	444,55	103,77	95,00
PSO1	100	5504,34	441,98	98,89	95,00
PSO2	100	1734,58	437,55	97,73	96,00

Tab. 2 Změřené hodnoty pro první scénu, kdy $w(t)$ je konstanta.

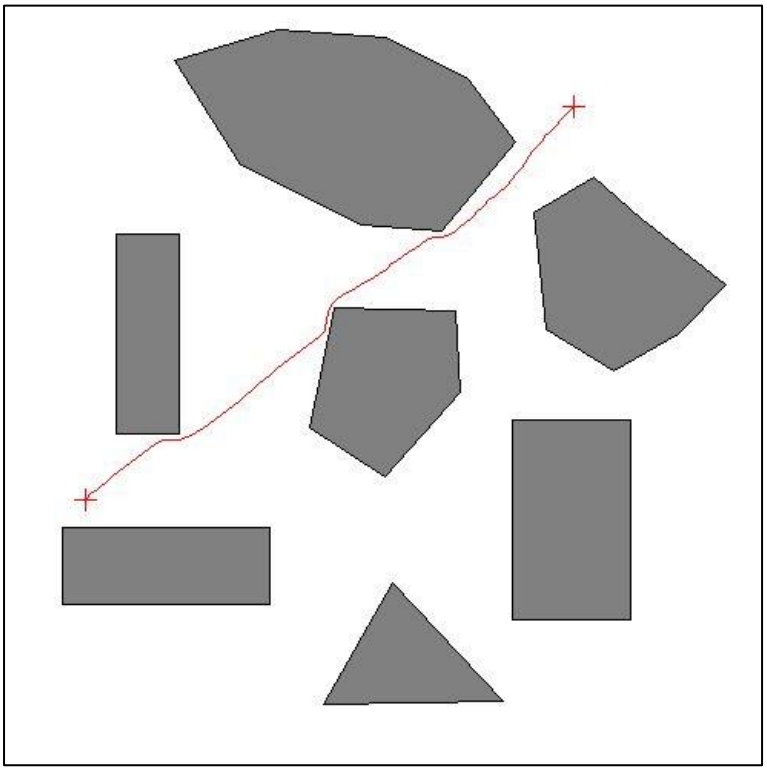


Obr. 38 Graficky znázorněná tab. 2.

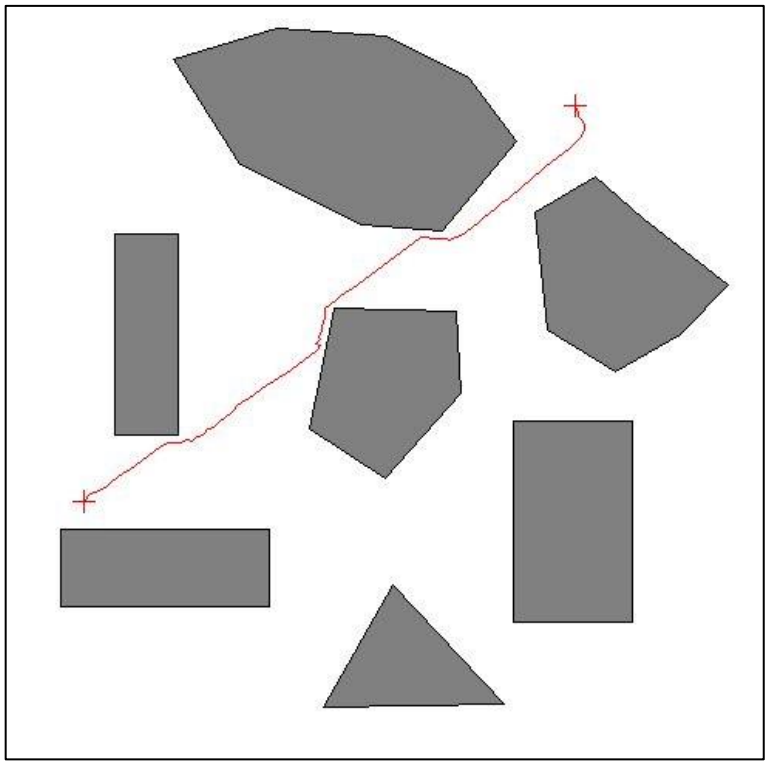
Z naměřených hodnot vyplývá, že co se týče průměrné rychlosti výpočtu, tak nejrychlejší je algoritmus PSO, což se v takovém případě dalo očekávat, jelikož fitness funkce PSO není tak složitá jako v ostatních algoritmech. Zde ale rychlost je na úkor nejdelší průměrné dráhy a nejvyšším počtem iterací v rámci PSO algoritmů. Průměrná délka dráhy je nejmenší u potenciálového pole, ale nijak výrazně. To odpovídá tomu, že potenciálové pole je stavěno, tak aby dosahovalo nejkratší cesty na úkor délky výpočtu. I jeho úspěšnost v experimentu je stoprocentní.



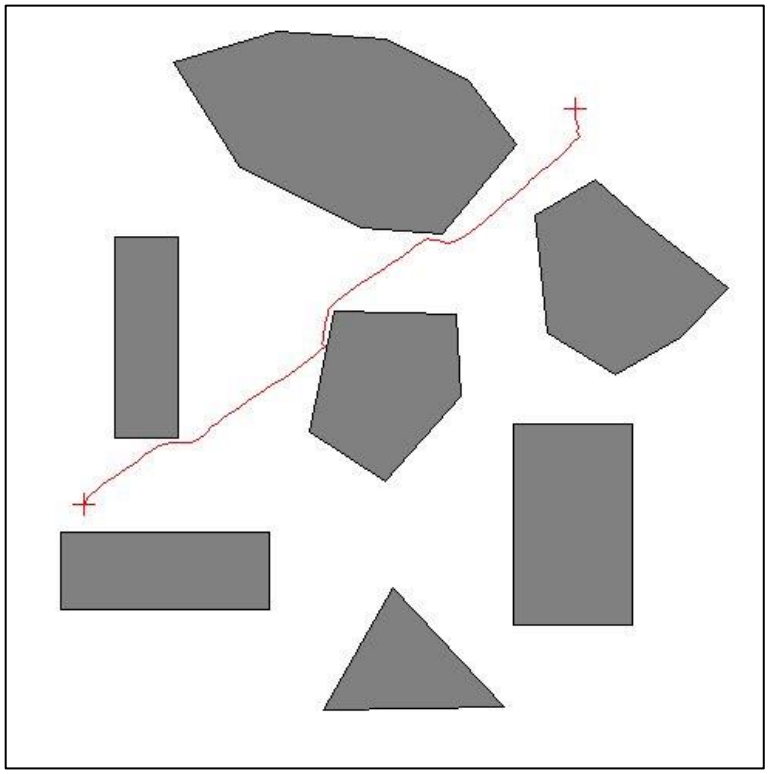
Obr. 39 Řešení pomocí metody potenciálového pole.



Obr. 40 Řešení pomocí PSO.



Obr. 41 Řešení pomocí PSO1.



Obr. 42 Řešení pomocí PSO2.

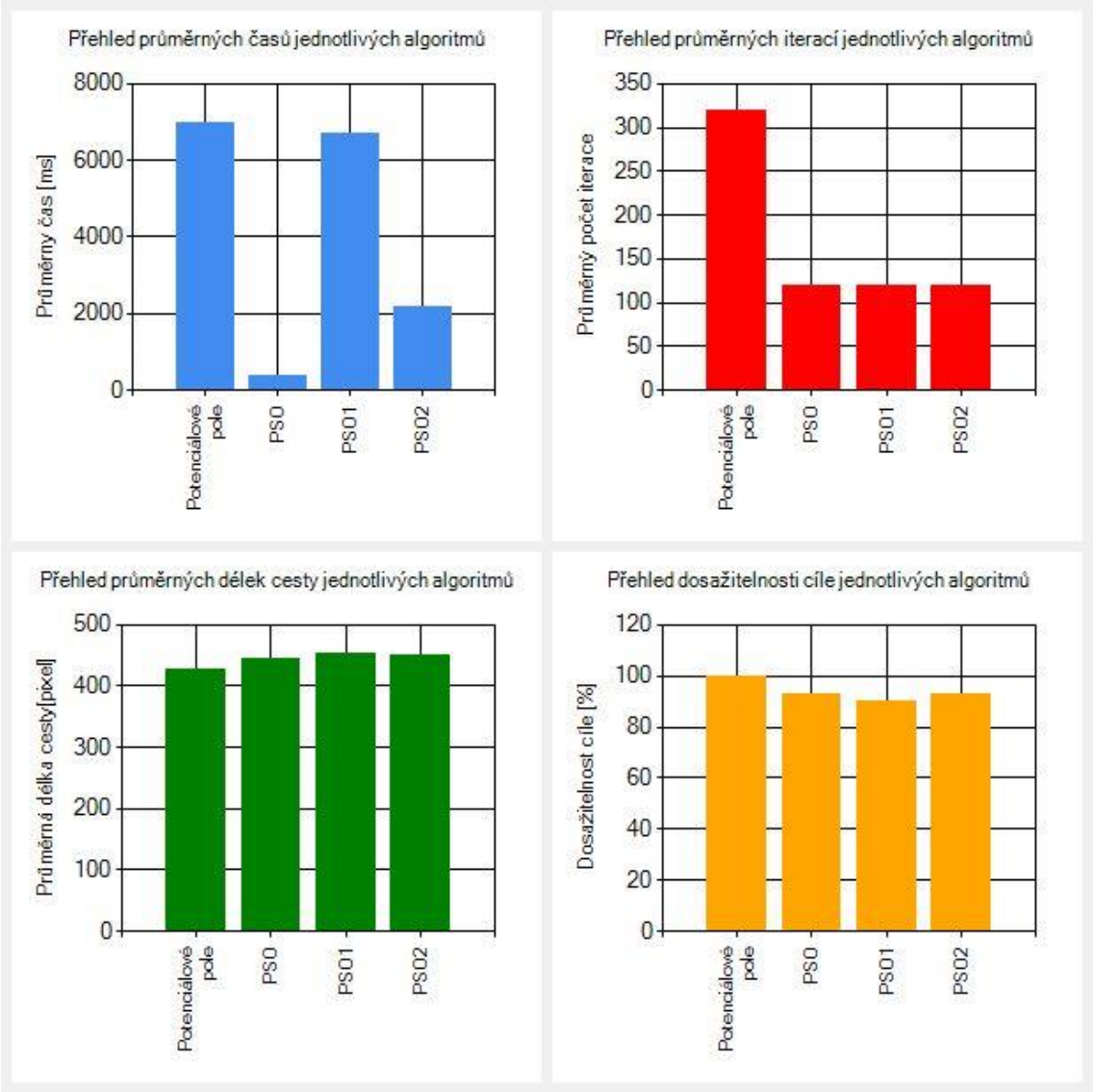
6.1.2 Setrvačná váha určená výpočtem

Nastavení PSO:	
Maximální počet iterací:	500
Velikost hejna:	50
c_1:	1,5
c_2:	1,5
Setrvačná váha $w(t)$:	
w_{max}:	1,0
w_{min}:	0,7
Nastavení fitness funkce pro PSO:	
Odstup od překážky:	3
Nastavení fitness funkce pro PSO1:	
w_1:	20
w_2:	1,2
Nastavení fitness funkce pro PSO2:	
k:	0,3
η:	20
ρ_0:	5

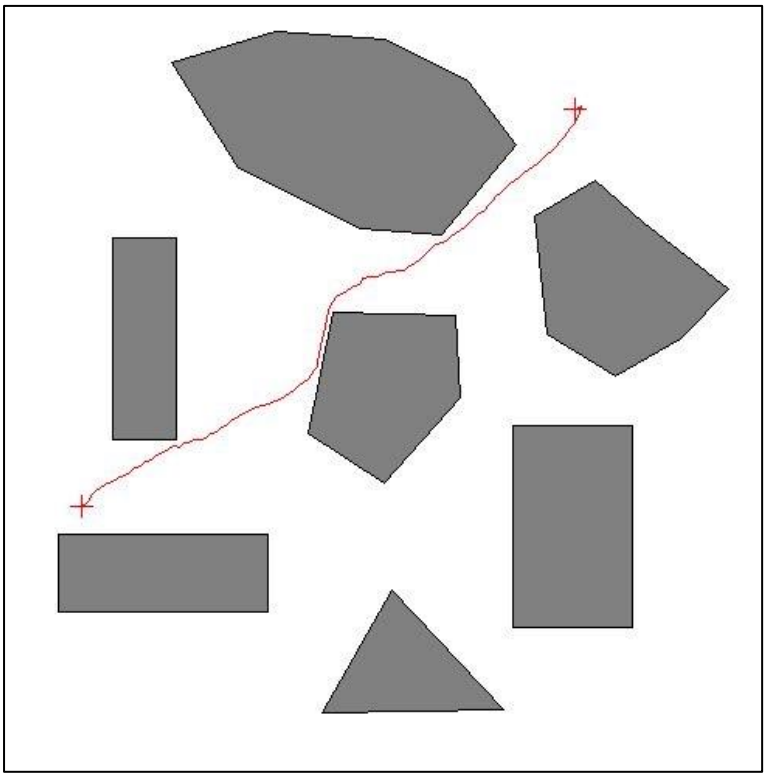
Tab. 3 Nastavení PSO algoritmů a jednotlivých fitness funkcí.

Algoritmus	Počet měření	Průměrný čas [ms]	Průměrná délka dráhy [pixel]	Průměrný počet iterací	Dosažitelnost cíle [%]
Potenciálové pole	100	6952,63	426,28	320,00	100,00
PSO	100	382,37	445,90	119,44	93,00
PSO1	100	6689,62	453,67	120,10	90,00
PSO2	100	2182,29	449,48	118,30	93,00

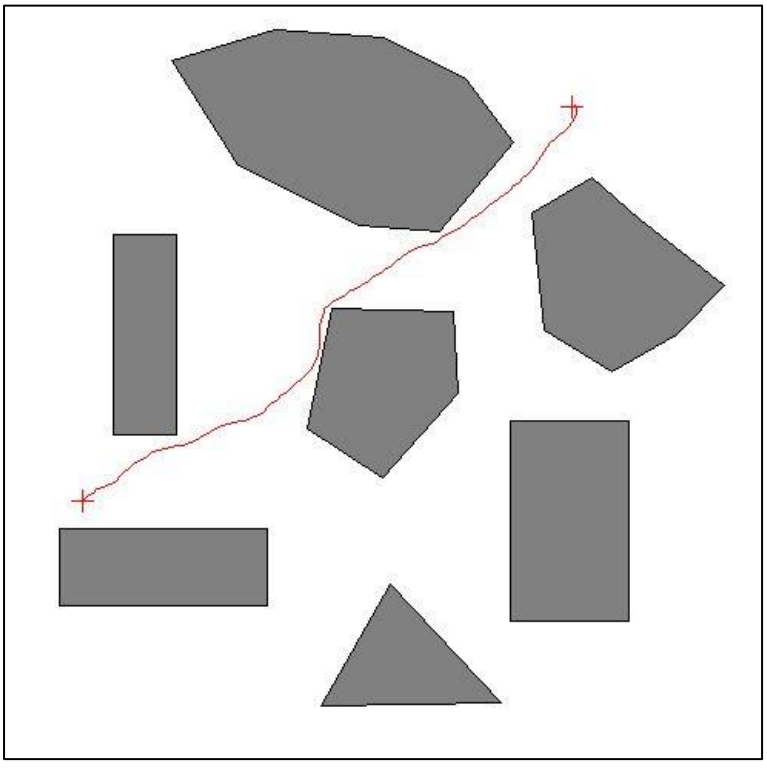
Tab. 4 Změřené hodnoty pro první scénu, kdy $w(t)$ se určí výpočtem.



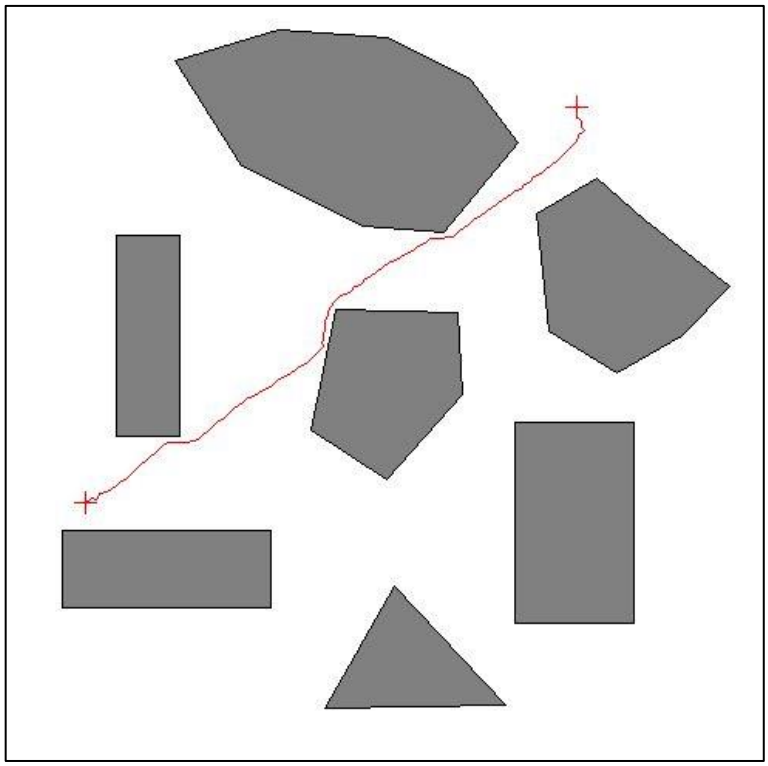
Obr. 43 Graficky znázorněná tab. 4.



Obr. 44 Řešení pomocí PSO.



Obr. 45 Řešení pomocí PSO1.

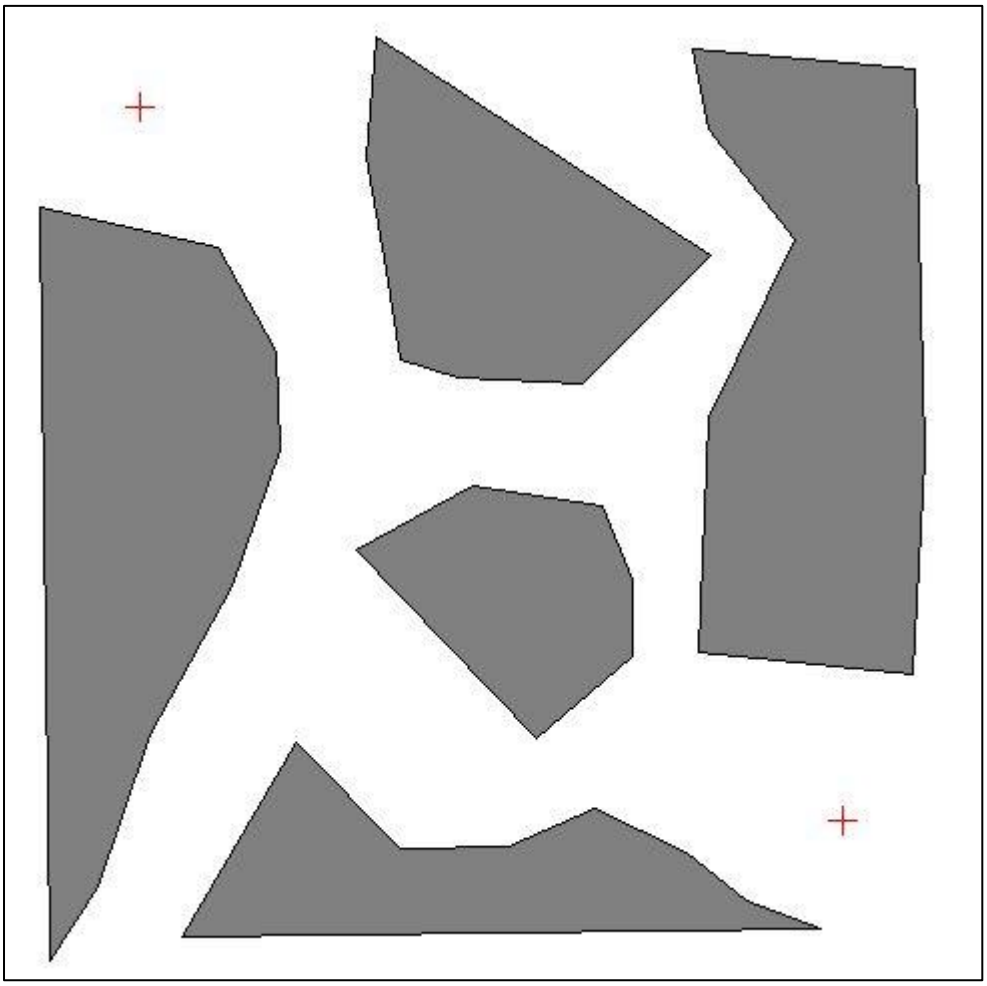


Obr. 46 Řešení pomocí PSO2.

Ve výsledcích tohoto experimentu se negativně podepsala setrvačná váha, která výsledky všech PSO algoritmů zhoršila. Dokonce i jednotlivé PSO algoritmy, které měly měřenou hodnotu jako nejlepší, se zhoršily až do té míry, že se v dané hodnotě staly nejhoršími algoritmy. V některých případech došlo jen k minimální změně, jako například průměrné délce dráhy algoritmu PSO. Dokonce i úspěšnost algoritmů se zhoršila o 2% až 5%. S ohledem na výsledky bych doporučil pro dané pracovní prostředí použít algoritmy PSO se setrvačnou váhou, která bude konstanta. V experimentu měl nejkratší výpočet PSO a PSO2.

6.2 Druhá scéna

V druhé scéně jsem nastavil pracovní prostředí, které je zobrazeno na obr. 47. Parametry byly nastaveny podle tab. 1 a tab. 3, jak PSO algoritmů, tak všech fitness funkcí. V prvním experimentu jsem nastavil $w(t)$ jako konstantu a v druhém experimentu $w(t)$ bylo určeno výpočtem pomocí vzorce (25). U všech experimentů jsem sledoval vypočtené průměrné časy jednotlivých algoritmů, průměrné délky jednotlivých drah, průměrný počet iterací a procentuální úspěšnost dosažení cíle, přičemž algoritmy byly spuštěny 100 krát. Tato scéna sloužila pro porovnání s první scénou, kdy jsem sledoval, jak se změní výsledky, pokud změníme pracovní prostředí.

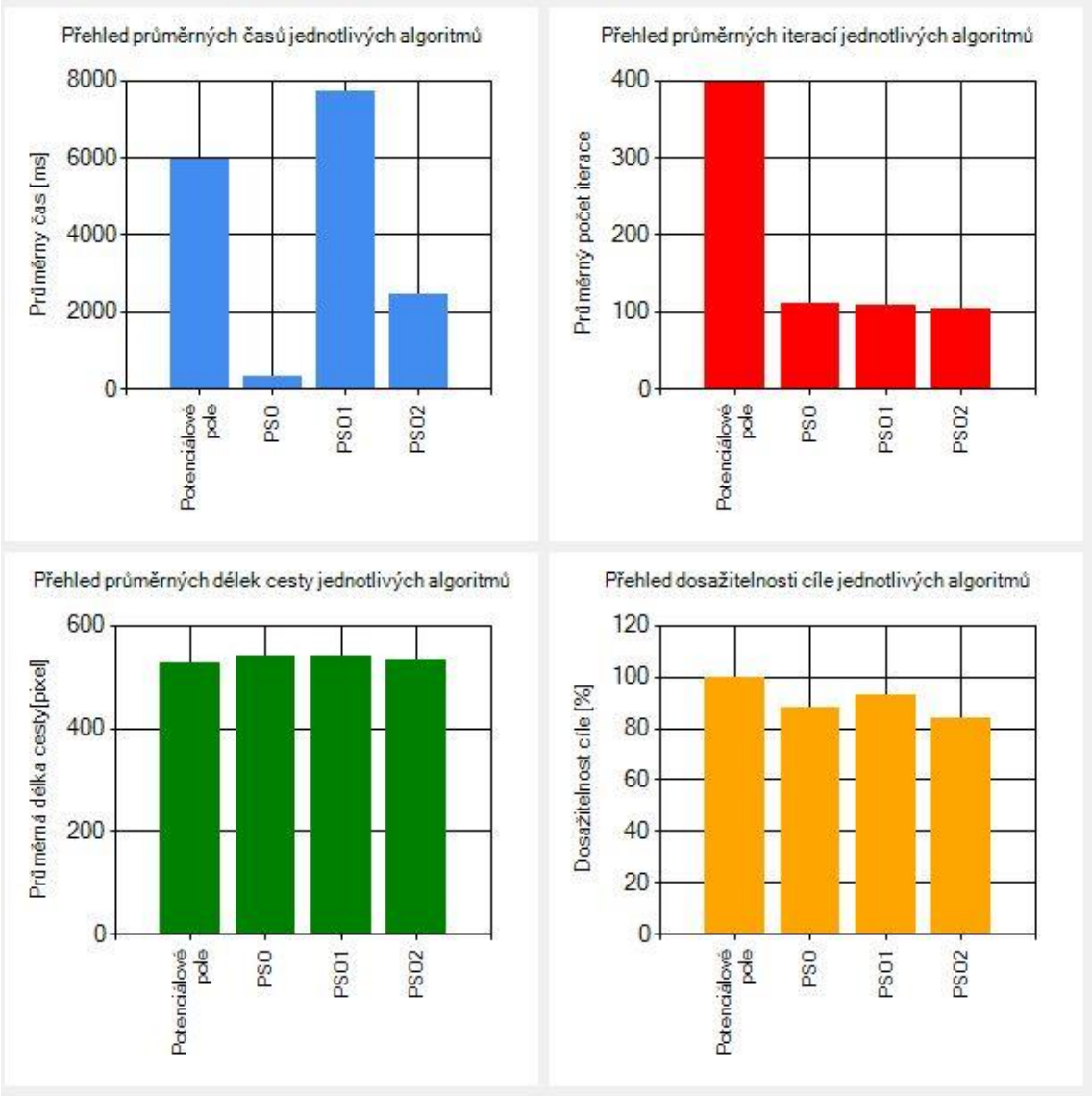


Obr. 47 Zvolený pracovní prostor pro druhou scénu.

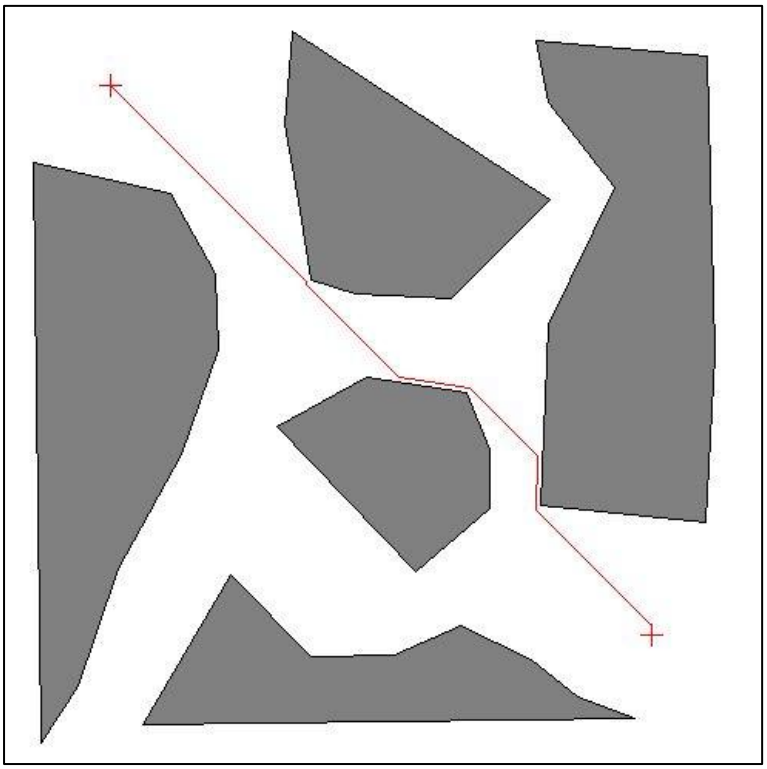
6.2.1 Setrvačná váha jako konstanta

Algoritmus	Počet měření	Průměrný čas [ms]	Průměrná délka dráhy [pixel]	Průměrný počet iterací	Dosažitelnost cíle [%]
Potenciálové pole	100	5959,15	525,06	396,00	100,00
PSO	100	346,97	541,01	111,56	88,00
PSO1	100	7699,37	540,35	108,82	93,00
PSO2	100	2470,81	532,62	104,08	84,00

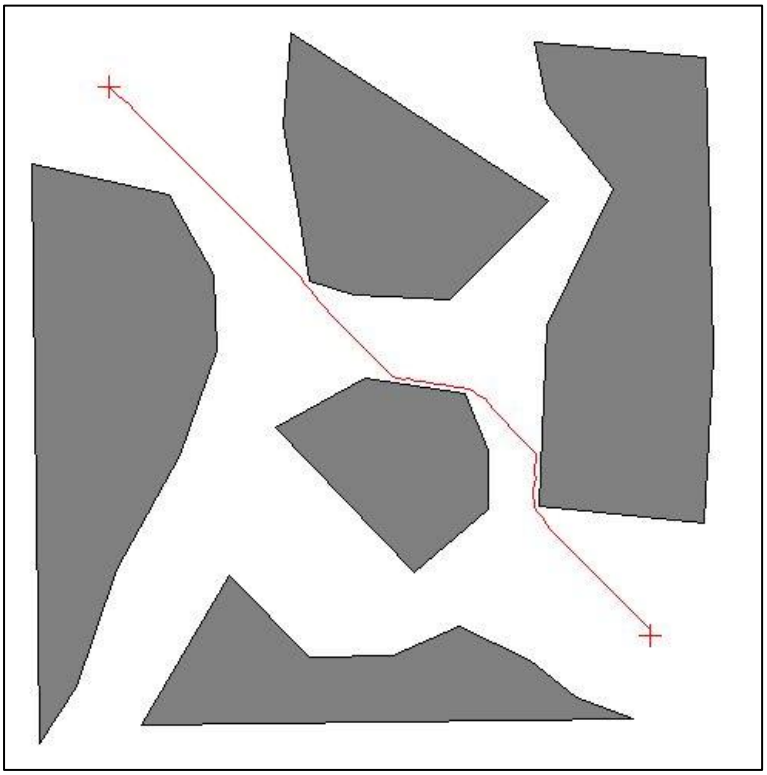
Tab. 5 Změřené hodnoty pro druhou scénu, kdy $w(t)$ je konstanta.



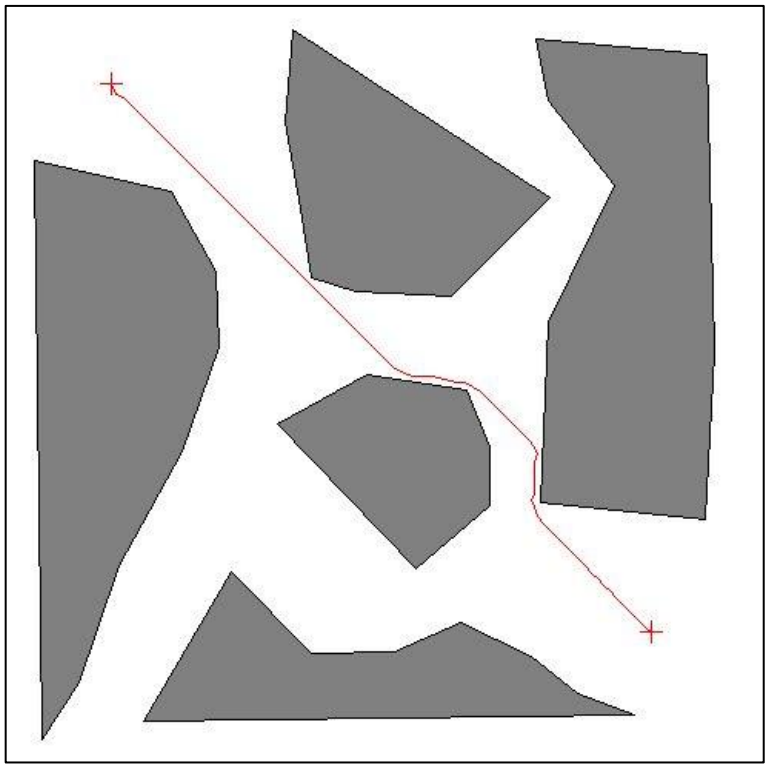
Obr. 48 Graficky znázorněná tab. 5.



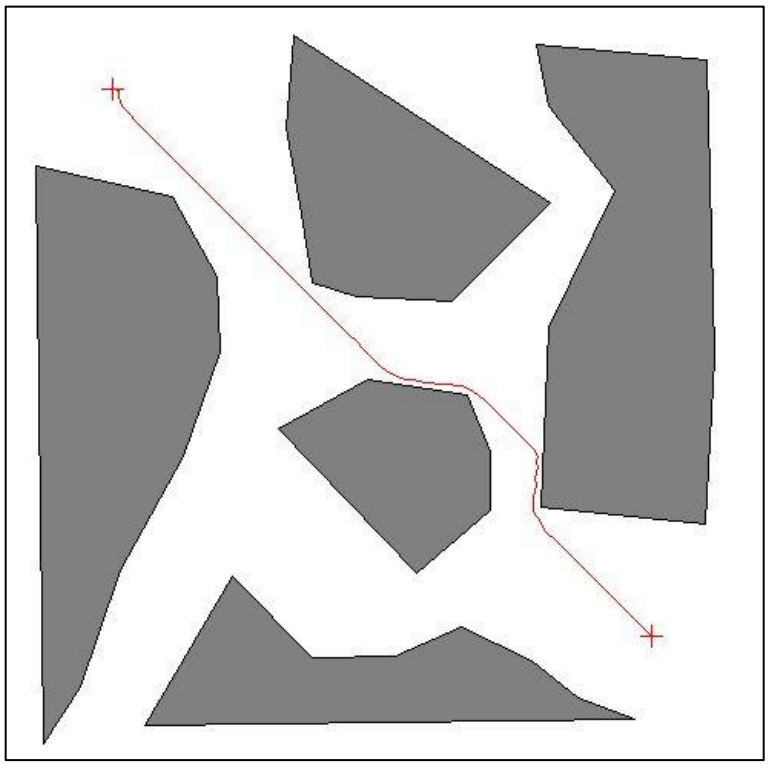
Obr. 49 Řešení pomocí metody potenciálového pole.



Obr. 50 Řešení pomocí PSO.



Obr. 51 Řešení pomocí PSO1.



Obr. 52 Řešení pomocí PSO2.

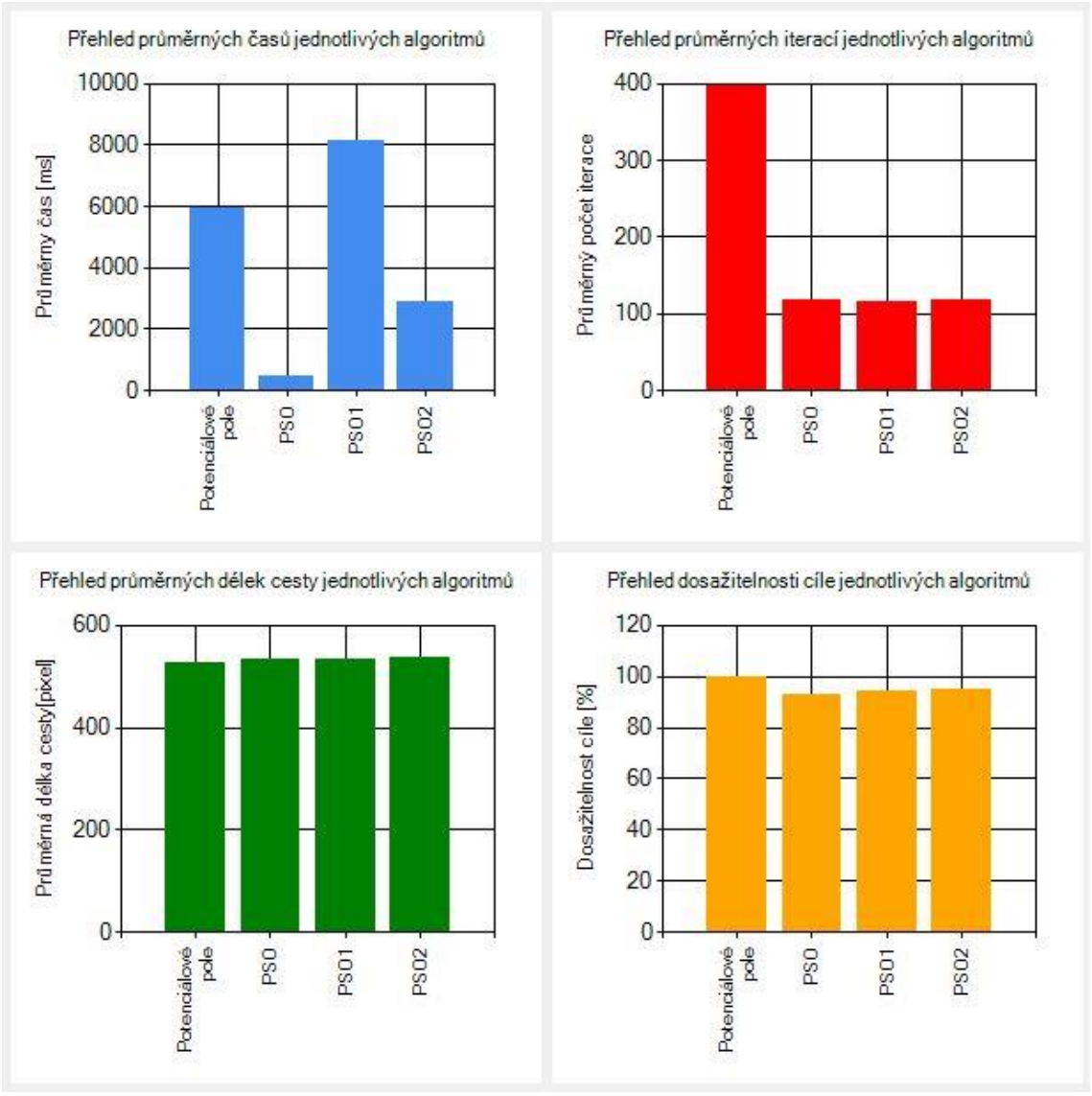
Výsledky tohoto experimentu kopírují výsledky předešlé scény, co se týče nejlepších průměrných časů, průměrné délky dráhy, průměrného počtu iterací. Jediný

rozdíl je v tom, že se zhoršila úspěšnost dosažení cíle u jednotlivých algoritmů PSO. Nejrychlejší výpočty měly PSO a PSO2.

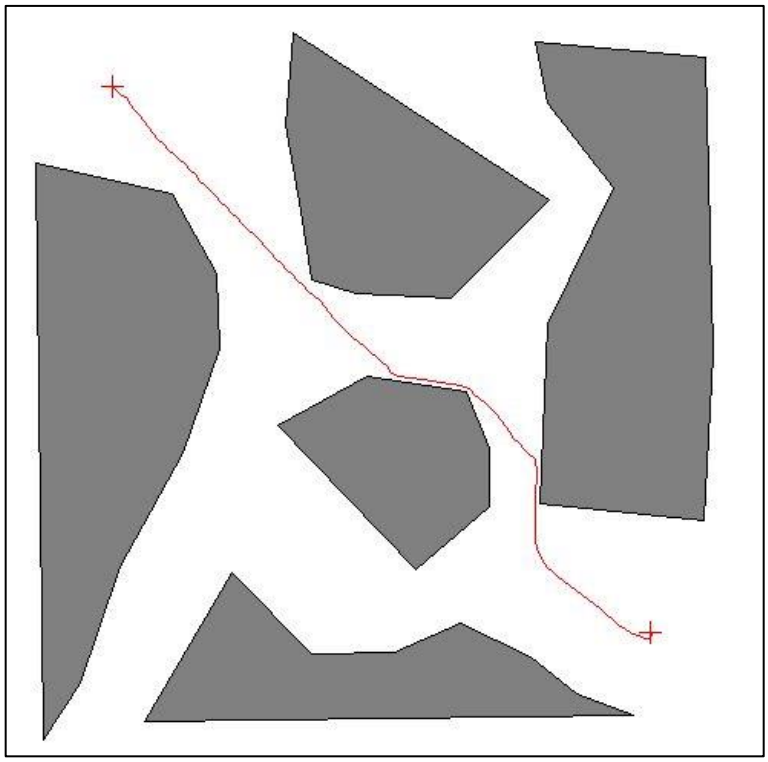
6.2.2 Setrvačná váha určena výpočtem

Algoritmus	Počet měření	Průměrný čas [ms]	Průměrná délka dráhy [pixel]	Průměrný počet iterací	Dosažitelnost cíle [%]
Potenciálové pole	100	5919,73	525,06	396,00	100,00
PSO	100	445,98	533,75	118,48	93,00
PSO1	100	8138,88	533,81	114,81	94,00
PSO2	100	2872,64	536,43	118,32	95,00

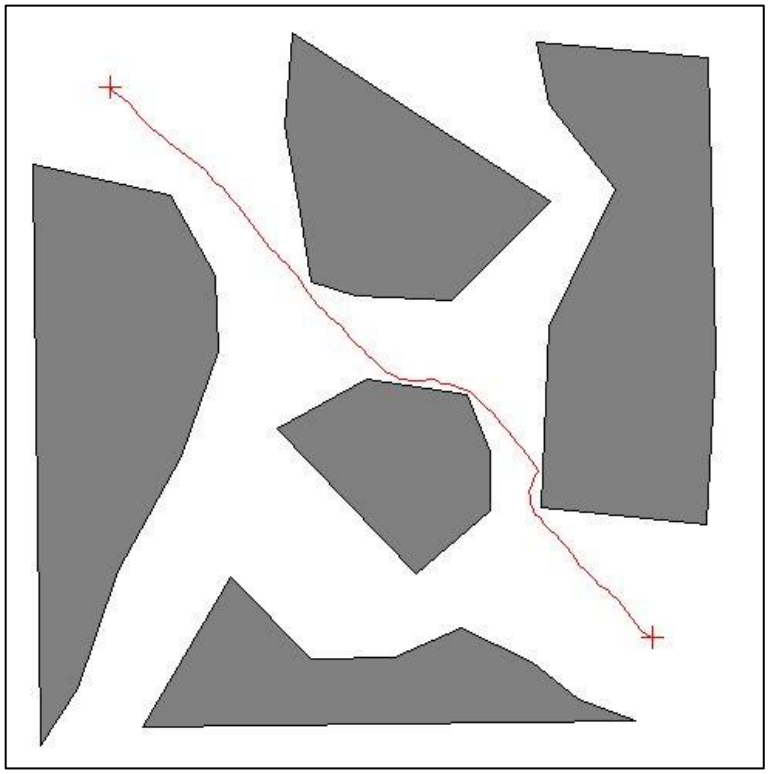
Tab. 6 Změřené hodnoty pro druhou scénu, kdy $w(t)$ se určí výpočtem.



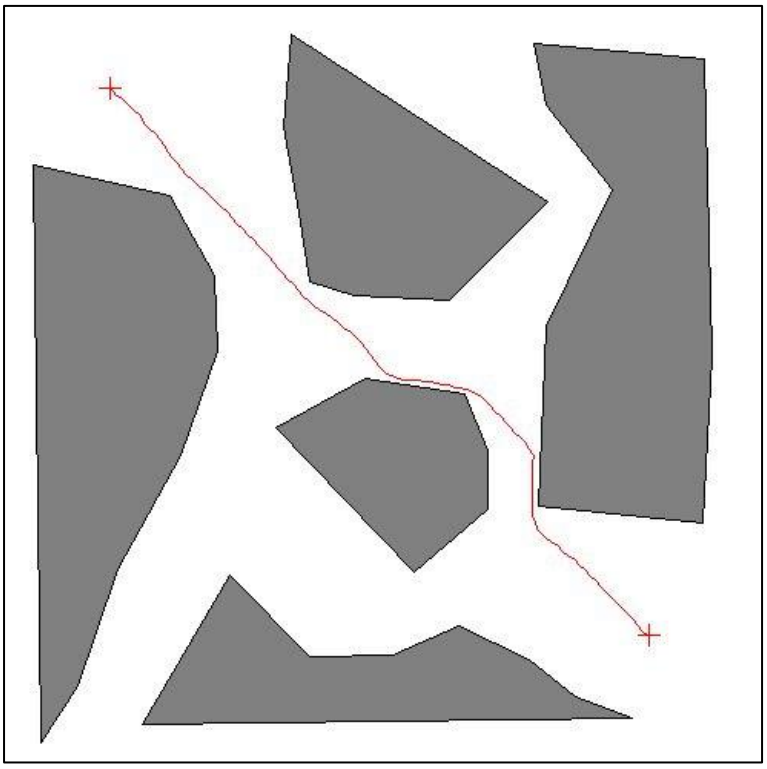
Obr. 53 Graficky znázorněná tab. 6.



Obr. 54 Řešení pomocí PSO.



Obr. 55 Řešení pomocí PSO1.



Obr. 56 Řešení pomocí PSO2.

Z výsledků je opět vidět, že setrvačná váha, kterou určujeme výpočtem, měla vliv na výsledek. Pro tuto scénu ale neměla jen negativní vliv. Průměrné délky drah a úspěšnost algoritmů se zlepšily, kromě průměrné délky dráhy u PSO2. Naproti tomu průměrné počty iterací a průměrné časy se zhoršily. Pro tuto scénu u PSO2 bych doporučil nastavit setrvačnou váhu jako konstantu a u ostatních algoritmů, jelikož se nám jedná o nejkratší dráhu, bych nastavil setrvačnou váhu pomocí výpočtu. V tomto experimentu měly opět nejkratší dobu výpočtu PSO a PSO2.

Co se týče potenciálového pole, tak délka dráhy a počet iterací byly po celou dobu jednotlivých výpočtů konzistentní. Jinými slovy pro každý nový výpočet vyšla délka dráhy a počet iterací stejně, zatímco u algoritmů PSO vycházely délky drah a počty iterací pokaždé jinak. Potenciálové pole mělo stoprocentní úspěšnost v provedených experimentech oproti algoritmům PSO. Co ale zaujme, je výsledek pro druhou scénu u PSO1, kdy setrvačná váha, která byla určena výpočtem. V tomto případě průměrná doba výpočtu přesáhla dobu výpočtu potenciálového pole. To je dáno tím, že PSO1 má fitness funkci, která pro každou částici neustále počítá minimální vzdálenost od jednotlivých překážek. Proto pokud máme velkou vzdálenost startu a cíle a poměrně početné a rozsáhlé překážky, tak výpočet trvá déle než výpočet potenciálového pole. Proto tento algoritmus je vhodný spíše na kratší vzdálenosti startu a cíle a menší počet a rozsah překážek.

7. Závěr

Cílem této diplomové práce bylo analyzovat přístupy k plánování cesty robota, popsat jednotlivé metody rojové inteligence, do kterých patří optimalizace mravenčí kolonií, optimalizace rojem světlušek, optimalizace rojem včel a optimalizace hejnem částic, a implementovat a experimentálně ověřit vybrané metody.

Pro návrh a implementaci systému pro plánování cesty robota jsem se po dohodě s vedoucím diplomové práce rozhodl navrhnout a implementovat systém pro plánování cesty robota pomocí optimalizace hejnem částic, která je inspirovaná chováním hejn ptáků a ryb. Pro porovnání jsem zvolil metodu potenciálového pole, kterou jsem upravil tak, abych snížil nebezpečí uváznutí v lokálním extrému.

Pro simulaci plánování cesty byl vytvořen program, který je součástí přílohy na CD. Tento program je popsán v této diplomové práci v kapitole 5. V programu jsou použity tři fitness funkce, se kterými hejnové algoritmy pracují. Tyto fitness funkce jsou inspirovány články [26], [27], [28] a [29]. Výhodou tohoto programu je možnost si vyzkoušet chování algoritmů při různých nastaveních a v rozmanitých pracovních prostorech.

Z experimentů vyplývá, že metoda potenciálového pole byla vždy úspěšná, přičemž vždy našla nejkratší cestu. To ale bylo vykoupeno dlouhým výpočtem. U optimalizace hejnem částic hodně záleží na i nastavení jednotlivých parametrů. Takový příkladem je nastavení setrvačné váhy.

Na základě provedených experimentů je možno konstatovat, že metody PSO a PSO2 jsou s ohledem na čas výpočtu a počet iterací lepší než metoda potenciálového pole a jsou tedy vhodné pro plánování cesty mobilního robota v prostředí se známými překážkami.

8. Použitá literatura

[1] SEDLÁK, V. Plánování cesty robota pomocí mravenčích systémů. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2011, 64 s. Vedoucí diplomové práce RNDr. Jiří Dvořák, CSc..

[2] ŠÍMA, M. Plánování cesty robota ve spojitém prostředí. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2006, 63 s. Vedoucí diplomové práce Ing. Petr Krček.

[3] SEDLÁK, V. Plánování cesty robota pomocí mravenčích systémů. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2009, 42 s. Vedoucí diplomové práce RNDr. Jiří Dvořák, CSc..

[4] ZOTOS, P. Path Planning with the humanoid robot iCub. Laussane, École Polytechnique Fédérale de Lausanne, 2008, 53 s. Semestrální práce.

[5] ŠEDA, M. Teorie grafů. Brno, Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2003, 89 s.

[6] DLOUHÝ, M. Pravděpodobnostní plánování. Dostupné z www: http://robotika.cz/guide/probplan/cs, [cit. 27.4.2013], 5.12.2003.

[7] BENI, G., WANG, J. Swarm Intelligence in Cellular Robotic Systems, Proceed. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 26–30, 1989.

[8] DORIGO, M., BIRATTARI, M., STÜTZLE, T. Ant Colony Optimization: Artificial Ants as a Computational Inteligence Technique. Université Libre de Bruxelles, Belgium. November 2006.

[9] DORIGO, M, STÜTZLE, T. Ant Colony Optimization. : MIT Press, 2004. 305 s.

[10] KRÖMER, P. Optimalizace pomocí mravenčích kolonií. Dostupné z www: http://homel.vsb.cz/~kro080/mravenci.pdf, [cit.: 1.5.2013], 22.2.2012, 16 s.

[11] KARABOGA, D., BASTURK, B. A Powerful Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm. Springer Science+Business Media B.V. 2007. 13 s.

[12] KARABOGA, D., BASTURK, B. Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems. Springer-Verlag Berlin Heidelberg 2007, 10 s.

[13] HADDAD, O., ABBAS, A., MARIÑO, M. Honey-Bees Mating Optimization (HBMO) Algorithm: A New Heuristic Approach for Water Resources Optimization. Water Resources Management, 20 (5):661–680, October 2006.

[14] MARINAKIS, Y., MARINAKIS, M. A Hybrid Honey Bees Mating Optimization Algorithm for the Probabilistic Traveling Salesman Problem. IEEE Congress on Evolutionary Computation, 2009, 8 s.

[15] ABBASS, H. A. Marriage in Honey Bees Optimization (MBO): A Haplometrosis Polygynous Swarming Approach. Proceedings of the 2001 Congress on Evolutionary Computation 2001, Sv. 1, 2001, s. 207-214.

[16] ČELEDA, T. Optimalizace pářením včelí královny. České Vysoké učení technické v Praze, Fakulta Elektrotechnická, 2011, 66 s.

[17] KRISHNANAND, K. N., GHOSE D. Glowworm Swarm Optimization for Simultaneous Capture of Multiple Local Optima of Multimodal Functions. Springer Science + Business Media, LCC 2008, 9 December 2008, 38 s.

[18] ŠINDELÁŘ J. Plánování cesty neholonomního mobilního robota. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2010, 75 s. Vedoucí diplomové práce Ing. Petr Krček.

[19] EBERHAR, R. C. Particle Swarm Optimization. Dostupné z www: ftp://ftp.cs.bham.ac.uk/.snapshot/hourly.0/pub/authors/W.B.Langdon/biblio/gecco2004/TUT027.pdf [Citováno dne:22.1.2013]

[20] KENNEDY, J., EBERHART, R. Partlicle Swarm Optimizaion. IEEEXplore Digital Libery. 1995, 7 s.

[21] HRČKA P. Plánování cesty robota pomocí hejnových algoritmů. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2010, 34 s.

[22] SASKA, M., MACAŠ, M., PŘEUČIL, L., LHOTSKÁ, L. Robot Path Planning using Particle Swarm Optimization of Ferguson Splines. IEEEXplore Digital Libery. 2006, 7 s.

[23] POLI, R., KENNEDY, J., BLACKVEL, T. Particle Swarm Optimization. Springer Science + Business Media. 2007, 25 s.

[24] KRISHNANAND, K. N., AMRUTH, P., GURUPRASAD, M. H., BIDARGADDI, S. V., GHOSE, D.. Glowworm-inspired Robot Swarm for Simultaneous Taxis towards Multiple Radiation Sources. Proceedings of the 2006 IEEE International Conference on Robotics and Automation Orlando, Florida - May 2006, 6 s.

[25] MASEHIAN E., SEDIGHIZADEH, D. A Multi-Objective PSO-based Algorithm fo Robot Path Planning. IEEEXplore Digital Library. 2010, 6 s.

[26] AHMADZADEH, S., GHANAVATI, M. Navigation of Mobile Robot Using the PSO Particle Swarm Optimization. Journal of Academic and Applied Studies (JAAS). 2012, 7 s.

[27] KENNEDY, J., EBERHART, R. Particle Swarm Optimization. IEEEXplore Digital Library. 1995. 7s.

[28] LI, G., YAMASHITA, A., ASAMA, H., TAMURA, Y. An Efficient Improved Artificial Potential Field Based Regression Search Method for Robot Path Planning. Proceedings of 2012 IEEE International Conference on Mechatronics and Automation August 5 - 8, Chengdu, China. 2011. 6s.

[29] GE, S. S., CUI, Y. J. New Potential Function fo Mobile Robot Path Planning. IEEE Transactions on Robotic and Automation, vol. 16, no. 5, October 2000, 6 s.

[30] LIU, C., GAO, Z., ZHAO, W. A New Path Planning Method Based on Firefly Algorithm. IEEEXplore Digital Library. 2012. 4s.