VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PLÁNOVÁNÍ TRASY BEZPILOTNÍCH LETOUNŮ POMOCÍ EVOLUČNÍCH ALGORITMŮ

UNMANNED AERIAL VEHICLE PATH PLANNING BY EVOLUTIONARY ALGORITHMS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Filip Miloševič**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jakub Kůdela, Ph.D.**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Filip Miloševič**
Studijní program:	Strojírenství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. Jakub Kůdela, Ph.D.**
Akademický rok:	2022/23

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 bakalářské práce:

Plánování trasy bezpilotních letounů pomocí evolučních algoritmů

Stručná charakteristika problematiky úkolu:

Student se seznámí s úlohami plánování trasy. Pro úlohu plánování trasy bezpilotního letounu v prostředí s hrozbami pak popíše odpovídající optimalizační model. Pro tento model vybere vhodné prostředí pro implementaci a použije několik evolučních algoritmů pro jeho řešení.

Cíle bakalářské práce:

Popis úlohy plánování trasy bezpilotního letounu v prostředí s hrozbami. Implementace optimalizačního modelu.
Implementace evolučních algoritmů.
Zhodnocení výsledků.

Seznam doporučené literatury:

WILLIAMS, P. H. Model Building in Mathematical Programming. Wiley, 2013.

TUY, H. Convex Analysis and Global Optimization. Springer, 2015.

BOYD, S. a VANDERBERGHE, L. Convex Optimization. Cambridge: Cambridge University Press, 2004.

PHUNG, M. D. a HA, Q. P. Safety-enhanced UAV path planning with spherical vector-based particle swarm optimization. Applied Soft Computing, 107, August 2021, 107376.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Tato bakalářská práce pojednává o problematice optimalizace tras bezpilotních letounů v 3D prostoru. Je tvořena rešerší současných znalostí této problematiky a implementací programu s grafickým rozhraním, který dokáže vytvořit optimalizovanou trasu pomocí algoritmů Artificial Bee Colony, Particle Swarm Optimalization a Whale Optimalization Algorithm. Následně byla v programu provedena série simulací, v nichž se jako nejvýkonnější pro řešení tohoto problému ukázal algoritmus Artificial Bee Colony.

ABSTRACT

This bachelor thesis deals with the problem of optimizing UAV routes in 3D space. It consists of a review of the current knowledge regarding the issue and the implementation of a program with a graphical interface that can create an optimized route using the Artificial Bee Colony, Particle Swarm Optimization and Whale Optimization Algorithms. Subsequently, a series of simulations were performed using the program, in which the Artificial Bee Colony algorithm proved to be the most effective algorithm in solving this issue.

KLÍČOVÁ SLOVA

Optimalizace tras bezpilotních letounů, evoluční optimalizační algoritmy, Artificial Bee Colony, Particle Swarm Optimalization, Whale Optimalization Algorithm

KEYWORDS

UAV Route Optimization, Evolutionary Optimization Algorithms, Artificial Bee Colony, Particle Swarm Optimalization, Whale Optimalization Algorithm

ÚSTAV AUTOMATIZACE A INFORMATIKY

2023

BIBLIOGRAFICKÁ CITACE

MILOŠEVIČ, Filip. *Plánování trasy bezpilotních letounů pomocí evolučních algoritmů.* Brno, 2023. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/149575. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Jakub Kůdela.

PODĚKOVÁNÍ

Poděkovat bych chtěl panu Ing. Jakubu Kůdelovi, Ph.D. za jeho cennou pomoc a odborné rady, které mi pomohly při tvorbě práce.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestně právních důsledků.

V Brně dne 20. 5. 2023 ……

Filip Miloševič

OBSAH

1 ÚVOD

Bezpilotní letouny a drony jsou v posledních letech čím dál více využívanými stroji. S jejich rostoucí oblibou však rostou i nároky na jejich efektivitu a bezpečnost. Tím pádem roste i důraz na efektivní plánování tras, které je pro bezpilotní letadla zásadní, jelikož jim umožňuje autonomně plnit své úkoly a zároveň optimalizovat určité cíle, jako jsou minimalizace letové doby, spotřeby energie nebo rizika. Evoluční algoritmy představují mocný nástroj pro hledání optimálních nebo téměř optimálních řešení složitých optimalizačních problémů, kterým optimalizace tras bezpilotních letounů je. Jsou inspirovány procesem přírodního výběru a evoluce, což jim pomáhá rychle a efektivně najít vhodné řešení.

Práce se zabývá touto problematikou a softwarovou implementací programu pro řešení optimalizačních problémů tohoto typu.

Úvodní kapitola obsahuje stručný úvod do problematiky optimalizace a základní dělení optimalizačních problémů. V další kapitole je čtenář obeznámen s modelem, který byl použit v praktické části. Následuje teoretický popis metaheuristických a evolučních algoritmů se zaměřením na specifické algoritmy, které byly využity při tvorbě programu, jenž je výsledkem praktické části. Jeho implementací se zabývá následující kapitola práce. Tento program obsahující uživatelské rozhraní je schopen najít optimalizovanou trasu pro zvolenou známou a neměnnou oblast. Optimalizace je provedena třemi různými evolučními algoritmy a je při ní kladen důraz především na bezpečnost letounu a minimalizaci délky letové dráhy. V závěrečné kapitole se nacházejí ukázky optimalizací s různými vstupními parametry a krátké zhodnocení použitých algoritmů.

2 OPTIMALIZACE

Optimalizace je proces hledání co nejlepšího řešení optimalizační úlohy, proto je vhodné objasnit, jak taková optimalizační úloha vypadá. Teorie k této kapitole byla čerpána z [1, 2, 3,4].

2.1 Optimalizace

Základní snahou při optimalizaci problému je najít minimální, popřípadě maximální hodnotu kriteriální (neboli účelové) funkce. Ta reprezentuje určité kritérium, které se snažíme optimalizovat. Při modelování problému je dále třeba brát zřetel na rozhodovací proměnné a omezení účelové funkce. Rozhodovací proměnné symbolizují určitá rozhodnutí, která by se měla při řešení modelu provést. Oproti tomu omezující funkce omezují rozsah jednotlivých proměnných.

Optimalizační model tedy může vypadat následovně:

$$\begin{gathered} min\ f(x_1, \dots, x_n) \\ \text{za podmínek } g_i(x_1, \dots, x_n) \ \leq b_i \end{gathered} \tag{1}$$

kde f je účelová funkce, $x_1, \dots, x_n$ jsou parametry, což mohou být rozhodovací proměnné nebo nezávislé proměnné, $g_i\ (x_1, \dots, x_n)$ je omezující funkce a b_i je konstanta, přičemž min může být nahrazeno *max* a $\leq$ může být nahrazeno jakýmkoliv jiným znaménkem rovnosti nebo nerovnosti.

Optimalizační úlohy lze rozdělit podle různých kritérií:

1. Rozdělujeme dle vlastností účelové funkce na:
   - lineární – v úloze jsou použity pouze lineární funkce
   - nelineární – některá z funkcí úlohy je nelineární
   - konvexní – v úloze jsou použity pouze konvexní funkce. V tomto případě je jakékoliv lokální minimum (případně maximum) i globálním minimem (maximem)
   - nekonvexní – některá z funkcí úlohy je nekonvexní

2. Dle povolených hodnot rozhodovacích proměnných:
   - Celočíselné úlohy – všechny proměnné mohou nabývat pouze celočíselných hodnot, jedná se tedy o diskrétní úlohy.
   - Smíšené celočíselné (Mixed-integer) úlohy – omezení na diskrétní hodnoty se vztahuje pouze na některé proměnné. Častým typem těchto úloh jsou binární úlohy, kdy některá z proměnných je omezena pouze na hodnoty1 a 0.
   - Spojité úlohy – proměnné mohou nabývat libovolných hodnot. Tyto úlohy se též nazývají reálné.

3. Dle počtu kriteriálních funkcí:
    - jednokriteriální – problém je popsán jednou kriteriální funkcí
    - multikriteriální – k popsání problému je zapotřebí alespoň dvou kriteriálních funkcí. Právě tímto typem problému je optimalizace tras, proto se jimi budeme zabývat podrobněji.

2.2 Multikriteriální modely

V případě složitějších problémů je potřeba k jejich popsání několik kriteriálních funkcí. Pokud tyto funkce nejsou triviální, často ani nejde najít jedno optimální řešení, jelikož zlepšením řešení pro jednu kriteriální funkci dojde ke zhoršení řešení pro některou z ostatních kriteriálních funkcí. Proto se u takových problémů hledá takzvané pareto-optimální řešení.

Pareto-optimální řešení je takové, kdy neexistuje změna, jež by zlepšila výsledek pro nějakou kriteriální funkci a zároveň by jej nezhoršila pro jinou. Množina pareto-optimálních řešení se nazývá Paretova množina. Ta však bývá u reálných problémů s více než dvěma kriteriálními funkcemi enormní, proto je výpočetně velmi náročné spočítat celou množinu a tím pádem i nepraktické. Z toho důvodu se multikriteriální modely zjednodušují na sadu jednokriteriálních. [1]

Dva často používané způsoby, jak toto zjednodušení provést, jsou:

- preemptivní (lexikografická) optimalizace – kriteriální funkce se srovnají v pořadí od nejdůležitější po nejméně důležitou a v tomto pořadí se provádí i optimalizace. Cíle se tedy zoptimalizují po jednom. Při použití této metody se předpokládá, že i sebemenší zlepšení pro důležitější funkci je preferováno před mnohem větším zlepšením méně důležité funkce. Výhodou preemptivní optimalizace je, že snadno vede k pareto-optimálnímu řešení. Pokud je každá funkce optimalizována jako jednokriteriální problém, nelze pro ni vybrat řešení, které by nezhoršilo výsledek pro některou z funkcí, která byla optimalizována před ní.
- Vážený součet – při této metodě se výsledky optimalizací pro jednotlivá kritéria sečtou do vážené sumy, pro kterou je následně celý model optimalizován stejně, jako by byl jednokriteriální. Výsledek je tedy v podstatě vybrán na základě váhy, která je přidělená každé kriteriální funkci. Oproti preemptivní optimalizaci je tato metoda mnohem více flexibilní, díky čemuž je použita i při optimalizaci zvoleného modelu.

Vhodnými algoritmy na řešení multikriteriálních úloh jsou evoluční a genetické algoritmy [5], jejichž popis je obsažen ve 4. kapitole této práce. Tato vlastnost je dána tím, že jsou schopny prohledat komplexní pole řešení a udržovat si více kandidátních řešení.

3 MODEL

Pro tuto práci byl vybrán model publikovaný v práci [6]. Jelikož se s ním dále v práci pracuje, je na místě jej popsat. Model je určen primárně pro bezpilotní letouny, které se zabývají leteckým snímkováním, mapováním a inspekcí povrchu. Předpokládá se, že letoun využívající model je ovládaný z pozemní stanice. Model byl zvolen, jelikož je v publikaci srozumitelně a podrobně popsán, a na základě doporučení vedoucího práce.

3.1 Popis modelu

Základem modelu je předem známá ohraničená mapa, která se v průběhu hledání optimální trasy nemění. Stejně tak i poloha počátečního a koncového bodu je během optimalizace konstantní. Aby byl model platný, nesmí letoun vyletět mimo hranice této mapy. Na mapě jsou rozmístěné překážky, kterým se musí letoun vyhnout. Jejich poloha je také předem známá. Stejně tak je známá výška neboli souřadnice z každého bodu na mapě.

Trajektorie je diskretizována pomocí *n* bodů, mezi kterými letoun letí po přímé trase. Tyto body jsou popsány souřadnicemi x, y a z.

Pro optimalizaci modelu byly vytvořeny čtyři kriteriální funkce, které popisují důležité vlastnosti nalezené trasy. Tyto kriteriální funkce jsou následně sečteny pomocí váženého součtu. Výsledný součet je minimalizován pomocí vybraných algoritmů.

3.2 První kriteriální funkce

První kriteriální funkce, popsaná rovnicí (2), minimalizuje celkovou délku trasy, což šetří letový čas a náklady na provoz letounu:

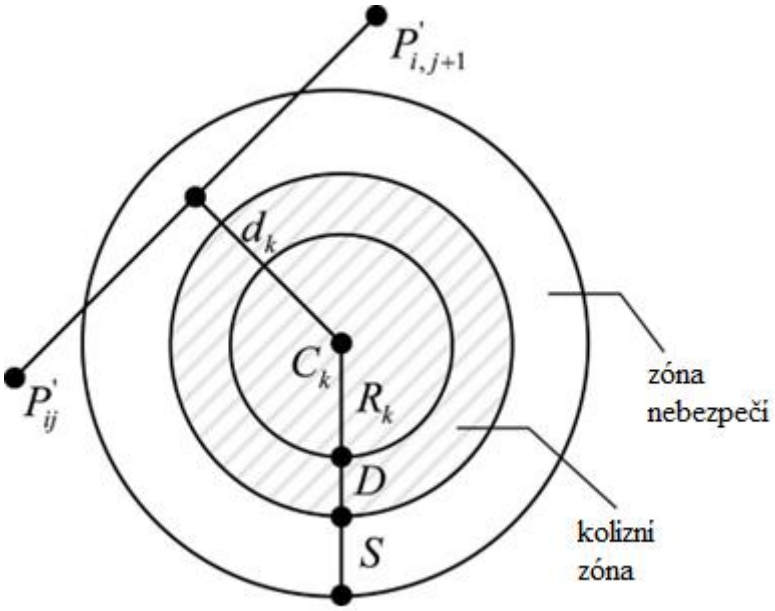
$$F_1(X_i) = \sum_{j=1}^{n-1} \left\| P_{ij}\vec{P}_{i,j+1} \right\| \tag{2}$$

F_1 v této rovnici popisuje první kriteriální funkci, X_i je letová trasa i, $P_{ij} = (x_{ij}y_{ij}z_{ij})$ jsou jednotlivé uzly, které odpovídají výše zmíněným bodům dráhy letu. $\left\| P_{ij}\vec{P}_{i,j+1} \right\|$ je vzdálenost dvou uzlů. První kriteriální funkce je tedy pouze sumou vzdáleností mezi uzly.

3.3 Druhá kriteriální rovnice

Optimální trasa musí být zároveň i bezpečná. Letoun tedy musí být schopen vyhnout se případným překážkám. Pro zjednodušení modelu se předpokládá, že všechny takové

překážky jsou kruhové se známou polohou a průměrem. Kolem každé z nich jsou pro snadnější výpočet hodnoty kriteriální funkce vytvořeny dvě imaginární zóny. Kolizní zóna a zóna nebezpečí. Obě zóny jsou vyobrazeny na obr. 1. Kolizní zóna má velikost průměru překážky, zvětšeného o velikost letounu. V této zóně hrozí bezprostřední nebezpečí kolize. Zóna nebezpečí má velikost průměru překážky zvětšeného o desetinásobek velikosti letounu. V zóně nebezpečí začíná hrozit srážka s překážkou.



Obr. 1: Grafické vyobrazení zón, převzato z [6]

d_k na obrázku a v rovnici (4) vyjadřuje vzdálenost letounu od překážky, C_k polohu překážky, R_k rádius překážky, D označuje vzdálenost od kolizní zóny a S představuje vzdálenost od zóny nebezpečí.

$$F_2(X_i) = \sum_{j=1}^{n-1} \sum_{k=1}^{K} T_k(P_{ij}\vec{P}_{i,j+1}) \tag{3}$$

Samotná kriteriální funkce F_2 je popsána v rovnici (3), kde K odpovídá množině všech překážek a T_k je funkce, jejíž hodnota závisí na vzdálenosti od překážky. Pokud se letoun nachází mimo zónu nebezpečí překážky, hodnota T_k je nulová. Pokud se letoun nachází v zóně nebezpečí, ale mimo kolizní zónu, hodnota T_k je rovna vzdálenosti letounu od hranice zóny nebezpečí. Pokud by však letoun vletěl až do kolizní zóny, pak by hodnota T_k byla rovna nekonečnu. Hodnoty T_k lze vidět v rovnici (4).

$$T_k\left(P_{ij}\vec{P}_{i,j+1}\right) = \begin{cases} 0, & pokud\ d_k > S + D + R_k \\ (S + D + R_k) - d_k, & pokud\ D + R_k < d_k < S + D + R_k \\ \infty, & pokud\ d_k \leq D + R_k \end{cases} \tag{4}$$

3.4 Třetí kriteriální funkce

Překážky však nejsou jediné, do čeho nesmí letoun narazit. Další problém, kterému musíme při optimalizaci trasy předejít, je náraz do země, což v modelu zajišťuje třetí kriteriální funkce.

Aby byl letoun schopný vykonávat některé své funkce, jako například letecké snímkování, je potřeba, aby se držel v určitém výškovém rozmezí od minimální výšky h_{min} po maximální výšku h_{max}, přičemž ideálně by si měl udržovat konstantní výšku nad povrchem země. Z tohoto důvodu zavádíme funkci H_{ij}. V případě, že letoun letí výš, než je maximální nastavená výška h_{max}, nebo níž než minimální výška h_{min}, funkce H_{ij} je rovna nekonečnu. Pokud se letoun nachází v povoleném rozmezí, funkce H_{ij} je rovna rozdílu mezi aktuální výškou letounu a průměrnou výškou povoleného rozmezí. To je patrné z rovnice (5), kde h_{ij} představuje výšku letounu v bodě j trasy i.

$$H_{ij} = \begin{cases} \left| h_{ij} \dfrac{(h_{max} + h_{min})}{2} \right|, \text{pokud } h_{min} \leq h_{ij} \leq h_{max} \\ \infty, \text{v ostatních případech} \end{cases} \tag{5}$$

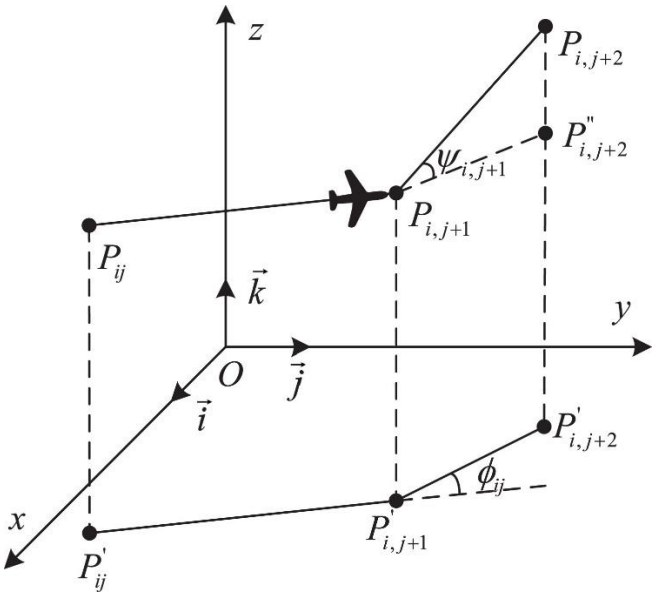
Kriteriální funkce F_3 je sumou funkce H_{ij} pro všechny body.

$$F_3(X_i) = \sum_{j=1}^{n} H_{ij} \tag{6}$$

3.5 Čtvrtá kriteriální funkce

Zatím může letoun v modelu provádět jakékoliv zatáčky, jako například náhlou otočku o 180°, což v realitě není možné. Aby výsledná trasa získaná optimalizací modelu byla proveditelná, o to se stará čtvrtá kriteriální funkce.

Tato kriteriální funkce se skládá ze dvou částí. První část vyhlazuje úhel zatáčení ϕ_{ij} a druhá vyhlazuje úhel stoupání ψ_{ij}. K jejich výpočtu je potřeba promítnout uzly P_{ij} a $P_{i,j+1}$ do horizontální roviny, jak je naznačeno v obr. 2:



Obr. 2: Znázornění úhlu stoupání a zatáčení, převzato z [6]

Segmenty trasy promítnuté do horizontální roviny $P'_{ij}\overrightarrow{P'_{i,j+1}}$ se spočítají dle rovnice (7), kde $\vec{k}$ je jednotkový vektor ve směru osy z a $P_{ij}\overrightarrow{P_{i,j+1}}$ jsou nepromítnuté segmenty:

$$P'_{ij}\overrightarrow{P'_{i,j+1}} = \vec{k} \times (P_{ij}\overrightarrow{P_{i,j+1}} \times \vec{k}) \tag{7}$$

Úhel zatočení se spočítá jako:

$$\phi_{ij} = \arctan\left(\frac{\left\| P'_{ij}\overrightarrow{P'_{i,j+1}} \times P'_{i,j+1}\overrightarrow{P'_{i,j+2}} \right\|}{P'_{ij}\overrightarrow{P'_{i,j+1}} \cdot P'_{i,j+1}\overrightarrow{P'_{i,j+2}}}\right) \tag{8}$$

Úhel stoupání se spočítá jako:

$$\psi_{ij} = \arctan\left(\frac{z_{i,j+1} - z_{ij}}{\left\| P'_{ij}\overrightarrow{P'_{i,j+1}} \right\|}\right) \tag{9}$$

Samotná kriteriální funkce F_4 se spočítá následovně, přičemž a_1 a a_2 jsou penalizační koeficienty za úhel zatočení a úhel stoupání v tomto pořadí:

$$F_4(X_i) = a_1 \sum_{j=1}^{n-2} \phi_{ij} + a_2 \sum_{j=1}^{n-1} \left|\psi_{ij} - \psi_{i,j-1}\right| \tag{10}$$

3.6 Nákladová funkce

Samotnou nákladovou funkci (cost function), kterou se snažíme minimalizovat, získáme váženou sumou všech čtyř kriteriálních funkcí, jak je patrné v rovnici (11). b_k v rovnici symbolizuje váhové koeficienty, pomocí kterých můžeme upravovat důraz, který je během optimalizace kladen na jednotlivé kriteriální funkce.

$$F(X_i) = \sum_{k=1}^{4} b_k F_k(X_i) \tag{11}$$

Hodnoty b_k, které byly použity v praktické části, jsou uvedeny v tab.1

	$\boldsymbol{b_1}$	$\boldsymbol{b_2}$	$\boldsymbol{b_3}$	$\boldsymbol{b_4}$
Použitá hodnota $\boldsymbol{b_k}$:	5	1	10	1

Tab. 1: Hodnoty b_k použité v praktické části

4 ALGORITMY

K optimalizaci jakéhokoliv modelu je potřeba algoritmus. U výpočetně náročných problémů, jakým je zvolený model, se často využívají takzvané metaheuristické algoritmy [7].

Metaheuristiky se vyznačují tím, že se nesnaží najít jedno nejvíce optimální, popřípadě pareto-optimální řešení. Místo toho se snaží najít co nejlepší možné řešení, a to před splněním některého z kritérií zastavení. Tím může být mimo jiné uplynulý čas nebo počet iterací algoritmu. Jsou tedy vhodné pro úlohy, u kterých stačí „dobré“ řešení, ale v omezeném čase [8]. To je i případem optimalizace tras bezpilotních dronů, kde je nežádoucí čekat dlouho na výsledek.

Metaheuristické algoritmy se dělí na dva druhy, jedno-výsledkové a populační.

- Jedno-výsledkové algoritmy – najdou jedno potenciální řešení, a to se následně snaží zlepšovat zkoumáním podobných řešení. Jsou proto vhodné v případech, kdy množina možných řešení není příliš velká. Jejich nevýhodou však je, že se mohou zaseknout na lokálním optimu, místo aby našly globální optimum. Proto jsou efektivní spíše v případech, kdy jsou kriteriální funkce hladké a spojité.

- Populační algoritmy – najdou několik potenciálních řešení. Počet těchto řešení závisí na populaci, se kterou algoritmus pracuje. Tato potenciální řešení jsou v průběhu optimalizace kombinována a zlepšována ve snaze najít řešení, které je co nejblíže optimálnímu nebo pareto-optimálnímu. Populační algoritmy jsou schopné vyhodnotit více potenciálních řešení, a tím pádem jsou vhodnější pro problémy, které mají mnoho možných řešení. Zároveň je u nich méně pravděpodobné, že by se zasekly na lokálním optimu. Potřebují však mnohokrát vyhodnotit kriteriální funkce, což s sebou nese výpočetní náročnost. [9,10]

Pro model optimalizovaný v této práci, kdy potenciálními body výsledné trasy jsou všechny body na mapě, jsou vhodnější populační algoritmy. Populační algoritmy jsou často inspirované přírodou, ať už chováním zvířat (Artificial bee colony, ant colony) nebo fyzikálními jevy (Gravitational search algorithm, Simulated Annealing), největší skupinou populačních algoritmů však jsou evoluční algoritmy.

4.1 Evoluční algoritmy

Evoluční algoritmy jsou inspirovány procesem přírodního výběru. Výběr nejvhodnějšího řešení u nich probíhá tak, že se náhodně vytvoří množina řešení, ze které se několik nejlepších řešení stane rodiči. Tato vybraná řešení jsou následně kombinována, aby byla vytvořena nová řešení [11]. Kombinace řešení probíhá různými způsoby, jako například křížením nebo mutací. Při křížení se některé části řešení rodičů

vymění, čímž vznikne nová generace řešení. Oproti tomu při mutaci se náhodně změní jeden jedinec. S takto vzniklou novou generací se stejný proces opakuje, dokud není splněno některé z předem daných kritérií. Tím může být mimo jiné maximální uplynulý čas, který algoritmus má na optimalizaci, nebo maximální počet iterací evolučního algoritmu.

Výše popsaný model byl optimalizován pomocí tří evolučních algoritmů. Těmi jsou Artificial Bee Colony, Particle Swarm Optimalization a Whale Optimalization Algorithm. Problém optimalizace tras bezpilotních letounů se však dá řešit i mnoha jinými algoritmy, například cuckoo search [12], differential evolution [13] nebo ant colony optimalization [14]

4.2 Artificial Bee Colony [15]

Artificial Bee Colony (ABC) je evoluční algoritmus, který navrhl Derviş Karaboğa v roce 2005. Algoritmus byl inspirován chováním včel při hledání potravy.

V algoritmu ABC jsou kandidáti na řešení reprezentováni populací včel. Ty se dělí na tři typy: dělnice, pozorovatele a průzkumníky. Dělnice reprezentují jednotlivá řešení optimalizovaného problému. Pozorovatelé symbolizují proces výběru nejvhodnějších řešení. Průzkumníci představují proces hledání nových řešení. Jednotlivá řešení jsou reprezentována D-dimenzionálním vektorem x_i, kde $i \in \{1,2,\dots,SN\}$ a SN je počet včel dělnic. D je rovno počtu optimalizačních parametrů.

Samotný algoritmus má tři hlavní fáze: inicializační fázi, fázi včel dělnic a fázi včel pozorovatelů. V inicializační fázi je všem včelám dělnicím náhodně přiřazeno jedno řešení, které si uchovají do paměti. Ve fázi včel dělnic najde každá včela dělnice nové potenciální řešení v_{ij} blízké tomu, které si již pamatuje. Pozice v_{ij} se určí za pomoci rovnice (12), kde $k \in \{1,2,\dots,SN\}$ a $j \in \{1,2,\dots,D\}$ jsou náhodně zvolené indexy. φ_{ij} je náhodné číslo v rozmezí $[-1,1]$.

$$v_{ij} = x_{ij} + \varphi_{ij}\left(x_{ij} - x_{kj}\right) \tag{12}$$

Toto nové řešení dělnice nejprve vyhodnotí. Vyhodnocení probíhá porovnáním *fitness* hodnoty, jejíž výpočet je uveden v rovnici (13), kde hodnota *fitness* pro řešení *i* je zapsána jako fit_i a f_i reprezentuje hodnotu kriteriální funkce pro řešení *i*.

$$fit_i = \frac{1}{1 + f_i} \tag{13}$$

Má-li dělnicí nově nalezené řešení větší hodnotu *fitness* než to staré, včela staré řešení zapomene a místo něj si zapamatuje nové. Pokud má naopak staré řešení větší *fitness*, tak si dělnice uchová v paměti pouze to staré.

Následuje fáze včel pozorovatelů. V této fázi vyhodnotí každý pozorovatel všechna řešení nalezená dělnicemi a z nich náhodně jedno vybere. Přestože je výběr náhodný, řešení s lepší hodnotou *fitness* mají větší šanci být vybrána, jelikož je všem

řešením přidělena pravděpodobnost výběru p_i, která je přímo úměrná jejich *fitness* hodnotě. Hodnota pravděpodobnosti p_i se spočítá dle rovnice (14).

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

Řešení, která se za určitý počet cyklů nepodaří vylepšit, jsou zahozena a nahrazena. Tento počet cyklů se nazývá *limit* a je jedním ze vstupních parametrů algoritmu.

Dělnice, jejichž řešení byla opuštěna, se stanou průzkumníky a hledají nová řešení, kterými nahradí nevybraná řešení. Nejlepší doposud nalezená řešení jsou na závěr zapsána a proces se od fáze dělnic opakuje, dokud nejsou splněna kritéria zastavení algoritmu.

ABC je výkonný a zároveň všestranný algoritmus s jednoduchou implementací. Kromě *limitu* je potřeba definovat pouze počet včel *N*. Díky své jednoduchosti není k použití algoritmu potřeba velké množství odborných znalostí a zároveň je velmi flexibilní a všestranný. Díky těmto vlastnostem je při optimalizaci methauristickým algoritmem populární volbou a kvůli nim byl i vybrán pro tuto práci.

4.3 Particle Swarm Optimalization [16,17,18]

Particle Swarm Optimalization (PSO) je optimalizační algoritmus vycházející z chování rojů hmyzu a hejn ptáků. Tento evoluční algoritmus, představený J. Kennedym a R. Eberhartem v roce 1995, funguje na principu roje částic, kde každá částice představuje potenciální řešení. Každá částice *i* má svoji dráhu, kterou postupně při hledání optima upravuje. Dráha je určena polohou x_i a rychlostí v_i. V každé poloze má každá částice *fitness* hodnotu, která odpovídá vhodnosti řešení v té dané poloze.

Na začátku optimalizace algoritmus vygeneruje částice s náhodnou pozicí. V každé iteraci algoritmu jsou dráhy částic aktualizovány na základě globální nejlepší pozice a osobní nejlepší pozice. Osobní nejlepší pozice odpovídá nejlepšímu řešení, které ta daná částice sama navštívila. Jak velký vliv na vylepšení dráhy částice bude osobní nejlepší pozice mít, určuje vstupní parametr míra učení C_1. Globální nejlepší pozice jsou nejlepší řešení nalezená kteroukoliv částicí roje. Vliv globálního nejlepšího řešení na dráhy částic určuje vstupní parametr míra učení C_2. Hodnoty C_1 a C_2 jsou zpravidla nastaveny v rozmezí od 0 do 2. Mohou být nastaveny i na vyšší hodnoty, ale většinou to má negativní vliv na výkon algoritmu. Vždy však musí být větší než 0.

V časovém úseku t a v D-dimenzionálním prostoru je pozice a rychlost i-té částice popsána D-dimenzionálním vektorem $x_i^t = (x_{i1}^t, x_{i2}^t, \dots, x_{iD}^t)^T$ a $v_i^t = (v_{i1}^t, v_{i2}^t, \dots, v_{iD}^t)^T$. Nejlepší doposud navštívené řešení i-tou částicí je také popsáno vektorem a to $p_i^t = (p_{i1}^t, p_{i2}^t, \dots, p_{iD}^t)^T$.

Aktualizace rychlosti a pozice je provedena dle následujících rovnic:

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{15}$$

$$v_{id}^{t+1} = v_{id}^{t} + C_1 r_1 (p_{id}^{t} - x_{id}^{t}) + C_2 r_2 (p_{gd}^{t} - x_{id}^{t}) \tag{16}$$

Index d v těchto rovnicích představuje dimenzi $d = 1,2,\dots,D$ a $i = 1,2,\dots,S$ je index částice. S symbolizuje počet částic. r_1 a r_2 jsou náhodná čísla z intervalu $[0,1]$.

PSO je vhodný algoritmus na komplexní optimalizační úlohy a je použitelný při řešení široké škály různých optimalizačních problémů. Má jednoduchou implementaci, pro kterou je potřeba určit 4 vstupní parametry. Těmi jsou populace N, míry učení (learning rate) C1 a C2 a váha setrvačnosti (inertia weight) ω. Na váze setrvačnosti závisí kompromis mezi hledáním nových potenciálních optim a zlepšováním nalezených hodnot. S vyšší hodnotou ω se částice pohybují rychleji, což jim umožňuje prozkoumat více rozdílných řešení, oproti tomu za nižších hodnot ω se částice pohybují pomaleji, a tím pádem důkladněji vyšetří slibné oblasti [19].

PSO má však také své nevýhody. Tou hlavní je náklonnost k předčasné konvergenci, kdy algoritmus může konvergovat k lokálnímu optimu namísto globálního optima. K řešení tohoto nedostatku existují modifikované verze PSO, jako například hybridní modely [20] nebo diversity-preservation [21]. Tyto metody však nebyly v této práci použity z důvodu jejich složitosti.

4.4 Whale Optimalization Algorithm [22]

Whale Optimalization Algorithm (WOA) je metaheuristický optimalizační algoritmus, který v roce 2016 navrhli Seyedali Mirjalili a Andrew Lewis. WOA je inspirován chováním keporkaků při lovu. Tyto velryby v algoritmu reprezentují možná řešení.

Velryby při lovu plavou po spirálové dráze a zároveň vydávají zvuky, aby lokalizovaly svou kořist. Jakmile je kořist lokalizována, použijí k jejímu zachycení bublinkovou síť a poté plavou vzhůru, aby ji chytily. Algoritmus WOA napodobuje toto chování pomocí tří různých technik: vyhledávání kořisti, obkličování kořisti a útok spirálovitou bublinkovou sítí.

Ve fázi hledání algoritmus náhodně zkouší různá řešení s cílem najít globální optimální řešení. To je provedeno těmito rovnicemi:

$$\vec{D} = \left| \vec{C} \cdot \overrightarrow{X_{rand}} - \vec{X} \right| \tag{17}$$

$$\vec{X}(t+1) = \left| \overrightarrow{X_{rand}} - \vec{A}.\vec{D} \right| \tag{18}$$

V rovnicích t označuje momentální iteraci, $\vec{D}$ je vzdálenost dané velryby od jedince s nejlepší nalezenou pozicí, $\vec{X}$ je vektor pozice a $\overrightarrow{X_{rand}}$ je pozice náhodné velryby.

$\vec{A}$ a $\vec{C}$ jsou vektory koeficientů a spočítají se následovně:

$$\vec{A} = 2\vec{a} \cdot \vec{r} - \vec{a} \tag{19}$$

$$\vec{C} = 2 \cdot \vec{r} \tag{20}$$

Hodnota koeficientu $\vec{a}$ je v průběhu iterací lineárně snižována ze 2 na 0 a $\vec{r}$ je náhodný vektor z intervalu $[0,1]$.

Ve fázi obkružování se algoritmus snaží obklopit dosud nejlepší nalezené řešení a zúžit množinu potenciálních optimálních řešení. Tento proces je popsán následujícími rovnicemi:

$$\vec{D} = \left|\vec{C}.\overrightarrow{X^*}(t) - \vec{X}(t)\right| \tag{21}$$

$$\vec{X}(t+1) = \overrightarrow{X^*}(t) - \vec{A}.\vec{D} \tag{22}$$

$\overrightarrow{X^*}$ je vektor pozice nejlepšího dosud nalezeného řešení. To by se mělo aktualizovat při každé iteraci.

Nakonec ve fázi spirály algoritmus konverguje k nejlepšímu řešení pomocí spirálové cesty, podobně jako keporkaci obkružují kořist v bublinkové spirále. Tato část algoritmu je popsána dvěma rovnicemi, z nichž je při iteraci jedna náhodně vybrána. Obě tedy mají 50% šanci, že budou použity:

$$\vec{X}(t+1) = \begin{cases} \overrightarrow{X^*}(t) - \vec{A}.\vec{D} \\ \vec{D'} \cdot e^{bl} \cdot \cos(2\pi l) + \overrightarrow{X^*}(t) \end{cases} \tag{23}$$

kde $\vec{D'}$ je rovno vzdálenosti velryby od kořisti, b je konstanta, která pomáhá definovat tvar logaritmické spirály, a l je náhodné číslo v rozmezí $[-1,1]$.

WOA byl vybrán z podobných důvodů jako PSO a ABC. Je to výkonný a všestranný algoritmus s jednoduchou implementací. Jediným parametrem, který je potřeba pro jeho implementaci definovat, je počet velryb N.

5 IMPLEMENTACE

Pro implementaci úloh byl zvolen programovací jazyk Julia programovaný v programovacím prostředí Pluto.jl. Programovací jazyk i prostředí byly vybrány především na základě předchozích zkušeností, mají však také značná pozitiva. Mezi ně patří jeho jednoduchost z uživatelského pohledu. Julia má syntax připomínající Python nebo Matlab, díky čemuž je snadno naučitelný. Pokud jde o výkon, je Julia obecně považován za rychlejší jazyk pro většinu úloh oproti výše zmíněným jazykům [23,24]. Zároveň má Julia mnoho balíčků, které usnadňují různé aspekty práce (viz. tabulka 2).

Knihovna	Použití
Metaheuristics	Optimalizace modelu
TiffImages	Nahrání tiff obrázku
LinearAlgebra	Matematické operace
Plots	Vykreslování grafů
PlutoUI	Tvorba uživatelského rozhraní

Tab. 2: Seznam použitých knihoven

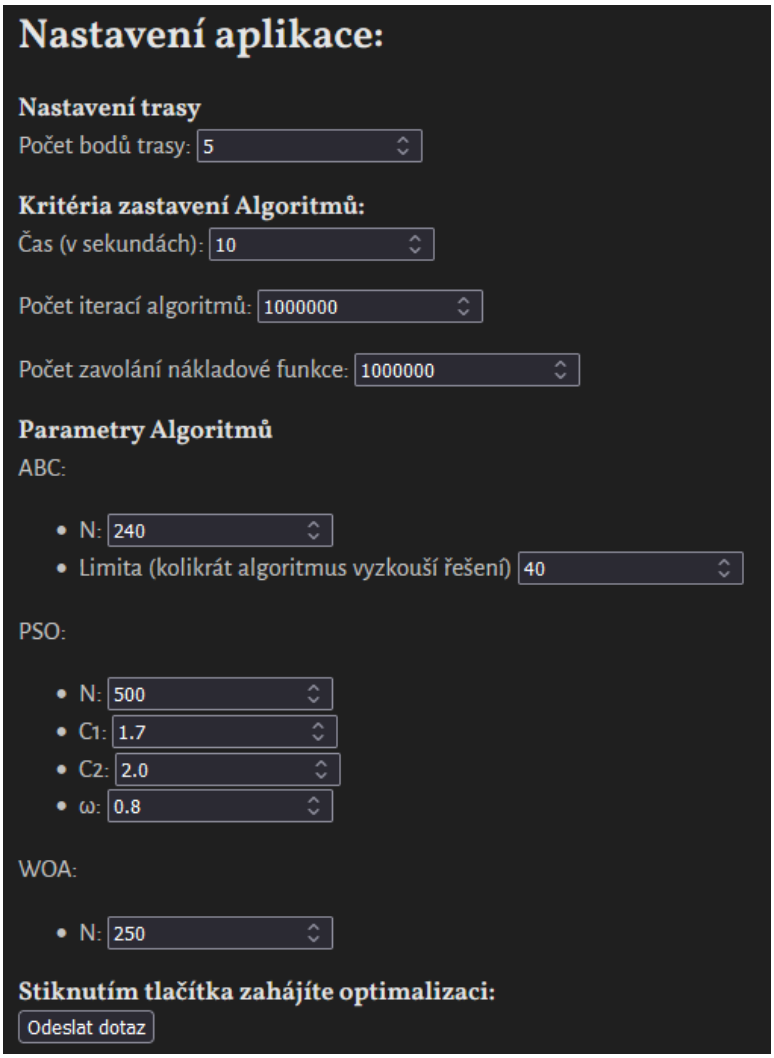
5.1 Pluto.jl

Prostředí Pluto.jl je reaktivní programovací prostředí, ve kterém je výsledný program tvořen z buněk. Tyto buňky jsou propojené, změna v jedné buňce se tedy okamžitě projeví ve všech, které jsou na ní závislé. Pro prostředí byla vyvinuta knihovna PlutoUI, která jej doplňuje o snadno implementovatelné prvky uživatelského rozhraní, jako například tlačítka nebo posuvníky. Následně je možné propojit tyto prvky s proměnnými, což spojeno s reaktivitou prostředí znamená, že například zmáčknutí tlačítka se okamžitě projeví v programu.

5.2 Uživatelské prostředí

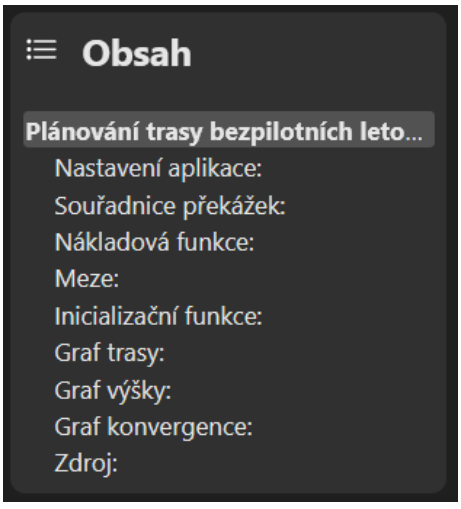
Reaktivita prostředí Pluto.jl je však v případě této práce nevýhodou. Znamená totiž, že při jakékoliv změně, kterou uživatel provede, započne ihned optimalizace modelu. Ta však může trvat od desítek sekund až po několik hodin dle nastavení. Muselo být proto implementováno potvrzovací tlačítko. Nastavené změny se tedy uplatní až po zmáčknutí tohoto tlačítka.

Samotné uživatelské prostředí se krom potvrzovacího tlačítka skládá z jedenácti číselných polí, ve kterých může uživatel nastavit počet bodů trasy *n*, kritéria zastavení a parametry algoritmů, jak je patrné z obr.3.



Obr. 3: Nastavení aplikace

Důležité části kódu jsou označeny nadpisem. Pro rychlejší orientaci mezi nadpisy byl implementován obsah, který je možné vidět na obr. 4.



Obr. 4: Obsah aplikace

5.3 Implementace modelu

Při spuštění programu se do něj nahraje soubor typu tiff. Tento soubor obsahuje data o velikosti plochy, se kterou bude model pracovat, a výšková data každého bodu plochy. Jako další se definují překážky. Souřadnice a rádius překážek je následně zapsán do matice. Následuje definování funkce *DistToThread*. Ta počítá vzdálenost letounu od jedné dané překážky v nejbližším bodě průletu.

Jako další je definována funkce *CostFunction*. Ta obsahuje všechny 4 kriteriální funkce a jejich vážený součet včetně váhových koeficientů. Vstupními proměnnými této funkce jsou proměnné x, y a z neboli vektory, které odpovídají souřadnicím bodů, jež má nalézt optimalizační algoritmus. Počet složek těchto vektorů je roven počtu bodů *n*, ze kterých se má skládat výsledná dráha letounu.

První kriteriální funkce obsažená ve funkci *CostFunction* byla oproti modelu publikovaném v [6] lehce upravena, viz obr. 5. Tato kriteriální funkce počítá vzdálenost mezi jednotlivými body trasy. V této práci však byla doplněna o dva dodatky. Zaprvé, pokud je vzdálenost dvou bodů větší něž 400 bodů mapy, hodnota kriteriální funkce je zdvojnásobena. Zadruhé, pokud je vzdálenost dvou bodů menší než 20 bodů mapy, hodnota kriteriální funkce je vynásobena tolikrát, o kolik bodů je vzdálenost menší než 20. Tyto dodatky byly zavedeny, jelikož zejména algoritmus WOA měl při optimalizacích s větším počtem bodů trasy *n* tendenci hromadit mnoho bodů velmi blízko sebe. Oproti tomu algoritmus PSO zase umisťoval body ve velkých vzdálenostech od sebe. Úpravy kriteriální funkce jsou snahou takovému chování předejít a podpořit rovnoměrnější rozmístění bodů trasy.

```
## F1 - optimalizace pro co nejkratší vzdálenost
F1=0
for i in 1:N-1
    length_diff=[x_all[i+1]-x_all[i];y_all[i+1]-y_all[i];z_abs[i+1]-z_abs[i]]
    if norm(length_diff) > 400
        F1=F1+norm(length_diff)*2
    elseif norm(length_diff) < 20
        F1=F1+norm(length_diff)*(20-norm(length_diff))
    else
    F1=F1+norm(length_diff)
    end
end
```

Obr. 5: Upravená první kriteriální funkce

Následuje druhá kriteriální funkce. K jejímu výpočtu se používá výše zavedená funkce *DistToThread*. Pro usnadnění výpočtu třetí a čtvrté kriteriální funkce je výška letounu přepočítána na nadmořskou výšku, viz rovnice (24).

$$Z_{abs} = Z_{rel} + Z_{mnm} \tag{24}$$

Z_{abs} v této rovnici vyjadřuje absolutní výšku letounu, se kterou počítají kriteriální funkce. Z_{rel} je relativní výška letounu oproti zemi pod ním. Z_{mnm} je nadmořská výška země v bodě kolmo pod letounem.

5.4 Optimalizace

Samotná optimalizace modelu byla provedena pomocí Julia knihovny Metaheuristics.jl, která tento proces značně usnadňuje.

Knihovna obsahuje 10 různých metaheuristických algoritmů. Po zvolení některého z nich, po zavedení jeho parametrů a kritérií zastavení se pomocí něj optimalizuje zvolená funkce zavoláním zabudované funkce *optimize*.

Balíček dále obsahuje prostředky, které pomáhají vizualizovat výsledek optimalizace i její průběh, a to například zobrazením konvergence k výsledné hodnotě optimalizace. Poslední součástí balíčku je sada problémů, jejichž optimalizací lze vyzkoušet různé přístupy a algoritmy. Balíček také umožňuje upravovat obsažené algoritmy nebo i tvořit nové, to však z důvodu nutnosti vysokých odborných znalostí není součástí této práce.

Algoritmy použité ve funkci *optimize* jsou nastavené tak, že přijímají funkce s jednou proměnnou. Avšak funkce *CostFunction*, která obsahuje výpočet kriteriálních funkcí a měla by tedy být optimalizována funkcí *optimize*, využívá 3 proměnné. Bylo proto nutné vytvořit funkci *Initialize*, která je použita ve funkci *optimize* namísto *CostFunction*. Vstupní proměnnou funkce *Initialize* je vektor *var* o velikosti 3**n*. Tato funkce přepočítá vektor *var* na vektory *x*, *y* a *z*, kde prvních *n* složek odpovídá vektoru *x*, druhých *n* složek vektoru y a třetích *n* složek vektoru *z*. Následně je zavolána funkce *CostFunction* s přepočítanými vektory *x*,*y* a *z* jako vstupními proměnnými.

5.5 Grafy

Grafy byly vytvořeny pomocí balíčku Plots.jl, který umožňuje vizualizovat výsledná data optimalizace.

Balíček ale neobsahuje funkci, která by vykreslila kruh s daným průměrem. Překážky modelu tedy musely být zakresleny jako body s velikostí odpovídající velikosti překážek v modelu. Velikost bodů je však dána v pixelech. Tudíž, aby se velikost překážek v grafu rovnala jejich velikosti v modelu, je nutné, aby šířka a výška grafu v pixelech byla stejná, jako je šířka a výška mapy, se kterou pracuje model.

Aplikace vytvořená pro tuto práci generuje po optimalizaci tři grafy. První dva vykreslují trasy, které jsou výsledkem optimalizace. Zatímco první vykresluje překážky a trasy v osách x a y, druhý vykresluje výšku bodů trasy neboli hodnoty osy z. Třetí graf ukazuje, jak algoritmy konvergovaly k výsledné hodnotě v průběhu optimalizace.

6 VÝSLEDKY

V této kapitole jsou popsány výsledky testů a simulací provedených s využitím aplikace, jež byla vytvořena v praktické části.

6.1 Analýza citlivosti

Před numerickými simulacemi byla nejprve provedena analýza citlivosti, aby se určily vstupní parametry algoritmu, které povedou k co nejlepším výsledkům optimalizace za co nejkratší čas.

Analýza citlivosti se téměř vždy provádí tak, že se model několikrát spustí s různými parametry [25]. Následně se na základě výsledků těchto testovacích spuštění hodnoty parametrů upravují tak, aby bylo dosaženo určitého kritéria, tedy v našem případě co nejmenší hodnoty nákladové funkce za určitý čas.

Tímto způsobem byla analýza provedena i při tvorbě této práce. Pro každý algoritmus byl model několikrát optimalizován za daný čas 100 sekund. Počet bodů *n* byl 10. Aby se eliminovala náhodná složka algoritmů, byl jim nastaven daný „seed“, tudíž akce, které by algoritmus obvykle prováděl náhodně, byly při několika různých spuštěních provedeny stejně.

Takto získané parametry byly nastaveny jako základní, tedy ty, které aplikace použije při prvním spuštění. Parametry jsou uvedeny v tab. 3.

Parametr algoritmu	Použitá hodnota
ABC – N	240
ABC – limit	40
PSO – N	500
PSO – C1	1.7
PSO – C2	2.0
PSO – ω	0.8
WOA – N	250

Tab. 3: Použité parametry algoritmů

Parametry získané analýzou senzitivity však jsou vhodné primárně pro použití na počítači, na kterém byla vykonána analýza. Na počítači s jinými hardwarovými specifikacemi mohou být vhodnější jiné parametry. Testy proběhly na osobním počítači s následujícími hardwarovými specifikacemi:

- Procesor: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
- Grafická karta: NVIDIA GeForce GTX 1060 6 GB
- Operační paměť: 16 GB RAM

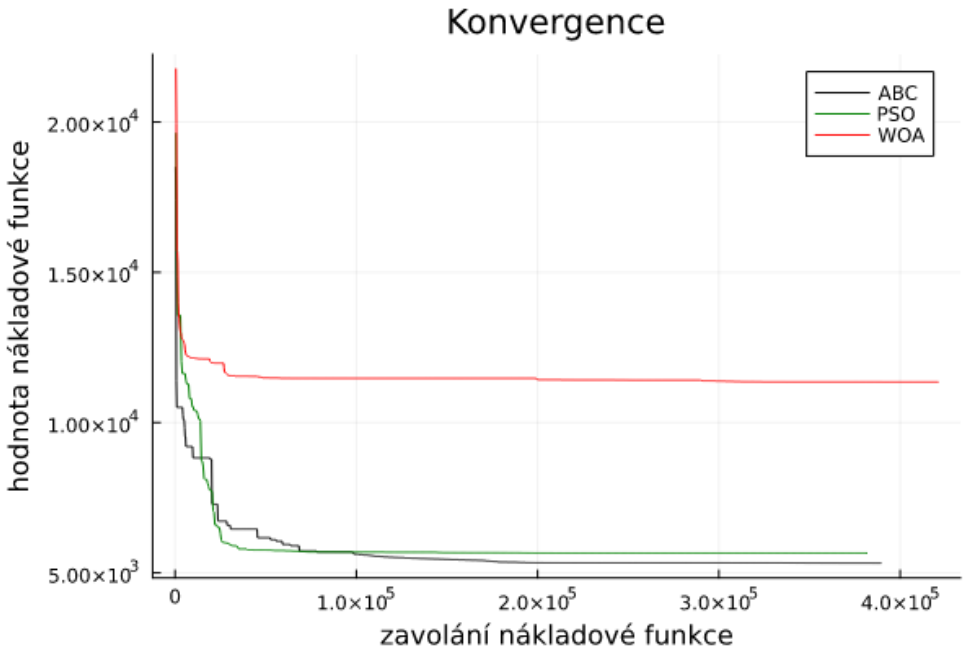
6.2 Numerické simulace

Pro kontrolu správného fungování aplikace a zjištění vlastností použitých algoritmů byla provedena řada simulací konkrétních tras bezpilotního letounu.

6.2.1 Simulace s rozdílným časem

První sada simulací byla provedena, aby bylo zjištěno, jaký má dopad množství času na chování algoritmů.

Při simulaci s počtem bodů $n = 5$ byly nejrychleji schopny najít použitelnou trasu algoritmy ABC a PSO, přičemž trasa nalezená algoritmem ABC měla nižší hodnotu nákladové funkce, jak je patrné z první simulace, viz obr. 6. Při této simulaci měly algoritmy na optimalizaci 60 sekund.



Obr. 6: Konvergence algoritmů při první simulaci

Rozdíly v hodnotách nákladové funkce u tras, které vznikly optimalizací v řádu desítek sekund, nebyly značné oproti trasám, které vznikly během desítek minut, jak je patrné z tabulky 3.

	Minimum (60 s)	*Počet iterací (60 s)*	*Minimum (300 s)*	*Počet iterací (300 s)*	*Minimum (900 s)*	*Počet iterací (900 s)*
ABC	5325.06	1 613	5111.83	7 702	5327.18	27 910
PSO	5660.16	763	5178.71	3 472	5113.20	12 394
WOA	11350.40	1 683	4894.57	8 148	5850.00	25 912

Tab. 4: Hodnoty optimalizací s různými časy zastavení ($n = 5$)

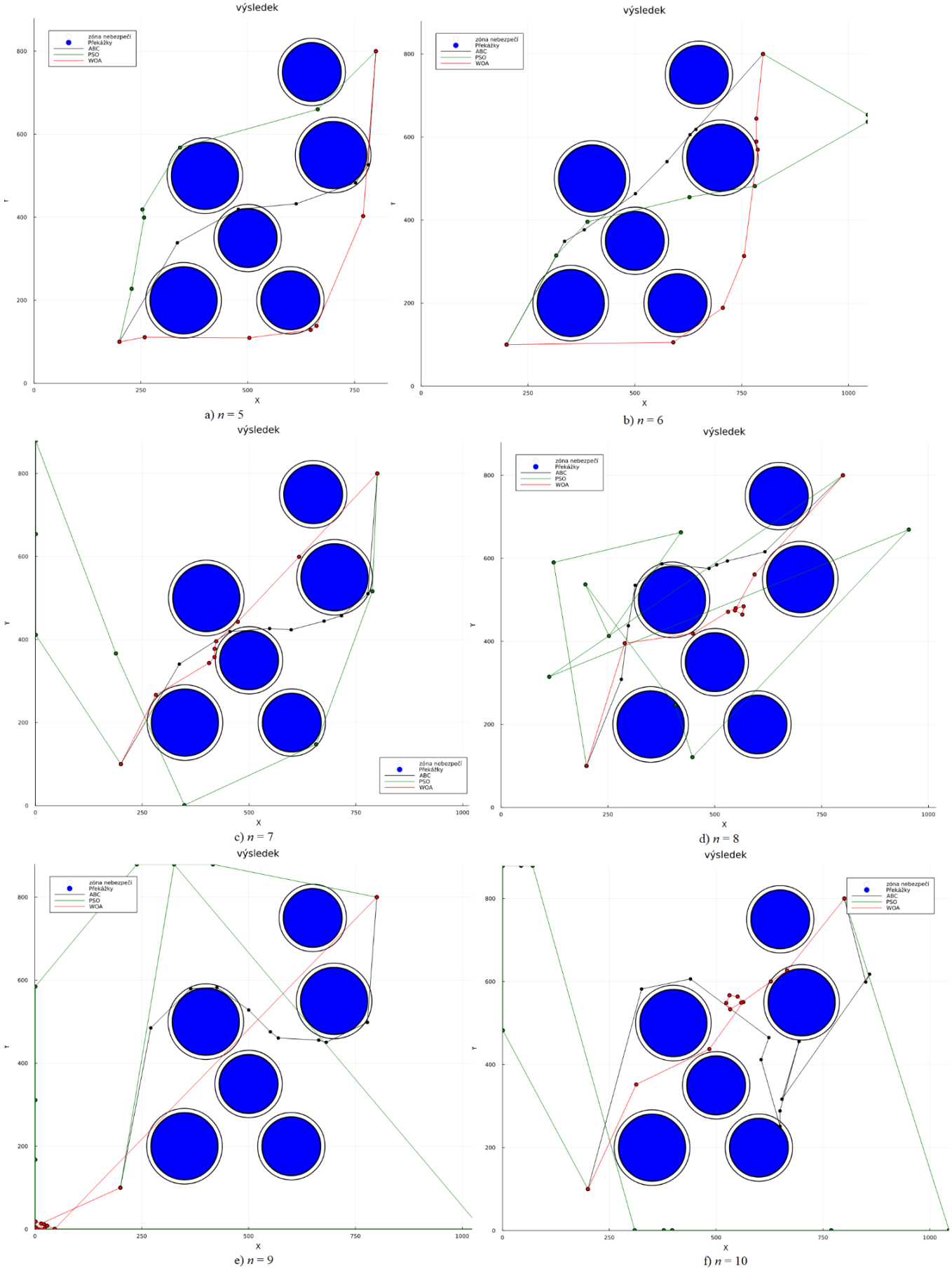
6.2.2 Simulace s rozdílným počtem bodů *n*

Druhá sada simulací byla vytvořena, aby byl zjištěn vliv počtu bodů trasy *n* na výkon optimalizačních algoritmů. V této sadě měl každý algoritmus na optimalizaci 15 minut a s každou simulací byl zvýšen počet bodů *n* o jedno. S rostoucím počtem bodů *n* rychle roste i výpočetní náročnost. Předpokladem tedy je, že se při růstu *n* bude zhoršovat kvalita tras. To je i dobře patrné na obr. 7.

Při optimalizacích s počtem bodů trasy $n = 5$ nebyly mezi algoritmy drastické rozdíly, to však neplatilo při vyšším počtu bodů. Již při $n = 6$ začínají některé body trasy optimalizované algoritmem PSO „ujíždět“ k rohům mapy. Tento efekt se při vyšších hodnotách *n* pouze umocňuje, kdy algoritmus umístí několik bodů trasy až na samotné okraje mapy. Již od hodnoty $n = 7$ jsou trasy, vytvořené tímto algoritmem, nepoužitelné.

Algoritmus WOA byl schopný tvořit relativně smysluplné trasy až do náročnosti $n = 8$. Při této hodnotě se u tras tohoto algoritmu začalo projevovat spirálovité hledání optima, v jehož důsledku se body trasy kupí na jedno místo.

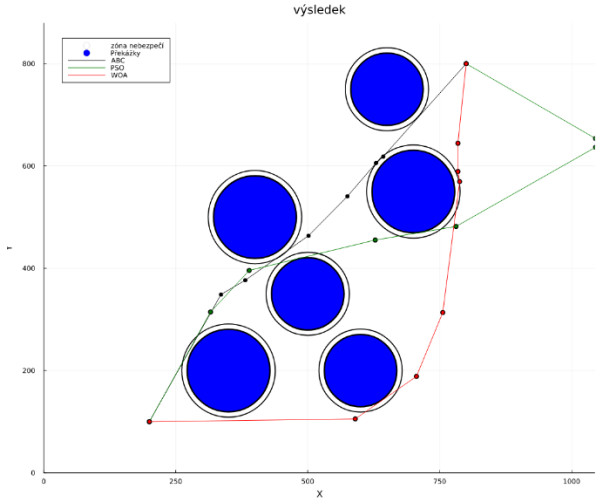
Jako nejvíce konzistentní se projevil algoritmus ABC. Ten byl schopný tvořit relativně kvalitní trasy až do $n = 10$, kdy již nestíhal v časovém limitu vyhladit nalezené trasy. V těch se proto začaly tvořit ostré zatáčky, které by v reálném prostředí byly těžko proveditelné.



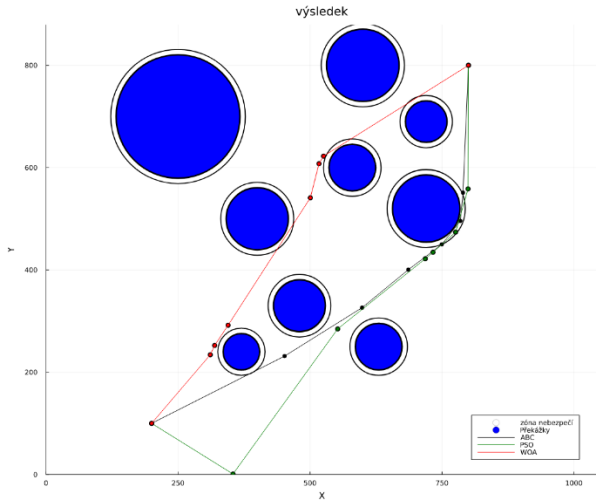
Obr. 7: Ukázka tras při různém počtu bodů n

6.2.3 Simulace s různým rozmístěním překážek

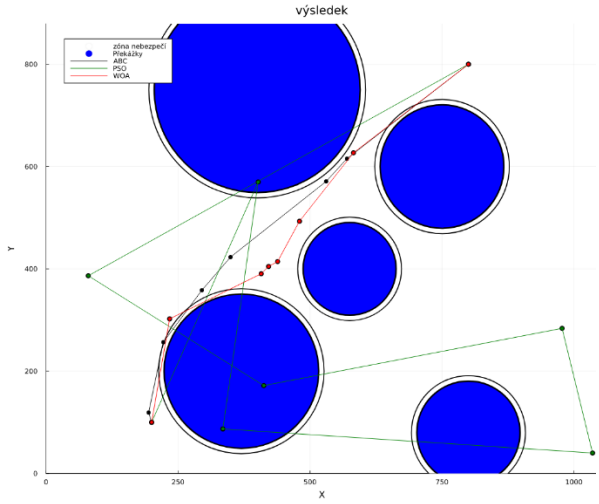
V třetí sadě byly v modelu použity různé sestavy překážek ve snaze simulovat rozdílná prostředí. Pro tyto simulace byl zvolen počet bodů trasy $n = 6$ a čas, který měly algoritmy na optimalizaci, byl nastaven na 15 minut.



a) 1. trasa



b) 2. trasa



c) 3. trasa

Obr. 8: Optimalizace s třemi různými rozmístěními překážek

Jak je patrné z obr. 8, změna rozmístění překážek nijak neovlivnila schopnost algoritmů ABC a WOA najít vhodná řešení. Stejně tak problémy, které měl algoritmus PSO při nastavení $n = 6$, také přetrvávají. Možným řešením tohoto problému je úprava algoritmu PSO, aby byl vhodnější pro tento typ úlohy. Tím by mohla být například deterministická inicializace a zavedení náhodných mutací [26] nebo využití paralelního programování [27]. Náhodné vzorkování a mutace jsou způsoby, jakými lze vylepšit také algoritmus WOA [28].

Avšak i přes tato potenciální zlepšení se program při přiměřeném nastavení parametrů optimalizace projevil jako plně funkční a schopný najít vhodné trasy za krátkou dobu.

Celkově se jako nejvíce konzistentní, a tím pádem nejvíce vhodným pro optimalizaci tohoto modelu, jeví algoritmus ABC. Ten byl schopný najít kvalitní řešení ve všech sadách simulací. Díky schopnosti rychlé konvergence k řešení byl schopný najít řešení v krátkém časovém intervalu v první sadě simulací, zároveň jej ale nejméně ovlivnila stoupající výpočetní náročnost optimalizace v třetí sadě simulací.

Druhým nejvhodnějším algoritmem byl WOA. Při nízkém počtu bodů trasy n sice konvergoval pomaleji než ABC a PSO, zato byl v průměru schopný najít lepší řešení než ostatní dva algoritmy. Při stoupající náročnosti ztratil schopnost najít použitelné řešení v daném časovém intervalu později než PSO, avšak dříve než ABC.

Nejméně vhodným algoritmem byl v simulacích PSO. Při nízkém počtu bodů trasy n byl schopný najít řešení v podobném čase jako ABC, dokonce často i rychleji. Toto nalezené řešení však většinou mělo o trochu větší hodnotu nákladové funkce než řešení ostatních algoritmů. Jeho hlavním neduhem však je, že jako první ztrácí schopnost najít použitelné dráhy letounu při zvyšujícím se počtu bodů n.

7 ZÁVĚR

Tato práce se zabývala problémem optimalizace tras bezpilotních letounů. V jejím rámci byl vytvořen nástroj, který po zadání vstupních parametrů pomocí uživatelského rozhraní navrhne a optimalizuje tři trasy, každou jiným optimalizačním algoritmem. Těmi jsou algoritmy Artificial Bee Colony, Particle Swarm Optimalization a Whale Optimalization Algorithm.

Nejprve je stručně popsán úvod do problematiky optimalizace a rozdělení optimalizačních úloh dle několika kritérií.

Následuje popis použitého optimalizačního modelu a jeho vlastností. Po úvodu do jeho fungování jsou přiblíženy kriteriální funkce, které model obsahuje, a jejich úloha v procesu optimalizace.

Kapitola 4 obsahuje teoretické základy metaheuristických algoritmů a jejich rozdělení. Dále jsou zde popsány tři evoluční algoritmy, které byly použity v praktické části, jejich fungování, vlastnosti a nutné vstupní parametry.

V další části se nachází obeznámení se softwarovou implementací, která byla provedena pomocí programovacího jazyka Julia, jejích balíčků a programovacího prostředí Pluto.jl. Kapitola obsahuje obeznámení s uživatelským rozhraním a řešením jednotlivých částí programu.

Pro kontrolu vlastností programu a jeho prezentaci byla provedena analýza citlivosti a několik simulací s různými vstupními parametry. Na základě těchto simulací se z použitých algoritmů jako nejvhodnější projevil algortimus Artificial Bee Colony.

Program je plně funkční, avšak jeho výkonnost by mohla být vylepšena úpravami využitých algoritmů.

SEZNAM POUŽITÉ LITERATURY

[1] WHEELER, Tim A. Algorithms for optimization. MIT Press. Cambridge, Massachusetts: The MIT Press, [2019]. ISBN 978-0262039420.

[2] WILLIAMS, P. H. Model Building in Mathematical Programming. Wiley, 2013.

[3] TUY, H. Convex Analysis and Global Optimization. Springer, 2015.

[4] BOYD, S. a VANDERBERGHE, L. Convex Optimization. Cambridge: Cambridge University Press, 2004.

[5] COELLO COELLO, Carlos A., LAMONT a David A. Van VELDHUIZEN. Evolutionary Algorithms for Solving Multi-Objective Problems. 2007. Dostupné z: doi:10.1007/978-0-387-36797-2

[6] PHUNG, Manh Duong a Quang Phuc HA. Safety-enhanced UAV path planning with spherical vector-based particle swarm optimization. Applied Soft Computing. 2021, 107. ISSN 15684946. Dostupné z: doi:10.1016/j.asoc.2021.107376

[7] KUDELA, Jakub. A critical problem in benchmarking and analysis of evolutionary computation methods. Nature Machine Intelligence. 2022, 4(12), 1238-1245. ISSN 2522-5839. Dostupné z: doi:10.1038/s42256-022-00579-0

[8] ŠANDERA, Č. Hybridní model metaheuristických algoritmů. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 154s. Vedoucí diplomové práce prof. RNDr. Ing. Miloš Šeda, Ph.D.

[9] BLUM, Christian a Andrea ROLI. Metaheuristics in combinatorial optimization. ACM Computing Surveys. 2003, 35(3), 268-308. ISSN 0360-0300. Dostupné z: doi:10.1145/937503.937505

[10] Talbi, E.-G.: Metaheuristics: From Design to Implementation. Wiley, Hoboken, New Jersey, 2009, ISBN 978-0-470-27858-1

[11] VIKHAR, Pradnya A. Evolutionary algorithms: A critical review and its future prospects. 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC). IEEE, 2016, 2016, 261-265. ISBN 978-1-5090-0467-6. Dostupné z: doi:10.1109/ICGTSPICC.2016.7955308

[12] SONG, Pei-Cheng, Jeng-Shyang PAN a Shu-Chuan CHU. A parallel compact cuckoo search algorithm for three-dimensional path planning. Applied Soft Computing. 2020, 94. ISSN 15684946. Dostupné z: doi:10.1016/j.asoc.2020.106443

[13] FU, Yangguang, Mingyue DING, Chengping ZHOU a Hanping HU. Route Planning for Unmanned Aerial Vehicle (UAV) on the Sea Using Hybrid Differential Evolution and Quantum-Behaved Particle Swarm Optimization. IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2013, 43(6), 1451-1465. ISSN 2168-2216. Dostupné z: doi:10.1109/TSMC.2013.2248146

[14] YU, Xue, Wei-Neng CHEN, Tianlong GU, Huaqiang YUAN, Huaxiang ZHANG a Jun ZHANG. ACO-A*: Ant Colony Optimization Plus A* for 3-D Traveling in Environments With Dense Obstacles. IEEE Transactions on Evolutionary Computation. 2019, 23(4), 617-631. ISSN 1089-778X. Dostupné z: doi:10.1109/TEVC.2018.2878221

[15] KARABOGA, Dervis a Bahriye BASTURK. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization. 2007, 39(3), 459-471. ISSN 0925-5001. Dostupné z: doi:10.1007/s10898-007-9149-x

[16] KENNEDY, J. a R. EBERHART. Particle swarm optimization. Proceedings of ICNN'95 - International Conference on Neural Networks. IEEE, 1995, 1942-1948. ISBN 0-7803-2768-3. Dostupné z: doi:10.1109/ICNN.1995.488968

[17] MITTAŠ, Eduard. Benchmarking pro hejnové optimalizační algoritmy. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2022, 115 s. Diplomová práce. Vedoucí práce: Ing. Jakub Kůdela, Ph.D.

[18] Poli, R., Kennedy, J. & Blackwell, T. Particle swarm optimization. Swarm Intell 1, 33–57 (2007). https://doi.org/10.1007/s11721-007-0002-0

[19] SHI, Y. a R. EBERHART. A modified particle swarm optimizer. 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360). IEEE, 1998, 69-73. ISBN 0-7803-4869-9. Dostupné z: doi:10.1109/ICEC.1998.699146

[20] ZHANG, Yudong, Shuihua WANG a Genlin JI. A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. Mathematical Problems in Engineering. 2015, 2015, 1-38. ISSN 1024-123X. Dostupné z: doi:10.1155/2015/931256

[21] LAI, Xiangjing, Jin-Kao HAO, Zhang-Hua FU a Dong YUE. Diversity-preserving quantum particle swarm optimization for the multidimensional knapsack problem. Expert Systems with Applications. 2020, 149. ISSN 09574174. Dostupné z: doi:10.1016/j.eswa.2020.113310

[22] MIRJALILI, Seyedali a Andrew LEWIS. The Whale Optimization Algorithm. Advances in Engineering Software. 2016, 95, 51-67. ISSN 09659978. Dostupné z: doi:10.1016/j.advengsoft.2016.01.008

[23] Speed comparison of programming languages. Github [online]. [cit. 2023-05-21]. Dostupné z: https://github.com/niklas-heer/speed-comparison

[24] Progressing from MATLAB to Julia. Datacamp [online]. [cit. 2023-05-21]. Dostupné z: https://www.datacamp.com/blog/progressing-from-matlab-to-julia

[25] HELTON, J.C., J.D. JOHNSON, C.J. SALLABERRY a C.B. STORLIE. Survey of sampling-based methods for uncertainty and sensitivity analysis. 2006, 91(10-11), 1175-1209. ISSN 09518320. Dostupné z: doi:10.1016/j.ress.2005.11.017

[26] PHUNG, Manh Duong, Cong Hoang QUACH, Tran Hiep DINH a Quang HA. Enhanced discrete particle swarm optimization path planning for UAV vision-based surface inspection. Automation in Construction. 2017, 81, 25-33. ISSN 09265805. Dostupné z: doi:10.1016/j.autcon.2017.04.013

[27] ROBERGE, Vincent, Mohammed TARBOUCHI a Gilles LABONTE. Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning. IEEE Transactions on Industrial Informatics. 2013, 9(1), 132-141. ISSN 1551-3203. Dostupné z: doi:10.1109/TII.2012.2198665

[28] DAI, Yaonan, Jiuyang YU, Cong ZHANG, Bowen ZHAN a Xiaotao ZHENG. A novel whale optimization algorithm of path planning strategy for mobile robots. Applied Intelligence. 2023, 53(9), 10843-10857. ISSN 0924-669X. Dostupné z: doi:10.1007/s10489-022-04030-0

SEZNAM ZKRATEK

ABC	Artificial Bee Colony
PSO	Particle Swarm Optimization
WOA	Whale Optimization Algorithm

SEZNAM PŘÍLOH

1: přiložený archiv 2023_BP_Milosevic_Filip_229487.zip
    Zdrojový kód aplikace
    Soubor ChrismasTerrain.tif s daty o terénu optimalizované plochy