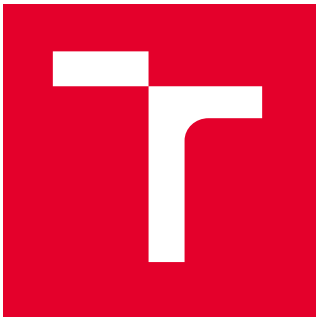


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

OPTIMALIZACE ROBOTICKÝCH TRAJEKTORIÍ POMOCÍ MODULÁRNÍCH EVOLUČNÍCH ALGORITMŮ

ROBOT TRAJECTORY OPTIMIZATION WITH MODULAR EVOLUTIONARY ALGORITHMS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jan Fiala

AUTHOR

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

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Jan Fiala**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Jakub Kůdela, Ph.D.**
Akademický rok:	2024/25

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:

Optimalizace robotických trajektorií pomocí modulárních evolučních algoritmů

Stručná charakteristika problematiky úkolu:

Optimalizace robotických trajektorií pomocí evolučních algoritmů je aktuálním a široce zkoumaným tématem. V posledních letech výrazně vzrostl zájem o automatický návrh vysoce výkonných evolučních algoritmů. Tyto přístupy používají dva základní prvky. Prvním je modulární framework pro návrh evolučního algoritmu a druhým pak vhodný nástroj pro automatickou konfiguraci tohoto algoritmu. Diplomová práce se bude zabývat popisem, implementačními detaily a aplikací modulárních frameworků pro návrh evolučních algoritmů jako např. Particle Swarm Optimization nebo Differential Evolution pro optimalizaci robotických trajektorií.

Cíle diplomové práce:

Rešerše přístupů k optimalizaci robotických trajektorií.
Rešerše modulárních frameworků pro návrh evolučních algoritmů.
Implementace vybraných modulárních frameworků ve spojení s nástrojem pro jejich automatickou konfiguraci pro optimalizaci robotických trajektorií.
Srovnání vhodnosti navržených přístupů.

Seznam doporučené literatury:

Juříček, Martin, Roman Parák, and Jakub Kůdela. "Evolutionary computation techniques for path planning problems in industrial robotics: A state-of-the-art review." Computation 11, no. 12 (2023): 245.

Camacho-Villalón, Christian L., Thomas Stützle, and Marco Dorigo. "Designing new metaheuristics: manual versus automatic approaches." Intelligent Computing 2 (2023): 0048.

Kůdela, Jakub, Martin Juříček, and Roman Parák. "A collection of robotics problems for benchmarking evolutionary computation methods." In International Conference on the Applications of Evolutionary Computation (Part of EvoStar), pp. 364-379. Cham: Springer Nature Switzerland, 2023.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M. and Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives, 3, pp.43-58.

Camacho-Villalón, Christian L., Marco Dorigo, and Thomas Stützle. "PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms." IEEE Transactions on Evolutionary Computation 26, no. 3 (2021): 402-416.

LaTorre, Antonio, Daniel Molina, Eneko Osaba, Javier Poyatos, Javier Del Ser, and Francisco Herrera. "A prescription of methodological guidelines for comparing bio-inspired optimization algorithms." Swarm and Evolutionary Computation 67 (2021): 100973.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2024/25

V Brně, dne

L. S.

doc. Ing. Zdeněk Hadaš, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Tato diplomová práce se zabývá návrhem a implementací modulárního frameworku pro optimalizaci trajektorií robotických systémů v prostředí s překážkami. Navržený systém kombinuje sampling-based plánovací algoritmus, variantu Rapidly-exploring Random Tree, s modulární verzí diferenciální evoluce. Díky integraci s frameworkem Irace umožňuje automatické ladění parametrů evolučních algoritmů bez potřeby manuálních zásahů. Výpočetní proces je realizován paralelně v prostředí Kubernetes, což zajišťuje škálovatelnost na vícejádrové i cloudové architektury. Součástí práce je knihovna testovacích scén, otevřená implementace v jazyce Python a webové rozhraní pro vizualizaci a analýzu výsledků. Závěrem je provedena srovnávací studie výkonnosti různých konfigurací algoritmu, která ověřuje univerzálnost navrženého přístupu.

ABSTRACT

This thesis focuses on the design and implementation of a modular framework for trajectory optimization of robotic systems in obstacle-filled environments. The proposed system combines sampling-based planning algorithm, Rapidly-exploring Random Tree variant, with a modular version of Differential Evolution. Through integration with the Irace framework, the system enables automated parameter tuning without manual intervention. The computational process is executed in parallel within a Kubernetes environment, ensuring scalability across multi-core and cloud architectures. The work includes a library of test scenarios, an open-source Python implementation, and a web-based interface for trajectory visualization and analysis. Finally, a comparative study evaluates the performance of different algorithm configurations, demonstrating the robustness and general applicability of the proposed approach.

KLÍČOVÁ SLOVA

robotické plánování trajektorií, evoluční algoritmy, modulární algoritmy, autotuning, kontejnerizace, paralerizace

KEYWORDS

robotic trajectory planning, evolutionary algorithms, modular algorithms, autotuning, containerization, parallelization

ÚSTAV AUTOMATIZACE A INFORMATIKY

2025

BIBLIOGRAFICKÁ CITACE

FIALA, Jan. *Optimalizace robotických trajektorií pomocí modulárních evolučních algoritmů* . Brno, 2025. Dostupné také z: `https://www.vut.cz/studenti/zav-prace/detail/165706`. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, Vedoucí práce: doc. Ing. Jakub Kůdela, Ph.D.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato diplomová práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího diplomové 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 následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 21. 5. 2025 .....

Jan Fiala

PODĚKOVÁNÍ

Na tomto místě bych rád poděkoval všem, kteří mě během psaní této diplomové práce podporovali a stáli při mně. Velké díky patří především těm, kteří mi poskytovali cenné odborné rady, připomínky a konzultace, jež mi pomohly lépe uchopit zvolenou problematiku a posunout práci vpřed. Dále děkuji své rodině a blízkým za trpělivost, motivaci a psychickou podporu, která byla pro mě po celou dobu neocenitelná.

OBSAH

1 ÚVOD

Úspěšnost robotického systému je často posuzována podle schopnosti bezpečně, efektivně a energeticky optimálně realizovat plánované trajektorie v reálném prostředí, které je charakteristické přítomností nejistot a variabilních podmínek. Klasické deterministické algoritmy pro plánování trajektorií umožňují nalézt kolizně bezpečnou cestu v konfiguracích prostředí, avšak zpravidla nedokážou zajistit splnění všech praktických požadavků současně. Typicky jde o plynulost výsledné trajektorie, optimalitu podle více kritérií, nebo schopnost adaptace v reálném čase bez nutnosti opakovaného ručního ladění vstupních parametrů.

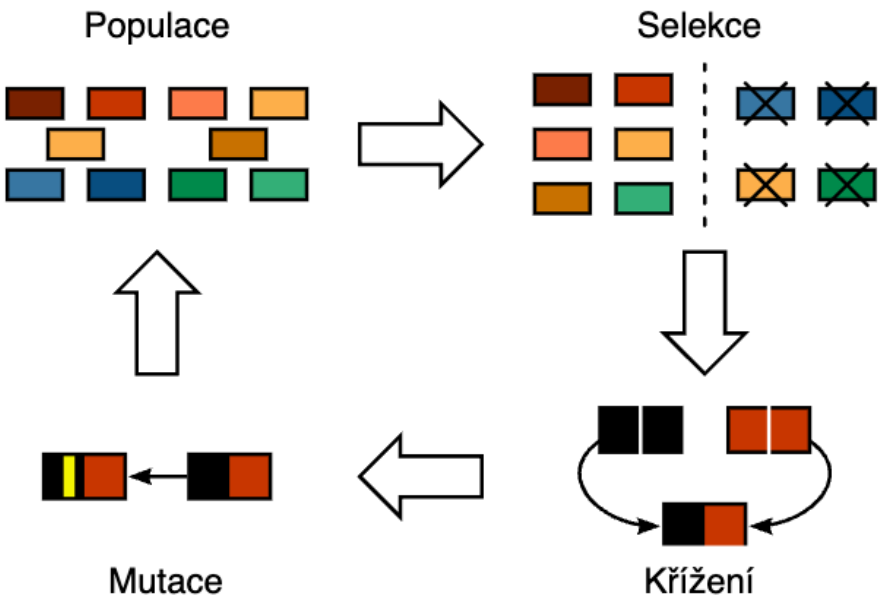
Alternativní přístup představují evoluční algoritmy (EA), které jsou založeny na paralelním a globálním průzkumu prostoru řešení, přičemž k optimalizaci nevyžadují explicitní znalost gradientů cílové funkce. Nevýhodou EA je však jejich výrazná citlivost na konfiguraci a vysoká výpočetní náročnost, která obvykle vede k potřebě značného manuálního ladění parametrů ze strany uživatele.

V posledních letech byla proto navržena řada modulárních evolučních frameworků, které jsou navrženy tak, že jednotlivé komponenty evolučního procesu – jako jsou operátory mutace, křížení, selekční strategie či adaptivní mechanismy – vystupují jako samostatně zaměnitelné moduly. Příkladem nástroje, který umožňuje automatizované nastavení těchto modulů, je framework Irace, jenž je schopen systematicky prohledávat prostor možných konfigurací algoritmu a nalézt nejvhodnější nastavení pro konkrétní typ úloh.

V rámci této diplomové práce je proto navržen, implementován a experimentálně ověřen plně modulární výpočetní framework pro optimalizaci robotických trajektorií. Vytvořený systém umožňuje automatické nastavení parametrů EA prostřednictvím zmíněného frameworku Irace, přičemž samotný výpočetní proces je realizován paralelně v prostředí Kubernetes. Toto řešení je schopné škálovat svůj výkon na vícejádrové architektury i cloudové clustery, a tedy umožňuje provádět rozsáhlé paralelní výpočty při zachování vysoké efektivity. Konkrétními cíli této práce je vytvoření knihovny robotických scén, které umožňují reprodukovatelná experimentální měření a porovnávání výsledků mezi různými konfiguracemi a implementacemi. Dále je cílem poskytnout otevřenou a dobře dokumentovanou implementaci modulární diferenciální evoluce v programovacím jazyce Python, připravit kontejnerizovaný testovací nástroj doplněný o webové aplikační rozhraní pro přehlednou vizualizaci a analýzu plánovaných trajektorií. Závěrem je srovnávací studie, která analyzuje výkonnost konfigurací evolučního algoritmu, a to s cílem ověřit robustnost a univerzálnost navrženého frameworku v praktických úlohách robotického plánování.

2 Evoluční algoritmy

EA představují optimalizační metody inspirované přírodní evolucí. Pracují s populací potenciálních řešení, která jsou iterativně vylepšována pomocí procesů obdobných biologické selekci, křížení a mutaci. Jednotlivá řešení každé generace jsou hodnocena pomocí tzv. fitness funkce . Ta odráží kvalitu řešení daného problému. Evoluční algoritmy tak spadají mezi globální heuristické metody optimalizace. Díky zabudované náhodnosti dokáže prohledávat složité vícerozměrné prostory stavů a čelit uvíznutí v lokálních extrémech lépe než konvenční gradientní metody [1].



Obr. 1: Základní pravidla evolučních algorimů [2].

Princip EA je založen na společných pravidlech znázorněných na obr. 1. Na počátku algoritmu je inicializovaná počáteční populace kandidátních řešení. Tato inicializace může být buď náhodná, nebo předvolená na začátku algoritmu. Následně algoritmus iterativně vytváří novou generaci. Dle hodnoty fitness vybírá nejlepšího jedince, metodou selekce. Následují metody křížení a mutace. Křížení kombinuje části dvou nebo více rodičů. Mutace provádí náhodné změny v řešení, variabilitu. Tímto způsobem je vybrán potomek. Potomci následně zcela nebo částečně nahrazují původní populaci a proces se opakuje. Ukončení algoritmu může být dosaženo pomocí několika kritérii. Vyčerpání omezeného počtu volání ohodnocující funkce pro validaci jedinců, dosažením dostatečně dobrého řešení, kde se výsledek nemění v určitém časovém úseku [3, 4].

Specifickou inspirací v přírodě bylo stanoveno několik hlavních směrů evolučních algoritmů. Odlišují se reprezentací řešení a detaily evolučních operátorů.

2.1 Genetické algoritmy

Genetické algoritmy (GA) představují patrně nejrozšířenější evoluční přístup, protože spojují mimořádnou flexibilitu reprezentace se snadnou implementací. Náhodná inicializace populace znamená, že počáteční množina řešení je generována náhodně, bez předchozí znalosti problému. Každý jedinec v populaci představuje možné řešení daného problému a je tvořen vhodnou reprezentací (například binárním řetězcem nebo reálnými čísly). GA využívá několik základních operací pro selekci, křížení a mutaci jedinců:

- Turnajová selekce vybírá jedince do další generace tak, že náhodně zvolí několik jedinců z populace a následně vybere toho nejlepšího podle jejich hodnoty fitness. Tento postup se opakuje, dokud není dosaženo požadovaného počtu jedinců.
- Ruletová selekce vybírá jedince s pravděpodobností úměrnou jejich fitness. Jedinci s vyšší hodnotou fitness mají větší šanci být vybráni.
- Křížení (crossover) kombinuje části dvou rodičovských jedinců, čímž vznikají noví potomci.
- Uniformní křížení generuje každou složku potomka náhodně od jednoho z rodičů, obvykle s pravděpodobností 0,5.
- Simulované binární křížení je metoda vhodná pro jedince s reálnou reprezentací, kdy se noví potomci generují na základě pravděpodobnostního rozdělení, které napodobuje princip binárního křížení v reálném prostoru.
- Mutace mění jednotlivé složky jedinců s určitou pravděpodobností, čímž pomáhá udržet diverzitu v populaci.
- Bit-flip mutace mění hodnotu jednotlivých bitů v binárním řetězci (z 0 na 1 nebo z 1 na 0).
- Gaussovská mutace přidává k hodnotám v reálné reprezentaci náhodné číslo vybrané z Gaussova (normálního) rozdělení.

Po roce 2010 byl GA zásadně rozšířen o mechanismy, které se samy učí vhodná nastavení parametrů. V adaptivní verzi GA se pravděpodobnosti křížení a mutace enkódují přímo do chromozomu, takže evoluce sama tlakem na fitness zvýhodňuje kombinace, jež vedou k rychlejší konvergenci [3]. Další možností adaptace algoritmu jsou metody, v nichž se jedinec ukládá jako vektor reálných čísel a rekombinace probíhá pouhou výměnou klíčů, což redukuje riziko neplatných potomků a výrazně zrychluje práci v kombinatorických úlohách [5]. Využívany jsou také multi-objektivní verze (NSGA-II/III, R-NSGA-II), které prostřednictvím nedominovaného třídění a ohodnocovací metriky konstruují rovnoměrné aproximace Pareto front a uplatňují se v energeticky úsporném návrhu budov, syntéze architektur hlubokých sítí či optimalizaci fotovoltaických polí [6]. Tam, kde je jedna evaluace extrémně nákladná, se GA kombinuje s metamodely (kriging, GPR), jež predikují fitness a umožňují vyšetřit řádově více kandidátů se stejným rozpočtem [7].

2.2 Particle Swarm Optimization (PSO)

Particle Swarm Optimization je optimalizační algoritmus, který simuluje kolektivní chování hejna, například ptáků nebo ryb. Každá částice v algoritmu představuje jedno možné řešení daného problému a její pohyb je řízen kombinací setrvačnosti, osobní zkušenosti a sociální přitažlivosti. Setrvačnost způsobuje, že částice pokračuje částečně ve svém původním směru, zatímco osobní zkušenost vede částici zpět k její dosud nejlepší nalezené pozici. Sociální přitažlivost ji zároveň táhne směrem k nejlepší pozici nalezené celým hejnem. V poslední dekádě se výzkum soustředil především na adaptivní řízení těchto sil, tedy dynamické přizpůsobování jejich intenzity během optimalizace. Pozornost byla věnována rovněž multi-swarmovému přístupu, při němž více menších hejn částic spolupracuje a koordinuje své chování za účelem efektivnějšího prohledávání prostoru řešení. Adaptivní varianty dynamicky mění inerciální váhu a akcelerační koeficienty podle fáze hledání nebo odhadované drsnosti krajiny, čímž výrazně snižují riziko předčasné konvergence [8]. Hybridní strategie, jako NDW-PSO, kombinují stochastický (náhodný) průzkum hejna s lokálním gradientním doladěním a prokazují rychlejší dosažení kvalitního optima například v optimalizaci hlubokých konvolučních sítí [9]. Souběžně se rozvinuly multi-objective verze (MOPSO), které využívají Pareto dominance nebo dekompoziční techniky k rovnoměrnému pokrytí pareto fronty a nacházejí uplatnění v energetice, logistice i robustním řízení [10]. Rozsáhlý přehled trendů, variant a aplikací z roku 2023 potvrzuje, že moderní adaptivní a ensemble PSO zůstávají výrazně konkurenceschopné vůči DE i novým swarmovým algoritmům [11].

2.3 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) patří k nejúčinnějším metodám pro spojitou black-box optimalizaci, protože průběžně tvaruje gaussovskou mutační distribuci podle empirické kovariace elitních vzorků. Moderní implementace využívají rychlé aktualizace rank-one + rank-μ, restarty s rostoucí velikostí populace a adaptivní strategii, která se sama učí parametry, například kroky učení a délku evolučních cest. Pro úlohy s vysokou dimenzionalitou byly navrženy LDLT-faktorizované a lineární LM-CMA varianty, jež snižují výpočetní náročnost na $O(n \log n)$. V přítomnosti šumu a při drahých simulacích se CMA-ES kombinuje se znovu vyhodnocováním a metamodelem, zatímco u omezených problémů se uplatňuje augmented Lagrangian nebo restart s penalizacemi. Diskrétní a smíšené proměnné řeší rozšíření na biased random keys či optimalizaci po bodech. Také multi-objective MO-CMA-ES s dekompozicí nebo dominance rankingem poskytuje robustní aproximace Pareto front v návrhu robotických trajektorií a kalibraci hydrologických modelů [12, 13, 14].

2.4 Diferenciální evoluce

Diferenciální evoluce (DE) řeší spojité a multimodální optimalizační úlohy pomocí jednoduché operace vektorového rozdílu. V každé iteraci se pro cílový vektor zvolí tři různí členi populace. K prvnímu z nich se přičte škálovaný rozdíl dalších dvou, čímž vznikne nový vektor. Ten projde binomickým křížením s původním rodičem a pokud vykáže nižší hodnotu cílové funkce, nahradí jej v populaci. Díky této kombinaci mutace, křížení a greedy selekce je algoritmus snadno implementovatelný, nezávislý na derivacích a přirozeně paralelizovatelný [4, 15]. Matematicky lze hlavní operace DE vyjádřit následujícími vztahy:

$$\mathbf{v}_{i,G} = \mathbf{x}_{r_1,G} + F(\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G})\,, \tag{1}$$

$$u_{i,G}^{(j)} = \begin{cases} v_{i,G}^{(j)}, & \text{pokud } \mathrm{rand}_j \leq C_r \text{ nebo } j = j_{\mathrm{rand}}, \\ x_{i,G}^{(j)}, & \text{jinak}, \end{cases} \tag{2}$$

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G}, & \text{jestliže } f(\mathbf{u}_{i,G}) \leq f(\mathbf{x}_{i,G}), \\ \mathbf{x}_{i,G}, & \text{jinak}. \end{cases} \tag{3}$$

- $\mathbf{x}_{i,G}$ je i-tý cílový vektor v generaci G,
- $\mathbf{x}_{r_1,G}$, $\mathbf{x}_{r_2,G}$, $\mathbf{x}_{r_3,G}$ jsou tři různé náhodně vybrané vektory z populace,
- $F \in (0, 1]$ je škálovací faktor,
- $C_r \in [0, 1]$ je pravděpodobnost křížení,
- rand_j je rovnoměrná náhodná hodnota z intervalu $[0, 1]$ pro j-tou složku,
- j_{rand} zaručuje, že alespoň jedna složka pochází z mutanta,
- $f(\cdot)$ je minimalizovaná cílová funkce,
- $\mathbf{v}_{i,G}$, $\mathbf{u}_{i,G}$ a $\mathbf{x}_{i,G+1}$ jsou mutant, potomek a vybraný jedinec podle rovnic (1)–(3) [4].

Specifickou předností DE je, že mutační krok je samo-škálující: rozdílový vektor odráží aktuální rozptyl populace a adaptivně přizpůsobuje velikost kroku tvaru optimalizované funkce. U nižší diverzity se kroky zkracují a algoritmus jemně dolaďuje řešení, zatímco při velké diverzitě prozkoumává prostor agresivněji. Tato vlastnost, společně s malým počtem řídicích parametrů (F, CR a velikost populace), dělá z DE robustní nástroj na problematické, špatně podmíněné funkce.

Současný vývoj se soustředí na odstranění ručního ladění parametrů a na zlepšení škálovatelnosti. Existuje řada adaptivních modifikací, jež odstraňují nutnost předem ladit škálovací faktor F a pravděpodobnost křížení CR. Například v modifikaci s názvem jDE se obě veličiny kódují do jedince, a tak se evolučně samo-nastavují [15]. Algoritmus SHADE ukládá do paměti úspěšné kombinace parametrů a z jejich rozdělení v každé iteraci losuje nové hodnoty, zatímco L-SHADE navíc nelineárně zmenšuje populaci a tím urychluje konvergenci u vysoké dimenze [4]. Moderní komparativní studie potvrzují, že varianty L-SHADE, jSO či NL-SHADE-LBC stabilně překonávají nejen původní DE, ale

i mnoho jiných swarmových heuristik v benchmarkových soutěžích CEC [16]. Díky vektorové povaze operací se DE snadno implementuje na GPU i v ostrovních modelech, což umožňuje lineární škálování a zkracuje dobu ke konvergenci [17]. Celkově tak diferenciální evoluce představuje flexibilní, snadno rozšiřitelný a stále aktivně rozvíjený nástroj globální optimalizace.

2.5 Srovnání pro robotické trajektorie

GA vynikají univerzalitou kódování a snadným začleněním diskrétních omezení, avšak u spojitých problémů čelí pomalejší konvergenci a citlivosti na nastavení velikosti populace [3]. PSO je atraktivní minimem řídicích parametrů a rychlým dosažením slušné kvality, nicméně u multimodálních problémů často stagnuje a vyžaduje dodatečné mechanismy proti předčasnému shluku částic [8, 11]. DE kombinuje samoškálující mutaci s greedy selekcí a díky tomu udržuje diverzitu až do pozdních fází hledání. Samoadaptivní varianty jako SHADE či jSO navíc eliminují ruční ladění parametrů [4, 16]. Pro plánování robotických trajektorií, kde jde o optimalizaci spojitých řídicích bodů pod časovými a dynamickými omezeními, se DE ukazuje nejvhodnější.

3 Modulární algoritmy

V předchozích kapitole byl obecně popsán přístup EA k řešení složitých optimalizačních úloh. Ukazuje se však, že při zvyšování nároků na flexibilitu a kvalitu řešení v reálných provozních podmínkách (např. v robotických systémech s mnoha stupni volnosti nebo s komplikovanými omezeními prostředí) mohou klasické evoluční algoritmy vyžadovat četné úpravy a doplňky.

Modulární algoritmy představují přístup k návrhu a analýze evolučních či heuristických metod tak, aby bylo možné snadno kombinovat, porovnávat nebo vylepšovat jednotlivé stavební bloky (moduly) a sledovat tak vzájemné interakce různých komponent. V oblasti vícekriteriální i jednokriteriální optimalizace robotických trajektorií se tento princip ukázal zvláště užitečný, neboť prostřednictvím modularity lze rychle tvořit různé varianty algoritmů (např. s rozdílným mutačním operátorem či adaptací parametrů) a pečlivě zkoumat dopady těchto změn na výslednou výkonnost [18, 19, 20].

V klasickém přístupu se každý nový nápad či vylepšení (například nová mutační strategie, různé způsoby křížení či adaptace parametrů) integruje do základního EA, a ten je poté porovnáván se svou původní verzí. To však často ztěžuje férové hodnocení, jelikož se mohou lišit i drobné implementační detaily či implicitní nastavení parametrů [19]. Modulární rámec tuto situaci řeší tím, že poskytuje jednotnou implementaci hlavních stavebních bloků (např. operátory mutace, křížení, archivace či různé strategie výběru a řízení parametrů), přičemž každá z takových složek funguje jako samostatný modul. Využitím automatizované konfigurace je pak možné systematicky prohledávat prostor možných kombinací a hledat ty, které dosahují nejlepšího výkonu pro danou skupinu testovacích problémů (například v plánování robotických trajektorií s ohledem na kolize nebo délku dráhy). Jako příklady modulárních systémů, které lze využít i při řešení robotických trajektorií. Díky tomu lze systematicky zkoumat, jak se jednotlivé volby promítají do kvality dráhy, rychlosti konvergence či robustnosti vůči omezením prostředí [18, 19]. V následujících odstavcích rozšiřujeme popis tří nejpoužívanějších modulárních rámců.

3.1 Modulární CMA-ES (Covariance Matrix Adaptation Evolution Strategy)

Modularizovaná verze CMA-ES (modCMA) rozděluje algoritmus do desítek volitelných komponent, z nichž nejdůležitější jsou samplovací schéma (gaussovské, ortogonální či cauchyovské), adaptace velikosti kroku (klasická CSA, dvě kumulační cesty nebo různé varianty path length control), aktualizace kovarianční matice (plná, omezená paměť, aktivní či diagonalizovaná) a restartovací strategie (IPOP, BIPOP, NBIPOP) [20, 21]. Díky tomu lze vytvářet jak lehké varianty vhodné pro vysoké dimenze (LM-CMA-ES), tak robustní konfigurace pro vysoce multimodální úlohy s omezeními.

Rámec IOHprofiler-ModularCMAES poskytuje implementaci, v níž lze každou volbu zapínat přepínačem a automaticky ladit například velikost rodičovské váhy, způsob zpracování jedinců mimo přípustný prostor (zrcadlení, saturace) či elitismus při výběru přeživších. Studie zaměřené na predikci výkonu ukazují, že význam jednotlivých modulů se liší podle dimenze i rozpočtu funkčních hodnocení. Například modul mirrored sampling výrazně zlepšuje výsledky v nižších dimenzích, zatímco u vysokých dimenzí dominuje výběr základního sampleru [22]. Díky jednotnému kódu lze zároveň jednoduše replikovat klasickou CMA-ES a férově ji poměřit s novými sestavami.

3.2 PSO-X

PSO-X je komponentově orientovaný rámec navržený pro automatizovaný návrh algoritmů Particle Swarm Optimization [23]. Uživatel může pro každou část roje volit zvlášť strategii inicializace (latinské čtverce, halton, gaussian), topologii sousedství (globální, kruhová, Von Neumann, rastr se měnícím se poloměrem), vzorec aktualizace rychlosti (klasická inerce, dynamická inerce, adaptivní FIPS), mechanismus omezení rychlosti (clamping vs. constriction) a adaptaci parametrů (deterministické zmenšování, fuzzy regulátor, jPSO).

Modulární struktura dovoluje zahrnout i pokročilé funkce, jako je více-rojová kooperační PSO, resetování částic na základě diverzity nebo učení hodnot pro parametry pomocí reinforcement learningu. V experimentech na testovacích sadách BBOB i na reálných inverzních kinetických problémech dokázala automatická konfigurace PSO-X najít výrazně rychleji konvergující varianty než ručně laděné verze PSO, přičemž zachovala robustní chování v přítomnosti šumu a omezení [24]. Pro plánování trajektorií je klíčové, že modul pro topologii může dynamicky měnit vazby mezi částicemi podle tvaru prostředí, což podporuje prohledávání úzkých pasáží v kolizních prostorech [25].

3.3 Modulární diferenciální evoluce

Modulární diferenciální evoluce (MODDE) rozšiřuje klasickou DE o modulární kontrolu nad všemi stádii generování potomků [18]. V mutaci lze kromě klasických vzorů zvolit i vícero diferenčních vektorů, vážení faktorů F dle parametrické nebo adaptivní distribuce, referenční vektor pro varianty SHADE a využití archivu. Křížení podporuje binární, exponenciální či vlastní prostorovou transformaci eigen-křížení a hranová korekce nabízí více než deset možností. Adaptivní moduly v DE zahrnují různé přístupy pro dynamické řízení parametrů algoritmu a velikosti populace během optimalizace. Mezi významné metody patří například stochastická změna velikosti populace, používaná v algoritmu L-SHADE, která automaticky upravuje počet jedinců v populaci během procesu optimalizace. Dále sem patří metoda jDE, využívající samoadaptaci parametrů diferenciální evoluce F (faktor škálování) a CR (míra křížení), které se přizpůsobují samy během běhu algoritmu na základě průběžně dosahovaných výsledků. Dalším příkladem adaptivního modulu je metoda využívající adaptivní parametry algoritmu JSO (Joint-Swarm Optimization), což je varianta optimalizace kombinující více skupin řešení. Tyto parametry určují stupeň interakce a spolupráce mezi jednotlivými skupinami [18, 22].

Tento rámec umožnil vytvořit více než pět set různých variant algoritmu DE, ze kterých automatická konfigurace parametrů a adaptivních modulů dokázala identifikovat dosud nepopsanou kombinaci metod. Tato nově nalezená varianta dosáhla při testování na benchmarkových funkcích sady BBOB výrazně lepších výsledků než algoritmy L-SHADE a JADE, a to konkrétně o více než jeden řád velikosti méně potřebných hodnocení funkce pro dosažení podobné přesnosti [22]. Analýza metodou ANOVA navíc odhalila, že v nižších dimenzích dominuje vliv modulu pro redukci populace, zatímco ve vyšších dimenzích jsou rozhodující kombinace mutačních strategií a hranové korekce. Pro robotické trajektorie je důležité, že hranová projekce může zohlednit tvar pracovního prostoru, a adaptivní moduly pak umí agresivněji zmenšovat krok, jakmile se řešení přiblíží bezpečným oblastem bez kolizí [18]. Tab. 1 rozlišuje několik variant každého modulu a parametru, přičemž každá z nich ovlivňuje chování algoritmu jiným způsobem.

Tab. 1: Parametry modulární diferenciální evoluce.

Operace	**Modul**	**Zkratka**	**Typ**	**Možné hodnoty (Doména)**
Inicializace	Základní sampler	Sampler	c	{gaussian, sobol, halton, uniform}
Inicializace	Opoziční inicializace	Opposition	c	{true, false}
Mutace	Základní vektor	Base	c	{rand, best, target}
Mutace	Referenční vektor	Ref	c	{none, pbest, best, rand}
Mutace	Počet rozdílů	Diffs	c	{1, 2}
Mutace	Vážený faktor F	WeightedF	c	{true, false}
Mutace	Použití archivu	Archive	c	{true, false}
Křížení	Metoda křížení	Crossover	c	{bin, exp}
Křížení	Transformace vlastních hodnot	EigenX	c	{true, false}
Korekce hranic	Korekce	SDIS	c	{none, saturate, unif-resample}
Adaptace	Adaptace faktoru F	AdaptF	c	{none, shade, shade-modified, jDE}
Adaptace	Adaptace míry křížení (CR)	AdaptCR	c	{none, shade, jDE}
Adaptace	Redukce velikosti populace	LPSR	c	{true, false}
Adaptace	JSO caps pro F a CR	Caps	c	{true, false}
Parametr	Velikost populace	λ	i	$\{4, \ldots, 200\}$, doporučená: $4 + \lfloor 3 \log(D) \rfloor$
Parametr	Faktor škálování	F	r	$[0, 2]$ (doporučená $0,5$)
Parametr	Míra křížení	CR	r	$[0, 1]$ (doporučená $0,5$)

- Modul **Sampler** určuje způsob inicializace populace:
  - Gaussian inicializuje jedince kolem středu domény podle normálního rozdělení, což vede k vyšší koncentraci populace ve středu.
  - Sobol a Halton generují kvazi-náhodnou, rovnoměrně rozmístěnou mřížku bodů pomocí nízkodiscrepčních sekvencí.
  - Uniform vybírá jedince zcela náhodně s rovnoměrnou pravděpodobností v celé doméně.
- Modul **Opposition** přidává k náhodným jedincům jejich protějšky zrcadlené vůči středu domény:
  - True tuto techniku aktivuje a podporuje počáteční průzkum prostoru.
  - False tuto metodu vypíná.
- Modul **Base** určuje referenční vektor pro mutaci:
  - Rand vybírá libovolného člena populace.
  - Best používá nejlepšího jedince v populaci.
  - Target využívá přímo rodičovský vektor.
- Modul **Ref** určuje další referenční vektor:
  - None tento vektor nepoužívá.
  - Pbest vybírá náhodně z horních $p\,\%$ populace.
  - Best vybírá vždy nejlepšího jedince.
  - Rand vybírá náhodně z celé populace.

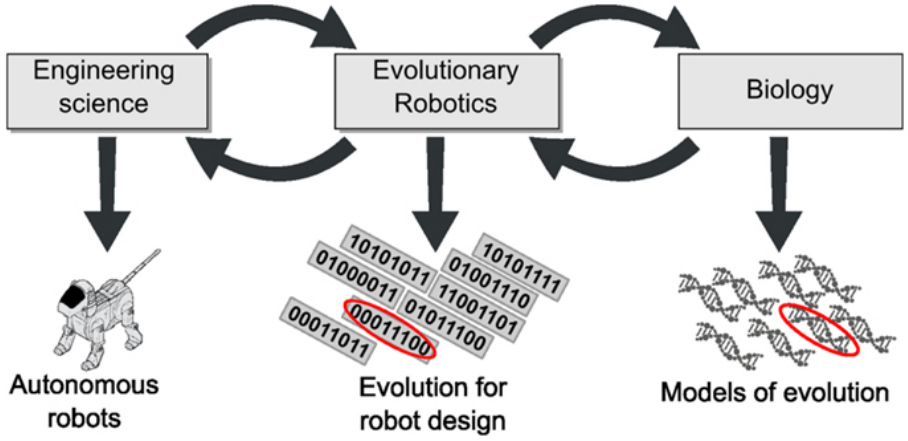
- Modul **Diffs** určuje počet rozdílových vektorů použitých při mutaci. Hodnota 2 zvyšuje diverzitu, ale také výpočetní náročnost.
- Modul **WeightedF**:
  - True umožňuje každému jedinci používat vlastní škálovací faktor F.
  - False sdílí populace jeden společný faktor F.
- Modul **Archive** stanovuje, zda je povoleno využívat archiv odmítnutých jedinců při tvorbě rozdílových vektorů.
- Modul **Crossover** určuje schéma křížení:
  - Binomické (bin) rozhoduje o každé komponentě nezávisle s pravděpodobností CR.
  - Exponenciální přepisuje souvislé bloky komponent, což je vhodnější pro zachování strukturovaných podčástí.
- Modul **EigenX** provádí křížení v bázi vlastních vektorů kovarianční matice populace (pokud true), což je výhodné pro eliptické či zkosené problémy.
- Modul **SDIS** řeší komponenty překračující hranice domény:
  - None ponechá hodnotu beze změny (může vytvořit neplatné body).
  - Saturate ořízne hodnotu na nejbližší platnou hranici.
  - Unif-resample znovu vylosuje náhodnou platnou hodnotu.
- Modul **AdaptF** určuje adaptaci faktoru F:
  - None používá konstantní F.
  - Shade aktualizuje F podle úspěšných mutací pomocí Cauchyho rozdělení.
  - Shade-modified navíc využívá Lehmerův průměr.
  - jDE provádí adaptaci F individuálně na úrovni jedinců.
- Modul **AdaptCR** analogicky adaptuje hodnotu CR.
- Modul **LPSR**, pokud je aktivní (true), lineárně snižuje velikost populace a zvyšuje tlak na konvergenci.
- Modul **Caps** (JSO caps), pokud je aktivní (true), omezuje adaptované hodnoty F a CR do bezpečného intervalu, aby se zabránilo extrémním hodnotám.
- Parametr λ určuje velikost populace, představující kompromis mezi výpočetní náročností a diverzitou.
- Škálovací faktor F řídí velikost mutačního kroku: hodnoty kolem 0.5 jsou univerzální, vyšší podporují průzkum a nižší jemné ladění.
- Míra křížení CR stanovuje, kolik komponent jedince se mění během křížení: vysoké hodnoty podporují větší změny, nízké hodnoty zachovávají strukturu jedince.

Modulární přístup k návrhu evolučních algoritmů spočívá v tom, že všechny varianty vycházejí ze společného programového základu a liší se pouze volbou jednotlivých

modulů. Díky této jednotné implementaci lze při optimalizaci robotických trajektorií spravedlivě porovnávat například různé mutační strategie či mechanismy řízení diverzity, aniž by výsledky zkreslily rozdíly v kódu [18]. Rámec je zároveň snadno rozšiřitelný, protože každý nový operátor, například speciální schéma archivace řešení nebo adaptivní řízení krokové velikosti pro robotické úlohy, přibude jako samostatný modul, který lze libovolně kombinovat s ostatními [19]. Modulárnost také umožňuje hloubkovou analýzu interakcí mezi komponentami, takže lze odhalit, které kombinace (např. určitá strategie výběru trajektorií a parametry křížení) si vzájemně pomáhají a které se naopak ruší. Modulární frameworky se dobře doplňují s metodami automatické konfigurace, jež dokážou na trénovacích instancích automaticky vybrat nejvhodnější moduly a jejich parametry. Například u robotických trajektorií zachovávají plynulost dráhy a zároveň minimalizují riziko kolizí.

4 Praktické využití evolučních algoritmů v robotice

EA představují pro moderní robotiku univerzální nástroj, protože dokáží globálně prohledávat prostor řešení, nevyžadují explicitní derivace a jsou málo citlivé na lokální extrémy. V průmyslové praxi se osvědčily zejména tam, kde se tradiční analytické postupy potýkají s vysokou dimenzionalitou, nelinearitou či vícekriteriálními omezeními. Typickými příklady jsou hledání optimální trajektorie manipulačního robota, doladění řídicích strategií u spolupracujících cobotů nebo návrh samotné morfologie stroje [26, 27, 28]. Na obr. 2 je znázorněno, že evoluční robotika je situována na rozhraní mezi inženýrskými vědami a biologií. Je inspirována biologickými principy a zároveň k biologii přispívá modely evoluce. Z inženýrství jsou přebírány metody pro návrh autonomních robotů, které jsou dále optimalizovány pomocí EA. Evoluční robotika tak slouží jako most mezi těmito dvěma obory [28].



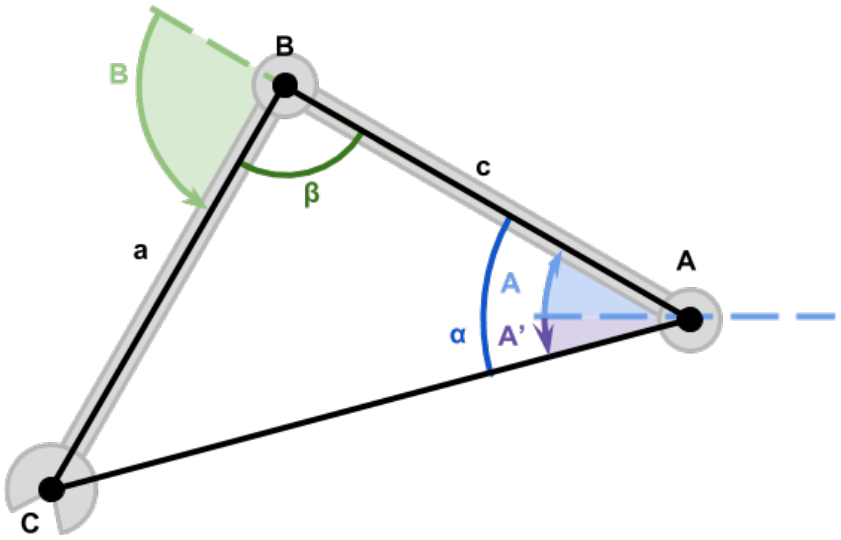
Obr. 2: Využití EA v robotice [28].

Paralelní populace kandidátních řešení spolu s mutacemi a křížením dávají EA schopnost objevovat navzájem odlišné oblasti vyhledávacího prostoru. Tím se snižuje pravděpodobnost uvíznutí v lokálním extrému a zároveň vzniká přirozený rámec pro vícekriteriální optimalizaci. Průmyslové trendy sdružené pod pojmem Průmysl 4.0 kladou důraz na flexibilitu a rychlou adaptaci výrobních linek. Zde EA profitují ze své schopnosti se přeučit na nové podmínky nebo na změněné prioritní váhy mezi časem cyklu, spotřebou energie či opotřebením komponent [26, 29].

4.1 Příklad inverzní kinematiky průmyslového manipulátoru

Pro planární dvoučlánkový manipulátor se dvěma klouby na obr. 3 jsou hledány kloubové úhly A (rameno) a B (loket), aby koncový efektor C dosáhl požadované polohy $\mathbf{p}_d = [x_d, y_d]^T$. Při známých délkách článků $L_1 = \|\overline{AB}\|$ a $L_2 = \|\overline{BC}\|$ stačí vypočítat délku

$d = \|\overline{AC}\|$ a pomocí kosinové věty odvodit vnitřní úhly trojúhelníku, z nichž přímo získáme kloubové natočení A a B. Výsledkem je uzavřená (ne-iterační) formulace inverzní kinematiky pro libovolnou dosažitelnou pozici $\mathbf{p}_d$. EA nabízejí alternativu: každý jedinec představuje kandidátní vektor kloubů a fitness vyhodnocuje eukleidovskou chybu pozice, úhlovou chybu orientace, vzdálenost od singularit a případně i kolizní riziko [30]. Pro redundantní manipulátory lze do fitness zahrnout i vedlejší cíle, například minimalizaci potenciální energie ramen nebo rozprostření tepelné zátěže mezi motory.



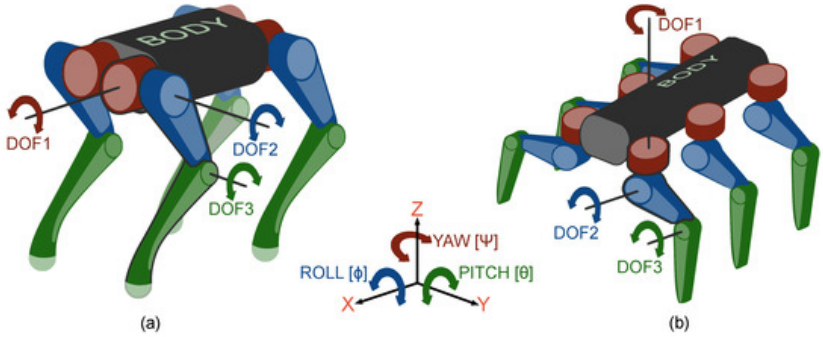
Obr. 3: Schéma inverzní kinematiky robotického ramene [31].

Kalibrace navazuje na inverzní kinematiku tím, že parametrizuje chyby v délkách článků, osy kloubů a rozložení hmotnosti. Tradiční lineární metody vyžadují dobrou inicializaci a mnoho opakovaných měření na kalibračních stavech. Evoluční přístup umožňuje řešit úlohu jako globální minimizační problém, kde se fitness odvozuje z rozdílů mezi skutečnými a modelovými trajektoriemi nástroje zaznamenanými sondou nebo kamerou [32]. Nelineární vazby mezi parametry tím nepředstavují překážku a EA se navíc snadno rozšíří o regularizační složky omezující fyzicky nesmyslné konfigurace (záporné délky, nejednoznačné otočení os).

4.2 Optimalizace robotické morfologie

Modulární konstrukce a aditivní výroba dovolují společně s EA automaticky navrhovat nejen řízení, ale i tělo robotu. Genotyp může explicitně popisovat posloupnost modulů a jejich vazeb, nebo používat nepřímou generativní reprezentaci. Během dekódování vzniká geometrie článků, typy pohonů a stanoviště senzorů. Fitness vyhodnocuje pracovní dosah, tuhost, momenty setrvačnosti a předpokládané výrobní náklady [33, 34].

Příkladem je evoluční návrh šesti nohou robota inspirovaného hmyzem, ilustrovaný na obr. 4. V navržené metodě je EA využíván jako vícekriteriální optimalizátor, v němž je každý jedinec reprezentován buďto explicitní posloupností stavebních modulů, nebo kompaktním přepisovacím pravidlem L-systému. Během dekódování je z genotypu automaticky vytvářena parametrizovaná CAD geometrie. Již v této fázi jsou aplikována omezení aditivní výroby, například minimální tloušťka stěn a maximální objem tiskové komory. Populace je následně vyhodnocována v GPU-akcelerované multifyzikální simulaci, v níž jsou současně počítány kinematické dosažitelnosti, vlastní frekvence, lokální napětí, akumulovaná únava i odhad anisotropních vlastností materiálu vzniklých orientací vrstev. Hodnota fitness je stanovena tříděním s preferencemi na zvětšení pracovního prostoru a tuhosti, minimalizaci hmotnosti a omezení výrobních nákladů. Pro redukci driftu kódu je mutace definována jako odchylka parametrů růstových pravidel, zatímco křížení zachovává celé podstromy gramatiky, čímž je podporováno dědění funkčně soudržných morfologických motivů. Během několika hodin je prohledán prostor řešení, jehož náročné testování by bylo konvenčními CAD/CAE postupy prakticky nemožné.



Obr. 4: Návrh nohou robota inspirovaného biologickou stavbou těla hmyzu [34].

Účinnost postupu byla ověřena na hexapodním robotovi, jehož evolučně odvozená morfologie dosáhla 50% delšího kroku, 95,5% většího pracovního dosahu a 7,9% nižší průměrné akcelerace pohonů než referenční šablonový model, přičemž byla současně snížena četnost rázových impulsů o 21,1% [34].

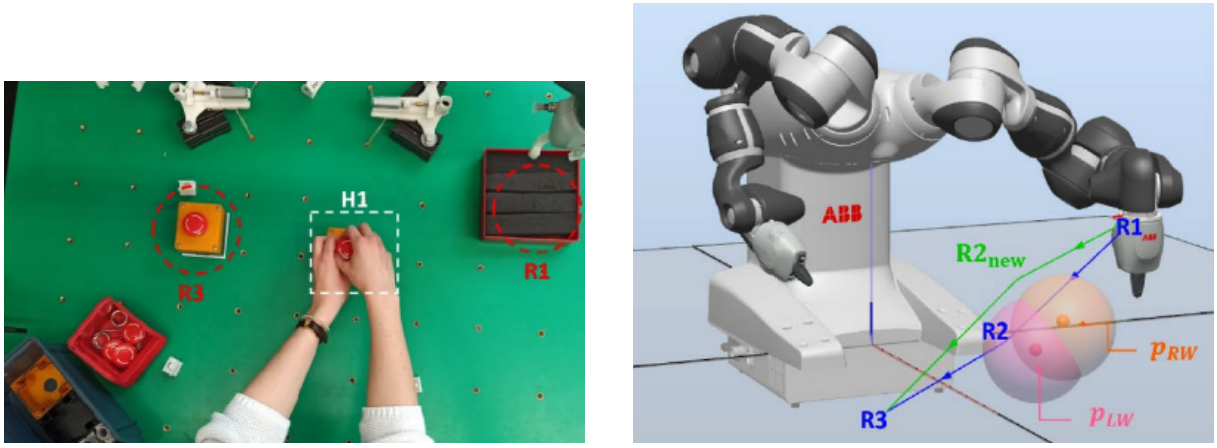
Obdobným postupem byly optimalizovány i složené robotické paže v omezeném kolizním prostoru buněk [33] a mřížkové výplně lehkých struktur generované nepřímým kódováním pro splnění lokálních tuhostních profilů kde v simulovaném prostředí lze modelovat i poruchy materiálu, takže evoluce preferuje odolnější tvary [26, 29].

4.3 Kolaborativní robotika a multi-robotické systémy

V kolaborativním prostředí je třeba optimalizovat trajektorii tak, aby byla pro člověka předvídatelná, aby nárazová síla nepřekročila limit a aby se více robotů vzájemně neblo-

kovalo. EA řeší úlohu v plně vícekriteriálním režimu, kde hard penalizace brání kolizím a soft penalizace upřednostňuje hladký chod. Polypopulační strategie umožňuje každému robotu lokální evoluci, přičemž elitní jedinci se sdílejí mezi skupinami [33, 29].

V kolaborativní robotice se do optimalizace přidává další vrstva: trajektorie musí respektovat nejen kinematiku robota, ale i předvídatelnost pro člověka, omezenou sílu nárazu a případný společný pracovní prostor. Metoda byla ověřena v průmyslovém scénáři montáže drobných součástek, kde byl dvouramenný cobot ABB YuMi provozován v těsné blízkosti operátora. Nominální trajektorie byla parametrizována přímo na úrovni instrukcí pohybu. GA byly vypočteny průchozí body, rychlosti a poloměry zaoblení, zatímco lexikografická vícekriteriální fitness minimalizovala pravděpodobnost aktivace bezpečnostní funkce a současně dobu cyklu. Každý kandidát byl okamžitě simulován v digitálním dvojčeti buňky, jež zahrnovalo model regulátoru i pravděpodobnostní mapu obsazenosti člověka. Po několika stovkách iterací byla nalezena trajektorie zkracující takt o 27 % bez spuštění nouzového režimu, čímž byla potvrzena průmyslová vhodnost evoluční optimalizace v kolaborativní robotice [35].



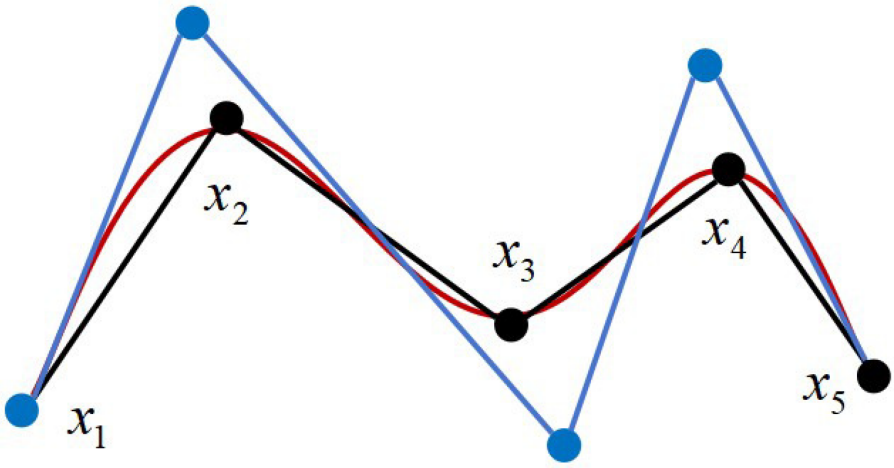
(a) Reálná scéna s operátorem. (b) Evaluace s digitálním dvojčetem.

Obr. 5: Testované kolaborativní prostředí [35].

Při koordinaci více robotů se navíc objevuje úloha rozdělení úkolů a asynchronního ladění cyklu tak, aby se jednotky vzájemně nepotkávaly v časoprostorovém konfliktu. EA zde slouží k současné optimalizaci cesty, rychlostního profilu a plánovače pořadí operací, přičemž polypopulační paradigma dovoluje každému robotu evolvovat lokálně se sdílením elit mezi skupinami [33]. Bezpečnostní kritéria, jako je minimální vzdálenost od pracovníka, se zapisují jako tvrdé penalizace. Měkčí preferenci (hladší pohyb, nižší hlučnost) lze zahrnout jako další složku fitness [29].

5 Plánování trajektorií s evolučními algoritmy

Schopnost rychle a bezpečně naplánovat trajektorii je zásadní pro úsporu času, energie i materiálu [36, 37]. Čím více stupňů volnosti robotický systém má, tím složitější je jeho konfigurační prostor (C-space). V případě manipulátorů obsahuje všechny možné kombinace kloubových úhlů. Bezkolizně bezpečné a optimální trajektorie mohou být výpočetně náročné. Tradiční sampling-based metody, jako jsou Probabilistic Roadmaps (PRM) a Rapidly-exploring Random Trees (RRT), obcházejí nutnost explicitní reprezentace celého C-space tím, že náhodně vzorkují jeho části a pokouší se tak nalézt dráhu od startu k cíli. EA se v plánování pohybu uplatňují dvojím způsobem. Prvním je přímá optimalizace celé trajektorie: genotyp kóduje sled interpolovaných bodů nebo kontrolních bodů B-spline křivky a fitness kombinuje délku, čas, energetický integrál momentů a penalizaci kolizí, viz obr. 6. Druhým je doladění parametrů dráhy předgenerované metodami RRT/PRM: například váhy mezi rychlostí a vzdáleností od překážek [38, 39, 30, 40, 41].



Obr. 6: Vyhlazení trajektorie pomocí B-spline křivky [42].

5.1 Probabilistic Roadmaps

Základní algoritmus PRM pracuje ve dvou oddělených krocích. Nejprve proběhne učící fáze v níž se v celém volném konfiguračním prostoru nezávisle generuje rovnoměrně náhodných vzorků. Každý vzorek prochází kolizním testem a pouze ty, které leží v oblasti C-space, se stávají vrcholy roadmapy. Pro každý vrchol se pak vyhledají jeho geometricky nejbližší sousedé (obvykle podle eukleidovské metriky). Pokud mezi danými dvojicemi existuje lokálně kolizně volná trajektorie, přidá se do grafu hrana. Tato lokální trajektorie se vypočítá lokálním plánovačem, jenž bývá jednoduchý lineární interpolátor nebo kinematicky omezený přechod. Úspěch testu zakládá hranu na obou uzlech se zadanou

váhou (délka, energie). Po vybudování grafu následuje dotazovací fáze. Aktuální start se dočasně přidají jako vrcholy, spojí se s nejbližšími kolizně volnými uzly roadmapy a hledá se nejkratší cesta. Pokud existuje alespoň jedna cesta, PRM je pravděpodobnostně úplný, což znamená, že s rostoucím počtem vzorků pravděpodobnost úspěchu konverguje k jedné. Výhodou je, že jednou vytvořenou roadmapu lze znovu použít pro mnoho různých dotazů bez nutnosti nového kolizního testování, takže on-line plánování probíhá v řádu milisekund, zatímco nevýhodou bývá paměťová náročnost pro velké prohledávané prostory, nižší efektivita ve velmi úzkých průchodech [43, 44].

5.2 Rapidly-exploring Random Tree

Jedním z nejběžnějších sampling-based přístupů je RRT. Tento algoritmus byl původně vyvinut pro rychlé prohledávání C-space v případech, kdy nejsou známy explicitní modely nebo je příliš rozsáhlý. Postup RRT je popsaný v Algoritmu 1. Začíná jediným uzlem ve startovní konfiguraci a opakovaně provádí cyklus: nejprve uniformně náhodně vygeneruje bod q_{rand} v C_{space}, najde k němu eukleidovsky nejbližší existující vrchol q_{nearest}, poté funkcí `Steer` vytvoří nový kandidát q_{new} ve směru q_{rand} s pevnou délkou kroku δ (tzv. step size), a pokud je spojnice $q_{\text{nearest}}q_{\text{new}}$ kolizně volná, přidá vrchol i hranu do stromu T. Tento proces se opakuje, dokud se některý q_{new} neocitne v ε-okolí cíle q_{goal}, nebo dokud nevyprší nastavený počet iterací N. Výsledkem je strom, jehož hrany přirozeně sledují volné oblasti konfiguračního prostoru a poskytují trasu z počátku do cíle, třebaže bez garance optimální délky [43, 45, 46].

Algoritmus 1 Zjednodušený pseudokód RRT.

$T \leftarrow \{\text{Start}\}$ ▷ inicializace stromu
for $i = 1$ to N **do**
  $q_{\text{rand}} \leftarrow \text{RandomConfig}()$
  $q_{\text{nearest}} \leftarrow \text{Nearest}(T, q_{\text{rand}})$
  $q_{\text{new}} \leftarrow \text{Steer}(q_{\text{nearest}}, q_{\text{rand}}, \text{stepSize})$
  if $\text{CollisionFree}(q_{\text{nearest}}, q_{\text{new}})$ **then**
    $T \leftarrow T \cup \{(q_{\text{nearest}}, q_{\text{new}})\}$
    if $\text{Distance}(q_{\text{new}}, q_{\text{goal}}) < \varepsilon$ **then**
      return T ▷ nalezen cíl
    end if
  end if
end for
return T ▷ cíl nenalezen, vrací se vybudovaný strom

5.3 Hybridní přístupy

Najít pouhou cestu v konfiguračním prostoru je obrovským krokem kupředu, ale pokud chceme optimalizovat dodatečná kritéria (délku, doby, energii, kolize s ostatními roboty), je potřeba pracovat s vhodnými ohodnocovacími funkcemi. Jakmile RRT vybuduje kolizně volnou cestu, dojde ke vyhlazení a optimalizaci trajektorie pomocí polynomických křivek, spline interpolací či evolučního ladění. Nebo se může přímo vkládat penalizační funkce do vyhodnocování stromu — např. pokud je nová hrana příliš drahá, nebude přidána. Poté, co RRT najde sekvenci mezi konfigurací, lze EA využít k postupnému vylepšování těchto uzlů v ohledu na vybrané kritérium. Typické cíle zahrnují minimalizaci celkového času cyklu, hladký rychlostní profil (minimalizaci trhavosti), snížení špičkových zrychlení a momentů a maximální vzdálenost od překážek [36, 47].

6 Autotuning

Autotuning je řízený proces, který systematicky prohledává prostor parametrů algoritmu s cílem nalézt konfigurace přinášející co nejlepší výkon na předem vybrané třídě úloh. Postup začíná popisem všech relevantních parametrů, jejich domén a podmíněných vazeb, pokračuje výběrem reprezentativní sady instancí a definicí rozpočtu experimentů. Poté se v iterativní smyčce automaticky navrhují nové konfigurace, spouštějí se na instancích a zaznamenává se hodnota cílové metriky. Na základě získaných dat se průběžně staví či aktualizuje model výkonnosti, který vede vyhledávání do slibných oblastí a postupně zužuje pátrání. Díky tomu se snižuje lidská pracnost, eliminuje subjektivní bias a odhalují se neintuitivní interakce mezi parametry [23, 48, 49].

6.1 SMAC

SMAC (Sequential Model-based Algorithm Configuration) využívá bayesovskou optimalizaci s náhradním modelem, který předpovídá vhodnost řešení na základě kombinace více rozhodovacích stromů, jež agregují výsledky svých jednotlivých predikcí pro zvýšení přesnosti a robustnosti. Model aproximuje vztah mezi konfigurací systému a jeho výkonem. Akviziční funkce pak vybírá taková nastavení, u nichž se očekává, že přinesou co největší zlepšení výkonu. K rozhodnutí, kterého kandidáta dále testovat, používá SMAC mechanismus nazývaný intensification: slibné konfigurace dostávají více instancí, dokud není rozhodnuto, zda překonají současné optimum. Nástroj podporuje reálné i kategoriální parametry, hierarchické a podmíněné prostory a umožňuje paralelní evaluace. V novější verzi SMAC3 lze navíc využít multi-fidelity schémata a časové rozpočty, což výrazně zrychluje ladění hlubokých sítí i kombinačních solverů [50, 51].

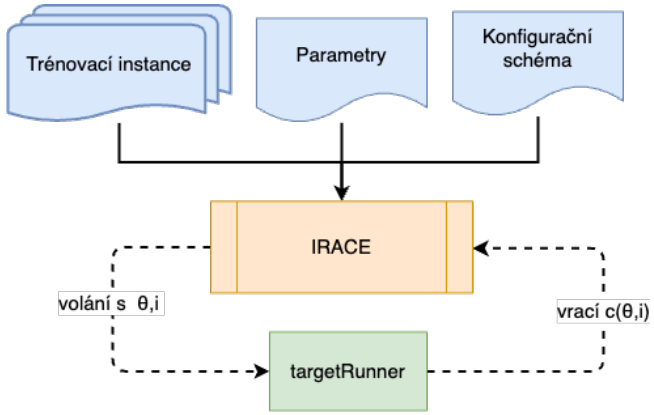
6.2 ParamILS

ParamILS staví na iterovaném lokálním vyhledávání v diskrétním prostoru konfigurací. Fáze local search střídá kroky zlepšování sousedními změnami s perturbačními skoky, které umožňují únik z lokálních optima. Klíčovou součástí je adaptivní omezení konfigurací. Pokud se během běhu ukáže, že nová konfigurace nemůže překonat dosavadního vítěze, evaluace se předčasně ukončí a ušetřený čas se věnuje jiným kandidátům. Rozšíření MO-ParamILS rozpracovává scénář s více cílovými metrikami a vrací množinu Pareto-optimálních konfigurací. Díky jednoduché implementaci a nízkým nárokům na předchozí znalosti je ParamILS často první volbou pro diskrétní prostory střední velikosti [52, 53].

6.3 Irace

Irace používá strategii iterovaných závodů. V každém kole spouští stejnou skupinu konfigurací na shodné podmnožině instancí a po dohrání testů aplikuje neparametrické statistické testy, aby okamžitě vyřadil konfigurace s prokazatelně horším výkonem. Uvolněná místa obsadí potomci elitních konfigurací, generovaní řízenými mutacemi s adaptivní velikostí kroku. Díky agresivnímu filtrování a průběžnému snižování počtu kandidátů dokáže Irace velmi efektivně hospodařit s rozpočtem a je proto vhodný tam, kde je jedna evaluace nákladná. Moderní implementace poskytuje detailní log s trajektorií konfigurací, volitelný soft restart při stagnaci a nástroj iraceplot pro vizuální analýzu výstupů [54, 55, 56].

Na vstupu do autotuneru (obr. 7) stojí trénovací instance, soubor laděných parametrů a jejich konfigurační schéma. Algoritmus z těchto informací generuje náhodný vzorek konfigurací $\boldsymbol{\theta}$, které předává spouštěči targetRunner. Ten na každé instanci i spustí cílový program s konfigurací $\boldsymbol{\theta}$ a vrátí hodnotu účelové funkce $c(\boldsymbol{\theta}, i)$. Iterace pokračují, dokud není vyčerpán globální rozpočet nebo dokud přírůstek kvality mezi dvěma po sobě jdoucími koly neklesne pod prahovou hodnotu. Pokud se výkon dlouhodobě nemění, spustí se soft restart: ponechá se nejlepší dosavadní konfigurace, zúží se intervaly pro reálné parametry a prioritně se prozkoumají dosud málo navštívené volby. Celý průběh včetně výsledků každé konfigurace, použitých instancí a průběžných statistických metrik je ukládán a lze jej následně analyzovat například pro odhad významnosti jednotlivých parametrů [54, 55, 56].



Obr. 7: Schéma iterativního racingu v Irace [54].

6.4 Výhody autotuningu

Autotuning sjednocuje podmínky experimentně a minimalizuje lidský vliv na volbu parametrů, takže výsledky jednotlivých algoritmů jsou férově a reprodukovatelně porovnatelné [54]. Díky tomu, že experimentální rozhraní i metrika úspěchu jsou pevně definovány nad společnou sadou instancí, vzniká transparentní datová stopa, která usnadňuje pozdější revizi a sdílení. Měřitelným přínosem je také odhalení netriviálních interakcí mezi parametry. Autotuner navrhuje konfigurace, jež by člověk pravděpodobně nikdy ručně nezkombinoval. Často tím nachází varianty s výrazně lepším výkonem.

Irace se pro časově nákladné fitness funkce odlišuje od SMAC a ParamILS způsobem jak hospodaří s rozpočtem. Zatímco SMAC investuje zdroje do opatrného zpřesňování odhadu výkonu slibných konfigurací a ParamILS provádí relativně mnoho úplných evaluací, Irace po každém kroku aplikuje neparametrické testy a bez milosti vyřadí konfigurace, jejichž výsledky jsou statisticky horší než výsledky elitní skupiny [55]. Tím rapidně klesá počet nutných spuštění drahé podkladové aplikace, protože špatná řešení jsou zastavena dřív, než vyčerpají celý limit.

Další výhodou je adaptivní strategie generování potomků: mutace se řídí šířkou rozptylu přeživších parametrů, takže se lokálně zvyšuje hustota vzorkování jen v opravdu perspektivních oblastech. Ve scénářích, kde jedno spuštění trvá desítky minut nebo hodin, poskytuje právě tato kombinace agresivního vyřazování a cílené lokální exploatace výrazně lepší poměr kvalita/cena než alternativní metody. Uživatel navíc získává kvalitní zpětnou vazbu se statistickými charakteristikami každého kola. Ta umožňuje detailně analyzovat, které parametry určují výkon a kde se vyplatí další ruční či automatické vylepšování [56].

7 Paralelizace evolučních algoritmů

Optimalizace robotických trajektorií se dnes opírá o výpočetně náročné simulace, v nichž se společně hodnotí kinematika, dynamika i kolizní omezení celého mechanismu. Jediný průchod simulačním modelem trvá desítky sekund až minut a EA navíc potřebují takových průchodů tisíce až miliony, protože každá generace populace se skládá z mnoha nezávislých jedinců [57]. Bez masivního paralelního běhu by proto i jednoduchá ladicí série zabrala dny až týdny. Paralelizace dovoluje rozdělit hodnoticí funkci trajektorie na více uzlů a zkrátit dobu experimentu řádově – při lineární škálovatelnosti lze z jednoho týdne srazit proces na hodiny.

Vedle primárního zrychlení přináší souběžné zpracování ještě jeden zásadní benefit: možnost testovat EA ve více scénářích současně. Robustní konfiguraci parametrů totiž obvykle hledáme napříč různými typy trajektorií, zatěžovacích stavů a rozmístění překážek. Paralelní běh těchto pod-úloh zabezpečí statisticky průkazné výsledky, aniž by se celkový čas experimentu prodlužoval [58].

7.1 Modely paralelizace

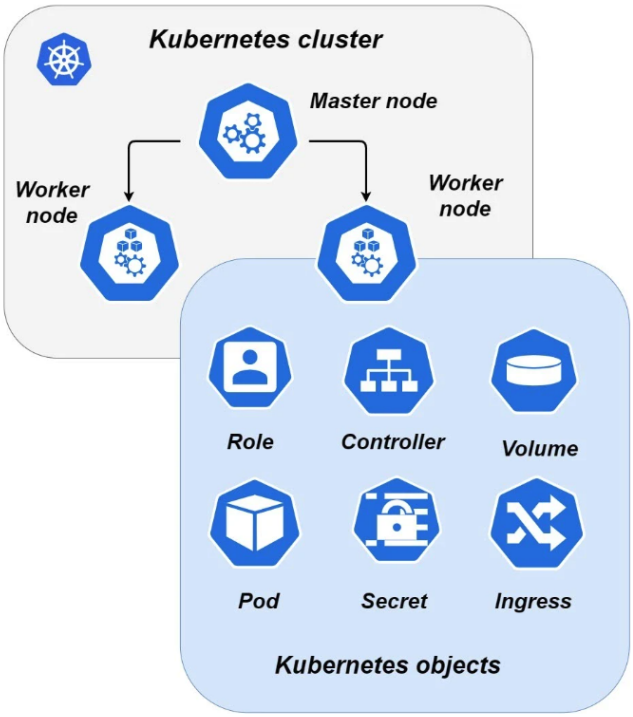
V praxi se u evolučních výpočtů prosadily tři základní modely. Master–slave varianta ponechává celou populaci na centrálním uzlu a rozesílá jen hodnoticí úlohy. Je jednoduchá, avšak špatně se škáluje. Island varianta rozděluje populaci na subpopulace, které evolvují nezávisle a jen periodicky migrují nejlepší jedince, čímž se snižuje komunikační režie a zvyšuje diverzita [58, 57]. Nejjemnější rozklad nabízí Celulární varianta, kde je každý jedinec svázán s buňkou 2-D mřížky. Tento přístup se dobře mapuje na GPU a umožňuje tisícům vláken paralelně vyhodnocovat fitness [59].

7.2 Kontejnerizace

Paralelní výpočetní úlohy v robotice typicky využívají rozličné knihovny pro simulaci dynamiky, detekci kolizí i numerickou optimalizaci a jejich závislosti se napříč operačními systémy liší. Kontejnerizace, reprezentovaná zejména Dockerem, řeší problém závislostí na specifické balíčky potřebné na spuštění specifických knihoven tím, že celý software uloží do obrazů přenosných na libovolný hostitelský systém [59]. Z pohledu vývojáře to znamená konzistentní prostředí od místního notebooku až po velký cluster, z pohledu provozovatele pak snadnou správu verzí a rychlé nasazení aktualizací. Protože každý kontejner běží izolovaně, lze na jednom uzlu nezávisle spustit mnoho různých variací evolučního běhu, aniž by se navzájem ovlivňovaly.

7.3 Architektura Kubernetes

Kubernetes staví nad Dockerem (či jiným runtime) řídicí rovinu, jejímž úkolem je plánovat, monitorovat a škálovat kontejnery. Základním stavebním kamenem je Pod – skupina jednoho či více kontejnerů sdílejících síť a svazky. Pody se spouštějí na pracovních (worker) Nodech, zatímco řídicí componenta běží na sestavě řídicích služeb, API server, scheduler a řídicí manager) [60, 61]. Konfigurace celého systému je deklarativní a popisuje se v souboru manifestu.



Obr. 8: Přehled Kubernetes objektů [62].

Obr. 8 ukazuje, že Kubernetes cluster tvoří řídicí uzel master Node, který plánuje a spravuje stav, a pracovní uzly worker Nodes, kde se kontejnery skutečně spouštějí. Nad touto infrastrukturou Kubernetes nabízí šest klíčových API objektů. Pod, nejmenší nasaditelná jednotka s jedním či více kontejnery. Controller, řídicí smyčka (např. Deployment), která udržuje požadovaný počet Podů. Volume, úložiště připojitelné k Podu. Secret, bezpečná distribuce citlivých dat (hesla, tokeny, klíče). Ingress, brána směrující HTTP(S) provoz na příslušné služby a Role, jemně odstupňovaná oprávnění k API clusteru.

Scheduler přiděluje Pody na uzly s ohledem na požadavky CPU, paměti či GPU a na uživatelské afinity. Při selhání uzlu control-plane okamžitě přeplánuje nedokončené Pody jinam a zaručí tak kontinuitu dlouhých evolučních experimentů [63]. Autoscaler umí reagovat na vlastní metriky, například délku fronty „fitness“ úloh, a téměř lineárně zvětší či zmenší počet výpočetních replik.

7.4 Paralerizační alternativy

Docker Swarm představuje minimalistickou vrstvu pro klastrový chod kontejnerů. Vychází ze stejného Docker prostředí, takže uživatel nemusí studovat novou syntaxi manifestů. Swarm používá řídící pracovní uzly, nasazuje služby založené na replikovaných nebo globálních úlohách a nabízí interní load-balancing. Díky jednodušší architektuře vyžaduje méně paměti, rychleji se konfiguruje a je vhodný pro menší clustery, kde postačí základní kontrola zdraví a restart aplikací [64]. Na druhou stranu postrádá pokročilé koncepty, oddělené prostory (namespaces) pro podporu více nezávislých uživatelů, pokročilé síťové moduly nebo mechanismy automatického škálování.

7.5 Porovnání paralerizačních metod pro robotické plánování

Ve srovnání s Docker Swarmem nebo ručním skriptem nabízí Kubernetes plnohodnotný ekosystém: deklarativní správu stavu, automatické převzetí při chybě, horizontální i vertikální škálování a bohatou sadu rozšíření (např. Kubeflow pro strojové učení). Tyto vlastnosti jsou klíčové pro paralelní evoluční optimalizace robotických trajektorií, kde experimenty běží dlouhé hodiny, dynamicky mění spotřebu zdrojů a vyžadují spolehlivý restart při pádu uzlu. Studie z roku 2024 prokázala, že správně naladěná instance Kubernetes dokáže na hybridním clusteru zvýšit efektivní propustnost výpočtu o 47 % oproti ručně spravovanému prostředí, zatímco provozní režie správy se snížila o třetinu [60]. Díky velké komunitě, rychlému vývoji a podpoře všech hlavních cloudů je Kubernetes dnes de-facto standardem pro robustní a škálovatelnou paralelizaci v robotice i dalších odvětvích.

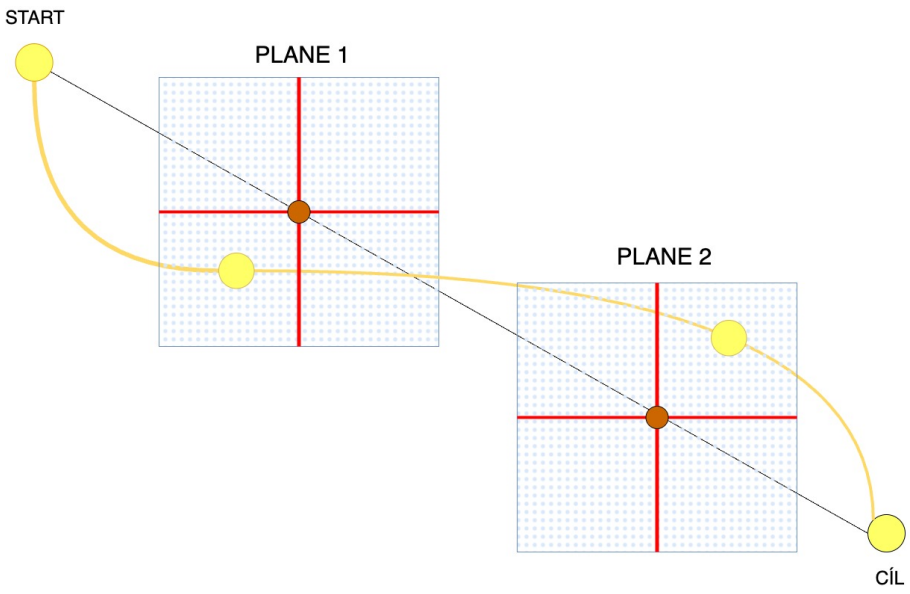
8 Implementace optimalizace robotických trajektorií

Cílem práce je návrh systému pro optimalizaci robotických trajektorií, který vychází z rešerše uvedené v předchozích částech práce. Na základě zvoleného přístupu bylo cílem vytvořit kompletní systém, který bude schopen generovat konfigurace stanovených robotických trajektorií ve zvolených scénářích. S ohledem na charakter problematiky je klíčové navrhnout vhodnou hodnotící funkci, která umožní objektivně hodnotit kvalitu jednotlivých řešení, a zároveň definovat konfigurační scény, ve kterých bude optimalizace probíhat.

8.1 Fitness funkce pohybu ve 3D prostoru

Aby bylo možné efektivně optimalizovat robotické trajektorie ve 3D prostoru, bylo nutné navrhnout vhodnou hodnotící funkci, která by jednoduše a zároveň objektivně hodnotila kvalitu generovaných trajektorií. Úloha byla zjednodušena na plánování pohybu robota mezi dvěma definovanými body v prostoru. Tyto body byly nejprve spojeny přímkou, která následně sloužila jako základní osnova pro generování trajektorie.

Přímka mezi těmito body byla rovnoměrně rozdělena na určitý počet segmentů. V každém bodě rozdělení byla vytvořena tečná rovina, definovaná vektorově vzhledem k předchozímu bodu. Na těchto rovinách jsou následně generovány řídicí body, jejichž rozmístění je řízeno parametry trajektorie. Zjednodušené schéma výsledné fitness funkce je na obr. 9.



Obr. 9: Schéma fitness funkce.

Po získání množiny řídicích bodů je trajektorie sestavena pomocí spline interpolace, konkrétně metodou best spline, čímž vzniká hladká a spojitá křivka reprezentující výslednou trajektorii.

Výsledná fitness funkce, která hodnotí trajektorii, je tedy složena ze dvou částí:

1. Minimalizace délky trajektorie: celková délka interpolované spline křivky je změřena a vynásobena váhovým faktorem, čímž je zajištěna preference kratších drah.
2. Penalizace kolizí: spline křivka i řídicí body jsou kontrolovány, zda se nenacházejí v definovaných zakázaných (kolizních) zónách. Každý průnik s kolizní oblastí je penalizován fixní hodnotou.

Matematicky lze hodnotící funkci vyjádřit následující rovnicí:

$$F(\mathbf{x}) \;=\; w \cdot L(\mathbf{x}) \;+\; \sum_{i=1}^{n_{\text{cp}}} \Big[\mathbf{1}_{FZ}\big(\mathbf{p}_i(\mathbf{x})\big) \cdot P \Big] \;+\; \sum_{j=1}^{n_{\text{spl}}} \Big[\mathbf{1}_{FZ}\big(\mathbf{s}_j(\mathbf{x})\big) \cdot P \Big], \tag{4}$$

kde

- $F(\mathbf{x})$ je výsledná hodnota fitness pro vektor parametrů $\mathbf{x}$
- w je váhový faktor délky trajektorie (path_length_weight)
- $L(\mathbf{x})$ označuje délku spline křivky určené parametry $\mathbf{x}$
- n_{cp} je počet vybraných řídicích bodů (tj. tečných rovin)
- $\mathbf{p}_i(\mathbf{x})$ je i-tý řídicí bod vypočtený z parametrů $\mathbf{x}$
- n_{spl} je počet vzorků na aproximované spline křivce
- $\mathbf{s}_j(\mathbf{x})$ je j-tý bod získaný při diskrétním vyhodnocení spline
- P je penalizace (`penalty_value`)
- $\mathbf{1}_{FZ}(\mathbf{q})$ je indikační funkce, rovna 1, pokud bod $\mathbf{q}$ leží v zakázané zóně, jinak 0

Výsledná trajektorie, optimalizovaná touto funkcí, tedy představuje nejkratší možnou křivku, která se vyhýbá všem definovaným zakázaným oblastem. Tato fitness funkce slouží jako základní podklad pro EA, jehož úkolem je nalézt optimální konfiguraci parametrů trajektorie.

8.2 Robotické konfigurační trasy

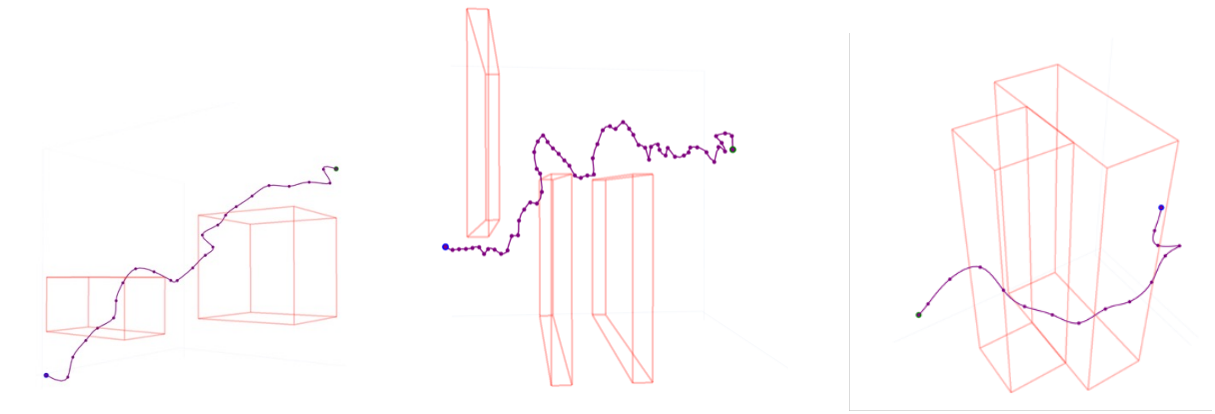
Pro účely evaluace rozdílných přístupů k optimalizaci trajektorií bylo vytvořeno celkem pět specifických scén, které umožňují detailní analýzu výkonu použitých metod. Každá z těchto scén má své jedinečné charakteristiky, čímž zajišťuje komplexní testování optimalizačního algoritmu. Každá scéna je sestavena ze dvou bodů, počátečního a cílového, ve formátu souřadnice ve 3D prostoru. Dále je jsou zde definovány zakázané zóny, reprezentující oblasti, kudy robot nemůže projít. Zvolená podoba všech těchto oblastí jsou kubické objekty. Ty jsou sestaveny vždy ze dvou částí. První je bod reprezentující levý spodní roh v prostoru. Dále je k němu přidružen druhý list parametrů, který reprezentuje šířku, délku a výšku. Výsledkem jsou tedy sestavené scény uložené ve formátu JSON (JavaScript Object Notation), což je odlehčený formát pro výměnu dat. Touto reprezentací je zajištěna jednoduchá modifikace, aktualizace a případná expanze testovacích případů. Pro účely validace vytvořeného frameworku, bylo vytvořeno 5 validačních scén s odlišnými charakteristikami.

- **Scéna 1** zkoumá schopnost algoritmu efektivně najít bezkolizní cestu přes hrany postupně klesajících zakázaných zón.
- **Scéna 2** hodnotí podobný problém, ale s důrazem na optimalizaci délky trajektorie při pohybu mezi třemi různě umístěnými kolizními objekty.
- **Scéna 3** porovnává dvě velmi podobné trasy, přičemž jedna trasa je znatelně delší, a testuje schopnost algoritmu preferovat kratší dráhu.
- **Scéna 4** simuluje reálnou úlohu robotické manipulace, kde je trajektorie určena pro přemístění objektu z jednoho prostoru do druhého.
- **Scéna 5** integruje prvky všech předchozích scén a přidává tzv. slepé cesty, čímž výrazně zvyšuje komplexnost úlohy.

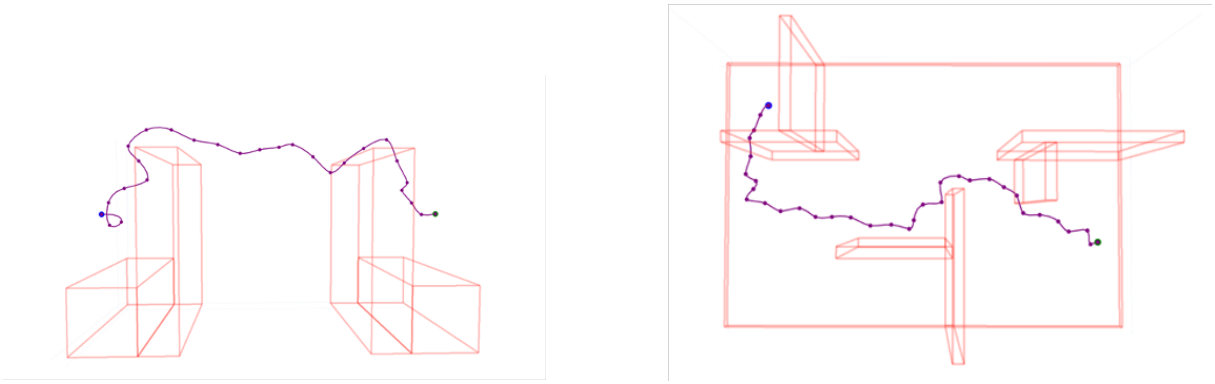
Pro každou scénu je dále vytvořena referenční RRT trajektorie, která slouží jako výchozí populace pro EA. Tyto trajektorie pomáhají algoritmu rychleji konvergovat k přípustnému řešení tím, že předdefinují počáteční hrubou trasu. Každý bod této trasy, který leží na generovaných tečných rovinách, se stává počátečním bodem evoluční populace. Tyto body jsou součástí JSON definic scén a jsou aktivně využívány při konfiguraci MODDE.

8.3 Generování referenční trajektorie metodou RRT

Pro každou scénu byly načteny souřadnice počátečního, cílového bodu a pravoúhlých překážek definovaných jejich rozměry. Následně byly vypočteny hranice vzorkovaného prostoru. Algoritmus RRT následně 10 000x náhodně vzorkoval bod, určoval k němu nejbližší uzel stromu. Vytvářel tak nový uzel a kontroloval kolizi s překážkami vzorkováním úsečky v jednotlivých krocích. Pokud úsek žádnou překážku neprotínal, uzel se připojil do stromu. Jakmile se nový uzel přiblížil k cíli a přímá spojnice byla bez kolize, trajektorie byla považována za nalezenou. Celý proces se kvůli stochastické povaze algoritmu opakoval 100x. Z každého běhu se zaznamenala délka cesty a nejkratší z nich, nejlepší cesta. Ta byla vložena do JSON definice scény, aby její vrcholy sloužily jako počáteční populace pro MODDE. Všechny vytvořené scény byly sestaveny v 3D prostoru pomocí python balíčku pyplot a jejich zjednodušená vizualizace včetně vygenerovaných RRT tras je na obr. 10.



(a) Scéna 1. (b) Scéna 2. (c) Scéna 3.



(d) Scéna 4. (e) Scéna 5.

Obr. 10: Zvolené robotické scény. Fialovou barvou je vyznačena trasa vygenerovaná RRT.

8.4 Architektura systému

Pro návrh a provoz rámce určeného k výpočtu optimálních konfigurací robotických mechanismů byla zvolena architektura založená na kontejnerizační platformě Kubernetes, konkrétně na její odlehčené distribuci MicroK8s. Volba MicroK8s umožňuje spustit plnohodnotný Kubernetes na jediném uzlu s minimálními systémovými nároky, přičemž díky snadné instalaci formou snap balíčku ho lze rychle nasadit do laboratorního prostředí bez potřeby externí orchestrátorové infrastruktury. Klíčovým argumentem byly vestavěné privátní kontejnerové registry, které zjednodušují publikaci a načítání obrazů přímo ze stroje hostujícího clusteru. Spolu s volitelnými doplňky typu `dns`, `hostpath-storage` nebo interaktivního dashboardu, tak MicroK8s poskytuje kompletní ekosystém potřebný k lokální iteraci nad experimenty, aniž by bylo nutné spravovat doplňkové služby.

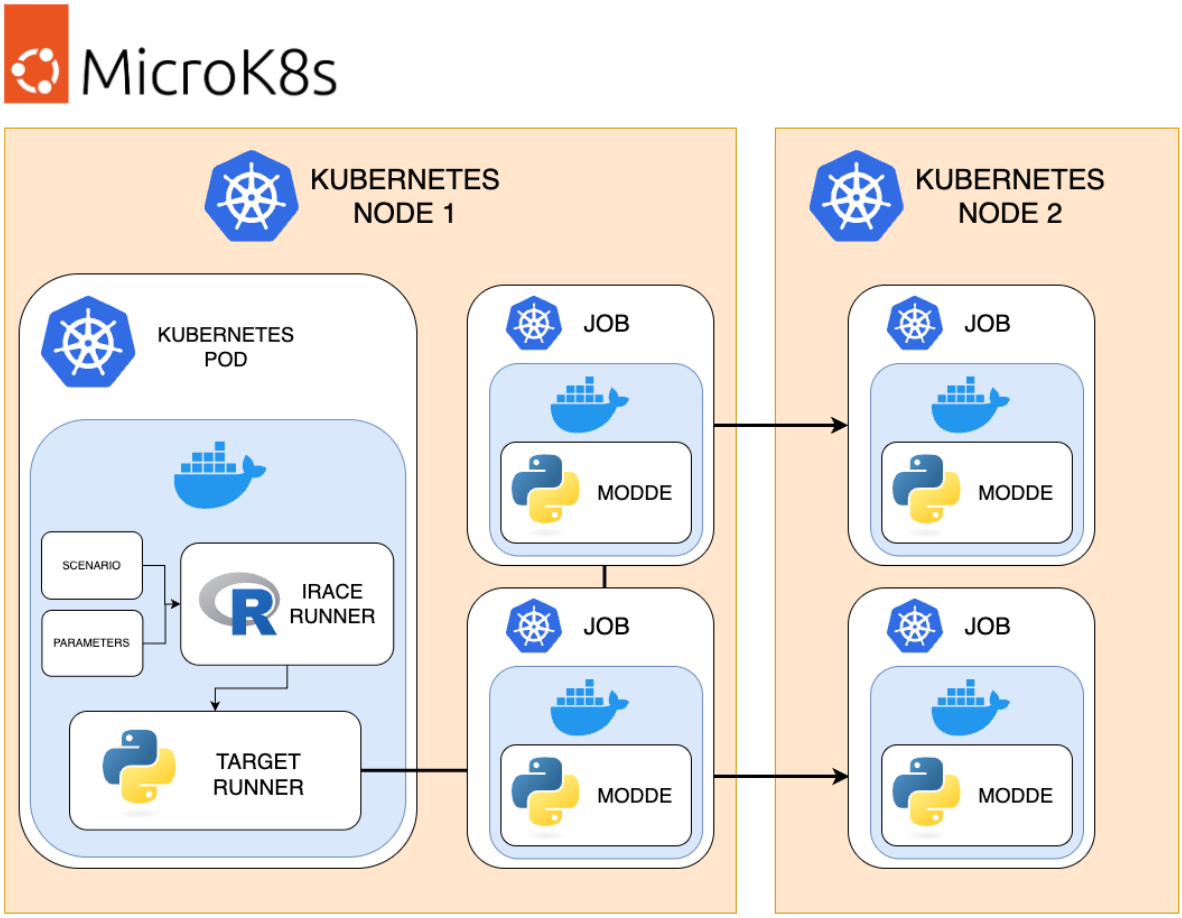
Samotné řešení je rozděleno do dvou logických komponent. Řídicí Pod `irace-controller` obsahuje obraz, v němž je předinstalován jazyk R s balíčkem Irace, knihovnu Pythonu a nástroj kubectl. Pod je svázán se statickým uzlem spuštěného řídícího serveru pomocí parametru `NodeSelector`. Jeho práva na restart jsou zakázána, aby se po dokončení hlavního optimalizačního běhu zamezilo opakovanému spouštění. Do cesty kořenové složky je přidružený perzistentní svazek, tedy vstupní i výstupní data zůstávají zachovány mezi jednotlivými běhy. V Podu se spouští skript `run-irace`, který generuje a dynamicky odesílá na API server manifesty Jobů pro výpočty kandidátních konfigurací.

Každý jednotlivý výpočet je realizován zmíněným Kubernetes Jobem. Manifest je programově vytvářen funkcí `create-job-manifest`, která doplňuje unikátní jméno Jobu, volitelný seznam argumentů předávaný spouštěnému skriptu `de-worker.py` a nastavuje časovač pro automatické odklizení metadat po doběhnutí. Záložní nastavení je nulové, čímž se předchází opakovaným pokusům v případě selhání, neboť chybové stavy jsou vyhodnocovány přímo v řídicím skriptu. Stejně jako kontroler montuje pracovní PVC do kořenové složky, čímž se zajišťuje sdílený adresář pro vstupní dataset i uložení meziproduktů a výsledných metrik. Všechny Joby jsou rovněž fixovány na společný server, což zaručuje, že sdílený svazek ukazuje na stejné místo v souborovém systému.

Trvalé úložiště je tvořeno objektem `PersistentVolume` se zvolenou kapacitou 5 GB a přístupovým módem `ReadWriteMany`, aby mohl k souborům simultánně přistupovat jak kontroler, tak všechny paralelní Joby. Úložiště je mapováno na adresář v hostitelském uzlu a je svázáno s `PersistentVolumeClaim irace-pvc`. Reklamační politika ponechává data i po odstranění Podů, takže je lze následně archivovat či opakovaně analyzovat.

Sestavení a nasazení celého prostředí se automatizuje skriptem `irace-k8s.sh`. V režimu `build` skript odstraní zastaralé lokální obrazy, znovu sestaví obraz kontroleru i pracovníka na základě příslušných `Dockerfile` a otaguje je pro interní registry `localhost:32000`. Po úspěšném nasazení na registr následuje fáze `start`, která jediným příkazem načte hlavní manifest `irace-manifest.yaml` a vytvoří všechny popsané zdroje. Režim `stop` pak manifest zase smaže a uvolní tak cluster pro další experiment. Tento

pracovní postup dovoluje rychle iterovat nad konfiguracemi algoritmu, protože změna kódu nebo scénáře se promítne pouze do přestavby příslušného obrazu a nového spuštění skriptu.



Obr. 11: Navržená architektura.

Výsledkem je modulární systém popsaný schématem na obr. 11, v němž řídicí proces Irace uvnitř `controler-podu` generuje paralelní dávky Kubernetes Jobů, které spouštějí evoluční dělené výpočty nad společným datovým prostorem. Díky MicroK8s jsou všechny součásti lokalizovány na jediném fyzickém serveru, takže se minimalizují režijní latence při předávání souborů a zároveň zůstává zachována plná kompatibilita s cloudovými clustery, pokud by bylo potřeba řešení škálovat na více uzlů. Na Ústavu automatizace FSI VUT byly výpočty prováděny na dvojici identicky konfigurovaných linuxových serverů (lab-uai a lab2-MS-7A32) s operačním systémem Ubuntu 22.04.5 LTS a jádrem 6.8. Oba uzly byly vybaveny procesorem AMD Ryzen 7 2700X se šestnácti logickými jádry (8 fyzických jader s hyper-threadingem, základní takt 3,7 GHz) a 32 GB operační paměti DDR4. Identická hardwarová konfigurace zjednodušuje symetrické rozložení zátěže v lokálním clusteru MicroK8s a zajišťuje dostatečný počet výpočetních jader pro paralelní běh kontejnerů i testování škálovatelnosti navrženého řešení.

8.5 Implementace vybraných frameworků

Vstupním nastavením frameworku Irace jsou konfigurační soubory, které definují průběh optimalizačního procesu. Ty jsou rozděleny do čtyř hlavních typů. První typ tvoří soubor `scenario`, který obsahuje základní nastavení algoritmu. V něm je mimo jiné definována absolutní cesta k parametrům `targetRunner`, což je soubor spouštěný přímo z konzole Irace, cesta k `parameterFile`, jenž definuje rozsahy a varianty parametrů a také `trainInstancesFile`, ve kterém jsou uvedeny jednotlivé scény, jež slouží k tréninkové fázi algoritmu.

Samotný běh optimalizačního procesu je řízen sekcí omezení v souboru `scenario`, kde parametrem `maxExperiments` určujeme celkový počet povolených volání cílového skriptu `targetRunner`, parametrem `nbIterations` nastavujeme maximální počet iterací frameworku Irace. Pomocí `NbConfigurations` definujeme počet paralelně testovaných konfigurací v každé iteraci. Poslední skupinu parametrů představují testovací instance, což jsou scény, na nichž Irace vyhodnocuje nejlepší nalezené konfigurace po dokončení tréninkové fáze.

Celý proces je iniciován spuštěním python skriptu `run_irace`, který představuje hlavní rozhraní mezi uživatelem a R knihovnou Irace. Skript nejprve ověří přítomnost a správnost konfiguračních souborů. Poté, na základě údajů extrahovaných ze souboru parametrů, vytvoří specifický adresář pro ukládání výsledků s časovým razítkem a identifikací konkrétní scény. Skript rovněž umožňuje obnovení přerušeného běhu pomocí záchytných bodů v podobě souboru `irace-backup.Rdata`.

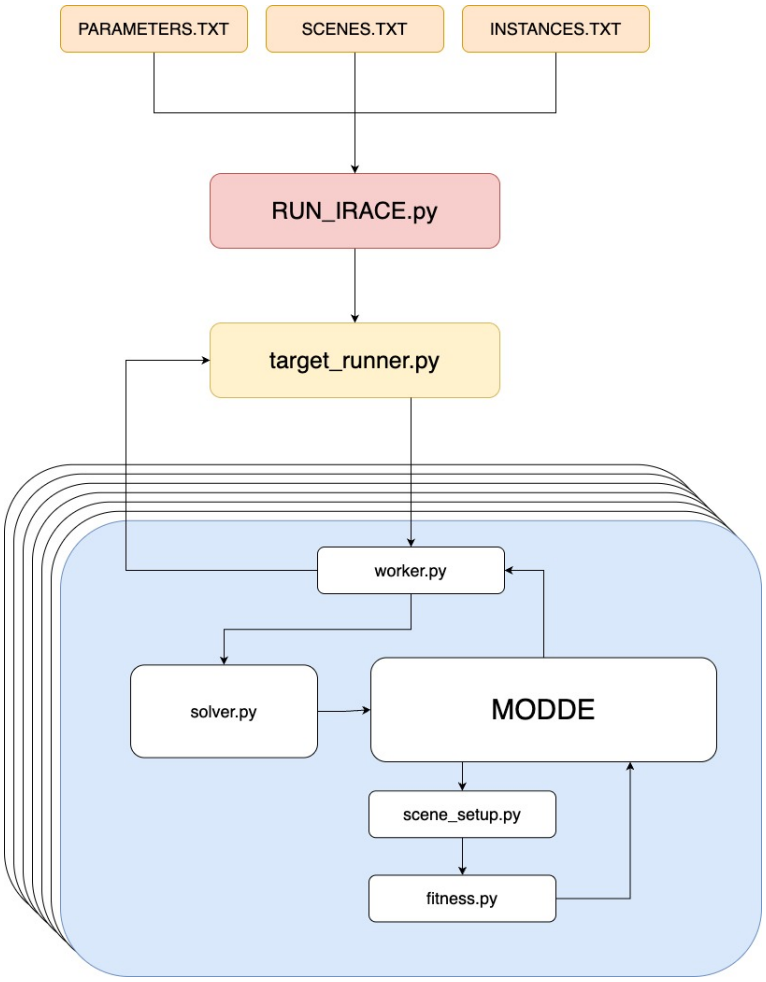
Po tomto úvodním nastavení skript `run_irace` vytvoří a spustí dočasný R skript, který přímo inicializuje běh Irace. V tomto R skriptu je detailně popsáno, jakým způsobem má Irace interpretovat jednotlivé parametry. Skript zajišťuje, že výsledky jednotlivých experimentů jsou zaznamenány do souboru `irace.log`, kde je později možné identifikovat elitní konfigurace. Po dokončení běhu Irace jsou generovány také doplňkové vizualizace, například graf vývoje výkonu nejlepší konfigurace, krabicový graf výkonů všech konfigurací či sloupcový graf průměrných výsledků testovacích konfigurací.

Z konzole Irace je dále volán modifikovaný skript `targetRunner`, který představuje rozhraní mezi optimalizačním algoritmem Irace a výpočetním prostředím Kubernetes. Skript `targetRunner` převede vstupní parametry konfigurace, dodané Iracem, do odpovídajícího formátu terminálových argumentů. Tyto argumenty jsou předávány skriptu, který připraví specifický Job manifest pro Kubernetes. Každá testovaná konfigurace dostane unikátní identifikátor, podle kterého je následně vytvořen jedinečný Kubernetes Job. Tento Job je spouštěn ve výpočetním clusteru, kde jsou izolovaně prováděny výpočty.

Každý Job, spuštěný v Kubernetes prostředí, obsahuje balíček skriptů zajišťujících samotné vyhodnocení konfigurace. Primárním skriptem je `de_worker`, který nejprve parsuje vstupní argumenty z prostředí Kubernetes a následně spouští optimalizační algoritmus - MODDE [18]. Před spuštěním algoritmu jsou argumenty využity k inicializaci

parametrů pro sestavení geometrické scény. Během této inicializace je vytvořena sada tangentních rovin, určující podmínky prostoru pro výpočet spline křivky, a nastavena fitness funkce, definující kvalitu nalezeného řešení.

MODDE je implementována skriptem `modular_de.py`, kde pro každou iteraci generuje populaci kandidátních řešení podle předaných parametrů. Každá populace kandidátních řešení je hodnocena pomocí fitness funkce obsažené ve skriptu `fitness.py`, která měří kvalitu nalezené cesty mezi počátečním a koncovým bodem na základě počtu překročení zakázaných zón, hladkosti spline křivky a dalších kritérií. Algoritmus pracuje s omezením stanoveným parametrem `budget`, což zaručuje ukončení procesu při dosažení stanoveného maximálního počtu volání fitness funkce. Diagram navrženého frameworku je na obr. 12 [18].



Obr. 12: Vývojový diagram výpočetního frameworku.

Po ukončení běhu optimalizačního algoritmu jsou výsledky zahrnující nejlepší nalezenou hodnotu fitness funkce, parametry optimalizovaného řešení a další metadata, zapsány do jednotné složkové struktury. Tyto výsledky jsou poté zpětně přenášeny do skriptu `targetRunner`, který je extrahuje z logů Kubernetes Jobů. Skript následně komunikuje

dosažené výsledky zpět do frameworku Irace, který na základě dosažených hodnot fitness funkcí rozhoduje o další selekci nejlepších konfigurací.

Tento cyklus se opakuje, dokud není dosaženo limitů definovaných počtem iterací nebo experimentů v původním scénáři Irace. Po ukončení celého optimalizačního procesu framework Irace automaticky generuje přehled výsledků zahrnující kompletní záznamy z optimalizačních iterací, nejlepší sady parametrů a vizualizace výkonů jednotlivých konfigurací. Všechny výstupní soubory jsou přehledně uloženy v již zmíněné adresářové struktuře spolu s detailním konfiguračním souborem v jazyku R, který je možné využít k replikaci dosažených výsledků.

Celý framework je koncipován modulárně, což umožňuje snadnou výměnu komponent, testování nových optimalizačních strategií či rozšíření o další scénáře a varianty geometrických modelů. Díky integraci s výpočetním prostředím Kubernetes je navíc celý systém škálovatelný a připravený na paralelní výpočty na více výpočetních uzlech, což výrazně urychluje optimalizační proces. Vytvořený kód frameworku je v příloze A.

8.6 Vytvoření aplikace

Pro snadnou vizuální kontrolu a analýzu výsledků optimalizačních běhů byl vyvinut samostatný prohlížeč, který je balen do vlastního kontejneru a spouštěn jako `irace-app-pod`. Pod je, stejně jako ostatní komponenty, připnut na uzel `lab-uai` a sdílí tentýž perzistentní svazek. Následně ho pomocí `irace-pvc` mapuje do kořenového adresáře, takže má přímý přístup k všem generovaným scénám, logům i výsledkovým strukturám JSON. Kontejner startuje jediným příkazem, jenž spustí Flask server poslouchající na portu 6000. Exponování aplikace mimo cluster zajišťuje služba `irace-app-svc` typu `NodePort`, která směruje příchozí TCP provoz z veřejného portu 30001 na vnitřní port 6000 a tím zpřístupňuje webové rozhraní na IP adrese hostitelského uzlu.

Back-end je postaven na minimalizovaném Flasku a zahrnuje několik REST koncových bodů. Kořenová cesta vrací statickou šablonu `index.html`. V ní je implementováno JavaScriptové dynamické načítání dat. Živé logy lze odebírat přes dlouhé spojení na cestě logů, kde generátor průběžně streamuje nové řádky a umožňuje tak sledovat průběh optimalizace bez opakovaného obnovení stránky.

Pro diagnostiku clusteru je k dispozici `endpoint-pods`, jenž využívá oficiálního python Kubernetes klienta a vrací textový výpis všech Podů včetně jejich aktuální fáze. Soubor `parameters.txt` obsahující podrobnosti o testovaných hyperparametrech je dostupný přes tlačítko parameters. Nejdůležitější část aplikace tvoří dvojice cest `results-list` a `results-details`, které procházejí adresář výsledných běhů, vyhledávají jednotlivé běhy a u každého extrahují klíčovou statistiku z JSON souborů. Přes funkci `view-run` lze následně otevřít detail vybraného souboru, kde se generuje interaktivní vizualizace v python knihovně Plotly zobrazující optimální trajektorii, zakázané oblasti a další relevantní prvky scény. Vedle 3D pohledu je do stránky vložena i tabulka parametrů sestavená přímo z JSON dat a jednoduchý JavaScriptový přepínač panelů pro rychlé přepínání mezi grafem a numerickými hodnotami.

Analogickým způsobem jsou obsluhovány statické testovací scény v podadresáři scén. Koncový bod `scene-list` poskytuje jejich seznam a vrací plnohodnotné HTML, lze tedy zobrazit 3D pohled na scénu bez opuštění dashboardu. Vizuální náhled je generován pomocí funkce, která kombinuje data o zakázaných zónách se základní cestou vypočtenou algoritmem RRT a vykresluje je opět prostřednictvím Plotly, aby uživatel mohl intuitivně posoudit, zda nalezená konfigurace respektuje prostorové omezení manipulátoru. Aplikace je ilustrována na obr. 13.

Obr. 13: Vizuální stránka aplikace s prohlížečem trajektorií.

Aplikace poskytuje kompletní servis od spuštění optimalizačního procesu přes on-line sledování jeho průběhu až po post-mortem analýzu jednotlivých běhů, a to vše s využitím společného datového úložiště, takže odpadá nutnost exportu souborů či složité synchronizace mezi kontejnery. Protože je aplikace spuštěna v odděleném Podu a její služba je v Kubernetes manifestu vystavena ven pomocí typu `NodePort`, lze ji bezpečně provozovat i na vzdáleném stroji mimo laboratoř, aniž by to ohrozilo zbytek výpočetního prostředí.

9 Srovnání konfigurací, evaluace výsledků

Pro porovnání nalezených konfigurací bylo testování rozděleno na dva rozdílné přístupy. Jedním je získání indiviuálních konfigurací pro každou scénu zvlášť, trénovanou na scéně samotné, a druhý přístup s nalezením jednoho globálního nastavení pro všechny scény a jejich výsledné porovnání. Dalším parametrem testování bylo ověření vlivu předgenerované trasy pomocí RRT a výsledků bez této iniciální populace.

V porovnání s obecnou sadou modulů uvedenou v tab. 1 byla použitá konfigurace současně zjednodušena i rozšířena. Nebyly využity moduly Sampler, AdaptF, AdaptCR, LPSR a Caps, neboť při předběžných experimentech nebyl jejich vliv na kvalitu řešení shledán statisticky významným, zatímco byla zaznamenána vyšší časová režie. Naopak byly zařazeny parametry budget, oversampling-factor, oppositional-generation-probability, inialize-custom-pop a init-stats. Parametrem budget je omezován maximální počet vyhodnocení fitness, oversampling-factor určuje poměrně vyšší počet kandidátů generovaných v každém kroku, oppositional-generation-probability nastavuje pravděpodobnost, s níž je místo potomka vytvářen jeho opoziční protějšek, inialize-custom-pop umožňuje načtení uživatelsky definované počáteční populace (RRT) a init-stats zapíná ukládání statistických údajů o počátečním rozložení. Dále byly zahrnuty proměnné tangent-size, divided-by, num-spline-points a penalty-value. Parametrem tangent-size je určována velikost ploch na křivce fitness, divided-by vymezuje délkovou konstantu pro dělení trajektorie na jednotlivé části, num-spline-points popisuje celkový počet hodnotících bodů na vytvořené křivce a penalty-value stanovuje výši penalizace za vstup do zakázaných oblastí. Těmito doplňky je řízen výpočetní rozpočet, monitorován průběh algoritmu a detailně popsána geometrická reprezentace optimalizované trajektorie, aby byla reflektována specifika řešené scény, jež v původním MODDE nebyla předpokládána.

9.1 Trénovací konfigurace

Experimentální část zahrnovala několik testovacích scénářů, jejichž cílem bylo nalézt efektivní konfiguraci parametrů algoritmu vzhledem k zadaným optimalizačním problémům. V rámci hodnocení strategie ladění byly testovány dva protikladné přístupy. První z nich, označený jako Individuální konfigurace, se snaží nalézt pro každou scénu samostatnou sadu parametrů optimalizačního algoritmu tak, aby co nejlépe zohledňovala její unikátní geometrické vlastnosti, omezení prostoru a specifickou strukturu zakázaných zón. Výsledkem je vysoce specializované nastavení, které bývá schopno dosáhnout výrazně lepších hodnot cílové funkce, avšak za cenu větší výpočetní náročnosti, protože proces hledání se musí opakovat pro každou scénu zvlášť. Oproti tomu globální konfigurace usiluje o nalezení jediné univerzální kombinace parametrů, která poskytuje co nejlepší kompromisní výkon napříč celou sadou testovacích scén. Takový postup klade důraz na robustnost generalizaci – i když nemusí dosahovat absolutně nejlepšího řešení na každé individuální úloze, minimalizuje riziko výrazných propadů kvality při změně prostředí, a navíc snižuje režii spojenou s opakovaným laděním.

Oba přístupy byly podrobeny soustavě experimentů s pevně stanovenými parametry. Každý běh optimalizačního algoritmu byl omezen na maximálně 50 000 vyhodnocení cílové funkce, a to bez ohledu na typ konfigurace, aby bylo možné výsledky spravedlivě porovnat. Testování proběhlo na všech scénách zahrnutých do experimentu. Pro zachování statistické významnosti byly navíc všechny konfigurace spouštěny paralelně ve 20 nezávislých instancích na každé z 5 scén, s nastavením celkem 6 000 samostatných běhů. Takto rozsáhlý design umožňuje nejen vyhodnotit průměr kvality řešení, ale také analyzovat rozptyl a odolnost vůči náhodným fluktuacím ve vyhledávacím prostoru, čímž poskytuje ucelený pohled na výkonnost obou konfiguračních strategií.

9.2 Testovací konfigurace – individuální sady parametrů

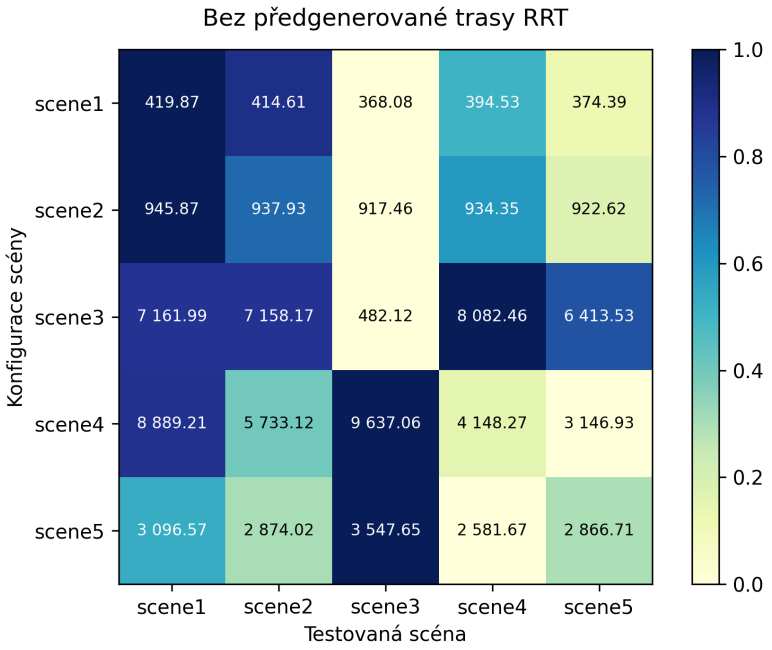
Cílem experimentu bylo ověřit, jak se budou parametry, naladěné samostatně pro každou ze scén `scene1` až `scene5`, chovat při nasazení do odlišných prostředí, a zároveň zjistit, zda předgenerovaná RRT významně přispěje ke zvýšení robustnosti nalezené trasy. Pro každou ze zmiňovaných scén byla za pomoci vyvinutého frameworku spuštěna optimalizace parametrů s pevným maximálním rozpočtem 50 000 volání hodnoticí funkce. Trénovací i validační instance byly ve všech případech vybírány výhradně z téže scény, aby se laděné parametry co nejvíce přizpůsobily jejím specifickým vlastnostem a bylo možné přesně zhodnotit přenositelnost těchto parametrů do jiných prostředí. Nalezené konfigurace jednotlivých scén jsou v tab. 2.

Tab. 2: Parametry pro jednotlivé scény.

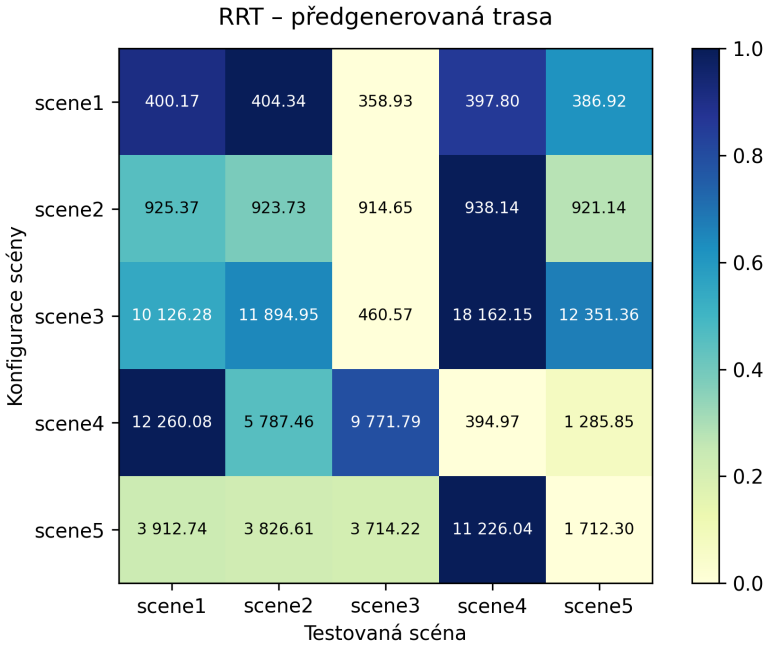
Parametr	scéna 1	scéna 2	scéna 3	scéna 4	scéna 5
ID	72	50	65	7	68
F	0.2135	0.2566	0.4036	0.3535	0.1986
CR	0.8989	0.9402	0.8367	0.9347	0.7734
mutation_base	rand	best	rand	target	best
pop_size	273	479	123	935	548
budget	50000	50000	50000	50000	50000
lambda	26	5	21	38	42
bound_correction	saturate	saturate	saturate	saturate	saturate
mutation_reference	rand	rand	best	best	rand
mutation_n_comps	2	2	2	2	2
mutation_use_weighted_F	False	False	False	False	True
pbest_value	0.9781	0.951	0.513	0.6689	0.4489
crossover	bin	exp	exp	bin	exp
eigenvalue_crossover	True	True	False	True	True
oppositional_generation_probability	0.303	0.7931	0.4534	0.7103	0.4267
oversampling_factor	0.6713	0.5325	0.5865	0.1767	0.4072
oppositional_initialization	True	False	False	False	True
init_stats	False	False	False	False	False
tangent_size	4	4	4	2	2
divided_by	9	11	5	7	7
num_spline_points	1000	1000	1000	1000	1000
penalty_value	145	145	145	145	145

Po ukončení tréninku byla každá nalezená konfigurace otestována na všech scénách pomocí pevného rozpočtu 5 000 volání hodnoticí funkce. Tím vznikly dvě 5×5 matice (viz obr. 14a a obr. 14b), kde prvek (i, j) udává dosaženou průměrnou délku cesty konfigurace θ_i při testu na scéně S_j. Nejnižších hodnot (tedy nejkratších tras) bylo dosaženo téměř výhradně na diagonále obou matic, což znamená, že každá metoda si vede nejlépe právě na scéně, pro kterou byla natrénována. V praxi tak dochází k výrazné neuniverzálnosti naladěných parametrů. Vyjímka tohoto se objevila v případě konfigurace vytrénované na složitější scéně `scene4` a `scene5`, která ve své variantě překvapivě předčila ostatní konfigurace na `scene2`. Důvodem je vyšší komplexnost scén 3 až 5, které již obsahují charakteristické křivky z předchozích scén, algoritmus se tedy lépe dostává z lokálních minim v okolí zakázaných zón. Zavedení předgenerované RRT se výrazně projevilo u složitějších scén `scene3` a `scene5`. V těchto případech došlo ke zkrácení trasy o 10–20 % oproti základní diagonále, jak je zřejmé z tmavších polí v pravé matici. Na jednodušších

scénách byl vliv zanedbatelný. Žádná z testovaných konfigurací nepřinesla konzistentně lepší výsledky napříč všemi scénami. To potvrzuje hypotézu, že globální, univerzální sada parametrů zde neexistuje a je třeba uvažovat globální scénově specifické ladění.



(a) Individuální porovnání bez předgenerovaného RRT.



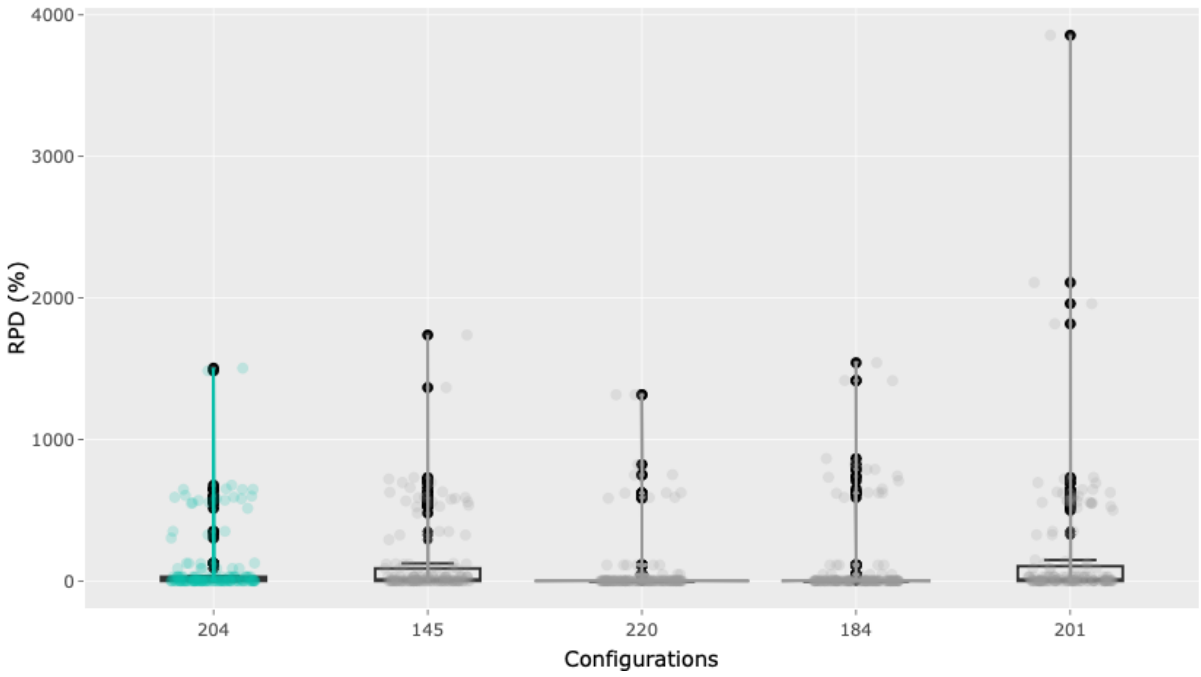
(b) Individuální porovnání s předgenerovanou RRT.

Obr. 14: Porovnání průměrné délky nalezených tras.

Experiment potvrdil, že parametry doladěné pro konkrétní scénu sice dosahují špičkového výkonu, mimo ni se však hůře uplatňují. Předgenerovaná RRT je největším přínosem tam, kde je vyhledávací prostor zvlášť složitý. Současně se ukázalo, že v rámci zvoleného postupu je nereálné nalézt jedinou konfiguraci, která by spolehlivě fungovala ve všech prostředích.

9.3 Testovací konfigurace – Globální sada parametrů

Předchozí kapitola dokazuje, že neexistuje jediná universální sada parametrů, která by spolehlivě fungovala ve všech scénách. Z toho vyplývá nutnost ladění buď scénově specifického, nebo globálního scénově robustního nastavení. V této části proto vyhodnocujeme experiment zaměřený právě na hledání takové globální konfigurace. Nejprve jsme vygenerovali trénovací i validační instance pro všech pět scén. Irace dostal 20 počátečních konfigurací s rozpočtem 5 000 volání hodnoticí funkce. Po vyčerpání rozpočtu zůstalo v elitním clusteru pět nejlepších konfigurací, které prošly ještě jedním nezávislým, univerzálním testem na stejné sadě instancí.



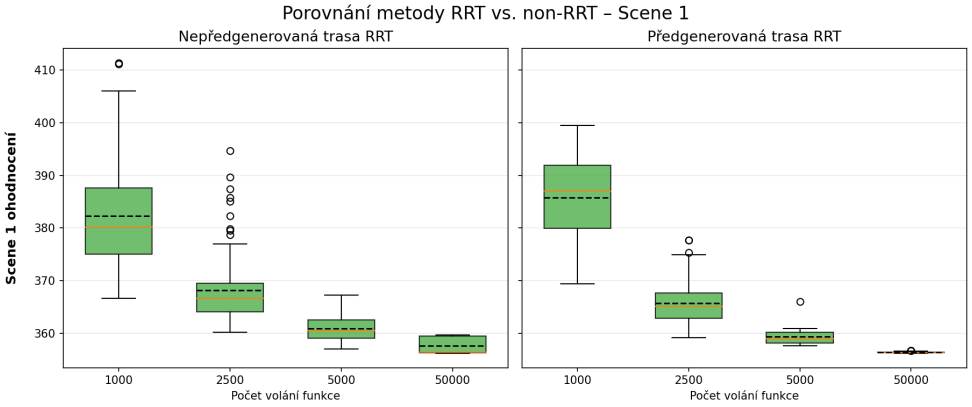
Obr. 15: Elitní volba Irace konfigurace 204.

Obr. 15 prezentuje rozdělení dosažených nákladů pro všech pět elitních konfigurací s ohledem na PRD (Percentage of Relative Difference), přičemž konfigurace 204 vykazuje nejen nejnižší medián, ale i nejmenší rozptyl výsledků, což dokládá její vysokou míru robustnosti napříč testovanými scénami. Výsledkem tohoto finálního srovnání je volba konfigurace 204, kde podrobný přehled hodnot parametrů je uveden v tab. 3.

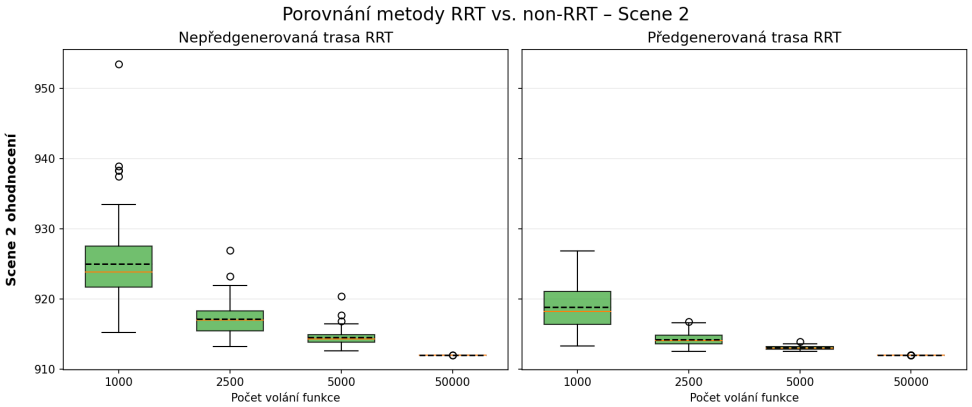
Tab. 3: Výsledné parametry elitní konfigurace Irace

Parametr	Hodnota
ID	204
F	0.3328
CR	0.9311
mutation_base	best
pop_size	164
budget	50000
lambda_	16
bound_correction	saturate
mutation_reference	best
mutation_n_comps	2
mutation_use_weighted_F	True
pbest_value	0.373
crossover	exp
eigenvalue_crossover	True
oppositional_generation_probability	0.2533
oversampling_factor	0.2908
oppositional_initialization	False
init_stats	False
tangent_size	2
divided_by	8
num_spline_points	1000
penalty_value	145

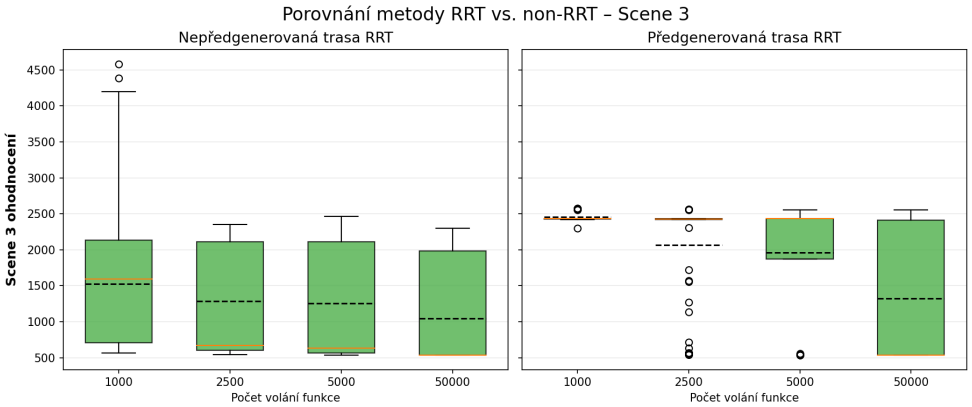
Při rozpočtu 5 000 volání hodnoticí funkce překonává tato jednotná sada parametrů všechny dříve naladěné scénově specifické varianty, a proto byla zvolena jako výchozí konfigurace pro následné experimenty. V další fázi bylo zkoumáno, zda předgenerování RRT stromu dokáže zlepšit kvalitu plánování při různých omezeních na počet volání: byly testovány rozpočty 1 000, 2 500, 5 000 a 50 000 volání hodnoticí funkce a porovnali dvě strategie plánování – s předgenerovanou RRT trasou a bez ní. Souhrn výsledků obou přístupů pro jednotlivé scény je zobrazen na obr. 16–obr. 20, které porovnávají chování bez předgenerované RRT a s využitím RRT struktury.



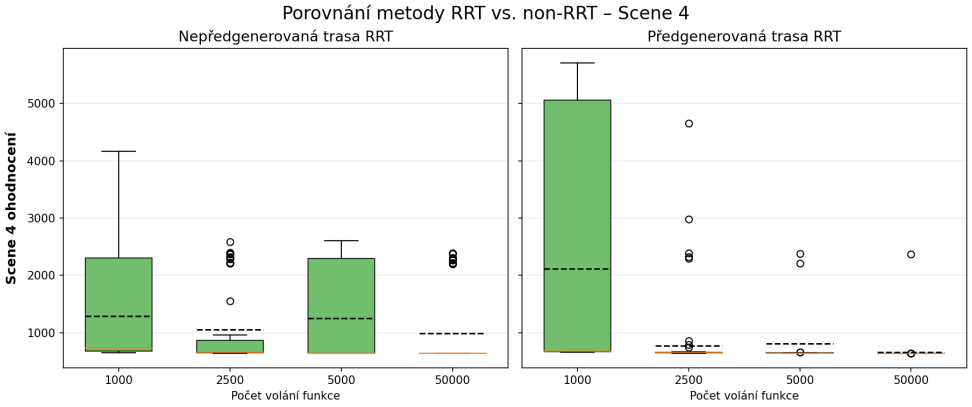
Obr. 16: Porovnání výsledků – Scéna 1.



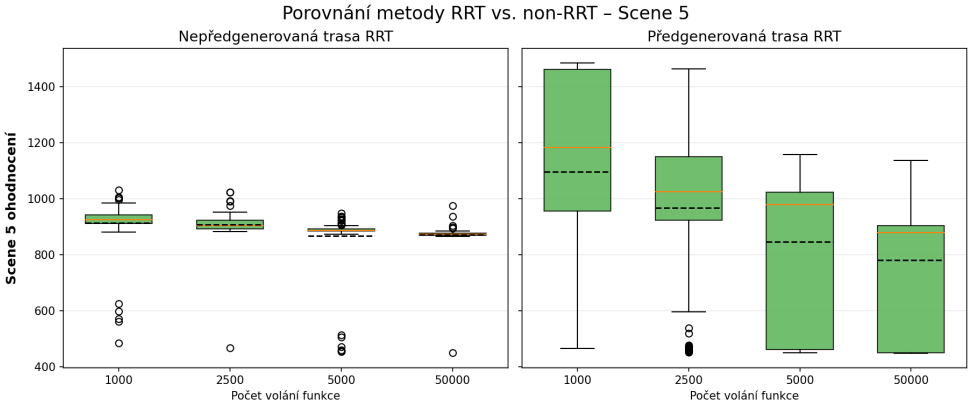
Obr. 17: Porovnání výsledků – Scéna 2.



Obr. 18: Porovnání výsledků – Scéna 3.



Obr. 19: Porovnání výsledků – Scéna 4.



Obr. 20: Porovnání výsledků – Scéna 5.

Z box-plotů je patrné, že předgenerovaná RRT přináší výhodu ve všech scénách i testovaných rozpočtech. Efekt je nejmarkantnější u nízkých rozpočtů (1 000 a 2 500 volání), které simulují potřebu rychlého nalezení přijatelné cesty v situacích, kdy se scéna rychle mění a aplikace běží v reálném čase. Například ve scéně 4 klesá medián nákladů zhruba na třetinu a rozptyl výsledků se výrazně zmenší. Se zvyšujícím se rozpočtem rozdíl mezi oběma metodami klesá, nicméně i při 50 000 voláních – což odpovídá scénářům, kde nevadí delší výpočet a cílem je co nejlepší trasa – vykazuje varianta s RRT stále nižší medián i menší počet extrémních odlehlých hodnot.

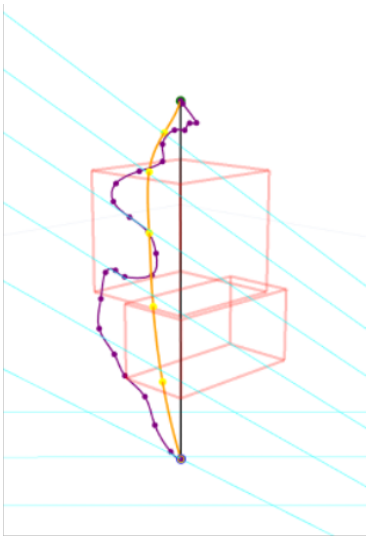
Při pohledu na jednotlivé scény lze vysledovat, že ve scénách 1 a 2, které jsou relativně jednoduché, je rozdíl menší, ale stále měřitelný. Naopak v scénách 3–5, jež obsahují úzká místa a větší počet překážek, se předgenerovaná RRT prosazuje výrazněji. To naznačuje, že metoda umí dobře využít předpočítanou strukturu pro rychlejší průnik do složitějších prostorů konfigurací.

9.4 Zhodnocení výsledků

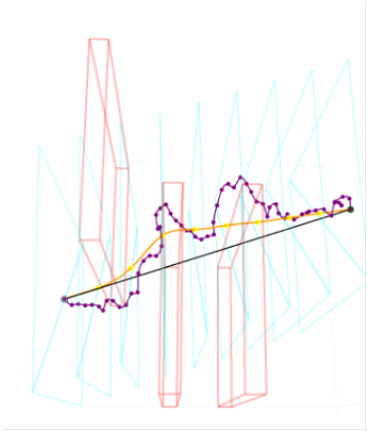
Z obr. 21 je zřejmé, že s jedinou globální konfigurací a předem vygenerovaným stromem RRT dokázal systém najít kvalitní trajektorie ve všech pěti scénách. Ve všech případech se nalezené dráhy zcela vyhnuly kolizním oblastem a zároveň minimalizovaly délku i dobu průchodu – což bylo jedním z hlavních cílů návrhu algoritmu. Přestože se jednotlivé scény liší hustotou překážek a topologií volného prostoru, výsledné cesty jsou konzistentně hladké, bez zbytečných oscilací a se zachovanou bezpečnostní rezervou vůči překážkám.

Konfigurační globální sada se tak ukazuje jako dostatečně robustní pro celé testované portfolio scén. Z hlediska optimalizace robotických tras to znamená, že navržené řešení kombinuje globální optimalitu díky RRT heuristice s lokální jemností bez nutnosti ručního přeladění parametrů pro každou scénu. To v praxi snižuje náklady na nasazení do heterogenních provozů – stejný plánovač lze nahrát do flotily robotů operujících v logistice, výrobě i skladech s odlišnou geometrií. Navíc jednotná konfigurace zjednodušuje údržbu: při případných aktualizacích stačí validovat jediný balík parametrů a není nutné znovu kalibrovat každý dílčí scénář.

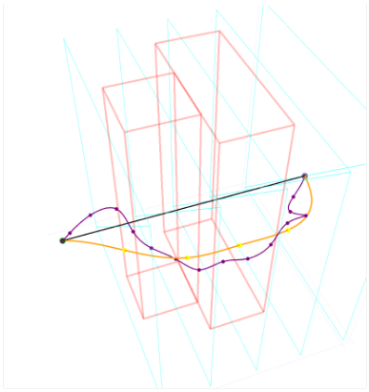
Celkově tedy platí, že spojením globální konfigurace 204 a předgenerovaného stromu RRT jsme dosáhli efektivního kompromisu mezi rychlostí výpočtu a kvalitou výsledné trasy. To potvrzuje, že je možné dosáhnout vysoké míry generalizace i bez scénově specifického ladění, což je v kontextu moderní robotické logistiky zásadní pro škálovatelnost a operativní flexibilitu. Další podrobnosti jsou k dispozici v příloze A, podsekci `tests`, ve složkách `Individual_data` a `Global_data`, které obsahují jednotlivé hodnoty pro porovnání a výsledné trasy v podsekci `results` .



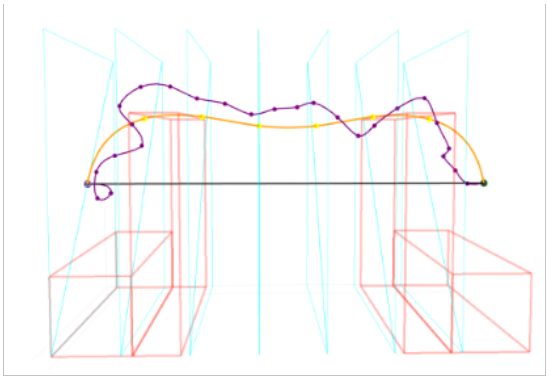
(a) Scéna 1.



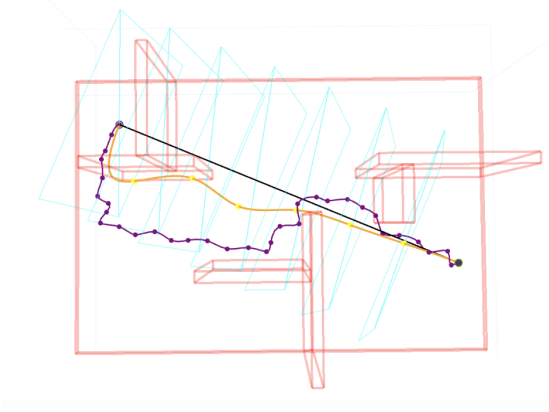
(b) Scéna 2.



(c) Scéna 3.



(d) Scéna 4.

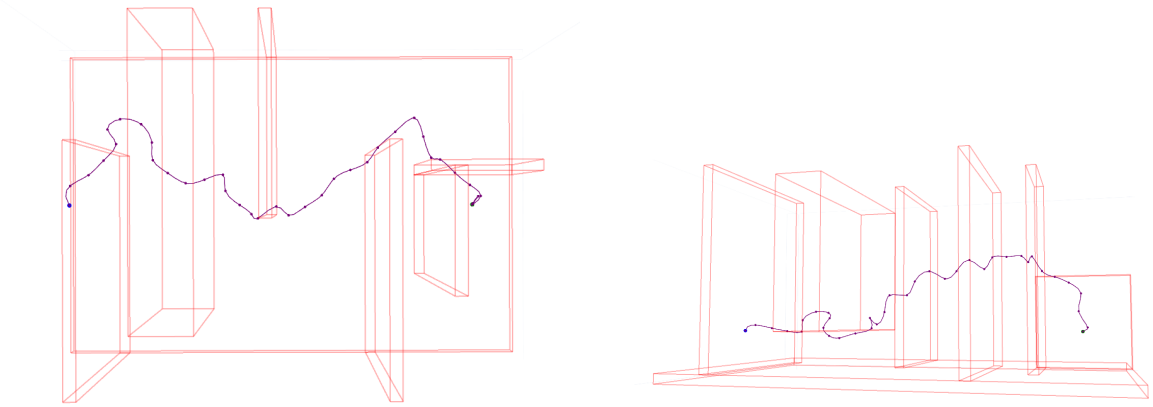


(e) Scéna 5.

Obr. 21: Nejlepší nalezené trajektorie pro všechny scény. Fialovou barvou je vyznačena trasa vygenerovaná metodou RRT, optimalizovaná trasa je zvýrazněna oranžově.

10 Srovnání konfigurace s alternativními metodami

Pro účely verifikace generalizačních schopností hlavní konfigurace, nalezené pomocí Irace a MODDE, byla připravena zcela nová robotická scéna. Tato scéna kombinuje klíčové vlastnosti všech trénovacích scén (překážky s prudkými hranami, úzké koridory a částečně otevřené prostory), na nichž se MODDE původně učil. Pohled na scénu je zobrazen na obr. 22. Cílem bylo vytvořit prostředí, v němž se projeví jak schopnost algoritmů vyhledat globálně výhodné trajektorie, tak i odolnost vůči lokálním minimům.



(a) Horní pohled na scénu.

(b) Boční pohled na scénu.

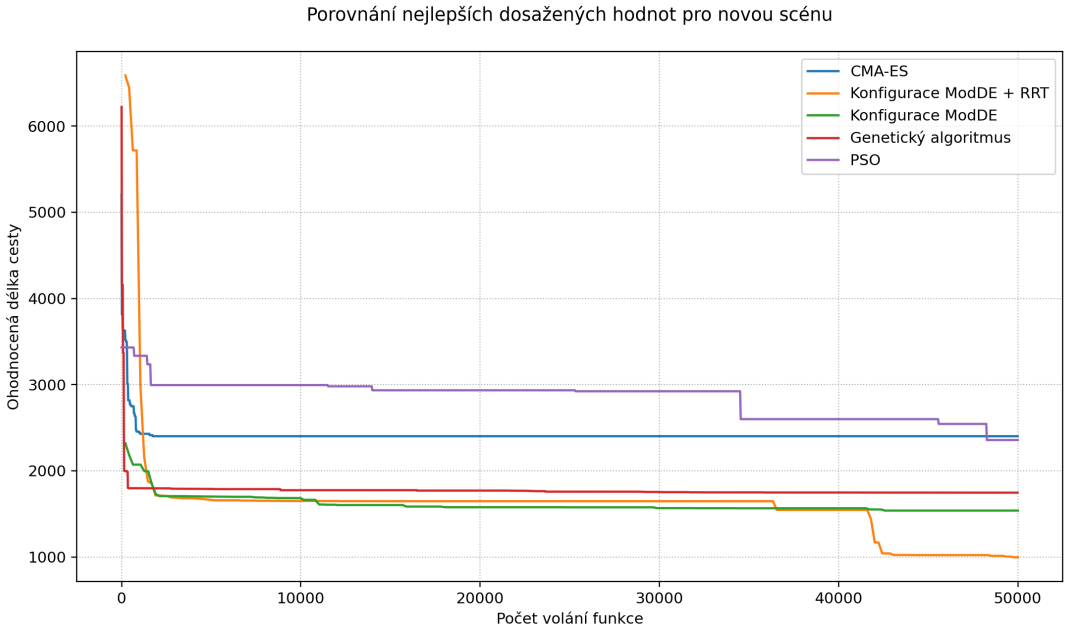
Obr. 22: Testovací scéna kombinující charakteristiky původních trénovacích prostředí.

V testu byly vedle finální konfigurace MODDE hodnoceny čtyři běžně používané globální optimalizační metody: Všichni kandidáti sdíleli jednotný rozpočet 50 000 volání fitness a stejný vyhledávací prostor. Neplatné trajektorie byly penalizovány, čímž se zabránilo předčasnému ukončení optimalizace a zároveň se zachovala srovnatelnost výsledků.

Algoritmus CMA-ES byl realizován pomocí knihovny pycma. Velikost populace byla nastavena na $\lambda = 50$, což odpovídá 1 000 generacím při uvedeném rozpočtu. Počáteční globální rozptyl byl $\sigma_0 = 0.1$. Parametry interní adaptace (konstanty pro aktualizaci kovarianční matice a krokového faktoru) zůstaly na výchozích hodnotách doporučených autory algoritmu. Restartovací strategie byla vypnuta, aby CMA-ES využil přesně vymezený počet vyhodnocení.

GA vycházel z knihovny DEAP a používal real-coded reprezentaci. Populace obsahovala 50 jedinců, křížení probíhalo operátorem cxblend, vhodným pro spojité optimalizační úlohy s parametrem $\alpha = 0.5$ a pravděpodobností $p_c = 0.9$. Mutace byla realizována Gaussovským operátorem se směrodatnou odchylkou $\sigma = 0.05$ a pravděpodobností $p_m = 0.2$. Selekce probíhala turnajem velikosti tři. Protože se v každé generaci vyhodnocuje právě padesát potomků, odpovídá daný rozpočet celkem 999 generacím včetně úvodního vyhodnocení startovní populace.

PSO byl implementován jako globální varianta gbest z knihovny pyswarms. Parametry PSO byly zvoleny podle konvergenční analýzy Clerca a Kennedyho [65], která zaručuje stabilitu a kontrakci hejna. Použité hodnoty $c_1 = c_2 = 1.496$ a $w = 0.7298$ odpovídají tzv. constriction factor PSO, který matematicky omezuje růst rychlosti částic a tím zajišťuje konvergenci k atraktoru v prostoru řešení. Rychlosti částic byly omezeny na polovinu délky vyhledávacího intervalu v každé dimenzi, aby se předcházelo odletům mimo dovolené hranice. Jedna iterace PSO vyžaduje dvě vyhodnocení fitness na částici, a proto byl proveden maximálně stanovený počet iterací, aby přesně naplnil povolený počet funkčních volání vyhodnocení.

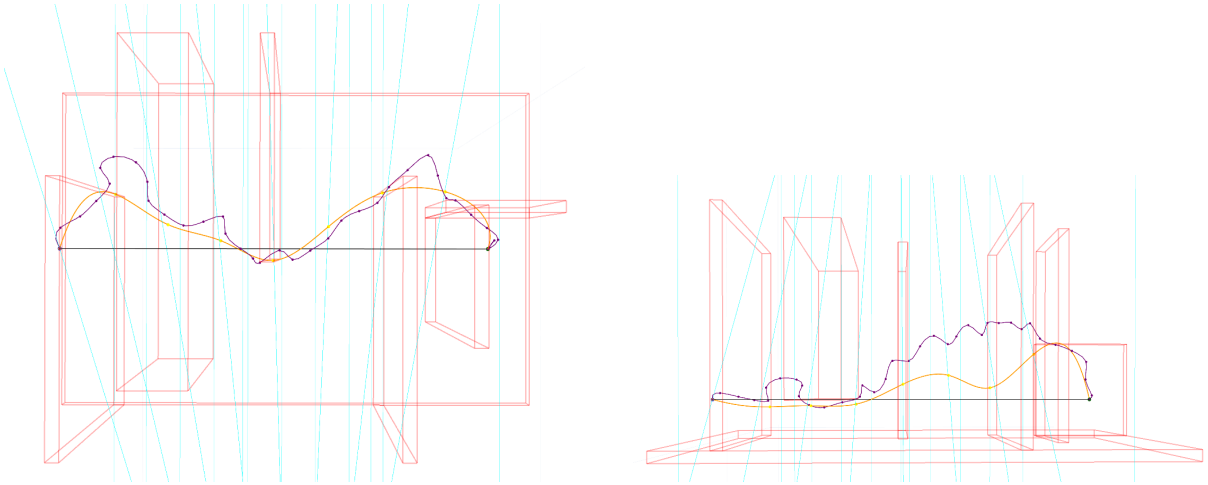


Obr. 23: Průběh nejlepší dosažené hodnoty fitness v závislosti na počtu vyhodnocení.

Z výsledků na obr. 23 je patrné, že kombinace MODDE + RRT (oranžová křivka) dosahuje nejnižší penalizované délky cesty po přibližně 45 000 evaluacích, zatímco čistý MODDE (zelená) konverguje k lokálnímu minimu s mírně horší hodnotou. CMA-ES (modrá) a GA (červená) rychle klesají zpočátku, avšak uvíznou ve vyšším minimu. PSO (fialová) konverguje nejpomaleji a je tedy ovlivněn topologií scény.

10.1 Výsledky a diskuze

Přidání předgenerované trajektorie prostřednictvím RRT zrychlilo konvergenci MODDE a umožnilo prozkoumávat globální strukturou scény dříve, než došlo k jemné optimalizaci tvaru spline. CMA-ES těžil z adaptivní kovarianční matice, nicméně byl limitován pevnou velikostí populace, což v pozdní fázi snižovalo diverzitu. GA vykazoval stabilní, avšak pomalý pokles; jeho výkon byl nejcitlivější na nastavení pravděpodobností křížení a mutace. PSO trpěl na rojení v úzkých pasážích a potřeboval více iterací k úniku z lokálních minim. Výsledky tohoto experimentu ukazují, že přidání předgenerované trajektorie pomocí RRT výrazně zvyšuje efektivitu plánování. Kombinace RRT a MODDE dosahuje nejnižší penalizované délky dráhy a zároveň vykazuje nejrychlejší konvergenci v počtu vyhodnocení fitness. Výsledná trasa je zobrazena na obr. 24.



(a) Pohled shora.

(b) Pohled z boku.

Obr. 24: Nejlepší nalezená trajektorie pro novou scénu. Fialovou barvou je vyznačena trasa vygenerovaná metodou RRT, optimalizovaná trasa je zvýrazněna oranžově.

Tento efekt podporují i výsledky jiných prací. Například v práci [66] navrhují hybridní strategii, kde kombinují PSO s heuristickým plánovačem Dynamic Window Approach (DWA). DWA slouží jako rychlý generátor kolizně volných trajektorií, zatímco PSO slouží k jemnému dolaďování parametrů pohybu. Autoři ukazují, že bez této hybridizace zůstává PSO náchylné k uvíznutí v lokálních minimech, především v komplikovaných scénách s úzkými průchody, kde PSO bez jakékoliv naváděcí pomoci konverguje nejpomaleji a trpí na předčasnou stagnaci. Jednotlivé hodnoty jsou dostupné v příloze A, podsekci `tests`, ve složce `Comparison_data`. Výsledná trasa pro scénu 6 je v podsekci `results` .

Ještě detailnější potvrzení tohoto principu přináší práce [67], kde autoři představují tzv. Multi-Strategy Fusion RRT (MSF-RRT). Tato metoda kombinuje několik zlepšovacích kroků nad klasickým RRT, včetně adaptivní délky kroku, biasovaného vzorkování

směrem k cíli a následné post-processing fáze, kde je původní hrubá trasa převedena na hladkou křivku pomocí spline interpolace. Tento dvoufázový přístup je téměř identický rešené problematice v diplomové práci.

Důležitým rozdílem oproti oběma zmíněným studiím je však to, že zvolený přístup zůstává plně parametrický a není vázán na žádnou konkrétní robotickou platformu. Na základě těchto pozorování lze říci, že hybridní plánovací strategie, ve kterých heuristický algoritmus poskytuje rychlou, byť hrubou aproximaci trasy a evoluční optimalizátor tuto trasu následně dolaďuje, jsou v prostředích s bohatou topologií značně výhodné. Ve zvoleném experimentu funguje RRT jako naváděcí pomocník, který ulehčuje MODDE počáteční globální orientaci v prostoru a tím zvyšuje efektivitu využití rozpočtu fitness.

Zároveň je patrné, že čisté evoluční metody jako CMA-ES nebo GA mají problémy s udržením diverzity v pozdějších generacích a často uvíznou v suboptimálních řešeních, obzvlášť pokud nejsou doplněny žádnou doménovou znalostí nebo heuristikou. To potvrzuje, že i v prostředí, kde se optimalizuje bez explicitního modelu dynamiky, je vhodné využívat hybridní strategie využívající výhod obou přístupů.

11 ZÁVĚR

Hlavním cílem této diplomové práce bylo experimentálně ověřit, zda je možné prostřednictvím MODDE a automatizovaného ladění parametrů efektivně nalézt univerzální konfiguraci algoritmu, která zajistí rychlé, bezpečné a kvalitní plánování robotických trajektorií napříč různorodými a komplexními scénami. K tomuto účelu byl navržen modulární výpočetní framework, jehož jádro tvoří implementace MODDE propojená s nástrojem pro automatické ladění Irace. Framework byl realizován formou sady kontejnerizovaných služeb, které byly následně orchestrálně řízeny prostřednictvím platformy Kubernetes. Tento přístup umožnil efektivní paralelizaci výpočtů a škálování na clusteru MicroK8s, což výrazně zkrátilo dobu potřebnou pro provedení komplexního autotuningu.

V rámci experimentálního ověření byly pomocí frameworku Irace systematicky prohledány konfigurace kombinací modulů a parametrů EA. Výsledky tohoto rozsáhlého experimentálního procesu ukázaly, že nejlépe nalezená globální konfigurace dosahuje ve všech pěti testovacích scénách kratší délky plánované trajektorie. Tato konfigurace byla navíc schopna konzistentně nalézt kvalitní řešení bez nutnosti individuálního ladění parametrů pro konkrétní scénu, což potvrzuje její univerzálnost a robustnost v praxi.

Dalším významným přínosem práce bylo ověření efektivity zařazení předpočítané trajektorie získané pomocí algoritmu RRT jako počáteční populace v evolučním procesu. Tento postup vedl k podstatnému urychlení konvergence EA. Ve složitějších scénách, kde standardní evoluční přístup vykazoval pomalejší konvergenci, snížilo využití předpočítané trajektorie výslednou délku plánované dráhy.

Framework MODDE byl úspěšně navržen a implementován v programovacím jazyce Python, kontejnerizován pro provoz v cloudovém clusteru a propojen s nástrojem Irace, což umožnilo automatizovaný a paralelní autotuning konfigurací EA. Experimentální ověření proběhlo na pěti scénách různé obtížnosti. Výsledky jednoznačně ukazují, že je možné nalézt globální konfiguraci, která je dostatečně univerzální pro heterogenní scény, pokud je podpořena vhodnou inicializací pomocí předpočítaného sampling-based algoritmu. Účinnost této konfigurace byla dále potvrzena na nově vytvořené kombinované testovací scéně a na srovnávacím experimentu. Zde se prokázalo, že zařazení předgenerované trajektorie získané algoritmem RRT jako počáteční populace urychluje konvergenci MODDE fitness a zkracuje výslednou dráhu ve složitých scénách. Hybridní varianta RRT + MODDE zároveň překonala CMA-ES, GA a PSO, které trpěly buď úbytkem diverzity, nebo uvíznutím v lokálních minimech, což koresponduje se závěry nedávných studií zaměřených na propojení heuristických a evolučních metod.

Pro další vývoj se nabízí řada perspektivních směrů a rozšíření vytvořeného řešení. Jedním z nich je implementace dynamického přeladění parametrů v reálném čase, což by umožnilo algoritmu efektivně reagovat na změny prostředí či neočekávané překážky. Další zajímavou možností rozvoje je integrace vícekriteriální optimalizace, například současné

minimalizace spotřeby energie a mechanického opotřebení robota, což by výrazně zvýšilo praktickou použitelnost systému. Perspektivní je rovněž doplnění frameworku o pokročilé simulační nástroje s reálnou fyzikou, které by umožnily přesnější plánování trajektorií reflektujících skutečné dynamické vlastnosti robotických systémů. Poslední významnou oblastí budoucího výzkumu je využití metod transfer learningu pro přenos konfigurací a modelů optimalizovaných ve virtuálním prostředí přímo na fyzické robotické platformy, což by dále usnadnilo přechod navrženého řešení z akademického výzkumu do průmyslové praxe.

SEZNAM POUŽITÉ LITERATURY

[1] LaTorre, A.; Molina, D.; Osaba, E.; Poyatos, J.; Del Ser, J. et al. A Prescription of Methodological Guidelines for Comparing Bio-Inspired Optimization Algorithms. *Swarm and Evolutionary Computation*, prosinec 2021, sv. 67, s. 100973. ISSN 2210-6502.

[2] Jade, C. *Evolutionary Optimization: A Review and Implementation of Several Algorithms*. Dostupné z: https://www.strong.io/blog/evolutionary-optimization.

[3] Katoch, S.; Chauhan, S. S. a Kumar, V. A Review on Genetic Algorithm: Past, Present, and Future, sv. 80, č. 5, s. 8091–8126. ISSN 1380-7501, 1573-7721. Dostupné z: https://link.springer.com/10.1007/s11042-020-10139-6.

[4] Tanabe, R. a Fukunaga, A. Success-History Based Parameter Adaptation for Differential Evolution. In: *2013 IEEE Congress on Evolutionary Computation*. IEEE, s. 71–78. ISBN 978-1-4799-0454-9 978-1-4799-0453-2 978-1-4799-0451-8 978-1-4799-0452-5. Dostupné z: http://ieeexplore.ieee.org/document/6557555/.

[5] Londe, M. A.; Pessoa, L. S.; Andrade, C. E. a Resende, M. G. C. *Biased Random-Key Genetic Algorithms: A Review*. arXiv. Dostupné z: https://arxiv.org/abs/2312.00961.

[6] Alexakis, K.; Benekis, V.; Kokkinakos, P. a Askounis, D. Genetic Algorithm-Based Multi-Objective Optimisation for Energy-Efficient Building Retrofitting: A Systematic Review, sv. 328, s. 115216. ISSN 0378-7788. Dostupné z: https://www.sciencedirect.com/science/article/pii/S037877882401332X.

[7] He, C.; Zhang, Y.; Gong, D. a Ji, X. A Review of Surrogate-Assisted Evolutionary Algorithms for Expensive Optimization Problems, sv. 217, s. 119495. ISSN 0957-4174. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0957417422025143.

[8] Tian, H.; Yuan, H.; Yan, K. a Guo, J. A Cosine Adaptive Particle Swarm Optimization Based Long-Short Term Memory Method for Urban Green Area Prediction, sv. 10, s. e2048. ISSN 2376-5992. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC11157564/.

[9] Yao, J.; Luo, X.; Li, F.; Dou, J. a Luo, H. *Research on Hybrid Strategy Particle Swarm Optimization Algorithm and Its Applications*. Dostupné z: https://www.researchsquare.com/article/rs-4160937/v1.

[10] Lalwani, S.; Singhal, S.; Kumar, R. a Gupta, N. A Comprehensive Survey: Applications of Multi-Objective Particle Swarm Optimization (MOPSO) Algorithm. University of Isfahan, sv. 2, č. 1, s. 39–101. ISSN 2251-8657, 2251-8665. Dostupné z: http://www.combinatorics.ir/?_action=showPDF&article=2834&_ob=2534709aaead83732828c7103a34a4a6&fileName=full_text.pdf.

[11] Fang, J.; Liu, W.; Chen, L.; Lauria, S.; Miron, A. et al. A Survey of Algorithms, Applications and Trends for Particle Swarm Optimization, s. 24–50. ISSN 2653-6226. Dostupné z: https://www.sciltp.com/journals/ijndi/2023/1/176.

[12] Hansen, N. a Ostermeier, A. Completely Derandomized Self-Adaptation in Evolution Strategies, sv. 9, č. 2, s. 159–195. ISSN 1063-6560, 1530-9304. Dostupné z: https://direct.mit.edu/evco/article/9/2/159-195/892.

[13] Atamna, A.; Auger, A. a Hansen, N. Augmented Lagrangian Constraint Handling for CMA-ES — Case of a Single Linear Constraint. In: Handl, J.; Hart, E.; Lewis, P. R.; López Ibáñez, M.; Ochoa, G. et al., ed. *Parallel Problem Solving from Nature – PPSN XIV*. Springer International Publishing, sv. 9921, s. 181–191. ISBN 978-3-319-45822-9 978-3-319-45823-6. Dostupné z: http://link.springer.com/10.1007/978-3-319-45823-6_17.

[14] Akimoto, Y.; Auger, A. a Hansen, N. Comparison-Based Natural Gradient Optimization in High Dimension. In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, s. 373–380. ISBN 978-1-4503-2662-9. Dostupné z: https://dl.acm.org/doi/10.1145/2576768.2598258.

[15] Brest, J.; Maucec, M. S. a Boskovic, B. Differential Evolution Algorithm for Single Objective Bound-Constrained Optimization: Algorithm J2020. In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, s. 1–8. ISBN 978-1-7281-6929-3. Dostupné z: https://ieeexplore.ieee.org/document/9185551/.

[16] Brest, J. a Sepesy Maučec, M. Comparative Study of Modern Differential Evolution Algorithms: Perspectives on Mechanisms and Performance, sv. 13, č. 10, s. 1556. ISSN 2227-7390. Dostupné z: https://www.mdpi.com/2227-7390/13/10/1556.

[17] Ahmad, M. F.; Isa, N. A. M.; Lim, W. H. a Ang, K. M. Differential Evolution: A Recent Review Based on State-of-the-Art Works, sv. 61, č. 5, s. 3831–3872. ISSN 1110-0168. Dostupné z: https://www.sciencedirect.com/science/article/pii/S111001682100613X.

[18] Vermetten, D.; Caraffini, F.; Kononova, A. V. a Bäck, T. Modular Differential Evolution. arXiv, duben 2023, arXiv:2304.09524. Dostupné z: https://arxiv.org/abs/2304.09524.

[19] Nobel, J. de; Vermetten, D.; Wang, H.; Doerr, C. a Bäck, T. *Tuning as a Means of Assessing the Benefits of New Ideas in Interplay with Existing Algorithmic Modules*. arXiv, únor 2021. Dostupné z: https://arxiv.org/abs/2102.12905.

[20] Kostovska, A.; Vermetten, D.; Džeroski, S.; Doerr, C.; Korošec, P. et al. *The Importance of Landscape Features for Performance Prediction of Modular CMA-ES Variants*. Dostupné z: https://arxiv.org/abs/2204.07431.

[21] Loshchilov, I. *A Computationally Efficient Limited Memory CMA-ES for Large Scale Optimization*. Dostupné z: https://arxiv.org/abs/1404.5520.

[22] Nikolikj, A.; Kostovska, A.; Vermetten, D.; Doerr, C. a Eftimov, T. *Quantifying Individual and Joint Module Impact in Modular Optimization Frameworks*. Dostupné z: https://arxiv.org/abs/2405.11964.

[23] Camacho Villalón, C. L.; Stützle, T. a Dorigo, M. Designing New Metaheuristics: Manual Versus Automatic Approaches. American Association for the Advancement of Science, sv. 2, s. 0048. Dostupné z: https://spj.science.org/doi/10.34133/icomputing.0048.

[24] Miranda, P. B. a Prudêncio, R. B. Generation of Particle Swarm Optimization Algorithms: An Experimental Study Using Grammar-Guided Genetic Programming, sv. 60, s. 281–296. ISSN 15684946. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S1568494617303836.

[25] Camacho-Villalon, C. L.; Dorigo, M. a Stutzle, T. PSO-X: A Component-Based Framework for the Automatic Design of Particle Swarm Optimization Algorithms. *IEEE Transactions on Evolutionary Computation*, červen 2022, sv. 26, č. 3, s. 402–416. ISSN 1089-778X, 1089-778X, 1941-0026.

[26] Juříček, M.; Parák, R. a Kůdela, J. Evolutionary Computation Techniques for Path Planning Problems in Industrial Robotics: A State-of-the-Art Review. Multidisciplinary Digital Publishing Institute, sv. 11, č. 12, s. 245. ISSN 2079-3197. Dostupné z: https://www.mdpi.com/2079-3197/11/12/245.

[27] Kůdela, J.; Juříček, M. a Parák, R. A Collection of Robotics Problems for Benchmarking Evolutionary Computation Methods. In: Correia, J.; Smith, S. a Qaddoura, R., ed. *Applications of Evolutionary Computation*. Springer Nature Switzerland, sv. 13989, s. 364–379. ISBN 978-3-031-30228-2 978-3-031-30229-9. Dostupné z: https://link.springer.com/10.1007/978-3-031-30229-9_24.

[28] Doncieux, S.; Bredeche, N.; Mouret, J.-B. a Eiben, A. E. G. Evolutionary Robotics: What, Why, and Where To. Frontiers, sv. 2. ISSN 2296-9144. Dostupné z: https://www.frontiersin.orghttps://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2015.00004/full.

[29] Doncieux, S.; Bredeche, N.; Mouret, J.-B. a Eiben, A. E. G. Evolutionary Robotics: What, Why, and Where To. Frontiers, sv. 2. ISSN 2296-9144. Dostupné z: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2015.00004/full.

[30] Starke, S.; Hendrich, N.; Magg, S. a Zhang, J. An Efficient Hybridization of Genetic Algorithms and Particle Swarm Optimization for Inverse Kinematics. In: *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, s. 1782–1789. ISBN 978-1-5090-4364-4. Dostupné z: http://ieeexplore.ieee.org/document/7866587/.

[31] Zucconi, A. *Inverse Kinematics in 2D - Part 1.* Dostupné z: https://www.alanzucconi.com/2018/05/02/ik-2d-1/.

[32] Chen, X.; Zhang, Q. a Sun, Y. Evolutionary Robot Calibration and Nonlinear Compensation Methodology Based on GA-DNN and an Extra Compliance Error Model, sv. 2020, s. 1–15. ISSN 1024-123X, 1563-5147. Dostupné z: https://www.hindawi.com/journals/mpe/2020/3981081/.

[33] Larsen, L.; Schuster, A.; Kim, J. a Kupke, M. Path Planning of Cooperating Industrial Robots Using Evolutionary Algorithms, sv. 17, s. 286–293. ISSN 23519789. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S2351978918311648.

[34] Billeschou, P.; Bijma, N. N.; Larsen, L. B.; Gorb, S. N.; Larsen, J. C. et al. Framework for Developing Bio-Inspired Morphologies for Walking Robots, sv. 10, č. 19, s. 6986. ISSN 2076-3417. Dostupné z: https://www.mdpi.com/2076-3417/10/19/6986.

[35] Zanchettin, A. M.; Messeri, C.; Cristantielli, D. a Rocco, P. Trajectory Optimisation in Collaborative Robotics Based on Simulations and Genetic Algorithms, sv. 6, č. 4, s. 707–723. ISSN 2366-5971, 2366-598X. Dostupné z: https://link.springer.com/10.1007/s41315-022-00240-4.

[36] Wang, Y.; Pandit, P.; Kandhari, A.; Liu, Z. a Daltorio, K. A. Rapidly Exploring Random Tree Algorithm-Based Path Planning for Worm-Like Robot. Multidisciplinary Digital Publishing Institute, sv. 5, č. 2, s. 26. ISSN 2313-7673. Dostupné z: https://www.mdpi.com/2313-7673/5/2/26.

[37] Xu, T. Recent Advances in Rapidly-exploring Random Tree: A Review, sv. 10, č. 11, s. e32451. ISSN 2405-8440. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2405844024084822.

[38] Kim, J.-O. a Khosla, P. Real-Time Obstacle Avoidance Using Harmonic Potential Functions, sv. 8, č. 3, s. 338–349. ISSN 1042296X. Dostupné z: http://ieeexplore.ieee.org/document/143352/.

[39] Seereeram, S. a Wen, J. A Global Approach to Path Planning for Redundant Manipulators, sv. 11, č. 1, s. 152–160. ISSN 1042296X. Dostupné z: http://ieeexplore.ieee.org/document/345948/.

[40] Liu, X.; Jiang, D.; Tao, B.; Jiang, G.; Sun, Y. et al. Genetic Algorithm-Based Trajectory Optimization for Digital Twin Robots. Frontiers, sv. 9. ISSN 2296-4185. Dostupné z: https://www.frontiersin.org/journals/bioengineering-and-biotechnology/articles/10.3389/fbioe.2021.793782/full.

[41] Wang, T.; Xin, Z.; Miao, H.; Zhang, H.; Chen, Z. et al. Optimal Trajectory Planning of Grinding Robot Based on Improved Whale Optimization Algorithm, sv. 2020, s. 1–8. ISSN 1024-123X, 1563-5147. Dostupné z: https://www.hindawi.com/journals/mpe/2020/3424313/.

[42] He, X.; Zhou, Y.; Liu, H. a Shang, W. Improved RRT*-Connect Manipulator Path Planning in a Multi-Obstacle Narrow Environment, sv. 25, č. 8, s. 2364. ISSN 1424-8220. Dostupné z: https://www.mdpi.com/1424-8220/25/8/2364.

[43] Karaman, S. a Frazzoli, E. *Sampling-Based Algorithms for Optimal Motion Planning*. Dostupné z: https://arxiv.org/abs/1105.1186.

[44] Elbanhawi, M. a Simic, M. Sampling-Based Robot Motion Planning: A Review, sv. 2, s. 56–77. ISSN 2169-3536. Dostupné z: https://ieeexplore.ieee.org/document/6722915.

[45] Muhsen, D. K.; Raheem, F. A. a Sadiq, A. T. A Systematic Review of Rapidly Exploring Random Tree RRT Algorithm for Single and Multiple Robots, sv. 24, č. 3, s. 78–101. ISSN 1314-4081. Dostupné z: https://www.sciendo.com/article/10.2478/cait-2024-0026.

[46] Primatesta, S.; Osman, A. a Rizzo, A. MP-RRT#: A Model Predictive Sampling-based Motion Planning Algorithm for Unmanned Aircraft Systems, sv. 103, č. 4, s. 59. ISSN 0921-0296, 1573-0409. Dostupné z: https://link.springer.com/10.1007/s10846-021-01501-3.

[47] Wang, N. a Sanfelice, R. G. *A Rapidly-Exploring Random Trees Motion Planning Algorithm for Hybrid Dynamical Systems*. Dostupné z: https://arxiv.org/abs/2210.15082.

[48] Schede, E.; Brandt, J.; Tornede, A.; Wever, M.; Bengs, V. et al. *A Survey of Methods for Automated Algorithm Configuration*. arXiv. Dostupné z: https://arxiv.org/abs/2202.01651.

[49] Eryoldaş, Y. a Durmuşoglu, A. A Literature Survey on Offline Automatic Algorithm Configuration, sv. 12, č. 13, s. 6316. ISSN 2076-3417. Dostupné z: https://www.mdpi.com/2076-3417/12/13/6316.

[50] Hutter, F.; Hoos, H. H. a Leyton Brown, K. Sequential Model-Based Optimization for General Algorithm Configuration. In: Coello, C. A. C., ed. *Learning and Intelligent Optimization*. Springer Berlin Heidelberg, sv. 6683, s. 507–523. ISBN 978-3-642-25565-6 978-3-642-25566-3. Dostupné z: http://link.springer.com/10.1007/978-3-642-25566-3_40.

[51] Lindauer, M.; Eggensperger, K.; Feurer, M.; Biedenkapp, A.; Deng, D. et al. *SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization*. Dostupné z: https://arxiv.org/abs/2109.09831v2.

[52] Hutter, F.; Stuetzle, T.; Leyton Brown, K. a Hoos, H. H. *ParamILS: An Automatic Algorithm Configuration Framework*. arXiv. Dostupné z: https://arxiv.org/abs/1401.3492.

[53] Blot, A.; Hoos, H. H.; Jourdan, L.; Kessaci Marmion, M.-E. a Trautmann, H. MO-ParamILS: A Multi-objective Automatic Algorithm Configuration Framework. In: Festa, P.; Sellmann, M. a Vanschoren, J., ed. *Learning and Intelligent Optimization*. Springer International Publishing, sv. 10079, s. 32–47. ISBN 978-3-319-50348-6 978-3-319-50349-3. Dostupné z: http://link.springer.com/10.1007/978-3-319-50349-3_3.

[54] López Ibáñez, M.; Dubois Lacoste, J.; Pérez Cáceres, L.; Birattari, M. a Stützle, T. The Irace Package: Iterated Racing for Automatic Algorithm Configuration, sv. 3, s. 43–58. ISSN 2214-7160. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2214716015300270.

[55] De Souza, M.; Ritt, M. a López Ibáñez, M. Capping Methods for the Automatic Configuration of Optimization Algorithms, sv. 139, s. 105615. ISSN 03050548. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0305054821003300.

[56] Massen, F.; López Ibáñez, M.; Stützle, T. a Deville, Y. *Experimental Analysis of Pheromone-Based Heuristic Column Generation Using Irace*. Springer Berlin Heidelberg. Dostupné z: https://link.springer.com/10.1007/978-3-642-38516-2_8.

[57] Montiel, O.; Sepúlveda, R. a Orozco Rosas, U. Optimal Path Planning Generation for Mobile Robots Using Parallel Evolutionary Artificial Potential Field, sv. 79, č. 2, s. 237–257. ISSN 0921-0296, 1573-0409. Dostupné z: http://link.springer.com/10.1007/s10846-014-0124-8.

[58] Alba, E. a Troya, J. M. A Survey of Parallel Distributed Genetic Algorithms, sv. 4, č. 4, s. 31–52. ISSN 1076-2787, 1099-0526. Dostupné z: https://onlinelibrary.wiley.com/doi/10.1002/(SICI)1099-0526(199903/04)4:4<31::AID-CPLX5>3.0.CO;2-4.

[59] Zheng, B.; Cheng, R. a Tan, K. C. *EvoRL: A GPU-accelerated Framework for Evolutionary Reinforcement Learning*. Dostupné z: https://arxiv.org/abs/2501.15129v2.

[60] Chippagiri, S. Optimization of Kubernetes for the High-Performance Computing with Kubernetes, Performance Analysis, and Dynamic Workload Placement toward the Enhancement of Cloud Computing, sv. 13, č. 12, s. 107–114. ISSN 23197064. Dostupné z: https://www.ijsr.net/getabstract.php?paperid=SR241201012040.

[61] Mondal, S. K.; Zheng, Z. a Cheng, Y. On the Optimization of Kubernetes toward the Enhancement of Cloud Computing, sv. 12, č. 16, s. 2476. ISSN 2227-7390. Dostupné z: https://www.mdpi.com/2227-7390/12/16/2476.

[62] Alexander, S. *Kubernetes as a Modern Platform for Hosting Enterprise Applications - Fastdev*. Dostupné z: https://devops.fastdev.com/articles/kubernetes-as-a-modern-platform-for-enterprise.html.

[63] Tesliuk, A.; Bobkov, S.; Ilyin, V.; Novikov, A.; Poyda, A. et al. Kubernetes Container Orchestration as a Framework for Flexible and Effective Scientific Data Analysis. In: *2019 Ivannikov Ispras Open Conference (ISPRAS)*. IEEE, s. 67–71. ISBN 978-1-7281-6055-9. Dostupné z: https://ieeexplore.ieee.org/document/8990167/.

[64] Adhipta, D.; Selo a Widyawan. Docker Swarm Orchestration for High Performance Computing Cluster as Container. In: *2023 6th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*. IEEE, s. 307–312. ISBN 979-8-3503-5834-6. Dostupné z: https://ieeexplore.ieee.org/document/10467287/.

[65] Clerc, M. a Kennedy, J. The Particle Swarm - Explosion, Stability, and Convergence in a Multidimensional Complex Space, sv. 6, č. 1, s. 58–73. ISSN 1089778X. Dostupné z: http://ieeexplore.ieee.org/document/985692/.

[66] Li, J.; Wan, L.; Huang, Z.; Chen, Y. a Tang, H. Hybrid Path Planning Strategy Based on Improved Particle Swarm Optimisation Algorithm Combined with DWA for Unmanned Surface Vehicles, sv. 12, č. 8, s. 1268. ISSN 2077-1312. Dostupné z: https://www.mdpi.com/2077-1312/12/8/1268.

[67] Lei, S.; Li, T.; Gao, X.; Xue, P. a Song, G. Research on Improved RRT Path Planning Algorithm Based on Multi-Strategy Fusion, sv. 15, č. 1, s. 13312. ISSN 2045-2322. Dostupné z: https://www.nature.com/articles/s41598-025-92675-5.

[68] Fiala, J. *Optimalizace robotických trajektorií pomocí modulárních evolučních algoritmů*. Zenodo, 2025. Dostupné z: https://doi.org/10.5281/zenodo.15489101.

SEZNAM ZKRATEK A SYMBOLŮ

EA evoluční algoritmus – Evolutionary Algorithm

GA genetický algoritmus – Genetic Algorithm

PSO optimalizace částicových rojů – Particle Swarm Optimization

PSO-DWA Particle Swarm Optimization s Dynamic Window Approach

DE diferenciální evoluce – Differential Evolution

MODDE modulární diferenciální evoluce – Modular Differential Evolution

CMA-ES evoluční strategie s adaptací kovarianční matice – Covariance Matrix Adaptation Evolution Strategy

SMAC sekvenční modelové ladění algoritmů – Sequential Model-based Algorithm Configuration

C-space Konfigurační prostor

RRT Rapidly-exploring Random Tree

PRM Probabilistic Roadmaps

JSON JavaScript Object Notation – textový formát pro strukturovaná data

PRD Procentuální relativní odchylka - Percentage of Relative Difference

CPU centrální procesorová jednotka – Central Processing Unit

GPU grafická procesorová jednotka – Graphics Processing Unit

API server aplikačního programového rozhraní – Application Programming Interface server

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ANOVA Analýza rozptylu

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

A Příloha s implementací optimalizačního frameworku

`JanFiala_217141_DP_2025_UAI.zip`, který je dostupný v [68]. Podadresář obsahuje zdrojové kódy, výsledky testů a konfigurační soubory pro optimalizaci trajektorií pomocí modulárních evolučních algoritmů.

- **app** – Zdrojové soubory aplikace.
- **child** – Adresář pro jednotlivé joby s ModDE.
- **manifests** – YAML manifesty a Dockerfile.
- **results** – Nalezené nejlepší trasy pro jednotlivé scény.
- **RRT** – Implementace algoritmu RRT.
- **scenario** – testovací scénáře a parametry pro Irace.
- **scripts** – Python skripty řídící optimalizaci pomocí nástroje Irace.
- **tests** – Výsledky testů, konfigurační soubory a skripty pro vyhodnocení získaných konfigurací.