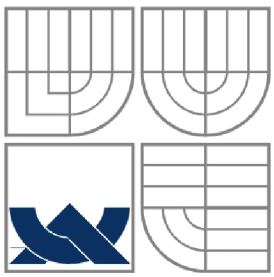


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PLÁNOVÁNÍ CESTY AUTONOMNÍHO LOKOMOČNÍHO ROBOTU NA ZÁKLADĚ STROJOVÉHO UČENÍ

AUTONOMOUS LOCOMOTIVE ROBOT PATH PLANNING ON THE BASIS OF MACHINE LEARNING

DISERTAČNÍ PRÁCE
DISSERTATION THESIS

AUTOR PRÁCE — Ing. PETR KRČEK
AUTHOR

VEDOUCÍ PRÁCE — RNDr. JIŘÍ DVOŘÁK, CSc.
SUPERVISOR

BRNO 2010

Abstrakt

Jak již plyne z názvu, tato disertační práce se zabývá plánováním cesty autonomního lokomočního robotu na základě strojového učení. Úkolem plánování cesty robotu je nalezení cesty z počáteční do cílové pozice bez kolize s překážkami tak, aby ohodnocení cesty bylo minimální. Autonomní robot je takový stroj, který je schopen vykonávat úkoly zcela samostatně i v prostředích s dynamickými změnami. Plánování cesty v dynamickém částečně známém prostředí je však obtížným problémem. Schopnost autonomního robotu přizpůsobovat svoje chování změnám prostředí může být zajištěna pomocí metod strojového učení. V souvislosti s plánováním cesty se z metod strojového učení uplatňují především případové usuzování, neuronové sítě, posilované učení, rojová inteligence a genetické algoritmy.

Prvá část disertační práce seznamuje čtenáře se současným stavem výzkumu v oblasti plánování cesty. Přehled metod je věnován základním všesměrovým robotům i robotům, na které jsou kladena diferenciální omezení.

V práci je navržena řada metod pro plánování cesty všesměrových robotů i robotů s diferenciálním omezením. Tyto navržené metody jsou založeny především na případovém usuzování a genetických algoritmech.

Případové usuzování řeší nový problém adaptací známých řešení podobných problémů, které byly již řešeny v minulosti. Zdá se, že případové usuzování je pro navigaci robotu vhodnou metodou, neboť v řadě aplikací robot řeší často podobné úkoly. Případové usuzování při plánování cest robotu je nemožné použít bez některé další metody hledání cest. Použití této metody je nezbytné v situacích, kdy systém případového usuzování začíná pracovat s prázdnou bází případů, nebo když získané řešení není dost dobré a musí být přepracováno. Jako pomocné metody je v této práci použito nejčastěji genetického algoritmu.

Genetické algoritmy (GA) jsou řazeny mezi heuristické evoluční metody, které jsou často používány k řešení složitých optimalizačních problémů. Nedostatkem raných aplikací genetických algoritmů na problematiku plánování cesty bylo použití klasických GA s binární reprezentací chromozomu pevné délky a pouze dvěma základními genetickými operátory (pro křížení a mutaci). Pozdější úspěšnější aplikace modifikovaly klasický GA zavedením nebinární reprezentace, proměnné délky chromozomu a především problémově specifických genetických operátorů, doplněných případně nějakými heuristickými znalostmi.

Všechny navržené metody byly implementovány v simulačních aplikacích. Výsledky experimentů prováděných v těchto aplikacích jsou součástí této práce. U každého experimentu je proveden rozbor výsledků. Z experimentů plyne, že navržené metody jsou schopné konkurovat běžně používaným metodám, neboť ve většině případů dosahují lepších výsledků.

Klíčová slova

Plánování cesty, neholonomní robot, případové usuzování, genetické algoritmy.

Abstract

As already clear from the title, this dissertation deals with autonomous locomotive robot path planning, based on machine learning. Robot path planning task is to find a path from initial to target position without collision with obstacles so that the cost of the path is minimized. Autonomous robot is such a machine which is able to perform tasks completely independently even in environments with dynamic changes. Path planning in dynamic partially known environment is a difficult problem. Autonomous robot ability to adapt its behavior to changes in the environment can be ensured by using machine learning methods. In the field of path planning the mostly used methods of machine learning are case-based reasoning, neural networks, reinforcement learning, swarm intelligence and genetic algorithms.

The first part of this thesis introduces the current state of research in the field of path planning. Overview of methods is focused on basic omnidirectional robots and robots with differential constraints.

In the thesis, several methods of path planning for omnidirectional robot and robot with differential constraints are proposed. These methods are mainly based on case-based reasoning and genetic algorithms.

Case-based reasoning (CBR) solves a new problem by adapting known solutions of similar previously solved problems. It seems that the CBR is a suitable method for robot control because robotic applications usually include repeated tasks. Case-based path planning is impossible if not combined with other path planning methods. Using these methods is necessary in situations, where a CBR system begins its work with empty case base, or a retrieved solution is not good enough and must be rejected or adapted. The genetic algorithm is the most used complementary method in this work.

Genetic algorithms (GA) are ranked among evolutionary heuristic methods, which are often used to solve complex optimization problems. Early applications of genetic algorithms to problems of path planning were insufficient due to the use of conventional GA with binary representation of the chromosome of fixed length, and only two basic genetic operators (for crossing and mutation). Subsequent successful applications modified the classical GA by non-binary representation, variable length chromosome, and especially problem-specific genetic operators, coupled with some heuristics.

All proposed methods were implemented in simulation applications. Results of experiments carried out in these applications are part of this work. For each experiment, the results are analyzed. The experiments show that the proposed methods are able to compete with commonly used methods, because they perform better in most cases.

Keywords

Path planning, nonholonomic robot, case-based reasoning, genetic algorithms.

Poděkování

Tímto děkuji RNDr. Jiřímu Dvořákovi, CSc. za cenné připomínky a rady při vypracovávání této práce.

Tato práce vznikla v rámci vědecko-výzkumných záměrů MSM 0021630518 "Simulační modelování mechatronických soustav" a MSM 0021630529 „Inteligentní systémy v automatizaci“.

Čestné prohlášení

Prohlašuji, že jsem disertační práci zpracoval samostatně dle pokynů vedoucího disertační práce a s použitím uvedené literatury.

Obsah

1. Úvod

Tato disertační práce se zabývá plánováním cesty robotu na základě strojového učení. Plánování cesty je částí tzv. *navigace*. Encyklopedický význam navigace spočívá v postupech, kterými lze určit polohu přemísťovaného objektu v prostoru a následně stanovit vhodnou cestu, podle které dosáhne objekt své cílové pozice. Toho lze dosáhnout různými způsoby, např. jednoduše využít orientačních značek v prostoru (dopravní značení, značení domů) nebo použít složitějších systémů (družicový navigační systém a prohledávání mapy).

Uvažujeme-li, že přesun bude provádět lokomoční robot (tedy, že zadaný úkol provádí stroj schopný pohybu), je možné, aby postup provádění přesunu do robotu snadno vložil zvenčí operátor jako posloupnost příkazů. V tomto případě hovoříme o robotech s pevnými programy. Tyto roboty jsou ale nepoužitelné v nestálých prostředích s dynamickými změnami. Jistým krokem dopředu byly roboty s proměnnými programy, které mají možnost okamžité volby programu ze zásobníku programů v závislosti na aktuálních informacích ze svých čidel. Plán cesty je však i v tomto případě do robotu vkládán zvenčí.

Potřeba použití robotů v nestálých prostředích vedla k vytvoření *autonomního robotu*. Autonomní robot je takový stroj, který je za použití metod umělé inteligence schopen vykonávat úkoly zcela samostatně. Takový robot musí sám optimálně reagovat na dynamické změny v prostoru. Navigaci provádí vlastní řídicí systémem robotu. Většina autonomních mobilních (lokomočních) robotů používá při navigaci operace nad mapou prostředí (v této souvislosti se hovoří o tzv. *mapově orientované navigaci*).

1.1 Mapově orientovaná navigace

Obecně je mapově orientovaná navigace často dělena do tří procesů (Meyer 2003, Nehmzow 2003): lokalizace, učení mapy a plánování cesty.

Lokalizace je proces odvození aktuální pozice robotu. Tento proces je velmi náročný, neboť lokalizace by měla zpracovat oba druhy informací, se kterými při navigaci pracujeme. Jedná se o informace ze dvou rozdílných, robotu dostupných zdrojů (Filliat 2003):

První je *idiothetický* zdroj, který poskytuje vnitřní informace o pohybech robotu. Tato informace se může týkat rychlosti, zrychlení, natočení řídicího kola kolového robotu či polohy nohy kráčejícího robotu. Zpracováním těchto dat dostaneme odhadovanou pozici robotu. Tyto procesy jsou nazývány jako hrubý odhad polohy či *odometrie*.

Druhý zdroj informací je *allothetický* zdroj, který poskytuje vnější informace o prostředí. Odpovídající podnět může pocházet ze zpracování obrazu (vidění), z laserových dálkoměrů, sonarů, spínačů doteku apod. Allothetické informace mohou být použity buďto přímo k rozpoznání místa nebo situace, nebo z nich mohou být odvozeny geometrické vlastnosti prostředí (např. pozice objektů).

Nevýhody a výhody těchto dvou zdrojů zpráv jsou vzájemně komplementární. Hlavní problém vyvolaný idiothetickými informacemi spočívá v kumulativní chybě vzniklé při zpracování dat. To vede k postupnému zvyšování nepřesnosti, a proto taková informace nemůže být použitelná po dlouhé časové období. Naopak kvalita allothetické informace je v čase stálá, ale trpí problémem klamného vnímání (*perceptual aliasing problem*), tj. skutečností, že pro daný senzorický systém se dvě odlišná místa v prostředí mohou zdát stejná. To znamená, že allothetická informace musí kompenzovat idiothetickou informační odchylku, zatímco idiothetická informace musí pomoci zajistit jednoznačné vnímání allothetické informace.

Učení mapy je proces pamatování dat, která jsou získávána během provozu robotu. V souvislosti s učením mapy rozlišujeme mapy na *metrické* a *topologické*. V metrických mapách jsou v běžné soustavě souřadnic uloženy pozice různých objektů (především překážek), se kterými se může robot setkat. Naproti tomu v topologické mapě jsou uloženy oblasti, kterých robot může dosáhnout. Spolu s těmito oblastmi jsou uloženy informace o jejich vzájemné pozici. Protože se učení mapy zabývá procedurami, které pomáhají udržovat také souvislost těchto map, je nutné volit vhodnou reprezentaci dat (např. mřížku, grafovou strukturu).

Přehled strategií pro lokalizaci robotu je uveden v článku (Filliat 2003) a přehled metod pro učení mapy je uveden v práci (Meyer 2003).

Plánování cesty je proces výběru dílčích kroků k dosažení cíle z aktuální pozice tak, aby v průběhu plnění plánu nenastala kolize s překážkami v pracovním prostoru. Lokalizace a učení mapy jsou vzájemně závislé procesy. Používání mapy pro lokalizaci robotu vyžaduje existující mapu, budování mapy vyžaduje pozici k odhadování vztahu k neúplné doposud naučené mapě. Plánování cesty je naopak dosti nezávislý proces, který je spuštěn, až když mapa existuje a je odhadnuta pozice robotu. V souvislosti s plánováním cesty rozlišujeme spojitý prostor a diskrétní prostor.

Plánování cesty v dynamickém částečně známém či neznámém prostředí je obtížným problémem. Schopnost robotu přizpůsobovat svoje chování změnám prostředí může být zajištěna pomocí metod strojového učení.

Strojové učení je souhrn metod umělé inteligence, které jsou určeny pro získávání znalostí. Tyto znalosti jsou následně upotřebeny v inteligentních systémech, kde zlepšují schopnost přizpůsobit se změnám okolního prostředí. V souvislosti s plánováním cesty se z metod strojového učení uplatňují především případové usuzování, neuronové sítě, posilované učení, ale i genetické algoritmy, které jsou mezi metody strojového učení také někdy řazeny (Mařík 1993). Problém totiž nemusíme začínat řešit s náhodně vygenerovanou počáteční populací, ale můžeme použít již natrénovanou populaci z předchozích podobných řešení.

1.2 Cíle práce

Výsledkem disertační práce má být inteligentní systém plánování cesty respektující kinematická omezení robotu. Jako základní stavební kámen této práce byl zvolen systém případového usuzování, který se na základě dosavadních zkušeností jeví pro roboty s diferenciálními omezeními jako nejlépe použitelný.

Vstupem do tohoto systému bude počáteční a koncový stav robotu. Výstupem systému bude cesta charakterizovaná akční trajektorií. Navržený systém musí mít přístup k metrické mapě. K tomu, aby navržený systém udržoval bázi případů, musí mít přístup ke stavovým údajům robotu v průběhu vlastního provádění plánu. Systém bude navržen tak, aby nebyl vázán na jeden kinematický model robotu. Funkčnost systému bude podložena experimenty.

Zbývající úkoly lze tedy popsat následujícími body:

- návrh vhodné metody plánování cesty uvažující diferenciální omezení, která je nezbytná pro spolupráci se systémem případového usuzování
- návrh metody využívající případové usuzování a s ní související návrh reprezentace případového grafu

- nalezení vhodné metody pro určení podobnosti případů, která je potřebná pro udržování aktuální báze případů a nalezení kritérií pro vypouštění případů
- ověření možnosti nastavování parametrů navržené metody další metodou strojového učení
- implementace vybraných známých metod plánování cesty uvažující diferenciální omezení, srovnání a experimentální ověření efektivnosti navrženého systému

1.3 Struktura práce

Prvá část práce zahrnující kapitoly 1, 2 a 3 tvoří přehledovou část disertace. Čtenář je v kapitolách 1 a 2 uveden do problematiky mapově orientované navigace a architektur používaných pro řízení autonomních robotů. Kapitola 3 seznamuje se základními typy diskretizace stavového prostoru, ale především poskytuje ucelený přehled stávajících metod pro plánování cesty robotu. Podrobnější popis je věnován metodám využívajících strojové učení. Samostatná podkapitola je také věnována plánování cesty, které uvažuje diferenciální omezení robotu.

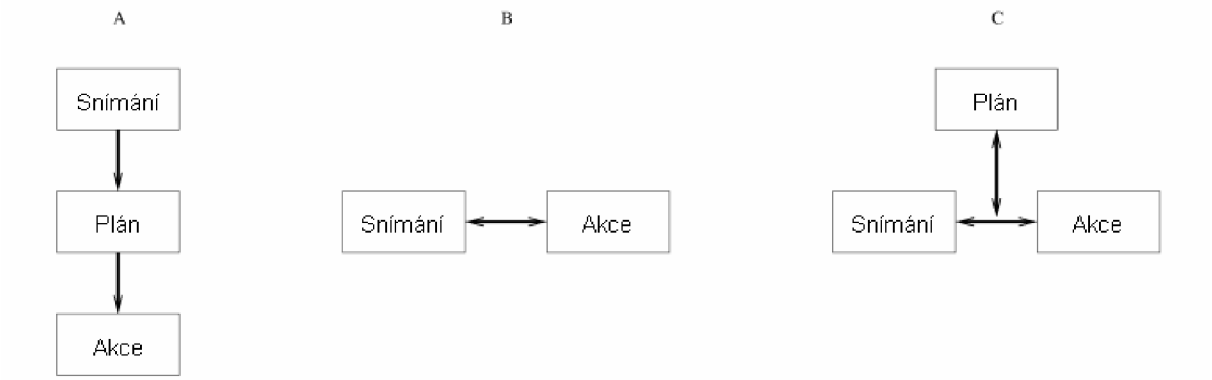
Druhá část práce zahrnuje kapitoly 4 a 5. Kapitola 4 podrobně popisuje navržené metody plánování. Je zde navrženo několik různých metod. Návrhy začínají jednoduchým plánováním cesty všesměrového robotu na mřížce a končí obtížnějším plánováním cesty robotu s diferenciálním omezením pro částečně známé prostředí. V kapitole 5 je pak uveden popis implementace navržených metod a obsluha simulační aplikace.

Poslední část práce tvoří kapitoly 6 a 7. V kapitole 6 je čtenář seznámen s výsledky experimentů prováděných s navrženými metodami v simulační aplikaci. Jsou zde také uvedeny popisy testovaných modelů robotů. Závěr práce (kapitola 7) především shrnuje dílčí závěry jednotlivých experimentů.

Protože byla většina navržených metod již publikována, jsou některé části textu i s obrázky převzaty z autorských publikací. Odkazy na publikace, ve kterých byly metody prezentovány, jsou v textu samozřejmě uvedeny.

2. Architektury pro řízení autonomních robotů

Provádění plánu, který předpisuje akce za účelem dosažení cíle, provází několik problémů. První spočívá ve skutečnosti, že téměř vždy existují rozdíly mezi mapou a aktuálním prostředím. Tyto rozdíly mohou být způsobeny například neznámými překážkami. Následkem toho přesný plán vygenerovaný používáním nepřesné mapy nemusí být možné provést. Druhý problém plyne ze skutečnosti, že akce robotu mohou být rušeny, například kvůli proklouznutí kola. Zde tedy, i když robot přesně provádí správný plán, cíl může minout.



Obr. 2.1. *Rozdílné architektury pro řízení autonomních robotů (Meyer 2003). (a) Architektura založená na hierarchické dekompozici, (b) Architektura založená na reaktivním systému, (c) Architektura založená na kombinovaném přístupu.*

Naposledy zmíněný problém způsobuje špatné fungování v reálných prostředích těm robotům, jejichž architektura řídicího systému je založena na *hierarchické dekompozici* (obr. 2.1a). Důvod je ten, že tyto systémy nemají žádnou kontrolu nad prováděním vypočteného plánu (Meyer 2003). V souvislosti s touto architekturou se také hovoří o *deliberativní strategii*.

Na přesně opačném přístupu je založena architektura, která pracuje na základě *reaktivního systému*. Tato architektura neuvažuje všechny možné vnitřní stavy (např. model mapy prostředí) a snaží se pro každá data ze senzorů ihned řídit akční členy. Tento přístup se tedy vyznačuje uzavřenou smyčkou *snímání-akce* (obr. 2.1b) a z toho plynoucím nedeterministickým chováním. Jednoduché roboty tímto řeší mnoho problémů vyplývajících z modelů prostředí. Velmi často je jako typický příklad této architektury citován článek (Brooks 1985). Reaktivní řídicí systém založený na fuzzy přístupu lze nalézt také např. v (Krček & Dvořák 2004).

Nejvíce současných robotických systémů kombinuje tyto dva opačné přístupy použitím *hybridní deliberativní/reaktivní* řídicí architektury. Kontrolér na vyšší úrovni je odpovědný za učení mapy, lokalizaci a plánování cesty. Reaktivní kontrolér na nižší úrovni odpovídá za provádění pohybů předepsaných kontrolérem na vyšší úrovni a současně reaguje na nepředvídané situace, např. neznámé překážky (obr. 2.1c). Souhra těchto dvou úrovní poskytuje robotům možnost rychle reagovat na změny jejich prostředí při schopnosti efektivně vykonávat dlouhodobý plán. Kombinovaným systémem se zabývá např. práce (Hu 1996). Přehled používaných architektur lze také najít v práci (Štěpán 2001).

3. Plánování cesty

Když má robot použitelnou mapu, má k dispozici odhad své pozice uvnitř této mapy a je zadána cílová pozice uvnitř této mapy, potom by robot měl být schopen pohybu z jeho aktuální pozice do cílové pozice. Plánování cesty spočívá v nalezení postupu, jak má robot dosáhnout cíle. S přehledem metod pro plánování cesty robotu seznamují např. práce (Meyer 2003), (Doyle 1996) nebo (Krček 2003). Vyčerpávající popis dále uvedených metod je možno nalézt v knize (LaValle 2006).

Při plánování (LaValle 2006) se každá situace robotu nazývá *stav* a označuje se jako x. Množina všech možných stavů se nazývá *stavový prostor* označovaný symbolem X. Aplikujeme-li na aktuální stav x akci u, tak se aktuální stav změní na stav x', který je daný přechodovou funkcí $x' = f(x, u)$. Akce, které mohou být aplikovány na stav x, tvoří *akční prostor* stavu x označovaný jako $U(x)$. Všechny možné akce všech stavů tvoří množinu $U = \bigcup_{x \in X} U(x)$. Množina $X_G \subset X$ obsahuje přípustné cílové stavy. Úkolem plánování je nalézt konečnou posloupnost akcí, jejichž aplikováním postupně transformujeme počáteční stav x_1 do některého stavu z X_G.

Plánování cesty přímo nad metrickou mapou není z hlediska časové náročnosti efektivní, neboť odpovídající stavový prostor X je nespočetně nekonečný. Časovou náročnost lze snížit vyhledáváním cesty v topologické mapě. Topologickou mapu obdržíme z metrické mapy tzv. *diskretizací* prostoru. Diskretizace se stala ústředním problémem plánování cesty ve spojitém prostoru. Většina plánovacích metod tedy začíná diskretizací a potom používá některou z metod hledání v topologické mapě. Hovoříme potom o *diskrétním plánování*. Po provedení diskretizace je prostor X spočetný a často i konečný.

Nejsou-li na robot při plánování kladena dynamická diferenciální omezení (viz kap. 4), pak se někdy hovoří o stavovém prostoru X jako o konfiguračním prostoru C a o stavu x jako o konfiguraci q, platí tedy rovnost $X = C$. Počet rozměrů konfiguračního prostoru často odpovídá počtu stupňů volnosti robotu.

V souvislosti s překážkami se zavádí pojem *volného konfiguračního prostoru* C_{free}, který je podmnožinou konfiguračního prostoru a obsahuje pouze takové konfigurace robotu, které nejsou v kolizi s žádnou překážkou.

V následujících podkapitolách jsou uvedeny metody pro *základní* (*holonomní*) *plánování cesty*, pro něž platí, že cesta může být snadno určena mezi dvěmi konfiguracemi v C přímou čarou, nejsou-li konfigurace na příslušné úsečce v kolizi s překážkou. Omezení kladená na robot jsou dána pouze množinou povolených konfigurací v C_{free}. Tato omezení se označují jako *globální*.

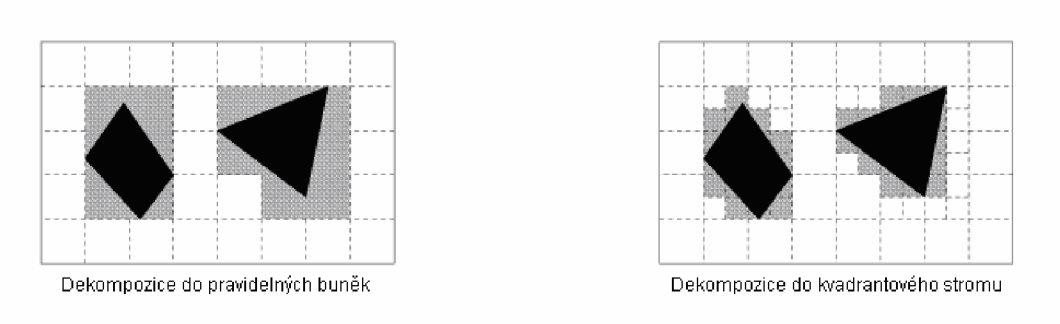
Práce (LaValle 2001) popisuje holonomní plánování cesty v konfiguračním prostoru C jako úlohu provádějící výpočet cesty z počáteční konfigurace q_{init} do cílové q_{goal}. V průběhu výpočtu obdržíme novou konfiguraci q_{new} aplikací akce $u \in U$ na konfiguraci q podle vztahu $q_{new} = q + u\Delta t$. Množina akcí U zde tvoří množinu všech rychlostí $\dot{q}$ takových, že $\|\dot{q}\| \leq const$ pro nějakou kladnou konstantu *const*.

Jestliže se rozhodneme pro plánování cesty přímo nad metrickou mapou, je dílčím výsledkem plánování funkce, která nám poskytuje údaj o přesunutí robotu v každém bodě volného prostoru. Většina těchto metod je založena na *poli potenciálů*, např. (Agirrebeitia 2005). V metodě pole potenciálů vrací zmíněná funkce součet kladného potenciálu emitovaného cílem a záporných potenciálů emitovaných překážkami. Směr pohybu v nějakém

bodě odpovídá gradientu funkce v tomto bodě. Pro odstranění nevýhody této metody, možnosti uváznutí v lokálním minimu, bylo navrženo několik vylepšení. Hlavní nevýhodu, velkou časovou náročnost výpočtu, se však odstranit nepodařilo.

3.1 Základní typy diskretizace prostoru

První třída metod pro diskretizaci extrahuje topologickou mapu z metrické mapy dekompozicí volného konfiguračního prostoru C_{free} do buněk korespondujících s uzly v topologické mapě (Meyer 2003). Do této třídy metod patří typicky *rastrové metody*. Topologická mapa se v tomto případě dostane pomocí proložení mřížky přes metrickou mapu (obr. 3.1). Jiné metody konstruují topologickou mapu na základě dekompozice volného prostoru do konvexních polygonů. Mezi tyto metody patří např. *vertikální buňková dekompozice* a metody založené na *triangulaci* (LaValle 2006).



Obr. 3.1. *Příklady rastrových metod (Meyer 2003)*

Druhá třída metod pro diskretizaci provádí rozklad volného konfiguračního prostoru C_{free} do dílčích přípustných cest, které propojují klíčové body rozptýlené v tomto prostoru (Meyer 2003). Takto zkonstruovaný graf se nazývá *mapa cest*. V základních variantách metod této třídy je získaná mapa cest vhodná pro bodový robot. Chceme-li plánovat cestu pro nebodový robot, je nutné provést úpravu metrické mapy tzv. *zvětšením překážek*.

Obr. 3.2. *Příklady exaktních metod (Meyer 2003)*

O některých metodách konstruujících mapu cest se hovoří jako o *exaktních metodách* (obr. 3.2). Diskretizace prohledávaného prostoru extrahuje topologickou mapu z metrické tak, že topologická mapa zcela přesně reprezentuje původní scénu. Jedná se o složitý výpočet, který však garantuje nalezení cesty, případně informaci o neexistenci řešení. Mapa cest může

být odvozena například z *grafu viditelnosti* (Priya 2006), který spojuje přímkami takové vrcholy překážek, které jsou navzájem viditelné. V tomto případě klíčové body mapy cest jsou vrcholy překážek, které umožňují výpočet nejkratších cest ležících blízko překážek. Blízkost překážek může být však nežádoucí, například pokud se robot musí pohybovat rychle. Metody pro výpočet mapy cest, které maximalizují vzdálenost od překážek, využívají *Voronoiův diagram* (Kobayashi 2002, Šeda 2005). V tomto případě klíčové body mapy cest jsou body ekvidistantní nejméně třem překážkám.

Dalším případem metod generujících mapu cest jsou *metody založené na vzorkování*. Princip těchto metod spočívá v prohledávání nespočetně nekonečného konfiguračního prostoru C podle daného typu vzorkování. Algoritmus vzorkování je však ukončen po dosažení daného počtu vzorků, tedy přípustných konfigurací robotu v metrické mapě. Vzorky představují uzly grafu, který tvoří topologickou mapu. Nepřípustné konfigurace nenáležící do C_{free} jsou odhaleny pomocí *detekce kolize* D, kde D: $C \rightarrow \{true, false\}$. Vzhledem k tomu, že metody z této třídy jsou většinou založeny na náhodném vzorkování, tak nezaručují úplnost vzorků a tudíž ani nalezení cesty.

Pokud jsou klíčové body v mapě cest vzorkovány náhodně, potom se také někdy hovoří o *pravděpodobnostních metodách*. Práce (Overmars 1994) nazývá diskretizaci prostoru jako učící fázi a následné prohledávání grafu jako dotazovací fázi. V učící fázi se konstruuje pravděpodobnostní mapa cest v konfiguračním prostoru. Tato mapa cest je graf, jehož uzly korespondují s náhodně vybranými konfiguracemi ve volném prostoru a hrany korespondují s jednoduchými nekoliznímí pohyby mezi uzly. Tyto jednoduché pohyby jsou vypočteny použitím rychlé lokální metody. Dotazovací fáze hledá cesty v mapě cest mezi odlišnými páry konfigurací. Pokud není cesta nalezena, k rozšíření mapy cest je volána opět učící fáze. Další pravděpodobnostní metody lze najít např. v (Geraerts 2006), (Kavraki 1998).

Některé metody jsou založeny na *inkrementálním vzorkování a prohledávání*, tzn., že topologická mapa je postupně rozšiřována o uzly, které jsou dosažitelné z některých již stávajících uzlů. Pro zjištění dosažitelnosti musí být k dispozici také model robotu. Typickým představitelem jsou metody z rodiny *rychle mapujících hustých stromů (Rapidly exploring Dense Trees – RDTs)*. U těchto metod se topologická mapa vytváří při každém hledání cesty jako strom, který pozvolna zvyšuje pokrytí konfiguračního prostoru. Metody typu RDTs končí s generováním stromu v okamžiku nalezení řešení. Pokud jsou přidávány konfigurace do stromu náhodně, pak se jedná o metodu *rychle mapujících náhodných stromů (Rapidly exploring Random Trees – RRTs)*. Detailně popisuje tuto metodu např. článek (LaValle 2001). Byly navrženy také různé modifikace tohoto algoritmu (např. Krejsa 2005).

Po provedení vyhovující diskretizace některou metodou (kromě metody založené na inkrementálním vzorkování) je před vlastním plánováním cesty směrem k cíli nejprve nutné provést výpočet cesty mezi počáteční pozicí robotu a nejbližším bodem v diskretizovaném prostředí. Pomocí diskrétního plánování je potom hledána cesta až k bodu blízkému k cíli. Když byl tento bod dosažen, je počítán finální pohyb vedoucí z tohoto bodu do cíle. Jakmile je celá cesta nalezena, je možné ji ještě optimalizovat pro odstranění zajížděk, což jsou vedlejší efekty diskretizace.

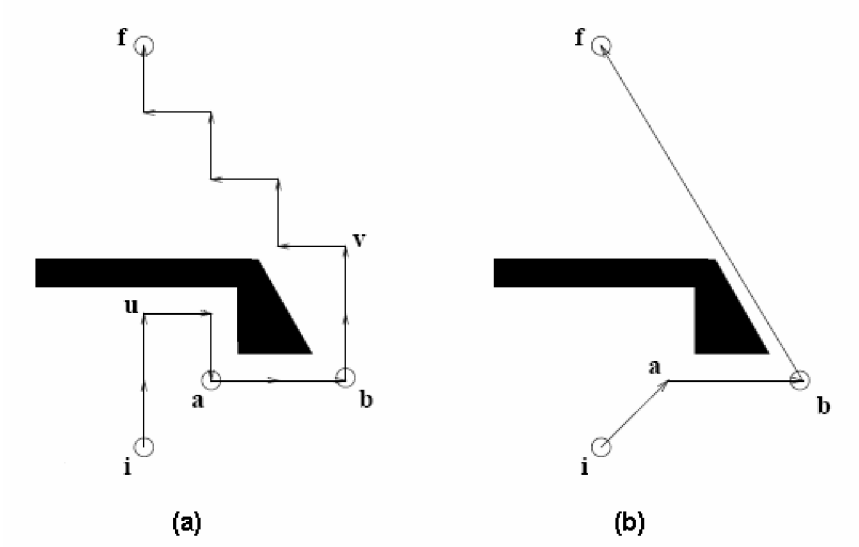
3.2 Metody plánování cesty

Pro diskrétní plánování cesty v topologické mapě lze použít klasických přístupů prohledávání grafu nebo použít některou z metod strojového učení, která poskytuje pro danou aplikaci rychlejší výpočet anebo vhodnější řešení.

Mezi skupinu klasických algoritmů pro prohledávání grafu patří především *Dijkstrův algoritmus*, *A** a jeho varianty. Pro přijatelné velikosti map obvykle používané v robotice poskytují tyto algoritmy účelné použití. Pokud je mapa příliš velká, je navíc možné užít *dynamické programování*, využívající Bellmanův princip optimality (Meyer 2003). Pokud se hledá cesta v neohodnocené mřížce, může posloužit jednoduché *prohledávání do šířky* z cíle, známé též jako *algoritmus záplavového vyplňování* (*wavefront-expansion*).

Mezi méně známé varianty algoritmu A* patří algoritmy *D** a *ARA**. Algoritmus D* je schopen se vyrovnat se změnou v grafu v průběhu již spouštěného prohledávání, takže je předurčen pro použití v částečně známých prostředích. Algoritmus ARA* lze zařadit mezi *anytime* algoritmy. Tyto algoritmy jsou schopny poskytnout výsledek po velmi krátké době výpočtu, což je vhodné pro aplikace pracující v reálném čase. Výsledek však nelze považovat za nejlepší řešení. V průběhu výpočtu se výsledek zpřesňuje. V práci (Ferguson 2005) je kromě algoritmů D* a ARA* představena také metoda oba přístupy kombinující.

Existují však i metody, které nejsou založeny na grafové struktuře. V článku (Donnart 1995) je použit přístup pro plánování cesty pomocí pravidlového systému. Řízení pohybu robotu i popis mapy obstarává hierarchický klasifikační systém. Reaktivní modul zajišťuje pomocí reaktivních pravidel aktuální pohyb robotu, plánovací modul určuje prostřednictvím plánovacích pravidel právě platný cíl a auto-analyzační modul upravuje v online režimu na základě detekce překážek množinu plánovacích pravidel. Topologická mapa prostředí zde není reprezentována grafem, ale právě množinou plánovacích pravidel. Pokud plánovací pravidla neuvažují doposud překážku, uplatňuje se strategie pohybu přímo na cíl s případným objížděním překážky (obr. 3.3a). Poté, co je cíle dosaženo, nastane úprava plánovacích pravidel a při dalším hledání cesty jsou tyto znalostí již uplatněny (obr. 3.3b).



Obr. 3.3. *Přístup k plánování cesty pomocí pravidel (Donnart 1995)*

Úkolem plánovacích pravidel je rozložit úlohu (dosažení cíle) na sérii dílčích úloh (dosažení podcílů) podle aktuálních informací ze senzorů (pozice robotu). Plánovací pravidlo má tedy následující tvar:

IF <informace ze senzoru> AND <aktuální úloha> THEN <podúloha> (3.1)

Každé takovéto pravidlo plánovacího modulu je charakterizováno dvěma hodnotami intenzity: lokální a globální intenzitou. Lokální intenzita ohodnocuje užitečnost rozkládání

úlohy na podúlohy. Globální intenzita je naopak použita pro detekci a potlačování pravidel, která nejsou vhodná pro použití. O aktualizaci těchto intenzit se stará modul posílení, který je také částí hierarchického klasifikačního systému.

V práci (Martínez 1998) je použit pro získání nekolizní optimální trajektorie mezi pevnými polygonálními překážkami algoritmus simulovaného žíhání. Pro reprezentaci trajektorií byly zvoleny B-spline křivky.

3.3 Metody využívající strojové učení

Tato podkapitola rozšiřuje předchozí podkapitolu o přístupy využívající strojového učení (neuronové sítě, genetické algoritmy, rojová inteligence, případové usuzování a posilované učení).

3.3.1 Neuronové sítě

První možnost řešení plánování cesty pomocí neuronové sítě vychází z *Hopfieldovy neuronové sítě,* a to z její spojité realizace. Konkrétní úloha se formuluje a zadává podobně jako při řešení NP úplných problémů pomocí Hopfieldovy neuronové sítě a to pomocí matice vah (Šnorek 1996). Učení je tedy nahrazeno nastavením vah podle zadání úlohy. V případě statického zadání úlohy je vybavování Hopfieldovy sítě obvykle iterační do ustálení. Ustálený stav pak představuje řešení úlohy. V dynamickém prostředí nejsou pozice a tvar překážek a cíle dopředu známy. S těmito informacemi se pracuje až v průběhu plánování, obvykle snímáním scény v režimu real-time. Pokud totiž zajistíme nepřetržité vybavování sítě na základě dat ze senzorů, opakující se dotazování na následující úsek cesty není již výpočetně náročné. Proto jsou takovéto systémy vhodné jako plánovač do architektur založených na hierarchické dekompozici, jež svůj cyklus snímání-plánování-akce mohou opakovat dostatečně rychle.

Jedním z prvních článků o této problematice je (Glasius 1995). Autoři používají Hopfieldovu neuronovou síť se spojitými neurony pro plánování cesty v dynamickém prostředí s libovolným tvarem překážek. Navržený plánovací systém plánuje cestu obecně v d-dimenzionálním konfiguračním prostoru, a proto jsou neurony uspořádány v d-dimenzionální mřížce. Neurony jsou pospojovány jen k jejich nejbližším sousedům. Váhová matice T je diagonálně symetrická. Hopfieldova síť obsahuje právě tolik neuronů, kolik má vstupů a každý neuron koresponduje s určitou oblastí v konfiguračním prostoru. Neurony korespondující s překážkami mají minimální aktivitu, neuron korespondující s cílovou pozicí má maximální aktivitu a počáteční pozice může korespondovat s jakýmkoliv neuronem, který nekoresponduje s překážkou. Výsledná cesta je dána gradienty aktivit neuronů. Navržený systém připomíná Dijkstrův algoritmus pro nalezení nejkratší cesty z jednoho uzlu do všech zbývajících uzlů v grafu. Ovšem jak uvádí autoři, tak obtížnost Dijkstrova algoritmu $O\left(N^2\right)$ je zde jejich přístupem snížena na $O\left(N\right)$. Tato složitost je poněkud zavádějící, neboť se zřejmě jedná o složitost poslední iterace v cyklu vybavování. Podle autorů je také výhodou snadnější reakce na náhlé změny v prostředí a možnost hardwarové implementace analogové verze této neuronové sítě.

Na Hopfieldově neuronové síti je také založen systém prezentovaný v práci (Ritthipravat 2002). Pomocí modifikované Hopfieldovy sítě je zde řešeno dynamické plánování cesty dvou mobilních robotů. Modifikace spočívá v úpravě topologie sítě, váhová matice je asymetrická, vždy upravená pro pozici cíle. Pro robot je cesta určena podobně jako

v (Glasius 1995) s tím, že druhý robot se mu jeví jako dynamická překážka. Toto platí pro oba roboty.

Mírně odlišným přístupem od použití Hopfieldovy sítě je použití neuronové sítě založené na *odporové mřížce* (Bugmann 1995). Odporová mřížka je rozdělena do množiny malých d-dimenzionálních buněk, které tvoří uzly odporové mřížky. Každý uzel je propojen s $2d$ nejbližšími sousedy přes $2d$ odporů R. Uzly na hranách mřížky a v rozích mají menší počet sousedů. Odpory R mají v celé mřížce stejnou hodnotu. Při použití této mřížky pro plánování cesty uzel odpovídající cílové pozici prohlásíme jako zdrojový a nastavíme jej na kladný potenciál. Cestu z libovolného uzlu najdeme pomocí gradientů potenciálů. Pokud cesta existuje, tak navržená metoda garantuje její nalezení, neboť metoda odporové mřížky nepodléhá problémům plynoucím z lokálních minim. Neuronová síť je dvouvrstvá o stejném počtu neuronů na obou vrstvách. Příčná spojení mezi neurony v horní vrstvě přenášejí informace o potenciálech sousedních uzlů v mřížce. Horní vrstva tak vlastně představuje odporovou mřížku. Spodní vrstva reprezentuje paměť prostoru, ve které jsou uloženy informace o pozicích překážek a cíle. Každý neuron v horní vrstvě má jeden vstup napojen na neuron ve spodní vrstvě, který koresponduje se stejnou pozicí v prostoru. Ostatní vstupy jsou propojeny s $2d$ sousedními neurony.

V článku (Kassim 1999) je popsána neuronová síť nazvaná *wave expansion neural network* (WENN) a je ukázáno, jak ji lze použít pro plánování cesty pomocí různých variant umělých potenciálových polí. Topologie je opět podobná topologiím výše uvedeným, avšak v tomto případě má pro informace o prostředí každý neuron dva externí vstupy. Odlišností od předcházejících sítí je také fakt, že po inicializaci má WENN kromě nulových aktivit neuronů také nulovou matici vah. Proces učení sítě je zde totiž sloučen s procesem vybavování.

Neuronová síť s časově nenáročným výpočtem aktivit neuronů (Lebedev 2005) se nazývá *dynamic wave expansion neural network* (DWENN) a je proto vhodná pro plánování v reálném čase. Jedná se opět o jednovrstvou síť se vzájemně propojenými neurony, avšak o síť bez externích vstupů. Pro cílový neuron se uplatňuje odlišný výpočet aktivity a překážky se uvažují při výpočtu vah. DWENN slučuje také proces vybavování s procesem učení. Ve zmíněné práci je DWENN porovnávána v dynamických experimentech s některými výše uvedenými neuronovými sítěmi. DWENN dosahuje nejlepších výsledků jak v rychlosti výpočtu, tak i v délkách nalezených cest.

V článku (Arleo 2000) je představen výpočetní model tkáně mozkové komory krysy, který zajišťuje prostorové rozpoznávání a navigaci. Nalezení cesty je zde modelováno neuronovou sítí, v níž se pro učení synaptických vah užívá posilovaného učení. Dvouvrstvá neuronová síť obsahuje polohové a akční neurony. Označme aktivitu polohového neuronu i jako r_i, a aktivitu akčního neuronu $a \in A$ jako r_a, kde A je množina akcí, $A = \{\text{sever}, \text{jih}, \text{východ}, \text{západ}\}$. Pozice robotu s je reprezentována jako vektor aktivit polohových neuronů $r(s) = [r_1(s), r_2(s), ..., r_n(s)]$, kde n je počet polohových buněk. Nechť $w^a = (w_1^a, ..., w_n^a)$ je synaptická projekce z polohových neuronů do akčního neuronu. Aktivita r_a závisí lineárně na pozici robotu s a na synaptických vahách w^a:

$$r_a(s) = (w^a)^T r(s) = \sum_{i=1}^{n} w_i^a r_i(s) \tag{3.2}$$

Úloha učení spočívá v aktualizaci vektoru w^a tak, aby neuronová síť poskytovala pro každý stav s optimální aktivitu akčního neuronu $r_a(s)$, dle které se vypočte pohyb robotu ve stavu s. K tomuto je zde použita lineární gradientní verze Watkinsonova algoritmu Q-učení.

Neuronová aktivita $r_a(s)$ je tedy interpretována jako očekávaný zisk, pokud použijeme akci *a* ve stavu robotu *s*.

Myšlenka použití posilovaného učení pro nastavení některých vah neuronové sítě byla použita již v práci (Millán 1995). Jako metoda posilovaného učení zde byla použita TD metoda (*temporal difference learning*). Neuronová sít zajišťovala současně reaktivní řízení robotu podle dat ze senzorů i navigaci robotu na cíl.

3.3.2 Genetické algoritmy

Plánování cesty představuje obecně složitý optimalizační problém. K řešení složitých optimalizačních algoritmů se velmi často používají heuristické evoluční metody, k nimž patří také *genetické algoritmy* (GA). Nedostatkem raných aplikací genetických algoritmů na tuto problematiku bylo použití klasických GA s binární reprezentací chromozomu pevné délky a pouze dvěma základními genetickými operátory (pro křížení a mutaci). Pozdější úspěšnější aplikace modifikovaly klasický GA zavedením nebinární reprezentace, proměnné délky chromozomu a především problémově specifických genetických operátorů, doplněných případně nějakými heuristickými znalostmi. Většina aplikací GA používá pro reprezentaci prostředí robotu dvoudimenzionální pravoúhlou mřížku, přičemž některé na základě toho omezují směry pohybu robotu na horizontální, vertikální a případně diagonální. Některé GA však provádějí vlastní diskretizaci a pracují tak nad spojitým prostorem. GA jsou schopné pokrýt rozsáhlý prostor prohledávání při relativně malých nárocích na výpočetní zdroje. Další předností GA je schopnost adaptace nalezeného řešení na spojitě se měnící dynamické prostředí a generovat řešení v reálném čase. Tuto schopnost demonstrují např. práce (Burchardt 2006) a (Zheng 2004). Nedostatkem GA je, že nezaručují nalezení globálně optimálního řešení.

Práce (Homafair 2001) popisuje evoluční plánovací systém pracující ve spojitém prostoru a používající kódování pomocí reálných čísel. Základní tvar chromozomu se skládá ze tří genů obsahujících souřadnice startovního, cílového a vnitřního uzlu cesty (takto vypadají chromozomy po vygenerování počáteční populace, po křížení a po mutaci. Operace křížení a mutace pracují s vnitřním genem (jádrem). Souřadnice jádra potomka se získají tak, že se náhodně při použití rovnoměrného rozdělení stanoví hodnoty z intervalů určených dvojicemi odpovídajících souřadnic jader rodičů a k nim se pak připočítají náhodně vygenerované hodnoty z normálního rozdělení. Mutace se pak aplikuje na prvou nebo druhou souřadnici jádra potomka. Na základní tvar chromozomu se pak aplikuje opravný operátor, který do chromozomu doplňuje opravné uzly tak, aby nedocházelo ke kolizím cesty s překážkami. Cesta je přitom tvořena úsečkami spojujícími sousední uzly chromozomu. Pro výběr jedinců ke křížení je použita fuzzy turnajová selekce, která je založena na těchto kritériích: délka cesty jako součet délek jednotlivých lineárních segmentů, součet změn ve směrnicích jednotlivých úseků a průměrná změna směrnice cesty.

Další prací, ve které je však metoda plánování navržena nad dvourozměrnou mřížkou, je (Gemeinder 2003). Chromozom má proměnnou délku a reprezentuje cestu určenou jako posloupnost sousedících buněk mřížky. Počet možných směrů pohybu je omezen na osm. Křížení je jednobodové, přičemž bod křížení se náhodně vybere jako průsečík rodičovských cest. Mutace se provádí tak, že se náhodně vyberou dva body cesty a segment cesty mezi těmito body se nahradí dvěma lineárními segmenty spojujícími uvedené body s náhodně vybraným bodem ležícím mimo cestu. Součástí těchto dvou operátorů jsou také opravné akce, které z cesty odstraňují nežádoucí jevy jako např. uzavřené smyčky. Dalšími operátory jsou operátor napínání cesty a operátor obcházení překážek.

Prostředím robotu v práci (Hu 2004) je dvourozměrná mřížka, jejímž buňkám jsou přiřazeny celočíselné indexy. Cesta je složena z lineárních segmentů spojujících buňky mřížky. Do chromozomu je cesta kódována jako posloupnost indexů buněk, přičemž první představuje start, poslední cíl a vnitřní indexy představují krajní body segmentů. Počet vnitřních indexů není pevný. Autoři zavádějí šest problémově specifických operátorů: křížení, mutaci, opravu uzlu, opravu segmentu, mazání uzlu a zlepšení cesty. Hodnotící funkce vychází ze součtu délek segmentů a z délek úseků, kterými segmenty protínají překážky.

Zheng et al. (2004) řeší problém koordinovaného plánování dráhy pro skupinu bezpilotních letadel. Každý chromozom koresponduje s dráhou letu jednoho letadla. Každý uzel chromozomu obsahuje tři souřadnice průsečíku lineárních segmentů, ze kterých je dráha složena, stavovou proměnnou, která určuje, zda je proveditelný průlet průsečíkem a následujícím segmentem a odkaz, který ukazuje na další uzel téhož chromozomu a definuje tak v podstatě následující segment. Autoři používají operátor křížení a pro tuto úlohu speciálně navržených šest operátorů mutace (změna souřadnic uzlu, vložení uzlu, rušení uzlu, výměna uzlu, vyhlazování cesty a speciální operátor změny předposledního uzlu s cílem dosáhnou vhodný úhel pro přistání). Hodnota funkce fitness vychází z hodnotící funkce, která je závislá na délkách segmentů, průměrných nadmořských výškách segmentů a mírách pronikání segmentů do nebezpečných oblastí.

Burchardt a Salomon (2006) plánují pomocí GA cestu pro všesměrový robot, který je určen pro týmový fotbalový zápas. Cesta, která je reprezentována chromozomem, je plánována v dynamickém prostředí, kde překážky tvoří ostatní roboty. Autoři používají operátory křížení a mutace. Chromozom je proměnné délky a jeho geny představují celočíselné souřadnice vnitřních uzlových bodů cesty. Algoritmus je optimalizován pro běh v mikrokontroléru, kde v režimu on-line neustále plánuje cestu až do dosažení cíle. Fitness funkce zohledňuje délku cesty a pokutu závislou na míře kolize.

3.3.3 Rojová inteligence

Rojová inteligence zahrnuje metody, které vychází z chování skupiny živočichů stejného druhu (např. ptáci). Každý agent v roji s sebou nese možné řešení a podobně jako u genetických algoritmů je prvotně náhodně vygenerovaný roj měněn v dalších populacích. Oproti genetickým algoritmům má při vytváření nové populace významnou roli vzájemné ovlivňování agentů v roji. Mezi nejznámější metody rojové inteligence patří *metoda rojení částic* a *metoda mravenčí kolonie*. Obecně je výhodou těchto metod schopnost přizpůsobit se dynamickým změnám v prostoru.

Metoda rojení částic byla inspirována sociálním chováním ptáků a ryb. Tato metoda používá skupinový přístup k řešení problémů. V systému rojové optimalizace simultánně koexistuje a spolupracuje více kandidátů řešení. Kandidát řešení se nazývá částice a skupina částic (populace) je nazývána rojem. Každá částice je reprezentována pozičním vektorem a příslušným vektorem okamžité rychlosti. Dále si pamatuje svou individuální nejlepší hodnotu funkce fitness a pozici, která vedla k této hodnotě. Částice se pohybuje v prostoru daného problému a hledá optimální pozici. V průběhu času upravuje částice svou pozici jednak podle své vlastní zkušenosti, jednak podle zkušeností sousedních částic.

V práci (Quin 2004) je pro plánování cesty ve spojitém dvourozměrném prostoru použito metody rojení částic. Nejprve je provedena diskretizace stavového prostoru (navrženou metodu lze zařadit mezi exaktní metody), čímž je získán graf, zde nazvaný jako *Maklink*. V tomto grafu je nalezena cesta pomocí Dijkstrova algoritmu. Autoři dále uvádějí, že z hlediska senzorického systému by se měl robot spíše pohybovat blíže u překážek než po

takto nalezené cestě. Proto je cesta nalezená v Maklink grafu optimalizována pomocí metody rojení částic pokračující diskretizací. Metoda minimalizuje jak délku cesty, tak vzdálenost od překážek.

Metoda rojení částic je taktéž představena v (Lei 2006). Cesta ve dvourozměrném spojitém prostoru je hledána přímo touto metodou. V práci je vyzdvižena schopnost rychlého přeplánování v důsledku pohyblivé dynamické překážky. Cesta je ovšem reprezentována funkcí $y = f(x)$, z čehož plyne použitelnost pouze pro velmi jednoduché scény.

Metoda mravenčí kolonie jsou inspirovány chováním skutečných mravenců při hledání potravy. Mravenci jsou schopni najít nejkratší cestu z množiny alternativních cest mezi mraveništěm a zdrojem potravy na základě feromonových stop. Tyto stopy a rychlost jejich vypařování jsou klíčové faktory pro hledání nejkratší cesty. Po kratší cestě za stejný čas projde více mravenců než po delší cestě, což spolu s faktem vypařování feromonu vede k tomu, že kratší cesta má silnější feromonovou stopu a tudíž je dalšími mravenci více preferována.

Prohledávání topologické mapy pomocí metody mravenčí kolonie je navrženo v práci (Garro 2006). Metoda mravenčí kolonie je zde kombinována s genetickým algoritmem, který je použit pro stanovení některých parametrů mravenčího algoritmu. Rozšíření metody o genetický algoritmus zde však nepřináší žádné výhody, protože pro nalezení nejlepšího řešení vyžaduje větší počet iterací a větší počet mravenců. Vzhledem k tomu, že je topologická mapa dána neorientovaným grafem, je zarážející, že nebylo provedeno srovnání s klasickými grafovými algoritmy. Liu et al. (2005) kombinují při hledání nejkratší cesty mravenčí algoritmus se simulovaným žíháním. Simulované žíhání se uplatňuje při výpočtu pravděpodobností, podle nichž se mravenci rozhodují o volbě další cesty.

Zdokonalený mravenčí algoritmus pro plánování nekolizních cest jednotlivých robotů robotického systému ve složitém statickém prostředí je navržen v práci (Liu 2006). Zdokonalení spočívá ve zlepšení selektivní strategie, snižující tendenci uvíznutí v lokálním optimu, a přidání penalizační funkce, zabraňující uvíznutí ve slepé uličce. Viet et al. (2008) řeší problém plánování cesty s obcházením překážek pomocí několika kooperujících mravenčích kolonií, které si mezi sebou vyměňují informace.

3.3.4 Případové usuzování

Strojové učení může být také realizováno použitím *případového usuzování* (*Case Based Reasoning – CBR*). Případové usuzování řeší nový problém adaptací známých řešení podobných problémů, které byly již řešeny v minulosti. Zdá se, že případové usuzování je pro navigaci robotu vhodnou metodou, neboť v řadě aplikací robot řeší často podobné úkoly. Hlavní cyklus případového usuzování může být popsán čtyřmi základními kroky (Aamodt 1994): (i) nalezení nejvíce podobného případu či případů; (ii) použití informací a znalostí z tohoto případu (případů) k řešení daného problému; (iii) kontrola navrženého řešení; (iv) uložení vhodných částí tohoto řešení k opětnému použití při řešení budoucích podobných problémů. Případové usuzování při plánování cest robotu je nemožné použít bez některé další metody hledání cest. Použití této metody je nezbytné v situacích, kdy systém případového usuzování začíná pracovat s prázdnou bází případů, nebo když získané řešení není dost dobré a musí být přepracováno.

V práci (Haigh 1994) se hovoří o případové bázi jako o *případovém grafu* (*case graph*). Každý případ reprezentující část cesty je zde aproximován do lineárního segmentu, který tvoří hranu v tomto grafu. V případě přidání případu, který protíná již existující segmenty, je nutné přidávaný případ a protnuté segmenty rozdělit na menší segmenty, protože segmenty se

mohou protínat pouze v krajních bodech. Každý případ je ohodnocen (délka, stav silnice, průchodnost, …). Podle tohoto ohodnocení se dále určuje, zda použít daný případ, nebo dát přednost neprozkoumané oblasti. Ohodnocení neznámé oblasti se vypočte z délky přímého spojení. V práci je také definován termín *trasa (route)*. Trasa představuje spojitou jednoduchou cestu v rovině, která zahrnuje několik segmentů i nerozpoznaných oblastí. Problém nalezení vhodné množiny případů je redukován na nalezení optimální trasy ze startovního do cílového uzlu grafu. Případové segmenty obsažené v nalezené trase jsou předány plánovači, který podle nich sestaví výslednou cestu.

Haigh et al. (1994) rozšiřují navrženou metodu o dynamické aktualizace z dostupných informací, tzv. učení během provádění plánu. Autoři zde uvádějí metodu modifikace ohodnocení případů či modifikace celých případů, pokud během skutečného provádění plánu nastane kolize s dočasnou překážkou, dopravní zácpa, situace zrušení ulice apod. V práci je také obecně zmíněna problematika přidávání případů do knihovny (redundance informací, chybné případy, zapomínání případů).

Jiné výsledky bádání v oblasti plánování cesty pomocí případového usuzování jsou shrnuty v práci (Kruusmaa 2003). Jedná se však o zcela odlišný přístup, protože cesty se zde do báze případů ukládají kompletní, k žádnému rozkladu do případového grafu zde nedochází. Celá metoda pracuje nad mřížkou. Po absolvování každé cesty robotem je určena průjezdnost této cesty podle její délky, obtížnosti (zdržení, počet zásahů lokálního plánovače pro řešení kolizí s neznámými překážkami) a podle ohodnocení ideální cesty. Poté je cesta i s vypočteným ohodnocením průjezdnosti uložena do případové báze. Dynamické změny prostředí autoři zachycují pouze v případové bázi ve formě průjezdnosti cest, zaznamenání těchto změn do mapy prostředí vůbec neprovádí. Redukce počtu případů v případové bázi je zde dosaženo rozpoznáváním podobných případů a zapomínáním případů. Podobnost dvou cest se určuje pomocí Hausdorffovy vzdálenosti. Nová cesta je uložena jen tehdy, pokud báze neobsahuje cestu jí podobnou. V práci je uvedeno několik strategií zapomínání případů (zapomínání nejméně používaných případů, průměrných případů nebo nejlepších případů). Pokud není v bázi případů nalezena přímo cesta z počáteční do cílové buňky, hledání nové cesty je potom ihned zkoušeno zřetězením cest v případové bázi. Pokud není možné nalézt cestu pro konkrétní počáteční nebo cílovou buňku, je hledána cesta pro jejich nejbližší sousední buňky. Adaptace případu potom spočívá v dohledání chybějících úseků navrženou pomocnou metodou.

3.3.5 Posilované učení

Posilované učení bylo zmíněno již v podkapitole 3.3.1, kde bylo použito pro učení vah neuronové sítě. Posilované učení lze však také použít přímo pro plánování cesty.

Práce (Drummond 2002) navrhuje systém, který zrychluje takovýto výpočet cesty tím, že jako hodnoty předpokládaných odměn používá naučené hodnoty z dříve řešených podobných případů. O této metodě se dá hovořit také jako o systému případového usuzování, kde je jako druhé metody použito algoritmu Q-učení.

Algoritmus Q-učení předpokládá diskrétní stavy i akce. Stavovým prostorem v práci Drummond (2002) je tedy mřížka a povolenými akcemi je osm možných směrů pohybu robotu v této mřížce. Q-hodnota zde souvisí se vzdáleností do cíle. Akce a je v každém stavu s vybrána podle příslušné maximální Q-hodnoty, nebo náhodně z důvodu prozkoumávání stavového prostoru. Aplikací akce a na stav s přejdeme do nového stavu s' a podle úspěšnosti získáme posílení r. Odměna v parametru r nastane jen tehdy, pokud je s' cílovým stavem. Aktualizace příslušné Q-hodnoty je provedena podle vztahu

$$Q_{s,a}^{t+1} = (1-\alpha)Q_{s,a}^{t} + \alpha(r + \gamma \max_{a'} Q_{s',a'}^{t}), \tag{3.3}$$

kde α je parametr učení a γ je srážkový faktor. Výběr akce a aktualizace příslušné Q-hodnoty jsou opakovány v každém časovém kroku *t*.

Naučené Q-hodnoty vztahující se k příslušné části scény jsou přiřazeny k rovinnému grafu. Každé místnosti zkoumané scény totiž odpovídá rovinný graf, jehož uzly mohou být rohy místnosti, dveře a případně také cíl. Všechny tyto informace jsou uloženy v případové bázi. Při změně cíle ve scéně je pomocí těchto grafů vyhledán podobný případ a do příslušných oblastí stavového prostoru jsou po transformaci přidány naučené Q-hodnoty. Tím dostaneme nové pole Q-hodnot. Sestavení těchto Q-hodnot mohou doprovázet chyby, a proto nové Q-hodnoty zřejmě nezajistí správný výsledek. Je tedy nutné na tyto hodnoty aplikovat opět proces učení, který bude ovšem dosahovat podstatně kratších časů, čímž je celý proces nalezení řešení urychlen.

Algoritmus Q-učení je použit také v práci (Kamei 2004), jehož parametry jsou nastavovány pomocí genetických algoritmů. Autoři tímto zrychlili fázi učení až o 30% a snížili počet akcí potřebných k dosažení cíle o polovinu.

3.4 Neholonomní plánování cesty

O *neholonomním* plánování cesty se hovoří tehdy, jsou-li na robot kladena diferenciální omezení, která nejsou plně integrovatelná, tzn., že nemohou být převedena do omezení nezahrnujících derivaci (LaValle 2006). Neholonomní plánování zahrnuje jak plánování s kinematickými, tak s dynamickými omezeními. Pro častější kinematické plánování platí, že $X = C$, což pro dynamické plánování již neplatí. Stavový prostor X zde bývá tvořen fázovým prostorem konfiguračního prostoru C a potom tedy stav $x = (q, \dot{q})$ pro $q \in C$.

Diferenciální omezení udávají povolenou rychlost v každém stavu. Takto představují *lokální* omezení, která tvoří doplněk ke globálním omezením (viz úvod kapitoly 3). Základní metody plánování uvedené v podkapitolách 3.2 a 3.3 ignorují tato lokální omezení a předpokládají, že diferenciální omezení budou řešena až v průběhu provádění plánu.

Diferenciální omezení je často vyjádřeno jako $\dot{x} = \varphi(x, u)$ na spojitém stavovém prostoru X. Jedná se vlastně o v čase spojitý protějšek k přechodové funkci $x' = f(x, u)$ (LaValle 2006). Spojitá přechodová funkce $\varphi(x, u)$ udává místo nového stavu robotu *x* jeho rychlost v tomto stavu. Budoucí stav, který splňuje diferenciální omezení, je získán integrací rychlosti. Výslednou cestu charakterizuje *akční trajektorie* $\tilde{u} : \langle 0, \infty) \to U$, kde U je akční prostor $U \subseteq R^m$. Akce v konkrétním čase *t* je vyjádřena jako $\tilde{u}(t)$. Pro stavově závislé modely se předpokládá splnění podmínky $\tilde{u}(t) \in U(x(t)) \subseteq U$. Z akční trajektorie lze odvodit s*tavovou trajektorii* $\tilde{x}$ pomocí integrace

$$\tilde{x}(t) = \tilde{x}(0) + \int_0^t \varphi(\tilde{x}(t'), \tilde{u}(t'))dt', \tag{3.4}$$

která integruje $\dot{x} = \varphi(x, u)$ z počátečního stavu $\tilde{x}(0)$. Pro výpočet je často užívána numerická integrace (např. Eulerova metoda či metody Runge-Kutta).

Prostor stavů robotu je obvykle charakterizován orientovaným grafem $G = (V, E)$, kde V je množina uzlů odpovídajících stavům robotu a E je množina hran reprezentujících akční trajektorie $e : \langle 0, \Delta t \rangle \to U$. Obvykle každé hraně grafu odpovídá jedna akce. Počáteční vrchol

hrany reprezentuje výchozí stav a koncový vrchol hrany odpovídá stavu, který získáme aplikací této akce na výchozí stav po dobu Δt. Jestliže tedy máme nějakou hranu e a jí odpovídající akci u, pak platí $e(t) = u$ pro všechna $t \in \langle 0, \Delta t \rangle$. Stavovou trajektorii hrany $\tilde{x}_e$ získáme pomocí (3.4), kde $\tilde{u}(t) = e(t)$.

Neholonomní plánovací metody vycházejí z metod založených na vzorkování. Exaktní metody se zde neuplatňují, neboť nejsou schopny s diferenciálními omezeními pracovat. Obecně se dají používané metody pracující s diferenciálními omezeními popsat následujícími kroky (LaValle 2006):

1. **Inicializace:** Nechť $G = (V, E)$ representuje orientovaný graf, kde V obsahuje počáteční stav x_I a další možné stavy v X_{free} a kde E je prázdná množina hran.
2. **Výběr uzlu pro expanzi:** Vybereme stav $x_{cur} \in \bigcup_{e \in E} \bigcup_{t \in \langle 0, \Delta t \rangle} x_e(t)$. Pokud je množina E prázdná, položíme $x_{cur} = x_I$.
3. **Lokální plánování:** Generujeme akci u takovou, aby $\tilde{x}(0) = x_{cur}$ a $\tilde{x}(\Delta t) = x_r$ pro nějaké $x_r \in X_{free}$, které může nebo nemusí být vrcholem grafu G. Vzniklá stavová trajektorie musí obsahovat uzly, které náleží X_{free}. Není-li tomu tak, opakujeme krok 2.
4. **Vložení hrany do grafu:** Akci u reprezentující akční trajektorii e vložíme do E. Pokud $x_{cur} \notin V$ nebo $x_r \notin V$, pak je do V přidáme. Pokud x_{cur} obsahuje stavová trajektorie $\tilde{x}_e$ některé hrany $e \in E$, potom je nutno tuto hranu rozdělit podle nového uzlu x_{cur}.
5. **Kontrola řešení:** Ověříme, zda graf G neobsahuje hledanou cestu. Pokud ano, algoritmus končí úspěšně. V některých případech je možné ve výsledné stavové trajektorii tolerovat malou mezeru.
6. **Návrat zpět na krok 2:** Je-li splněna nějaká jiná ukončovací podmínka, než nalezení řešení, algoritmus končí neúspěchem. V opačném případě se vrátíme na krok 2.

Výběr uzlů x_{cur} a x_r již záleží na konkrétní metodě. LaValle (2006) uvádí tři takové přístupy: *začlenění diskretizace stavového prostoru*, metody *RDT* a *prohledávání na mřížce*, které je však především určeno pro plánování cesty s dynamickými omezeními.

3.4.1 Systematické inkrementální vzorkování

LaValle (2006) o této metodě hovoří jako o metodě se začleněním diskretizace stavového prostoru. Tato metoda uvažuje graf G jako stromový graf, který je konstruován inkrementálně z x_I prováděním systematické expanze.

Pro zrychlení doby výpočtu je nejprve rozložen stavový prostor X do kolekce buněk D, bez uvažování případných kolizí. Nejčastěji se jedná o n-rozměrné buňky stejné velikosti, které jsou získány kvantováním každé souřadnice. Pro buňku $d \in D$ platí, že $d \subset X$. Pokud je vrchol prohledávaného grafu G obsažen v buňce d, pak je buňka d označena jako *navštívená*. Na počátku generování grafu G jsou všechny buňky označeny jako *nenavštívené*. Prohledávací algoritmus této metody popisuje (LaValle 2006) následujícími kroky (startovní konfigurace je x_I, cílová konfigurace je x_G):

1. Do prioritní fronty Q vložíme startovní stav x_I.
2. Do grafu G vložíme startovní stav x_I.
3. Pokud je x_G navštívený, pak algoritmus ukončíme úspěchem. Pokud je $Q = \emptyset$, pak ukončíme algoritmus neúspěchem.
4. Z fronty Q vyjmeme stav x_{cur}.
5. Po dobu Δt aplikujeme na stav x_{cur} akci u. Pokud vzniklá stavová trajektorie neobsahuje kolizní stavy, pak pro koncový stav x_r této trajektorie opakujeme kroky 6 až 9. Krok 5 opakujeme pro všechny přípustné akce.
6. Pokud je buňka obsahující x_r navštívená, pak pokračujeme další akcí kroku 5.
7. Do fronty Q vložíme stav x_r.
8. Do grafu G přidáme vrchol x_r a hranu odpovídající akci u.
9. Označíme buňku obsahující x_r jako navštívenou.
10. Pokračujeme krokem 3.

Pokud cesta není nalezena, pak je nutné prohledávací algoritmus opakovat s jemnějším rozložením stavového prostoru nebo zkrácením doby Δt. Pokud cesta existuje, pak je nalezena. Nevýhodou je však velmi dlouhá doba výpočtu.

Řazení prvků prioritní fronty Q může být prováděno např. podle součtu délky nalezené cesty ze startu do prvku fronty a odhadem vzdálenosti mezi prvkem fronty a cílovým stavem. V tomto případě se v podstatě jedná o prohledávání algoritmem A*.

Na použití *algoritmu A** a jeho modifikací je založena řada dalších metod plánování cesty pro neholonomní roboty. Metoda, kterou navrhli Pruski a Rohmer (1997), spočívá v modelování prostoru přípustných konfigurací množinou hranolů a následném nalezení trajektorie mezi dvěma konfiguracemi algoritmem A*. Hodnotící funkce bere do úvahy neurčitosti, které by se mohly vyskytnout během navigace. Graf et al. (2001) používají algoritmus A* pro nalezení nejkratší cesty v *redukovaném grafu viditelnosti*. Tato nejkratší cesta je vyhodnocena pro zjištění, zda může být použita jako referenční cesta k vytvoření přípustné cesty pro daný mobilní robot. Jestliže ne, je tato cesta vyřazena a je vybrána a vyhodnocena následující nejkratší cesta. Tento krok se opakuje tak dlouho, dokud není nalezena vhodná referenční cesta. Nakonec je nalezená cesta přizpůsobena geometrickým a kinematickým vlastnostem robotu. Podsedkowski et al. (2001) prezentují metodu založenou na A* pro prohledávání grafu s uzly umístěnými v diskretizovaném konfiguračním prostoru. Analyzují různé heuristické nákladové funkce a uvádějí novou formulaci takové funkce. Jejich metoda je vybavena procedurami pro rychlé přeplánování cesty. Metoda Linkera a Basse (2008) se opírá o konfigurační prostor vozidla a používá algoritmus A* k určení optimální cesty, přičemž bere do úvahy omezení specifická pro typ vozidla a uvažovaného prostředí. Tato omezení jsou vyjádřena jako penalizační členy vážené uživatelsky definovanými parametry, které odrážejí typ cesty implicitně hledané uživatelem.

3.4.2 Metody RDT

Zkratka RDT znamená *rychle rostoucí husté stromy* (Rapidly-exploring Dense Trees). Metody založené na RDT se tedy snaží o rychlou konstrukci prohledávacího stromu G. Tyto

metody předpokládají nekonečnou hustou posloupnost α obsahující vzorky stavů z X. (LaValle 2006) uvádí následující obecný algoritmus tvorby grafu G:

1. Do grafu G vložíme startovní stav x_I.
2. Pro i nabývající hodnot 1 až k (počet iterací k je předem dán) opakujeme kroky 3 až 5.
3. Ze stavových trajektorií všech hran grafu G vybereme stav x_{cur} takový, aby vzdálenost mezi x_{cur} a $\alpha(i)$ byla minimální (pokud graf ještě neobsahuje žádné hrany, je $x_{cur} = x_I$). Jestliže stav x_{cur} je vnitřním stavem stavové trajektorie některé hrany, je tato hrana rozdělena a stav x_{cur} je přidán do G.
4. Na stav x_{cur} aplikujeme po dobu Δt akci u takovou, aby koncový bod vzniklé trajektorie x_r byl nejblíže stavu $\alpha(i)$.
5. Do grafu G přidáme vrchol x_r a hranu odpovídající akci u.

Nejznámější metodou RDT je metoda RRT, což znamená *rychle rostoucí náhodné stromy* (Rapidly-exploring Random Trees). Tuto metodu podrobně rozebírají LaValle a Kuffner (2001). Metoda RRT volí $\alpha(i)$ zcela náhodně a kroky 3 až 5 opakuje tak dlouho, dokud není dosaženo cílového stavu. Konvergence k cíli je však v tomto případě příliš pomalá a tak bylo navrženo několik vylepšení. Nejjednodušší vylepšení nazvané *RRT-GoalBias* (RRT-B) upravuje $\alpha(i)$ tak, že s velmi malou pravděpodobností (asi 0,05) vrací namísto náhodného stavu stav cílový. Další vylepšení je nazváno *RRT-GoalZoom* (RRT-Z). V tomto případě namísto náhodného stavu vrací $\alpha(i)$ s velmi malou pravděpodobností konfiguraci z okolí cílového stavu. Velikost okolí odpovídá vzdálenosti mezi cílovým stavem a nejbližším vrcholem grafu G. Vylepšení *RRT-GoalRegionBiasedConf* (RRT-R) je navrženo v práci (Qu 1999). V tomto vylepšení vrací $\alpha(i)$ s velmi malou pravděpodobností cílový stav, v opačném případě vrací náhodnou konfiguraci s pravděpodobností 0,5, jinak je vrácena náhodná konfigurace z okolí cílového stavu. Velikost okolí je určena stejným způsobem jako u RRT-Z.

Szadeczky-Kardoss a Kiss (2006) rozšiřují metodu RRT o *klíčové konfigurace*, které jsou potenciálně důležitými body cesty.

3.4.3 Genetické algoritmy

Existuje také skupina metod pro neholonomní plánování cesty založených na *genetických algoritmech*. Liu et al. (2004), na rozdíl od dále zmíněných autorů, používají chromozomy pevné délky tvořené následovně. Nejprve se zkonstruuje úsečka spojující startovní a cílový bod a pak se rozdělí do 32 stejných částí. Následně se v dělicích bodech zkonstruuje 32 vertikálních přímek. Každý chromozom se skládá ze 32 genů, které jsou situovány na vertikálních přímkách. Každý gen reprezentuje bod jeho souřadnicemi a stupněm nepřípustnosti. Funkce fitness je založena na váženém součtu délky cesty a stupně nepřípustnosti. Jsou používány dva genetické operátory: křížení a mutace. Jestliže nový chromozom nesplňuje neholonomní omezení, je pokládán za nepodařený a je vymazán.

Chromozom v algoritmu Chenga et al. (2006) je sekvencí řídicích vstupů, kde každý gen reprezentuje úhel natočení. Funkce fitness bere do úvahy nejen délku cesty, ale také její hladkost. Genetické operátory zahrnují dva obvyklé operátory – křížení a mutaci, a nový

operátor – růst. Operátory křížení a mutace jsou upraveny tak, aby vyhovovaly danému problému. Růstový operátor způsobuje, že chromozom roste tak, aby dosáhl cíle.

Erinc a Carpin (2007) používají chromozomy složené z genů obsahujících následující informace: konfiguraci (souřadnice a orientaci), dvě akční proměnné (rychlost a úhel natočení) a booleovskou proměnnou nesoucí informaci o přípustnosti. Jejich přístup začíná s cestami splňujícími neholonomní omezení a zajišťuje, že během evoluce nejsou generovány žádné cesty, které by tato omezení porušovaly. Pro vytvoření počátečních chromozomů je použit algoritmus RRT, ale bez kontroly kolizí. Plánovač tedy vytváří řešení, která obecně mohou protínat překážky. Genetické operátory zahrnují křížení a čtyři druhy mutace a užívají obousměrný RRT pro spojování segmentů cest. Fitness řešení je definována jako převrácená hodnota nákladů, které jsou dány váženým součtem čtyř příspěvků charakterizujících cestu: délky, bezpečnosti, hladkosti a přípustnosti.

Hung et al. (2007) studují evoluční plánovací strategie pro hladký pohyb mobilních robotů po efektivních nekolizních cestách ve známých statických prostředích. Náklady každé kandidátské cesty sestávají z délky cesty a váženého součtu hloubek průniku do polygonálních překážek. Cesta je složena z množiny segmentů kubické spirály s omezenou křivostí spojených předem specifikovaným počtem mezilehlých konfigurací. Algoritmus používá následující operátory: aritmetické křížení (lineární interpolaci mezi dvěma chromozomy), mutaci a manipulační operátor operující na nepřípustných segmentech cest, které protínají překážky.

3.4.4 Ostatní přístupy

Již v práci (Barraquand a Latombe 1993) je zmíněno, že nalezení přípustné volné cesty pro neholonomní robot je mnohem obtížnější než nalezení volné cesty pro holonomní robot mající stejnou geometrii a stejný rozměr konfiguračního prostoru. Uvedení autoři analyzují řiditelnost neholonomních robotů a navrhují plánovač cesty, který generuje cestu zkoumáním konečné podmnožiny konfiguračního prostoru definovaného diskretizací řídicích parametrů robotu. Dokazují, že tento plánovač je asymptoticky optimální a úplný (tj. jestliže problém nalezení cesty má řešení, pak je plánovač najde, pokud je diskretizace prohledávaného prostoru dostatečně jemná).

Divelbiss a Wen (1997) prezentují přístup založený na metodách teorie řízení. Nejprve uvažují problém plánování cesty bez překážek, transformují jej do nelineárního problému nejmenších čtverců v rozšířeném prostoru a pak jej iterativně řeší. Vyhýbání se překážkám je zahrnuto ve formě omezení typu nerovností. Pro konverzi nerovností na rovnice jsou použity externí penalizační funkce a pak je aplikována táž nelineární metoda nejmenších čtverců.

Marchese (2002) uvádí algoritmus založený na směrovém šíření odpuzujících a přitahujících hodnot potenciálu v modelu vícevrstvého celulárního automatu. Tento algoritmus nalézá všechny optimální nekolizní trajektorie sledující údolí minim potenciálové hyperplochy začleněné do 4D prostoru a tvořené s ohledem na zadaná omezení.

Přístup, který navrhli Barrios-Aranibar a Alsina (2004), je založen na posilovaném učení. Metoda Lianga et al. (2005) generuje cesty tvořené nejvýše pěti segmenty: nejvýše dvěma segmenty maximálně zakřivené kubické spirály s nulovou křivostí na obou koncích ve spojení s až třemi lineárními segmenty. Tato metoda je založena na řešení úlohy lineárního programování.

K plánování cesty neholonomních robotů se využívá i metody rojení částic. V práci (Saska 2006) je tato metoda použita pro plánování cesty neholonomního robotu typu auta.

Cesta je reprezentována řetězcem kubických splajnů a plánování cesty je ekvivalentní optimalizaci parametrů těchto splajnů. Chen a Li (2006) navrhují zdokonalený algoritmus rojové optimalizace pro plánování hladkých cest vhodných pro neholonomní roboty. Navržený algoritmus má vysokou schopnost prozkoumávání stavového prostoru, takže může pracovat s menší velikostí hejna a tak redukovat výpočetní náročnost. V práci (Zhang 2007) je pro plánování pohybu neholonomních robotů navržen hybridní algoritmus hejnové optimalizace s mutací, která zajišťuje lepší prohledání stavového prostoru.

4. Návrhy metod plánování cesty

4.1 Plánování cesty na mřížce

V této podkapitole jsou navrženy metody pro plánování cesty robotu v pravoúhlé mřížce. Kromě podkapitoly 4.1.5 je uvažován holonomní robot. Doménou těchto navržených metod je oblast případového usuzování.

4.1.1 Okolní prostředí robotu

Pravoúhlou mřížku $[1, m] \times [1, n]$ získáme pomocí vhodné rastrové metody diskretizace. Buňka c této mřížky je určena dvojicí souřadnic: $c = (x, y)$, kde $x \in \{1, 2, \ldots, m\}$, $y \in \{1, 2, \ldots, n\}$. Pro jednoduchost uvažujeme čtvercový tvar buněk o velikosti větší než je velikost robotu. Každé buňce c je přiřazena hodnota $r(c) \in \langle 0; 1]\rangle$, která reprezentuje míru nebezpečí buňky. Tato hodnota může být interpretována jako pravděpodobnost toho, že je buňka obsazena překážkou nebo jako stupeň blízkosti buňky k překážce. Pokud je $r(c) = 1$, buňka nemůže být pro pohyb použita. Hodnota $r(c)$ by měla být aktualizována po každém průchodu buňkou. Dále také předpokládáme, že se robot pohybuje konstantní rychlostí jen ve vodorovném, svislém a úhlopříčném směru. Tyto směry zde ovšem nejsou reprezentovány akcemi, jak definují LaValle a Kuffner (2001), ale jsou kódované čísly $w \in \{0, 1, \ldots, 7\}$. Tato reprezentace byla vybrána z důvodu kompatibility s použitým genetickým algoritmem, který byl použit jako pomocná metoda případového usuzování. Namísto termínu konfigurace jsou v podkapitole 4.1 také používány termíny buňka nebo bod.

Vzdálenost (nezohledňující překážky) mezi body $c_i = (x_i, y_i)$ a $c_j = (x_j, y_j)$ může být definována jedním z následujících vztahů:

$$d(c_i, c_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{4.1}$$

$$d(c_i, c_j) = |x_i - x_j| + |y_i - y_j| \tag{4.2}$$

$$d(c_i, c_j) = \max\{|x_i - x_j|, |y_i - y_j|\} \tag{4.3}$$

$$d(c_i, c_j) = \min\{|x_i - x_j|, |y_i - y_j|\}(\sqrt{2} - 1) + \max\{|x_i - x_j|, |y_i - y_j|\} \tag{4.4}$$

Vzdálenost definovaná vztahem (4.4) odpovídá tomu, že je povolen pouze vodorovný, svislý a úhlopříčný směr pohybu. Pomocí vztahu (4.3) definujme *okolí* $N(c_i, \delta)$ buňky c_i jako:

$$N(c_i, \delta) = \{c_j \mid d(c_i, c_j) \le \delta\} \tag{4.5}$$

Cesta $P(c_s, c_g)$ ze startovní buňky c_s do cílové buňky c_g je definována jako trojice (c_s, c_g, W), kde $W = \{w_1, w_2, \ldots, w_n\}$ a $w_i \in \{0, 1, \ldots, 7\}$. Číslice w_i kóduje směr pohybu robotu z aktuální buňky do sousední. Každá cesta je charakterizována hodnotou funkce $F(P)$, která může být definována takto:

$$F(P) = \lambda_1 L(P) + \lambda_2 D(P) + \lambda_3 R(P) \tag{4.6}$$

kde λ_1, λ_2 a λ_3 jsou kladné váhy a $L(P)$, $D(P)$, $R(P)$ jsou délka, obtížnost a nebezpečnost cesty P. Definujme

$$L(P) = \sum_{i=1}^{n} \varphi(w_i), \quad R(P) = \frac{1}{2} \sum_{i=1}^{n} \left(r(c_i) + r(c_{i+1})\right) \varphi(w_i) \tag{4.7}$$

kde $c_1 = c_s$, $c_{n+1} = c_g$ a

$$\varphi(w_i) = \begin{cases} \sqrt{2} & \text{pokud } w_i \text{ je liché} \\ 1 & \text{jinak} \end{cases} \tag{4.8}$$

Obtížnost cesty může být vyjádřena např. počtem změn směru

$$D(P) = \sum_{i=1}^{n-1} \psi(w_i, w_{i+1}), \qquad \text{kde } \psi(w_i, w_{i+1}) = \begin{cases} 0 & \text{pokud } w_i = w_{i+1} \\ 1 & \text{jinak} \end{cases} \tag{4.9}$$

4.1.2 Případově orientované plánování cesty

Uvažujeme-li systém případového usuzování, pak případy reprezentují cesty, které robot absolvoval. Cesta P je v případové bázi uložena s hodnotou funkce $F(P)$, která charakterizuje průjezdnost této cesty. Pokud pro danou počáteční buňku c_s a danou cílovou buňku c_g báze případů neobsahuje cestu vedoucí z c_s do c_g, vyhledává se potom cesta podobná. Kruusmaa a Svensson (2003) určují podobnou cestu vztahem:

$$P(c'_s, c'_g) = \arg\min\left\{F\left(P(c_1, c_2)\right) \,\middle|\, c_1 \in N(c_s, \delta) \wedge c_2 \in N(c_g, \delta)\right\} \tag{4.10}$$

To znamená, že se podobná cesta vyhledává jen z uložených kompletních cest. Přestože Kruusmaa a Svensson (2003) uvažují pro znovupoužití i zřetězení navazujících kompletních cest, je stupeň využití minulých zkušeností značně redukován. Tuto nevýhodu se snaží řešit následující algoritmus, který získává podobnou cestu i z částí uložených cest. Tento algoritmus byl publikován v práci (Dvořák, Krček & Samohýl 2004).

Nalezení cesty v bázi případů i s její adaptací je prováděna prostřednictvím rekurzivní funkce *FindBest* s parametry L, c_s a c_g (L je dočasný seznam případů). Algoritmus této funkce lze popsat následujícími kroky.

1. Pokud $c_s \in N(c_g, \delta)$, hledáme cestu z c_s do c_g prostřednictvím GA a algoritmus ukončíme.
2. Pro každý případ C ze seznamu L určíme segment $C' = (c'_1, c'_2, W')$ případu C, kde

   $$c'_1 = \arg\min\left\{d(c_s, c) \,\middle|\, c \in C\right\}, \; c'_2 = \arg\min\left\{d(c_g, c) \,\middle|\, c \in C\right\} \tag{4.11}$$

   Spočteme také ohodnocení tohoto segmentu $G(C') = \lambda\, d(c_s, c'_1) + F(C') + \lambda\, d(c'_2, c_g)$, kde λ je kladná konstanta.
3. Určíme nejlepší segment

   $$C^* = \arg\min\left\{G(C') \,\middle|\, c'_1 \neq c'_2 \wedge \left(c'_1 \in N(c_s, \delta) \vee c'_2 \in N(c_g, \delta)\right)\right\} \tag{4.12}$$

   Pokud C^* neexistuje nebo $G(C^*)$ není možné akceptovat, hledáme cestu z c_s do c_g prostřednictvím GA a algoritmus ukončíme.
4. Případ korespondující se segmentem C^* je odstraněn ze seznamu L a jeho části bez segmentu C^* jsou přidány do seznamu L. Tímto získáme seznam L'. Spojíme cesty $P_1^* = FindBest(L', c_s, c'_1)$, C^* a $P_2^* = FindBest(L', c'_2, c_g)$ prostřednictvím GA a algoritmus ukončíme.

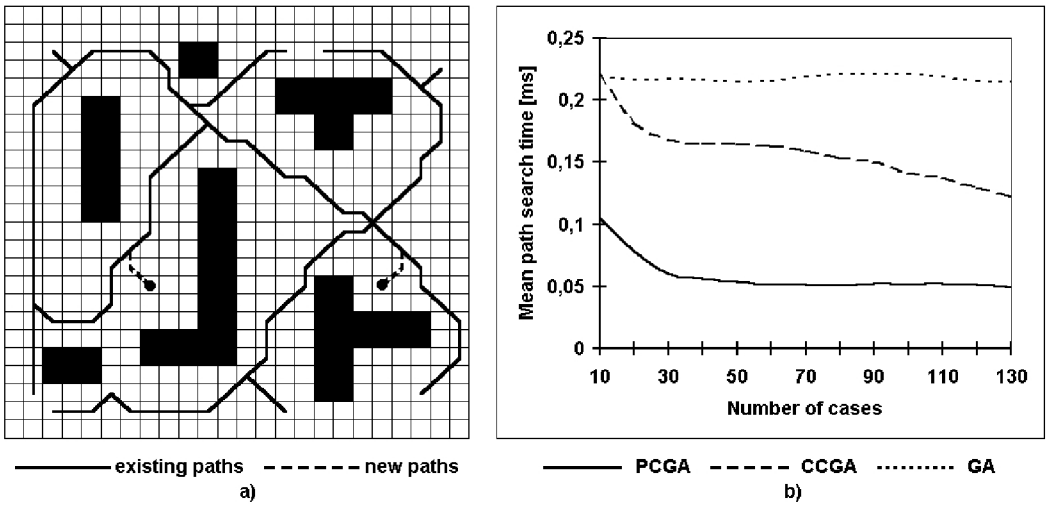
Pokud cesta $P^* = FindBest(L, c_s, c_g)$ není akceptovatelná, potom hledáme novou cestu z c_s do c_g prostřednictvím genetického algoritmu.

Nechť P je cesta absolvovaná robotem. Tato cesta může být odlišná od cesty nalezené výše popsaným algoritmem, protože systém lokální navigace ji může v případě výskytu neznámé nebo dynamické překážky změnit. Cesta je uchována pouze tehdy, pokud báze případů neobsahuje žádný případ, který je podobný P a má nižší ohodnocení. Podobné případy s vyšším ohodnocením budou cestou P nahrazeny. Stejně jako Kruusmaa a Svensson (2003), používáme míru podobnosti založenou na *Hausdorffově vzdálenosti*

$$D(P_1, P_2) = \max_{c_i \in P_1} \min_{c_j \in P_2} d(c_i, c_j) \tag{4.13}$$

Případ C je podobný případu P, jestliže platí $D(C, P) \leq \varepsilon$.

V práci (Dvořák, Krček & Samohýl 2004) byly pro srovnání implementovány tři metody plánování cesty: metoda založená pouze na GA, metoda kombinující kompletní podobné případy určené vztahem (4.10) s výsledky GA (CCGA) a navrženou metodu kombinující části případů s výsledky GA (PCGA). Z obr. 4.1b je zřejmé, jak doba hledání cesty závisí na počtu případů v případové bázi. V těchto experimentech byla uvažována mřížka 50×50 a GA byl ukončen po nalezení první cesty. Obr. 4.1a ilustruje znovupoužití existujícího případu při konstruování cesty mezi dvěmi danými buňkami.



Obr. 4.1. *(a) Znovupoužití částí případů při plánování cesty, (b) Závislost doby hledání cesty na počtu případů v případové bázi*

4.1.3 Genetický algoritmus

Jako pomocná metoda algoritmu navrženého v podkapitole 4.1.2 byl zvolen genetický algoritmus. Pro jednoduchost byl uvažován algoritmus, který je implementovaný v práci (Sedláček 2000). Tato přístup používá chromozómy pevné délky, která závisí na velikosti prohledávaného prostoru. Chromozómy jsou posloupnosti $S = \{w_1, w_2, \ldots, w_N\}$, kde každý gen $w_i \in \{0, 1, \ldots, 7\}$ kóduje směr pohybu robotu do sousední buňky. Každý chromozóm reprezentuje cestu z dané počáteční buňky c_s, ale tato cesta nemusí obsahovat cílovou buňku c_g. Pokud je cílová buňka dosažena dříve, než jsou vyčerpány všechny geny, jsou zbývající geny ignorovány.

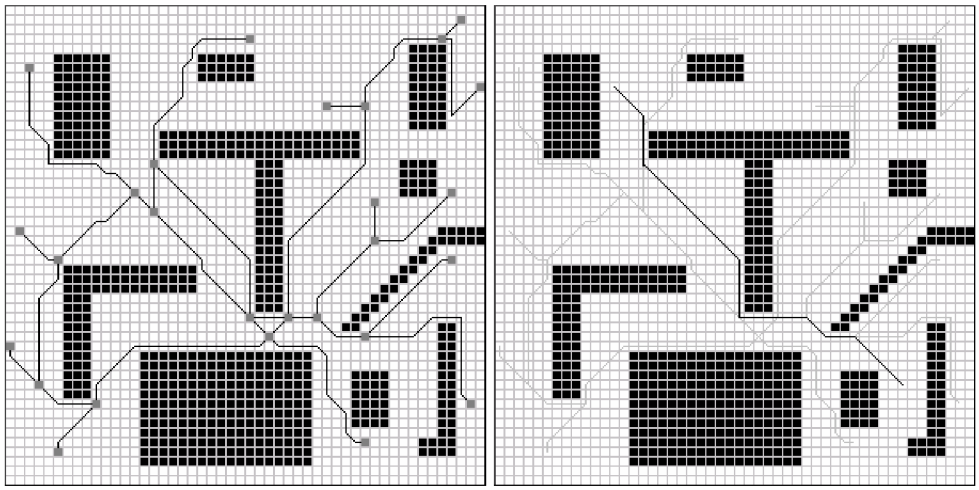
Počáteční populace je generována náhodně. Kvalitu chromozómů určují dvě fitness funkce aplikované v následujícím pořadí. První funkce reprezentuje vzdálenost mezi koncovou buňkou cesty a cílovou buňkou c_g, druhá funkce reprezentuje cenu této cesty. Algoritmus používá binární turnajovou selekci, uniformní křížení, mutaci jednoho náhodně generovaného genu a inkrementální výměnu populace. Výpočet algoritmu je ukončen po dosažení daného počtu generací. V kombinaci s případovým usuzováním, kde genetické algoritmy vyhledávají části výsledné cesty, je počet generací a velikost chromozomů určována v závislosti na vzdálenosti $d(c_s, c_g)$ dané vztahem (4.3).

V případě nenalezení cesty pomocí genetického algoritmu v daném časovém limitu se vytvoří z mřížky graf, jehož uzly tvoří všechny buňky mřížky bez pevných překážek a hrany odpovídají všem možným směrům pohybu robotu do sousedních buněk. Ohodnocení těchto hran jsou součtem vzdáleností sousedních buněk vypočtených podle (4.1) a z ceny těchto buněk. K vyhledání cesty v takto získaném grafu je potom použito Dijkstrova algoritmu.

4.1.4 Případový graf

Na přístupu, ve kterém jsou jako případy ukládány celé cesty (viz podkapitola 4.1.2), lze pozorovat nedostatek plynoucí ze dvou faktů: izolovanost (případy jsou izolovány, nejsou spojeny v žádné struktuře) a nadbytečnost (v případové bázi jsou uloženy jen kompletní cesty, i když obsahují stejné části cest).

Tyto nevýhody mohou být odstraněny prostřednictvím struktury zvané případový graf (*case graph*), která byla použita v práci (Haigh 1994) pro spojité prostředí. Po absolvování cesty je cesta do báze případů uložena buďto celá jako jeden případ, nebo je rozdělena do několika částí, které jsou uloženy jako samostatné případy (pokud se ovšem již tyto případy nebo případy jim podobné v bázi případů nevyskytují). Segmenty cest (uložené případy) mohou mít společné pouze své krajní body. Pokud cesta protíná nějaký již existující případ, pak tato cesta i případ jsou rozděleny do menších případů tak, aby byla splněna výše uvedená podmínka. Případy (segmenty) reprezentují hrany výsledného grafu a uzly tohoto grafu jsou krajními body těchto segmentů (obr. 4.2).



Obr. 4.2. *Případový graf a nově nalezená cesta pomocí tohoto grafu*

Algoritmus pracující nad případovým grafem byl navržen v práci (Krček, Dvořák & Hodál 2005), kde byl kombinován s genetickým algoritmem popsaným v podkapitole 4.1.3. V práci (Dvořák & Krček 2005) je jako pomocná metoda volen již algoritmus A* nebo Dijkstrův algoritmus (DA), neboť v implementaci s haldou dosahují tyto algoritmy kratší doby výpočtu než zmíněný genetický algoritmus. DA i A* prohledávají graf určený mřížkou takto: vrcholy jsou takové buňky c, pro které platí $r(c) < 1$ a hrany korespondují s možnými pohyby do sousedních buněk. Tyto hrany jsou ohodnoceny hodnotou funkce $F\big(P(c_1, c_2)\big)$, kde c_1 a c_2 jsou sousední buňky. Jako heuristické funkce algoritmu A* může být použito vzdáleností (4.1) nebo (4.2).

Navržený algoritmus pracující nad případovým grafem lze popsat následujícími kroky (c_s je počáteční buňka, c_g je cílová buňka):

1. Jestliže $c_s \in N(c_g, \delta)$, najdeme cestu z c_s do c_g prostřednictvím algoritmu A* a algoritmus ukončíme.
2. Určíme množinu případů, které protínají okolí buňky c_s a množinu případů protínajících okolí buňky c_g:

$$E_s = \{C \mid C \cap N(c_s, \delta) \neq \varnothing\},\ E_g = \{C \mid C \cap N(c_g, \delta) \neq \varnothing\} \tag{4.14}$$

3. Pokud jsou obě množiny E_s a E_g neprázdné, dočasně modifikujeme případový graf G takto: rozšíříme tento graf o uzel c_s a přidáme hrany spojující tuto buňku s nejbližšími buňkami případů obsažených v E_s. Tyto hrany jsou hledány v mřížce prostřednictvím A* a jsou ohodnoceny hodnotou funkce $F(P)$ podle (4.6). Stejné rozšíření provedeme pro cílovou buňku c_g. V modifikovaném grafu G' hledáme optimální cestu spojující buňky c_s a c_g.
4. Pokud tato cesta existuje a je akceptovatelná, algoritmus ukončíme. Pokud není akceptovatelná (jako kritérium může být použita např. obtížnost cesty nebo poměr délky cesty k Euklidovské vzdálenosti), zkoušíme nalézt lepší cestu bez použití případového grafu a algoritmus ukončíme.
5. Jestliže cesta v G' neexistuje (graf G' není souvislý), nebo aspoň jedna z množin E_s a E_g je prázdná, hledáme cestu z c_s do c_g v mřížce prostřednictvím algoritmu A* a algoritmus ukončíme.

V prvních verzích tohoto algoritmu, bylo také zkoušeno používat případový graf v situacích uvažovaných v kroku 5, ale tato verze poskytovala horší výsledky než algoritmy A* a DA.

Popisovaný algoritmus může být zjednodušen zjemněním struktury případového grafu (Dvořák & Krček 2005). "Jemná struktura" případového grafu je konstruována takto: vrcholy jsou buňky uložených cest a hrany jsou dány dvojicí sousedních buněk těchto cest. Tento algoritmus je složen z následujících kroků:

1. Konstruujeme graf G' jako sjednocení případového grafu G a okolí $N(c_s, \delta)$ a $N(c_g, \delta)$.
2. V grafu G' hledáme optimální cestu spojující buňky c_s a c_g. Pokud tato cesta existuje a je akceptovatelná, algoritmus ukončíme.
3. V opačném případě, hledáme cestu z c_s do c_g bez použití případového grafu a algoritmus ukončíme.

Problematika plánování cesty v mřížce je dále rozebírána v práci (Hodál, Dvořák & Krček 2005), kde je uvažován případový graf, který je kombinován s heuristickým lokálním hledáním.

4.1.5 Zjednodušené plánování cesty pro neholonomní robot

Metody popsané v předcházející podkapitole jsou také použitelné pro neholonomní roboty, které jsou schopny rotace. V tomto případě může být vyjádřena obtížnost cesty jako počet změn směrů:

$$D(P) = \sum_{i=1}^{n-1} \min\{|w_i - w_{i+1}|, 8 - |w_i - w_{i+1}|\} \tag{4.15}$$

Pokud neholonomní robot není schopen rotace a má omezený poloměr otáčení, algoritmus prohledávání grafu použitý pro plánování cesty musí být modifikován. Důsledkem tohoto omezení je to, že robot nemůže pokračovat z uzlu přes libovolnou hranu. Volba následující hrany závisí na tom, která hrana vstupující do daného uzlu byla použita. Prohledávání grafu je založeno na úpravě Dijkstrova algoritmu, která byla původně navržena v práci (Krček & Dvořák 2006) pro spojité prostředí. Modifikace tohoto algoritmu pro mřížku byla publikována v práci (Dvořák & Krček 2006).

Pro jednoduchost budeme označovat uzly grafu (buňky mřížky) namísto symbolu c_i indexem i. Nechť $G = (V, E)$ je ohodnocený orientovaný graf, kde V je množina uzlů a E je množina orientovaných hran, $E \subseteq \{(i, j) \mid i, j \in V\}$. Případový graf popsaný v předcházející podkapitole je neorientovaný. Snadno však může být transformován do orientovaného grafu výměnou každé neorientované hrany $\{i, j\}$ párem orientovaných hran (i, j) a (j, i).

Zaveďme následující symboly:

$\delta_i(i, j)$ směr přímého segmentu hrany (i, j) začínající v uzlu i s orientací od tohoto uzlu do hrany; jedná se o hodnotu z množiny $\{0, 1, \ldots, 7\}$;

$\delta_j(i, j)$ směr přímého segmentu hrany (i, j) začínající v uzlu j s orientací od tohoto uzlu do hrany;

$e(i, j)$ ohodnocení hrany (i, j); toto ohodnocení je určeno vztahem (4.6);

$d_s(i, j)$ vzdálenost z uzlu s do uzlu j přes hranu (i, j); pokud neexistuje cesta z uzlu s do uzlu j přes hranu (i, j), potom $d_s(i, j) = \infty$; vzdálenost $d_s(i, j)$ znamená minimální ohodnocení všech cest z s do j přes (i, j) a ohodnocení cesty je dáno součtem ohodnocení všech hran této cesty;

$p_s(i, j)$ počáteční uzel hrany bezprostředně předcházející hraně (i, j) na nejkratší cestě z uzlu s do uzlu j přes hranu (i, j); hodnota $p_s(i, j) = -1$ znamená, že tato cesta z uzlu s do uzlu j přes hranu (i, j) neexistuje;

s' fiktivní předchůdce uzlu startu s; platí $\delta_s(s', s) = (w_s + 4) \bmod 8$ kde w_s je směr robotu v uzlu s;

$E(i)$ množina hran incidentních s uzlem i;

E_D množina hran jejichž hodnoty $d_s(i, j)$ jsou konečné;

Nalezení optimální cesty z uzlu startu s do cílového uzlu g může být popsáno následujícími kroky (předpokládáme, že počáteční směr robotu v uzlu s je dán):

1. Položíme $d_s(s', s) = 0$ a $d_s(i, j) = \infty$, $p_s(i, j) = -1$ pro každou hranu (i, j). Množina $E_D = \varnothing$, $(k, r) = (s', s)$.
2. Hledáme všechny hrany $(r, i) \in E - E_D$ splňující podmínku $P(k, r, i)$. Pokud $d_s(r,i) > d_s(k,r) + e(r,i)$, potom položíme $d_s(r,i) = d_s(k,r) + e(r,i)$, $p_s(r,i) = k$.
3. Hledáme takovou hranu $(p, q) \notin E_D$, pro kterou $d_s(p,q) = \min\{d_s(i,j) \mid (i,j) \notin E_D\}$. Pokud $d_s(p,q) = \infty$, potom algoritmus ukončíme (z uzlu s není dosažitelná další hrana a cesta z s do g neexistuje). V opačném případě, vložíme hranu (p, q) do E_D a položíme $(k, r) = (p, q)$.
4. Pokud $E(g) \subset E_D$, potom pokračujeme krokem 5 (cesty do cíle přes všechny jeho sousedy byly nalezeny). V opačném případě pokračujeme krokem 2.
5. Položme $S = \{\}$. Hledáme takovou hranu (j, g), pro kterou platí $d_s(j,g) = \min\{d_s(i,g) \mid (i,g) \in E_D\}$ a položme $k = g$.
6. Vložme hranu (j, k) na začátek seznamu S. Pokud $j = s$, potom algoritmus ukončíme (seznam S representuje optimální cestu).
7. Položme $i = p_s(j,k)$, $k = j$, $j = i$ a pokračujeme krokem 6.

Modifikovaný A* algoritmus se liší od výše uvedeného krokem 3, kde hrana $(k, r) \notin E_D$ je určena tak, že $d_s(k,r) + d(k,r) = \min\{d_s(i,j) + d(i,j) \mid (i,j) \notin E_D\}$, kde $d(i,j)$ je definováno vztahem (1) nebo (2).

Podmínka $P(i, j, k)$ v kroku 2 pro robot pohybující se z hrany (i, j) do hrany (j, k) závisí na charakteru neholonomních omezení. Pokud se robot může pohybovat jen vpřed nebo diagonálně vpřed, potom definujeme tuto podmínku jako:

$$\min\{\alpha(i,j,k), 8 - \alpha(i,j,k)\} \le 1 \tag{4.16}$$

kde

$$\alpha(i,j,k) = \left|\left(\delta_j(i,j) + 4\right) \bmod 8 - \delta_j(j,k)\right| \tag{4.17}$$

Pokud se robot může pohybovat také vzad a diagonálně vzad, potom podmínku $P(i, j, k)$ definujeme takto:

$$\begin{gathered} \min\{\alpha(i,j,k), 8 - \alpha(i,j,k)\} \le 1 \vee \\ \vee \exists (j,l) : \min\{\alpha(i,j,l), 8 - \alpha(i,j,l)\} \le 1 \wedge \min\{\alpha(l,j,k), 8 - \alpha(l,j,k)\} \le 1 \end{gathered} \tag{4.18}$$

V případě pohybu vzad nebo diagonálně vzad je zapotřebí přidat k výrazu $d_s(k,r) + e(r,i)$ v kroku 2 ohodnocení segmentu hrany (j, l), kterou robot musí použít před pohybem zpět do hrany (j, k).

4.2 Plánování cesty ve spojitém prostoru

V této podkapitole jsou navrženy metody pro plánování cesty robotu ve spojitém prostoru. Metody zde navržené provádí nejen vlastní prohledávání konfiguračního prostoru,

ale především diskretizaci tohoto prostoru. Jsou zde představeny jak metody použitelné pro holonomní roboty, tak i metody pro neholonomní roboty.

4.2.1 Kombinace případového grafu a pravděpodobnostních stromů

Po prvních pokusech s algoritmem RRT byl v práci (Krček & Dvořák 2006) představen přístup pro plánování cesty robotu typu „neholonomní tříkolka s diferenciálním řízením“, který spočívá v kombinaci případového usuzování s RRTs. Konfigurační prostor je zde $C = X = R^2 \times \langle 0, 2\pi \rangle$ a složkami konfigurace jsou souřadnice polohy geometrického středu robotu ve 2D pracovním prostředí a úhel natočení robotu. Pro zjednodušení předpokládáme, že se robot může pohybovat pouze konstantní rychlostí a je pro něj dán minimální poloměr otáčení. Díky tomuto zjednodušení můžeme cestu snadno popsat *triviální hranou*, což je vlastně křivka v R^2, která odpovídá dráze geometrického středu robotu. Triviální hrana se skládá nejvýše ze dvou úseček a nejvýše z jednoho kruhového oblouku o velikosti úhlu menším než π a poloměru větším než minimální poloměr otáčení robotu (poloměr oblouku volíme co největší). Triviální hraně grafu může odpovídat jedna až tři akce $u \in U \subseteq R^2$.

Klasický algoritmus RRTs, popsaný např. v (LaValle 2001), modifikujeme tak, že umožňuje vyhledat cesty z počáteční konfigurace $x_s \in X_{free}$ nejen do jedné, ale do více cílových konfigurací obsažených v množině $Q \subset X_{free}$. Tato úprava má za cíl nalezení cest z dané konfigurace do blízkých uzlů případového grafu. Součástí množiny Q může být i původně zadaná cílová konfigurace x_g. Je nutno ale poznamenat, že zadaný počet iterací K by měl být při hledání cest do blízkých uzlů případového grafu (a eventuálně konfigurace x_g) podstatně nižší, než když hledáme cestu pouze do konfigurace x_g.

Modifikovaný algoritmus RRT sestává z následujících kroků:

1. Jako kořen stromu T zvolíme x_s Potom v K iteracích opakujeme kroky 2 až 8.
2. Jako konfiguraci x_{rand} zvolíme se zadanou pravděpodobností p náhodně vygenerovanou konfiguraci z X nebo s pravděpodobností $1 - p$ náhodně vybranou konfiguraci z množiny cílů Q.
3. Jako konfiguraci x_{near} zvolíme jednu konfiguraci z T takovou, pro kterou je vzdálenost mezi x_{near} a x_{rand} minimální a pro kterou mezi konfiguracemi x_{near} a x_{rand} existuje triviální hrana. Vzdálenost počítáme jako euklidovskou vzdálenost mezi polohami středu robotu.
4. Nebyla-li konfigurace x_{near} v kroku 3 nalezena, pak jdeme na krok 2. Pokud $x_{rand} \in Q$, pak jdeme na krok 7.
5. Novou konfiguraci x_{new} určíme jako $x_{new} = \Phi(x_{near}, x_{rand}, \varepsilon)$. Výsledkem funkce Φ je konfigurace ležící na triviální hraně mezi x_{near} a x_{rand} taková, že délka úseku hrany mezi x_{near} a x_{new} je rovna ε. Pokud délka triviální hrany je menší než ε, pokračuje se krokem 2.
6. Pokud není žádná z konfigurací na triviální hraně mezi x_{near} a x_{new} v kolizi, pak strom T rozšíříme o uzel x_{new} a o triviální hranu mezi x_{near} a x_{new}. Pokračujeme krokem 8.
7. Jestliže není žádná z konfigurací na triviální hraně mezi x_{near} a x_{rand} v kolizi, pak strom T rozšíříme o uzel x_{rand} a o triviální hranu mezi x_{near} a x_{rand}. Konfiguraci x_{rand} vyřadíme z množiny Q.
8. Pokud již proběhlo K iterací nebo je $Q = \emptyset$, pak algoritmus ukončíme, jinak jdeme na krok 2.

Jestliže algoritmus nalezl cestu ze startu do cílové konfigurace, snadno ji nyní získáme procházením stromu T od cílového uzlu směrem ke kořenu. Cíle, do kterých algoritmus cestu nenalezl, představují zbylé prvky v množině Q. Cesta $P(x_0, x_k)$ ze startovní konfigurace $x_0 = x_s$ do cílové konfigurace x_k je definována jako orientovaný sled konfigurací a hran x_0, h_1, x_1, h_2, x_2, …, h_k, x_k, kde každá hrana h_i představuje triviální hranu spojující konfigurace x_{i-1} a x_i. Předpokládáme, že robot může absolvovat tutéž cestu i v opačném směru otočením všech jejích konfigurací o úhel π. Příklad použití modifikovaného algoritmu RRT ukazuje obr. 4.3.

Báze případů je organizována jako *případový graf* $G = (V, E)$, kde V je množina uzlů, $V = \{0, 1, 2, \ldots, n\}$ a E je množina orientovaných hran $E \subseteq \{(i, j) \mid i, j \in V\}$, přičemž každý případ je reprezentován dvojicí opačně orientovaných hran. Při ukládání úspěšně absolvované cesty robotu do báze případů se jako samostatný případ ukládá každá triviální hrana cesty. K uložení jednotlivých hran ovšem může dojít pouze tehdy, pokud se v bázi případů ještě nevyskytují podobné případy. Uzly případového grafu odpovídají konfiguracím robotu ohraničujícím jednotlivé triviální hrany a jsou ohodnoceny souřadnicemi středu robotu. Pokud hrana v případovém grafu protíná jinou hranu, potom se snažíme propojit tyto hrany (pokud je to možné) doplněním dalších uzlů a triviálních hran.

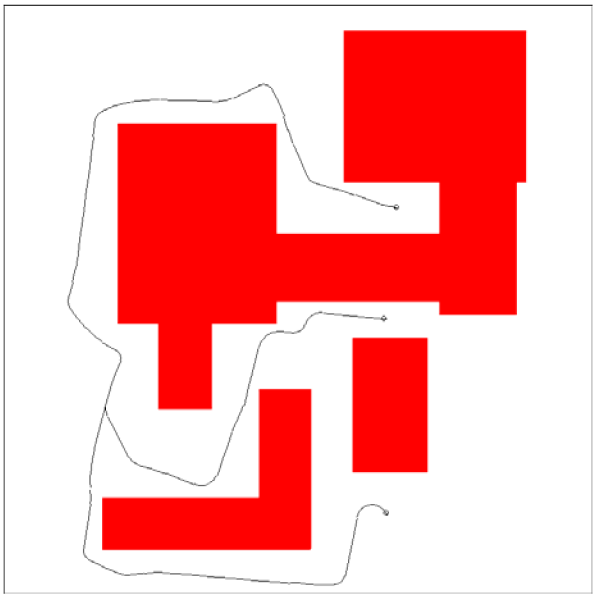
Každá triviální hrana je v bázi případů uložená s hodnotou funkce $e(i, j) = f(l, r)$, která reprezentuje cenové ohodnocení daného případu. Parametr l je délka hrany a parametr r charakterizuje nebezpečnost (nebo obtížnost) hrany a měl by být upravován během každého skutečného průchodu touto hranou. Dále je hrana charakterizována dvojicí úhlů $\alpha_i(i, j)$ a $\alpha_j(i, j)$, které svírají směrové vektory tečny hrany (i, j) v uzlech i a j s osou x. Směrový vektor tečny v uzlu směřuje dovnitř hrany.

Vyhledávání podobných cest je založeno na okolí konfigurace robotu. Definujme okolí $V(x_i, \delta)$ konfigurace x_i v grafu G jako množinu uzlů, jejichž euklidovská vzdálenost od geometrického středu robotu v konfiguraci x_i je menší než dané δ.

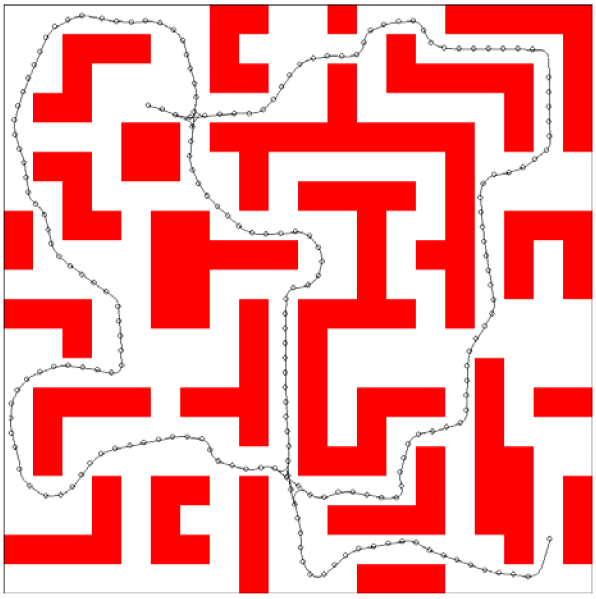
Námi navrhovaný algoritmus pracující nad případovým grafem lze popsat následujícími kroky (x_s je počáteční konfigurace, x_g je cílová konfigurace):

1. Určíme okolí konfigurací x_s a x_g v případovém grafu G:
$$V_s = V(x_s, \delta),\ V_g = V(x_g, \delta)$$
2. Pokud je některá z množin V_s nebo V_g prázdná, cestu z x_s do x_g najdeme pomocí RRT a algoritmus ukončíme.
3. Dočasně modifikujeme případový graf G takto: rozšíříme tento graf o uzel korespondující s x_s a přidáme nekolidující triviální hrany spojující tuto konfiguraci s nejbližšími konfiguracemi vzniklými z uzlů obsažených ve V_s. Z každého uzlu $i \in V_s$ vznikne tolik konfigurací, kolik je v grafu G hran vycházejících z tohoto uzlu. Natočení robotu v konfiguraci odpovídající uzlu i a hraně (i, j) je rovno úhlu $\alpha_i(i, j)$. Všechny konfigurace z V_s, pro které neexistují nekolidující triviální hrany z x_s, vložíme do množiny Q a do této množiny vložíme i konfiguraci x_g. Nyní spustíme RRT algoritmus se startem x_s a množinou cílů Q. Všechny takto nalezené cesty přidáme do modifikovaného grafu G'. Všechny přidané hrany jsou ohodnoceny svými délkami.
4. Stejné rozšíření jako v kroku 3 provedeme pro cílovou konfiguraci x_g. Protože předpokládáme možnost absolvování cesty v opačném směru, jako startovací konfiguraci volíme x_g otočenou o π.

5. V modifikovaném grafu G' použijeme pro hledání optimální cesty spojující x_s a x_g upravený Dijkstrův algoritmus, popsaný v následujícím odstavci. Nenašel-li tento algoritmus cestu, vyhledáme ji pomocí RRT a algoritmus ukončíme.



Obr. 4.3. *Cesty do dvou cílů vyhledané algoritmem RRT*



Obr. 4.4. *Příklad případová báze*

Následující úprava Dijkstrova algoritmu je nutná z toho důvodu, že při hledání optimální cesty v grafu nemůžeme v důsledku kinematických omezení robotu pokračovat z aktuálního uzlu libovolnou hranou. Volba navazující hrany závisí na tom, po jaké hraně se robot do aktuálního uzlu dostal.

Symbolem $d_s(j, k)$ je označena vzdálenost z uzlu s do uzlu k přes hranu (j, k) v grafu $G = (V, E)$. Jestliže neexistuje cesta z uzlu s do uzlu k přes hranu (j, k), klademe

$d_s(j, k) = \infty$. Označme symbolem $p_s(i, j)$ počáteční uzel hrany bezprostředně předcházející hraně (i, j) na nejkratší cestě z uzlu s do uzlu j přes hranu (i, j). Je-li $p_s(i, j) = -1$, znamená to, že cesta z uzlu s do uzlu j přes hranu (i, j) neexistuje. Označme symbolem D množinu hran, o nichž víme, že hodnota $d_s(j, k)$ je již definitivní. Dále značme symbolem s' fiktivního předchůdce uzlu s a položme $\alpha_s(s', s)$ rovno úhlu počátečního natočení robotu zvětšeného o π. Symbolem $E(i)$ označme množinu hran, které incidují s uzlem i.

Výpočet vzdáleností z uzlu s do uzlu g přes všechny jeho sousedy (při dané orientaci robotu v uzlu s) lze popsat následujícími kroky:

1. Polož $d_s(s', s) = 0$ a pro každou hranu (i, j) polož $d_s(i, j) = \infty$, $p_s(i, j) = -1$. Dále polož $D = \emptyset$, $(k, r) = (s', s)$.
2. Prozkoumej všechny hrany $(r, i) \in E - D$, pro něž $\alpha_r(r, i) = \alpha_r(k, r) + \pi$. Jestliže $d_s(r, i) > d_s(k, r) + e(r, i)$, pak polož $d_s(r, i) = d_s(k, r) + e(r, i)$, $p_s(r, i) = k$.
3. Je-li $E(g) \in D$, pak konec (byly nalezeny cesty do cíle přes všechny jeho sousedy). V opačném případě najdi hranu $(k, r) \notin D$ takovou, že $d_s(k, r) = \min\{d_s(i, j) \mid (i, j) \notin D\}$.
4. Je-li $d_s(k, r) = \infty$, pak konec (žádná další hrana již není dostupná z vrcholu s). V opačném případě zařaď hranu (k, r) do množiny D a pokračuj krokem 2.

Předpokládejme, že uzel g je takový, že existuje hrana $(i, g) \in D$. Cesta robotu z konfigurace x_s do konfigurace x_g bude určena jako posloupnost konfigurací a hran uložených v zásobníku Z, přičemž počáteční konfigurace se bude nacházet na vrcholu zásobníku. Určení nejkratší cesty z x_s do x_g lze popsat takto:

1. Polož $Z = \emptyset$. Urči hranu (j, g) takovou, že $d_s(j, g) = \min\{d_s(i, g) \mid (i, g) \in D\}$. Polož $k = g$.
2. Vlož konfiguraci x_k určenou uzlem k a úhlem $\alpha_k(j, k) + \pi$ do zásobníku Z.
3. Je-li $k = s$ a $j = s'$, pak konec.
4. Vlož hranu (j, k) do zásobníku Z.
5. Polož $i = p_s(j, k)$, $k = j$, $j = i$ a pokračuj krokem 2.

Nevýhodou navrženého přístupu je vysoká vazba na model robotu, ale bohužel i řešení podsystémů doplňování nových případů a údržby báze případů v aktuálním stavu, které se během vývoje ukázalo být příliš komplikované. Za určitou nevýhodu může být považováno také to, že tato metoda neuvažuje natočení robotu v cílové pozici. Pozornost byla tedy zaměřena na holonomní plánování ve spojitém prostoru a následnou transformaci řešení pro neholomní robot.

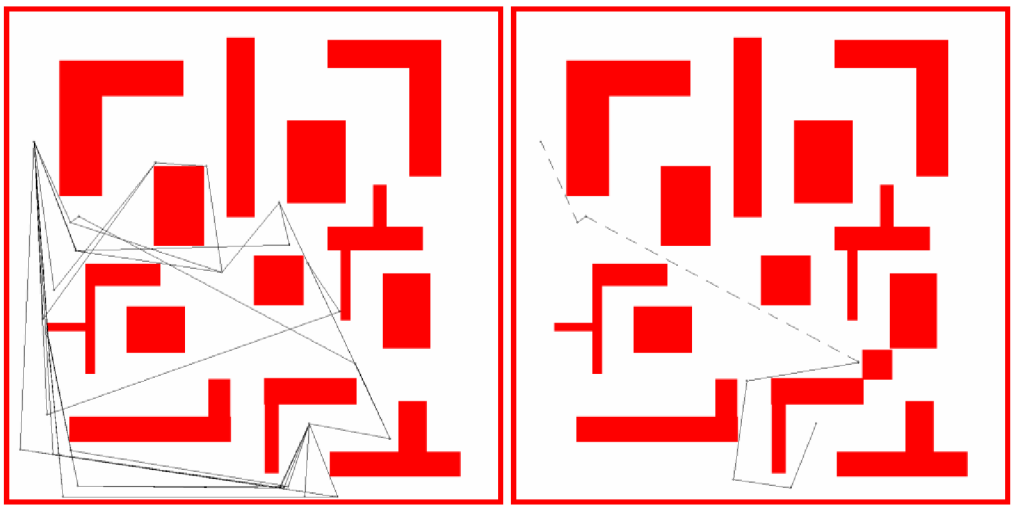
4.2.2 Genetický algoritmus

Již v práci (Krček & Dvořák 2007) je navržen genetický algoritmus, u nějž byla zkoumána možnost použití části „naučené“ populace z nějakého předchozího řešení jako část počáteční populace nové úlohy. Při znovupoužití vhodné části „naučené“ populace se tato adaptace ukázala být účelná především u úloh, kde je nutné aktuální řešení přeplánovat v důsledku výskytu náhodné překážky (obr. 4.5). V práci (Dvořák & Krček 2008) je tento genetický algoritmus vylepšen o další specifické operátory.

Plánování cesty probíhá ve dvourozměrném spojitém prostoru, v němž se vyskytují polygonální překážky popsané svými vrcholy. Dále je předpokládán holonomní robot, který je pro jednoduchost reprezentován jako bod v prostoru. Pro uvažování rozměrů holonomního

robotu je nutné zvětšit překážky tak, aby se robot mohl bezpečně pohybovat po hranách těchto zvetšených překážek.

Chromozom vyjadřuje cestu v prostoru a jeho geny představují uzlové body, ve kterých cesta mění svůj směr. Každý gen je přitom tvořen dvojicí souřadnic x a y. U všech chromozomů platí, že počáteční gen reprezentuje pozici startu a poslední gen pozici cíle. Cesta je tudíž tvořena úsečkami spojujícími sousedící geny chromozomu. Chromozom má proměnlivou délku v závislosti na složitosti cesty.



Obr. 4.5. *„Naučená" populace a nalezená cesta při změně v prostředí*

Počáteční populace je získána náhodným generováním, avšak do chromozomů jsou generovány pouze takové uzly, které leží mimo překážky. Počáteční délka každého chromozomu je rovna součtu dané minimální délky a náhodně zvoleného čísla z daného rozsahu. Velikost počáteční populace je dána a zůstává stejná i v následujících generacích.

Fitness funkce je definována dvěma způsoby: pro nepřípustné cesty a pro přípustné cesty. Fitness funkce pro nepřípustnou cestu je dána vztahem:

$$F(P) = f_{collisions}(P) + 1 \tag{4.19}$$

kde $f_{collisions}(P)$ je počet kolizí cesty P s překážkami. Fitness funkce přípustné cesty P zohledňuje její délku, počet genů a její hladkost takto:

$$F(P) = w_1 f_{length}(P) + w_2 f_{nodes}(P) + w_3 f_{smooth}(P) \tag{4.20}$$

kde w_k jsou nezáporné normalizované váhy (jejich součet je roven 1),

$$f_{length}(P) = 1 - \frac{d(s,g)}{\sum_{j=1}^{n-1} d(j, j+1)},\quad f_{nodes}(P) = 1 - \frac{2}{n},\quad f_{smooth}(P) = 1 - \frac{\sum_{j=2}^{n-1} \alpha(j)}{(n-2)\pi} \tag{4.21}$$

kde $d(s,g)$ je Euklidovská vzdálenost mezi startem a cílem, $d(j, j+1)$ je délka segmentu mezi uzly j a $j + 1$, n je počet uzlů cesty a $\alpha(j)$ je úhel mezi segmenty $(j - 1, j)$ a $(j, j + 1)$.

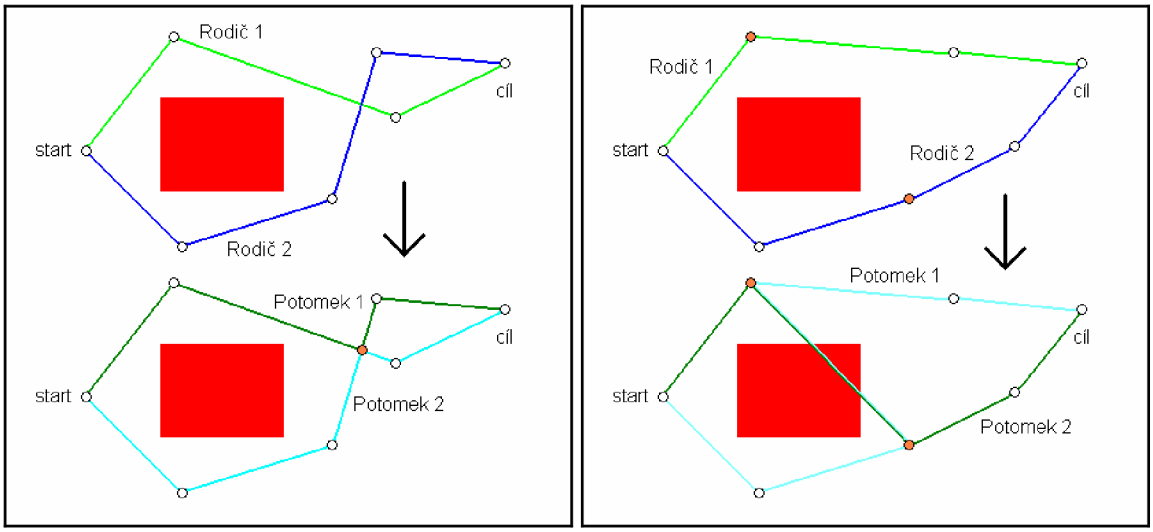
Dílčí kritéria jsou normalizována tak, že jejich nejlepší (ideální) hodnota je rovna 0 a horní mez jejich hodnot je rovna 1. Protože i váhy w_k jsou normalizované, hodnoty fitness funkce (4.20) leží v intervalu $\langle 0; 1 \rangle$ s nejlepší hodnotou rovnou 0. Minimální hodnota vztahu (4.19) koresponduje s horní mezí tohoto intervalu. Cílem plánování cesty je nalézt přípustnou cestu minimalizací fitness funkce (4.20). Díky fitness funkci (4.19) zrychlíme dobu výpočtu, neboť pro nepřípustné cesty není nutné počítat hodnoty dílčích kritérií.

Dále je navržen následující systém operátorů:

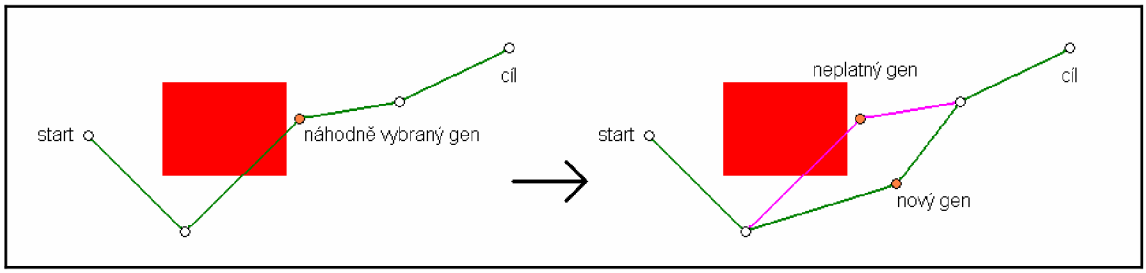
- Pro každý pár vybraných rodičů (přípustných i nepřípustných) je prováděno *křížení* s danou, obvykle velmi vysokou pravděpodobností. Nejprve se určí všechny geometrické průsečíky cest těchto dvou chromozomů. Je-li počet těchto průsečíků větší než nula, pak se náhodně vybere jeden z nich. Tento průsečík slouží jako dělicí místo, které rozdělí chromozom na dvě části. Potomci potom vzniknou tak, že se spojí prvá část chromozomu prvního rodiče se druhou částí druhého rodiče a prvá část chromozomu druhého rodiče se druhou částí prvního rodiče. Mezi obě části se do potomků vloží nový gen, reprezentující vybraný průsečík (obr. 4.6a). Byl-li však počet průsečíků roven nule, pak se dělicí místo v obou rodičovských chromozomech určí jako náhodně vybraný gen. Vkládání nového genu se v tomto případě již neprovádí (obr. 4.6b). V obou případech se jedná se o variantu jednobodového křížení, operátor je možné však snadno modifikovat pro křížení vícebodové.
- Operátor *mutace* náhodně vybere jeden gen z daného chromozomu a náhodně změní jeho souřadnice (obr. 4.7). Změněný gen musí ležet mimo uvažovanou cestu a mimo překážky. Jsou navrženy dva druhy mutace. První z nich (*velká mutace*) může být aplikována na přípustné i nepřípustné cesty a vzdálenost nové polohy genu od minulé není omezena. Druhá (*malá mutace*) je aplikována jen na přípustné cesty a provádí jemnou změnu souřadnic náhodně vybraného genu tak, aby příslušné segmenty byly nekolizní.
- *Opravný* operátor (obr. 4.8a) je aplikován na přípustné chromozomy. Náhodně vybraný kolizní segment (křížící překážku) je vyjmut a na jeho místo je vložena cesta obcházející tuto překážku (podél hran zvětšené překážky).
- Operátor *výměny* může být aplikován na přípustné i nepřípustné chromozomy. Eliminuje dvě následné ostré zatáčky zaměněním souřadnic vybraných sousedících genů (obr. 4.8b).
- Operátor *vyhlazení* je aplikován jen na přípustné chromozomy a udržuje jejich přípustnost. Tento operátor vyhlazuje cestu seříznutím ostré zatáčky. Gen příslušný ostré zatáčce je odstraněn a na jeho místo jsou vloženy dva geny ležící na segmentech vycházejících z odstraněného genu (obr. 4.9a).
- Operátor *zkrácení* je aplikován jen na přípustné chromozomy a udržuje jejich přípustnost. Pokud je to možné, operátor vyjme část cesty mezi dvěma vybranými uzly (obr. 4.9b).
- Operátor *odstranění* je použit v opravném operátoru a v operátorech vyhlazení a zkrácení. Může být však aplikován odděleně na přípustné cesty na základě nějakých heuristických znalostí. Např. je možné odstranit uzel, který téměř nemění směr cesty nebo uzly, které jsou příliš blízko sousednímu uzlu.
- Operátor *zlepšení* systematicky aplikuje na přípustné chromozomy operátory zkrácení a vyhlazení.

Změna populace je prováděna inkrementálně. Prostřednictvím binární turnajové selekce je pro křížení vybrán daný počet párů. Po křížení jsou na získané potomky aplikovány zbývající operátory. Tito potomci jsou přidáni do staré populace a poté se provede seřazení jedinců v populaci podle hodnoty fitness. Pro získání nové populace se provede odstranění chromozomů s nejnižší hodnotou fitness tak, aby velikost populace odpovídala velikosti počáteční populace. Tím je zajištěno, že nejlepší chromozomy staré populace postoupí do populace nové.

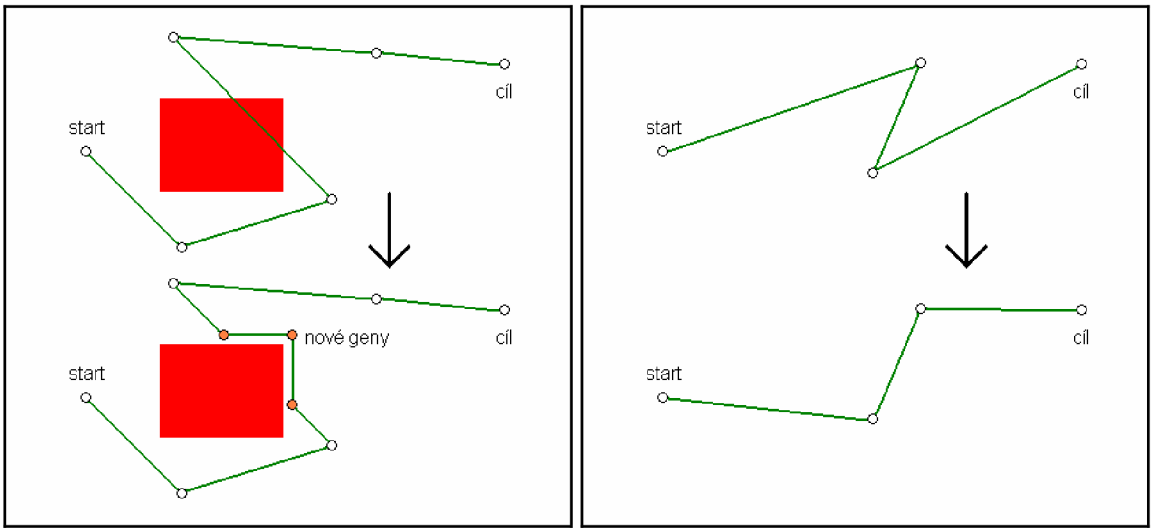
Kritérium ukončení algoritmu může být určeno daným počtem generací nebo podmínkou, že nejlepší hodnota fitness v generacích se již nemění více než o malou konstantní hodnotu.



Obr. 4.6. *Princip operátoru křížení*

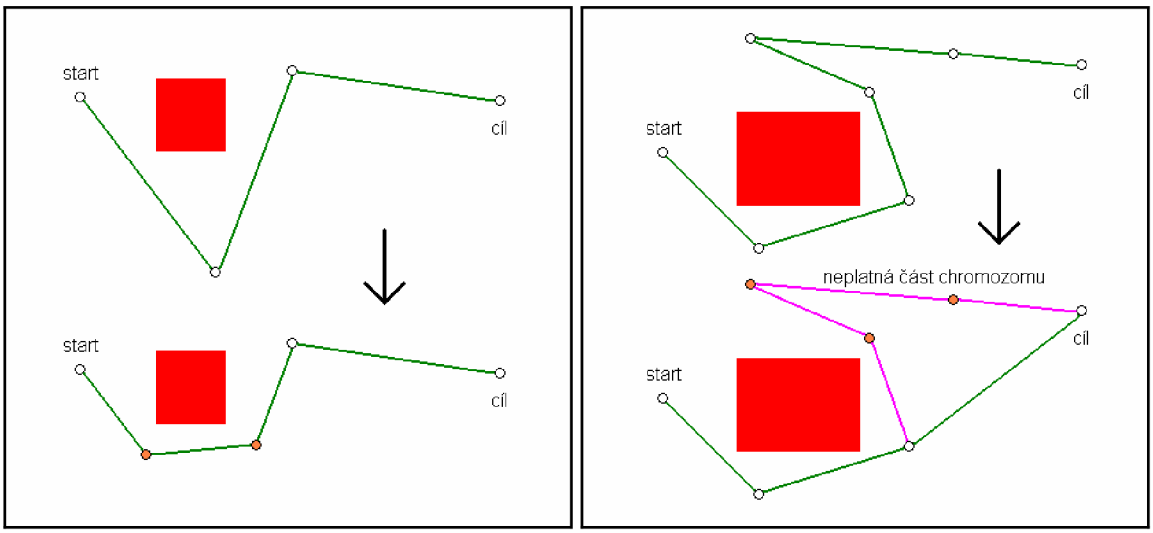


Obr. 4.7. *Princip operátoru mutace*



(a) (b)

Obr. 4.8. *Principy operátorů opravy (a) a výměny (b)*

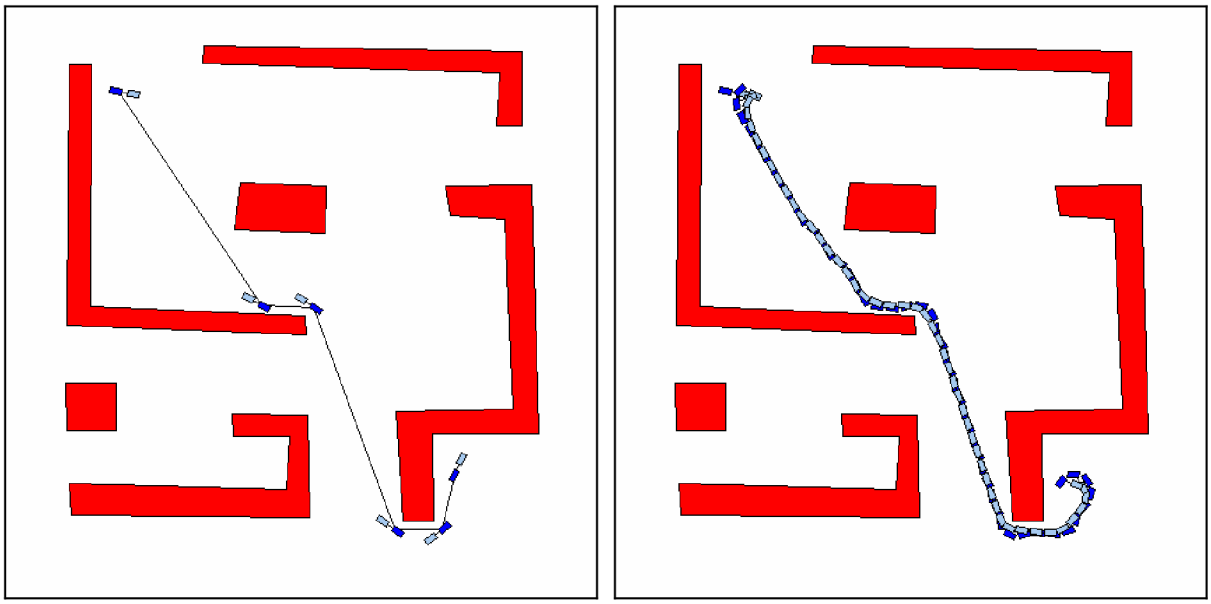


(a) (b)

Obr. 4.9. *Principy operátorů vyhlazení (a) a zkrácení (b)*

4.2.3 Opravný algoritmus

Cesty navržené metodou popsanou v předcházející kapitole nelze použít pro neholonomní roboty. Proto je v pracích (Krček & Dvořák 2009) a (Dvořák & Krček 2009) navržen opravný algoritmus, který diskretizuje konfigurační prostor a modifikuje cestu nalezenou pro holonomní robot pomocí heuristické funkce (obr. 4.10b). Tento algoritmus vytváří stromový graf, kde uzly reprezentují konfiguraci robotu a hrany reprezentují odpovídající akce mezi uzly. Kořen stromového grafu odpovídá počáteční konfiguraci robotu. Expandován je vždy uzel s nejlepší hodnotou heuristické funkce.



(a) (b)

Obr. 4.10. *Cesta nalezená GA (a) a opravená cesta pomocí modifikovaného A* pro robot typu auto s přívěsem (b)*

Cesta nalezená pro holonomní robot je reprezentována posloupností souřadnic pozice robotu, opravný algoritmus pro neholonomní robot však pracuje s konfiguracemi, které musí obsahovat minimálně souřadnice těžiště robotu x, y a natočení robotu θ. Podle typu robotu může konfigurace obsahovat další složky, které jsou potřebné např. pro řešení kolizí. Natočení robotu ve startu a cíli je dáno, ostatní natočení dopočítáme (natočení je určeno směrnicí přímky kolmé k ose úhlu mezi sousedícími hranami cesty, obr. 4.10a). Tím dostaneme seznam konfigurací P, kde první konfigurace odpovídá startu a poslední cíli.

Cesta respektující neholonomní omezení může být modifikována A* algoritmem. Tento algoritmus konstruuje strom, jehož uzly jsou ohodnoceny následující funkcí:

$$f(Uzel) = g(Start, Uzel) + d(Uzel, Cíl) + M \tag{4.22}$$

kde $g(Start, Uzel)$ je délka cesty ze startu do daného uzlu, $d(Uzel, Cíl)$ je Euklidovská vzdálenost z daného uzlu do cíle, a M je pokuta dána vztahem $M = N_P K$, kde N_P je počet konfigurací v seznamu P a K je dostatečně velká konstanta. Tato pokuta je postupně snižována, když dojde k dosažení podcílů (tj. prvků seznamu P). Tímto způsobem jsou zvýhodněny akce vedoucí k cíli. Když použijeme následující ohodnocení

$$f(Uzel) = d(Uzel, Goal) + M \tag{4.23}$$

obdržíme modifikaci algoritmu B*, který je popsán např. v (Berliner 1979).

Kromě seznamu P algoritmus používá dva další seznamy: seznam *OPEN* (seznam uzlů, které nebyly dosud expandovány; seznam je implementován jako halda) a seznam *CLOSED* (seznam expandovaných uzlů). Uzly v těchto dvou seznamech jsou reprezentovány následujícími strukturami: (konfigurace, ohodnocení konfigurace, rodičovská konfigurace, akce, kterou byla konfigurace dosažena z rodičovské konfigurace).

Algoritmus hledání cesty může být popsán následujícími kroky:

1. Vyprázdníme seznamy *OPEN* a *CLOSED*.

2. Inicializujeme proměnnou C prvním prvkem seznamu P (tj. konfigurace startu) a vyjmeme tento prvek ze seznamu P.
3. Inicializujeme proměnnou G prvním prvek seznamu P (tj. první podcíl) a vyjmeme tento prvek ze seznamu P.
4. Přiřadíme uzlu C ohodnocení $f(C) = d(C, G) + M$ (v tomto kroku proměnná C reprezentuje kořen stromu).
5. Vložíme uzel C do seznamu *OPEN*.
6. Pokud uzel C není podobný uzlu G, pak pokračujeme krokem 10 (konfigurace jsou podobné, když Euklidovská vzdálenost mezi jejich polohami a rozdíl mezi jejich natočeními leží v dané toleranci).
7. Pokud $N_P = 0$, algoritmus ukončíme, cesta byla nalezena (hledaná cesta může být postupně určena předchůdci uzlu C a korespondujícími akcemi).
8. Určíme novou hodnotu pokuty $M = M - K$.
9. Určíme nový podcíl nastavením G = první prvek seznamu P a vyjmeme prvek z P.
10. Expandujeme uzel C, tzn. že opakujeme následující kroky pro všechny akce $u \in U$.
    a. Aplikujeme akci u na uzel C.
    b. Pokud obdržíme nekolizní konfiguraci *NC* a seznamy *OPEN* a *CLOSED* neobsahují podobnou konfiguraci, vypočteme *f*(*NC*) a vložíme tuto konfiguraci *NC* do seznamu *OPEN*.
11. Vyjmeme konfiguraci C ze seznamu *OPEN* a vložíme ji do seznamu *CLOSED*.
12. Nastavíme konfiguraci jako C = konfigurace s minimálním ohodnocením ze seznamu *OPEN* a konfiguraci vyjmeme. Pokud je seznam prázdný, algoritmus ukončíme, cesta nebyla nalezena. V opačném případě pokračujeme krokem 6.

4.2.4 Genetický algoritmus pro neholonomní robot

Problém při transformaci cesty nalezené pro holonomní robot (viz. podkapitola 4.2.3) může nastat tehdy, když není natočení robotu ve startu nebo cíli blízké natočení sousední hraně. Pokud v takovéto situaci není robot v prostředí schopen snadného otočení, je nutné, aby s natočením robotu ve startu a cíli počítal již GA. Podobný problém může nastat, když hrany cesty spolu svírají ostrý úhel anebo jsou hrany cesty příliš krátké. Pro řešení těchto problémů je navržena úprava GA z podkapitoly 4.2.2.

První změna se týká fitness funkcí (4.19) a (4.20). Nová fitness funkce pro nepřípustnou cestu je dána vztahem:

$$F(P) = f_{collisions}(P) + f_{ShortSegment}(P) + f_{ColSegment}(P) + f_{ColStart}(P) + f_{ColGoal}(P) + 1 \quad (4.24)$$

kde $f_{collisions}(P)$ je počet kolizí cesty P s překážkami, $f_{ShortSegment}(P)$ je počet hran cesty P kratších než stanovená mez d_{Min} a pro zbývající funkce platí:

$$f_{ColSegment}(P) = \sum_{i=1}^{n-1} \begin{cases} 1 & \text{pokud} \left|\beta_i - \beta_{i+1}\right| > \beta_{SegMax} \\ 0 & \text{jinak} \end{cases}, \quad (4.25)$$

$$f_{ColStart}(P) = \begin{cases} 1 & \text{pokud} \left|\beta_1 - \beta_S\right| > \beta_{StartMax} \\ 0 & \text{jinak} \end{cases}, \tag{4.26}$$

$$f_{ColGoal}(P) = \begin{cases} 1 & \text{pokud} \left|\beta_n - \beta_G\right| > \beta_{GoalMax} \\ 0 & \text{jinak} \end{cases}, \tag{4.27}$$

kde β_i je natočení *i*-té hrany cesty, β_S (β_G) je natočení robotu ve startu (cíli), β_{SegMax} je maximální povolený úhel mezi každými dvěma hranami a $\beta_{StartMax}$ ($\beta_{GoalMax}$) je maximální povolený úhel mezi natočením robotu ve startu (cíli) a sousedící hranou. Ve fitness funkci pro přípustné cesty *P* je zavedena nová složka $f_{AngleStartGoal}(P)$, která rozšiřuje funkci (4.20) takto:

$$F(P) = w_1 f_{length}(P) + w_2 f_{nodes}(P) + \frac{w_3}{2} f_{smooth}(P) + \frac{w_3}{2} f_{AngleStartGoal}(P), \tag{4.28}$$

kde $f_{length}(P)$, $f_{nodes}(P)$ a $f_{smooth}(P)$ jsou dány vztahy (4.21) a pro $f_{AngleStartGoal}(P)$ platí:

$$f_{AngleStartGoal}(P) = \frac{\left|\beta_1 - \beta_S\right| + \left|\beta_{n-1} - \beta_G\right|}{2\pi}. \tag{4.29}$$

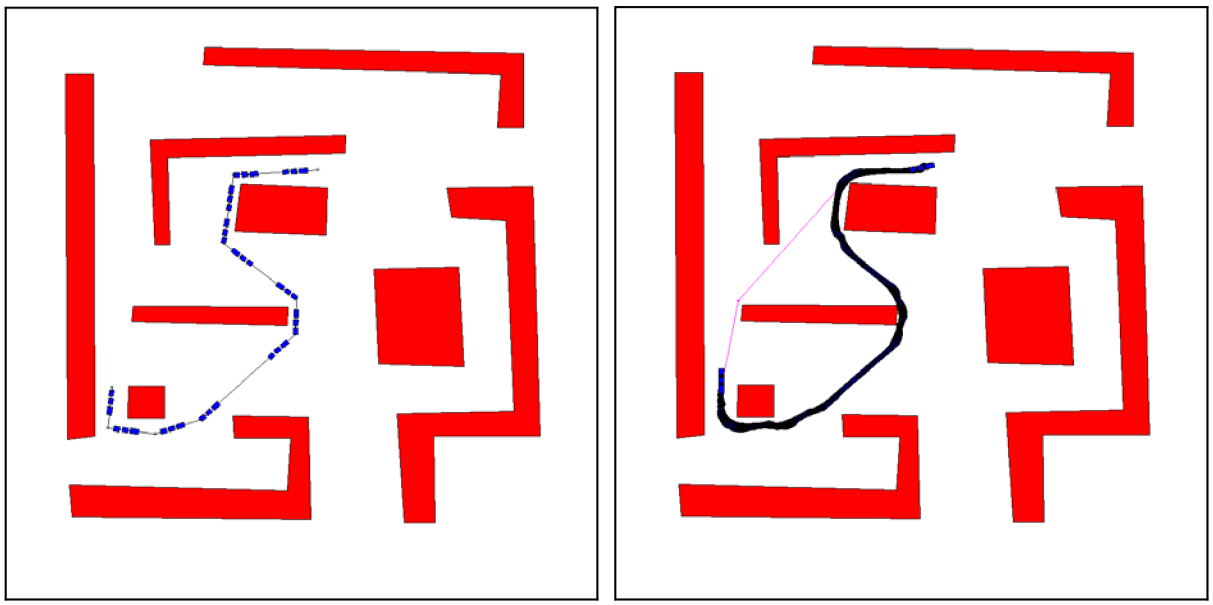
Druhá změna se týká genetických operátorů. V této úpravě již neuvažujeme operátory zkrácení a zlepšení. Naopak zavádíme následující nové operátory:

- Operátor *viditelnosti* nahrazuje operátory zkrácení a zlepšení. Je aplikován na přípustné i nepřípustné chromozomy. Tento operátor vyhlazuje cestu nalezením vhodné cesty grafem viditelnosti, jehož vrcholy jsou náhodně rozmístěny kolem všech původních bodů cesty. Mezi vrcholy grafu je vhodné také zařadit blízké uzly obchůzkových tras překážek (viz. opravný operátor v podkapitole 4.2.2). Je hledána pouze taková cesta, která splňuje požadavky na hladkost cesty i kritéria pro natočení hran sousedících se startem a cílem. Prohledávání grafu je odvozeno z upraveného Dijkstrova algoritmu navrženého v podkapitole 4.2.1.
- Operátor *úhlu startu* se snaží zlepšit natočení první hrany vůči natočení robotu ve startu náhodnou malou změnou druhého genu chromozomu. Změna genu nastane pouze pokud není výrazně zhoršena hladkost cesty vůči třetímu genu. Operátor *startu* zachovává přípustnost chromozomu.
- Operátor *úhlu cíle* se snaží zlepšit natočení poslední hrany vůči natočení robotu v cílové pozici. Princip je stejný jako u operátoru startu.

Pro nalezení posloupnosti akcí je možné použít transformaci popsanou v podkapitole 4.2.4. Protože ale transformujeme cestu, která již s neholonomními omezeními počítá (i když jen ve zjednodušené podobě), byla provedena také úprava algoritmu transformace cesty, díky níž byla snížena doba výpočtu transformace.

Nejprve je nutné dopočítat konfigurace podle nalezené cesty genetickým algoritmem. Z každé hrany kratší než $3d_{SubGoal}$, kde $d_{SubGoal}$ je vhodně zvolená konstanta, obdržíme jednu konfiguraci. Poloha robotu v této konfiguraci by měla být umístěna ve střední části hrany a natočení robotu by mělo být rovno natočení hrany. Z každé delší hrany dopočteme dvě konfigurace. Polohy robotu v těchto konfiguracích by měly být umístěny ve vzdálenosti

$d_{SubGoal}$ od koncových bodů hrany, natočení robotu je opět rovno natočení hrany (viz obr. 4.11a). Získané konfigurace spolu s konfiguracemi startu a cíle tvoří seznam konfigurací P.



(a) (b)

Obr. 4.11. *Cesta pro robot typu auto s dvěma přívěsy nalezená upraveným GA (a) a po provedené transformaci (b). Růžovou barvou je vyznačena cesta nalezená GA z podkapitoly 4.2.2.*

Cestu respektující neholonomní omezení hledáme obousměrným A* algoritmem. Tento algoritmus konstruuje dva stromy. Uzly v těchto stromech jsou reprezentovány následujícími strukturami: (konfigurace, ohodnocení konfigurace, rodičovská konfigurace, akce kterou byla konfigurace dosažena z rodičovské konfigurace). Uzly prvního stromu jsou ohodnoceny následující funkcí:

$$f_1(Uzel) = g(Start, Uzel) + d(Uzel, Cíl) \tag{4.30}$$

kde $g(Start, Uzel)$ je součet vzdáleností mezi konfiguracemi na cestě ze startu do daného uzlu, $d(Uzel, Cíl)$ je vzdálenost mezi konfigurací příslušející danému uzlu a cílovou konfigurací. Vzdálenost mezi konfiguracemi je nutno určit individuálně podle významu jednotlivých složek konfigurací. Uzly druhého stromu jsou ohodnoceny následující funkcí:

$$f_2(Uzel) = g(Cíl, Uzel) + d(Uzel, Start) \tag{4.31}$$

kde $g(Cíl, Uzel)$ je součet vzdáleností mezi konfiguracemi na cestě z cíle do daného uzlu, $d(Uzel, Start)$ je vzdálenost mezi konfigurací příslušející danému uzlu a startovní konfiguraci.

Protože algoritmus pracuje se dvěma stromy, je nutné také rozšířit počet pracovních seznamů. Seznamy *OPEN1* a *OPEN2* obsahují dosud neexpandované uzly a jsou implementovány jako haldy. Seznamy *OCCUP1* a *OCCUP2* slouží k rozpoznávání podobnosti uzlů a současně k testování setkání stromů. Pro rychlé vyhledávání v těchto seznamech jsou tyto seznamy rozděleny do několika dalších, které jsou uspořádány v mřížce a jsou tedy indexovány dvourozměrnými souřadnicemi. Při vkládání prvků do těchto seznamů se nejprve podle složek x a y konfigurace vkládaného prvku určí odpovídající buňka a tím tedy seznam pro vložení. Při hledání podobné konfigurace v těchto seznamech se neprochází seznamy všechny, ale jen seznam odpovídající konfiguraci, jejíž podobné konfigurace hledáme a seznamy s tímto seznamem sousedící.

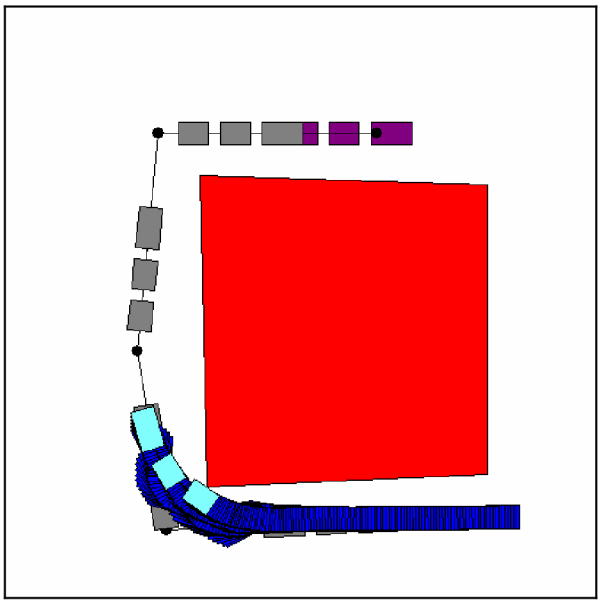
Algoritmus transformace cesty může být popsán následujícími kroky:

1. Ze seznamu P vyjmeme první prvek a vložíme ho do S.
2. Pokud S je podobné cílové konfiguraci, pak je algoritmus ukončen, cesta je nalezena.
3. Ze seznamu P vyjmeme první prvek a vložíme ho do G.
4. Vyprázdníme seznamy *OPEN1*, *OPEN2, OCCUP1* a *OCCUP2*.
5. Uzel S vložíme do seznamu *OPEN1* a do seznamu *OCCUP1*.
6. Uzel G vložíme do seznamu *OPEN2* a do seznamu *OCCUP2*.
7. Pokud je seznam *OPEN1* prázdný, algoritmus ukončíme (cesta nebyla nalezena), jinak z tohoto seznamu vyjmeme konfiguraci s minimálním ohodnocením a vložíme ji do C.
8. Pokud je natočení konfigurace S blízké konfiguraci G ($|\beta_S - \beta_G| < \beta_{Near}$, pro vhodně zvolené β_{Near}), pak do množiny U_{neigh} vložíme akci příslušnou uzlu C a z množiny U vložíme takové akce, které způsobí jen malou změnu směru oproti použití akce uzlu C. Jinak položíme $U_{neigh} = U$.
9. Expandujeme uzel C, tzn. že opakujeme následující kroky pro všechny akce $u \in U_{neigh}$:
   a. Aplikujeme akci u na uzel C.
   b. Pokud obdržíme nekolizní konfiguraci NC a seznam *OCCUP1* neobsahuje podobnou konfiguraci, vypočteme $f_1(NC)$ a vložíme NC do seznamů *OPEN1* a *OCCUP1*. Jinak pokračujeme aplikací další akce v kroku 9a.
   c. Pokud seznam *OCCUP2* obsahuje konfiguraci PC podobnou konfiguraci NC, potom zjistíme akce na cestě mezi uzly PC a G, tyto akce převedeme na inverzní a postupně je aplikujeme na uzel NC. Pokud takto obdržíme konfiguraci podobnou G, cesta mezi S a G je nalezena (hledaná cesta může být postupně určena konfiguracemi předchůdců uzlu NC a korespondujícími akcemi, zbývající akce odpovídají inverzním akcím předchůdců uzlu PC, zbývající konfigurace dopočteme aplikací těchto akcí na NC) a po vložení G do S pokračujeme krokem 2.
10. Pokud je seznam *OPEN2* prázdný, pokračujeme krokem 7, jinak z tohoto seznamu vyjmeme konfiguraci s minimálním ohodnocením a vložíme ji do proměnné C.
11. Pokud je natočení konfigurace S blízké konfiguraci G, pak do množiny U_{neigh} vložíme akci příslušnou uzlu C a z množiny U vložíme takové inverzní akce, které způsobí jen malou změnu směru oproti použití akce uzlu C. Jinak invertujeme všechny akce takto: $U_{neigh} = inv(U)$.
12. Expandujeme uzel C, tzn., že opakujeme následující kroky pro všechny akce $u \in U_{neigh}$:
    a. Aplikujeme akci u na uzel C.
    b. Pokud obdržíme nekolizní konfiguraci NC a seznam *OCCUP2* neobsahuje podobnou konfiguraci, vypočteme $f_2(NC)$ a vložíme NC do seznamů *OPEN2* a *OCCUP2*. Jinak pokračujeme aplikací další akce v kroku 12a.
    c. Pokud seznam *OCCUP2* obsahuje konfiguraci PC podobnou konfiguraci NC, potom zjistíme akce na cestě mezi uzly NC a G, tyto akce převedeme na inverzní a postupně je aplikujeme na uzel PC. Pokud takto obdržíme konfiguraci podobnou G, cesta mezi S a G je nalezena (hledaná cesta může být postupně určena konfiguracemi předchůdců uzlu PC a korespondujícími akcemi, zbývající akce odpovídají inverzním akcím

předchůdců uzlu *NC*, zbývající konfigurace dopočteme aplikací těchto akcí na *PC*) a po vložení *G* do *S* pokračujeme krokem 2.

13. Pokračujeme krokem 7.

V algoritmu může nastat stav, kdy není možné z nového startu dále rozšiřovat nový strom z důvodu zablokování robotu blízkou překážkou (obr. 4.12). Některá z blízce předcházejících konfigurací byla podobná cíli minulé dvojice stromů, kde však ještě nebylo jasné, že se jedná o konfiguraci v nevhodné větvi stromu. Tato situace nastává s malou pravděpodobností a to v případech nastavení krátké obchůzkové trasy (viz opravný operátor v podkapitole 4.2.2) anebo při nastavení malé podobnosti podcílů. Protože není možné zablokovanou konfiguraci dále expandovat, tento problém se projeví předčasným vyprázdněním seznamu *OPEN1*.



Obr. 4.12. *Zablokování konfigurace v průběhu transformace*

Problém je možné řešit tak, že v případě prázdného seznamu *OPEN1* vložíme v kroku 7 do proměnné *S* konfiguraci, která je v dosud nalezené cestě umístěna *n* konfigurací před konfigurací zablokovanou a pokračujeme krokem 4. Toto v kroku 7 opakujeme vždy, když je seznam *OPEN1* prázdný. Experimenty (pro robot na obr. 4.12) však ukazují, že není potřeba více než pět opakování (pro $n = 5$).

4.2.5 Optimalizace parametrů

Jedním z cílů práce bylo ověření možnosti nastavování parametrů navržené metody další metodou strojového učení. Z tohoto důvodu byl navržen a otestován genetický algoritmus (GA1), jehož výsledkem jsou hodnoty parametrů genetických algoritmů (GA2) navržených v kapitolách 4.2.2 a 4.2.4 pro konkrétní mapu a pro dané váhy fitness funkcí.

Chromozom má velikost deset genů, přičemž první až devátý gen odpovídá postupně pravděpodobnostem operátorů křížení, velké mutace, malé mutace, opravy, výměny, vyhlazení, zkrácení (úhlu startu a cíle), odstranění a zlepšení (viditelnosti). Desátý gen odpovídá počtu párů chromozomů při selekci. První až devátý gen může nabývat reálných hodnot z intervalu $\langle 0; 1 \rangle$, hodnota desátého genu je celočíselná a nesmí být větší než polovina

z počtu chromozomů v populaci optimalizovaného GA2. Počáteční populace je vytvořena zcela náhodně.

Pro každý chromozom GA1 je spuštěn GA2 pro danou testovací množinu startů a cílů. Optimalizujeme-li kvalitu řešení hledaného pomocí GA2, potom je fitness hodnota chromozomu GA1 rovna průměrné hodnotě fitness nejlepších nalezených cest. Pokud optimalizujeme dobu výpočtu GA2, potom je fitness hodnota chromozomu GA1 rovna průměrné době výpočtu nejlepších nalezených cest pomocí GA2. Je zřejmé, že je pomocí vhodných normalizovaných vah možné současně optimalizovat kvalitu hledané cesty i dobu výpočtu.

GA1 používá binární turnajovou selekci a jednobodové křížení. Operátor mutace provádí náhodné vygenerování nové hodnoty náhodně vybraného genu v chromozomu. Pomocí křížení a mutace je vygenerována nová populace vždy o stejném počtu chromozomů jako populace stará. Ukončení výpočtu GA1 je provedeno tehdy, když již dochází pouze k malým změnám průměrné (nebo nejlepší) hodnoty fitness v populaci.

4.2.6 Kombinace případového grafu a GA

V této podkapitole je navržen systém případového usuzování, v jehož případové bázi jsou udržovány cesty pro neholonomní mobilní robot. Jako pomocná metoda případového usuzování je použit genetický algoritmus, představený v podkapitole 4.2.4. Podobně jako v podkapitole 4.1.4, případová báze je implementována jako případový graf. Případový graf u této metody má ovšem odlišnou strukturu, neboť musí uvažovat neholonomní omezení.

Cesta je po absolvování robotem do báze případů vložena jako několik samostatných lineárních segmentů (případů). Každý případ je reprezentován dvojicí opačně orientovaných hran případového grafu $G = (V, E)$, kde V je množina uzlů, $V = \{0, 1, 2, \ldots, n\}$ a E je množina orientovaných hran $E \subseteq \{(i, j) \mid i, j \in V\}$. K vložení nového případu dojde pouze tehdy, pokud báze případů neobsahuje již případ podobný vkládanému případu. Případ, jehož jedna z dvojice hran je (u, v), je podobný případu, jehož jedna z hran je (i, j), pokud $MB((u,v),(i,j)) < d_{proximity}$, kde $d_{proximity}$ je maximální hodnota míry blízkosti a $MB((u,v),(i,j))$ je míra blízkosti daná vztahem

$$MB\big((u,v),(i,j)\big) = \min\begin{Bmatrix} \max\{d(i,u),\ \min\{vz(j,u,v),\ vz(v,i,j)\}\}, \\ \max\{d(i,v),\ \min\{vz(j,u,v),\ vz(u,i,j)\}\}, \\ \max\{d(j,u),\ \min\{vz(i,u,v),\ vz(v,i,j)\}\}, \\ \max\{d(j,v),\ \min\{vz(i,u,v),\ vz(u,i,j)\}\} \end{Bmatrix}, \tag{4.32}$$

kde $d(i, j)$ je euklidovská vzdálenost uzlů i a j a $vz(v,i,j)$ je vzdálenost bodu v od hrany (i, j). Pomocí převrácené hodnoty míry blízkosti je možné odvodit míru podobnosti.

Hrany grafu mohou mít společné své krajní body. Pokud přidávaný případ protíná již existující hranu grafu, pak jsou cesta i hrana rozděleny do menších případů pouze tehdy, pokud rozdělením nevznikne případ o délce kratší, než je minimální stanovená délka. Je tedy povoleno křížení hran, aniž by musel být v jejich průsečíku uzel.

Případ (dvojice opačně orientovaných hran) je v bázi případů uložen se strukturou $(t_{Sum}, t_{SumW}, n_1, n_2, n_3)$, kde $t_{Sum} = \sum t$ je součet normálních dob průchodu případem, $t_{SumW} = \sum t_W$ je součet dob čekání t_W při průchodu případem, n_1 je celkový počet průchodů

případem, n_2 je počet průchodů případem s čekáním $t_W \le t_{MaxW}$ a n_3 je počet případů, kdy byla použita jiná cesta ($t_W > t_{MaxW}$). Ohodnocení případu t_E je dáno následujícím vztahem:

$$t_E = \frac{t_{Sum}}{n} + \frac{n_2}{n}\left(\frac{t_{Sum}}{n_1} + \frac{t_{SumW}}{n_2}\right) + \frac{n_3}{n}\left(t_{MaxW} + M\right), \tag{4.33}$$

kde $n = \sum_{i=1}^{3} n_i$, t_{MaxW} je maximální doba čekání a M je vhodně volená penalizace neprůchodné hrany (např. nejdelší cesta ze startu do cíle se započtením čekacích dob).

V důsledku kinematických omezení robotu nelze pro prohledávání grafu použít běžných metod prohledávání. Dále popsaný algoritmus hledání cesty rozšiřuje algoritmus, který je navržený v podkapitole 4.2.1 o fiktivní uzel g', díky kterému můžeme nyní uvažovat i cílové natočení robotu.

Symbolem $d_s(j, k)$ je označena doba průchodu z uzlu s do uzlu k přes hranu (j, k) v grafu $G = (V, E)$. Symbol $p_s(i, j)$ označuje počáteční uzel hrany bezprostředně předcházející hraně (i, j) na nejkratší cestě z uzlu s do uzlu j přes hranu (i, j). Symbol $\beta(i, j)$ odpovídá natočení hrany (i, j).

Rozšířený algoritmus hledání přípustné cesty z uzlu s do uzlu g lze popsat následujícími kroky:

1. Do V dočasně přidáme fiktivní uzly s' a g' tak, aby se natočení fiktivních hran (s', s) a (g, g') rovnalo natočení robotu ve startu $\beta(s', s) = \beta_S$ a cíli $\beta(g, g') = \beta_G$.
2. Do V dočasně přidáme m uzlů náhodně vygenerovaných v okolí startu i cíle a nekolidujících s překážkami..
3. Do E dočasně přidáme takové nekolizní hrany (i, j), které jsou incidentní alespoň s jedním dočasným uzlem a současně splňují podmínku délky $d(i, j) \ge d_{Min} \wedge d(i, j) \le d_{Max}$.
4. Do E dočasně přidáme nekolizní hrany (i, j) spojující ostatní uzly takové, které nekříží žádnou z již existujících hran v E a současně splňují podmínku délky $d(i, j) \ge d_{Min} \wedge d(i, j) \le d_{Max}$.
5. Pro hranu (s', s) položme ohodnocení $d_s(s', s) = 0$ a pro ostatní hrany položme $d_s(i, j) = \infty$.
6. Do haldy H vložíme všechny hrany z E.
7. Z haldy H vyjmeme hranu s minimálním ohodnocením a označíme jako (k, r).
8. Pokud $(k, r) = (g, g')$, algoritmus ukončíme, cesta je nalezena.
9. Pokud je ohodnocení $d_s(k, r) = \infty$, pak algoritmus ukončíme, cestu grafem G není možné vyhledat.
10. Prozkoumáme všechny hrany $(r, i) \in E$, pro něž platí $d(r, i) \ge d_{Min} \wedge d(r, i) \le d_{Max}$. Pokud $(k, r) = (s', s)$, pak musí též platit $|\beta(k, r) - \beta(r, i)| \le \beta_{StartMax}$, pokud $(r, i) = (g, g')$, pak musí také platit $|\beta(k, r) - \beta(r, i)| \le \beta_{GoalMax}$, jinak musí platit $|\beta(k, r) - \beta(r, i)| \le \beta_{SegMax}$. Když hrana (r, i) splňuje tyto podmínky a $d_s(r, i) > d_s(k, r) + t_E$, pak položíme $d_s(r, i) = d_s(k, r) + t_E$ a $p_s(r, i) = k$. Ohodnocení t_E je vypočteno podle vztahu (4.33) pro hranu (r, i). Pokud je (r, i) dočasná hrana, pak je t_E odhad doby průchodu touto hranou.

11. Návrat na krok 7.

Cesta robotu z uzlu s do uzlu g bude určena jako posloupnost hran uložených v zásobníku Z, přičemž počáteční hrana se bude nacházet na vrcholu zásobníku. Určení nejkratší cesty z s do g lze popsat takto:

1. Položíme $Z = \emptyset$. Položíme $(k, r) = (g, g')$.
2. Do zásobníku Z vložíme hranu (k, r).
3. Pokud je $(k, r) = (s', s)$ pak ze zásobníku vyjmeme fiktivní hrany (hrany na vrcholu a dně zásobníku), z grafu G odstraníme dočasné hrany i uzly a algoritmus ukončíme, cesta je uložena v zásobníku Z.
4. Položme $(k, r) = (p_s(k, r), k)$ a pokračujeme krokem 2.

Pokud v případovém grafu G nebyla cesta nalezena nebo selhala-li následná transformace této cesty, pak cestu hledáme pomocí GA. Cestu nalezenou GA se pokusíme upravit podle případového grafu tak, že pokud leží některý z vnitřních uzlů nalezené cesty v okolí uzlů množiny V, pak tento vnitřní uzel cesty nahradíme nejbližším uzlem z množiny V. Tímto dosáhneme toho, že po absolvování cesty robotem bude v případovém grafu aktualizována již existující hrana, případně bude vytvořena nová hrana mezi existujícími uzly. Pokud touto úpravou vznikne nepřípustná cesta nebo selže-li transformace cesty, je nutné použít cestu nalezenou GA. V některých scénách může být vhodné provést úpravu cesty nalezené GA, i když cesta grafem existuje. Tímto můžeme dosáhnout prozkoumání hran s horším ohodnocením za účelem zjištění, zda je horší ohodnocení ještě stále aktuální.

Po přidání cesty nebo po aktualizaci její struktury $(t_{Sum}, t_{SumW}, n_1, n_2, n_3)$ je vhodné odstranit staré a málo používané případy. Tímto dosáhneme možného přidání nových hran, které mohly být těmito starými málo používanými avšak podobnými případy blokovány. Po dosažení maximální velikosti případové báze je možné spustit odstranění starých případů samostatně. Tímto sice můžeme přijít na nějakou dobu o kvalitní případy, ale současně se naskýtá možnost najít v prostředí s dynamickými změnami případy lepší.

5. Implementace

Pro ověření funkce a experimentování s navrženými metodami bylo v průběhu vývoje vytvořeno několik simulačních programů. K tvorbě programů bylo použito vývojové prostředí Borland Delphi 7.0. Metody pracující na mřížce (viz podkapitola 4.1) jsou implementovány v programu *pCase.exe*. Metoda kombinující případový graf a pravděpodobnostní stromy (viz podkapitola 4.2.1) je realizována v programu *case_continue.exe*. Zkušenosti získané při tvorbě těchto programů byly uplatněny při tvorbě nové navazující aplikace *Robot2010.exe*. V této aplikaci jsou implementovány metody z podkapitol 4.2.2 až 4.2.6. Protože tyto metody lze považovat za nejvíce přínosné, je tato kapitola věnována právě této finální aplikaci.

Navržené metody jsou implementovány tak, aby nebyly na simulační aplikaci *Robot2010.exe* závislé a byly jednoduše použitelné i mimo tuto aplikaci. Pro snadné pochopení programového rozhraní jsou v podkapitole 5.1 popsány důležité třídy definované v jednotlivých jednotkách spolu s jejich podstatnými veřejnými metodami. Popis ovládání simulační aplikace je uveden v podkapitole 5.2. Aplikace zkompilovaná do prostředí win32 je i s kompletním zdrojovým kódem k dispozici na přiloženém CD-ROM.

Pokud budou čtenáře této disertační práce zajímat implementační detaily programů *pCase.exe* nebo *case_continue.exe*, je možné toto nastudovat ze zdrojových kódů, které lze najít také na přiloženém CD-ROM.

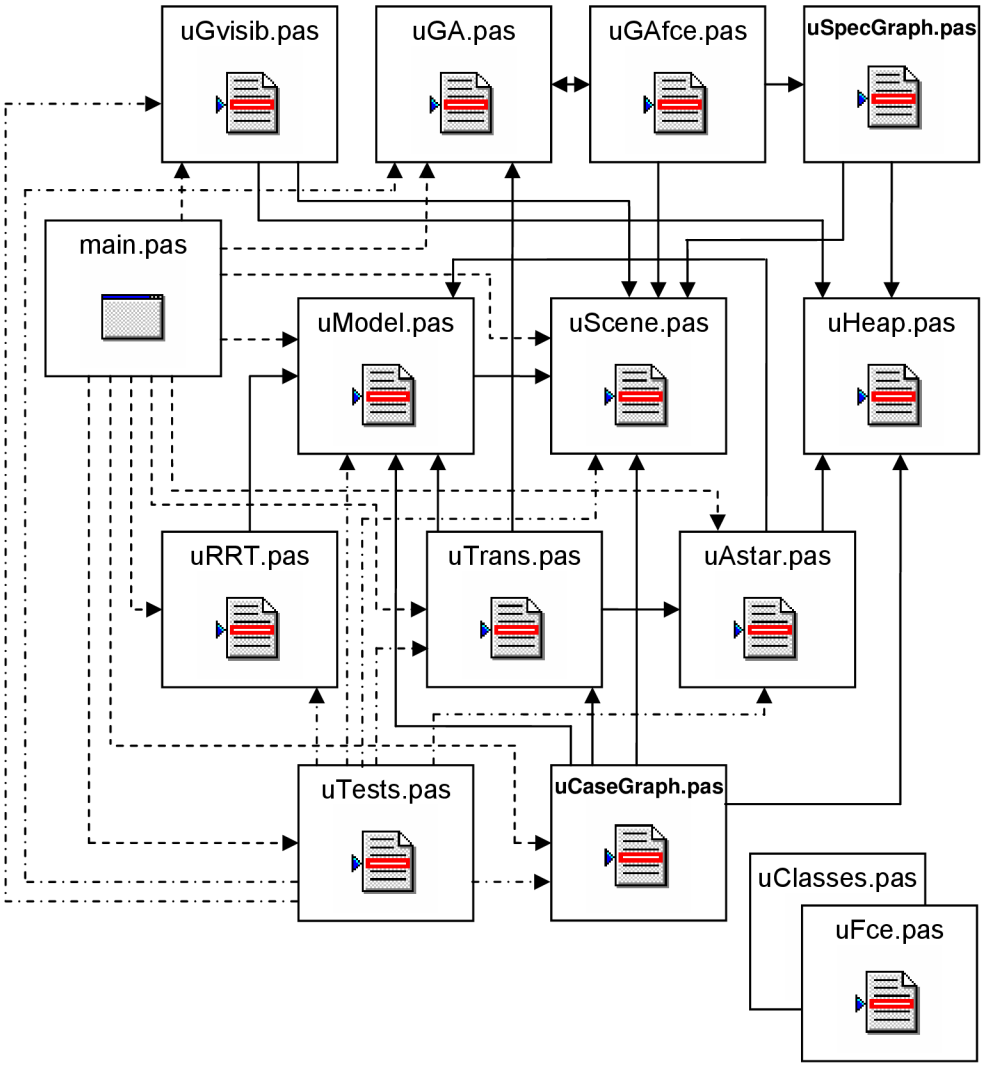
5.1 Popis jednotek

Struktura jednotek aplikace *Robot2010.exe* je znázorněna na obrázku 5.1. Plnými šipkami jsou znázorněny závislosti volání funkcí mezi jednotkami. Přerušovanými šipkami jsou zakresleny volání funkcí z jednotky *main.pas* svázané s formulářem simulační aplikace. Tyto šipky představují možná volání funkcí z uživatelského rozhraní. Čerchované šipky znázorňují volání funkcí z vestavěných testovacích podprogramů simulační aplikace. Na obrázku 5.1 nejsou zachyceny závislosti na jednotkách *uClasses.pas* a *uFce.pas*, které obsahují základní definice tříd a obecné funkce používané téměř všemi jednotkami. Při četbě následujících odstavců této podkapitoly je doporučeno nahlížet do zdrojového kódu popisované jednotky.

V jednotce *uClasses.pas* je definována třída *TCesta*. Objekt této třídy používají všechny implementované metody hledání jako výslednou nalezenou cestu. Nenalezne-li některá z metod cestu, pak namísto objektu vrací hodnotu *nil*. *TCesta* obsahuje seznam *Segmenty*, jehož prvky ukazují na části cesty definované třídou *TSegment*. Každá instance třídy *TSegment* obsahuje souřadnice *s* a *g*, které charakterizují začátek a konec příslušné části cesty. V případě holonomního robotu segmenty odpovídají lineárním úsekům cesty. Pomocí třídy *TCesta* lze však reprezentovat i cestu pro neholonomní robot. Pro tyto účely třída *TCesta* obsahuje ukazatel na počáteční konfiguraci cesty *sq* a třída *TSegment* posloupnost akcí v seznamu *Akce*. Koncovou konfiguraci cesty dostaneme postupnou aplikací všech akcí všech segmentů na *sq*. *TSegment* dále obsahuje dvě vlastnosti *t* a *tw*. Tyto dvě proměnné každého segmentu jsou plněny skutečnými dobami průchodu při průchodu robotu cestou. Cesta s takto aktualizovanými segmenty je poté zpět předávána systému případového usuzování.

Jednotka *uScene.pas* obsahuje definici třídy *TScene*. Objekt této třídy reprezentuje mapu prostředí, nad kterou probíhá veškeré plánování cesty. Velikost prostředí je dána reálnými vlastnostmi *MaxX* a *MaxY*. Ukazatele na překážky jsou uloženy v seznamu *Obstacles*. Překážka je definovaná třídou *TObs*. Jedná se o polygonální překážku, kde maximální počet vrcholů je roven 100. Pro přidání překážky do mapy prostředí slouží metoda *PridejPrekazku*, jejíž parametry tvoří pole vrcholů překážky a počet vrcholů v poli. Ve třídě *TScene* jsou také

definovány metody pro detekci kolize. Kolizní bod je možné detekovat metodou *BodVKolizi* a kolizní úsečku lze detekovat metodou *UseckaVKolizi*. Pro detekci kolize se zvětšenými překážkami slouží metody *BodVKolizi2* a *UseckaVKolizi2*. Zvětšení překážek zajistí metoda *SpoctiObchuzky*, jejíž parametr představuje vzdálenost mezi příslušnými vrcholy skutečných a zvětšených překážek.



Obr. 5.1. *Struktura jednotek simulační aplikace*

Jednotka *uGvisib.pas* obsahuje implementaci metody hledání cesty holonomního robotu pomocí grafu viditelnosti. Pro vytvoření grafu podle zvětšených překážek použijeme metodu *Inic* ve třídě *TGrafViditelnosti*. Prohledávání grafu je možné spustit metodami *Hledej* nebo *HledejAStar*. Parametry těchto metod jsou startovní a cílová konfigurace, návratovou hodnotou je v obou případech objekt typu *TCesta*. Metoda *Hledej* prohledává graf viditelnosti pomocí Dijkstrova algoritmu, metoda *HledejAStar* pomocí algoritmu A*. Pro vyhledávání prvku s minimálním ohodnocením využívají obě metody haldy, která je implementována v jednotce *uHeap.pas*.

Třídy *TDiff*, *TAuto* a *TTahac* reprezentující modely neholonomních robotů popsaných v podkapitole 6.6 jsou zavedeny v jednotce *uModel.pas*. Všechny uvedené třídy jsou potomky abstraktní třídy *TModel*, která definuje veřejné metody každého modelu. Pokud by byl do aplikace přidáván další model, je nutné, aby byl také potomkem třídy *TModel*, neboť všechny implementované metody plánování cesty neholonomního robotu pracují s modelem výhradně přes metody definované v této abstraktní třídě. Každá třída modelu pracuje také s vlastní třídou konfigurace, která je definovaná jako potomek třídy *TKonfigurace* pro holonomní robot. Je to z toho důvodu, aby mohly být metody plánování cesty navrženy jen pro jednu obecnou třídu konfigurace. Každému modelu odpovídá také vlastní třída akce, která je ovšem definovaná jako záznam. Metody plánování cesty pracují s akcemi pouze jako s ukazateli, obsahu akcí rozumí pouze příslušný model robotu. Akce modelu jsou uloženy v seznamu *Akce*, inverzní akce pak v seznamu *InvAkce*.

V tomto odstavci jsou popsány metody třídy *TModel*. Pomocí funkce *Kolize* detekujeme kolizní konfiguraci, která tvoří spolu s ukazatelem na scénu parametry této funkce. Funkce *NovaKonfiguraceDialog* vytváří nový objekt třídy konfigurace, jejíž složky odpovídající souřadnici referenčního bodu jsou předávány jako parametr funkce a zbývající složky jsou zadávány uživatelem přes dialogové okno. Hodnota funkce *NovaKonfiguracePodleAkce* odpovídá také nové konfiguraci, získané aplikací akce *a* na konfiguraci *c* po dobu *t*, kde *c*, *a*, *t* jsou parametry funkce. Dalšími parametry jsou ukazatel na scénu a odkaz na proměnnou, do které je zaznamenána případná informace o kolizi vzniklé v průběhu výpočtu. Pro výpočet nové konfigurace je použita Eulerova metoda integrace o integračním kroku 0,5 časových jednotek. Funkce *Rand_Conf* vrací náhodně vygenerovanou konfiguraci ve scéně. Funkce *Rand_Conf_Okno* vrací náhodně vygenerovanou konfiguraci ve scéně do vzdálenosti *ar* od konfigurace *q*, kde *ar* a *q* jsou parametry funkce. Vzdálenost mezi konfiguracemi vrací funkce *Distance*. Funkce *VedlejsiAkce* vrací seznam takových akcí, které způsobí jen malou změnu směru oproti akci, která je předávaná jako parametr. Výpočet odlišnosti akcí zajišťuje funkce *OdlisnostAkci*. Funkce *dAkce* vypočte podle akce předané parametrem výslednou rychlost robotu při použití této akce. Podobnost konfigurací je zjišťována pomocí funkce *Podobne*. Natočení robotu v určité konfiguraci vrací funkce *UdejSmer*. Pro transformaci cesty je důležitá metoda *NovaKonfiguraceUsecka*, která dopočítává konfigurace na dané úsečce.

Jednotka *uRRT.pas* obsahuje třídu *TMetodaRRT*. Voláním funkce *BuildRRT* vytvoříme stromový graf metody RRT. Funkce *BuildRRT* vyžaduje předání několika parametrů. Parametry *qInit* a *qGoal* představují startovní a cílovou konfiguraci. Parametry *aImage* a *Meritko* je zapotřebí nastavit na platné hodnoty jen pokud si přejeme, aby bylo generování stromu v průběhu výpočtu zobrazováno. Hodnota parametru *aT* odpovídá době působení akce. Parametr *aD* představuje maximální povolenou vzdálenost mezi referenčními body konfigurace nalezeného cíle a konfigurací *qGoal*, parametr *aU* představuje maximální rozdíl natočení konfigurace nalezeného cíle a konfigurace *qGoal*. Parametrem *aRandConf* stanovíme variantu RRT pro provedení výpočtu. Pro RRT-B nastavíme tento parametr na znak 'B', pro RRT-Z volíme 'Z' a RRT-R odpovídá znak 'R'. Posledním parametrem je maximální doba výpočtu. Návratová hodnota funkce *BuildRRT* informuje o úspěšném vytvoření stromu. Pokud je tedy strom úspěšně vytvořen, nalezenou cestu získáme pomocí funkce *FindPath*, jejíž návratová hodnota je typu *TCesta*.

V jednotce *uAstar.pas* je definována třída *TMetodaA*, pomocí které generujeme strom systematickým inkrementálním vzorkováním. Hledanou cestu získáme jako objekt třídy *TCesta*, který je návratovou hodnotou funkce *SpustVypocet*. Dále jsou popsány parametry této funkce. Parametry *aStart* a *aCil* představují startovní a cílovou konfiguraci. Parametrem *aMetoda* volíme pořadí expandování uzlů. Znak 'A' odpovídá algoritmu A*, znak 'B' odpovídá algoritmu B*, znak 'D' odpovídá Dijkstrovu algoritmu a hodnota 'F' odpovídá

prostému prohledávání do šířky. Parametry *rdObsaz*, *ruObsaz*, *rdCil*, *ruCil*, *rdSpoj* a *ruSpoj* představují maximální hodnoty vzdáleností referenčních bodů konfigurací a maximální rozdíly natočení konfigurací uvažovaných postupně při přidávání prvků do seznamů *OCCUP1* a *OCCUP2*, při nalezení cíle a při spojení stromů. Parametrem *aZavisleAkce* redukujeme seznam akcí pro expandování uzlu jen na akce v seznamu *VedlejsiAkce* odpovídajícího modelu. Parametr *aAkceTime* odpovídá době působení akce. Nastavení parametru *aVlakna* způsobuje výběr vícevláknové implementace metody. Parametr *aPresnyCil* určuje důležitost podobnosti cíle (viz vztah 6.24). Parametr *aMaxT* stanovuje maximální dobu výpočtu.

V jednotce *uGA.pas* jsou definovány třídy *TGen*, *TChromozom*, *TPopulace*, *TNastaveniGA* a *TMetodaGA*, které jsou potřebné pro výpočet cesty pomocí genetického algoritmu. Spouštění plánování cesty se provádí pomocí objektu třídy *TMetodaGA*, který je správcem instancí ostatních zmíněných objektových tříd. Pomocné funkce genetického algoritmu jsou definovány v jednotkách *uGAfce.pas* a *uSpecGraph.pas*. Konstruktor objektu třídy *TMetodaGA* vyžaduje zadat mimo ukazatele na scénu i ukazatel na záznam třídy *TNastaveniGA*, který má v sobě nastaveny pravděpodobnosti všech operátorů, počet chromozomů v populaci, počet chromozomů pro selekci, rozsah počtu genů v počátečním chromozomu a váhy složek fitness funkce. Ihned po vytvoření objektu třídy *TMetodaGA* je možné volat pro nalezení cesty holonomního robotu metodu *FindPath*, jejíž parametry představují souřadnice startu a cíle, volbu *aNacti* pro načtení počáteční populace ze souboru a hodnotu maximální doby výpočtu. Pro hledání cesty neholonomního robotu je třeba před voláním *FindPath* nastavit globální logickou proměnnou *AlfaKont* na *true* a také nastavit reálné globální proměnné *AlfaStart*, *AlfaCil*, *uSt*, *uCil*, *uMezi* a *dMin*, které představují postupně natočení robotu ve startu a cíly, maximální povolené úhly $\beta_{StartMax}$, $\beta_{GoalMax}$, β_{SegMax} a minimální povolenou délku hrany. Návratovou hodnotou funkce *FindPath* je objekt třídy *TCesta* reprezentující nalezenou cestu. Po dokončení výpočtu je také ve vlastnosti *Best* uložen ukazatel na nejlepší získaný chromozom v konečné populaci. K celé konečné populaci je možné také přistupovat přes objekt s ukazatelem uloženým ve vlastnosti *aPop*, jehož metodou *SaveToFile* je možné uložit celou populaci do souboru.

Jednotka *uTrans.pas* obsahuje definici třídy *Ttrans*. Funkce této třídy *HledejCestu* má pouze dva parametry *aStart* a *aCil* představující startovní a cílovou konfiguraci. Návratovou hodnotou této funkce je objekt třídy *TCesta*, který reprezentuje nalezenou cestu. Při vytváření objektu třídy *Ttrans* je nutné prostřednictvím konstruktoru předat záznam typu *TNastaveniTrans*, který obsahuje všechny potřebné hodnoty parametrů metody navržené v podkapitole 4.2.4. Jedná se o parametry genetického algoritmu popsané v předcházejícím odstavci a o parametry následné transformace *rdMax*, *rVzd* a *Paralel*. Parametr *rdMax* označuje maximální povolenou délku segmentu cesty před transformací. Parametrem *rVzd* nastavujeme vzdálenost podcílů od konců segmentu. Výpočet pomocí vícevláknové implementace závisí na nastavení parametru *Paralel*. V záznamu třídy *TNastaveniTrans* je také nutné uvést ukazatele objektů scény a modelu robotu.

Systém případového usuzování je implementován v jednotce *uCaseGraph.pas*. Hlavní třídou je třída *TCaseGraph*. Tato třída je potomkem třídy *Ttrans*, neboť rozšiřuje tuto třídu o případový graf, přičemž použití genetického algoritmu i transformace zůstává nadále potřebné. Při vytváření objektu třídy *TCaseGraph* je zapotřebí prostřednictvím konstruktoru předat záznam typu *TNastaveniCaseGraph*, který se skládá ze záznamu typu *TNastaveniTrans* a parametrů *dPodobne*, *dOkoli* a *im*. Hodnota *dPodobne* představuje maximální hodnotu blízkosti podobných případů, hodnota *dOkoli* odpovídá velikosti okolí pro napojení na případový graf a *im* je počet generovaných bodů v okolí. Hledanou cestu získáme pomocí metody *HledejCestu* stejně jako u třídy *Ttrans*. Přidání cesty (nebo aktualizace ohodnocení již

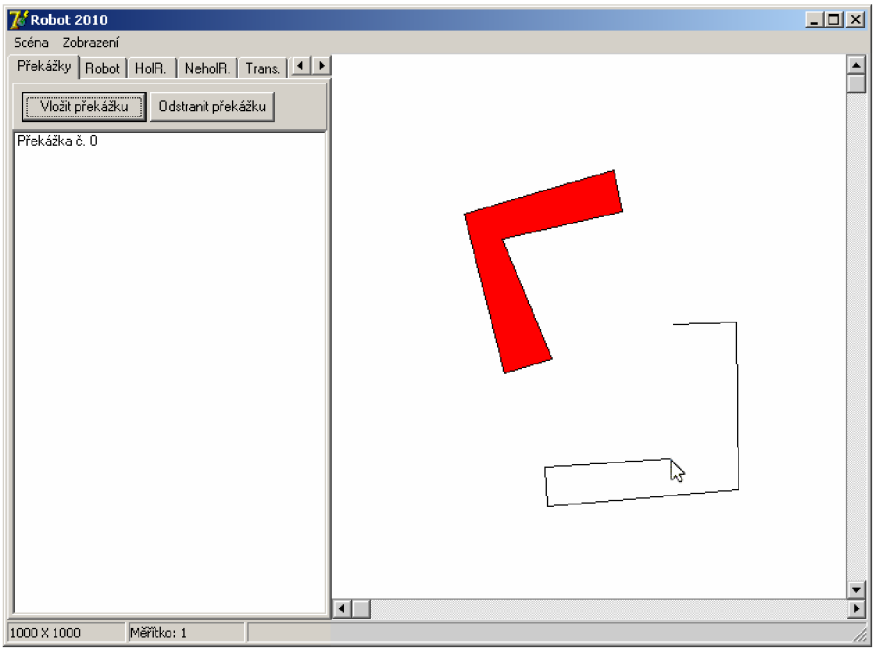
existující cesty) provedeme pomocí funkce *PridejCestu*. V této implementaci je předpokládáno, že $t_{MaxW} = \infty$. Voláním procedury *OdstranStare* dojde k odstranění případů starších než *aStari*, pro které platí $n_1 \leq an$ a $n_2 \leq an2$, kde *aStari*, *an* a *an2* jsou parametry této procedury.

V jednotce *uTests.pas* jsou definovány procedury několika testů. Všechny testy pracují s testovací množinou konfigurací, která je implementována pomocí třídy *TestMnozina*. Typy implementovaných testů jsou vysvětleny v následující podkapitole.

5.2 Ovládání programu

Uživatelské rozhraní simulační aplikace *Robot2010.exe* je zobrazeno na obr. 5.2. Po spuštění aplikace je vytvořena nová scéna o velikosti 1000×1000 jednotek. V levé části na záložce *Překážky* je možné přidávat a odstraňovat překážky ve scéně. Pro přidání překážky je nutné stisknout tlačítko *Nová překážka*, poté klikáním ve scéně vložit postupně vrcholy překážky a na závěr stisknout tlačítko *Vložit překážku*. Odstranění překážky ze scény se provede výběrem překážky v seznamu (vybraná překážka je zobrazena modrou barvou) a následným stiskem tlačítka *Odstranit překážku*.

V menu *Scéna* je možné uložit vytvořenou scénu, vytvořit scénu novou nebo načíst scénu uloženou. V menu *Zobrazení* je možné měnit měřítko zobrazení scény pomocí voleb *Zvětšit* + či *Zmenšit* –, smazat všechny vykreslené cesty a ponechat jen překážky pomocí volby *Obnovit*, zkopírovat scénu do obrázku v souboru *_snapshot.bmp* volbou *Vytvoř .bmp* nebo zvolit, že při vykreslování překážek má být vykresleno i jejich zvětšení volbou *Vykreslovat zvětšené překážky*.

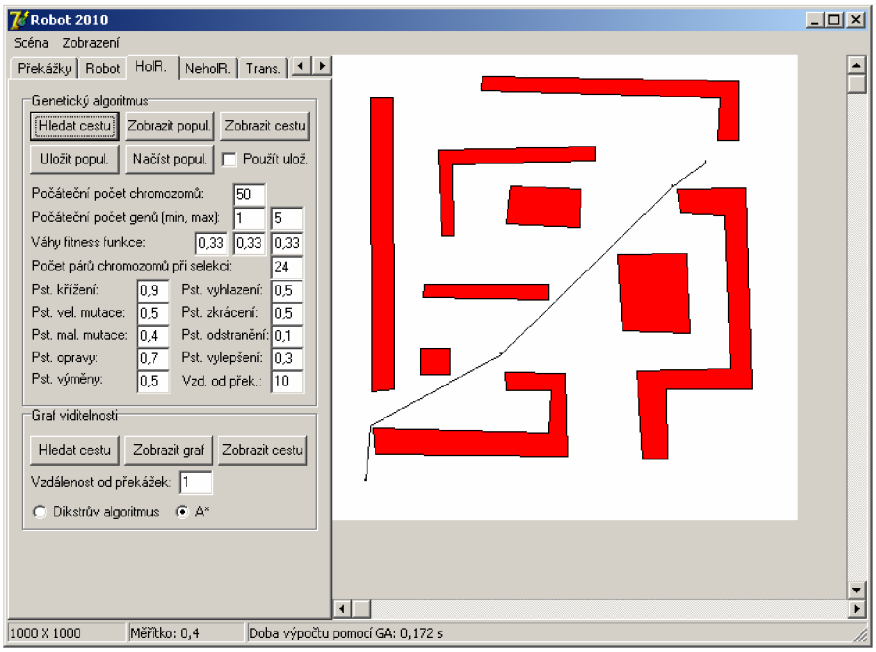


Obr. 5.2. *Záložka simulační aplikace „Překážky" a scéna ve stavu přidávání překážky*

Na záložce *Robot* (obr. 5.3) je možné pomocí tlačítek *Nový start* a *Nový cíl* nastavit startovní a cílovou konfiguraci holonomního robotu. Umístění holonomního robotu odpovídá souřadnici referenčního bodu neholonomního robotu. Ostatní složky konfigurace zadává uživatel až při spuštění plánování dle vybraného typu neholonomního robotu. Výběr typu neholonomního robotu se provádí taktéž na této záložce.

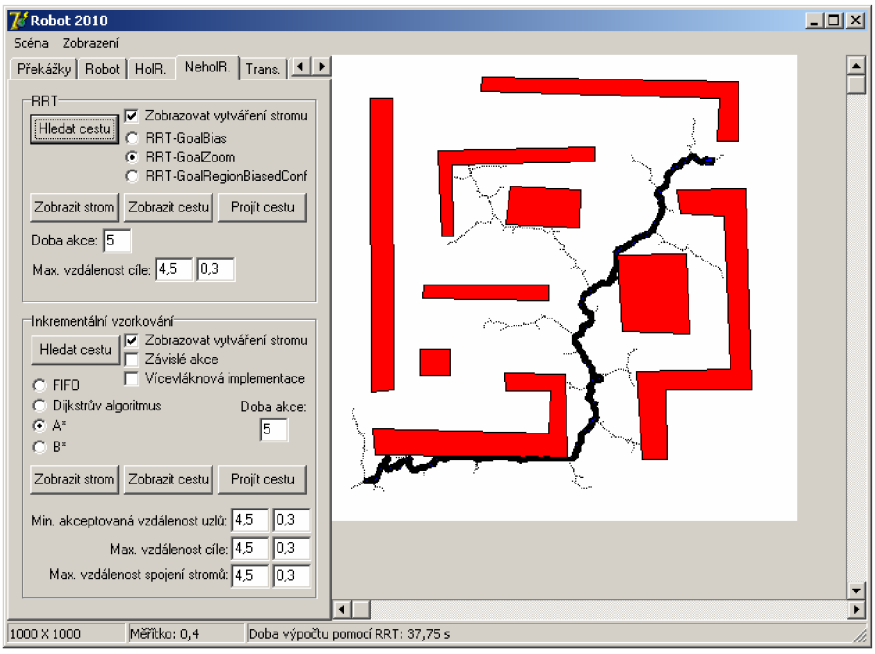
Obr. 5.3. *Záložka simulační aplikace „Robot“*

Metodu pro plánování cesty holonomního robotu navrženou v podkapitole 4.2.2 lze spouštět ze záložky *HolR*. Nastavení parametrů metody je zřejmé z obrázku 5.4. Stiskem tlačítka *Hledat cestu* zahájíme výpočet. Po dokončení výpočtu je nalezená cesta zobrazena. Poslední nalezenou cestu lze také vykreslit stiskem tlačítka *Zobrazit cestu*. Tlačítko *Zobrazit popul.* vykreslí do scény chromozomy poslední populace genetického algoritmu. Poslední „naučenou“ populaci genetického algoritmu lze uložit a opětovně načíst ze souboru *GA.pop* pomocí tlačítek *Uložit popul.* a *Načíst popul.* Má-li být uložená populace použita při novém hledání jako populace počáteční, musí být zaškrtnuta volba *Použít ulož.* Na této záložce je také možné spustit nalezení cesty pomocí grafu viditelnosti. Tlačítkem *Zobrazit graf* dojde k vykreslení tohoto grafu.



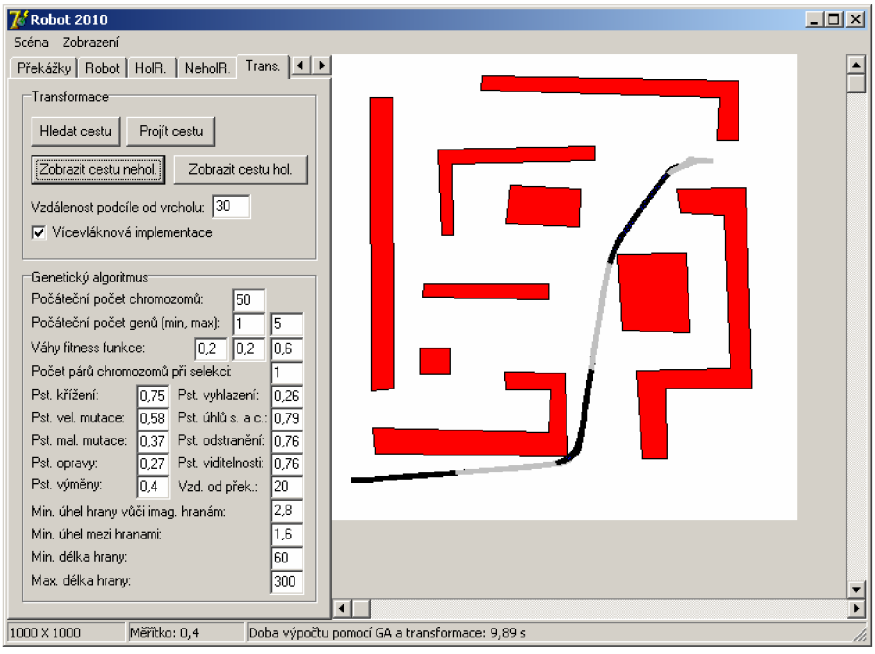
Obr. 5.4. *Záložka simulační aplikace „HolR.“ a příklad cesty nalezené genetickým algoritmem pro holonomní robot*

Na záložce *NeholR.* je možné spouštět základní metody plánování cesty pro neholonomní robot. Nastavení parametrů metod je opět intuitivní (viz obr. 5.5). Význam tlačítek *Hledat cestu* a *Zobrazit cestu* je stejný jako na záložce *HolR.* Tlačítkem *Zobrazit strom* je možné vykreslit strom vygenerovaný v průběhu výpočtu. Po stisku tlačítka *Projít cestu* dojde ke spuštění animace pohybu robotu po této cestě. Zobrazovat vytváření stromu v průběhu výpočtu ve vícevláknové implementaci metody inkrementálního vzorkování není možné.



Obr. 5.5. *Záložka simulační aplikace „NeholR.“ a příklad cesty nalezené metodou RRT pro neholonomní robot*

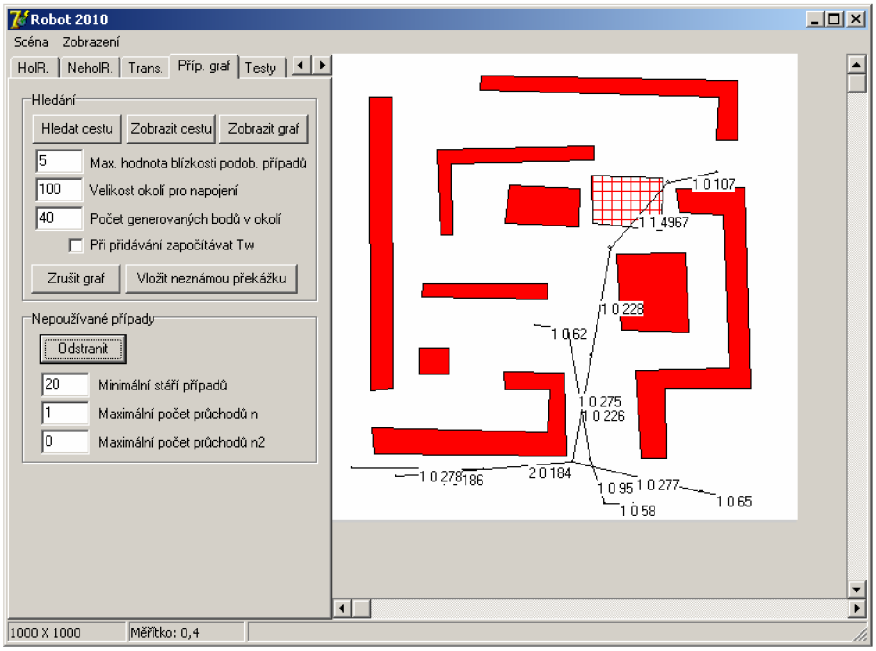
Metodu navrženou v podkapitole 4.2.4 je možné ovládat na záložce *Trans.* V části této záložky nazvané *Genetický algoritmus* se provádí nastavení parametrů genetického algoritmu pro neholonomní robot. Význam tlačítek *Hledat cestu* a *Projít cestu* zůstává zachován. Zobrazení poslední nalezené cesty před provedením transformace je možné provést stiskem tlačítka *Zobrazit cestu hol.* a po provedení transformace stiskem tlačítka *Zobrazit cestu nehol.*



Obr. 5.6. *Záložka simulační aplikace „Trans.“ a příklad cesty pro neholonomní robot nalezené metodou genetického algoritmu s následnou transformací*

Funkčnost metody kombinující případový graf a genetický algoritmus lze ověřit na záložce *Příp. graf.* Pro vytvoření částečně známé scény slouží tlačítko *Vložit neznámou překážku*. Tuto překážku vložíme stejným způsobem jako na záložce *Překážky*.

Pravděpodobnost výskytu neznámé překážky je 0,8. Doba výskytu této překážky se generuje z rozsahu 5000 až 10000 časových jednotek. Změna těchto parametrů je možná pouze ve zdrojovém kódu. Případový graf je rozšiřován či aktualizován podle každé nalezené cesty, pro kterou je ihned spouštěna simulace průchodu prostředím. Případový graf můžeme zobrazit stiskem tlačítka *Zobrazit graf*. Nastavení parametrů metody je opět zřejmé z obr. 5.7. Při zrušení volby *Při přidání započítávat Tw* bude při přidávání případu vždy $t_W = 0$. Odstranění nepoužívaných případů je možné provést stiskem tlačítka *Odstranit*.



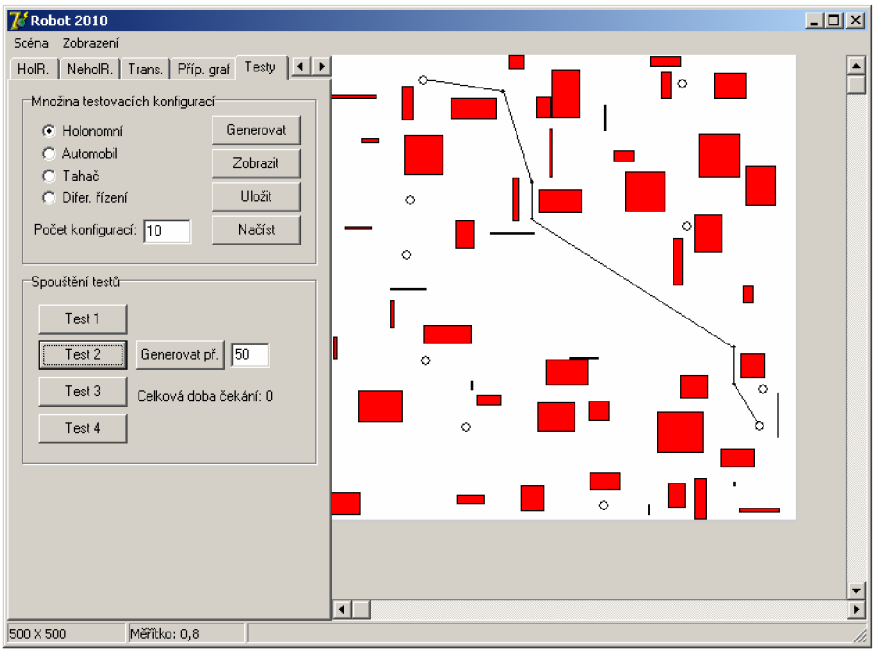
Obr. 5.7. *Záložka simulační aplikace „Příp. graf" a ukázka případového grafu ve scéně s neznámou překážkou*

Poslední záložka *Testy* obsahuje prvky pro manipulaci s testovací množinou a tlačítka pro spouštění některých testů (viz obr. 5.8). Po vygenerování testovací množiny tlačítkem *Generovat* je možné ji zobrazit stiskem tlačítka *Zobrazit* nebo uložit do souboru stiskem tlačítka *Uložit*. Opětovné načtení je provedeno po stisku tlačítka *Načíst*.

Všechny testy jsou prováděny nad vygenerovanou testovací množinou a nad vytvořenou scénou. Tlačítko *Test 1* spouští optimalizaci genetického algoritmu. Optimalizované parametry genetického algoritmu jsou v průběhu optimalizace ukládány do textového souboru *Testlog_GA_optimalizace.txt*.

Tlačítkem *Test 2* spouštíme porovnání genetického algoritmu pro holonomní robot a plánování pomocí grafu viditelnosti (viz podkapitola 6.5). Pro náhodné vygenerování scény je možné stisknout tlačítko *Generovat př.*, čímž dojde k vytvoření daného počtu náhodných překážek ve scéně (viz obr. 5.8). Parametry metod jsou přebírány ze záložky *HolR*. Výsledky tohoto testu jsou postupně přidávány do souboru *Testlog_porovnani_GA_GV.txt*.

Tlačítkem *Test 3* spouštíme experiment s učením případového grafu. Během experimentu je cesta vyhledávána pomocí případového grafu i prostřednictvím metody implementované na záložce *Trans*. Tímto získáváme výsledky experimentu podkapitoly 6.8.1. Pokud scéna obsahuje neznámou překážku, pak získáváme výsledky experimentu podkapitoly 6.8.2. Parametry metod jsou přebírány ze záložek *Trans* a *Příp. graf*. Výsledky jsou zaznamenávány do souborů *Testlog_3_CG.txt* a *Testlog_3_GATRpar.txt*.



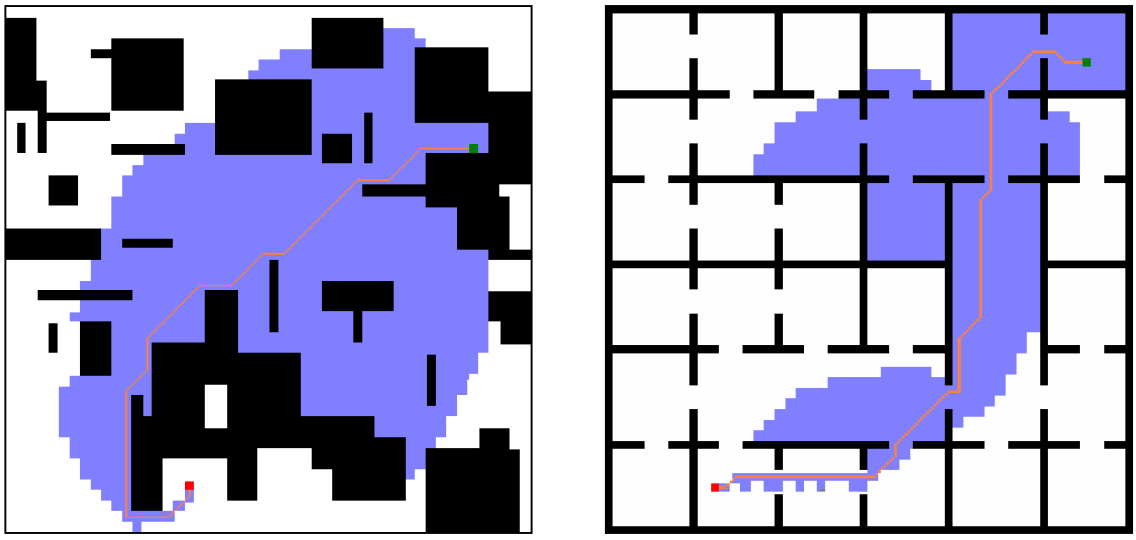
Obr. 5.8. *Záložka simulační aplikace „Testy“ a ukázka cesty pro holonomní robot v náhodně vygenerované scéně.*

Stiskem tlačítka *Test 4* je spuštěna řada výpočtů porovnávající metody pro neholonomní robot (viz podkapitola 6.7). Nastavení parametrů metod je přebíráno ze záložek *NeholR.* a *Trans.* Výsledky testů jsou ukládány postupně do textových souborů *Testlog_4_RRT_Z.txt*, *Testlog_4_RRT_R.txt*, *Testlog_4_GAA1.txt*, *Testlog_4_GAA1par.txt*, *Testlog_4_A.txt* a *Testlog_4_Apar.txt*

6. Experimentální ověření

6.1 Experimenty na mřížce

Experimenty v této podkapitole prokazují funkčnost algoritmů navržených v podkapitole 4.1.4. Srovnávali jsme čtyři metody plánování cesty holonomních robotů: metodu založenou jen na Dijkstrovu algoritmu (DA), metodu založenou jen na A*, metodu kombinující A* s normálním případovým grafem (CG1A*) a metodu kombinující A* s „jemnou strukturou“ případového grafu (CG2A*). Experimenty byly prováděny na PC s procesorem AMD Athlon XP1700+ a 512 MB RAM. Cesty a hrany případového grafu byly ohodnoceny jen jejich délkou. Uvažovány byly dva druhy prostředí: venkovní a vnitřní (labyrint). Příklady těchto prostředí s cestami nalezenými pomocí algoritmu A* jsou uvedeny na obr. 6.1 (překážky jsou černé, start je zelený, cíl je červený, nalezená cesta je oranžová a prohledané buňky jsou modré). Obr. 6.2 zobrazuje příklad hledání pomocí CG2A* (cesty případového grafu jsou tmavě modré, světle modrá barva označuje buňky grafu G' a žlutá barva značí prohledané buňky).



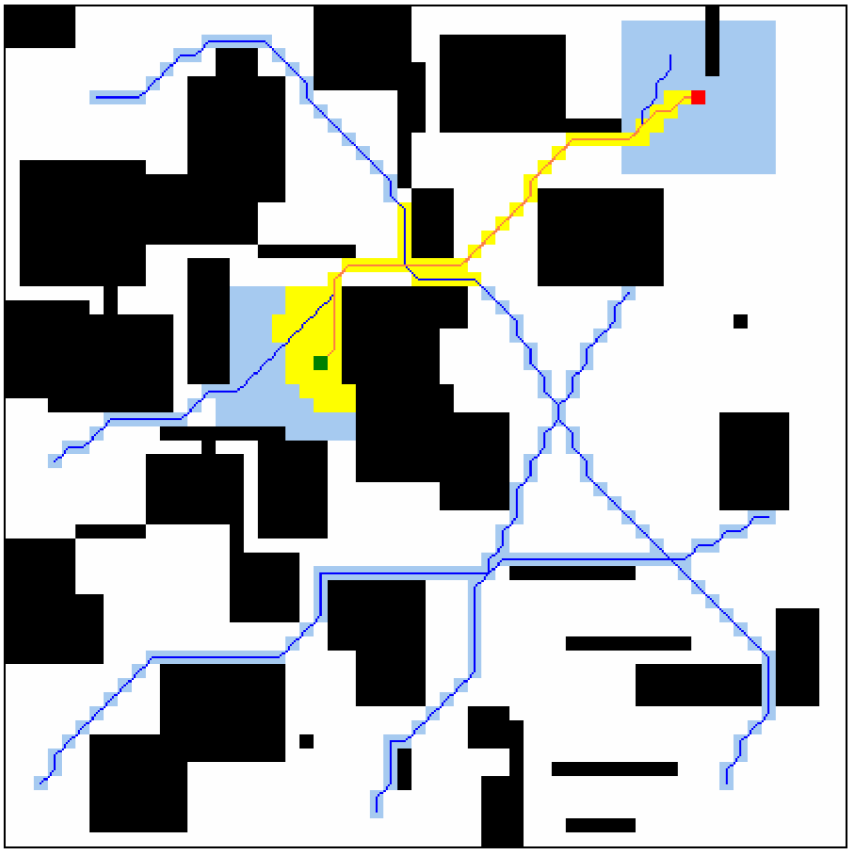
(a) (b)

Obr. 6.1. *Příklady prostředí a cest nalezených algoritmem A*. (a) venkovní prostředí, (b) vnitřní prostředí*

Obr. 6.3 a obr. 6.4 zobrazují, jak průměrná doba hledání cesty závisí na velikosti mřížky. Pro každou velikost mřížky byla vytvořena náhodně případová báze a potom byly hledány cesty pro 100 párů náhodně vygenerovaných buněk (nalezené cesty nebyly ukládány do případové báze). Můžeme pozorovat, že nejlepší metodou pro obě prostředí je metoda CG2A*. Metoda CG1A* pracuje rychleji než DA od velikosti mřížky 300×300 buněk pro všechny typy prostředí. Algoritmus A* je lepší než CG1A* do velikosti mřížky 350×350 buněk pro vnitřní prostředí a do velikosti mřížky 450×450 buněk pro venkovní prostředí. Z obr. 6.3 a obr. 6.4 také plyne fakt, že algoritmus A* je méně vhodný pro vnitřní prostředí než venkovní.

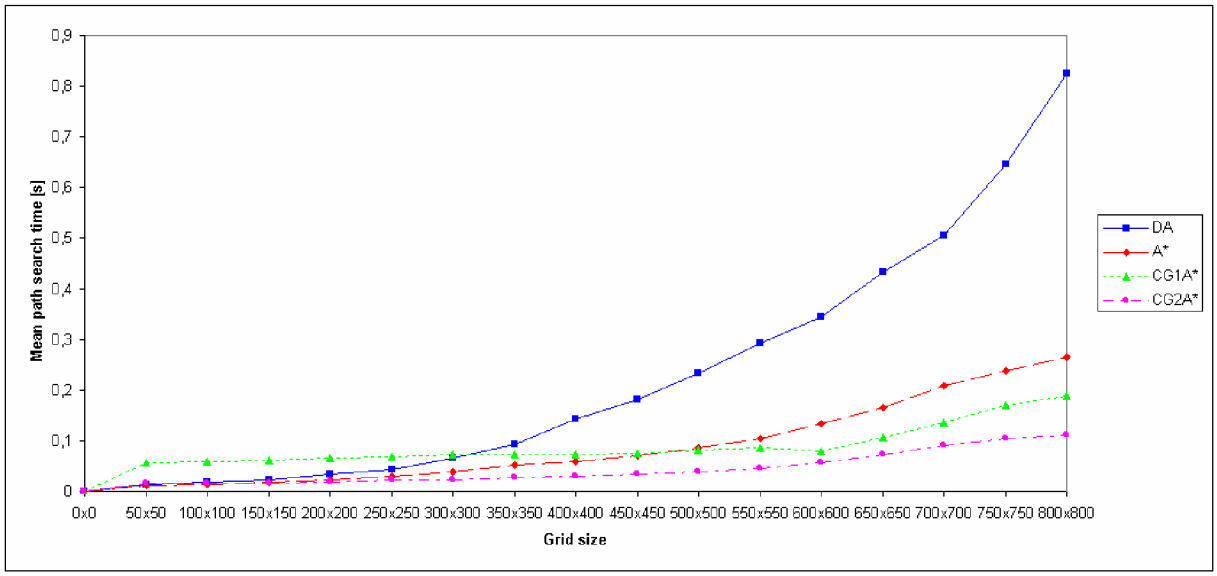
V obr. 6.5 a obr. 6.6 je zakreslena průměrná doba hledání cesty v závislosti na počtu případů pro různé velikosti okolí. Experimenty byly prováděny pro venkovní prostředí a velikost mřížky 600×600 buněk. Tyto experimenty byly spouštěny s prázdnou bází případů a nalezené cesty byly ukládány. Můžeme pozorovat, že přibližně od 50 případů obě metody

CG1A* i CG2A* pracují rychleji než algoritmus A* pro všechny testované velikosti okolí. Kromě okolí o velikosti 25 v obr. 6.6, můžeme na všech dalších křivkách spatřit bod, odkud začíná průměrná doba hledání cesty růst. Tento nárůst je ale velmi pomalý. Je však evidentní, že větší okolí jsou lepší pro malé počty případů, kdežto menší okolí jsou lepší pro větší případové báze. Proto by bylo vhodné velikost okolí metod CG1A* a CG2A* dynamicky měnit v závislosti na velikosti případové báze. Můžeme sledovat, že metoda CG2A* je více citlivější na velikost okolí v porovnání s metodou CG1A* (to je důsledek zjemnění struktury případového grafu použité v metodě CG2A*).

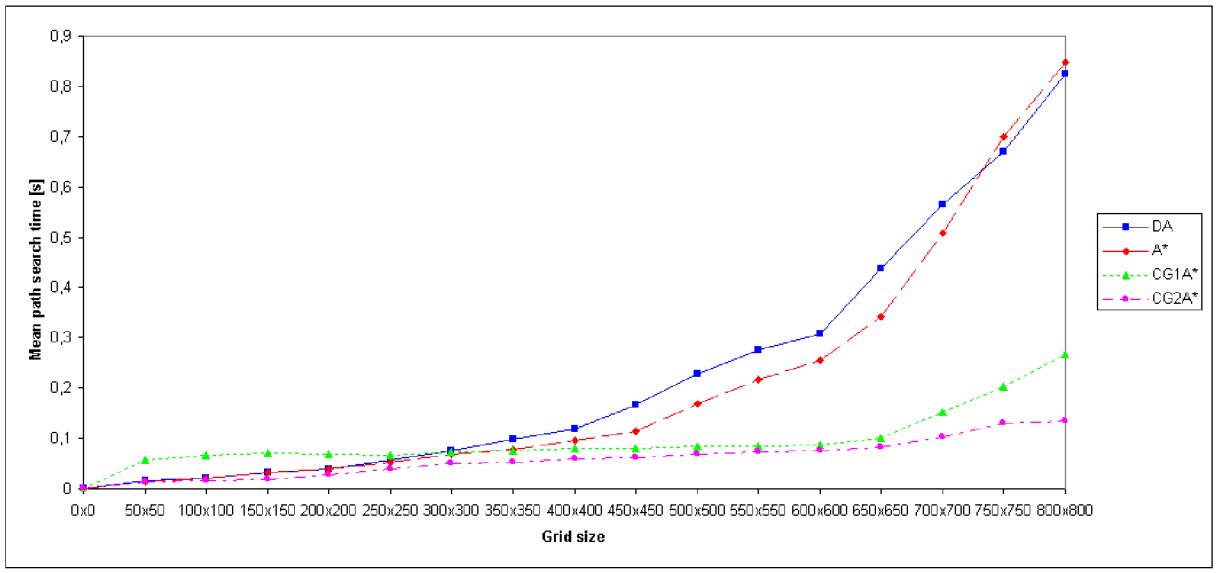


Obr. 6.2. *Příklad hledání cesty prostřednictvím metody CG2A**

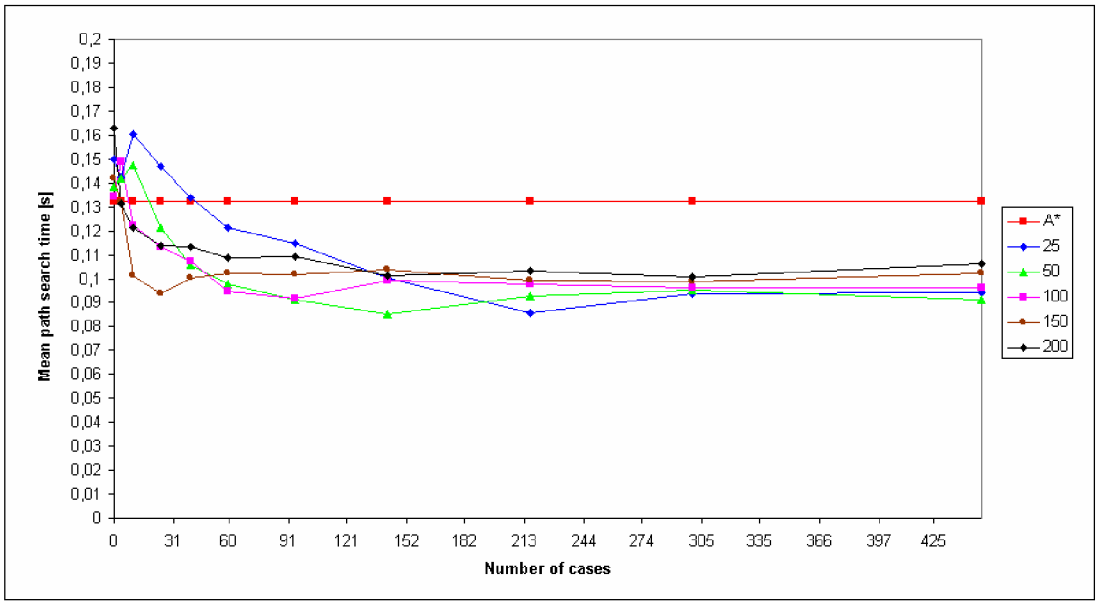
Obr. 6.7 a obr. 6.8 zobrazují závislost průměrné hodnoty poměru délky cesty CGxA* k délce cesty A* na počtu případů a velikosti okolí. Experimenty byly spouštěny stejným způsobem jako v předcházejícím případě. Je zřejmé, že větší počet případů a velikost okolí snižují průměrný poměr délek u obou metod. Kratší cesty jsou nalézány pomocí metody CG2A* (pro okolí 200 jsou délky hledaných cest téměř stejné jako v případě algoritmu A*).



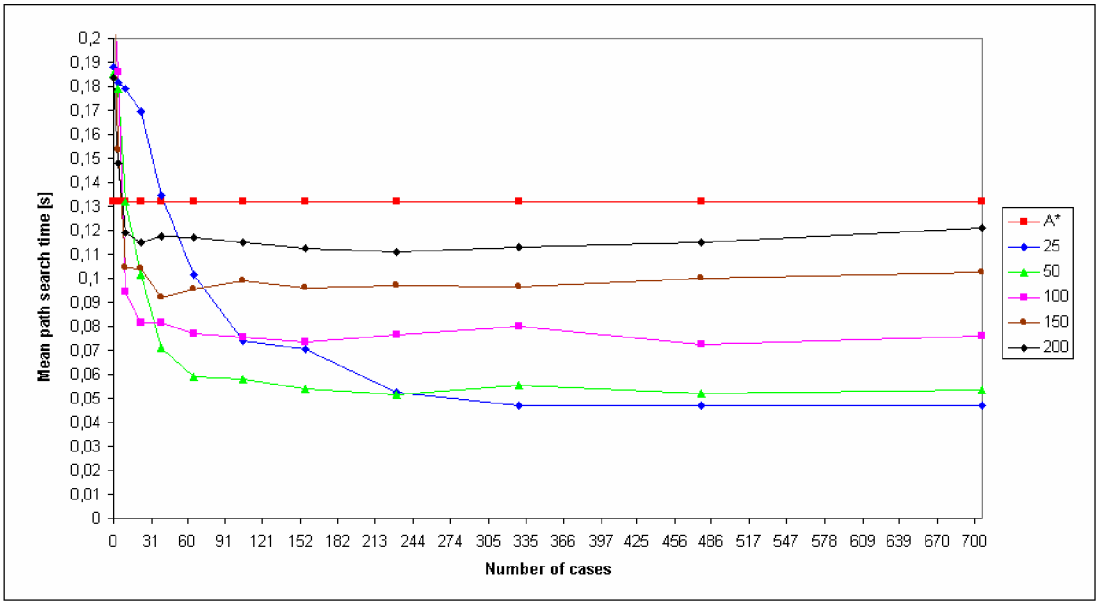
Obr. 6.3. *Průměrná doba hledání cesty v závislosti na velikosti mřížky ve venkovním prostředí*



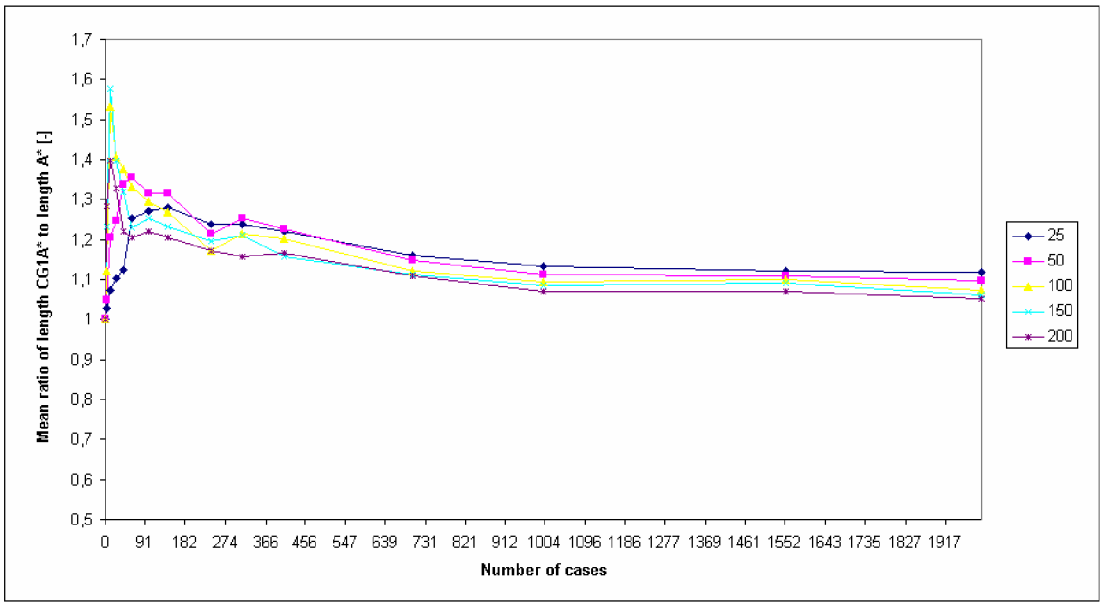
Obr. 6.4. *Průměrná doba hledání cesty v závislosti na velikosti mřížky ve vnitřním prostředí*



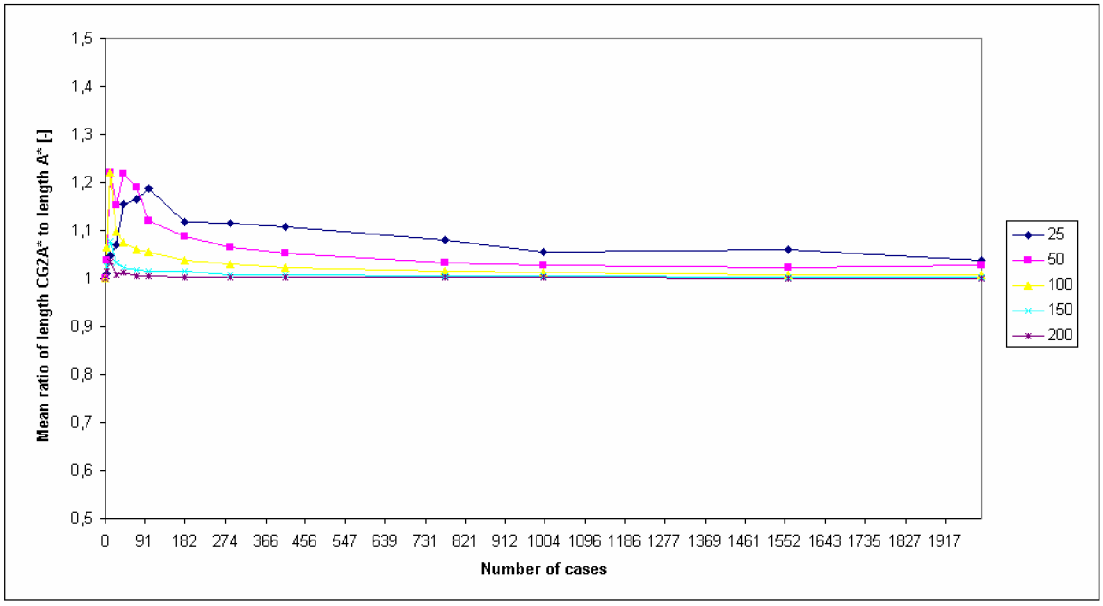
Obr. 6.5. *Průměrná doba hledání cesty v závislosti na počtu případů pro různé velikosti okolí (mřížka 600 × 600, venkovní prostředí, metoda CG1A*)*



Obr. 6.6. *Průměrná doba hledání cesty v závislosti na počtu případů pro různé velikosti okolí (mřížka 600 × 600, venkovní prostředí, metoda CG2A*)*



Obr. 6.7. *Průměrná hodnota poměru délky cesty $CG1A^*$ k délce cesty A^* v závislosti na počtu případů (mřížka 600 × 600, venkovní prostředí)*



Obr. 6.8. *Průměrná hodnota poměru délky cesty $CG1A^*$ k délce cesty A^* v závislosti na počtu případů (mřížka 600 × 600, venkovní prostředí)*

6.2 Kombinace případového grafu a pravděpodobnostních stromů

Pro ověření algoritmů navržených v podkapitole 4.2.1 bylo vytvořeno simulační prostředí. Experimenty byly provedeny ve scéně 800×800 jednotek se zadanými překážkami. V této scéně byla náhodně vygenerována posloupnost 101 konfigurací taková, že vždy pro následující konfiguraci existuje cesta z předcházející konfigurace. Bylo tedy vyhledáváno 100 cest a pro každé hledání byla měřena doba výpočtu a délka nalezené cesty. Vyhledávání se provádělo jednak algoritmem RRT a jednak pomocí algoritmu kombinujícího případový graf

a RRT (CGRRT). Pro experimenty s algoritmem CGRRT byly použity dvě uměle vytvořené báze případů (viz obr. 4.4 a 6.9). Tab. 6.1 obsahuje parametry pro algoritmus RRT. Výsledky spolu s nastavením CGRRT (týká se velikosti okolí δ a použité báze případů) jsou uvedeny v tab. 6.2. Pro simulace byl uvažován neholonomní mobilní tříkolový robot s diferenciálním řízením zadních kol. Rozměry robotu jsou 10×10 jednotek, robot má povolen pouze dopředný pohyb a jeho minimální poloměr otáčení je 10 jednotek. Výběr minima v upraveném Dijkstrově algoritmu byl implementován pomocí haldy. Experimenty byly prováděny na počítači PC s procesorem AMD Athlon XP1700+ a 768 MB RAM.

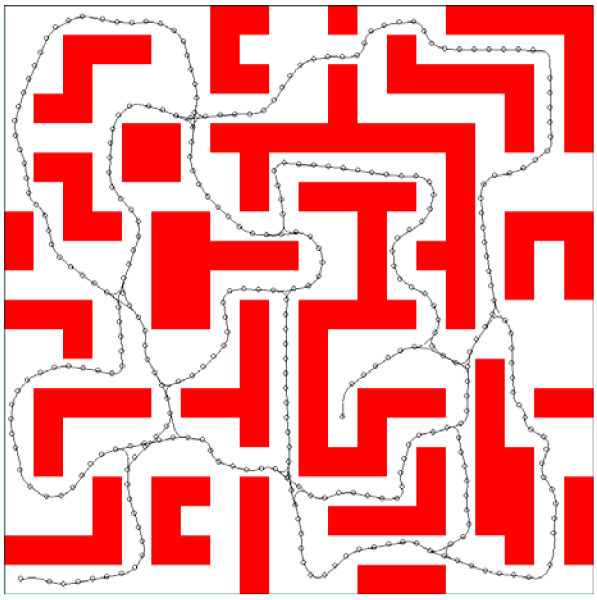
Tab. 6.1. Parametry pro algoritmus RRT

Použití RRT	K	ε [jednotky]	p
pro vyhledávání celých cest	30 000	20	0,95
pro spojení s případovým grafem	2 000	20	0,95

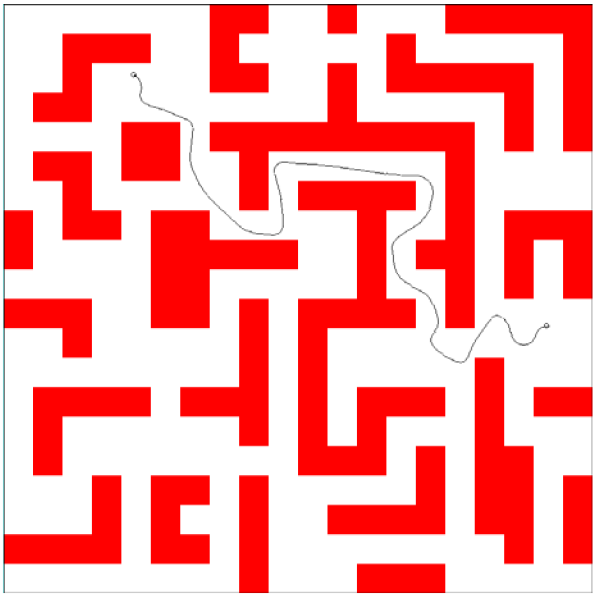
Tab. 6.2. Naměřené doby výpočtů a nastavení CGRRT

Algoritmus	Báze případů	δ [jednotky]	Průměrná doba výpočtu [s]	Průměrná délka trasy [jednotky]	Neúspěšnost CGRRT
RRT	-	-	6,493	1342,242	-
CGRRT	č. 1	100	3,441	1433,048	27 %
CGRRT	č. 2	100	2,612	1206,721	9 %
CGRRT	č. 2	50	2,678	1379,276	23 %

Z provedených experimentů vyplývá, že při vhodné volbě okolí a případové bázi dostatečně pokrývající svými případy scénu, lze dosáhnout vůči samostatnému používání RRT jak zrychlení doby výpočtu, tak i vyhledání kratších cest.



Obr. 6.9. *Případová báze č. 2*



Obr. 6.10. *Příklad cesty vyhledané pomocí algoritmu CGRRT s bází případů č. 2*

6.3 Znovupoužití populace GA

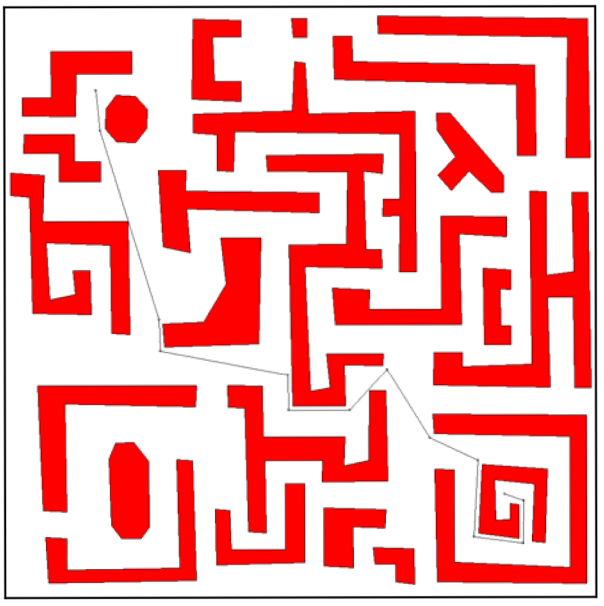
V těchto experimentech bylo zkoumáno chování genetického algoritmu navrženého v podkapitole 4.2.2. Experimenty byly prováděny ve scéně 2000×2000 jednotek s danými překážkami. Parametry genetického algoritmu byly nastaveny experimentálně (tab. 6.3).

Tab. 6.3. *Parametry genetického algoritmu*

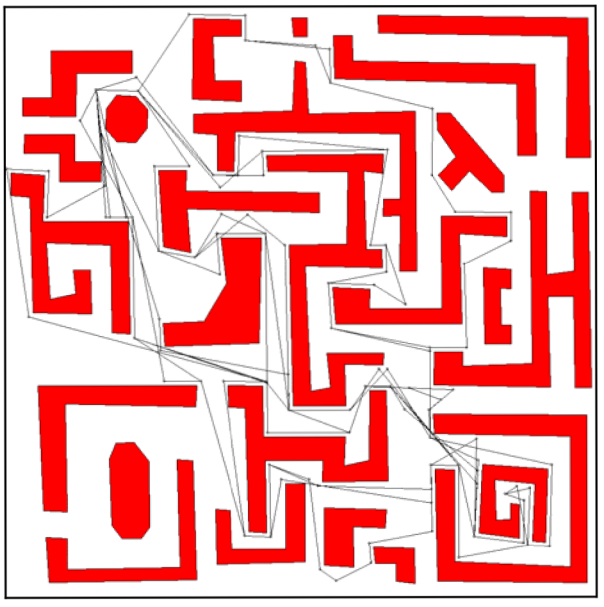
počet chromozomů v populaci	50	pravděpodobnost malé mutace	0,2
počet potomků v jedné generaci	24	pravděpodobnost opravy	0,4
počáteční minimální počet genů	3	pravděpodobnost výměny	0,5
počáteční rozsah počtu genů	3 - 7	pravděpodobnost vyhlazení	0,5
váhy fitness funkce	$w_1 = 1/3$ $w_2 = 1/3$ $w_3 = 1/3$	pravděpodobnost zkrácení	0,5
pravděpodobnost křížení	0,9	pravděpodobnost odstranění	0,5
pravděpodobnost velké mutace	0,1	pravděpodobnost zlepšení	0,2

Schopnosti tohoto algoritmu byly testovány ve scéně podobné bludišti (obr. 6.11). V prvním experimentu byla zkoumána možnost použití části naučené populace předchozího řešení jako část počáteční populace nového problému, který je charakterizován novým startem a/nebo novým cílem ve stejné scéně. Ve druhém experimentu, byla zkoumána možnost použití konečné populace současného řešení pro přeplánování cesty z důvodu výskytu neznámé překážky. V tomto případě, přeplánování začíná v novém startu, ale cíl zůstává stejný. V obou experimentech je každá cesta ve staré populaci upravena tak, že nový start (cíl) je spojen s nejbližším uzlem této cesty. Porovnány jsou výsledky získané pro tyto tři případy: počáteční populace obsahuje jen nové chromozomy (NCH), počáteční populace obsahuje polovinu starých a polovinu nových chromozomů (ONCH) a počáteční populace obsahuje jen staré chromozomy (OCH).

V prvním experimentu vedlo použití chromozomů z naučené populace ke snížení doby výpočtu (obr. 6.12). Ve druhém experimentu došlo použitím chromozomů z naučené populace také ke zrychlení výpočtu (obr. 6.13) bez zhoršení hodnoty fitness funkce (obr. 6.14). Z experimentů je patrné, že pro testovanou scénu je nejlepších výsledků dosaženo, když celá stará populace je použita jako počáteční populace nového problému.

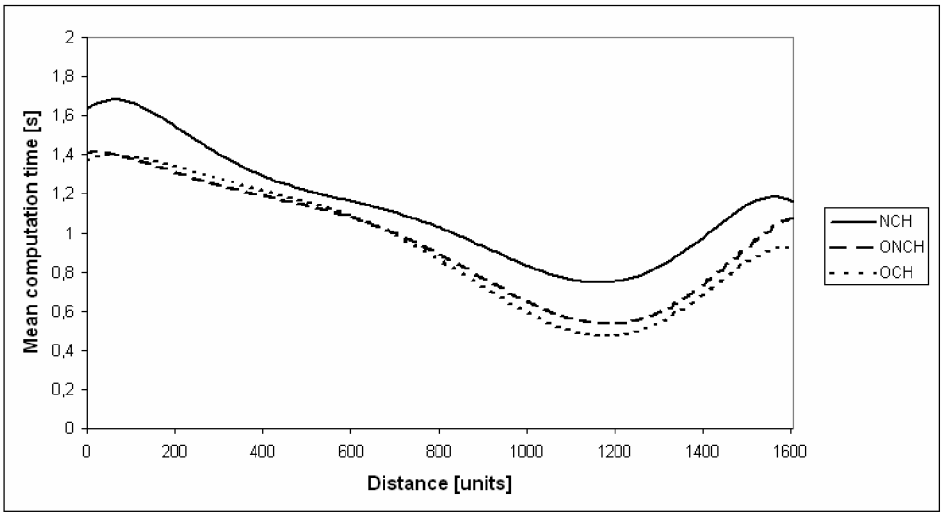


(a)

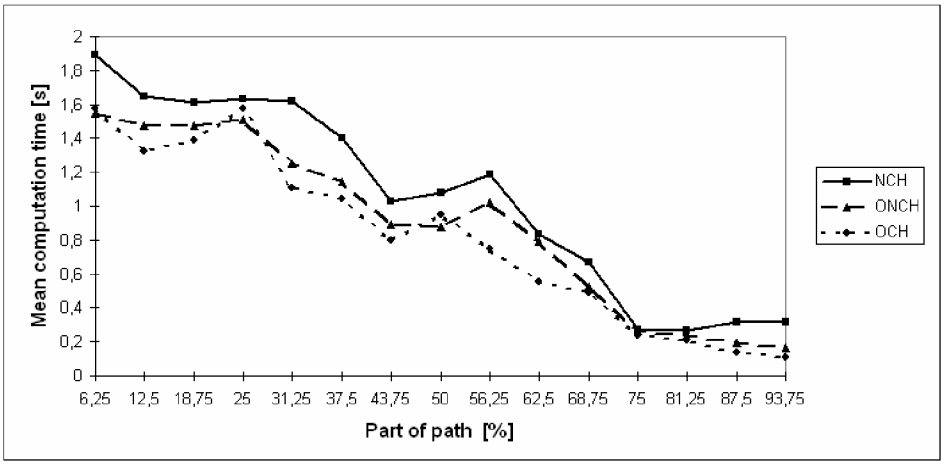


(b)

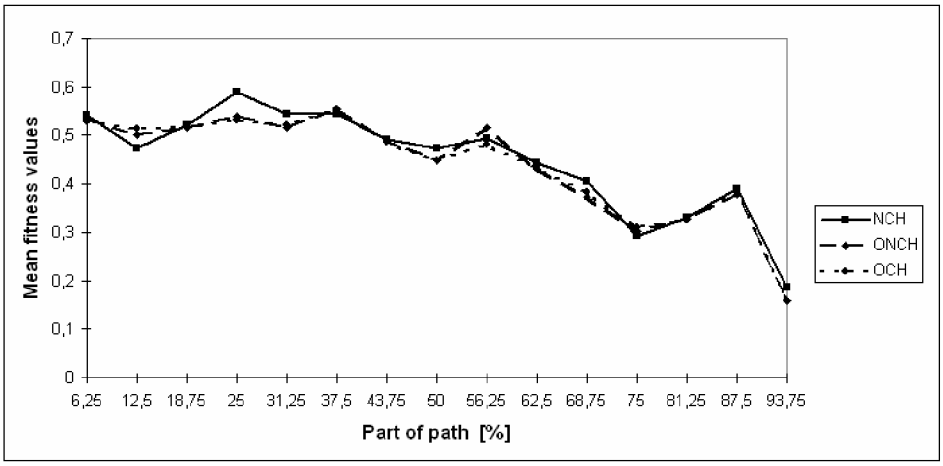
Obr. 6.11. *Příklad nalezené cesty (a) a přípustné chromozomy korespondující populace (b)*



Obr. 6.12. *Závislosti doby výpočtu na průměrné vzdálenosti z nového do starého startu a vzdálenosti z nového do starého cíle*



Obr. 6.13. *Závislosti doby výpočtu na procentuálně vyjádřené části cesty absolvované před přeplánováním*



Obr. 6.14. *Závislosti fitness hodnot na procentuálně vyjádřené části cesty absolvované před přeplánováním*

6.4 Optimalizace parametrů GA

V této podkapitole je provedena optimalizace parametrů genetických algoritmů navržených v podkapitole 4.2.2 a 4.2.4. Optimalizace je prováděna prostřednictvím dalšího genetického algoritmu (GA1). Princip optimalizace je vysvětlen v podkapitole 4.2.5. Ve všech následujících experimentech byly použity zde vypočtené optimalizované hodnoty parametrů.

6.4.1 GA pro holonomní robot

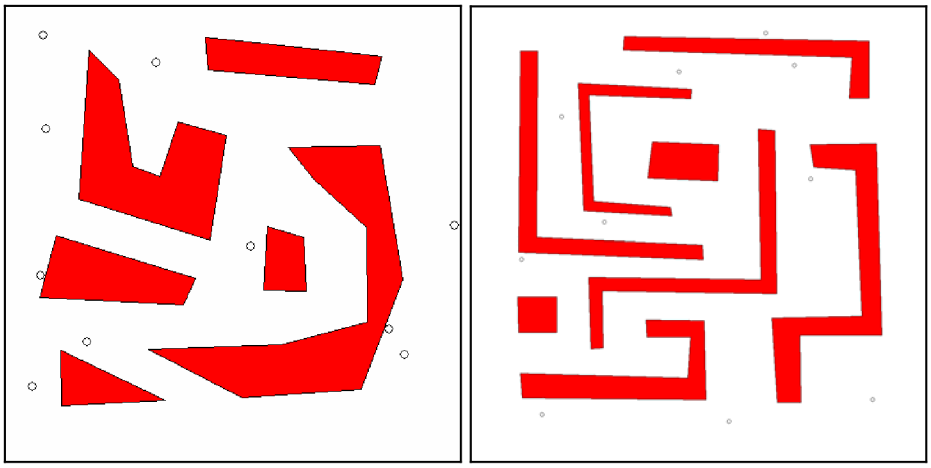
Optimalizace parametrů genetického algoritmu (GA2) pro holonomní robot (viz podkapitola 4.2.2) byla prováděna pro tři prostředí o velikostech 500×500 (obr. 6.15a), 1000×1000 (obr. 6.15b) a 2000×2000 (obr. 6.11). Pro GA2 byla náhodně vygenerována testovací množina o počtu 10 konfigurací. V průběhu optimalizace bylo spouštěno hledání vždy pro jednu sadu hodnot parametrů mezi všemi testovacími konfiguracemi, celkem tedy proběhlo 45 hledání. Pevné parametry GA2 jsou uvedeny v tab. 6.4. Ostatní nastavení parametrů bylo předmětem optimalizace. Zvětšení překážek bylo provedeno o hodnotu 1. Parametry GA1 uvádí tab. 6.5. Fitness hodnota chromozomu GA1 je rovna průměrné hodnotě fitness nejlepších nalezených cest.

Tab. 6.4. *Pevné parametry genetického algoritmu GA2*

počet chromozomů v generaci	50
počáteční velikost chromozomu	<1, 5>
váhy w_1, w_2, w_3	1/3; 1/3; 1/3

Tab. 6.5. *Parametry genetického algoritmu GA1*

počet chromozomů v generaci	30
pravděpodobnost křížení	0,75
pravděpodobnost mutace	0,05



(a) (b)

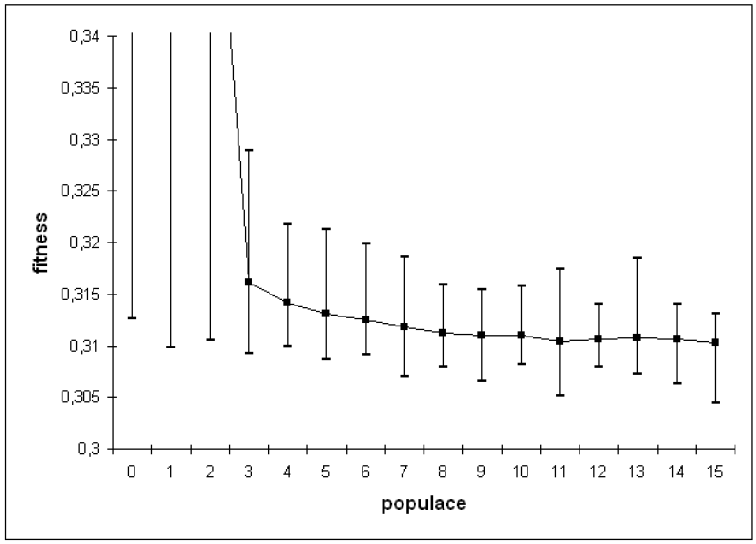
Obr. 6.15. *Prostředí 500×500 (a) a 1000×1000 (b) uvažované při optimalizaci parametrů GA2. Na obrázcích jsou vyznačeny testovací konfigurace.*

Průběhy optimalizovaných hodnot parametrů GA2 byly pro všechny prostředí velmi podobné. Na ukázku (obr. 6.17 až 6.26) byly vybrány průběhy hodnot parametrů během optimalizace pro prostředí 1000×1000. Na obr. 6.16 je zobrazeno zlepšování hodnot fitness funkce v každé populaci. Na obrázcích jsou také zobrazeny minimální a maximální hodnoty příslušných parametrů v celé populaci. V těchto závislostech lze pozorovat rostoucí tendenci počtu párů ke křížení (obr. 6.26), hodnoty pravděpodobnosti operátoru opravy (obr. 6.20) nebo hodnoty pravděpodobnosti operátoru vyhlazení (obr. 6.22). Lze je tak považovat za nejdůležitější nastavení. Oproti tomu pravděpodobnost operátoru odstranění poměrně rychle klesá (obr. 6.24), lze se tedy domnívat, že GA2 nebude na tomto operátoru příliš závislý. Hodnoty pravděpodobností ostatních operátorů se nijak výrazně nevzdalují od hodnoty 0,5.

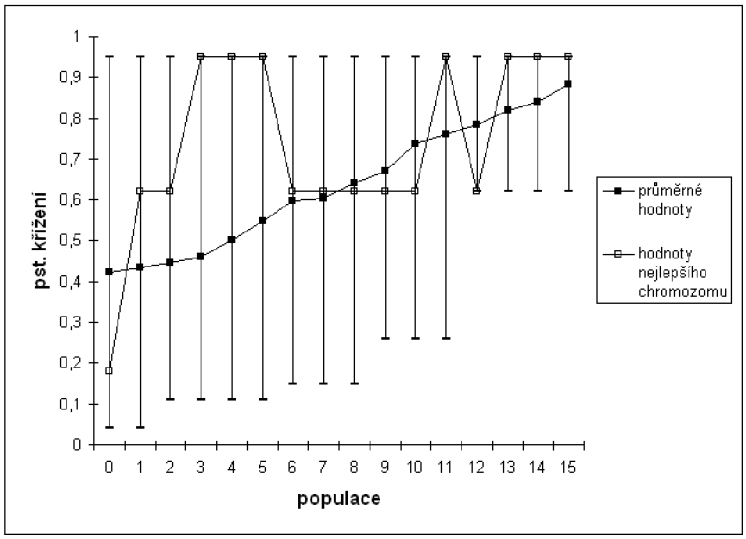
Na základě získaných závislostí pro všechna tři prostředí, byly stanoveny hodnoty parametrů GA2, které jsou uvedeny v tabulce 6.6.

Tab. 6.6. *Optimalizované parametry genetického algoritmu GA2*

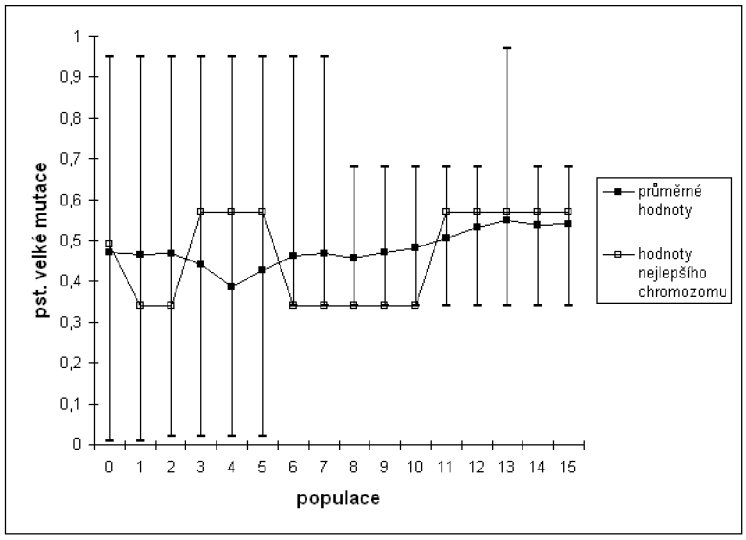
pravděpodobnost operátoru křížení	0,9
pravděpodobnost operátoru velké mutace	0,5
pravděpodobnost operátoru malé mutace	0,4
pravděpodobnost operátoru opravy	0,7
pravděpodobnost operátoru výměny	0,5
pravděpodobnost operátoru vyhlazení	0,5
pravděpodobnost operátoru zkrácení	0,5
pravděpodobnost operátoru odstranění	0,1
pravděpodobnost operátoru vylepšení	0,3
počet chromozomů při selekci	24



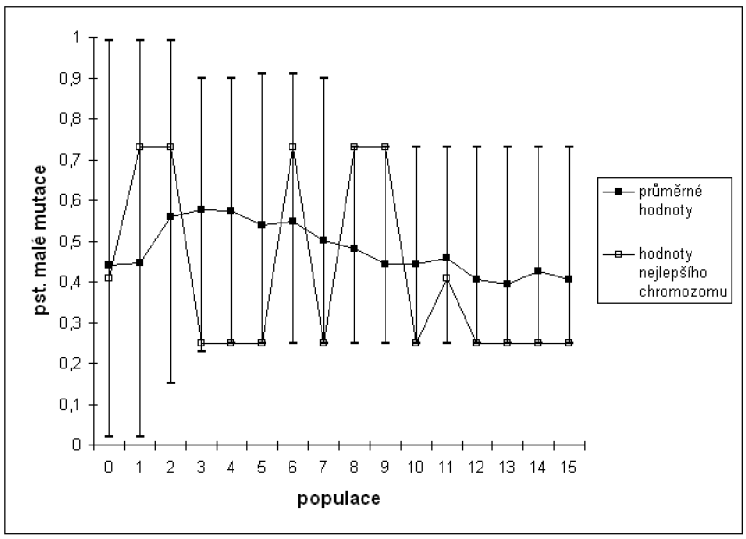
Obr. 6.16. *Průběh hodnot fitness funkce*



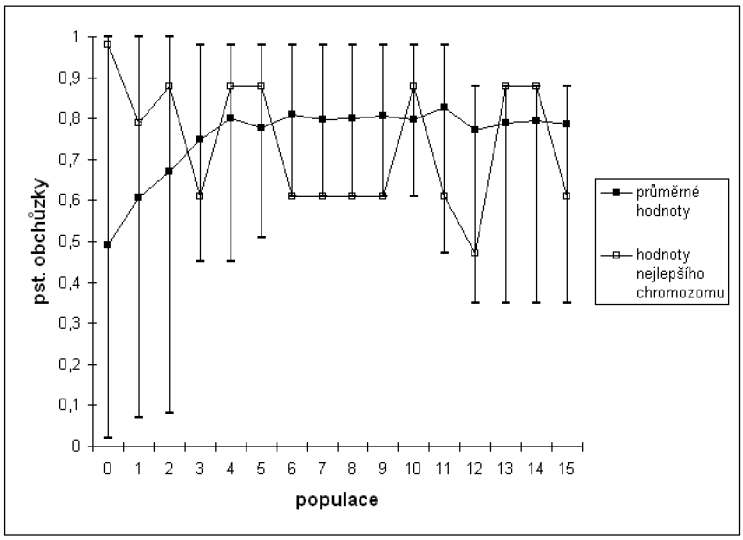
Obr. 6.17. *Průběh optimalizace pravděpodobnosti operátoru křížení*



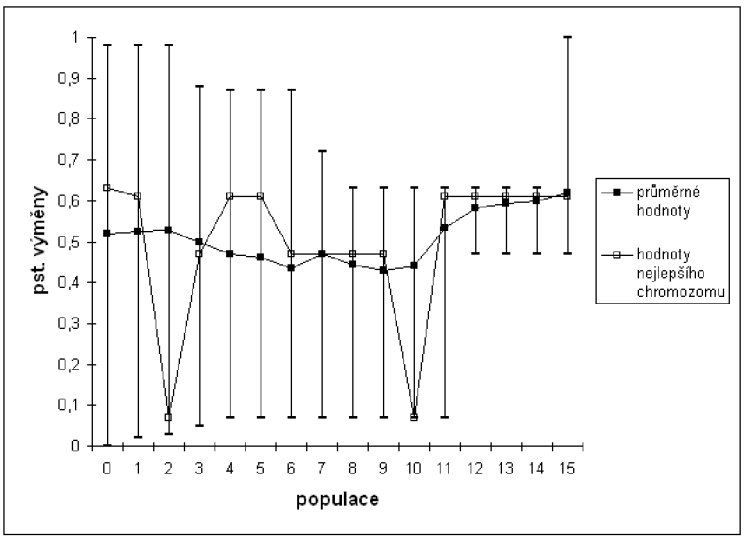
Obr. 6.18. *Průběh optimalizace pravděpodobnosti operátoru velké mutace*



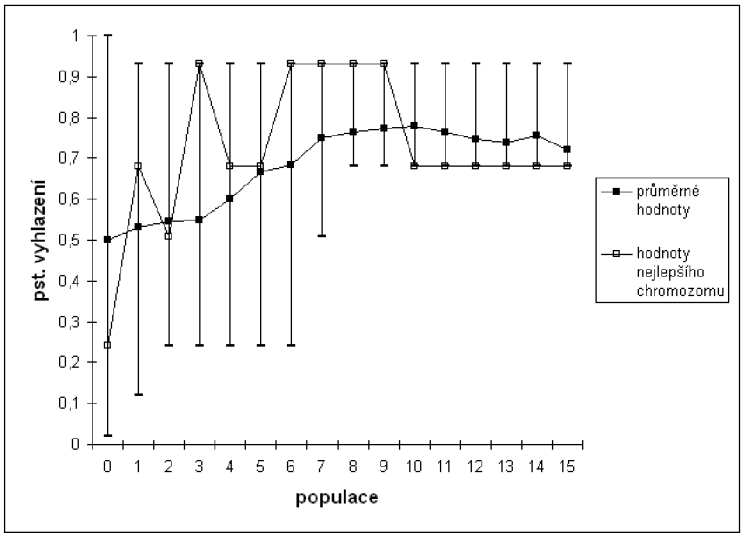
Obr. 6.19. *Průběh optimalizace pravděpodobnosti operátoru malé mutace*



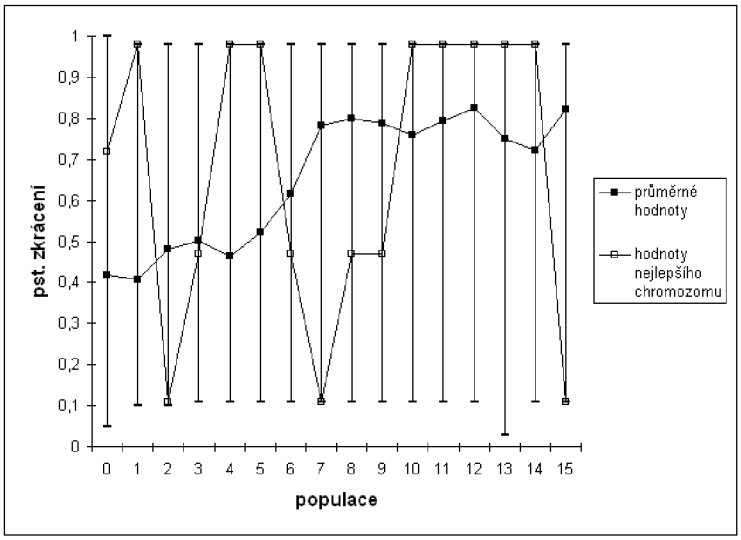
Obr. 6.20. *Průběh optimalizace pravděpodobnosti operátoru opravy*



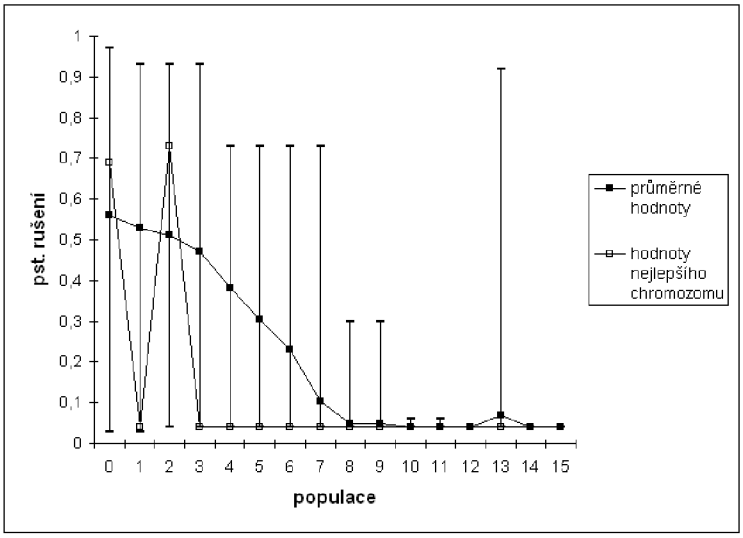
Obr. 6.21. *Průběh optimalizace pravděpodobnosti operátoru výměny*



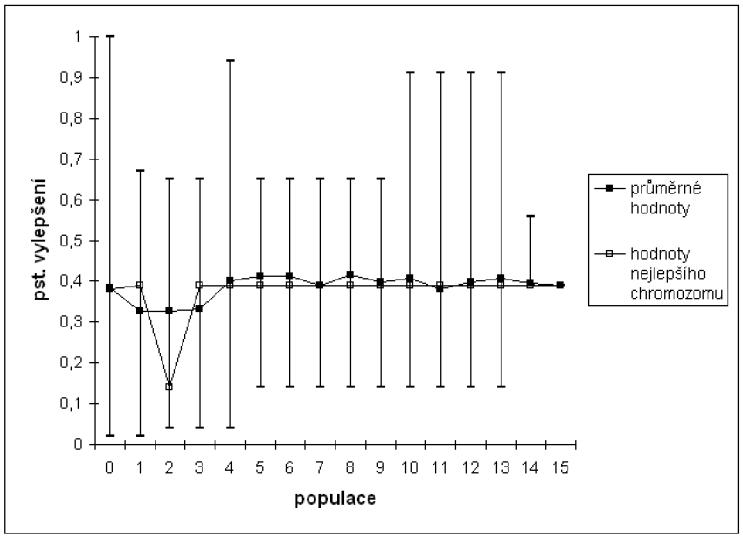
Obr. 6.22. *Průběh optimalizace pravděpodobnosti operátoru vyhlazení*



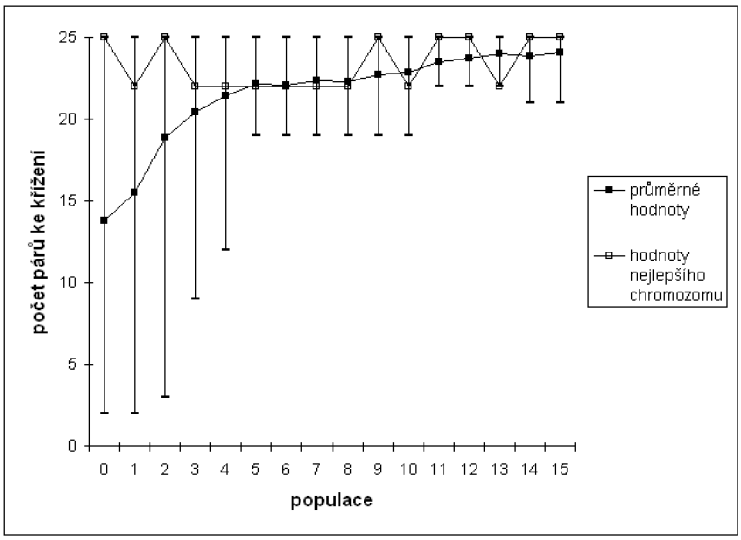
Obr. 6.23. *Průběh optimalizace pravděpodobnosti operátoru zkrácení*



Obr. 6.24. *Průběh optimalizace pravděpodobnosti operátoru odstranění*



Obr. 6.25. *Průběh optimalizace pravděpodobnosti operátoru vylepšení*



Obr. 6.26. *Průběh optimalizace počtu párů ke křížení*

6.4.2 GA pro neholonomní robot

Optimalizace parametrů genetického algoritmu (GA2) pro neholonomní robot (viz podkapitola 4.2.4) byla prováděna pro prostředí o velikosti 1000×1000 (viz obr. 6.53). Pro GA2 byla náhodně vygenerována testovací množina o počtu 10 konfigurací. Nebyly však uvažovány konfigurace, ke kterým nebylo možné vytvořit sousední hranu splňující omezení d_{Min} a $\beta_{StartMax}$ nebo $\beta_{GoalMax}$. Pro jednu sadu hodnot parametrů bylo spuštěno hledání cesty pro každou uspořádanou dvojici z testovací množiny, celkem tedy 90 hledání. Pevné parametry GA2 jsou uvedeny v tab. 6.7. Zvětšení překážek bylo provedeno o hodnotu 20. Parametry GA1 byly voleny stejně jako v předchozí podkapitole (viz tab. 6.5). Fitness hodnota f chromozomu GA1 (jedné sady hodnot parametrů GA2) je dána vztahem

$$f = \frac{1}{180}\sum_{i=1}^{90}\left(\frac{T_i}{T_{Max}} + F(P_i)\right), \tag{6.1}$$

kde T_i je doba výpočtu i-tého hledání cesty, T_{Max} je maximální povolená doba výpočtu a P_i je cesta (chromozom GA2) s nejlepší hodnotou fitness v poslední populaci i-tého hledání.

Tab. 6.7. *Parametry genetického algoritmu GA2*

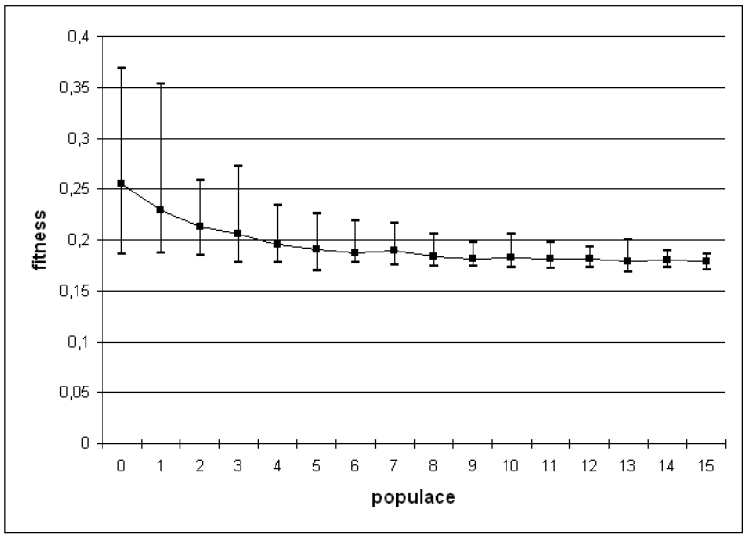
počet chromozomů v generaci	50
počáteční velikost chromozomu	<1, 5>
váhy w_1, w_2, w_3	0,2; 0,2; 0,6
omezení $\beta_{StartMax}$, $\beta_{GoalMax}$	2,8
omezení β_{SegMax}	1,6
omezení d_{Min}	60

Průběhy optimalizace parametrů a změny hodnoty fitness funkce jsou zobrazeny na obr. 6.27 až 6.37. Rostoucí hodnoty pravděpodobností lze pozorovat u operátorů křížení (obr. 6.28), úhlu startu a cíle (obr. 6.34), viditelnosti (obr. 6.36) ale i u operátoru odstranění (obr. 6.35). Naopak výrazně klesající charakter má počet párů ke křížení (obr. 6.37). Při podrobnějším zkoumání bylo zjištěno, že mechanismus křížení velmi výrazně prodlužuje

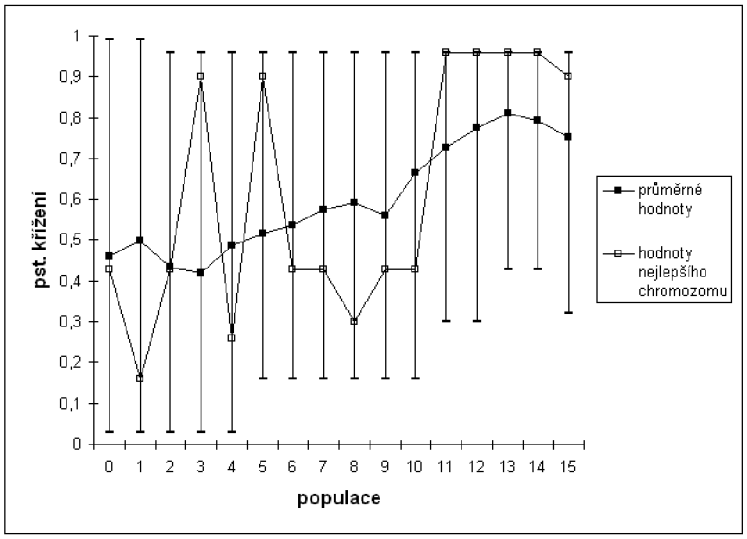
dobu výpočtu při zanedbatelném zlepšení hodnoty fitness funkce. Na základě provedených výpočtů byly stanoveny hodnoty parametrů GA2, které jsou uvedeny v tabulce 6.8.

Tab. 6.8. *Optimalizované parametry genetického algoritmu GA2*

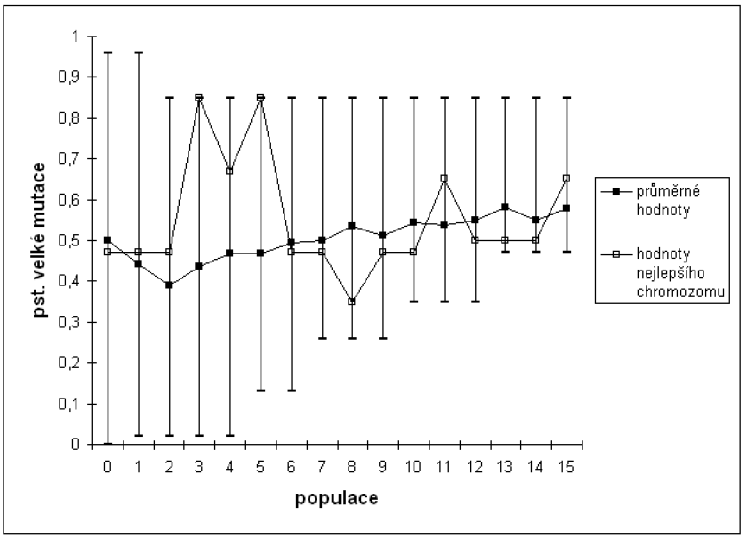
pravděpodobnost operátoru křížení	0,75
pravděpodobnost operátoru velké mutace	0,58
pravděpodobnost operátoru malé mutace	0,37
pravděpodobnost operátoru opravy	0,27
pravděpodobnost operátoru výměny	0,4
pravděpodobnost operátoru vyhlazení	0,26
pravděpodobnost operátoru úhlu startu a cíle	0,79
pravděpodobnost operátoru odstranění	0,76
pravděpodobnost operátoru viditelnosti	0,76
počet chromozomů při selekci	2



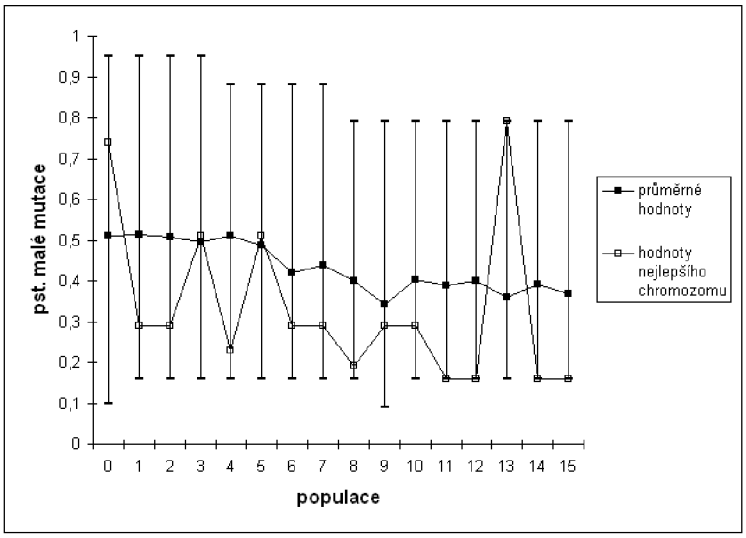
Obr. 6.27. *Průběh hodnot fitness funkce*



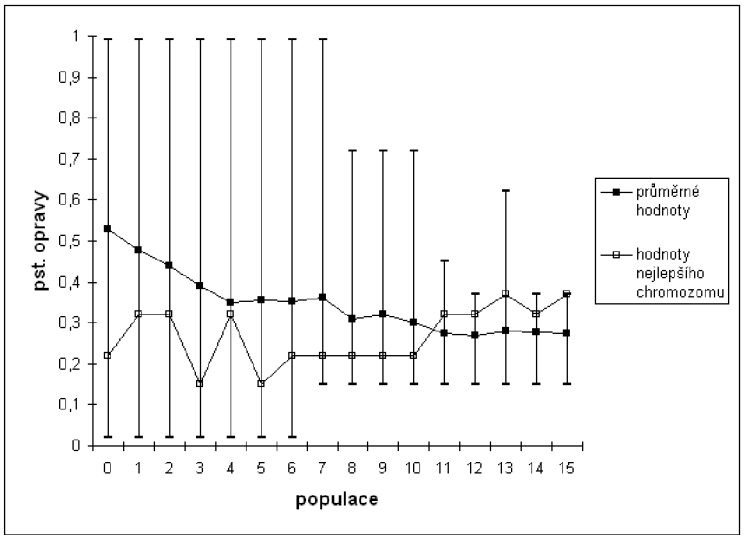
Obr. 6.28. *Průběh optimalizace pravděpodobnosti operátoru křížení*



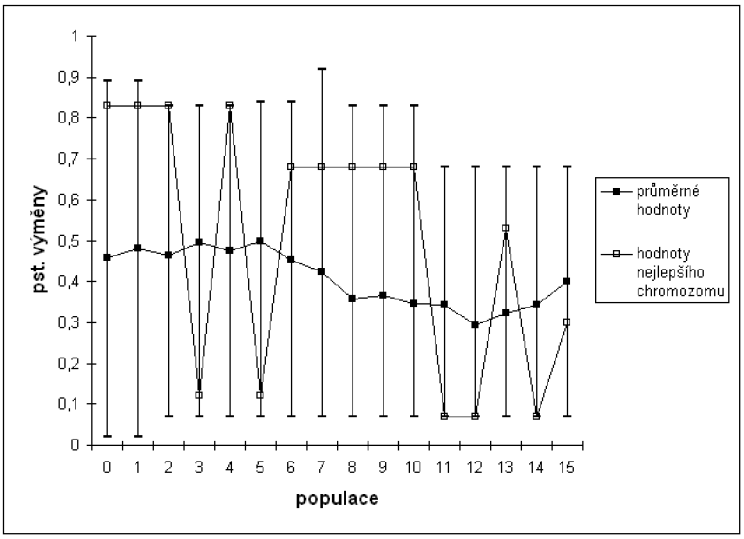
Obr. 6.29. *Průběh optimalizace pravděpodobnosti operátoru velké mutace*



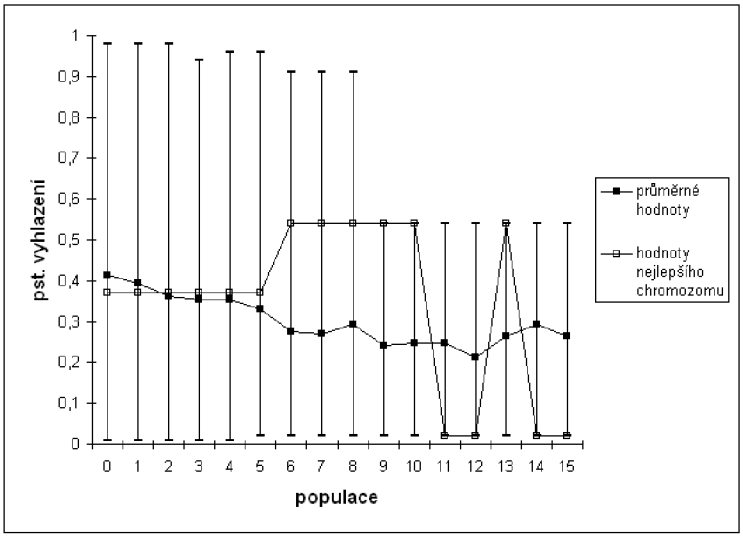
Obr. 6.30. *Průběh optimalizace pravděpodobnosti operátoru malé mutace*



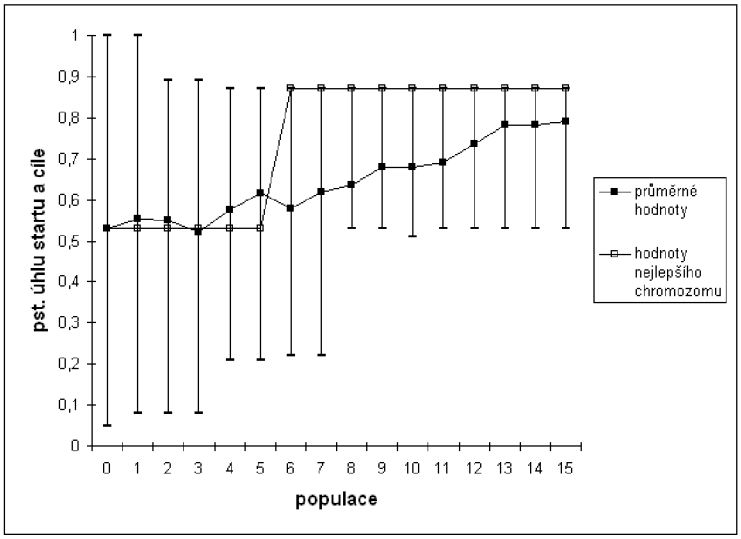
Obr. 6.31. *Průběh optimalizace pravděpodobnosti operátoru opravy*



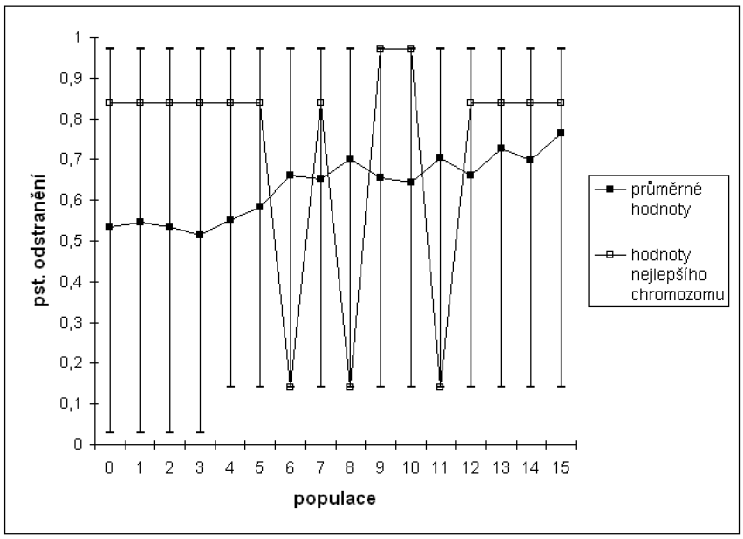
Obr. 6.32. *Průběh optimalizace pravděpodobnosti operátoru výměny*



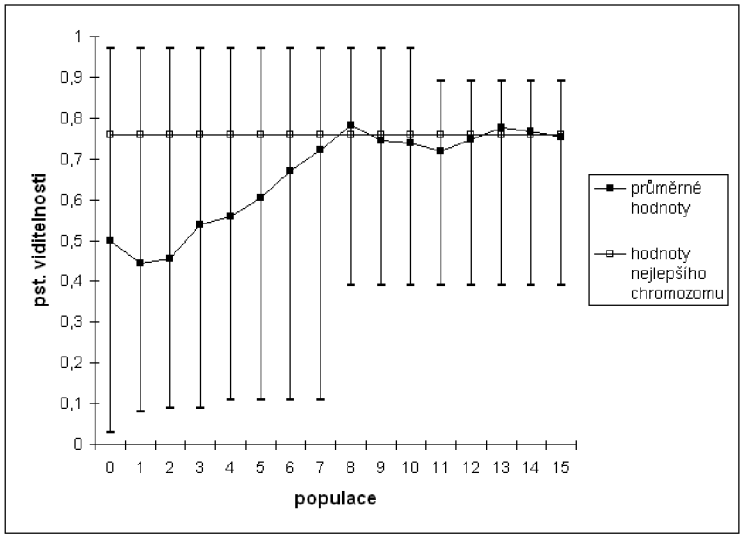
Obr. 6.33. *Průběh optimalizace pravděpodobnosti operátoru vyhlazení*



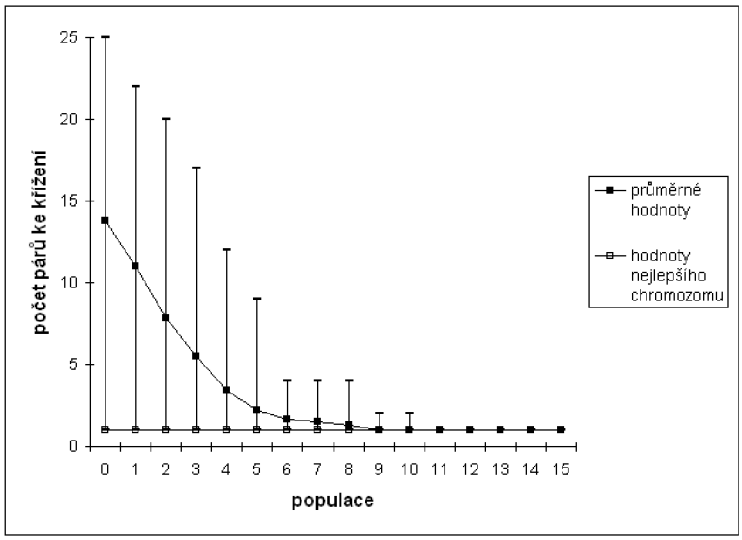
Obr. 6.34. *Průběh optimalizace pravděpodobnosti operátorů úhlu startu a cíle*



Obr. 6.35. *Průběh optimalizace pravděpodobnosti operátoru odstranění*



Obr. 6.36. *Průběh optimalizace pravděpodobnosti operátoru viditelnosti*

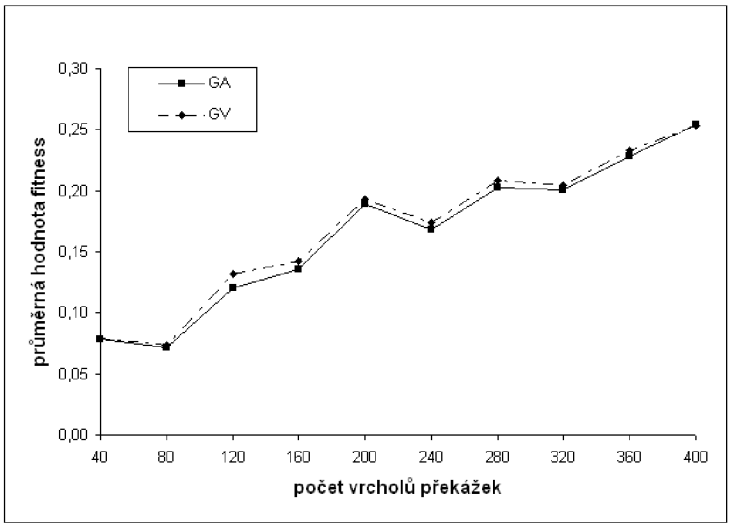


Obr. 6.37. *Průběh optimalizace počtu párů při selekci*

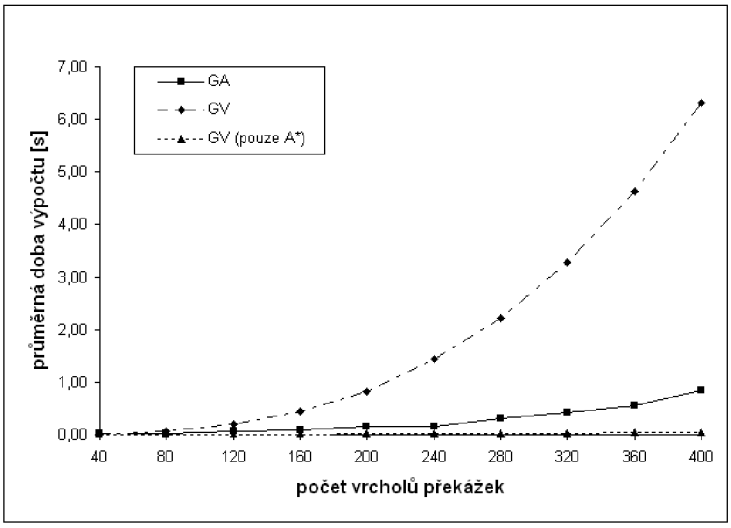
6.5 Porovnání GA a GV

Následující experimenty porovnávají genetický algoritmus pro holonomní robot navržený v podkapitole 4.2.2 s algoritmem používajícím graf viditelnosti (viz podkapitola 3.1). Experimenty probíhaly ve scéně velikosti 500×500, v níž byly náhodně generovány obdélníkové překážky maximální velikosti 50×50. Pro každou scénu byla vytvořena náhodná testovací množina o velikosti 10 konfigurací. Pro každou scénu byl také vytvořen graf viditelnosti (GV), přičemž bylo uvažováno zvětšení překážek o 1 jednotku. Pomocí GA a GV byly hledány cesty mezi všemi konfiguracemi testovací množiny. Cesty nalezené pomocí GV byly ohodnoceny fitness funkcí (4.20). GV byl prohledáván algoritmem A*. Parametry GA byly nastaveny na optimalizované hodnoty podle tab. 6.6. Zbývající hodnoty parametrů byly nastaveny podle tab. 6.4.

Z porovnání průměrných hodnot fitness na obr. 6.38 plyne, že GA je schopné GV konkurovat. GA poskytuje mírně lepší řešení než GV. Je to dáno tím, že v GV je hledána pouze nejkratší cesta a oproti GA není tato cesta vyhlazována. Obr. 6.39 zobrazuje průměrné doby výpočtu GA, tvorby GV s následným prohledáváním a prohledávání v již existujícím GV. Z tohoto experimentu je zřejmé, že předpokládáme-li opakované generování GV, pak při použití GA můžeme získat až šestinásobné zrychlení výpočtu.



Obr. 6.38. *Porovnání kvality cesty nalezené pomocí GA a pomocí GV*



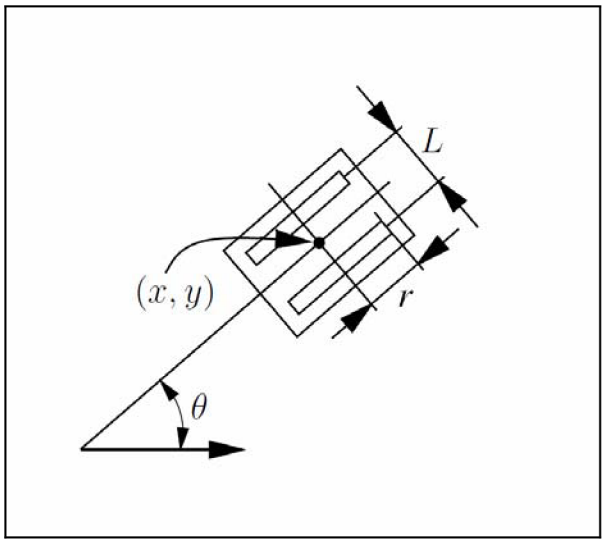
Obr. 6.39. *Porovnání průměrné doby výpočtu plánování cesty pomocí GA a pomocí GV*

6.6 Popisy modelů neholonomních robotů

Jelikož jsou v této práci navrženy metody plánování cesty jako nezávislé na typu robotu, experimenty byly prováděny se třemi různými modely o různých složitostech. Detailní rozbor použitých modelů robotů lze najít v práci (LaValle 2006).

6.6.1 Robot s diferenciálním řízením

První uvažovaný model odpovídá robotu s *diferenciálním řízením*. Pro tyto roboty jsou charakteristická dvě poháněná kola (každé je poháněno vlastním motorem) a případná další pasivní kola, která slouží jako opěrné body. Když se motory otáčí stejně velkou rychlostí se stejnou orientací, robot pojede přímým směrem. Pokud se motory otáčí různou rychlostí, pak robot zatáčí. Když se budou motory otáčet stejně velkou rychlostí ale s opačnou orientací, pak se bude robot otáčet na místě kolem středu nápravy, což také bývá nejčastější referenční bod (Winkler 2005). Místo poháněných kol může být robot vybaven pásy a potom se jedná o robot typu „tank“.



Obr. 6.40. *Robot s diferenciálním řízením. Opěrná kola zde nejsou zobrazena.*

Konfigurační prostor robotu s diferenciálním řízením je $C = R^2 \times (-\pi, \pi\rangle$, konfigurace $q \in C$ je tedy definována jako $q = (x, y, \theta)$, kde x a y jsou souřadnice referenčního bodu a θ je natočení robotu. Prostor akcí je pro experimenty zvolen jako $U = \{-1, 0, 1\}^2$, akce je potom $u = (u_r, u_l)$, kde u_r je úhlová rychlost pravého kola a u_l je úhlová rychlost levého kola. Složky přechodové funkce jsou definovány takto (LaValle 2006):

$$\dot{x} = \frac{r}{2}(u_l + u_r)\cos\theta, \tag{6.10}$$

$$\dot{y} = \frac{r}{2}(u_l + u_r)\sin\theta, \tag{6.11}$$

$$\dot{\theta} = \frac{r}{L}(u_l - u_r), \tag{6.12}$$

kde r je poloměr poháněného kola a L je vzdálenost mezi poháněnými koly. Vzdálenost mezi konfiguracemi je definována podobně jako v práci (Qu 1999):

$$d(p,q)=\sqrt{(x_p-x_q)^2+(y_p-y_q)^2+L^2\min\left\{(\theta_p-\theta_q)^2,(\theta_p-\theta_q+2\pi)^2,(\theta_p-\theta_q-2\pi)^2\right\}}. \quad (6.13)$$

Plánovací algoritmy využívají této vzdálenosti k ohodnocení uzlů grafu. Úspěšné ukončení algoritmu obvykle nastane, pokud je některá z konfigurací podobná konfiguraci cílové. Podobnost konfigurací je možné stanovit pomocí vzdálenosti (6.13). V experimentech však považujeme konfigurace p a q za podobné, pokud platí:

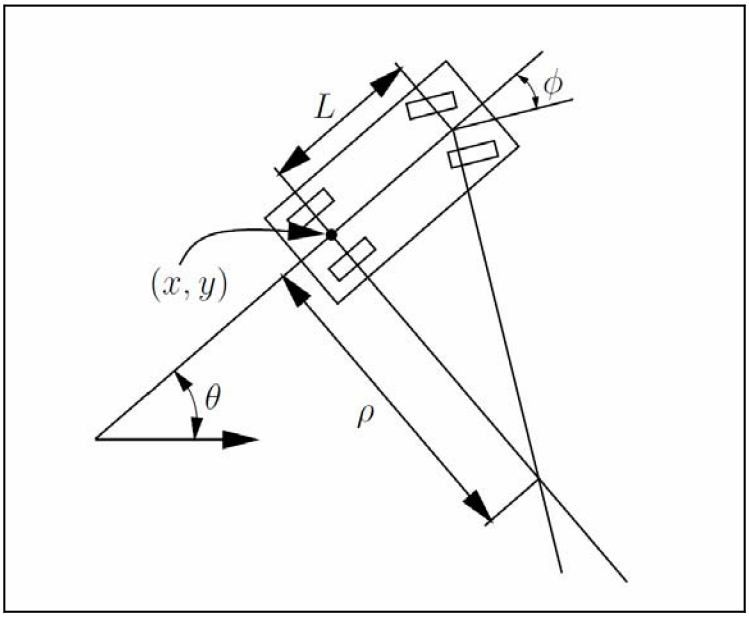
$$\sqrt{(x_p-x_q)^2+(y_p-y_q)^2}<d_{sim}\wedge\min\left\{|\theta_p-\theta_q|,|\theta_p-\theta_q+2\pi|,|\theta_p-\theta_q-2\pi|\right\}<\theta_{sim}, \quad (6.14)$$

kde d_{sim} je maximální vzdálenost mezi referenčními body konfigurací a θ_{sim} je maximální rozdíl natočení konfigurací. Díky zavedení těchto dvou složek můžeme toleranci dosažení cílové konfigurace definovat intuitivněji.

Rozměry testovaného robotu s diferenciálním řízením jsou 30×8, poloměr poháněného kola $r=1$ a délka nápravy $L=8$ délkových jednotek. Náprava je umístěna ve středu robotu. V tomto středu se také nachází referenční bod (x, y).

6.6.2 Robot typu „automobil“

Druhý testovaný model odpovídá robotu typu „automobil“. Tento kolový robot má dvě nápravy. Řiditelná kola jsou však jen na přední nápravě (viz obr. 6.41). Referenční bod (x, y) je u tohoto robotu umístěn do středu zadní nápravy.



Obr. 6.41. *Robot typu „automobil“ (LaValle 2006).*

Konfigurační prostor robotu tohoto typu je stejný jako u robotu s diferenciálním řízením, konfigurace je tedy i zde definována jako $q=(x, y, \theta)$. Akce robotu je definována jako $u=(u_s, u_\phi)$, kde u_s je rychlost robotu a u_ϕ je natočení kol řídicí nápravy. Prostor akcí je pro experimenty zvolen jako $U=\{-1, 1\}\times\left\{-\frac{\pi}{4}+\frac{i}{10}\frac{\pi}{4}\right\}$ pro $i\in\{0, 1, \ldots, 20\}$. Složky přechodové funkce jsou definovány podle (LaValle 2006) takto:

$$\dot{x} = u_s \cos\theta, \tag{6.15}$$

$$\dot{y} = u_s \sin\theta, \tag{6.16}$$

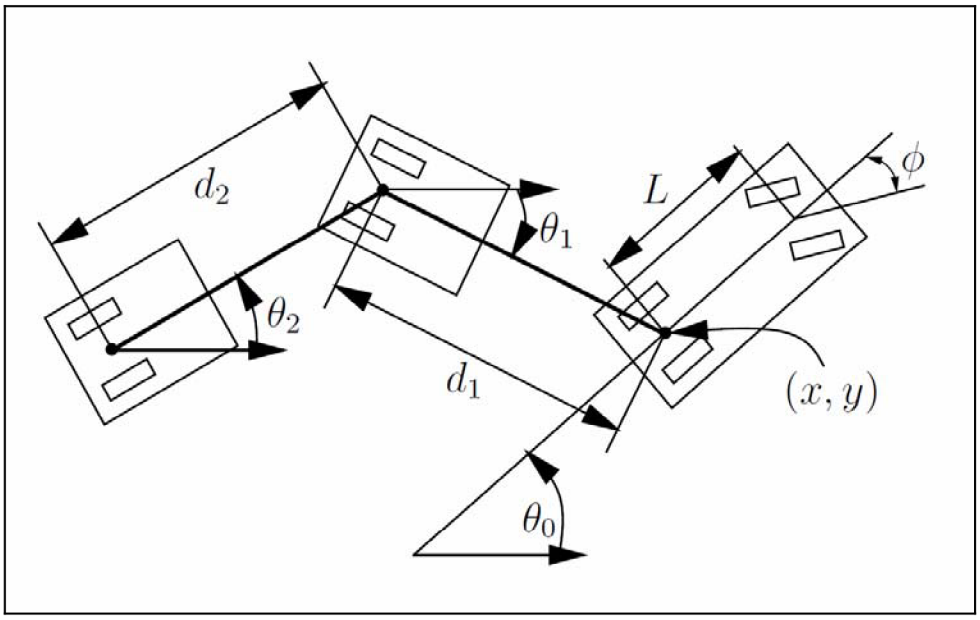
$$\dot{\theta} = \frac{u_s}{L} \tan u_\phi, \tag{6.17}$$

kde L je vzdálenost mezi nápravami. Vzdálenost mezi konfiguracemi i podobnost konfigurací jsou definovány stejně jako u předchozího modelu. Vzdálenost mezi konfiguracemi je tedy dána vztahem (6.13) a pro podobné konfigurace musí platit podmínka (6.14).

V experimentech byl uvažován robot s rozměry 8×14 a vzdáleností náprav $L = 10$ délkových jednotek. Vzdálenost mezi nápravou a okrajem robotu je tedy rovna 2.

6.6.3 Souprava robotu a přívěsů

Poslední model uvažovaný v experimentech popisuje soupravu robotu a dvou přívěsů. Jedná se v podstatě o model typu „automobil“, který je rozšířený o další složky pro přívěsy. Robot a dva tažené přívěsy jsou zakresleny na obr. 6.42.



Obr. 6.42. *Souprava robotu a přívěsů (LaValle 2006).*

Konfigurační prostor robotu se dvěma přívěsy je $C = R^2 \times (-\pi, \pi)^3$, konfigurace $q \in C$ je definována jako $q = (x, y, \theta_0, \theta_1, \theta_2)$, kde x a y jsou souřadnice referenčního bodu, θ_0 je natočení robotu, θ_1 a θ_2 je natočení přívěsů. Akce je definovaná stejně jako pro robot typu automobil $u = (u_s, u_\phi)$. Protože není snadné zhotovit soupravu robotu a přívěsů takovou, která by byla schopna podle plánu také couvat, je z akčního prostoru vypuštěna záporná rychlost. Pro rychlost musí vždy platit $u_s = 1$. Natočení řídicích kol nabývá stejných hodnot jako v předchozím případě. Složky přechodové funkce definuje (LaValle 2006) takto:

$$\dot{x} = u_s \cos\theta, \tag{6.18}$$

$$\dot{y} = u_s \sin\theta, \tag{6.19}$$

$$\dot{\theta}_0 = \frac{u_s}{L} \tan u_\phi, \tag{6.20}$$

$$\dot{\theta}_1 = \frac{u_s}{d_1}\sin(\theta_0 - \theta_1), \tag{6.21}$$

$$\dot{\theta}_2 = \frac{u_s}{d_2}\cos(\theta_0 - \theta_1)\sin(\theta_1 - \theta_2), \tag{6.22}$$

kde d_1 je vzdálenost mezi referenčními body robotu a prvního přívěsu a d_2 je vzdálenost mezi referenčními body přívěsů. Vzdálenost mezi konfiguracemi je opět inspirována prací (Qu 1999):

$$d(p,q) = \sqrt{\begin{aligned}&(x_p - x_q)^2 + (y_p - y_q)^2 \\ &+ L^2 \min\left\{(\theta_{0p} - \theta_{0q})^2, (\theta_{0p} - \theta_{0q} + 2\pi)^2, (\theta_{0p} - \theta_{0q} - 2\pi)^2\right\} \\ &+ L^2 \min\left\{(\theta_{1p} - \theta_{1q})^2, (\theta_{1p} - \theta_{1q} + 2\pi)^2, (\theta_{1p} - \theta_{1q} - 2\pi)^2\right\} \\ &+ L^2 \min\left\{(\theta_{2p} - \theta_{2q})^2, (\theta_{2p} - \theta_{2q} + 2\pi)^2, (\theta_{2p} - \theta_{2q} - 2\pi)^2\right\}\end{aligned}} \tag{6.23}$$

U soupravy robotu a přívěsů považujeme konfigurace p a q za podobné, pokud platí:

$$\begin{aligned}&\sqrt{(x_p - x_q)^2 + (y_p - y_q)^2} < d_{sim} \\ &\wedge \min\left\{|\theta_{0p} - \theta_{0q}|, |\theta_{0p} - \theta_{0q} + 2\pi|, |\theta_{0p} - \theta_{0q} - 2\pi|\right\} < c_0\theta_{sim} \\ &\wedge \min\left\{|\theta_{1p} - \theta_{1q}|, |\theta_{1p} - \theta_{1q} + 2\pi|, |\theta_{1p} - \theta_{1q} - 2\pi|\right\} < c_1\theta_{sim} \\ &\wedge \min\left\{|\theta_{2p} - \theta_{2q}|, |\theta_{2p} - \theta_{2q} + 2\pi|, |\theta_{2p} - \theta_{2q} - 2\pi|\right\} < c_2\theta_{sim}\end{aligned} \tag{6.24}$$

pro $c_i = 1$. Kromě této podobnosti je zavedena ještě jedna méně důležitá podobnost pro $c_i = i + 1$, která je používána dvoustromovým algoritmem transformace pro spojení stromů a pro vyhledávání podcílů.

Pro experimenty byly voleny rozměry robotu 8×14, rozměry obou přívěsů 8×10, vzdálenosti $d_1 = d_2 = 14$ a vzdáleností náprav $L = 10$ délkových jednotek. Vzdálenost mezi nápravou a okrajem robotu i přívěsů je rovna 2.

6.7 Porovnání metod pro neholonomní roboty

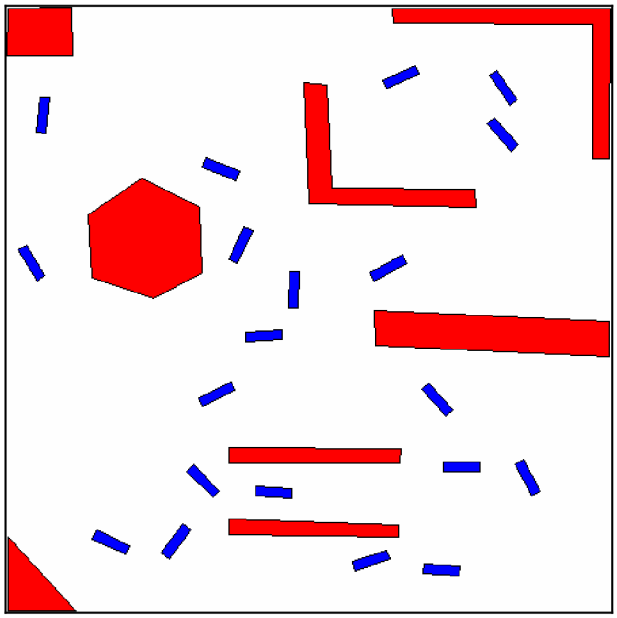
V následujících experimentech je ověřena funkčnost a použitelnost metody genetického algoritmu s následnou transformací, navržené v podkapitole 4.2.4. Tato metoda je implementována jako jednovláknová (GA-TR). Pro spuštění metody ve víceprocesorovém (či vícejádrovém) systému je tato metoda také implementována pro paralelní zpracování jako vícevláknová aplikace (GA-TR-par). V tomto případě zajišťuje generování každého stromu jedno vlákno.

Experimenty byly prováděny ve scéně 500×500, ve které bylo náhodně vygenerováno 20 nekolizních konfigurací tak, aby bylo možné tyto konfigurace dále expandovat a naopak, aby bylo možné těchto konfigurací nějakou cestou dosáhnout (viz obr. 6.43). Hledání cesty bylo spuštěno pro každou uspořádanou dvojici z těchto konfigurací. Dále předkládané výsledky jsou tedy spočtené na základě 380ti hledání.

Nalezené cesty byly hodnoceny podle tří kritérií. Prvním kriteriem je plynulost průchodu cestou, druhým je délka cesty a třetím je doba výpočtu plánování cesty. Pokud

během testování metoda GA-TR či GA-TR-par nenalezla cestu, bylo provedeno opakované spuštění. Výsledná doba výpočtu zahrnuje všechny pokusy hledání.

V experimentech, které byly prováděny pro robot s diferenciálním řízením, bylo provedeno srovnání GA-TR (GA-TR-par) se stávajícími metodami RRT-Z a RRT-R (viz podkapitola 3.4.2). Tyto metody jsou pro tento jednoduchý model rychlé, avšak často u těchto metod docházelo k uváznutí v lokálním minimu. Proto byly metody RRT pro tento model v případě neúspěchu spouštěny opakovaně a výsledná doba výpočtu zahrnuje všechny pokusy hledání. Dále bylo pro tento jednoduchý model provedeno srovnání s obousměrným A* algoritmem. Do stanovené maximální doby výpočtu tento algoritmus cestu nenalezl v 23,2% případů. Vícevláknová verze (A*-par) selhala v 11,6% případů. Nenalezené cesty nebyly pro srovnání uvažovány, výsledky pro A* algoritmus nejsou tedy zcela přesné.



Obr. 6.43. *Množina náhodně vygenerovaných konfigurací v testovací scéně 500×500 pro robot s diferenciálním řízením.*

V experimentech prováděných pro robot typu „automobil“ selhává obousměrný algoritmus A* v 68,2% případů, verze A*-par selhává v 32,4% případů. Algoritmy A* a A*-par tedy nebyly do srovnání vůbec zařazeny a bylo provedeno pouze porovnání metod GA-TR, GA-TR-par, RRT-Z a RRT-R.

U experimentů pro soupravu robotu s přívěsy se projevuje neschopnost metod RRT pracovat se složitějšími modely, které nejsou schopny zpětného pohybu. I když byla pro RRT-Z a RRT-R nastavena trojnásobně větší tolerance cíle, metoda RRT-Z selhává v 45,3% případů a RRT-R selhává v 49,5% případů. Obousměrný A* zde selhává v 68,4% případů. Pro robot s přívěsy bylo tedy provedeno pouze srovnání navržených metod GA-TR a GA-TR-par.

Nastavení parametrů testovaných metod je patrné z tabulek 6.9, 6.10 a 6.11. U všech metod byla stanovena doba působení akce na 5 časových jednotek. Všechny úhlové rozměry jsou uváděny v radiánech a vzdálenosti jsou uváděny v jednotkách délky. Parametry genetického algoritmu byly stanoveny na základě výsledků experimentů provedených v podkapitole 6.4.2.

Tab. 6.9. *Parametry metod GA-TR a GA-TR-par.*

	robot s dif. řízením	robot typu automobil	robot s přívěsy
pravděpodobnost op. křížení	0,75		
pravděpodobnost op. velké mutace	0,58		
pravděpodobnost op. malé mutace	0,37		
pravděpodobnost op. opravy	0,27		
pravděpodobnost op. výměny	0,4		
pravděpodobnost op. vyhlazení	0,26		
pravděpodobnost op. úhlu startu a cíle	0,79		
pravděpodobnost op. odstranění	0,76		
pravděpodobnost op. viditelnosti	0,76		
počet chromozomů (celkem, při selekci)	50; 2		
počáteční velikost chromozomu	<1, 5>		
váhy w_1, w_2, w_3	0,2; 0,2; 0,6		
$\beta_{StartMax}$	1	2,2	2,8
$\beta_{GoalMax}$	1	2,2	2,8
β_{SegMax}	1	1,4	1,6
d_{Min}	30	30	60
d_{Max}	150	300	300
$d_{SubGoal}$	15	15	30
d_{Sim}	5		
θ_{Sim}	0,3		
d_{Sim} pro vkládání do seznamů *OBST*	4,5 (0,01 při uplatnění β_{Near})		
θ_{Sim} pro vkládání do seznamů *OBST*	0,3 (0,05 při uplatnění β_{Near})		
d_{Sim} pro spojení stromů	4		
θ_{Sim} pro spojení stromů	0,3		
β_{Near}	0,35		
maximální doba výpočtu [s]	15 + 20 n, kde n je počet segmentů cesty		

Tab. 6.10. *Parametry algoritmů A* a A*-par.*

	robot s dif. řízením	robot typu automobil	robot s přívěsy
d_{Sim}	5		
θ_{Sim}	0,3		
d_{Sim} pro vkládání do seznamů *OBST*	4,5		
θ_{Sim} pro vkládání do seznamů *OBST*	0,3		
d_{Sim} pro spojení stromů	4		
θ_{Sim} pro spojení stromů	0,3		
maximální doba výpočtu [s]	60	60	120

Tab. 6.11. *Parametry metod RRT-Z a RRT-R*

	robot s dif. řízením	robot typu automobil	robot s přívěsy
PG	0,05		
PG2 (pro RRT-R)	0,5		
k (pro RRT-R)	1		
d_{Sim}	5		15
θ_{Sim}	0,3		0,9
maximální doba výpočtu RRT-Z [s]	10	120	
maximální doba výpočtu RRT-R [s]	20	120	

Experimenty byly prováděny na stroji PC s dvou jádrovým procesorem Intel Core 2 Duo pracujícím na kmitočtu 3 GHz a s 2 GB paměti. Aplikaci však pro běh postačuje 100 MB volné paměti.

6.7.1 Porovnání plynulosti průchodů cestou

Plynulost průchodu cestou P je závislá na hodnotě součtu odlišností vždy po sobě následujících akcí cesty P:

$$K(P) = \sum_{i=1}^{n-1} da(u_i, u_{i+1}), \tag{6.25}$$

kde $da(u_i, u_{i+1})$ je odlišnost akcí u_i a u_{i+1} a n je počet akcí cesty P. Odlišnost akcí $u = (u_r, u_l)$ a $v = (v_r, v_l)$ definujme jako

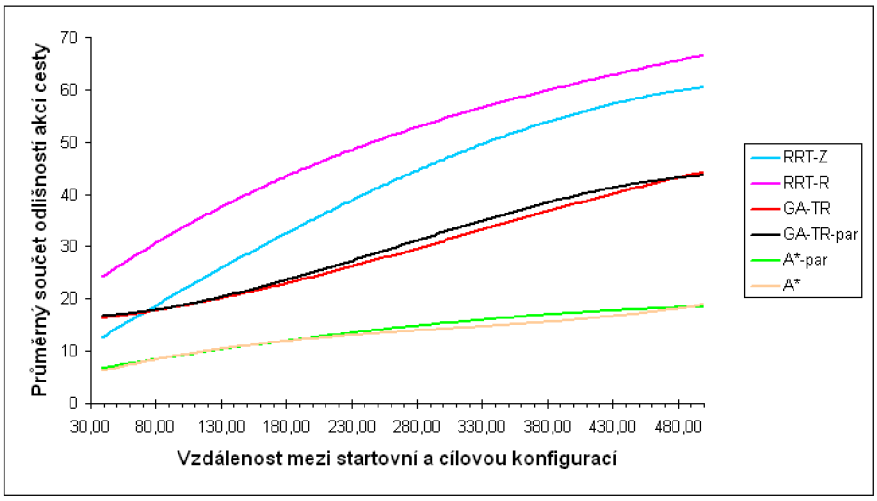
$$da(u,v) = |u_r - v_r| + |u_l - v_l|. \tag{6.26}$$

Odlišnost akcí $u = (u_s, u_\phi)$ a $v = (v_s, v_\phi)$ definujme jako

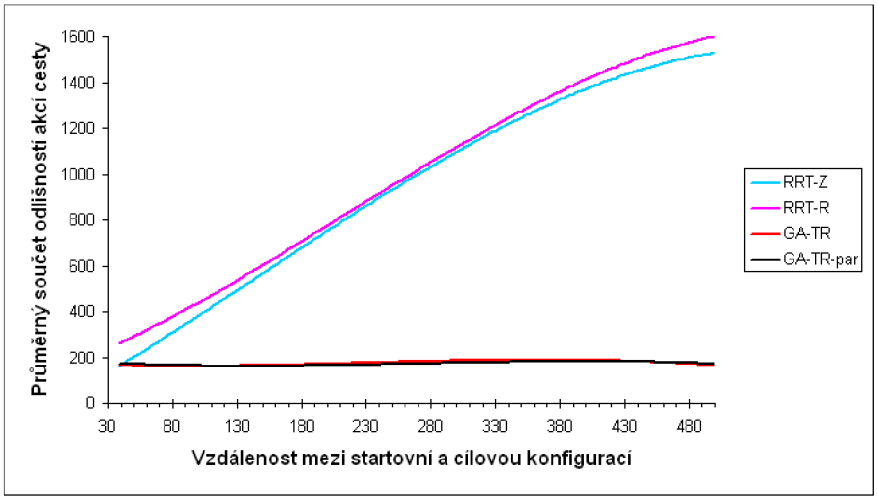
$$da(u,v) = \begin{cases} 21 & \text{pokud } u_s \neq v_s \\ \frac{40}{\pi}|u_\phi - v_\phi| & \text{jinak} \end{cases}. \tag{6.27}$$

Jak je patrné, čím je součet odlišností mezi akcemi větší, tím bude průchod cestou méně plynulý.

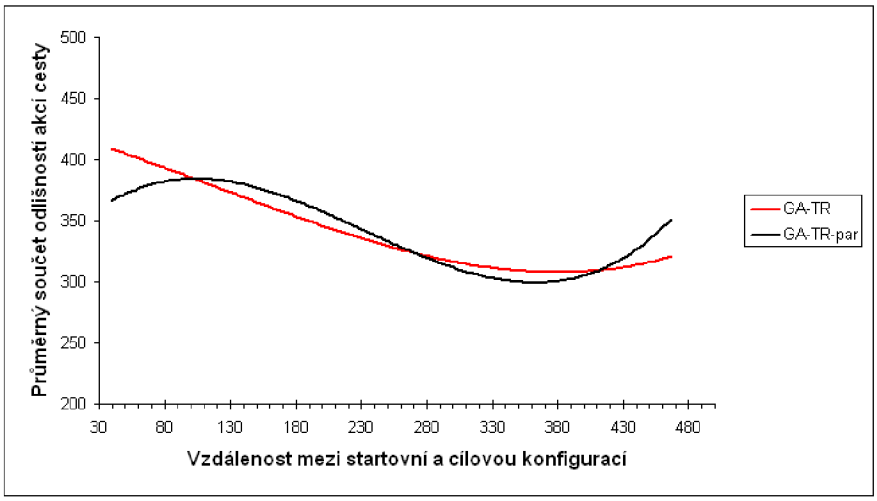
Výsledky těchto experimentů jsou zobrazeny na obrázcích 6.44, 6.45 a 6.46, postupně pro všechny modely. Jak lze vidět, nejlepší průchod poskytují cesty nalezené metodou GA-TR (nebo její paralelní verzí). Největší rozdíl je patrný u modelu robotu typu automobil, kde je při nejdelších vzdálenostech mezi konfiguracemi startu a cíle dosaženo až sedmkrát lepší průchodnosti než poskytují metody RRT. U robotu s diferenciálním řízením dosahuje lepší průchodnosti algoritmus A*, avšak tento algoritmus v některých případech selhal (viz předcházející podkapitola). Vzrůstající průchodnost je způsobena vzrůstající délkou nalezené cesty. Horší průchodnost lze pozorovat i u robotu s přívěsy pro krátké vzdálenosti mezi startem a cílem. Toto je způsobeno cestami mezi některými blízkými konfiguracemi tohoto robotu, u kterých nelze při transformaci použít strategii pohybu po přímce (uplatnění β_{Near}).



Obr. 6.44. *Závislosti průměrného součtu odlišností akcí K(P) nalezené cesty P na vzdálenosti mezi startovní a cílovou konfigurací robotu s diferenciálním řízením.*



Obr. 6.45. *Závislosti průměrného součtu odlišností akcí K(P) nalezené cesty P na vzdálenosti mezi startovní a cílovou konfigurací robotu typu „automobil“.*

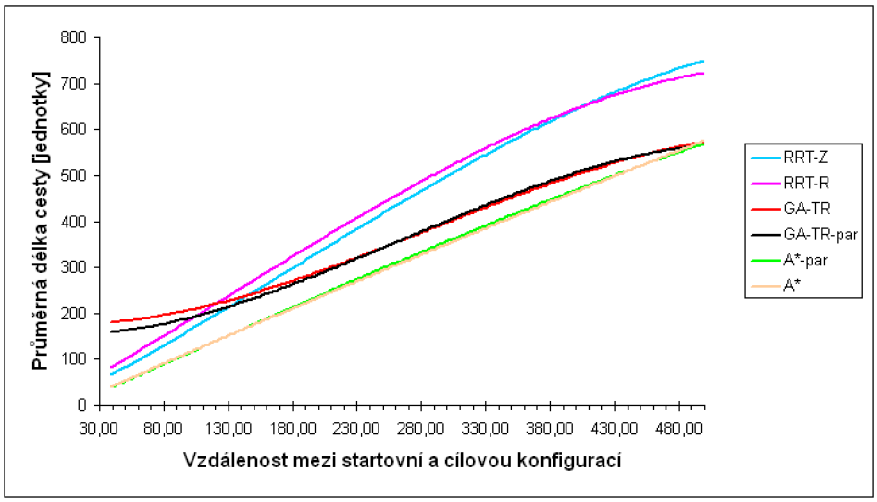


Obr. 6.46. *Závislosti průměrného součtu odlišností akcí K(P) nalezené cesty P na vzdálenosti mezi startovní a cílovou konfigurací robotu s přívěsy.*

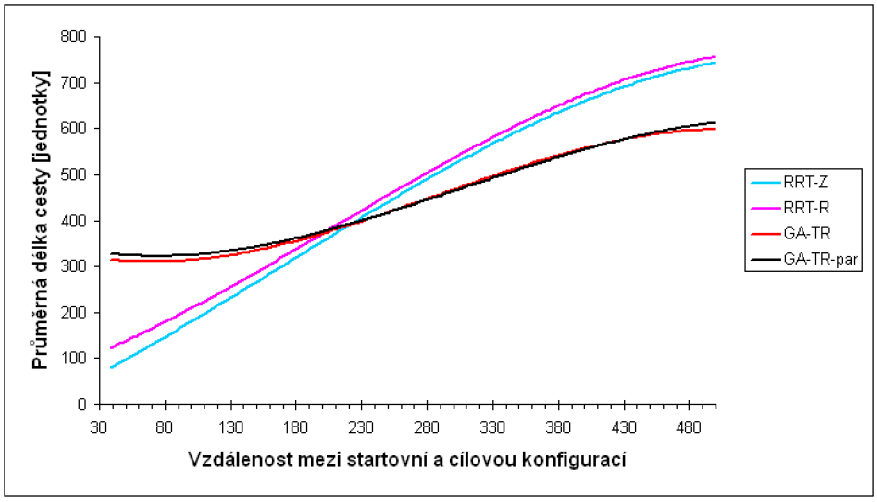
6.7.2 Porovnání délek nalezených cest

Délka nalezené cesty odpovídá délce křivky, kterou zanechá referenční bod robotu po průchodu cestou. Výsledky experimentů jsou patrné z obrázků 6.47, 6.48 a 6.49. Pro robot s diferenciálním řízení a vzdálenosti konfigurací menší než 130 jednotek, poskytují nejkratší cesty metody RRT. Pro robotu typu „automobil“ podává RRT kratší cesty do vzdálenosti přibližně 200 jednotek. Uvážíme-li však průchodnost cesty (viz předchozí podkapitola), pak můžeme tvrdit, že kromě robotu s diferenciálním řízením a vzdálenosti konfigurací menší než 80 jednotek, nejlepší řešení poskytuje metoda GA-TR nebo její paralelní verze GA-TR-par. Metoda A* je zde uvedena opět jen pro demonstraci toho, že pokud je cesta tímto algoritmem nalezena, není výrazně lepší než cesta nalezená metodou GA-TR.

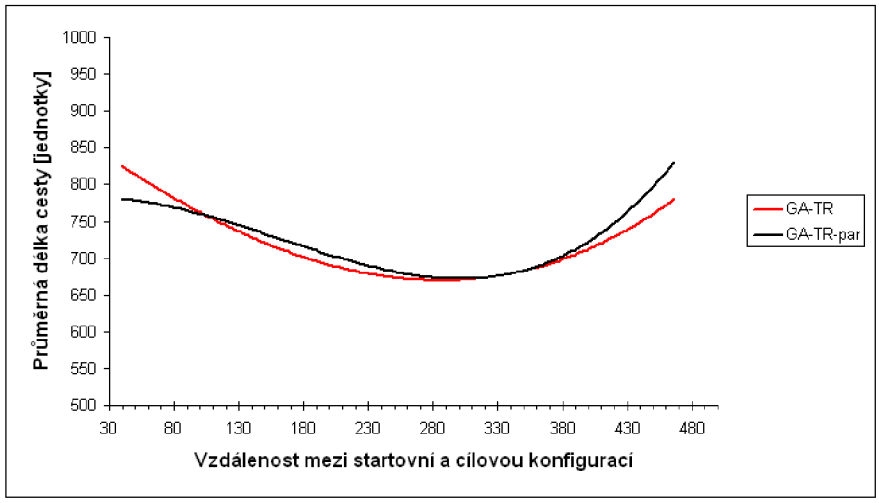
Vzrůstající délka cesty je způsobena vzrůstající vzdáleností mezi konfiguracemi startu a cíle. Velká délka cesty pro blízké konfigurace robotu s přívěsy je způsobena tím, že se tento robot nemůže pohybovat vzad a tudíž i pro dosažení některých blízkých konfigurací musí absolvovat dlouhé cesty.



Obr. 6.47. *Závislosti průměrné délky nalezené cesty na vzdálenosti mezi startovní a cílovou konfigurací robotu s diferenciálním řízením.*



Obr. 6.48. *Závislosti průměrné délky nalezené cesty na vzdálenosti mezi startovní a cílovou konfigurací robotu typu „automobil“.*

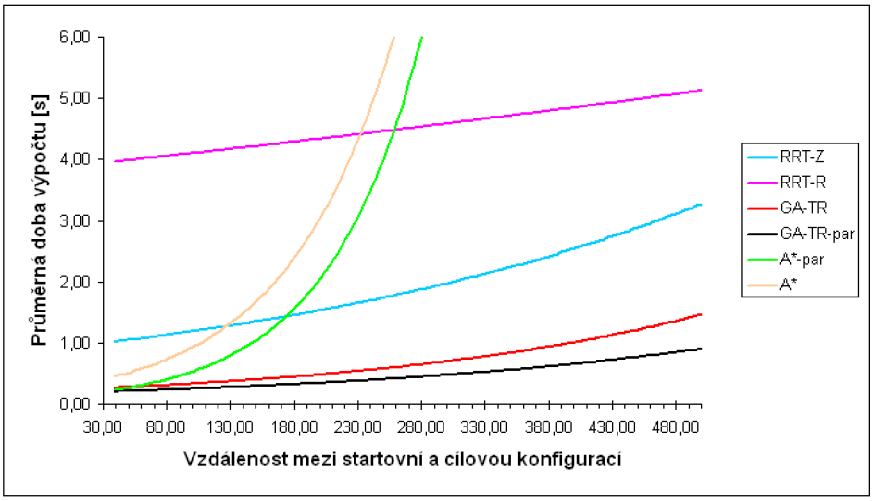


Obr. 6.49. *Závislosti průměrné délky nalezené cesty na vzdálenosti mezi startovní a cílovou konfigurací robotu s přívěsy.*

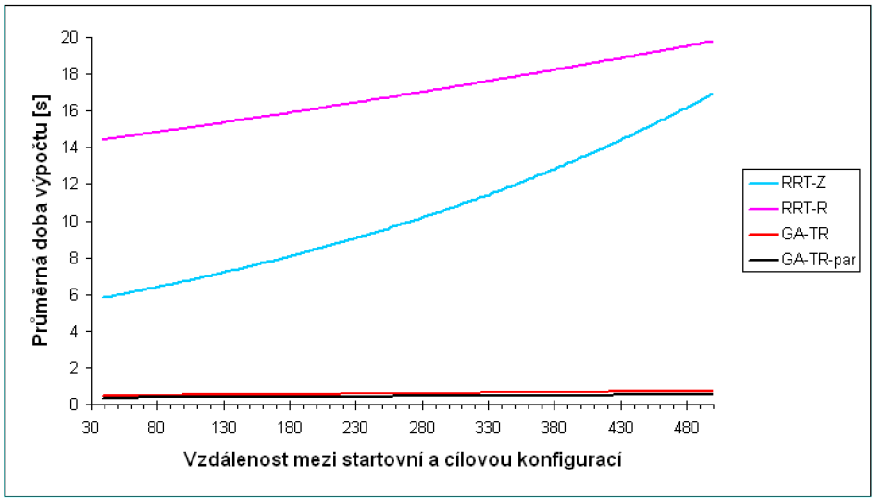
6.7.3 Porovnání dob výpočtu

Pro každé hledání cesty byla měřena doba výpočtu. Průměrné doby výpočtů jsou zakresleny na obrázcích 6.50, 6.51 a 6.52. Opět je zřejmé, že navržená metoda GA-TR (případně GA-TR-par) poskytuje nejrychlejší výpočet pro všechny typy robotů.

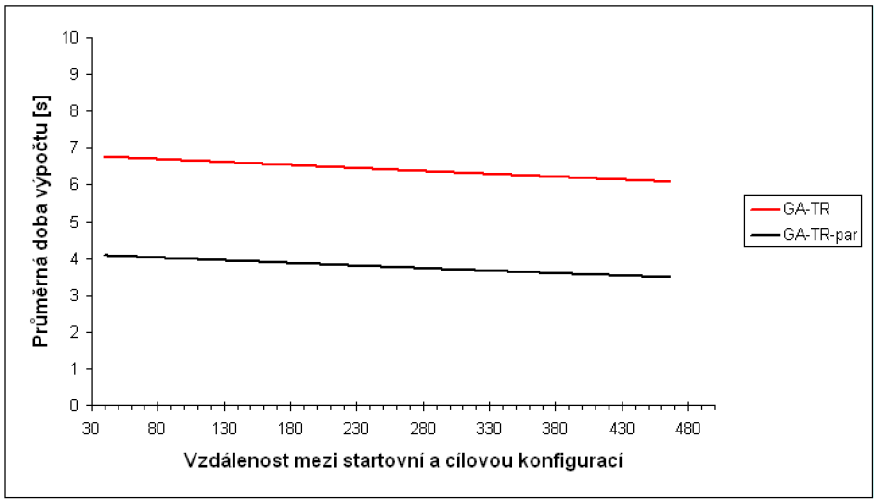
V případě robotu s přívěsy, pro které je generování stromů při transformaci již časově náročné, lze na obr. 6.52 pozorovat výhody paralelního zpracování, které zde snižuje čas výpočtu téměř o polovinu. Klesající tendenci doby výpočtu u experimentu na obr. 6.52 je možné vysvětlit tím, že při větších vzdálenostech mezi konfiguracemi startu a cíle je při transformaci uplatňována vícekrát strategie pohybu po přímce. Výpočet zatáček je pro tento model časově náročnější než u jiných modelů a tak strategie pohybu po přímce v podstatě snižuje průměrnou dobu výpočtu.



Obr. 6.50. *Závislosti průměrné doby výpočtu hledání cesty na vzdálenosti mezi startovní a cílovou konfigurací robotu s diferenciálním řízením.*



Obr. 6.51. *Závislosti průměrné doby výpočtu hledání cesty na vzdálenosti mezi startovní a cílovou konfigurací robotu typu „automobil“.*



Obr. 6.52. *Závislosti průměrné doby výpočtu hledání cesty na vzdálenosti mezi startovní a cílovou konfigurací robotu s přívěsy.*

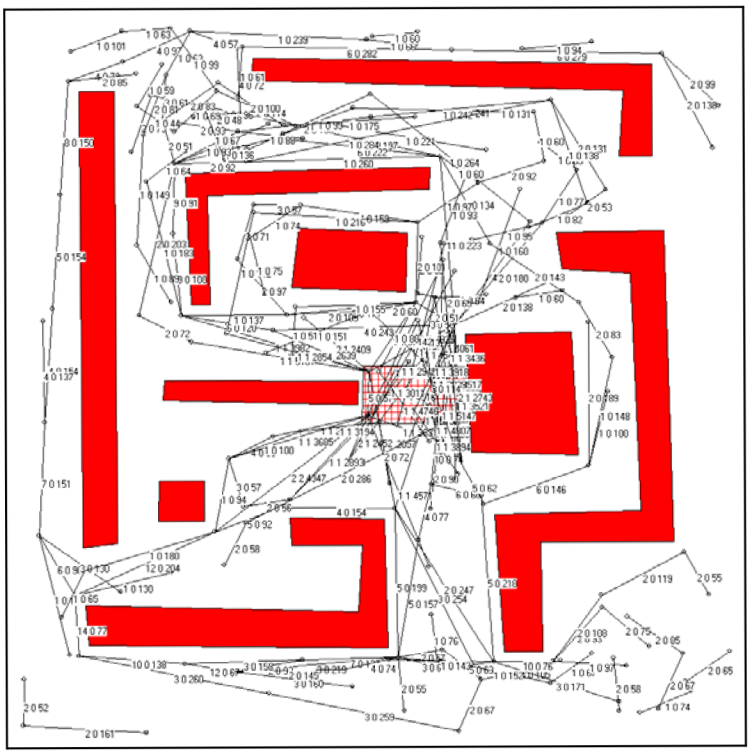
6.8 Experimenty s učením případového grafu

Pro testování systému případového usuzování v kombinaci s genetickým algoritmem (viz podkapitola 4.2.6) byl vybrán nejsložitější model robotu s přívěsy. Tento model požaduje oproti jednodušším modelům dosti vysoké nastavení hodnot $\beta_{StartMax}$, $\beta_{GoalMax}$, β_{SegMax}, d_{Min} a $d_{SubGoal}$ (viz. 6.9) a klade tak na vyhledávání cesty v případovém grafu největší omezení. Pro jednodušší modely podává tento navržený systém lepší výsledky.

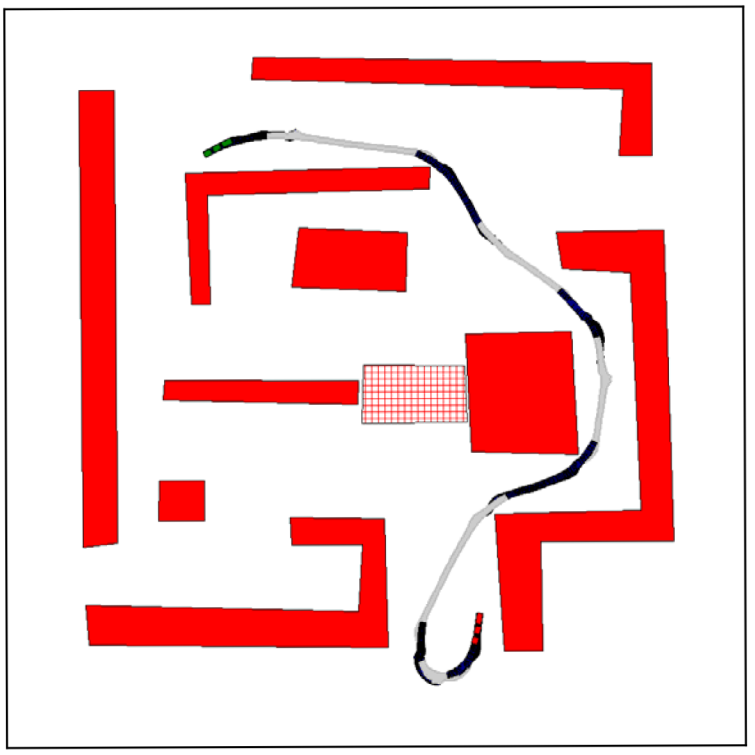
Pro účely experimentování s případovým grafem bylo náhodně vygenerováno 274 párů konfigurací. První konfigurace v páru představuje start a druhá konfigurace cíl. Podobně jako v předchozí podkapitole, byly uvažovány jen takové konfigurace, které bylo možné dále expandovat (startovní konfigurace) nebo inverzně expandovat (cílové konfigurace). Experimenty probíhaly ve scéně 1000×1000 jednotek (viz obr. 6.53). Experimenty začínaly vždy s prázdnou případovou bází a k učení případů tak docházelo v průběhu experimentu.

Experimenty v podkapitole 6.8.1 srovnávají cesty nalezené metodou GA-TR-par s cestami nalezenými pomocí systému případového usuzování s následnou transformací

v paralelní verzi (CG-TR-par). Protože je metoda případového usuzování určena především pro použití v částečně známých prostředích, experimenty v podkapitole 6.8.2 jsou prováděny ve scéně s neznámou překážkou. Příklad neznámé překážky a cesty, která byla plánována s ohledem na tuto překážku, je na obrázku 6.54.



Obr. 6.53. *Příklad případového grafu pro robot s přívěsy ve scéně 1000×1000*



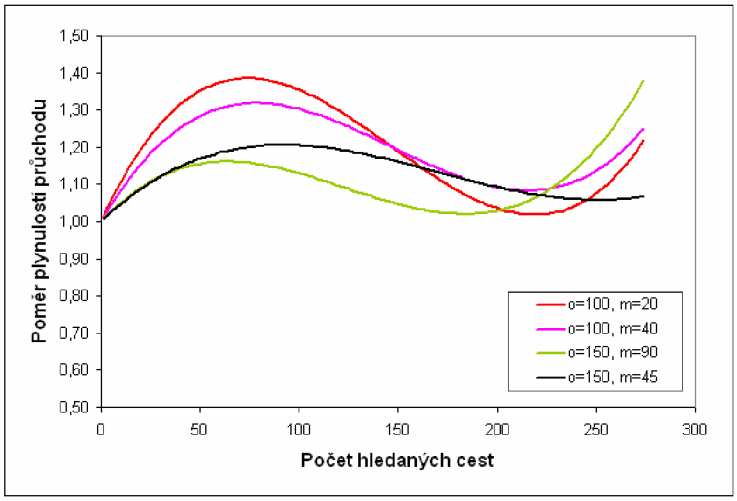
Obr. 6.54. *Příklad cesty nalezené pomocí případového grafu na obr. 6.53 ve scéně s neznámou překážkou*

Parametry metody GA-TR-par jsou v těchto experimentech nastaveny na stejné hodnoty jako v experimentech předcházejících (viz tab. 6.9). Pro nastavení parametrů případového usuzování ($\beta_{StartMax}$, $\beta_{GoalMax}$, β_{SegMax}, d_{Min}, $d_{SubGoal}$) a nastavení parametrů následné transformace platí taktéž hodnoty uvedené v tabulce 6.9. Nastavení počtu bodů m v okolí startu a cíle a velikost tohoto okolí o je uvedena dále u příslušného experimentu. Po každém přidání (či aktualizaci) případu do případové báze bylo provedeno odstranění případů starších dvaceti hledání, pro které platí $n_1 = 1$ a $n_2 = 0$. Maximální hodnota míry blízkosti podobných případů byla stanovena na $d_{proximity} = 5$. V experimentech bylo předpokládáno, že $t_{MaxW} = \infty$. Odstraňování případů při překročení maximální velikosti případové báze nebylo uvažováno, neboť zmíněné odstraňování starých případů zvládalo v průběhu experimentů udržovat velikost případové báze v přípustných mezích (do 300 případů).

Experimenty byly prováděny na stejném PC jako v předcházející podkapitole.

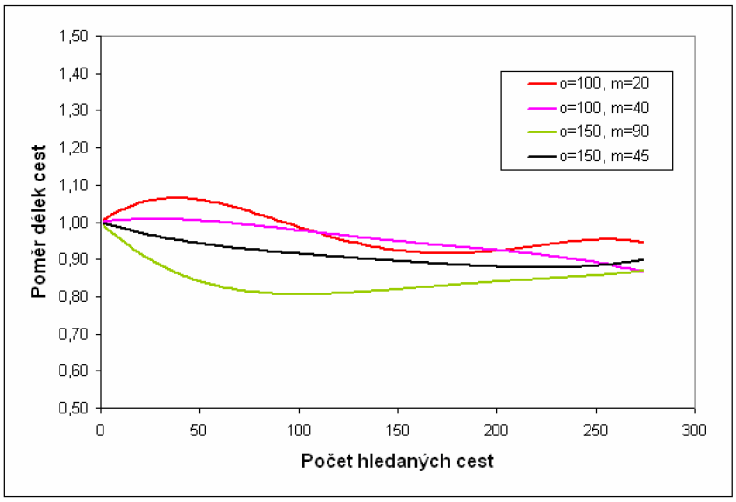
6.8.1 Porovnání s metodou GA-TR-par

V těchto experimentech byly pro každý pár náhodně vygenerovaných konfigurací spouštěny metody CG-TR-par a GA-TR-par. Stejně jako v podkapitole 6.7, byly nalezené cesty hodnoceny podle plynulosti průchodu cestou, délky cesty a doby výpočtu metody. Experimenty byly prováděny pro různé velikosti okolí o a počty generovaných bodů v tomto okolí m. Získané údaje pro metodu CG-TR-par byly dány do poměru s údaji metody GA-TR-par a po proložení polynomy zaneseny do grafů na obrázcích 6.55, 6.56 a 6.57.



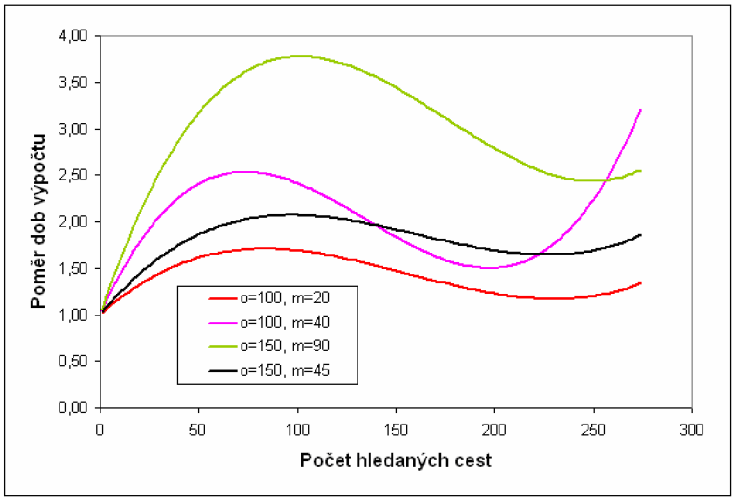
Obr. 6.55. *Závislosti poměru průměrných součtů odlišností akcí nalezených cest na počtu hledaných cest. Poměr je tvořen výsledky metod CG-TR-par ku GA-TR-par.*

Z obrázku 6.55 je zřejmé, že použití metody CG-TR-par oproti metodě GA-TR-par zhoršuje průchodnost cestou pro všechny velikosti okolí. Cesta nalezená v případovém grafu obsahuje obvykle více kratších segmentů než cesta nalezená GA a u následné transformace se tím tedy snižuje možnost využívat strategii pohybu po přímce. Proto dochází ke zhoršení průchodnosti přibližně od dvoustého hledání, kdy už začínají být takovéto cesty případovým grafem nalézány. Ke zhoršení plynulosti průchodu v počátcích učení dochází z toho důvodu, že cesty nalezené v nenaučeném případovém grafu jsou často delší než cesty nalezené GA anebo dochází ke stejnému efektu jako u naučeného případového grafu díky téměř neomezenému vkládání dočasných hran v této fázi učení.



Obr. 6.56. *Závislosti poměru délek nalezených cest na počtu hledaných cest. Poměr je tvořen výsledky metod CG-TR-par ku GA-TR-par.*

Délky nalezených cest jsou použitím metody CG-TR-par spíše zlepšovány (viz obr. 6.56). Pro okolí $o = 150$ je zřejmé, že i když systém není ještě naučený (dosud proběhlo málo hledání cest), jsou nalézány kratší cesty. Toto způsobují dočasné hrany v grafu, jejichž vkládání není do dosud nenaučeného případového grafu blokováno naučenými případy. Pro menší velikosti okolí jsou délky cest nejprve nepatrně delší, s učením nových případů však délky cest klesají.



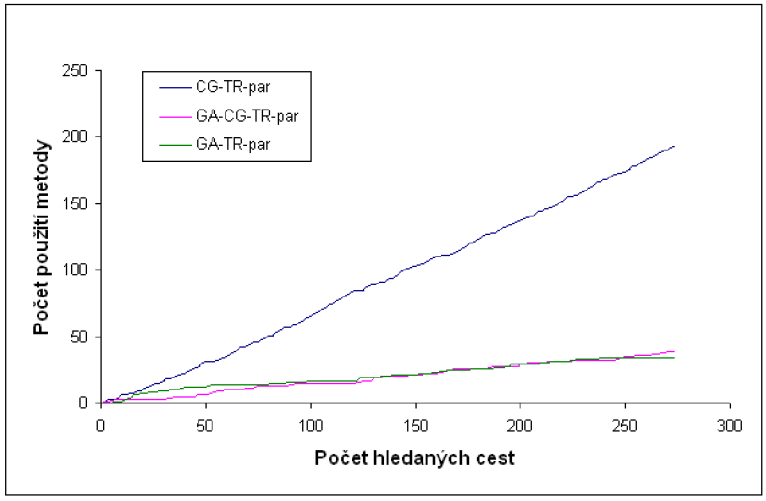
Obr. 6.57. *Závislosti poměru dob výpočtů nalezených cest na počtu hledaných cest. Poměr je tvořen výsledky metod CG-TR-par ku GA-TR-par.*

Průběhy dob výpočtů jsou podobné průběhům průchodů cestou. Dobu výpočtu metody CG-TR-par nejvíce prodlužuje právě transformace krátkých segmentů cesty. Na obr. 6.57 je možné pozorovat, že nejdelší doba výpočtu metody CG-TR-par nastává pro velikost okolí $o = 150$ a počet bodů v okolí $m = 90$. Doba výpočtu je v tomto případě navíc prodlužována detekcí kolizí dočasných hran mezi velkým počtem bodů v okolí.

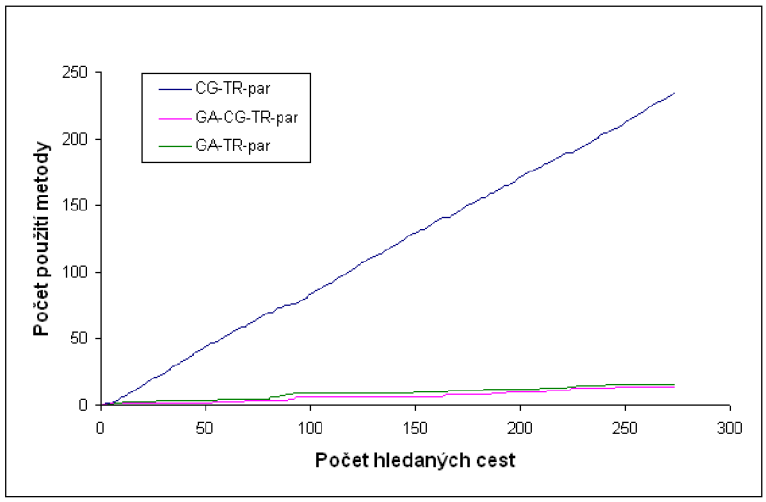
Na základě těchto experimentů je možné se domnívat, že použití navržené metody CG-TR-par nepřináší významné zlepšení plánování cesty. Pokud je ovšem případový graf udržován v aktuálním stavu a tedy se v něm odráží dynamické změny prostředí, pak je použití metody CG-TR-par již přínosné (viz následující podkapitola).

6.8.2 Plánování cesty v částečně známém prostředí

Pro správné fungování metody CG-TR-par v částečně známém prostředí je zapotřebí, aby vždy po průjezdu robotu nalezenou cestou byly důsledně aktualizovány případy v případové bázi podle informaci získaných při skutečném průjezdu cestou. Dále je nutné, aby byla cesta plánována především pomocí případového grafu a nespoléhalo se příliš na pomocnou metodu případového usuzování. Navržený systém toto splňuje. Průběhy počtů použití metod při plánování cesty jsou pro testované velikosti okolí $o = 100$, $m = 40$ a $o = 150$, $m = 45$ dosti podobné průběhům na obrázku 6.58 pro okolí $o = 100$ a $m = 20$. Vyššího využití případového grafu je dosaženo jen pro $o = 150$, $m = 90$ (viz obr. 6.59). Metoda, která cestu nalezne pomocí GA a následně upraví podle případového grafu je v těchto experimentech označena jako GA-CG-TR-par.



Obr. 6.58. *Počty použití metod v závislosti na počtu hledaných cest pro o=100 a m=20*



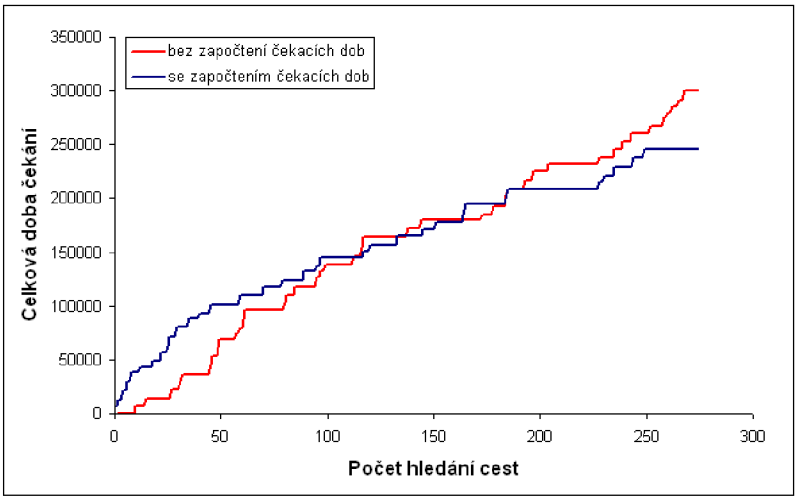
Obr. 6.59. *Počty použití metod v závislosti na počtu hledaných cest pro o=150 a m=90*

Pro experimenty v této podkapitole byla do scény vložena neznámá překážka o velikosti přibližně 50×50 jednotek. Pravděpodobnost výskytu této překážky byla stanovena na hodnotu 0,8. Minimální doba výskytu této překážky byla stanovena na 5000 časových jednotek a maximální doba výskytu na 10000 časových jednotek. Hledání cest bylo spuštěno pro stejnou

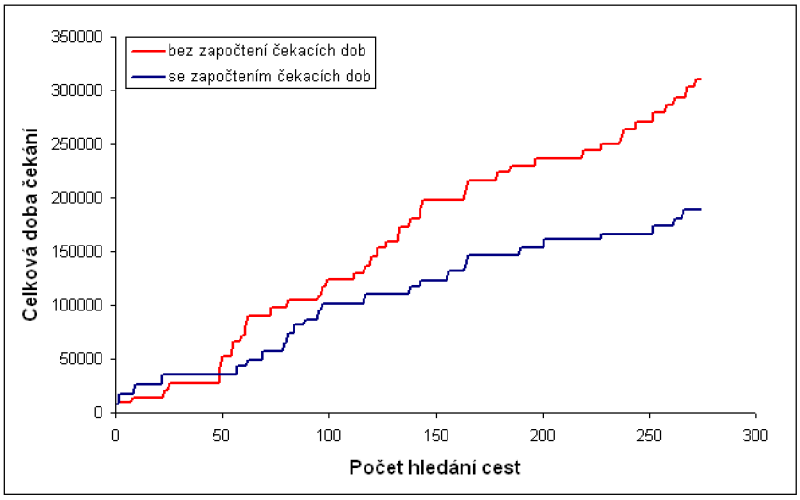
testovací množinu konfigurací jako v předcházející podkapitole. Pro tuto testovací množinu bylo prováděno hledání pomocí CG-TR-par dvakrát. V první sérii testů bylo uvažováno ohodnocení případů podle navrženého vztahu 4.33, v druhé sérii testů nebyla při přidávání (aktualizaci) případů uvažována doba čekání t_W.

Z experimentů plyne, že při vhodném nastavení přináší použití metody CG-TR-par zřetelné snížení celkové doby průchodu robotu prostředím s neznámou překážkou (viz obr. 6.61). Jako vhodná velikost okolí se jeví *o=100* při počtu generovaných bodů *m=40*. Jak dokazují experimenty v předcházející kapitole, při tomto nastavení také dochází oproti metodě GA-TR-par ke snížení délky nalezené cesty.

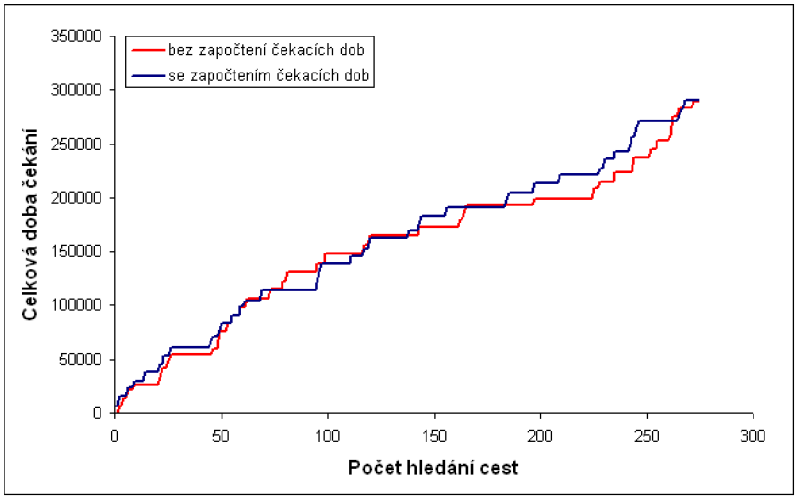
Nastavení velikosti okolí a počtu generovaných bodů je velmi důležité. Při menších počtech generovaných bodů v okolí nedochází k napojení startu nebo cíle na případový graf všemi možnými způsoby a často tak není možné nalézt cestu grafem bez průchodu přes hrany s čekáním (viz obr. 6.60). Naopak při větší velikosti okolí dochází ke generování dočasných hran přes neznámou překážku. Protože jsou dočasné hrany ohodnoceny pouze odhadem doby průchodu hranou bez čekání, dochází tak opět ke zvýšení čekacích dob (viz obr. 6.62).



Obr. 6.60. *Celková doba čekání v závislosti na počtu hledaných cest pro o=100 a m=20*



Obr. 6.61. *Celková doba čekání v závislosti na počtu hledaných cest pro o=100 a m=40*



Obr. 6.62. *Celková doba čekání v závislosti na počtu hledaných cest pro o=150 a m=45*

7. Závěr

Tato disertační práce zkoumá použití metod strojového učení pro plánování cesty autonomního lokomočního robotu. V souvislosti s plánováním cesty se z metod strojového učení uplatňují především případové usuzování, neuronové sítě, posilované učení, rojová inteligence a genetické algoritmy.

Prvá část práce je přehledem řady stávajících metod pro plánování cesty holonomních i neholonomních robotů. Ve druhé části práce je navržena řada originálních metod pro plánování cesty holonomního i neholonomního robotu, které jsou založeny především na případovém usuzování a genetických algoritmech. Třetí část práce je věnována popisu implementace těchto metod a čtvrtá část rozebírá výsledky experimentů s navrženými metodami.

Návrhy metod pro plánování cesty na mřížce jsou výsledkem společného výzkumu vedoucího práce a autora práce. Základní principy představených algoritmů těchto metod jsou dílem především vedoucího práce a veškeré implementace a experimentální ověření provedl autor práce. Návrhy metod pro plánování cesty ve spojitém prostoru jsou kromě základního návrhu genetického algoritmu s některými specifickými operátory a matematického popisu upraveného Dijkstrova algoritmu již původní prací autora, stejně jako implementace a experimentální ověření těchto metod.

Experimentální ověření bylo provedeno postupně se všemi navrženými metodami. V oblasti plánování cesty na mřížce byly srovnány dvě stávající metody plánování cesty holonomních robotů (metoda založená jen na Dijkstrovu algoritmu a metoda založená jen na A*) s metodami navrženými (metoda kombinující A* s normálním případovým grafem a metoda kombinující A* s „jemnou strukturou“ případového grafu). Z těchto experimentů plyne, že po naplnění báze případů pracují obě navržené metody rychleji než algoritmus A* pro všechny testované velikosti okolí.

Z experimentálního ověření metody kombinující případový graf a pravděpodobnostní stromy vyplývá, že při vhodné volbě okolí a případové bázi dostatečně pokrývající svými případy scénu lze dosáhnout vůči samostatnému používání RRT jak zrychlení doby výpočtu, tak i vyhledání kratších cest.

V dalších experimentech věnovaných navrženému genetickému algoritmu byla zkoumána možnost použití části naučené populace předchozího řešení v počáteční populaci nového problému a možnost použití konečné populace současného řešení pro přeplánování cesty z důvodu výskytu neznámé překážky. Z těchto experimentů je patrné, že pro testovanou scénu a holonomní robot je nejlepších výsledků dosaženo, když celá stará populace je použita jako počáteční populace nového problému.

V práci je také ověřena možnost nastavování parametrů navrženého genetického algoritmu dalším genetickým algoritmem. Toto je provedeno pro obě varianty navrženého algoritmu (pro holonomní i neholonomní robot). Klesající průběh hodnot fitness funkce v průběhu výpočtu dokazuje, že navržený způsob optimalizace je účelný.

V oblasti plánování cesty neholonomních robotů byly srovnávány navržené metody s metodami RRT a s obousměrným A* algoritmem. Experimenty byly prováděny se třemi různými modely neholonomních robotů. Ze srovnání metod plyne, že navržená metoda kombinující genetický algoritmus s následnou transformací cesty poskytuje v mnoha případech kvalitnější cesty a to i při kratší době výpočtu. Pro složitější modely robotů je navržená metoda dokonce jediná použitelná, protože ostatní implementované metody zde často selhávají.

Závěrečné experimenty byly prováděny v simulovaném částečně známém prostředí. Pomocí navrženého systému, který je založen na případovém usuzování, bylo dosaženo zřetelného snížení průměrné doby průchodu robotu prostředím.

Navržený systém plánování cesty neholonomního robotu je implementován tak, aby nebyl vázán na jeden kinematický model robotu. Díky použitému objektovému přístupu je možné další modely robotů do systému snadno implementovat. Podobně není navržený systém vázán na simulační prostředí a je možné jeho použití v reálném robotu.

8. Literatura

AAMODT, A.; PLAZA, E. 1994. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, vol. 7, no. 5, pp. 39-59.

AGIRREBEITIA, J.; AVILÉS, R.; de BUSTOS, I. F.; AJURIA, G. 2005. A New APF Strategy for Path Planning in Environments with Obstacles. *Mechanism and Machine Theory*, vol. 40, issue 6, pp. 645-658.

ARLEO, A.; GERSTNER, W. 2000. Spatial Cognition and Neuro-Mimetic Navigation: a Model of Hippocampal Place Cell Activity. *Biological Cybernetics*, vol. 83, pp. 287-299.

BARRAQUAND, J.; LATOMBE, J.-C. 1993. Nonholonomic Multibody Mobile Robots: Controllability and Motion Planning in the Presence of Obstacles. *Algorithmica*, vol. 10, pp. 121-155.

BARRIOS-ARANIBAR, D.; ALSINA, P. J. 2004. Reinforcement Learning-Based Path Planning for Autonomous Robots. In: *Jornada de Robótica Inteligente - XXIV Congresso da Sociedade Brasileira de Computação - SBC 2004*, Salvador. *Anais do XXIV Congresso da Sociedade Brasileira de Computação - SBC 2004*, pp. 1802-1811.

BERLINER, H. 1979. The B* tree search algorithm: A best-first proof procedure. *Artifical Intelligence*, vol. 12, issue 1, pp. 23-40.

BROOKS, R. A. 1985. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14-23.

BUGMANN, G.; TAYLOR, J. G.; DENHAM, M. 1995. Route Finding by Neural Nets. *Neural networks*, J. G. Taylor (Ed.), Alfred Waller Ltd., pp. 217-230.

BURCHARDT, H.; SALOMON, R. 2006. Implementation of Path Planning Using Genetic Algorithms on Mobile Robots. In *Proceedings of the IEEE World Congress on Computational Intelligence (WCCI 2006), Congress on Evolutionary Computation (CEC 2006)*, Vancouver, Canada, pp. 1831-1836.

DIVELBISS A. W.; WEN, J. T. 1997. A Path Space Approach to Nonholonomic Motion Planning in the Presence of Obstacles. *IEEE Transactions on Robotics and Automation*, vol. 13, no. 3, pp.443-451.

DONNART, J; MEYER, J 1996. Learning Reactive and Planning Rules in a Motivationally Autonomous Animat. *IEEE Transactions on Systems, Man and Cybernetics* - Part B, Cybernetics, 26(3), pp. 381–395.

DOYLE, A. B. 1995. *Algorithms and Computational Techniques for Robot Path Planning*. PhD thesis, University of Wales, School of Electronics Engineering and Computer Systems, Bangor.

DRUMMOND, C. 2002. Accelerating Reinforcement Learning by Composing Solutions of Automatically Identified Subtasks. *Journal of Artificial Intelligence Research*, 16, pp 59-104.

DVOŘÁK, J. & KRČEK, P. 2005. Mobile Robot Path Planning by Means of Case-Based Reasoning. *Engineering Mechanics*, vol. 12, No. A1, pp. 219-226.

DVOŘÁK, J.; KRČEK, P. 2006. Using Case-Based Reasoning and Graph Searching Algorithms for Mobile Robot Path Planning. In *Proceedings of the 12th International Conference on Soft Computing MENDEL 2006*, Brno, pp. 151-156.

DVOŘÁK, J.; KRČEK, P. 2008. Using Genetic Algorithms for Mobile Robot Path Planning. In *Proceedings of the 14th International Conference on Soft Computing MENDEL 2008*, Brno, pp. 32-37.

DVOŘÁK, J.; KRČEK, P. 2009. Path Planning for Nonholonomic Robot. In *Proceedings of the 15th International Conference on Soft Computing MENDEL 2009*, Brno, pp. 336-343.

DVOŘÁK, J.; KRČEK, P.; SAMOHÝL, P. 2004. Using Case-Based Reasoning and Genetic Algorithms for Mobile Robot Path Planning. *Elektronika*, No. 8-9, pp. 55-57.

ERINC, G.; CARPIN, S. 2007. A Genetic Algorithm for Nonholonomic Motion Planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Roma, pp. 1843 – 1849.

FERGUSON, D.; LIKHACHEV, M.; STENTZ, A. 2005. A Guide to Heuristic-based Path Planning, *American Association for Artificial Intelligence*.

FILLIAT, D.; MEYER, J.-A. 2003. Map-Based Navigation in Mobile Robots: I. A Review of Localization Strategies. *Cognitive Systems Research*, vol. 4, no. 4, pp. 243-282.

GARRO, B. A.; SOSSA, H.; VÁZQUEZ, R. A. 2006. Path Planning Optimization Using Bio-Inspirited Algorithms. In *Proceedings of the Fifth Mexican International Conference on Artificial Intelligence*.

GEMEINDER, M.; GERKE, M. 2003. GA-Based Path Planning for Mobile Robot Systems Employing an Active Search Algorithm. *Applied Soft Computing*, vol. 3, pp. 149-158.

GERAERTS, R.; OVERMARS, M. H. 2006. Sampling and Node Adding in Probabilistic Roadmap Planners. *Robotics and Autonomous Systems*, vol. 54, issue 2, pp. 165-173.

GLASIUS, R.; KOMODA, A.; GIELEN, S. C. A. M. 1995. Neural Network Dynamics for Path Planning and Obstacle Avoidance. *Neural Networks*, vol. 8, No. I, pp. 125-133.

GRAF, B.; WANDOSELL, J. M. H.; SCHAEFFER, Ch. 2001. Flexible Path Planning for Nonholonomic Mobile Robots. In *Proceedings The Fourth European Workshop on Advanced Mobile Robots (EUROBOT)*, Lund, Sweden, pp. 199–206.

HAIGH, K.; SHEWCHUK, J. 1994. Geometric Similarity Metrics for Case-Based Reasoning. *Case-Based Reasoning, The AAAI-94 Workshop*, Seattle, WA, August, AAAI Press, pp. 182-187.

HAIGH, K.; SHEWCHUK, J.; VELOSO, M. 1994. Route Planning and Learning from Execution. *AAAI 1994 Fall Symposium on Planning and Learning: On to Real Applications,* New Orleans, LA.

HODÁL, J.; DVOŘÁK, J.; KRČEK, P. 2005. Systém pro plánování cesty robota případovým usuzováním, In *Proceedings of XXVIIth Internatinal Autumn Colloquium Advanced Simulation of Systems (ASIS 2005)*, Přerov, s. 255-260.

HOMAIFAR, A.; TUNSTEL, E.; DOZIER, G.; BATTLE, D. 2001. Genetic and Evolutionary Methods for Mobile Robot Control and Path Planning. In Ali Zillouchian and Mo Jamshidi (eds.). *Intelligent Control Systems Using Soft Computing Methodologies*, CRC Press LLC, Boca Raton.

HU, H.; BRADY, M. 1996. A Parallel Processing Architecture for Sensor-Based Control of Intelligent Mobile Robots. *Robotics and Autonomous Systems*, vol. 17, pp. 235-257.

HU, Y.; YANG, S.X. 2004. A Knowledge Based Genetic Algorithm for Path Planning of a Mobile Robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, New Orleans, vol. 5, pp. 4350-4355.

HUNG, K.-T.; LIU, J.-S.; CHANG, Y.-Z. 2007. A Comparative Study of Smooth Path Planning for a Mobile Robot by Evolutionary Multi-Objective Optimization. In *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation CIRA 2007*, Jacksonville, USA, pp. 254-259.

CHEN, X.; LI, Y. 2006. Smooth Path Planning of a Mobile Robot Using Stochastic Particle Swarm Optimization. In *Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation*, pp. 1722-1727.

CHENG, W.; TANG, Z.; ZHAO, Ch.; TANG, L.; GUO, Z. 2006. Path Planning for Nonholonomic Car-Like Mobile Robots Using Genetic Algorithms. In *Proceedings of the 8th International Conference on Signal Processing*, vol. 4, pp. 16-20.

KAMEI, K.; ISHIKAWA, M. 2004. Determination of the Optimal Values of Parameters in Reinforcement Learning for Mobile Robot Nnavigation by a Genetic Algorithm. *International Congress Series 1269*, pp. 193-196.

KASSIM, A. A.; KUMAR, B. V. K. V. 1999. Path Planners Based on the Wave Expansion Neural Network. *Robotics and Autonomous Systems*, 26, pp. 1-22.

KAVRAKI, L. E.; LATOMBE, J., C. 1998. Probabilistic Roadmaps for Robot Path Planning. *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, K. Gupta and A. del Pobil (eds), John Wiley, pp. 33-53.

KOBAYASHI, K.; SUGIHARA, K. 2002. Crystal Voronoi Diagram and its Applications. *Future Generation Computer Systems*, vol. 18, issue 5, pp. 681-692.

KRČEK, P; DVOŘÁK, J. 2004. Mobile Robot Motion Control by Means of Fuzzy Rules. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2004*, Svratka, s. 157-158.

KRČEK, P; DVOŘÁK, J. 2006. Mobile Robot Path Planning by Means of Case-Based Reasoning and Rapidly-Exploring Random Trees. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2006*, Svratka, s. 180-181.

KRČEK, P; DVOŘÁK, J. 2007. Mobile Robot Path Planning by Means of Genetic Agorithms. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2007*, Svratka, s. 133-134.

KRČEK, P.; DVOŘÁK, J. 2009. Plánování cesty mobilního robotu pomocí genetických algoritmů. *Šedesát Let Kybernetiky*, Akademické nakladatelství CERM, pp. 90-95.

KRČEK, P. 2003. *Využití expertního systému pro plánování dráhy robota*. Diplomová práce, FSI VUT, Brno.

KRČEK, P.; DVOŘÁK, J.; HODÁL, J. 2005. Mobile Robot Path Planning by Means of Case-Graph and Genetic Algorithms. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2005*, Svratka, s. 169-170.

KREJSA, J.; VĚCHET, S. 2005. Rapidly Exploring Random Trees used for Mobile Robots Path Planning. *Engineering Mechanics*, vol. 12, no. 4. pp. 231-237.

KRUUSMAA, M. 2003. Global Level Path Planning for Mobile Robots in Dynamic Environments. *Journal of Intelligent and Robotic Systems*, 38, pp. 55-83.

LAVALLE, S. M.; KUFFNER, J. J. 2001. Rapidly-Exploring Random Trees: Progress and Prospects. In Donald, B. R., Lynch, K. M.; Rus, D. editors, *Algorithmic and Computational Robotics: New Directions*, A K Peters, Wellesley, MA, pp. 293-308.

LAVALLE, S. M. 2006. *Planning Algorithms*. Cambridge University Press, (http://planning.cs.uiuc.edu/).

LEBEDEV, D.; STEIL, J. J.; RITTER, H. J. 2005. The Dynamic Wave Expansion Neural Network Model for Robot Motion Planning in Time-Varying Environments. *Neural Networks*, 18, pp. 267-285.

LEI, K.; QIU, Y.; HE, Y. 2006. A Novel Path Planning for Mobile Robots Using Modified Particle Swarm Optimizer. In *Systems and Control in Aerospace and Astronautics, 2006. ISSCAA 2006. 1st International Symposium on*, Harbin, pp. 981-984.

LIANG T.-Ch.; LIU, J.-S.; HUNG G.-T.; CHANG, Y.-Z. 2005. Practical and Flexible Path Planning for Car-Like Mobile Robot Using Maximal-Curvature Cubic Spiral. *Robotics and Autonomous Systems*, vol. 52, issue 4, pp. 312-335.

LINKER, R.; BLASS, T. 2008. Path-Planning Algorithm for Vehicles Operating in Orchards. *Biosystems Engineering*, vol. 101, issue 2, pp. 152-160.

LIU, Z.-J.; HUANG, P.; HUANG, J.-L.; QUE, J.-L. 2004. Path Planning for Tractor-Trailer Mobile Robot Based on Heuristic Genetic Algorithm. In *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, Shanghai, vol. 2, pp. 1119-1124.

LIU S.-H.; LIN J.-S.; LIN Z.-S. 2005. A Shortest-Path Network Problem Using an Annealed Ant System Algorithm. In *Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*, pp. 245-250.

LIU, S.; MAO, L.; YU, J. 2006. Path Planning Based on Ant Colony Algorithm and Distributed Local Navigation for Multi-Robot Systems. In *Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation*, pp. 1733-1738.

MARCHESE, F. M. 2002. A Directional Diffusion Algorithm on Cellular Automata for Robot Path-Planning. *Future Generation Computer Systems*, 18, pp. 983-994.

MARTÍNEZ-ALFARO, H.; GÓMEZ-GARCÍA, S. 1998. Mobile Robot Path Planning and Tracking Using Simulated Annealing and Fuzzy Logic Control. *Expert Systems with Applications*, vol. 15, issues 3-4, pp. 421-429.

MAŘÍK, V.; ŠTĚPÁNKOVÁ, O.; LAŽANSKÝ, J. a kol. 1993. *Umělá inteligence 1*, Academia, Praha.

MEYER, J.-A.; FILLIAT, D. 2003. Map-Based Navigation in Mobile Robots: II. A Review of Map-Learning and Path-Planning Strategies. *Cognitive Systems Research*, vol. 4, no. 4, pp. 283-317.

MILLÁN, J. R. 1995. Reinforcement Learning of Goal-Directed Obstacle-Avoiding Reaction Strategies in an Autonomouos Mobile Robot. *Robotics and Autonomous Systems*, 15, pp. 275-299.

NEHMZOW, U. 2003. *Mobile Robotics, A Practical Introduction*. Springer-Verlag London.

OVERMARS, M. H.,; SVESTKA, P. 1994.. *A probablisitic Learning Approach to Motion Planning*. Technical report *UU-CS-1994-03*, Utrecht University.

PODSEDKOWSKI, L.; NOWAKOWSKI, J.; IDZIKOWSKI, M.; VIZVARY, I. 2001. A New Solution for Path Planning in Partially Known or Unknown Environment for Nonholonomic Mobile Robots. *Robotics and Autonomous Systems*, 34, pp. 145-152.

PRIYA, T. K.; SRIDHARAN, K. 2006. A parallel Algorithm, Architecture and FPGA Realization for High Speed Determination of the Complete Visibility Graph for Convex Objects. *Microprocessors and Microsystems*, vol. 30, issue 1, pp. 1-14.

PRUSKI, A.; ROHMER, S. 1997. Robust Path Planning for Non-Holonomic Robots. *Journal of Intelligent and Robotic Systems*, vol. 18, no. 4, pp.329-350.

QIN, Y.; SUN, D.; LI, N.; CEN., Y. 2004. Path Planning For Mobile Robot Using the Particle Swarm Optimization with Mutation Operator. In *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, Shanghai, pp. 2473-2478.

QU, J. 1999. Nonholonomic Mobile Robot Motion Planning. *Final Report,* Iowa State University (http://msl.cs.uiuc.edu/~lavalle/cs576_1999/projects/junqu/).

RITTHIPRAVAT, P.; MANEEWARN, T.; LAOWATTANA, D.; NAKAYAMA, K. 2002. Obstacle Avoidance Using Modified Hopfield Network for Multiple Robots. *Computer and Communications,* Phuket, Thailand, vol. 1, pp. 30-31.

SEDLÁČEK, M. 2000. *Optimalizace trajektorie robota*. Diplomová práce, FSI VUT, Brno.

SASKA, M.; MACAS, M.; PREUCIL, L.; LHOTSKA, L. 2006. Robot Path Planning using Particle Swarm Optimization of Ferguson Splines. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation, ETFA '06*, pp. 833-839.

SZADECZKY-KARDOSS, E.; KISS, B. 2006. Extension of the Rapidly Exploring Random Tree Algorithm with Key Configurations for Nonholonomic Motion Planning. In *Proceedings of the IEEE International Conference on Mechatronics*, pp. 363-368.

ŠEDA, M. 2005. Motion Planning in the Plane with Polygonal Obstacles. *Engineering Mechanics*, vol. 12, no. 4. pp. 253-258.

ŠNOREK, M.; JIŘINA, M. 1996. *Neuronové sítě a neuropočítače*. Scriptum, vydavatelství ČVUT, Praha.

ŠTĚPÁN, P. 2001. *Vnitřní reprezentace prostředí pro autonomní mobilní roboty*. Disertační práce, ČVUT, Praha.

VIET, N. H.; VIEN, N.A.; LEE, S. G.; CHUNG, T. Ch. 2008. Obstacle Avoidance Path Planning for Mobile Robot Based on Multi Colony Ant Algorithm. In *Proceedings of the First International Conference on Advances in Computer-Human Interaction*, pp. 285-289.

WINKLER, Z. 2005. Odometrie. (http://robotika.cz/guide/odometry/cs).

ZHANG, Q.-Z.; LIU, X.-H.; GE, X.-S. 2007. Non-Holonomic Motion Planning with PSO and Spline Approximation. In *Proceedings of the IEEE International Conference on Control and Automation, ICCA 2007*, pp. 2600-2604.

ZHENG, C.; DING, M.; ZHOU, C.; LI, L. 2004. Coevolving and Cooperating Path Planner for Multiple Unmanned Air Vehicles. *Engineering Applications of Artificial Intelligence*, vol. 17, issue 8, pp. 887-896.

Seznam publikací autora

DVOŘÁK, J.; KRČEK, P. 2005. Mobile Robot Path Planning by Means of Case-Based Reasoning. *Engineering Mechanics*, vol. 12, No. A1, pp. 219-226.

DVOŘÁK, J.; KRČEK, P. 2005. Mobile Robot Path Planning by Means of Case-Based Reasoning. In Březina, T. (ed.), *Simulation Modelling of Mechatronic Systems I.* Mechatronika. Brno University of Technology, Faculty of Mechanical Engineering, pp. 64-69.

DVOŘÁK, J.; KRČEK, P. 2006. Nonholonomic Mobile Robot Path Planning by Means of Case-Based Reasoning. In Březina, T. (ed.), *Simulation Modelling of Mechatronic Systems II.* Mechatronika. Brno University of Technology, Faculty of Mechanical Engineering, pp. 131-137.

DVOŘÁK, J.; KRČEK, P. 2006. Using Case-Based Reasoning and Graph Searching Algorithms for Mobile Robot Path Planning. In *Proceedings of the 12th International Conference on Soft Computing MENDEL 2006*, Brno, pp. 151-156.

DVOŘÁK, J.; KRČEK, P. 2008. Using Genetic Algorithms for Mobile Robot Path Planning. In *Proceedings of the 14th International Conference on Soft Computing MENDEL 2008*, Brno, pp. 32-37.

DVOŘÁK, J.; KRČEK, P. 2009. Path Planning for Nonholonomic Robot. In *Proceedings of the 15th International Conference on Soft Computing MENDEL 2009*, Brno, pp. 336-343.

DVOŘÁK, J.; KRČEK, P.; SAMOHÝL, P. 2004. Using Case-Based Reasoning and Genetic Algorithms for Mobile Robot Path Planning. *Elektronika*, No. 8-9, pp. 55-57.

DVOŘÁK, J.; KRČEK, P.; SAMOHÝL, P. 2004. Using Case-Based Reasoning and Genetic Algorithms for Mobile Robot Path Planning. In *Proceedings of XXVIth Internatinal Autumn Colloquium Advanced Simulation of Systems (ASIS 2004)*, Sv. Hostýn, pp. 185-190.

HODÁL, J.; DVOŘÁK, J.; KRČEK, P. 2005. Systém pro plánování cesty robota případovým usuzováním, In *Proceedings of XXVIIth Internatinal Autumn Colloquium Advanced Simulation of Systems (ASIS 2005)*, Přerov, s. 255-260.

KRČEK, P; DVOŘÁK, J. 2004. Mobile Robot Motion Control by Means of Fuzzy Rules. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2004*, Svratka, s. 157-158.

KRČEK, P; DVOŘÁK, J. 2006. Mobile Robot Path Planning by Means of Case-Based Reasoning and Rapidly-Exploring Random Trees. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2006*, Svratka, s. 180-181.

KRČEK, P; DVOŘÁK, J. 2007. Mobile Robot Path Planning by Means of Genetic Algorithms. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2007*, Svratka, s. 133-134.

KRČEK, P.; DVOŘÁK, J. 2007. Nonholonomic Mobile Robot Path Planning by Means of Case-Based Reasoning. In *Proceedings of the 13th International Conference on Soft Computing MENDEL 2007*, Praha, pp. 151-156.

KRČEK, P.,; DVOŘÁK, J. 2009. Plánování cesty mobilního robotu pomocí genetických algoritmů. *Šedesát let kybernetiky*, Akademické nakladatelství CERM, s. 90-95.

KRČEK, P., DVOŘÁK, J.; HODÁL, J. 2005. Mobile Robot Path Planning by Means of Case-Graph and Genetic Algorithms. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2005*, Svratka, s. 169-170.

Seznam příloh

CD-ROM obsahující aplikaci včetně zdrojových kódů