VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

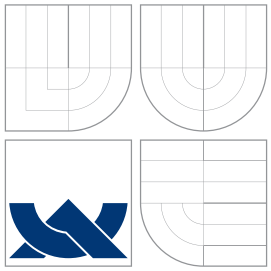
VARIACE EVOLUČNÍHO SOMA ALGORITMU PRO DYNAMICKÉ ÚLOHY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

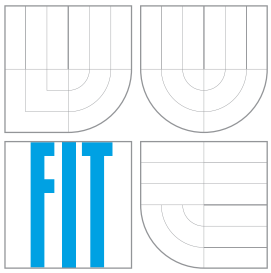
AUTOR PRÁCE
AUTHOR

JAN POKORNÝ

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VARIACE EVOLUČNÍHO SOMA ALGORITMU PRO DYNAMICKÉ ÚLOHY

VARIATION OF THE EVOLUTIONARY ALGORITHM FOR DYNAMIC PROBLEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN POKORNÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. JOSEF SCHWARZ, CSc.

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Pokorný Jan**
Obor: Informační technologie
Téma: **Variace evolučního SOMA algoritmu pro dynamické úlohy**
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte techniky a implementace evolučního algoritmu SOMA (samo se organizující migrační algoritmus) a bakalářskou práci [1].
2. Navrhněte další variace algoritmu vhodné pro dynamické úlohy.
3. Implementujte navrženou modifikaci SOMA algoritmu a otestujte ji na vybraných testovacích úlohách.
4. Zhodnoťte dosažené výsledky.

Literatura:

1. Hlavinka Michal: Diferenční evoluce pro dynamické problémy, BP, FIT VUT v Brně, 2005.
2. Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

1. Prostudujte techniky a implementace evolučního algoritmu SOMA (samo se organizující migrační algoritmus) a bakalářskou práci [1].

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Schwarz Josef, doc. Ing., CSc.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2
L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Jan Pokorný**
Id studenta: 89641
Bytem: U náhona 286/58, 503 01 Hradec Králové
Narozen: 21. 04. 1984, Hradec Králové
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**
Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Variace evolučního SOMA algoritmu pro dynamické úlohy
Vedoucí/školitel VŠKP: Schwarz Josef, doc. Ing., CSc.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☑ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy
   - (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Tato práce se zabývá používáním evolučního algoritmu SOMA a testováním jeho variant zaměřených na řešení dynamických problémů. Na začátku stručně seznamuje s problematikou evolučních algoritmů a poté se zaměřuje na evoluční algoritmus SOMA. Popisuje potíže, se kterými se potýká, jak pro statické tak i dynamické úlohy. Zmiňuje postupy, které se používají pro jejich odstraňování. Popisuje nejčastěji používané strategie – All To One, All To Random, All To All a All To All Adaptive a poukazuje na jejich výhody a nedostatky. Dále je navrhnuta i další strategie prohledávání zaměřená na funkce dynamicky se měnící nezávisle na běhu algoritmu. Samostatná kapitola je věnována projektové části práce. Je zde popsán postup implementace a propojování jednotlivých použitých programů. Tato část je dostupná na přiloženém CD spolu s výsledky testování jednotlivých strategií. Tabulky s průměrnými hodnotami jsou také součástí práce.

Klíčová slova

SOMA, evoluční algoritmy

Abstract

This study is focused on SOMA evolution algorithm and testing its versions aimed to solve dynamic problems. At the beginning it briefly explains principes of evolution algorithms a then it looks closer on SOMA algorithm. It describes its contemporary troubles for static and dynamic problems. There are also mentioned ways for their correction. It also describes the mostly used strategies – All To One, All To Random, All To All and All To All Adaptive and shows their advantages and disadvantages. Furthermore another searching strategy is proposed focused on dynamic functions that are changing independently on program. The separate chapter is about project part of the study. There is described implementation and merging of used programs. This part is available on included CD along with results of testing strategies. Tables with average values are also part of the thesis.

Keywords

SOMA, evolution algorithms

Citace

Jan Pokorný: Variace evolučního SOMA algoritmu
pro dynamické úlohy, bakalářská práce, Brno, FIT VUT v Brně, 2007

Variace evolučního SOMA algoritmu pro dynamické úlohy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Doc. Ing. Josefa Schwarze, CSc.

.....
Jan Pokorný
14. května 2007

Obsah

Kapitola 1

Úvod

S rozvojem informačních technologií se lidé začínají stále více zajímat o možnosti sestavení programu, který je schopen sám se učit a adaptovat na všechny zadané problémy. Jak bylo postupně zjišťováno, není zdaleka tak snadné dosáhnout v oblasti umělé inteligence pokroku pomocí konvenčního programovacího přístupu. Bylo nutné zvolit jiné metody.

Nyní se už nějakou dobu objevují algoritmy schopné vyhledávat řešení pomocí metod, které v sobě nemají jen pevně daný kód, ale navíc se v nich objevuje část, která se ne tak docela předvídatelně mění.

Výhody těchto evolučních algoritmů, jak se jim říká, jsou patrné na první pohled: Tam, kde je dostatečně přesné a rychlé řešení složitého problému třeba, mohou standardně používané algoritmy selhat kvůli výpočetní náročnosti. Analytická i numerická řešení mají svá časová omezení. Zde právě nastupují evoluční algoritmy s jejich netradičními přístupy.

Pokud si správně vyberete algoritmus, bývá řešení nalezeno mnohem rychleji a úsporněji.

Samozřejmě, jako téměř každá věc, mají i evoluční algoritmy svá omezení. Jejich asi hlavní a největší nevýhodou je částečná stochastičnost a z ní vyplývající nepředvídatelnost řešení. Vzhledem k této skutečnosti se matematické důkazy sestavují velmi obtížně. Poznatky o těchto algoritmech se tedy zakládají hlavně na empirických základech, které však jejich použitelnost a životaschopnost jednoznačně potvrzují.

Cílem práce je porovnat známé varianty a navrhnout modifikace SOMA algoritmu pro řešení dynamických problémů.

Kapitola 2

Evoluční algoritmy

Evoluční algoritmy jsou jednou z několika odvětví takzvaného *softcomputingu* – inteligentních výpočtů. Mimo ně tam patří například neuronové sítě a fuzzy systémy.

Zde se začíná programování dostávat do okamžiku, kdy kód pro nalezení řešení není pevně dán programem, ale slouží pouze k vytvoření prostředí, pomocí kterého bude řešení hledáno. Konkrétní postup není přesně dán ani znám, protože v softcomputingu se obvykle do jisté míry využívá náhody.

Základem hledání řešení pomocí evolučních algoritmů je převedení problému na účelovou funkci, u které bude v cyklech zvaných *generace* hledáno optimum.

Problém je pak řešen pomocí populace možných aproximací řešení, na které jsou aplikovány evoluční principy. Každá vhodná aproximace řešení (jedinec) postoupí do další generace nějak upravena. Nevhodné varianty podlehnou selekčnímu tlaku, kdy do další generace postupuje omezený počet jedinců. Takto se dosáhne vývinu populace a s každou další generací se populace více přibližuje skutečnému řešení.

Důležitou podmínkou funkčnosti evolučních algoritmů je fakt, že každá další generace musí získat lepší (nebo stejnou) aproximaci řešení. Pokud by byla tato podmínka porušena, je jakýkoliv algoritmus degradován na pouhé stochastické prohledávání stavového prostoru.

Jedním z často používaných evolučních algoritmů jsou *genetické algoritmy*. Ty pracují s jedinci reprezentovanými řetězci binárních čísel, do kterých je zakódováno kandidátní řešení. Tyto řetězce jsou ekvivalentem chromozomální DNA v přírodě. Stejně jako ona podléhají nejen změnám pomocí křížení a mutací, ale i selekčnímu tlaku „zvenčí“.

Podle požadované přesnosti nalezené aproximace řešení jsou zvoleny ukončovací podmínky. Zpravidla je buď stanoven konečný počet generací, anebo se ověřuje míra stagnace nejlepší aproximace oproti té předchozí.

Kapitola 3

Popis činnosti SOMA

3.1 Seznámení

Ačkoliv jsme se zmínili o jiných evolučních algoritmech, je tato práce primárně věnována SOMA – tedy SamoOrganizujícímu se Migračnímu Algoritmu.

Jeho první verze vznikla v roce 1999 a od té doby prošel tento algoritmus mnohými změnami a vylepšeními. Běžně se řadí mezi evoluční algoritmy, nicméně jeho činnost je však založena na geometrických principech a noví jedinci jsou interpretováni v krocích na trajektoriích přesunu. Jedné generaci v Genetických algoritmech odpovídá migrační kolo v SOMA [4]. Podobnost s genetickými algoritmy, či diferenciální evolucí spočívá hlavně ve stejných výsledcích po jednom evolučním cyklu (migračním kole).

Inspirací k tvorbě tohoto algoritmu byla spolupráce jedinců v přírodě při hledání např. zdroje potravy (mravenci). Jakmile někdo nalezne lepší zdroj, vydají se ostatní za ním.

Jako každý evoluční algoritmus i SOMA je navržena pro vyhledávání globálních extrémů zadané n-rozměrné funkce. Algoritmus je specializován pouze na hledání maxim funkce. Je jasné, že pokud budeme chtít hledat globální minimum, úplně postačí obrátit znaménko v účelové funkci. Výpočet povrchu celé této funkce je však pro nalezení extrému velmi obtížný, a proto se mu snažíme co nejvíce vyhýbat.

3.2 Základní princip funkce SOMA

Pro snazší pochopení další části bude dobré popsat alespoň některé níže použité pojmy:

- D – počet dimenzí účelové funkce
- *PathLength* – délka prozkoumávané cesty jedince v násobcích vzdálenosti od Leadera; nastavuje se v rozmezí $\langle 1, 1; 5\rangle$
- *Step* – zlomky PathLength, kde se provede vyhodnocení účelové funkce; nastavuje se v rozmezí $\langle 0, 11; PathLength\rangle$
- NP – počet jedinců v populaci; jejich množství je dobré nastavovat v přímé úměře s počtem dimenzí
- PRT – pertubační vektor

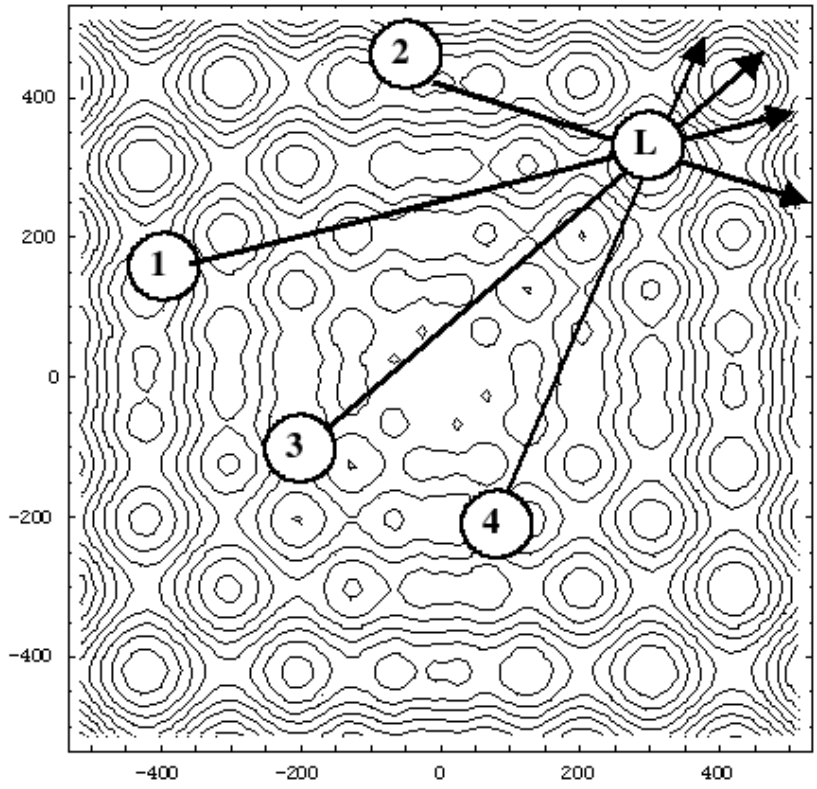
Jakmile se SOMA spustí jsou na hyperplochu náhodně vygenerováni jedinci a je vypočítána hodnota účelové funkce na jejich pozicích.

Na začátku každého kola je ze všech jedinců vybírán jedinec s nejlepší hodnotou funkce – *Leader*. Jeho pozice je v některých ohledech privilegovaná (například se nemusí hýbat).

Nyní začíná migrace. Její průběh se liší podle zvolené strategie, takže zatím popíšeme pouze základní verzi.

Zde se každý jedinec vydává na cestu za Leaderem. Délka cesty jedince (PathLength) je nastavena relativně podle jeho vzdálenosti od vedoucího jedince. Obdobně je získána délka jednoho kroku (Step).

Jakmile jsou pro každého jedince získány tyto hodnoty, vypočítá se pro každou souřadnici směrový vektor a každý jedinec podle něj začne skoky prohledávat body na hyperploše vytyčené délkou kroku, dokud se nedostane na konec své cesty 3.1.



Obrázek 3.1: Pricip SOMA – Jedinci (čísla) se pohybují směrem za Leaderem (L) a na cestě prohledávají body dané velikostí kroku.

Je výhodné, když cesta jedince nekončí na místě, kde se nalézá vedoucí jedinec, ale pokračuje až za něj. Díky tomu se zajistí určitá žádoucí diverzita populace – zabrání se tak přílišnému nahloučení jedinců na jednom místě. Ze stejných důvodů se také délka jednoho kroku nastavuje tak, aby jedinec neprováděl výpočet hodnoty na stejném místě, na jakém se právě nalézá Leader. Jak je vidno, toto umístění by mohlo znamenat do konečných důsledků až překrytí jedince s Leaderem, které by trvalo až do skončení běhu celého algoritmu.

Z celé své cesty si jedinec pamatuje vždy umístění s nejlepší možnou hodnotou. V okamžiku, kdy dojde na konec své cesty se na něj přemístí.

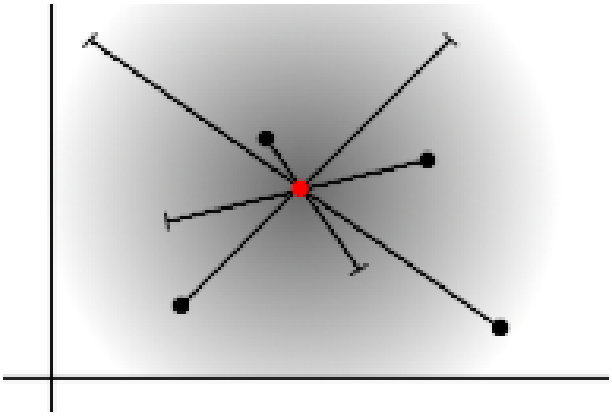
Jakmile svůj tah provedou všichni jedninci, končí migrační kolo a z „nových“ jedinců se opět vybere nový Leader.

Jak je vidět, je samotný princip algoritmu velice jednoduchý a nikde není nutné provádět výpočet složitější než jednoduché dělení [5].

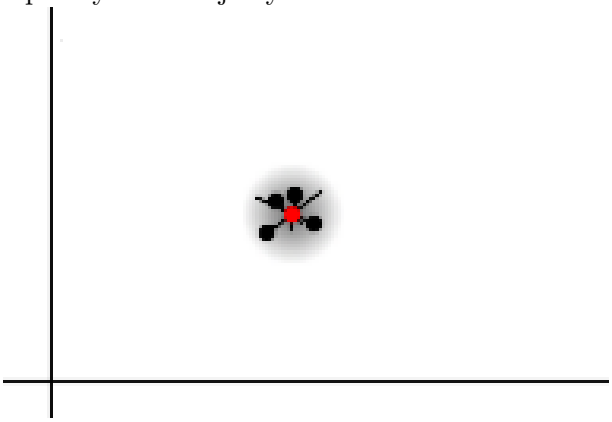
3.3 Pertubační vektor

Základní princip fungování algoritmu byl samozřejmě již dlouhou dobu vylepšován a upravován, aby SOMA podával co nejleší výsledky. Jedním z velkých problémů, kterými algoritmus trpěl, byla ztráta diverzity populace (obr. 3.3).

V okamžiku, kdy se jedinci příliš nahloučí kolem jednoho bodu, je velice těžké donutit je prohledávat i zbylou část stavového prostoru, kde se ve skutečnosti může nalézat globální extrém.



Obrázek 3.2: Ztráta diverzity populace – Zde jsou jedinci dosud rozmístěni tak, že se svou PathLength (vyznačena úsečkami; nastavena na 2,5) jsou schopni prozkoumat oblast zhruba o rozměrech šedé části plochy. Leader je vyznačen červeně.



Obrázek 3.3: Ztráta diverzity populace – Pokud se však jedinci příliš nahloučí kolem Leadera (červeně), plocha prohledávaného prostoru se může drasticky zmenšit.

Toto uváznutí v lokálním extrému se částečně řeší takzvaným *pertubačním vektorem*

(pertubace – porucha, rušení). Jednoduše to znamená, že přímý pohyb jedince je rušen tak, aby jeho dráha nemusela být přímá a jedinec navíc získal i možnost částečně měnit svůj směr. Byla vyzkoušena možnost donutit vybočovat jedince z dráhy pomocí změn jeho směrového vektoru. Kupodivu se však ukázalo, že velice podobně kvalitní výsledky přináší i prosté nulování některých ze souřadnic vektoru v jednotlivých krocích [5].

Nakonec je tedy náhodnost v pohybu jedinců řešena takto. Výpočet dalšího bodu na cestě se počítá pro každý rozměr každého jedince podle vzorce:

$$(Ind^i_{id\_leader} - Ind^i_n) \cdot d \cdot Step \cdot PRT^i + Ind^i_n,$$

kde Ind je hodnota i-tého rozměru n-tého jedince, Ind_{id_leader} obdobně hodnota Leadera, d počet kroků $Step$ a PRT znázorňuje konkrétní rozměr pertubačního vektoru, který je složen z jedniček a nul. Ze vzorce je vidět jak určuje, zda se bude jedinec v daném rozměru pohybovat.

Jak již bylo řečeno, hlavním přínosem zavedení pertubačního vektoru do SOMA je právě přispění k udržení diverzity populace. Další podrobnosti jsou popsány v části věnované statickým a dynamickým problémům algoritmu.

3.4 Statické problémy

Statickými problémy jsou myšleny situace, ke kterým může dojít, když je funkce statická – nemění se v čase (v průběhu evolučního procesu).

3.4.1 Ztráta diverzity

Jako každý podobný algoritmus má i SOMA své specifické potíže způsobující snížení efetivity, které je nutné řešit. Asi největším problémem SOMA je ztráta diverzity populace, o němž jsme se již krátce zmínili.

Pokud vše pracuje, jak má, jsou jednici na hyperploše rozprostřeni poměrně pravidelně. Jediným rozdílem jsou v takovém případě místa hledaných extrémů. Zde dochází k jistému nahloučení, což je v omezené míře nutné k upřesňování pozice hledaného místa globálního maxima. Potíže nastávají, jakmile pozici blízkou nějakému extrému obsadí příliš mnoho nebo dokonce všichni jedinci. Nezřídka se totiž stává, že ono místo není globálním, ale pouze lokálním extrémem.

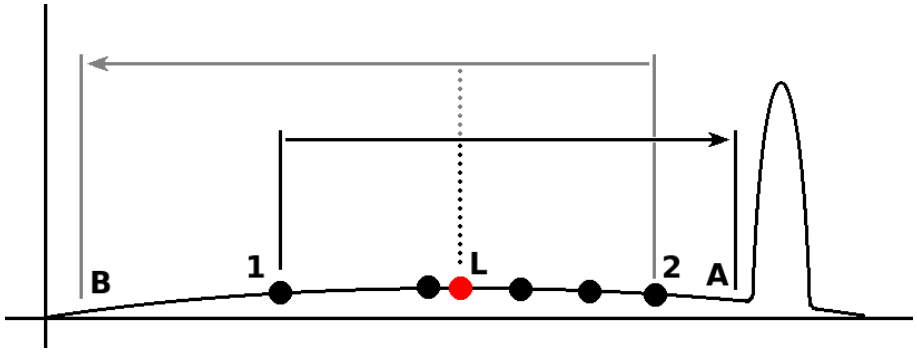
S jedinci rozmístěnými na jediném místě má algoritmus přehled o velmi omezené části stavového prostoru a globální extrém se bez odstranění nahloučení nemusí vůbec najít.

Příkladem tohoto stavu může být takzvaná „jehla v kupce sena“ (obr. 3.4 a 3.5).

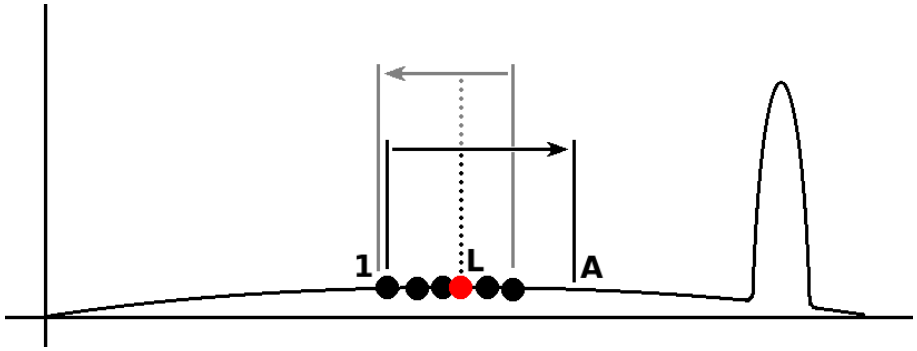
Jak je na obrázku vidět, jedinci se pohybují směrem k širokému vrcholu, přičemž skutečný extrém zůstává nepovšimnut. Vzhledem k povaze pohybu jedinců (pouze za Leaderem), je není možné přinutit k opětovnému „rozchodu“ a výrazně lepší hodnota nebude nikdy nalezena.

3.4.2 Uváznutí na rovné ploše

Druhým zmíněným problémem je uváznutí na rovné ploše. V tomto případě se situace komplikuje v speciálním případě, a to v momentě, kdy je většina hyperplochy rovina rovnoběžná s osami.



Obrázek 3.4: „Jehla v kupce sena“ – S PathLength = 2,5 se jedinec 1 může dostat nejdále do bodu A, kdy však stále nedosáhne extrému. Pokud by se jedinec 2 přesunul až do bodu B, nalezl by další cestou globální extrém. Nejlepší bod na jeho cestě ale leží v okolí Leadera (L), takže optima nebude dosaženo.



Obrázek 3.5: „Jehla v kupce sena“ – Jakmile se jedinci nahloučí kolem širokého lokálního extrému, není již možné bez úpravy algoritmu najít ten globální. Oproti obr. 3.4 je vidět, že bod A je mnohem dále od hledaného optima.

V takovém případě jedinci mohou ve svém pohybu minout dostatečně úzký vrchol a dojde tak k uváznutí, kdy všechna možná místa prohledávaná jedinci mají stejnou hodnotu. V tomto případě ustává pohyb jedinců.

Jedno z navržených řešení nutí jedince v případě nalezení bodu se stejnou hodnotou přesunout se tam. Takto by se jedinci vždy přesunuli na konec své možné dráhy a při vhodně nastavené *PathLength* by se celá populace postupně kyvadlovitě rozhoupávala a zvětšovala tak prohledávaný prostor. Pertubace by zároveň zajistila lepší prohledání míst, která by se jinak mohla nalézat mezi cestami jedinců.

Tento proces by však trval několik migračních kol, než by došlo k dostatečnému rozptýlení jedinců a až potom by začalo opětovné upřesnění pozice extrému.

Další informace o těchto problémech naleznete na [1].

3.5 Dynamické problémy

Zde se účelová funkce postupně mění s časem a tak s sebou přináší další rozměr potíží a klasická SOMA není schopna účinně sledovat globální extrém. Výše popsané statické problémy

nemusí nastávat příliš často, ale v dynamických funkcích jejich negativní ovlivňování kvality výpočtu dramaticky roste.

Dále tedy popíšeme jak tyto problémy řešit.

Ztráta diverzity populace, která se v případě statické funkce s jedním vrcholem může zdát neškodnou, způsobí se změnou pozice tohoto vrcholu obrovské potíže.

Jedinci se nahloučí na jednom místě a díky tomu není možné docílit nějakého většího skoku. Jakmile se tedy vrchol začne pomalu posouvat stranou, jedinci se ho pokusí následovat. Vše záleží na rychlosti posunu vrcholu a na míře s jakou je diverzita zachována.

Bude-li se vrchol pohybovat jen zvolna, může se jedincům dařit udržet se na jeho vrcholu. Při větší rychlosti pohybu vrcholu za ním budou nahloučení jedinci postupně zaostávat, až nakonec úplně ztratí jeho stopu. Pokud se však vrchol bude pohybovat velkými skoky, pak jej příliš nahloučená skupina jedinců může ztratit i v jediném kole.

Další kritická situace nastává, když se mění výška globálního extrému. Výše uvedený problém „jehla v kupce sena“ (obr. 3.4 a 3.5) se stává další pastí na jedince: Zatímco se jedinci dostanou na pozici uprostřed mírného kopce, začne na jeho okraji růst tenká jehla, která rychle přesáhne dosud globální maximum. Vzhledem k tomu, že jedinci nemají důvod opustit své pozice, zůstává skutečný extrém nepovšimnut.

V neposlední řadě také vyvstává otázka nalezení řešení funkce podle způsobu, jakým se mění.

V zásadě se funkce může měnit:

- po několika migračních kolech – před další změnou funkce mají jedinci dost času nalézt řešení
- každé migrační kolo – v takovém případě hodně záleží na velikosti změny funkce
- nezávisle na běhu algoritmu – V tomto případě nastávají potíže, jakmile část jedinců, kteří se již pohnuli, má náhle neplatné informace, protože tver účelové funkce se již posunul. Z těchto mylných informací však může vycházet zbytek populace, a vzniká tak problém.

Tyto potíže se podařilo výrazně zredukovat použitím nového vylepšení, které v roce 2006 zavedl Michal Hlavinka. Jeho přínos paradoxně spočívá v zavedení omezené délky života jedince.

Těm je v tomto případě přidělen čas, po který budou existovat. Jakmile uplyne počet kol, který mají stanoven, jedinec je vymazán z pozice, na které se nachází a okamžitě je opět vytvořen na náhodně vybraném místě s vynulovaným věkem. Výkon algoritmu se také odvíjí od tohoto faktoru – maximálního věku, kterého se může jedinec dožít.

Samozřejmostí je, že při počáteční inicializaci algoritmu není všem jedincům nastaven stejný věk, aby se dosáhlo rovnoměrného umírání.

Další důležitou věcí je privilegovanost Leadera. Jelikož nejlepšího jedince nelze ztratit (došlo by k degeneraci výsledků), je Leaderovi udělena vyjímka a nestárne.

Díky znouvytváření jedinců na náhodných pozicích je udržována rovnováha mezi dostatečnou diverzitou a přesností nalezeného řešení, která je pro správný chod algoritmu klíčová.

3.6 Varianty algoritmu

Až doposud jsme se věnovali nejrozšířenější strategii prohledávání prostoru. Během času, kdy byla SOMA vylepšována, se ale také objevily alternativní strategie. Jejich podstatou

je změna chování každého jedince podle jiného vzorce. Takových strategií existuje několik a každá má, jak už to bývá, své klady a zápory, které tu budou probrány.

3.6.1 All To Random

Strategie All To Random se už podle názvu orientuje na směr větší stochastičnosti při hledání extrému. Aby však nedošlo k degradaci algoritmu je stále nutné zachovat konvergenci výsledků ke správnému řešení. Jinak řečeno: Funkce doposud nelezených nejlepších jedinců musí být neklesající.

Základním principem této strategie je pohyb veškerých jedinců, ne za Leaderem – nejlepším jedincem, ale za náhodně vybraným jedincem, který se v tomto ohledu chová obdobně jako Leader ze strategie All To One.

Každé kolo je tedy náhodně vybrán nějaký jedinec, za kterým se ostatní vydají podle stejného principu jako ze základní strategie.

Všimněte si, že jediné, co je pro tuto strategii nutné změnit, je pouze určení, kdo bude následován.

Strategie All To Random má dozajista výhodu v lepším prohledávání celého stavového prostoru, protože náhodný pohyb jedinců omezuje šanci na ztrátu diverzity populace. Je jasné, že prakticky s každou další migrací se téměř všichni jedinci vydají zcela jiným směrem.

Nevýhodou by v takovém případě ovšem bylo obtížněji upřesnitelné doposud nalezené řešení. Souvisí to právě se zmenšením šance na ztrátu diverzity populace. Vzhledem ke skutečnosti, že se jedinci pohybují po velkém území a nejlepší jedinec je téměř ignorován, bude kolem něj minimální počet „následníků“.

3.6.2 All To All

Další možností prohledávání je strategie All To All. Zde už je rozdíl od základní strategie více patrný. Leader se sice chová stejně jako v předchozí strategii, ale pohyb jedinců se skládá z více než jedné jediné cesty.

Každý jedinec se chová tak, jako by Leaderem byl každý jiný jedinec – vše v jednom svém tahu. Je pro něj tedy vytyčeno tolik tras kolik je celkem ostatních jedinců. Z těchto cest je nakonec vybrán bod s nejlepším ohodnocením (princip prohledávání každé cesty je stejný jako v All To One, jen následovaný jedinec není pouze Leader), kam se jedinec přesune (obr. 3.6).

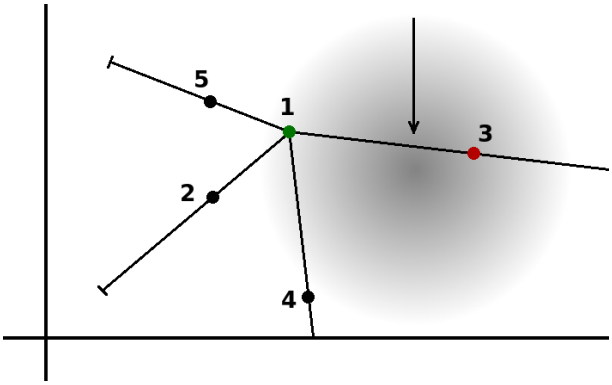
Již je vidět, že tato strategie prohledává stavový prostor velice důkladně a vzhledem k tomu, že mezi možnými cestami každého jedince je vždy i ta za Leaderem, tak nevzniká problém jako ve strategii All To Random, kde není tak důkladně prohledáváno okolí Leadera.

Místo toho má tato varianta algoritmu jiné úskalí. Každé migrační kolo s sebou velký počet evaluací. Navíc počet těchto výpočtů roste kvadraticky úměrně množství jedinců.

3.6.3 All To All Adaptive

Podobná předchozí variantě je strategie All To All Adaptive. Leader se chová stejně jako ve výše uvedeném All To All. Rozdíl spočívá ve způsobu pohybu jedince.

Zatímco v All To All se jedinec přemístí na novou pozici až po ukončení procházení všech vytyčených cest, v All To All Adaptive je prováděn přesun jedince na lepší pozici po každé prohlédnuté cestě. Dráha jedince se tedy liší od „hvězdy“ ze strategie All To All. Pohyb je spíš pobobný tvaru lomené čáry.



Obrázek 3.6: Ilustrace výpočtu pohybu jedince ve strategii All To All. Jedinec (zeleně) prohledá všechny trasy za ostatními jedinci (2, 3, 4 a 5), jejichž délka je daná velikostí PathLength (zde 2,5) a poté se přesune na nejlepší pozici, kterou nalezne. Leader je vyznačen červeně. Šipka ukazuje zhruba na místo, kam se jedinec přesune. Přesná pozice je ovlivněna délkou kroku jedince.

Pro lepší pochopení rozdílu v pohybu jedinců u strategií All To All a All To All Adaptive si můžete prostudovat obrázky 3.6 a 3.7.

Jak je vidět tato strategie bude mít stejné klady a zápory jako strategie předchozí.

3.6.4 Nová varianta algoritmu – All To Elite

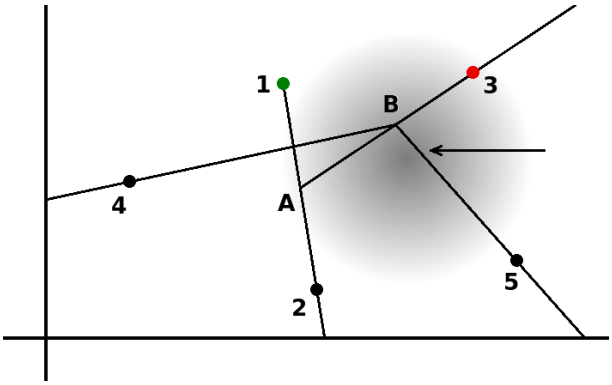
All To Elite je autorem práce vytvořená alternativa ke strategii pohybu jedinců po hyperploše. Alespoň částečně vychází ze všch výše uvedených strategií, nicméně principiálně je zase něco jiného.

Základem je v tomto případě informace, že jedinci se nepohybují jen za jedním jediným Leaderem.

Ten je zde pouze jedním členem elity – skupiny jedinců s nejlepším ohodnocením. Její velikost je nastavitelná, ale je dobré, je-li elity méně než polovina jedinců. Když nastavíme jako elitní všechny jedince, začne se algoritmus chovat podobně jako se strategií All To Random, nicméně principy vyhledávání nejlepších jedinců budou v takovém případě algoritmus zpomalovat.

Každé migrační kolo je nalezena nová elita a začne pohyb: Pro každého jedince (včetně elity) je náhodně vybrán jeden (jiný) člen elity, za kterým se vydá. Díky většímu počtu následovaných jedinců, kteří mají dobré ohodnocení může být prohledáno větší množství extrémů za menší počet kol. Protože je cílem často opět jiný elitní jedinec, předchází tato strategie zániku diverzity populace. Jakmile má funkce více maxim, jedinci je prozkoumají výrazně rychleji než v základní strategii. Pokud je na druhou stranu extrém jen jediný, je velmi pravděpodoné, že během krátké doby budou všichni elitní jedinci nahloučeni kolem něj.

Za toto samozřejmě strategie platí snížením prohledávání nejbližších míst kolem nejlepšího jedince a také vzrůstem výpočetní náročnosti kvůli porovnávání kvality jedinců.



Obrázek 3.7: Ilustrace výpočtu pohybu jedince ve strategii All To All Adaptive. Vybraný jedinec (zeleně), který se má pohybovat, z cesty za dalším jedincem v pořadí (2) vybere nejlepší bod (A). Ten se pro něj stává výchozím bodem pro jeho další pohyb. Opět nalezne nejlepší bod na cestě za 3 (B). Při průchodu cesty z B za jedincem č. 4, není nalezena lepší než výchozí pozice a jedinec zůstává na pozici. Šipka ukazuje zhruba na místo, kam se jedinec nakonec přesune. Přesná pozice je ovlivněna délkou kroku jedince. Leader je vyznačen červeně.

Kapitola 4

Popis implementace a úpravy kódu

4.1 Úloha s pohybujícími se vrcholy

Práce využívala zdrojových kódů programu Pohybující se vrcholy – Moving Peaks Benchmark.

Tento program nebylo potřeba přímo modifikovat a lze ho stáhnout ze [2]. Byl vytvořen a je určen pro generování vícerozměrné dynamické funkce s více vrcholy. Jako implementační jazyk byl zvolen C, což je pro propojení se zdrojovými kódy knihovny SOMA velmi výhodné.

Klíčovými funkcemi pro propojení byly funkce `dummy\_eval` a `eval\_movpeaks`, které obě vrací hodnotu funkce v bodě daným parametry. Rozdíl spočívá ve způsobu vyhodnocení. Zatímco `eval\_movpeaks` provede kompletní výpočet hodnot a započítá evaluaci funkce pro pozdější výpočet velikosti chyby a podobně, `dummy\_eval` pouze vrátí hodnotu funkce. Tato vlastnost je užitečná při inicializaci SOMA a při příležitostech, kdy by započítání hodnoty mělo pro výsledky zkreslující význam.

Součástí Moving peaks je i množství nastavitelných parametrů. Bylo by dobré zmínit alespoň ty, které nás budou nejvíce zajímat.

- change_frequency – počet vyhodnocení funkce, po kterém se změní
- number_of_peaks – počet vrcholů, který bude funkce mít
- geno_size – hodnota nastavuje počet rozměrů funkce
- vlength – nastavuje, o kolik se pohnou vrcholy při změně tvaru funkce
- height_severity – nastavuje, o kolik se změní výška vrcholů při změně tvaru funkce
- width_severity – nastavuje, o kolik se změní šířka vrcholů při změně tvaru funkce
- mincoordinate – hodnota minima na všech osách
- maxcoordinate – hodnota maxima na všech osách
- minheight – minimální výška vrcholu
- maxheight – maximální výška vrcholu
- standardheight – vytvoří vrcholy dané výšky; je-li nastaveno na nulu, tvoří je náhodně v daném rozmezí

- minwidth – minimální šířka vrcholu
- maxwidth – maximální šířka vrcholu
- standardwidth – vytvoří vrcholy dané šířky; je-li nastaveno na nulu, tvoří je náhodně v daném rozmezí

Tyto parametry se nastavují uvnitř souboru `movpeaks.c`.

4.2 SOMA knihovna

Hlavní částí implementace, která byla nutná pro testování, bylo propojení dvou programů. Prvním z nich byla knihovna SOMA, jejíž tvorbou a popisem se zabývá [1].

Byla naprogramovaná s důrazem kladeným na přenositelnost a použitelnost s jinými programy. Obsahuje metody pro běh SOMA se základní strategií All To One. Jako programovací jazyk byl zvolen C++. Vzhledem k objektové formě tohoto programu je velmi snadné mu porozumět, a provádět i případné změny v kódu. Součástí je i grafický výstup v podobě animace sloužící k snadnému zobrazení postupu hledání řešení. Ačkoliv jsou zobrazovány pouze první dva rozměry testovací funkce, je animace velmi dobrým způsobem, jak mít možnost vidět průběh řešení zadaného problému. Výstup se automaticky ukládá do souboru soma.gif (pokud je tedy zapnuté generování animace).

Bylo by dobré alespoň krátce vysvětlit, k čemu slouží jednotlivé soubory tohoto programu.

- `error.h`, `error.c` – tyto soubory slouží k výpisu chybových hlášek na standardní chybový výstup
- `function.h` – je hlavičkovým souborem pro soubor určený k propojení SOMA knihovny s jiným programem. Jeho název není pevně dán.
- `individual.h`, `individual.cpp` – tyto soubory obsahují třídu Individual (jedinec), kde jsou definovány operace s jedinci
- `main.cpp` – hlavní soubor SOMA, který obsahuje inicializaci a nastavení parametrů
- `mygif.h`, `mygif.cpp` – tyto soubory vykonávají funkce spojené s generováním animace. Nemají v sobě žádné části, které by nějak ovlivňovaly prohledávání
- `param.h`, `param.cpp` – zde je třída s parametry jedinců
- `soma.h`, `soma.cpp` – tyto soubory jsou jádrem celého programu, jsou zde používány třídy z ostatních souborů a metody, které popisují vlastní pohyb jedinců po hyperploše

Vzhledem k povaze práce bylo však nutné narušit strukturu knihovny a nyní je upravena tak, aby byla schopná správně komunikovat s druhým programem – Moving Peaks Benchmark. Vygenerováno bylo více podobných variant algoritmu, které se liší strategií prohledávání. Změny, které byly provedeny v původním kódu se tak liší v každé verzi strategie.

Základem bylo vytvoření souboru `dmovpeaks.h`, kde dochází ke komunikaci s oběma programy. Jedinou informací, kterou SOMA pro svůj běh potřebuje, je hodnota účelové funkce v zadaném bodě.

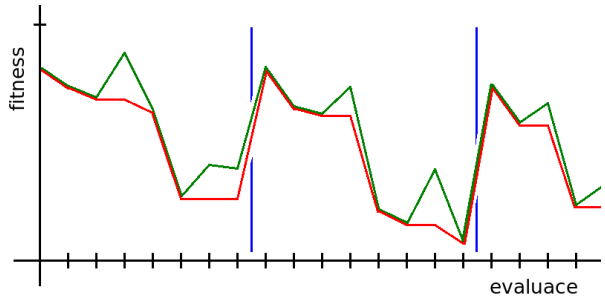
Vzhledem ke klíčovosti kódu tohoto souboru, jej bude dobré popsat celý:

```
extern „C\ {
#include „movpeaks.h\
#include „movpeaks.c\
}

class cF_dmovpeaks : public cFunction
{
  public:
    double *gen; /* pomocná proměnná pro převod do pole typu double */
    unsigned long successRate; /* proměnná preo výpočet success rate */

    cF_dmovpeaks() {
        init_peaks(); /* vytvoření funkce */
        successRate=0;

        /* nastavení počtu, typu a rozsahu rozměrů */
        addParameter(cParam(0,100,cParam::REAL));
        addParameter(cParam(0,100,cParam::REAL));
        addParameter(cParam(0,100,cParam::REAL));
        addParameter(cParam(0,100,cParam::REAL));
        addParameter(cParam(0,100,cParam::REAL));

        gen = new double[dim];
    }

    virtual ~cF_dmovpeaks() {
        free_peaks(); /* ukončení moving peaks */
        free(gen);

        /* výpis výsledků */
        std::cout<<get_offline_error()<<
           „\t\<<successRate/(0.0+get_number_of_evals())<<endl;
    }

    /* vrátí aktuální hodnotu globálního extrému */
    double BestVal() const { return global_max; }

    /* vrátí true v˜případě, že je nalezeno dostatečně přesné maximum */
    bool isSolution(const cIndividual &ind)
        const { return (ind.Value>global_max-0.01); }

    /* představení se funkce */
    void Introduce() const
    {
        std::cout<<„Dynamic: Moving peaks function\n\;
    }

    /* plnohodnotný výpočet hodnoty funkce v˜bodě daném parametrem */
```

```
    void costValue1(cIndividual &ind)
    {
        successRate+=get_right_peak();

        for (int i=0;i<dim;i++)
            gen[i]=ind[i];

        ind.Value = eval_movpeaks(gen);

        if (soma->getEvalCount()%5000 == 0)
        {
            change_peaks();
            soma->clearOffErrValue();
        }
    }

    /* výpočet pouze pro účely, kdy ho není
       dobré zahrnout do celkového stavu (inicializace apod.) */
    void costValue2(cIndividual &ind)
    {
        for (int i=0;i<dim;i++)
            gen[i]=ind[i];

        ind.Value = dummy_eval(gen);
    }
};
```

4.3 Kriteriální funkce

Pro testování bylo využito výpočtu hodnoty *offline error*. Tato hodnota udává rozdíl řešení nalezeného konkrétním jedincem od skutečného globálního maxima. Tato hodnota má udržovánu nerostoucí posloupnost a je resetována pouze v okamžitku změny funkce.



Obrázek 4.1: Ilustrace tvaru grafu offline error: Na ose Y je vynášena fitness konkrétní evaluace, na ose X jsou jednotlivé evaluace. Modré svislé čáry znázorňují okamžik změny tvaru účelové funkce. Zeleně je vynesen graf online error – odchylky od správného řešení. Červeně je vidět graf offline error, který je v úsecích mezi změnami funkce nerostoucí.

Výpočet hodnoty průměrného offline error tedy probíhá po každé evaluaci podle vzorce:

$$averageOfflineError = \frac{averageOfflineError \cdot evalCount + leastOffErrValue}{evalCount + 1},$$

kde *evalCount* odpovídá celkovému počtu dosud provedených vyhodnocení funkce a *leastOffErrValue* je nejmenší dosud nalezená hodnota chyby, která je s každou změnou tvaru funkce resetována.

Výpočet offline error byl ve starších verzích algoritmu implementován vlastní, později, po dalších úpravách algoritmu, bylo využito výpočtu této hodnoty přímo programem Moving Peaks Benchmark.

Další počítanou hodnotou byl Success rate – podíl počtu dosažení globálního optima k celkovému počtu pokusů/běhů. Tento výpočet bylo možné provést pomocí funkce z Moving Peaks Benchmark `get_right_peak()`. Tato funkce vrací výsledek typu `bool`, podle toho, zda bylo při posledním vyhodnocení funkce nalezeno optimum. Z celkového počtu vyhodnocení funkce `get_right_peak()` se poté spočítal aritmetický průměr.

Výsledné hodnoty pro jednotlivá testování jsou umístěna v kapitole Testy.

4.4 Implementace strategií

Další částí úprav algoritmu byla reimplementace SOMA knihovny pro další strategie než All To One. Vzhledem k malému počtu úprav nutnému k této změně byla první verze upraveného programu uzpůsobena tak, že mohla ovládat všechny později použité strategie, což by mohlo být pro komplexní testování, které by mohlo trvat velmi dlouho přínosem. Ukázalo se však, že s rostoucím množstvím použitých příkazů neúměrně klesá rychlost algoritmu.

Proto bylo přikročeno k implementacím jednotlivých strategií každé zvlášť. Postupem času se ukázalo, že je nutné programy propojit oběma směry, což je v rozporu se způsobem jakým je knihovna SOMA napsána. Nakonec bylo nutné provést změny, které použitou implementaci SOMA přímo uzpůsobují na komunikaci s Moving Peaks Benchmark.

Strategii All To One nebylo nutné nijak modifikovat, pouze byly přidány části kódu pro komunikaci s Moving Peaks Benchmark a pro výpočet potřebných hodnot a není nutné ji snad dále popisovat.

Malé úpravy byly potřeba pro strategii All To Random. V tom to případě bylo nutné zajistit, aby se jedinci pokaždé vydávali za jiným jedincem, přičemž Leader by měl zůstat nehybný.

Hlavní úprava byla provedena v souboru `soma.cpp`, kde bylo nutné pozměnit kód v metodě `doStep()`. Tato metoda se stala cílem změn i pro ostatní strategie. Oproti All To One přibyla v tomto případě proměnná `id_target`, která reprezentuje následovaného jedince. Do ní je každé kolo ukládána náhodná hodnota v rozmezí počtu jedinců. Následován je tak každé kolo náhodně vybraný jedinec.

Strategie All TO All Adaptive vyžadovala úpravu ve stejné metodě, ačkoliv výrazně složitější. Pro tento případ bylo nutné přidat další cykl tak, aby každý jedinec následoval postupně všechny ostatní jedince. Jako v předchozí variantě jedinci následují pokaždé někoho

jiného, ale v tomto případě každý z nich jako by prováděl několik migračních kol sám za sebe. Nakonec se přesune na nejlepší nalezenou polohu. Upraveno je také stárnutí jedince a to tak, aby věk přibýval po celé sérii migrací, jinak by mohlo docházet až k několikému úmrtí jedince během jediného migračního kola.

All to All přinesla kupodivu složitější úpravy kódu než její sesterská strategie All To All Adaptive. Hlavní změnou oproti ní je nutnost ukládání si hodnot nejlepších jedinců na dosud prošlých pozicích (obr. 3.6), aby bylo vůbec možné realizovat poslední krok – přesun jedince na pozici.

K uchování hodnot byla na pole upravena proměnná `pomBestInd`.

Poslední implementovanou strategií je All To Elite, kde bylo nutné zavést ukládání nejlepších jedinců do obousměrně přístupné fronty. V našem případě je to:

```
deque<unsigned long> bestID
```

V metodě `doStep()` je provedena změna tak, aby byl před každým pohybem jedince, zvolen náhodně vybraný člen elity, který bude v zápětí následován. Samozřejmostí je kontrola, zda jedinec nebude následovat sám sebe. Po nalezení nového umístění, je zjištěno, zda jedinec nemá lepší hodnotu než aktuální nejslabší elitní člen. Je-li tomu tak, je jedinec postupně porovnáván s členy `bestID`, dokud není nalezen elitní jedinec jehož hodnota je vyšší nebo není dosaženo konce. Poté je kandidát zařazen na úkor nejslabšího jedince do elity.

Zásahy bylo v tomto případě nutné provést i do dalších metod, kvůli inicializaci, reinicializaci a destrukci objektu SOMA.

Kapitola 5

Experimenty

Na SOMA byly provedeny série testů zaměřené na získání míry úspěšnosti při řešení funkcí generovaných pomocí Moving Peaks Benchmark. Výslednými hodnotami byl jednak offline error a také míra úspěšnosti nalezení správného vrcholu – success rate. Testovány byly strategie All To One a All To Elite.

Pro testování byly pevně nastaveny nastaveny tyto parametry:

- Nastavení programu Moving Peaks Benchmark
  - počet vrcholů: 10
  - pohyb vrcholu při změně funkce: 1,0
  - počet rozměrů: 5
  - míra změny výšky vrcholů: 1,0
  - míra změny šířky vrcholů: 0,01
  - hranice plochy pro všechny rozměry: $\langle 0{,}0; 100{,}0 \rangle$
  - rozmezí výšek vytvořených vrcholů: $\langle 30{,}0; 70{,}0 \rangle$
  - standardní výška vrcholu je tvořena náhodně
  - rozmezí šířek vytvořených vrcholů: $\langle 1{,}0; 12{,}0 \rangle$
  - standardní šířka vrcholu je tvořena náhodně
  - počet vyhodnocení funkce před změnou funkce nastaven na 0, změna je prováděna ručně voláním `change_peaks`
- Nastavení programu SOMA
  - PathLength: 2,8
  - Step: $PathLength/(20{,}0 \cdot 1{,}1) \doteq 0{,}127$
  - počet jedinců: 25-krát počet rozměrů
  - míra pertubace: 0,7
  - maximální věk jedince: 7
- Další nastavení
  - celkový počet vyhodnocení funkce: 500000
  - počet evaluací na změnu funkce: 5000

Počet vyhodnocení funkce může dosahovat o něco vyšších hodnot, protože algoritmus vždy dokončí celou migraci.

Každý řádek v tabulce byl získán výpočtem hodnot jednoho sta běhů algoritmu. Hodnota uvedená v závorkách je standardní odchylka.

Z tabulek je vidět, že hodnoty offline error dosahují i při malém množství vrcholů velkých hodnotových výkyvů. Vysvětlením je v takovém případě to, že tvar funkce se často mění uprostřed migračního kola a výpočty části jedinců tak získají hodnoty, které už nejsou správné. V takovém případě velikost chyby velice brzy stoupne do nepříjemně vysokých hodnot. Tento problém zatím není možné uspokojivě řešit.

5.1 All To One

Tato série testů byla zaměřena na chování algoritmu SOMA se strategií prohledávání All To One.

V tabulce 5.1 byl počítán offline error a success rate pro 1, 2, 3, 5, 8 a 10 rozměrů. Počet vrcholů byl fixně nastaven na 10 a míra pohybu vrcholu byla 1,0.

Počet rozměrů	Offline error		Success rate	
1	0,32	(0,02)	0,96	(0,01)
2	0,87	(0,04)	0,48	(0,03)
3	3,16	(1,76)	0,82	(0,21)
5	6,48	(5,53)	0,65	(0,20)
8	2,88	(3,01)	0,89	(0,21)
10	51,42	(19,6)	0,25	(0,00)

Tabulka 5.1: All To One – měnící se počet rozměrů

Jak je z výsledků vidět, s rostoucím počtem rozměrů náhle prudce roste chyba. Počet jedinců totiž roste úměrně s množstvím rozměrů a tím pádem stoupá i počet evaluací účelové funkce každé migrační kolo.

Při deseti rozměrech již počet evaluací účelové funkce během jedné migrace přesáhne 5000, což je hodnota, kdy se mění tvar účelové funkce. Během každé migrace tak dojde ke změně jejího tvaru, což se negativně odrazí na výsledcích. Počet evaluací za jedno kolo pro tento případ vypočteme pomocí vzorce:

$$\frac{(NP-1)\cdot PathLength}{Step},$$

kde NP znamená počet jedinců. Jakmile dosadíme použité hodnoty pro 10 rozměrů dostaneme:

$$\frac{(250-1)\cdot 2,8}{0,127} \doteq 5490,$$

V případě výpočtu s 8 rozměry vychází výsledek zhruba 4390, což znamená, že během migrace nedochází vždy ke změně tvaru účelové funkce.

Pro více než deset rozměrů je proto již navržen jiný vzorec pro výpočet počtu jedinců:

$$NP = 250 + 10(dim - 10),$$

kde *dim* odpovídá počtu rozměrů funkce. Pomocí tohoto vzorce přibývá s každým dalším přidaným rozměrem pouze výrazně méně jedinců. Tak se zabrání přílišnému nárůstu evaluací. Ukazuje se, že pro tyto hodnoty množství jedinců jsou výsledky srovnatelné [1].

V druhé tabulce 5.2 se měnil počet vrcholů v rozmezí 1 – 500, kdy už bylo výrazně znát zpomalení rychlosti výpočtu. Rozměrů bylo dáno pevně 5 a míra pohybu vrcholu byla 1,0.

Počet vrcholů	Offline error		Success rate	
1	11,43	(8,48)	1,00	(0,00)
3	12,32	(6,47)	0,71	(0,34)
5	4,35	(3,39)	0,83	(0,15)
8	21,76	(6,07)	0,13	(0,29)
10	4,42	(3,54)	0,82	(0,17)
12	18,73	(6,31)	0,15	(0,16)
15	8,96	(4,56)	0,09	(0,09)
18	11,97	(3,55)	0,05	(0,10)
50	9,60	(0,24)	0,00	(0,01)
100	7,00	(1,75)	0,02	(0,02)
200	4,49	(1,64)	0,03	(0,01)
500	5,53	(1,28)	0,01	(0,01)

Tabulka 5.2: All To One – měnící se počet vrcholů

Z výsledků se dá vypozorovat tendence poklesu offline error. Zatímco při méně vrcholech mají jedinci velkou pravděpodobnost umístění na špatné pozici, s rostoucím množstvím vrcholů se postupně „rodí“ na lepších pozicích s čím dál větší pravděpodobností. Je vidět, že i počáteční rozkolísanost odchylky se spíše urovnává. Samozřejmě spolu s rostoucím množstvím vrcholů klesá pravděpodobnost, že se jedinec při své cestě dostane na ten pravý.

Dalším kritériem testů byla vzdálenost, o kterou poskočily vrcholy při změně funkce. Hodnoty v tabulce 5.3 byly postupně nastaveny na 1, 5, 10 a 50. Výpočet byl prováděn pro 5 rozměrů a 10 vrcholů.

Velikost změny	Offline error		Success rate	
1	5,39	(3,45)	0,70	(0,09)
5	11,98	(1,23)	0,72	(0,00)
10	22,43	(0,99)	0,72	(0,00)
50	50,01	(0,78)	0,53	(0,03)

Tabulka 5.3: All To One – změna velikosti pohybu vrcholů

Zde je vidět, že u jedinců stále dochází k nahloučení, které se projevuje přímo úměrně roustoucí mírou chybovosti. Pokud se vrchol pohne více, vetšímu množství jedinců klesne fitness funkce.

5.2 All To Elite

Další sada obdobných testů se zaměřila na strategii All To Elite. Tato strategie měla pevně nastaveno deset elitních jedinců. Jak je již vidět z následujících tabulek, výsledky strategií jsou poměrně podobné.

Tabulka 5.4 ukazuje výsledky pro 1, 2, 3, 5, 8 a 10 rozměrů.

V porovnání se strategií All To One přináší hodnota offline error drobné zlepšení. Naproti tomu hodnoty success rate dosahují o něco horších výsledků. Vzhledem k povaze prohledávání All To Elite se ovšem zhoršení těchto hodnot dalo očekávat.

Počet rozměrů	Offline error		Success rate	
1	0,29	(0,02)	0,96	(0,01)
2	0,85	(0,03)	0,47	(0,01)
3	2,67	(1,26)	0,87	(0,13)
5	4,36	(1,00)	0,72	(0,02)
8	2,36	(1,27)	0,93	(0,10)
10	47,37	(19,91)	0,25	(0,00)

Tabulka 5.4: All To Elite – měnící se počet rozměrů

Srovnatelných výsledků se dosáhlo i při testování se změnou množství vrcholů v tabulce 5.5. Hodnoty All To Elite sice většinou dosahují lepšího offline error, nicméně ne vždy. A jako v případě změny počtu rozměrů je hodnota success rate v porovnání se strategií All To One o něco horší.

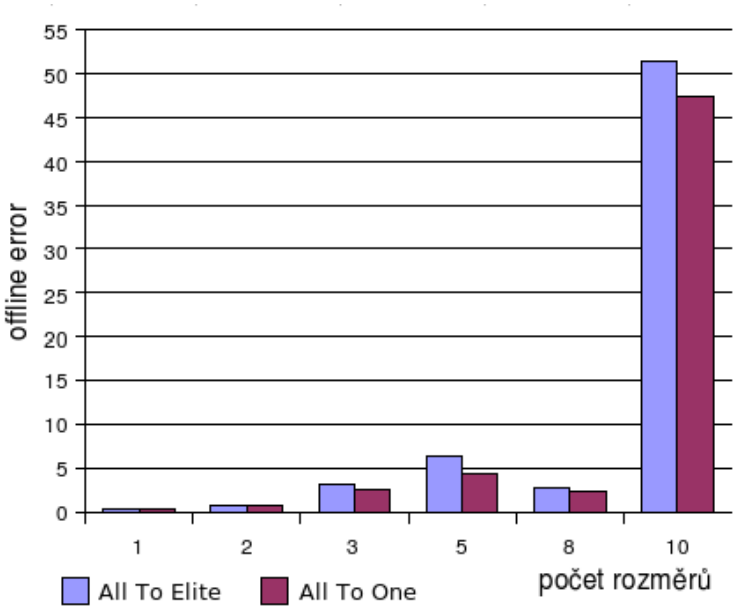
Počet vrcholů	Offline error		Success rate	
1	9,39	(5,35)	1,00	(0,00)
3	11,36	(6,28)	0,54	(0,36)
5	15,75	(4,30)	0,08	(0,19)
8	24,41	(6,39)	0,10	(0,29)
10	4,91	(3,20)	0,71	(0,07)
12	12,69	(2,03)	0,01	(0,01)
15	7,58	(1,95)	0,07	(0,08)
18	15,64	(2,78)	0,02	(0,07)
50	4,06	(0,07)	0,08	(0,01)
100	3,30	(0,30)	0,13	(0,02)
200	4,81	(1,04)	0,03	(0,03)
500	5,13	(0,79)	0,02	(0,01)

Tabulka 5.5: All To Elite – měnící se počet vrcholů

I v případě porovnávání podle velikosti skoku vrcholů (tabulka 5.6) vyhrává All To Elite na offline error pouze o malé rozdíly v hodnotách.

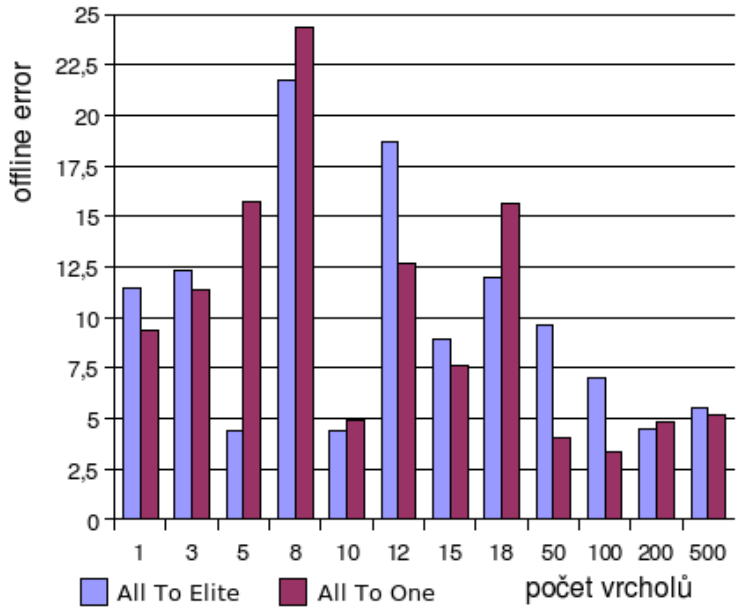
Velikost změny	Offline error		Success rate	
1	4,55	(1,67)	0,72	(0,03)
5	10,54	(0,48)	0,72	(0,00)
10	18,78	(0,65)	0,72	(0,00)
50	48,90	(0,87)	0,54	(0,03)

Tabulka 5.6: All To Elite – změna velikosti pohybu vrcholů

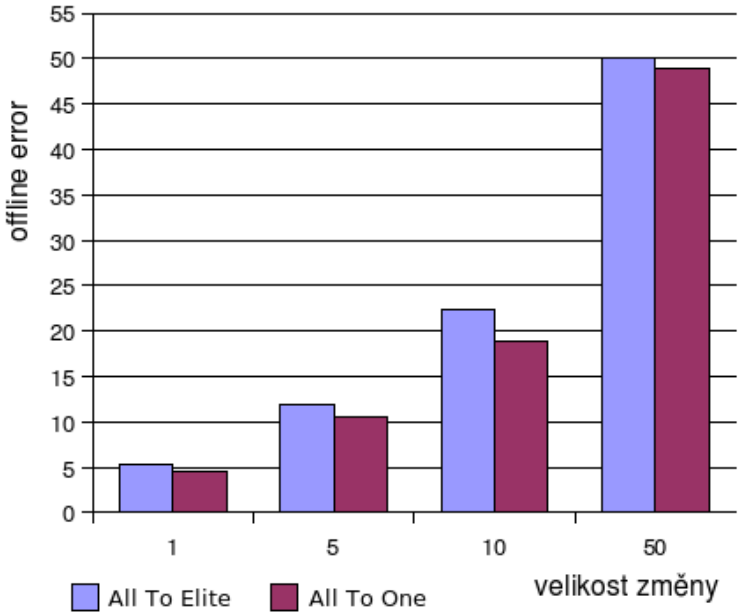


Obrázek 5.1: Graf offline error u strategií All To One a All To Elite pro různý počet rozměrů

Přesné výsledky jednotlivých experimentů, animace migrací všech uvedených strategií a testy strategií All To All/Random/Adaptive naleznete na přiloženém médiu.



Obrázek 5.2: Graf offline error u strategií All To One a All To Elite pro různý počet vrcholů



Obrázek 5.3: Graf offline error u strategií All To One a All To Elite pro různou velikost skoku vrcholů

Kapitola 6

Závěr

Dalo by se říci, že z testovaných strategií vyšla lépe All To Elite. Offline error v nalézání řešení při změně počtu rozměrů vyšel v průměru o 14 procent lépe než u strategie All To One. Obdobného úspěchu bylo dosaženo při porovnávání těchto strategií při změně velikosti skoku vrcholu. Zde vyšel All To Elite o necelých 12 procent lépe. Zhoršení oproti All To One však nastalo při změně počtu vrcholů, kdy offline error All To Elite vyšel v průměru o 11 procent hůře.

Velká část síly SOMA spočívá v její jednoduchosti, kde prakticky v celém algorimu není nutné použít operaci složitější než násobení a není tak snadné provádět úpravy, které by efektivitu algoritmu výrazně zvýšily.

Patrně by se dalo dosáhnout dalšího vylepšení algoritmu pomocí modifikace množství elitních jedinců v závislosti na počtu dimenzí, rozměrech plochy a množství použitých jedinců. Leč vzhledem k množství parametrů nastavitelných pro SOMA, nebylo zdaleka možné provést všechny testy, a proto výsledky nemusely vždy dosáhnout optimálních hodnot.

Pravdou tak stále zůstává tvrzení vážící se k evolučním algoritmům známá jako „no free lunch theorem“ [3]: V evolučních algoritmech neexistuje takový, který by byl schopen podávat nejlepší výsledky ve všech typech zadaných problémů. Specializace algoritmu s sebou přinese zhoršení pro jiné typy úkolů.

Další informace, včetně konkrétních výsledků testů, které zde nemusí být všechny uvedeny, jsou obsaženy na přiloženém médiu.

Literatura

[1] Michal Hlavinka. *Diferenční evoluce pro dynamické problémy – Bakalářská práce.* Vysoké učení technické v Brně - Fakulta informačních technologií, 2006.

[2] WWW stránky. Stránky věnované moving peaks benchmark [cit. 2007-05-08]. `http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/`.

[3] WWW stránky. Wikipedia – no free lunch in search and optimization [cit. 2007-05-08]. `http://en.wikipedia.org/wiki/No_free_lunch_in_search_and_optimization`.

[4] WWW stránky. Ivan Zelinka: Osobní stránky věnované SOMA [cit. 2007-05-08]. `http://www.ft.utb.cz/people/zelinka/soma/`.

[5] Ivan Zelinka. *Umělá inteligence v problémech globální optimalizace.* BEN – technická literatura, 2002. ISBN 80-73000-69-5.