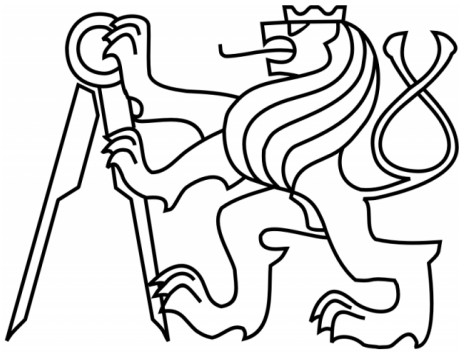
České vysoké učení technické v Praze
Fakulta elektrotechnická
Softwarové technologie a management
Inteligentní systémy



BAKALÁŘSKÁ PRÁCE
Katedra kybernetiky

Evoluční algoritmus řízený klasifikačním modelem

Vedoucí práce: **Ing. Petr Pošík, Ph.D.**
Student: **Tomáš Tököly**

květen 2011

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 27.5.2011 .....

Jakub .....

Podpis

Poděkování

Tímto bych velmi rád poděkoval především panu Petru Pošíkovi za vedení bakalářské práce a ochotnou odbornou pomoc a také mé rodině a všem přátelům za jejich morální i duchovní podporu.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Tomáš Tökölyová

Studijní program: Softwarové technologie a management

Obor: Inteligentní systémy

Název tématu: Evoluční algoritmus řízený klasifikačním modelem

Pokyny pro vypracování:

1. Nastudujte problematiku konstrukce a využití stromových klasifikátorů typu CART.
2. Ve zvoleném programovacím jazyce implementujte evoluční algoritmus využívající CART klasifikátor ke konstrukci nových kandidátských řešení.
3. Výsledný algoritmus porovnejte na testovacích optimalizačních úlohách s dalšími optimalizátory. Výsledky zhodnoťte a identifikujte potenciální přínos této metody.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Petr Pošík, Ph.D.

Platnost zadání: do konce letního semestru 2010/2011

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 2. 9. 2010

Abstrakt

Obsahem této bakalářské práce je popis evolučního algoritmu řízeného klasifikačním modelem (EACBT, Evolutionary Algorithm Controlled by Tree), konkrétněji CART stromem (Classification and Regression Tree), jeho implementace a následné testování na funkcích binární proměnné oproti klasickému evolučnímu algoritmu (EA) a náhodnému prohledávání.

Kritériem kvality optimalizátoru je především schopnost nalézt globální optimum, počet ohodnocení nutný k jeho nalezení a rychlost konvergence. Výsledky ukazují, že pro jednoduché monotónní funkce je klasický EA nejlepší, ovšem pro některé multimodální funkce má klasický EA tendence uváznout v lokálním optimu, kdežto EACBT je schopný tyto lokální optima překonat a dokonvergovat až do optima globálního. Náhodné prohledávání je statisticky ve většině případů horší, kromě nízkých dimenzí u složitějších funkcí, kde je naopak nejlepší, protože ostatní algoritmy uváznou v lokálních optimech. EACBT má určitě potenciál k tomu, aby byl podroben dalšímu zkoumání.

Klíčová slova: optimalizace, klasifikace, evoluční algoritmy

Abstract

The content of this thesis is a description of an evolutionary algorithm controlled by classification model (EACBT, Evolutionary Algorithm Controlled by Tree), specifically by CART tree (Classification and Regression Tree), its implementation and testing on functions of binary variable compared to classical evolutionary algorithm (EA) and random search.

The criterion for the quality of the optimizer is mainly the ability to find global optimum, the number of evaluations required for finding it and the speed of convergence. The results show that for a simple monotonic function is a classic EA best, but for some multimodal features a classic EA has a tendency to get stuck in local optimum, while EACBT is able to overcome these local optima and converge further to global optimum. Random search is statistically worse in most cases, except for low dimensions of complex functions, where it is on the contrary the best, because the other algorithms get stuck in local optimum. EACBT definitely has the potential to be submitted to further research.

Key words: optimization, classification, evolutionary algorithms

Obsah

Seznam obrázků a grafů

Seznam tabulek

Seznam a obsah příloh

1. Úvod

Na světě existuje celá řada různých optimalizačních úloh v různých odvětvích průmyslu, zdravotnictví, dopravy, školství, bankovnictví a dalších, které, pokud jsou optimálně vyřešeny, dokáží ušetřit nejen množství času a peněz. Může se jednat například o nalezení nejkratší cesty z jednoho bodu do druhého, o sestavení školních rozvrhů, o plánování výrobní linky, případně o nalezení nějakého extrému funkce jako v případě této práce.

Využívá se při tom tzv. optimalizačních algoritmů či metod, jejichž vstupem je nějaký optimalizační problém a výstupem je nalezení takové sekvence akcí, která vede do požadovaného stavu, případně nalezení takových hodnot proměnných pro které daná funkce nabývá maximální či minimální hodnoty.

V této bakalářské práci (BP) se zaměřím na optimalizaci funkcí binární proměnné pomocí klasických evolučních algoritmů (EA) a evolučních algoritmů řízených klasifikačním modelem (EACBT, Evolutionary Algorithm Controlled By Tree). Motivací k probádání právě tohoto způsobu optimalizace byl algoritmus popsaný v článku Learnable Evolution Model (LEM) [1], ve kterém používají k řízení EA pravidlové systémy. V této BP používám namísto pravidlového modelu, model stromu. Cílem práce tedy je vzájemné porovnání klasických EA, náhodného prohledávání a EACBT a zhodnocení jestli tento způsob optimalizace je přínosem či nikoliv. K testování poslouží pětice binárních funkcí s různou složitostí a pro různé dimenze.

Veškerý kód implementovaných algoritmů je napsaný v prostředí MATLAB a k práci je připojený jako příloha na DVD.

2. Evoluční algoritmy

Evoluční algoritmy se řadí mezi oblíbené optimalizační metody v oboru umělé inteligence. Evoluční se jim říká proto, že jsou inspirované přírodou, konkrétně jejím zvířecím reprodukčním cyklem, kde na základě Darwinových myšlenek povětšinou přežívají jen ti nejschopnější jedinci.

Oblíbenost EA spočívá především na tom, že jsou velmi robustní a tedy jsou vhodné k použití u tzv. „black-box“ funkcí. A také na tom, že dokáží i na velmi složitých problémech podávat dobré výsledky v relativně krátkém čase.

Následující podkapitola popisuje hlavní myšlenku a kostru evolučních algoritmů spolu s prvky, které se konkrétně týkají této BP.

2.1 Popis evolučních algoritmů

2.1.1 Základní princip

EA jsou algoritmy, které využívají kolektivního učení jednotlivých jedinců, kteří reprezentují řešení daného problému. Základním požadavkem je, abychom byli schopní ohodnotit jejich kvalitu a porovnávat je mezi sebou. Nejprve je potřeba **inicializovat úvodní populaci** skládající se z jednotlivých jedinců. Ta se většinou vygeneruje náhodně a následně ohodnotí pomocí tzv. fitness funkce. Jinými slovy, náhodně se vygeneruje genotyp jedinců, v binární reprezentaci tvořený řetězcem jedniček a nul, a poté se ohodnotí. Hodnota fitness funkce daného jedince nám říká, o jak kvalitního jedince se jedná a tedy jak moc je či není uzpůsoben k „přežití“. Způsob jakým se genotyp projevuje v prostředí, nazýváme fenotyp.

Dále nastupuje operátor **selekce**, kde se nějakým způsobem určí dva rodiče, kteří se mezi sebou budou křížit. Více viz 2.1.2.

Poté se provede operátor **křížení**. Dva vybraní jedinci se mezi sebou zkříží a vyprodukují jiné dva jedince. Detailnější popis viz 2.1.3.

Selekci a křížení opakujeme tak dlouho, dokud nevygenerujeme dostatečný počet jedinců, obvykle ve stejném počtu jako je počet rodičů.

Posledním operátorem je operátor **mutace**, kde nově vygenerovaní jedinci, v případě binární reprezentace obvykle reprezentovaní bitovým číslem, zmutují. Více viz 2.1.4.

Pak už záleží na konkrétní implementaci **náhradové strategie**, jakým způsobem pomocí takto vygenerovaných jedinců vytvoříme další populaci potomků. Zda-li za nové rodiče prohlásíme jen nově vytvořené jedince či jejich smícháním se stávajícími rodiči. Náhradových strategií existuje několik typů, níže jsou popsány tři druhy, z nichž dva používám v této BP.

Tělo evolučních algoritmů:

- inicializace a ohodnocení počáteční populace
- generační cyklus:
    - operátor selekce: výběr rodičů pro křížení
    - operátor křížení: zkřížení vybraných rodičů
    - operátor mutace: mutace nových potomků
    - ohodnocení nových potomků
    - dle náhradové strategie zařaď potomky do populace
- opakuj, dokud není splněna ukončovací podmínka

Obecně, čím více upřednostňujeme silnější jedince, tím rychleji algoritmus konverguje do optima, ovšem s rizikem že nalezené optimum bude lokální a ne hledané globální. Čím je

upřednostňujeme méně, tím je riziko uváznutí v lokálním optimu menší, ovšem algoritmus konverguje pomaleji. Optimální řešení je tedy někde mezi, a to, kde přesně mezi, je stále předmětem zkoumání a experimentů.

2.1.2 Operátor selekce

Operátor selekce je způsob jak vybrat dva rodiče určené ke křížení. První rodič se obvykle vybírá buď zcela náhodně z celé populace, nebo se pomocí turnaje vybere několik vyvolených, z nichž zvítězí ten s nejlepší fitness. Druhý rodič by měl být jiný než první rodič a může se vybírat obdobně jako v případě prvního rodiče.

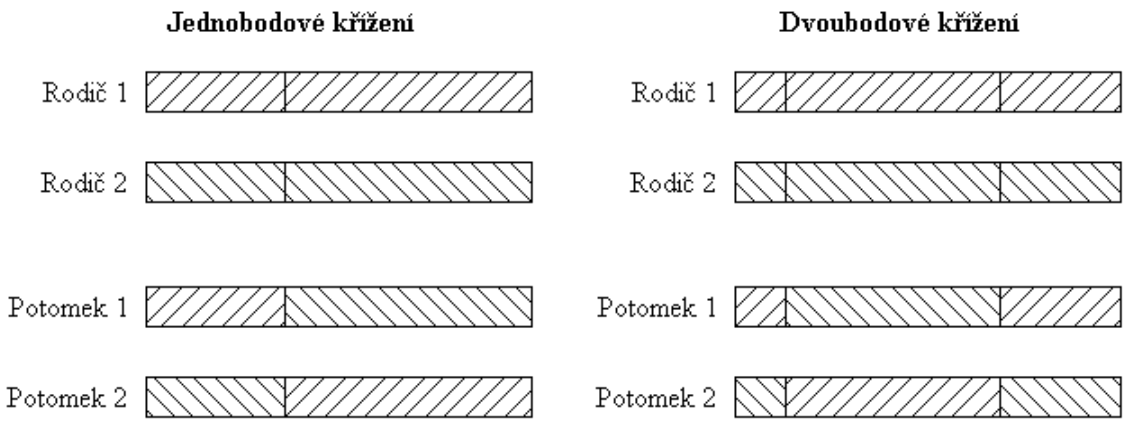
Toto je důležitý krok, na kterém stojí celý princip klasických EA, protože je zde nutné upřednostnit silnější, kvalitnější jedince před těmi horšími, slabšími. Ovšem ne je zcela vyloučit, protože i jedinec s horším řešením může nést část informace vedoucí k nalezení optimálního řešení. Toto se může provádět například ruletovým kolem či turnajem.

Při **turnaji** se obvykle vybere zcela náhodně předem určený počet jedinců (dle jeho velikosti) a zvítězí v něm ten jedinec, který má nejvyšší fitness, nejkvalitnější řešení. Při menších velikostech turnaje mají horší jedinci větší pravděpodobnost, že se zúčastní dalšího reprodukčního cyklu, při větších naopak menší, protože pravděpodobnost, že se mezi všemi účastníky turnaje vyskytne alespoň jeden kvalitní, je vyšší.

Druhého rodiče můžeme vybírat i v závislosti na prvním rodiči. Například turnajem vybrat též několik jedinců, z nichž nezvítězí ten s nejlepší fitness, ale ten který je prvnímu rodiči nejblíže nebo naopak nejdále. Podrobnější studii o vlivu výběru operátoru selekce na EA naleznete v [2].

2.1.3 Operátor křížení

Pro binární reprezentaci je nejčastěji používaným způsobem jedno či dvojbodové křížení. Vezmeme genotyp, binární řetězec obou rodičů a v případě jednobodového křížení je náhodně na jednom místě přeřízneme, čímž nám vzniknou dvě části od každého rodiče. Poté prohodíme jednu část (první či druhou) jednoho rodiče s druhým a tím nám vzniknou dva noví potomci nesoucí informaci jejich rodičů. Dvojbodové křížení je obdobné s tím, že se řez provede na dvou místech, čímž vzniknou od každého rodiče části tři. Poté středové části obou rodičů prohodíme, čímž též vzniknou dva noví potomci. Více viz obr. 1.



Obrázek 1: Princip jedno a dvojbodového křížení

2.1.4 Operátor mutace

V tomto kroku se projde genotyp všech jedinců v populaci, tedy každý jejich bit a s nějakou danou pravděpodobností se buď invertuje, nebo nastaví na nějakou hodnotu bez ohledu na původní hodnotu bitu.

2.1.5 Náhradové strategie

Generační náhradová strategie (GNS). Tato náhradová strategie je asi nejjednodušší k implementaci. Poté co z rodičů nakřížíme požadovaný počet jedinců, prohlásíme je rovnou za novou generaci určenou ke křížení a tu starou, generaci rodičů, zahodíme. Ovšem je dobré zachovat dosud nejlepšího nalezeného jedince, což se dá provést pomocí tzv. elitismu, viz 2.1.6.

Náhradové strategie v tzv. „Steady-State“ modelu (SSNS). Mezi „steady-state“ náhradové strategie se řadí všechny strategie, které nějakým způsobem umožňují přežití jedinců ze staré rodičovské populace do nové populace potomků. Například po vykřížení nových jedinců, můžeme obě populace rodičů a potomků spojit. Poté z nich budeme turnajem vybírat jedince, které budeme přesouvat do nového kontejneru, po jehož naplnění jej prohlásíme za novou generaci rodičů. I v tomto případě je vhodné použití elitismu.

Restricted Tournament Replacement (RTR). RTR náhradová strategie je součástí SSNS, uvádím ji ale zvlášť, protože v implementovaném kódu používám SSNS popsanou výše i RTR popsanou zde. V této strategii postupně procházíme všechny nově vygenerované jedince, kde pro každého z nich vybereme *x* jedinců ze staré rodičovské populace, kde *x* je velikost turnaje. Následně z vybraných jedinců vybereme toho, který je novému jedinci nejbližší. Např. pro náš binární případ používám Hammingovu vzdálenost[1]. Poté porovnáme fitness obou jedinců a toho lepšího z nich vložíme do nové populace potomků. Opakujeme, dokud neprojdeme všechny nově vygenerované jedince.

[1] Hammingova vzdálenost dvou řetězců značí na kolika pozicích se liší. Tedy v tomto případě na kolika pozicích se liší genom dvou jedinců.

Při použití této strategie není nutné používat elitismus, protože do nové populace se vybírají vždy buď lepší, nebo stejně dobří jedinci a tedy nedochází ke ztrátě dosud nejlepšího nalezeného řešení. Jelikož tato náhrada se snaží na místo nového potomka nalézt nejdříve vždy toho nejbližšího a až potom se rozhoduje na základě fitness, dosáhne toho, že populace si udržuje vyšší míru diverzity. Konverguje tedy pomaleji než SSNS, ovšem s menší šancí, že uvázne v případném lokálním optimu.

2.1.6 Elitismus

Úlohou elitismu je zachování *x* dosud nejlepších nalezených řešení, kde *x* je velikost elitismu. Dá se to provést například jednoduchým spojením staré, rodičovské populace a nové, právě vykřížené, populace potomků a jejich následné prohledání pro *x* nejlepších jedinců, které poté vezmeme a slepě je přesuneme do nové generace. Přesunutím zajistíme, že o nejlepší jedince nepřijdeme a že je již nebudeme používat v procesu nahrazování staré populace novou, dle zvolené strategie. Tedy že nejlepší jedinci nebudou v nové populaci vícekrát. Zbytek populace doplníme dle použité náhradové strategie.

2.2 Alternativní principy optimalizace

Způsobů jak řešit optimalizační úlohy je mnoho a čím více o tom daném problému víme, tím spíše můžeme použít algoritmus, který bude daný problém schopen řešit efektivněji. Obecně ale platí, že o řešeném problému nevíme zhola nic, či velmi málo. Takovým problémům se říká, jak jsem již zmínil v úvodu 2. kapitoly, tzv. „černá skříňka“ (anglicky „black-box“), která na nějaký vstup reaguje určitým výstupem. Ale o tom co se děje uvnitř nemáme žádnou představu. Jinými slovy, při nějak nastavených vstupních parametrech, dostaneme nějaký výstup, který zpravidla chceme mít na nějaké optimální hodnotě (minimální či maximální), ale nevíme, jaká kombinace vstupních parametrů k takové hodnotě vede. Stavový prostor je prohledávaná oblast, obsahující všechny možné hodnoty a kombinace vstupních parametrů, ve které se snažíme najít takovou kombinaci, která nás dovede do globálního optima. Uvedu zde metodu, která je schopná, více či méně efektivně, optimum ve stavovém prostoru černých skříněk nalézat. Jedná se o algoritmus LEM, o kterém se zmiňuji již v úvodu.

U klasického EA jsou rekombinační operátory (tedy křížení a mutace) stochastické operace a jako takové nemohou řídit generování nových potomků ze zkušenosti předchozích generací. Jedná se tedy v podstatě o jakousi formu souběžného provádění metody pokusu a omylu. Je ale možné potomky vytvářet i jiným než stochastickým způsobem. Na populaci můžeme pohlížet jako na ohodnocenou datovou sadu, extrahovat z ní znalosti a ty následně využít ke konstrukci nových potomků.

2.2.1 Popis a učení klasifikačního modelu LEM

Learnable Evolution Model je optimalizační metoda, o níž tvůrci prohlašují, že „výrazně překonala klasické evoluční algoritmy“.

Obdobně jako u klasických EA je i zde potřeba vygenerovat počáteční populaci, což se provádí náhodně.

Poté se spustí tzv. mód Strojového učení (anglicky Machine Learning mode), který sestává ze dvou procesů. V prvním se vytváří tzv. „hypotézy“, které se snaží popsat rozdíly mezi kvalitními a nekvalitními jedinci a jejich charakteristiku. Popisovaní jedinci můžou pocházet z aktuální generace, případně z aktuální a jedné či více minulých generací. V druhém procesu se pomocí takto naučených hypotéz vygenerují noví jedinci. Mód Strojového učení se opakuje, dokud není splněna jeho ukončovací podmínka, tedy dokud řešení není přijato jako uspokojivé nebo dokud nejsou vypotřebovány přiřazené výpočetní prostředky.

Poté se může a nemusí (záleží na nastavení algoritmu), spustit mód „Darwinistické evoluce“. V tomto módu se optimalizace zhostí klasický EA, fungující obdobně jako algoritmus popsaný v 2.1. Též opakujeme, dokud není splněna ukončovací podmínka.

Oba módy můžeme střídat nebo může běžet jen mód Strojového učení. Po nalezení uspokojivého řešení či po daném počtu iterací se algoritmus zastaví.

Pro vytváření hypotéz ve Strojovém módu se používá AQ learner, podrobnosti o něm i o celém algoritmu naleznete v [1].

Tělo algoritmu LEM:

- inicializace počáteční populace
- spusť mód Strojového učení:
    - rozděl populaci na dobrou a špatnou
    - aplikuj metodu strojového učení k vytvoření popisu dobrých jedinců a popisu špatných jedinců
    - vytvoření nové populace
    - opakuj, dokud není splněna ukončovací podmínka módu Strojového učení
- spusť mód Darwinistické evoluce:
    - viz popis EA v 2.1
    - opakuj, dokud není splněna ukončovací podmínka módu Darwinistické evoluce
- střídej oba módy, dokud není splněna LEM ukončovací podmínka

Důvodů, kterými LEM přispěl k tomu, proč se rozhodlo, že má cenu EACBT algoritmus vůbec zkoušet, je více. Chtěl jsem vyzkoušet, zda je pro úspěšnost algoritmu zásadní střídání

Strojového módu a módu Darwinistické evoluce či zda je zásadní použití modelu strojového učení ve formě pravidel. A také zda je algoritmus úspěšný i pro jinou reprezentaci než reálnou.

3. Naivní algoritmus

V této kapitole se nejdříve chci věnovat popisu základního modelu řízeného CART stromem a poté provedeným experimentům a z nich vyplívajících závěrů.

3.1 Popis základního principu

Myšlenka zde prezentovaného algoritmu (EACBT) spočívá v tom, že není nutné používat selekci a křížení k vytváření nových generací, ale je možné pokusit se řídit populaci jiným způsobem, přesněji CART stromem.

Jako i v jiných optimalizačních algoritmech i zde je nutné inicializovat počáteční populaci, což se provádí náhodně, a následně se ohodnotí fitness funkcí. Pak už nastupuje generační smyčka, kde v každé generaci vezmeme aktuální populaci rodičů a seřadíme ji dle fitness. Pak, dle zvoleného parametru ji v nějakém místě rozdělíme. Tu lepší část nazveme „OK“ populací, tu horší část „KO“ populací. Takto rozdělená populace je vstupem algoritmu pro tvorbu stromu, který se pomocí ní učí rozpoznávat kvalitní jedince od těch méně kvalitních.

Dalším krokem je nalezení všech listů stromu a rozdělení je na listy obsahující „OK“ jedince a „KO“ jedince. Také je potřeba zjistit obor listů, tedy definici oblasti stavového prostoru, kterou list pokrývá. Podrobněji o tom co je obor listu, viz 3.2.1. Každému listu přiřadíme váhy dle fitness jedinců, kteří do listu spadají. List, který „pokrývá“ kvalitnější jedince, bude mít větší váhu než list který ne. Zároveň je možné pomocí vstupního parametru regulovat jak velkou váhu budou mít „OK“ či „KO“ listy oproti sobě. A na základě přiřazených vah vypočteme, kolik jedinců bude pomocí daného listu vygenerováno pro novou generaci; a následně ji vygenerujeme. Generace nových jedinců probíhá tak, že se vygenerují náhodně a na místech kde je definován obor daného listu se nastaví dle jeho hodnot.

I zde nastupuje operátor mutace a poté pomocí fitness funkce novou populaci ohodnotíme. Pak již jen dle použité náhradové strategie vytvoříme novou populaci, kterou prohlásíme za příští generaci rodičů. Zde končí generační smyčka. Algoritmus můžeme zastavit, až nalezneme dostatečně dobré řešení či po určitém počtu ohodnocení.

Tělo EACBT:

- inicializace a ohodnocení počáteční populace
- generační cyklus:
    - rozděl populaci do dvou tříd: na kvalitní (OK) a nekvalitní (KO) část
    - na datech nauč rozhodovací strom
    - vytvoř potomky z listů stromu
    - mutace nových potomků
    - ohodnoť potomky
    - dle náhradové strategie zařaď potomky do populace
- opakuj, dokud není splněna ukončovací podmínka

3.2 Popis stromových modelů

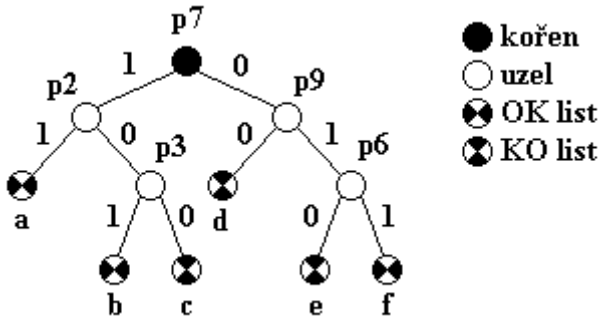
Stromové modely se využívají převážně pro klasifikační účely. Nejdříve, při jejich stavbě, se jim předkládají jedinci, o kterých jim řekneme do které kategorie patří[2], spolu s parametry které k jedinci patří; parametry mohou být jak kategoriální, tak regresní. A na základě těchto dat se snaží tvořit své uzly a listy s různými vahami tak, aby když jim po skončení procesu učení předložíme nového jedince, byly schopní správně rozpoznat jejich kategorii. Samozřejmě i zde jsou různá úskalí, která správnost modelu zhoršují, jako šumy v datech, možnost přeučení, nedostatečná či nesprávná trénovací množina dat apod. Více o CART stromech najdete v [3].

V mé implementaci jsou kategorie, mezi kterými chci rozpoznávat, kvalitní (OK) a nekvalitní (KO) jedinci. Jelikož testování probíhá pro binární reprezentaci, tak parametry jedince jsou pouze kategoriální; tedy binární 0 či 1. A jejich počet se odvíjí podle velikosti dimenze řešeného problému. Naučený strom je binární, tedy v každém uzlu (krom listů samozřejmě), se rozhoduje mezi dvěma stavy jednoho parametru jedinců. Přičemž v kořenovém uzlu je takový parametr, který je nejrelevantnější pro rozhodnutí do které kategorie patří. Jinými slovy kořenový uzel je takový uzel, který je schopen nejlépe oddělit OK a KO jedince. Čím hlouběji v stromu klesáme, tím více se tato relevantnost snižuje, až zůstanou pouze listy, ve kterých jsou zařazení jedinci buď pouze z jedné kategorie, nebo z druhé kategorie.

Například, mějme jedince, kterého v 10 dimenzionálním binárním prostoru chceme klasifikovat do jedné ze dvou kategorií, dle námi už vytvořeného stromu, viz obr. 2.

[2] Jedná se tedy o učení s učitelem.

Genotyp jedince bude řekněme [0 1 0 1 1 1 0 0 1 1].



Obrázek 2: Příklad klasifikačního CART stromu

Dle kořenového uzlu, který nám říká, že 7. parametr je v tomto případě nejrelevantnější se podíváme na našeho jedince a zjistíme, že se máme vydat větví vpravo, protože jeho hodnota je (binární) 0. Další uzel rozlišuje podle 9. parametru, který máme nastaven na (binární) 1, vydáme se tedy větví vpravo. A dle posledního uzlu, podle kterého se vydáme též do pravé větve, protože 6. parametr máme nastaven na hodnotu 1, zjistíme, že náš jedinec patří do listu „f“, tedy do „OK“ kategorie.

3.2.1 Obor listu

Obor listu, je definice oblasti stavového prostoru, kterou list pokrývá. Jinými slovy jedná se o cestu, kterou jsme museli projít od kořenového uzlu, abychom se do daného listu dostali. Jako příklad použijeme strom z obrázku 2. Chceme zjistit jaký obor má list „b“. Postupujeme od listu směrem ke kořenovému uzlu. Do rodičovského uzlu našeho listu se dostaneme tak, že 3. parametr musí být nastaven na hodnotu (binární) 1, pokračujeme dále a zjistíme, že druhý parametr musí mít hodnotu 0, až se ocitneme v kořenovém uzlu, který nám říká, že 7. parametr musí mít hodnotu 1.

Obor listu „b“ je tedy, pro 10 dimenzionální případ [-- 0 1 -- -- -- 1 -- -- --], s tím že „--“ jsou tam, kde hodnota příslušného parametru není definována. A tedy všechny možné kombinace genotypu jedinců, které budou mít na 2. místě (binární) 0, na 3. (binární) 1 a na 7. též 1, budou spadat do „OK“ kategorie, do listu „b“.

3.3 Experimenty a výsledky s naivním algoritmem

Cílem této BP je vzájemně porovnat EACBT, klasický EA a náhodné prohledávání a říct zda optimalizační algoritmus EACBT je přínosem či nikoliv. EACBT je implementován přesně tak, jak je popsán v 3.1. Klasický EA používá jednobodové křížení. U obou algoritmů byly testovány dvě náhradové strategie – RTR a SSNS, tak jak jsou popsány výše, viz 2.1.5. Pro **RTR** je velikost turnaje 10 a elitismus nepoužívá. **SSNS** používá velikost turnaje 8 a velikost elitismu

4. U obou algoritmů byla použita **mutace** s pravděpodobností $\frac{1}{D}$, kde D je velikost dimenze. Tedy nastavená tak, že v genotypu jedince se pravděpodobně prohodí 1 nějaký bit. Velikost populace je pro každou dimenzi jiná a její hodnota byla stanovena na $10 \cdot D$. **Maximální počet ohodnocení** je pro dimenze menší či rovny 20 určen vzorcem $\max eval = D^2 \cdot 1200$, pro dimenze vyšší vzorcem $\max eval = (D \cdot 20) \cdot 1200$. Další dva parametry u EACBT byli nastaveny na $oblastDS = 0{,}25$ a $pomerDS = 1$. Oblast DS určuje kde se má, před vytvořením stromu, seřazená populace „říznout“. Tedy jak velká část populace bude prohlášena za „OK“ a jak velká část za „KO“. Tento parametr nastaven na 0,25 nám říká, že 25% lepší části populace bude v kategorii OK a zbytek v kategorii KO. Poměr DS říká, kolik z celkového počtu nově generovaných jedinců bude vygenerovaných pomocí OK listů a kolik pomocí KO listů. Dle předběžných testů byl parametr poměr DS byl nastaven na 1, tedy všichni nově generovaní jedinci pocházejí z OK oblasti.

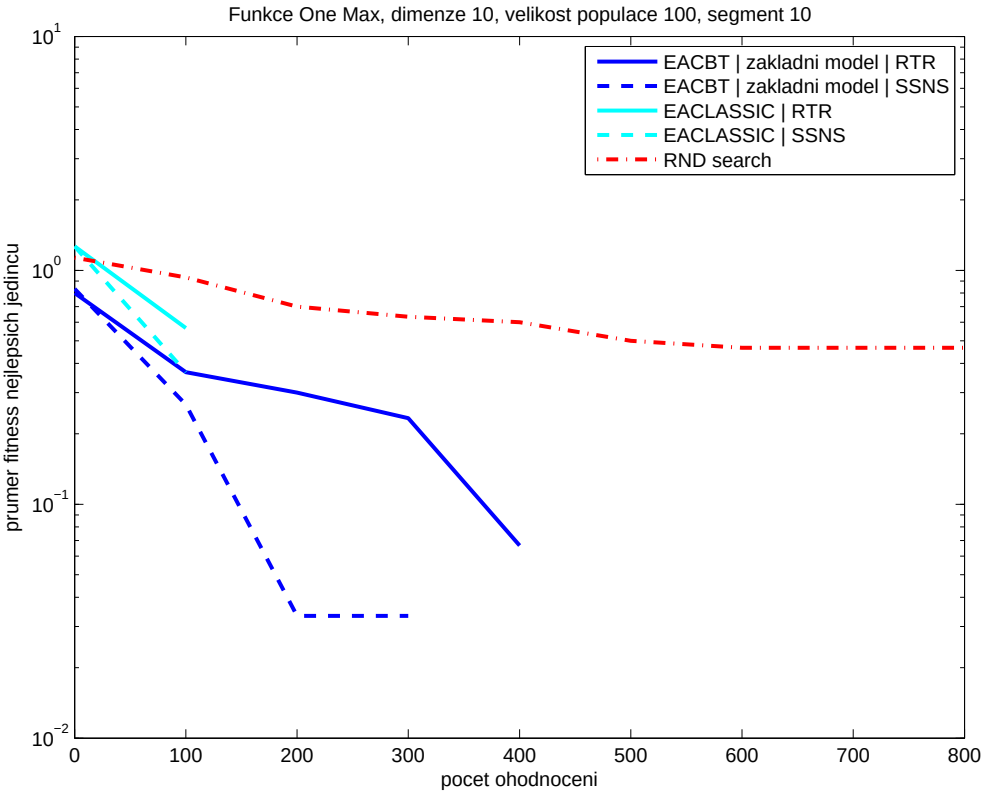
Testy probíhali na 5 funkcích: **One Max (OM), Equal Pairs (EP), Sliding XOR (SX), Trap (T) a Royal Road (RR)**. Jejich popis a definice naleznete v 1. příloze. U funkcí Equal Pairs a Sliding XOR jsem testoval jejich separovanou i neseparovanou verzi. Funkce Trap a Royal Road jsou v neseparované verzi pro optimalizaci příliš složité, proto jsem použil pouze verzi separovanou. Z definice One Max funkce v 1. příloze vyplývá, že na tuto funkci separovanost či neseparovanost nemá žádný vliv. Fitness jakéhokoliv jedince bude stejná pro obě verze. Pro všechny separované verze funkcí byla použita **velikost segmentu 5**. Všechny funkce mají **globální optimum v nule**, jedná se tedy o **minimalizaci**.

Velikost testovaných dimenzí tedy musí být dělitelná pěti, proto jsem zvolil velikosti 10, 15, 20 a 25. Pro získání nějakých statistických výsledků a zamezení tak vlivu vygenerování populace poblíž optima, byla optimalizace každé funkce a každé dimenze spuštěna celkem 30x.

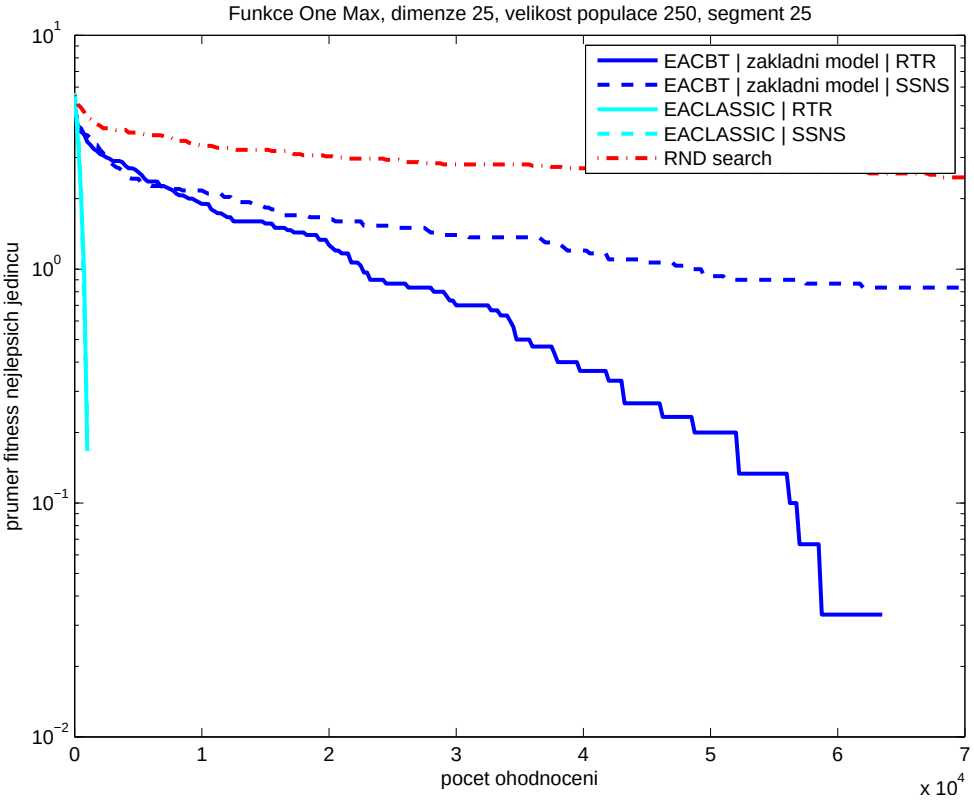
V grafech je na ose x vynesen počet ohodnocení a na ose y průměr fitness všech 30 dosud nejlepších nalezených jedinců. Jelikož má osa y logaritmické měřítko a globální optimum všech funkcí je tedy v nule a pokud se algoritmu povede ho nalézt ve všech 30 případech, poté tedy průměr všech nejlepších dosud nalezených jedinců je nula. Logaritmické měřítko neumí zobrazit nulu, a proto v grafech některé křivky najednou skončí.

Pro znázornění RTR náhrady jsem použil plnou čáru, pro znázornění SSNS jsem použil čárkovanou čáru. Pro odlišení EACBT a klasického EA jsem použil různé barvy, kde EACBT je znázorněn modrou barvou a klasický EA barvou azurovou. Náhodné prohledávání je znázorněno červenou čárko-tečkovanou čarou.

3.3.1 Grafy One Max funkce



Obrázek 3: Porovnání výkonu testovaných algoritmů na One Max funkci dimenze 10



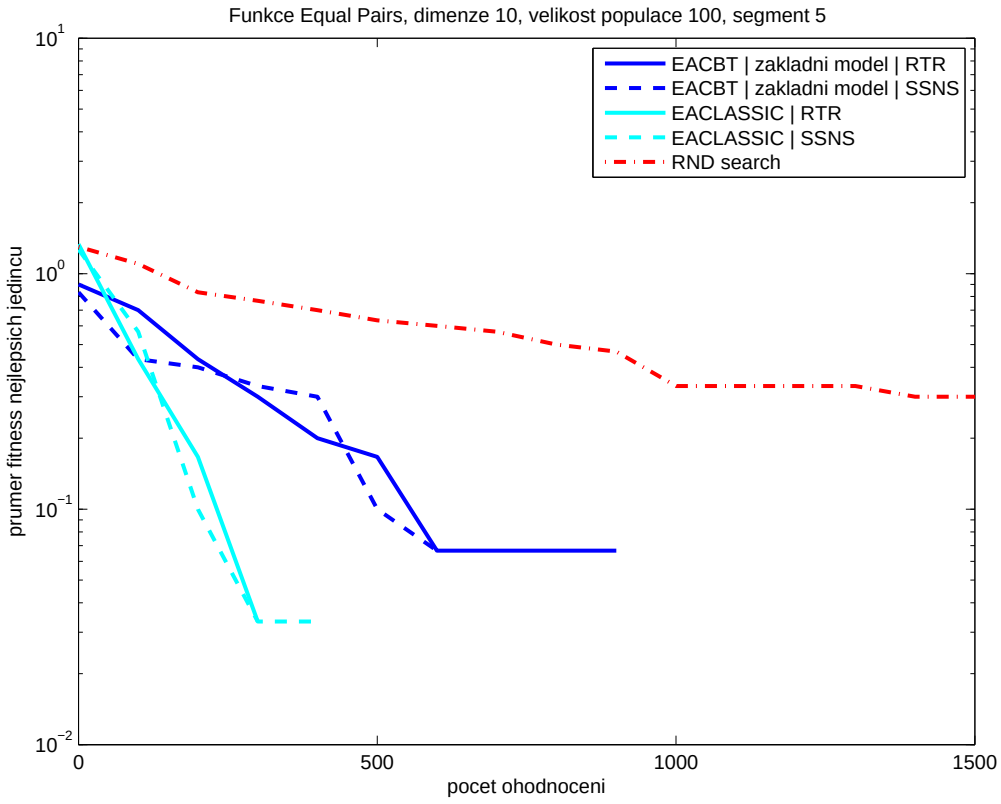
Obrázek 4: Porovnání výkonu testovaných algoritmů na One Max funkci dimenze 25

Graf z obrázku 3 nám ukazuje, že všechny algoritmy jsou schopné konvergence směrem k optimu. Ukazuje se taky, že na unimodální funkci, je SSNS rychlejší než RTR, která konverguje pomaleji a že klasický EA je rychlejší než EACBT a pro jednoduché problémy funguje efektivněji. Náhodné prohledávání je v tomto nejpomalejší. Nalézt spolehlivě optimum na 10 dimenzích mu v průměru trvá 4000 ohodnocení. Spolehlivě znamená pro alespoň 29 případů z 30.

Na obrázku 4 vidíme, že při větším počtu dimenzí se rozdíly mezi EA a EACBT prohlubují. V tomto případě změna náhradové strategie na klasický EA nemá velký vliv. Oproti tomu na EACBT je to rozdíl mezi schopností nalézt či nenalézt optimum. RTR náhrada je schopná spolehlivě nalézt optimum do 70 000 ohodnocení, kdežto SSNS nenalezla optimum v 11 případech z 30. Náhodné prohledávání dokázalo nalézt optimum pouze v jednom případě při maximálním počtu ohodnocení 600 000.

U grafů pro dimenze 15 a 20, které tu nejsou zobrazeny, je vidět jak pro EACBT algoritmus SSNS náhrada ztrácí na výkonu oproti RTR náhradě, až do bodu jak to vidíme u 25 dimenzí.

3.3.2 Grafy Equal Pairs funkce

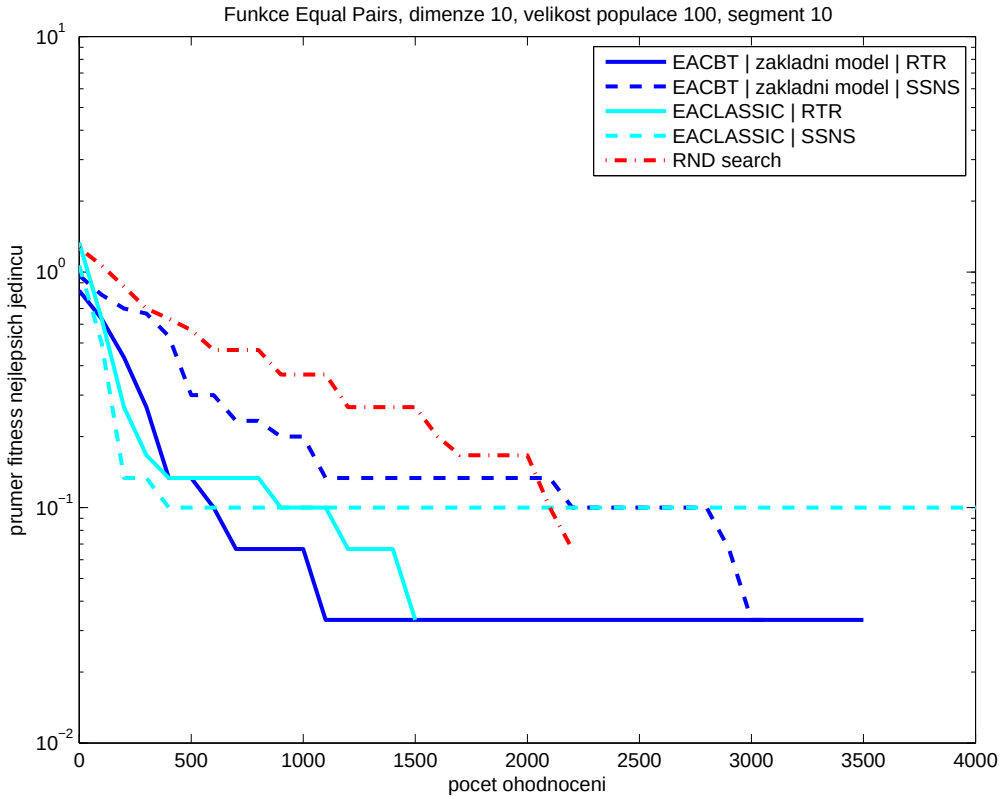


Obrázek 5: Porovnání výkonu testovaných algoritmů na Equal Pairs funkci dimenze 10

Separovaná verze funkce Equal Pairs má velmi obdobný průběh jako One Max funkce. S tím rozdílem, že vidíme, že optimalizace této funkce je o něco složitější a algoritmy tedy

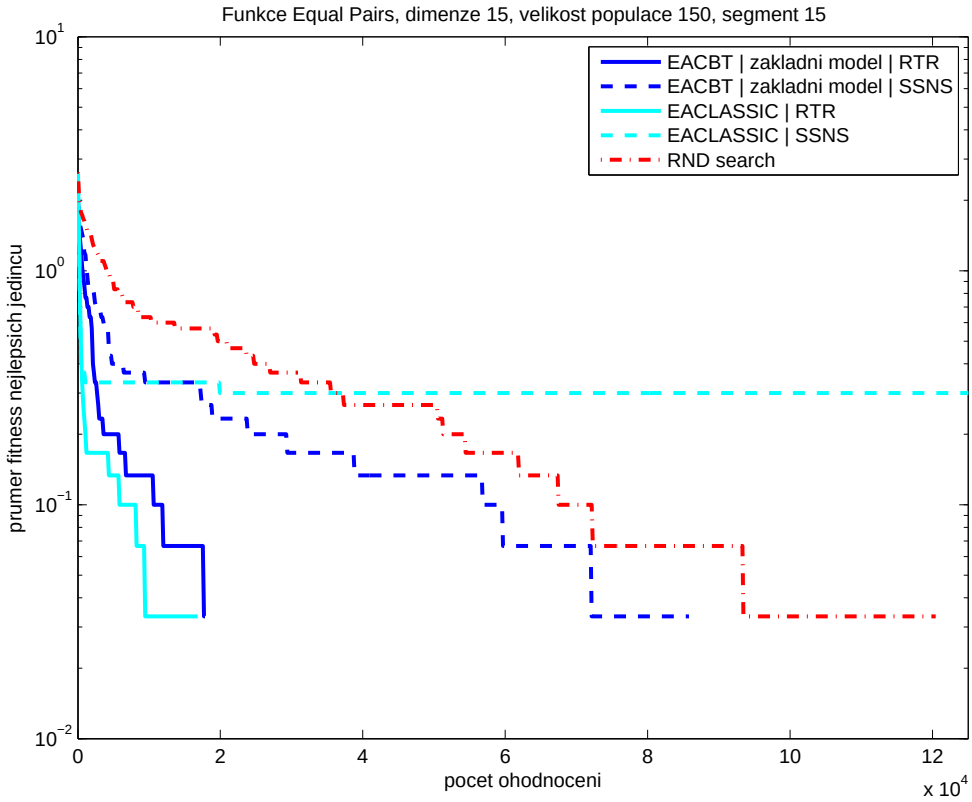
k nalezení optima potřebují větší počet ohodnocení, protože na rozdíl od One Max funkce, je tato funkce bimodální. Tedy je zde možnost uváznutí v lokálním optimu.

Neseparované verze funkce EP



Obrázek 6: Porovnání výkonu testovaných algoritmů na neseparované verzi funkce Equal Pairs dimenze 10

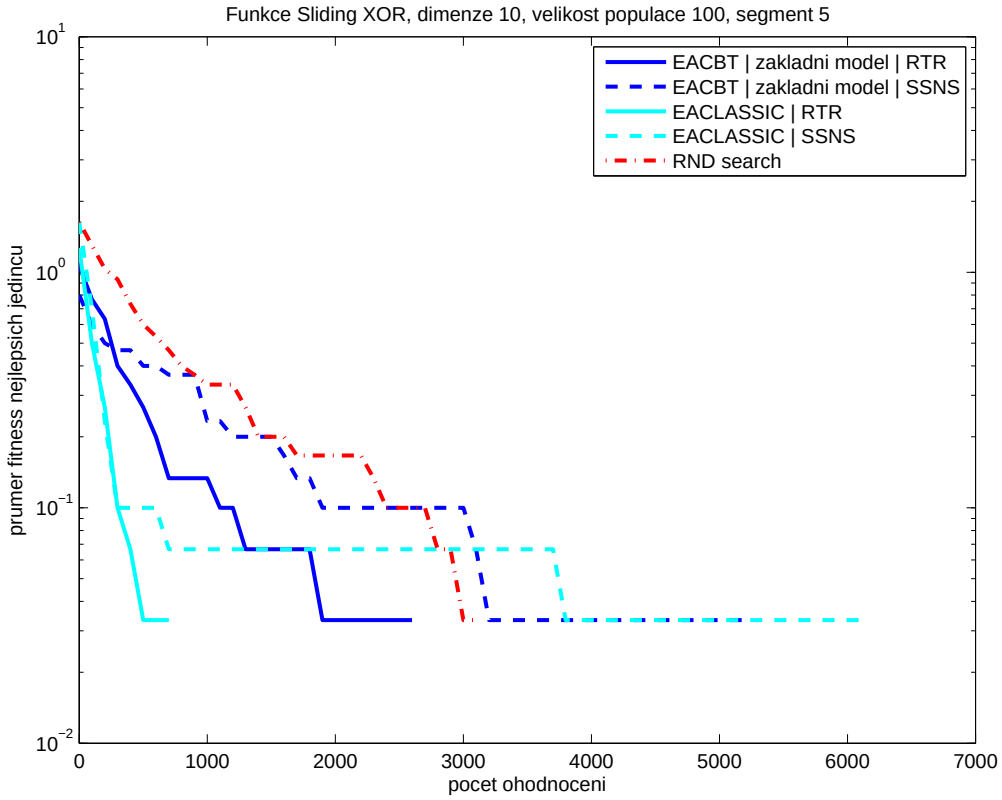
Na obrázku 6 je vidět, že RTR náhrada, která dokáže udržovat diverzitu populace déle než SSNS, zde dostává výrazně navrch. Klasický EA je stále nejrychlejší, ovšem jen s RTR náhradou, pro SSNS optimum nalézt dokázal též ve všech 30 případech, ovšem až při cca 23 000 ohodnocení. Zajímavé je, že náhodné prohledávání si vede velmi dobře a krom klasického EA s RTR porazila všechny ostatní případy.



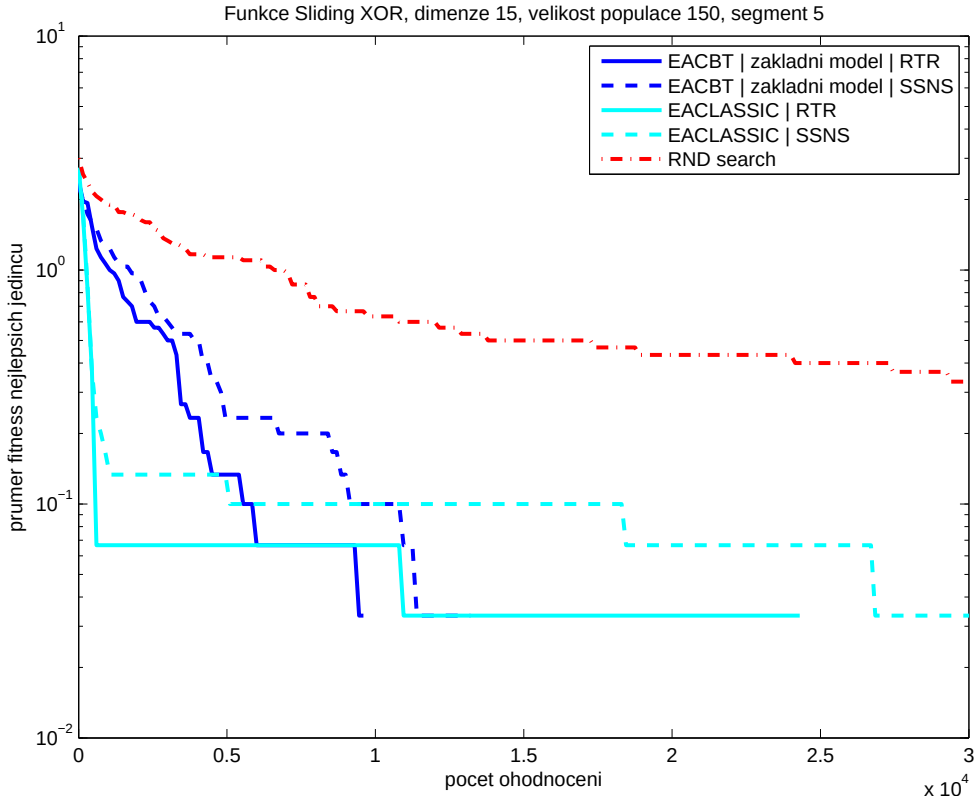
Obrázek 7: Porovnání výkonu testovaných algoritmů na neseparované verzi funkce Equal Pairs dimenze 15

Pro neseparovanou verzi funkce EP dimenze 15, jak je vidět na obrázku 7, vidíme, že EACBT a klasický EA si vedou s RTR strategií velmi obdobně. EACBT s SSNS konverguje o kousek rychleji než náhodné prohledávání. A klasický EA s SSNS dokonce v 8 z 30 případů uvázl v lokálním optimu.

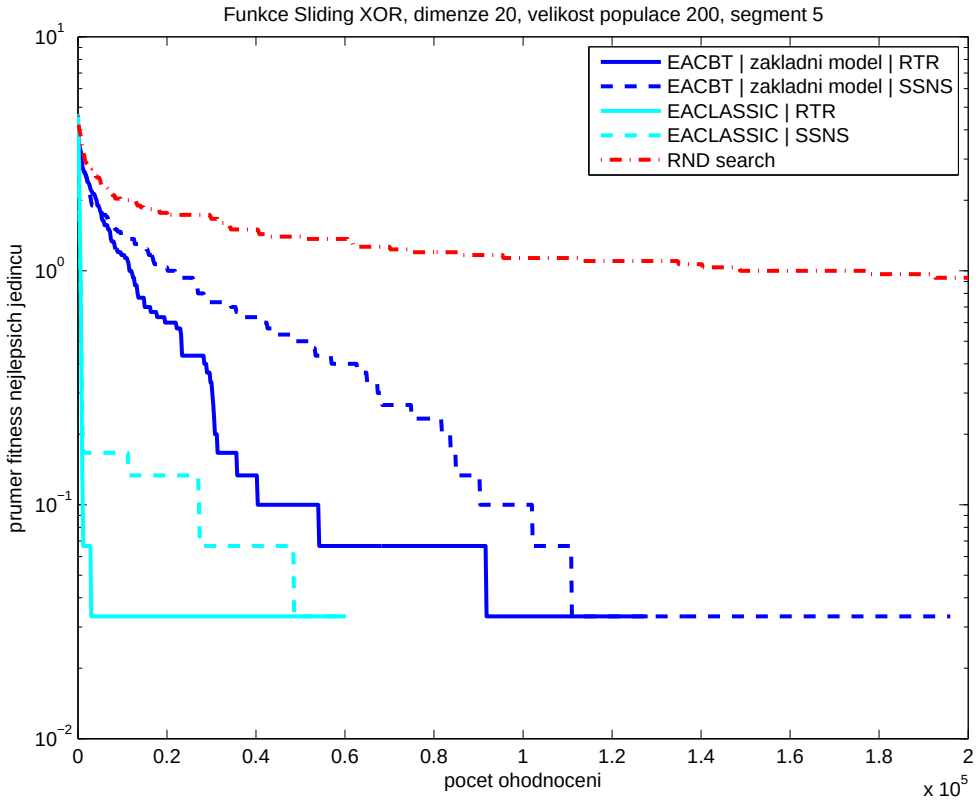
3.3.3 Grafy Sliding XOR funkce



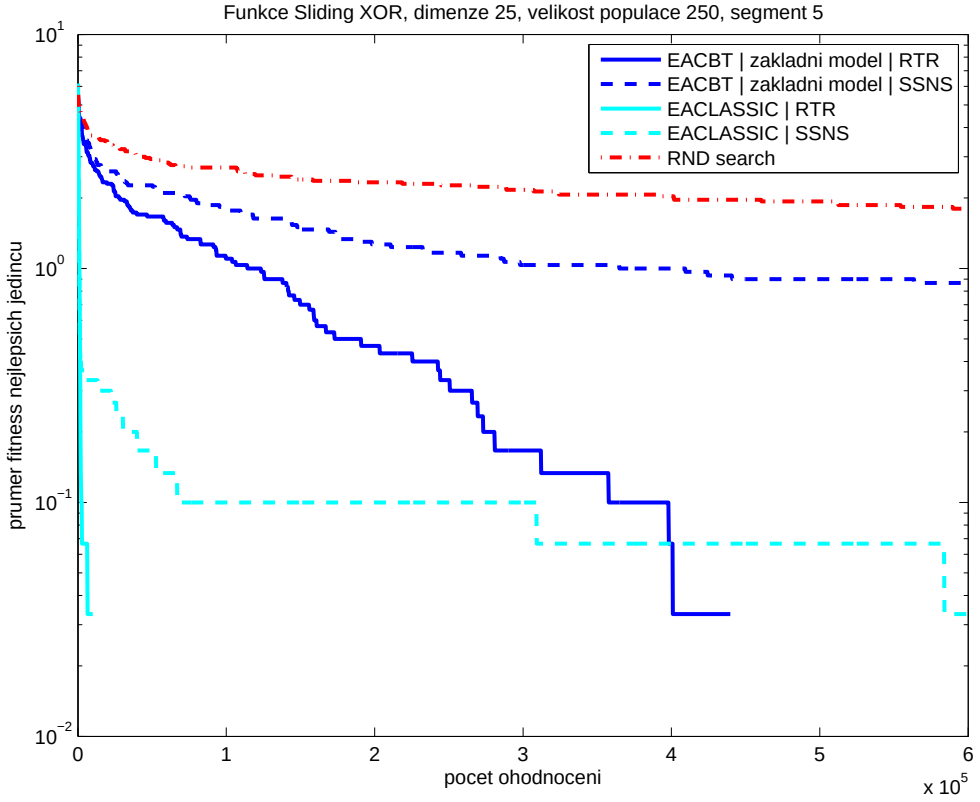
Obrázek 8: Porovnání výkonu testovaných algoritmů na Sliding XOR funkci dimenze 10



Obrázek 9: Porovnání výkonu testovaných algoritmů na Sliding XOR funkci dimenze 15



Obrázek 10: Porovnání výkonu testovaných algoritmů na Sliding XOR funkci dimenze 20

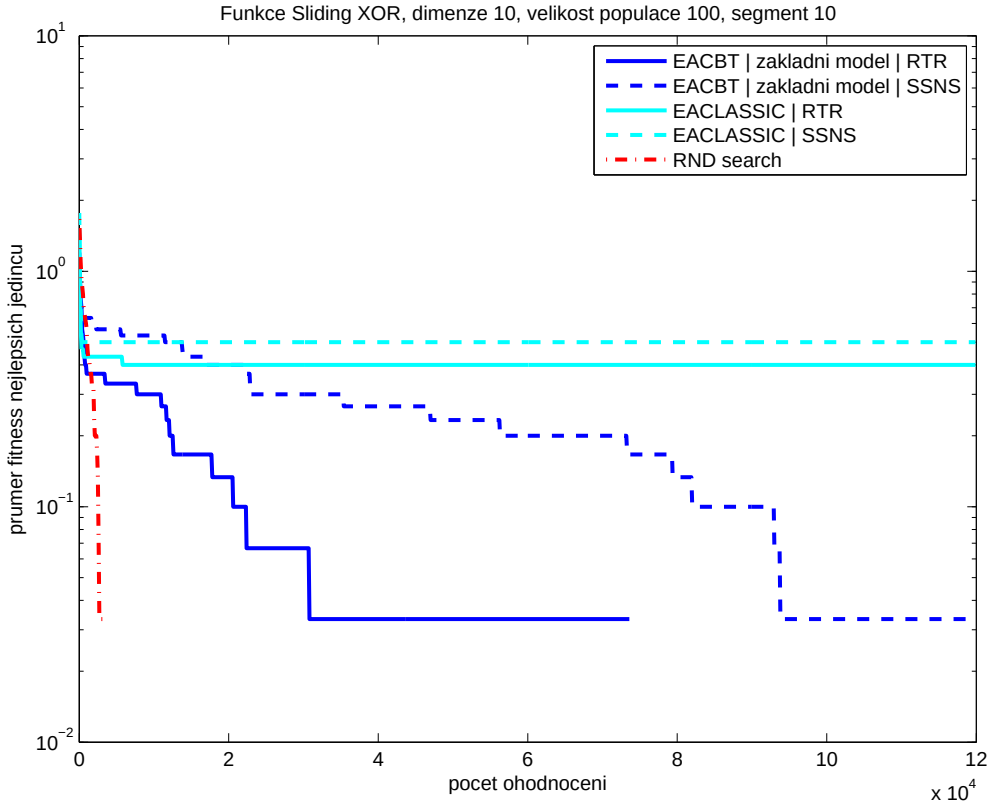


Obrázek 11: Porovnání výkonu testovaných algoritmů na Sliding XOR funkci dimenze 25

Na obrázcích 8 – 11 je vidět, že u SX funkce dokáží různé dimenze výrazně zahýbat s výkonností jednotlivých optimalizátorů. Obecně se pro všechny dimenze dá říci, že EACBT s SSNS podává oproti ostatním algoritmům velice slabé výsledky a že klasický EA s RTR

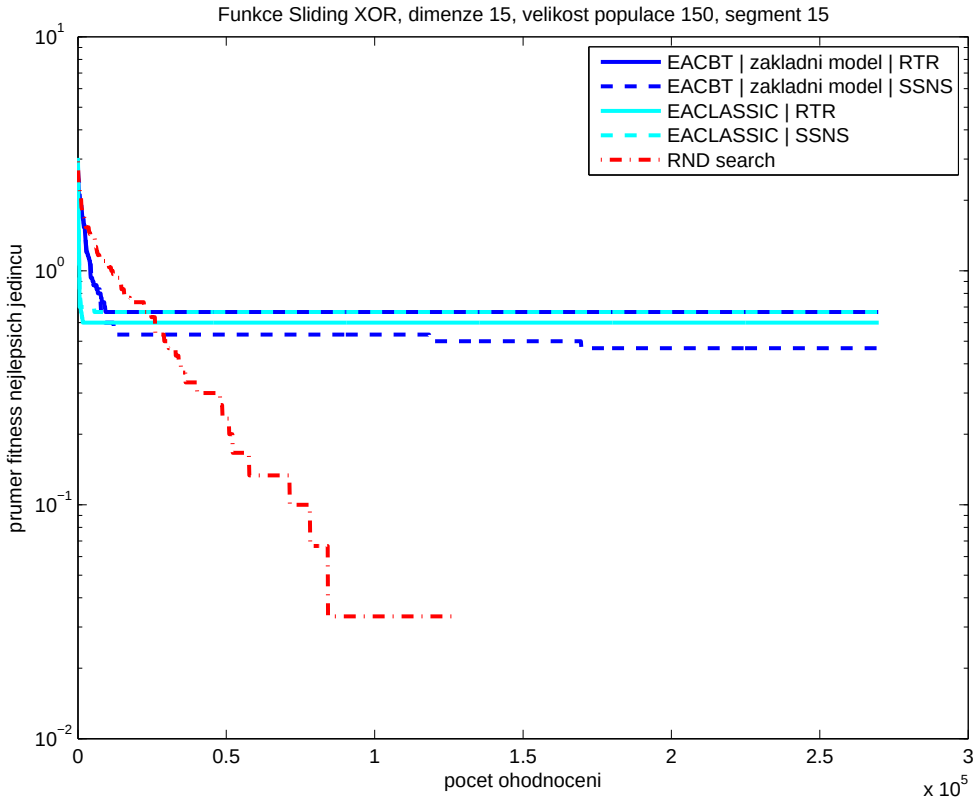
náhradou si vede naopak velice dobře. Náhodnému prohledávání se už pro větší dimenze než 10 či 15 daří nalézat globální optimum velmi obtížně. Klasický EA s SSNS podává pro různé dimenze velmi nevyrovnané výsledky. Pro dimenze 10 a 15 patří mezi nejhorší u dimenze 20 dokázal nalézt optimum rychleji ve všech 30 případech než EA s RTR a pro dimenzi 25 patří opět mezi nejhorší. EACBT s RTR podává též rozdílné výsledky. Pro menší dimenze 10 a 15 si vede obstojně (pro dimenzi 15 dokonce nejlépe), ale pro dimenze 20 a 25 už mu nalezení optima trvá výrazně déle než klasickému EA.

Neseparované verze funkce SX



Obrázek 12: Porovnání výkonu testovaných algoritmů na neseparované verzi funkce Sliding XOR dimenze 10

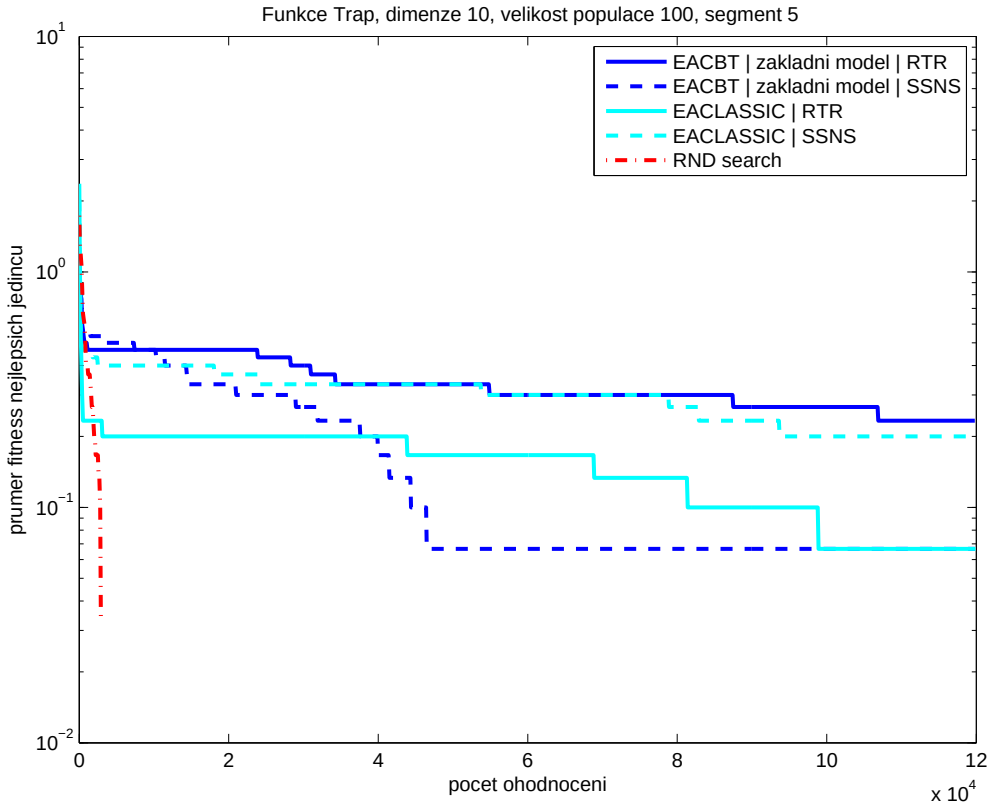
Na obrázku 12, vidíme, že optimalizace neseparované verze Sliding XOR funkce není vůbec jednoduchá a nejlépe se zde daří náhodnému prohledávání, které se tvarem funkce neřídí. Použití EACBT či klasického EA v tomto případě znamená rozdíl mezi nalezením či nenalezením globálního optima. Klasický EA uvázl v lokálním optimu pro RTR náhradu ve 12 z 30 případů a pro SSNS v 15 z 30 případů. EACBT s RTR nalezl optimum vždy a s SSNS také vždy, krom jediného případu.



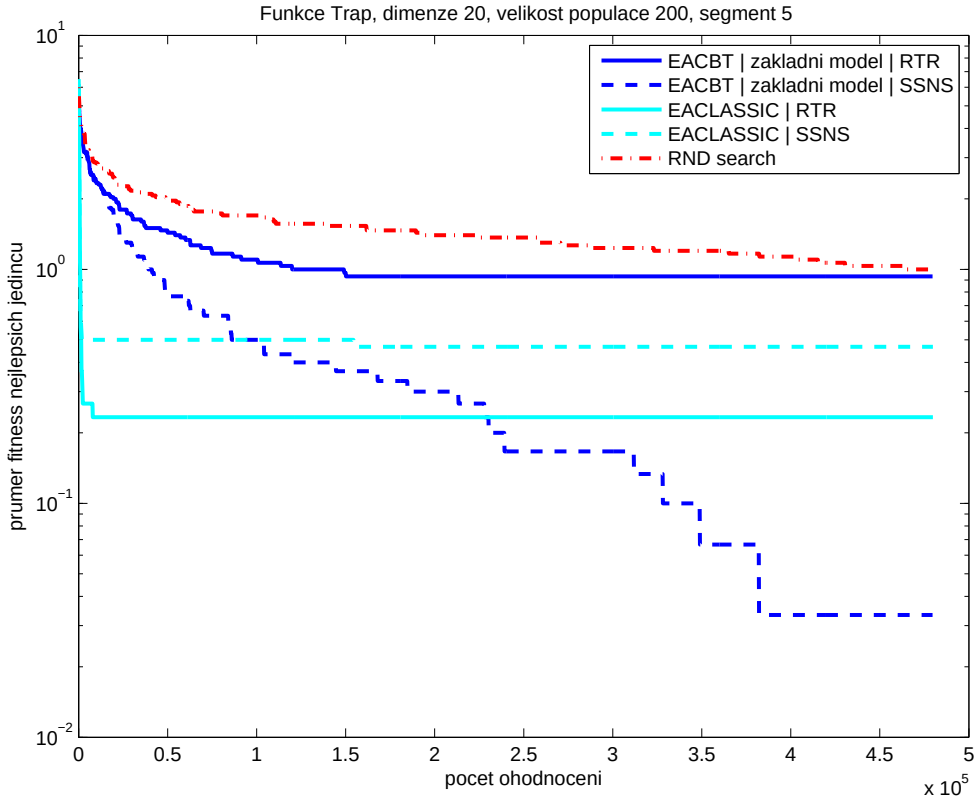
Obrázek 13: Porovnání výkonu testovaných algoritmů na neseparované verzi funkce Sliding XOR dimenze 15

Na obrázku je vidět, že s neseparovanou funkcí SX mají všechny algoritmy sledující hodnotu funkce velké problémy a jejich výkon je velmi podobný. Optimum dokázali nalézt pouze v cca 11 z 30 případů. Pouze EACBT s SSNS se to podařilo v 16 z 30 případů. Náhodné prohledávání hodnotu funkce nesleduje a do cca 120 000 ohodnocení se mu pro 15 dimenzionální případ daří vygenerovat jedince v globálním optimum spolehlivě.

3.3.4 Grafy Trap funkce



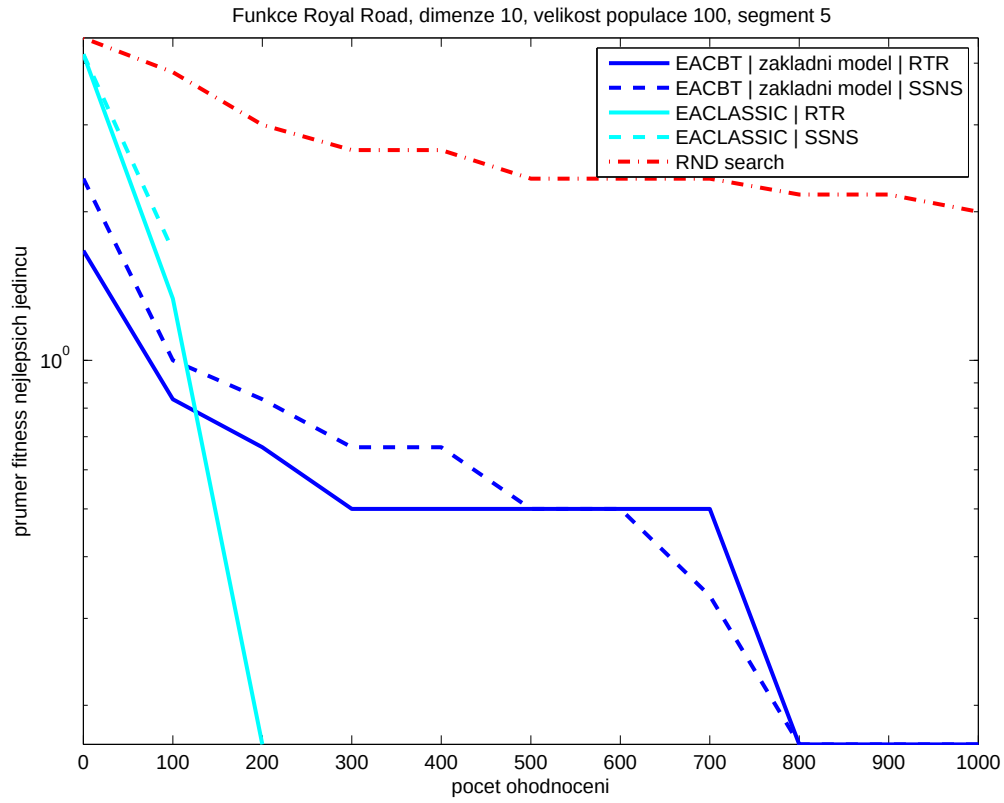
Obrázek 14: Porovnání výkonu testovaných algoritmů na Trap funkci dimenze 10



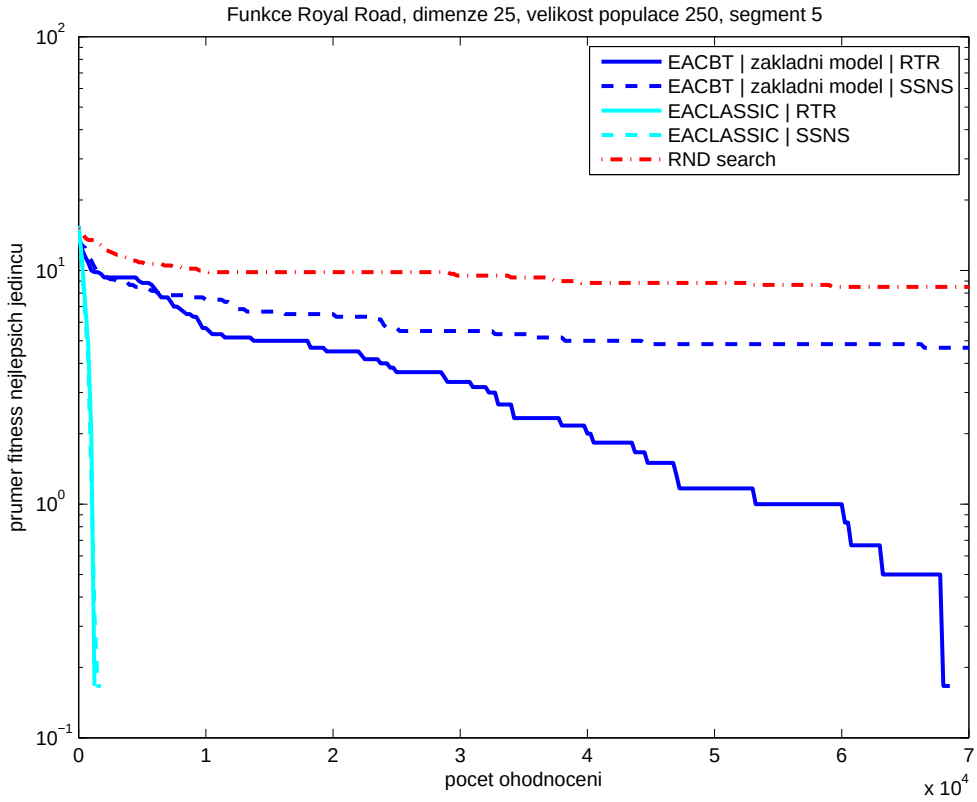
Obrázek 15: Porovnání výkonu testovaných algoritmů na Trap funkci dimenze 20

Na obrázcích 14 a 15, kde vidíme grafy funkce Trap je zajímavé, že kde lépe funguje RTR náhrada na jednom algoritmu, tak na druhém funguje lépe SSNS. Pro dimenzi 15 mají algoritmy obdobný průběh jako u dimenze 10. Náhodné prohledávání je pro malé dimenze 10 a 15 nejlepší. Ovšem pro dimenzi 20 je už stavový prostor funkce příliš veliký a maximální počet ohodnocení příliš malý. Optimum dokázalo nalézt jen v 4 případech. Pro klasický EA si RTR náhrada, která udržuje diverzitu populace více než SSNS, vede lépe, ovšem nakonec mají obě náhrady tendenci uváznout v lokálním optimu. EACBT s RTR si vede obdobně jako náhodné prohledávání, ale téměř vůbec nekonverguje, kdežto SSNS i u této funkce dokázal optimum nalézt ve 29 z 30 případů.

3.3.5 Grafy Royal Road funkce



Obrázek 16: Porovnání výkonu testovaných algoritmů na Royal Road funkci dimenze 10



Obrázek 17: Porovnání výkonu testovaných algoritmů na Royal Road funkci dimenze 15

Dle obrázků 16 a 17 je průběh algoritmů na funkci Royal Road obdobný jako průběh na One Max funkci s tím, že se její hodnota nemění s každým jedničkovým bitem, ale vždy až s najednou pěti jedničkovými bity, nalezení optima tedy trvá všem algoritmům (kromě náhodného prohledávání) o něco déle. Klasickému EA se zde vede nejlépe a mezi RTR a SSNS nejsou téměř žádné rozdíly. U EACBT se RTR náhradě daří lépe na větších dimenzích, na menších dimenzích si vede obdobně jak SSNS.

3.4 Diskuze výsledků

Náhodné prohledávání v našem případě funguje pro danou dimenzi na všech funkcích stejně. Jelikož každá zde použitá funkce má optimum v případě, že je řetězec složený ze samých nul či ze samých jedniček, algoritmus potřebuje akorát dostatečný počet ohodnocení k jeho vygenerování. Parametry jsou nastavené tak, že pro malé dimenze je náhodné prohledávání schopné nalézt optimum spolehlivě (alespoň v 29 z 30 případů) pro dimenzi 10 do cca 4 000 případně pro dimenzi 15 do cca 120 000 ohodnocení. Naopak pro dimenzi 20 téměř nikdy, protože v tomto případě je algoritmus schopný spolehlivě nalézat řešení okolo 2 000 000 ohodnocení a u dimenze 20 je maximální počet ohodnocení nastaven na 480 000.

Co se EACBT algoritmu týče, zdá se, že RTR náhrada je na většině funkcí ku prospěchu věci. Toto bude zřejmě způsobené tím, že aby model stromu dokázal dobře odlišit kvalitní a nekvalitní jedince, potřebuje k tomu mimo jiné také vhodně zvolenou trénovací množinu

jedinců, na kterých se je naučí správně rozeznávat. Vhodnost trénovací množiny ovlivňuje několik kritérií.

Prvním kritériem je počet trénovacích jedinců, kde platí čím více, tím lépe. Kdybychom vygenerovali celý stavový prostor, tedy všechny kombinace nul a jedniček pro danou dimenzi, následně bychom tyto jedince ohodnotili a lepší část označili za dobrou a horší část za špatnou a na této sadě dat bychom naučili model stromu rozpoznávat dobré od špatných, uměl by to se 100% přesností. Samozřejmě to by se rovnalo úplnému prohledání stavového prostoru, proto je trénovací množina, tedy pro nás velikost populace, mnohem menší.

Druhým kritériem je rozložení trénovacích dat ve stavovém prostoru. Čím rovnoměrněji, tím lépe. K tomu, aby se model naučil dobře rozpoznávat charakteristiky kvalitních jedinců, potřebuje nejen „vidět“ nějaké kvalitní jedince, ale i protipříklady, které mu naopak řeknou, jak kvalitní jedinec vypadat nemá.

Jelikož při testování zůstává pro danou dimenzi velikost populace konstantní, má zde větší váhu druhé kritérium. Nejrovnoměrněji rozložená populace je pravděpodobně na začátku, kdy je náhodně vygenerovaná, protože postupem času populace konverguje směrem, kterým klesá hodnota fitness funkce. A jelikož RTR náhrada svou podstatou dokáže udržet diverzitu populace více než SSNS, EACBT díky ní dosáhne efektivnější optimalizace, protože jak je popsáno výše, staví se díky ní kvalitnější modely stromů.

Klasickému EA RTR náhradová strategie též pomáhá, ale z jiného důvodu. Správné nastavení diverzity populace je jedním z klíčových prvků ke správné optimalizaci. Kdyby klasický EA udržoval populaci s příliš velkou diverzitou, nikam by nekonvergoval, byl by tedy podobný náhodnému prohledávání. Pokud by jí udržoval příliš malou, stal by se z něj algoritmus podobný lokálnímu prohledávání. Správné udržení diverzity populace je tedy klíčové pro schopnost neuváznout v lokálním optimu, ale zároveň pro schopnost konvergence do globálního optima.

U SSNS se tedy zdá, že udržuje diverzitu populace příliš malou na to, aby byli algoritmy EACBT i EA schopné u složitějších funkcí globální optimum nalézat. Na unimodálních funkcích je ovšem tato náhrada pro klasický EA výhodnější, protože zde není možnost uváznutí v lokálním optimu a rychlá ztráta divergence je zde ku prospěchu věci.

SSNS u EACBT je ve většině případů k neprospěchu věci, z důvodů popsaných výše. Ovšem zdá se, že na Trap funkci funguje naopak velmi dobře a pro větší dimenze je jako jediný schopný konvergence do globálního optima. Důvod by mohl být ten, že naučený model stromu, je naučený špatně a neumí dobře rozpoznávat kvalitní jedince od nekvalitních. Na většině zde testovaných funkcí to znamená, zhoršení schopnosti optimalizace, protože tvorba správných

modelů stromů je v tomto případě zásadní. Ale jelikož Trap funkce je klamná funkce a tedy klesá směrem od globálního optima, je v tomto případě to, že se strom naučí rozpoznávat „špatně“, ku prospěchu věci, protože ve skutečnosti je to pro tuto funkci vlastně „dobře“. Přesto musí algoritmus nějakou schopnost konvergence mít, protože jinak by to bylo obdobné jako náhodné prohledávání. SSNS u EACBT tedy na klamné funkci naopak diverzitu populace zvětšuje a zdá se, že výsledné nastavení je vhodné tak, že dokáže dokonvergovat do globálního optima i pro Trap funkci.

Přesto se zdá, že EACBT algoritmus je oproti klasickému EA či oproti LEM algoritmu v optimalizaci pomalejší. Po důkladnějším prozkoumání algoritmu se ukázalo, že není schopen stavět dostatečně hluboké stromy, tak aby dostatečně přesně vymezil „OK“ oblast, tedy oblast, která by mohla obsahovat globální optimum. Proto se rozhodlo, že zkusíme vzít v potaz i zkušenosti minulých generací, obdobně jak je tomu u LEMu, které by mohli pomoci tuto OK oblast vyměřit lépe a zhodnotit zda je EACBT s pamětí zlepšením či nikoliv. Více ve 4. kapitole.

4. Algoritmus s pamětí

4.1 Popis změn v algoritmu

Způsobů jak využít zkušeností z předchozích generací EACBT algoritmu je více. Jedním z nich může být takový, že pro učení modelu stromu v dané generaci využijeme populací, které byly použity k vytvoření stromů předchozích. Dalším způsobem může být, že využijeme již vytvořené stromy ke generování nových jedinců. Tento druhý způsob jsem se rozhodl implementovat a otestovat.

Jak je řečeno v kapitole 3.1 a 3.3, nový jedinci se generují z OK listů stromu z aktuální generace a rovnou se vkládají do nové populace. U EACBT s pamětí je tomu trochu jinak. Nový jedinci se též vytvoří z OK listů stromu z aktuální generace, ovšem nevkládají se rovnou do nové populace, ale nejdřív se každý z nich ohodnotí pomocí jednoho či více stromů z posledních generací. Pokud všechny tyto stromy prohlásí, že patří do „OK“ kategorie, přesuneme ho do nové populace. Pokud jediný strom jedince vyhodnotí jako „KO“ kategorii, jedince zahodíme a vygenerujeme nového. Minulé generace stromů tedy v podstatě fungují jako filtr nově generovaných jedinců. Počet stromů, které takto využijeme, závisí na vstupním parametru.

4.2 Experimenty a výsledky

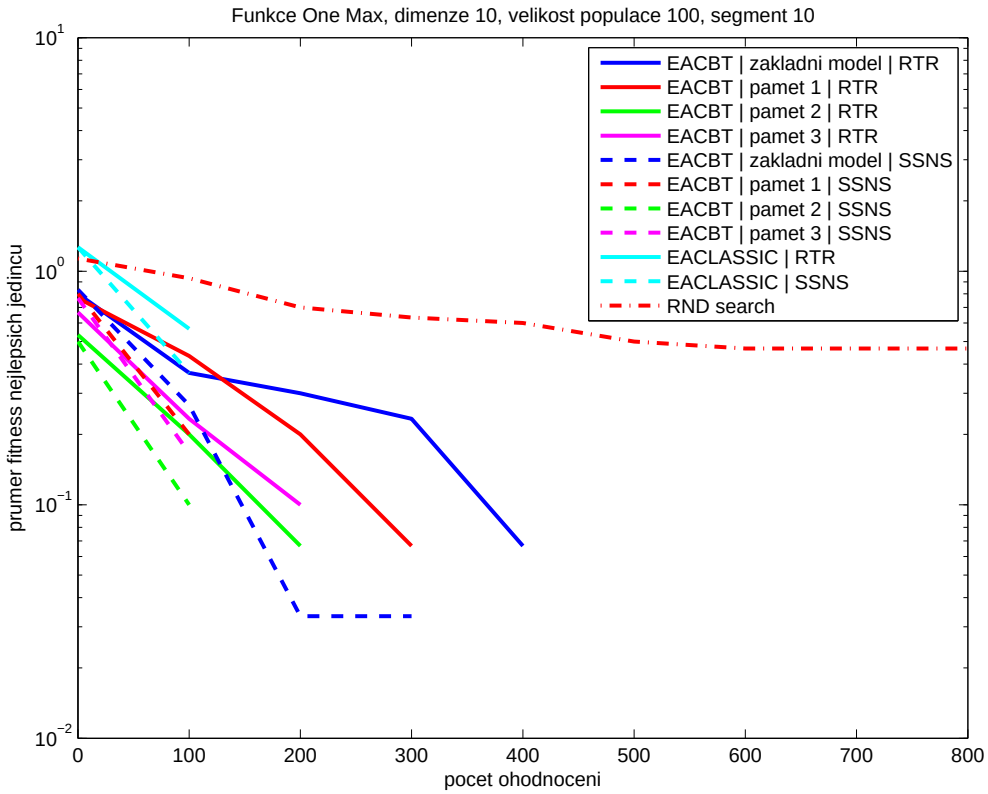
Nastavení a parametry algoritmů jsou stejné tak, jak jsou popsané v kapitole 3.3. S rozdílem, že zde je navíc vstupní parametr určující počet stromů použitých k filtraci nových jedinců. Tedy v podstatě velikost paměti. Jelikož optimální velikost tohoto parametru není známa, použil jsem hodnoty 1, 2 a 3. Nové křivky byly vloženy do grafů z předchozí kapitoly,

zaměřím se tedy zejména na popis EACBT s pamětí, protože zbylé křivky jsem komentoval už výše.

Pro odlišení různé velikosti paměti jsem též použil barvy. Pro paměť 1 je to červená, pro paměť 2 zelené a pro paměť 3 purpurová.

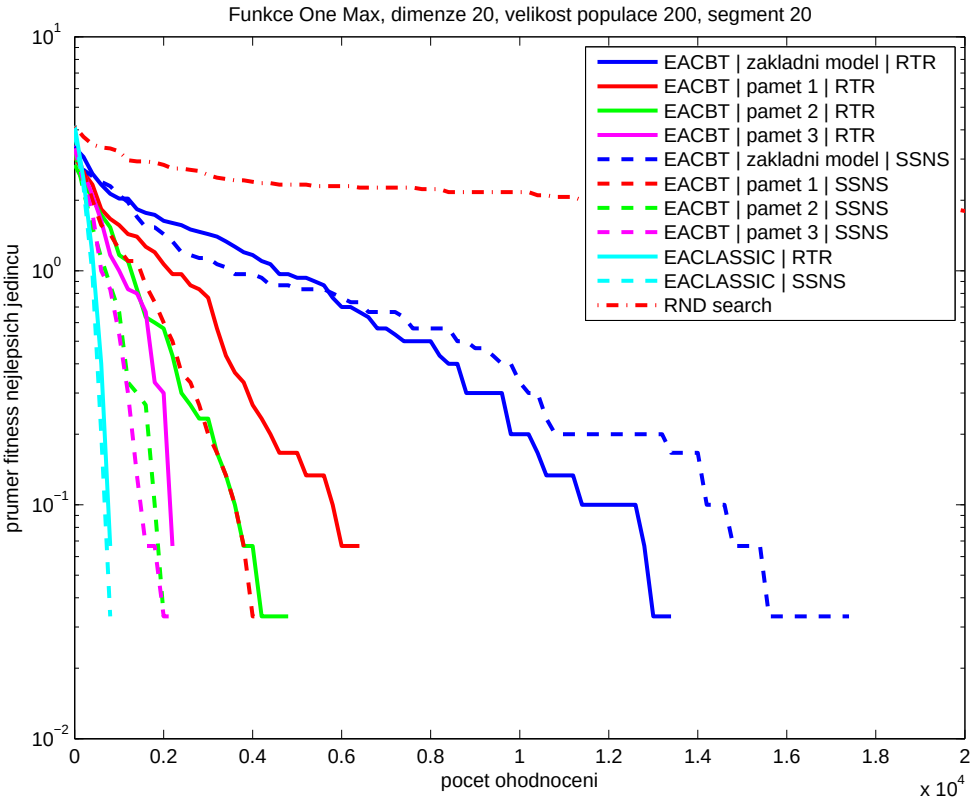
Z důvodu úspory času bylo na základě výsledků pro dimenze 10 až 20 rozhodnuto, že pro dimenze vyšší se bude testovat pouze paměť o velikosti 2. Více viz níže.

4.2.1 Grafy One Max funkce

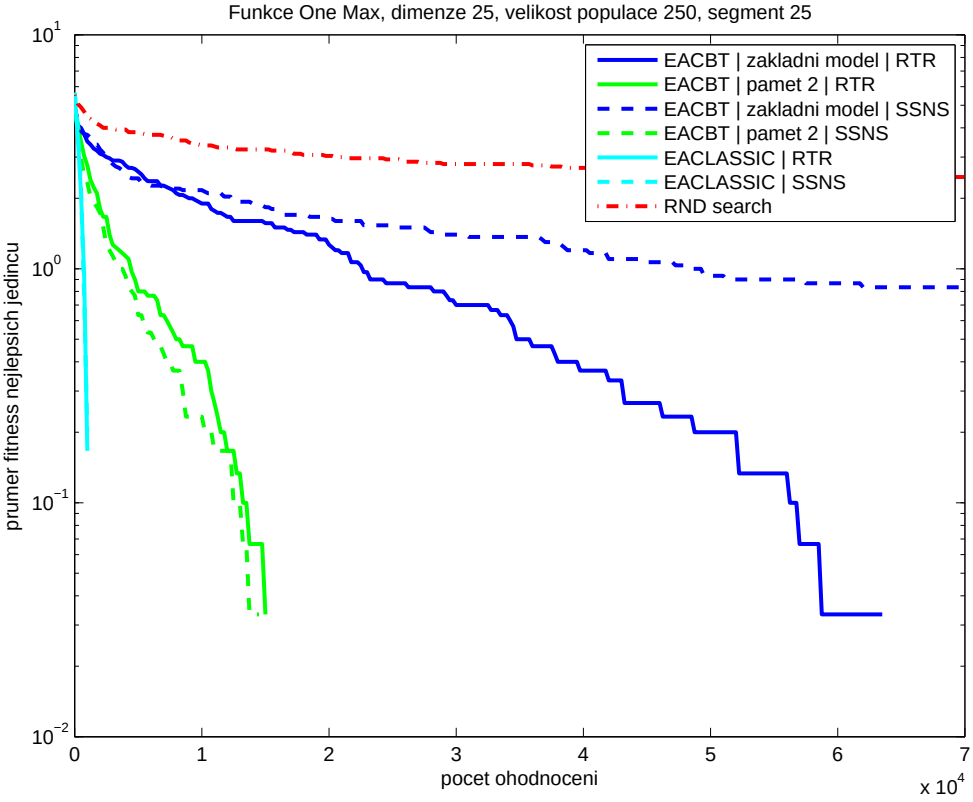


Obrázek 18: Porovnání výkonu testovaných algoritmů na One Max funkci dimenze 10

Z obrázku 18 vyplývá, že použití paměti zřejmě má smysl, protože si vede lépe než základní model. Snižují se také rozdíly mezi SSNS a RTR náhradou. Ovšem jednotlivé velikosti paměti na tomto obrázku nevykazují příliš veliké rozdíly.



Obrázek 19: Porovnání výkonu testovaných algoritmů na One Max funkci dimenze 20

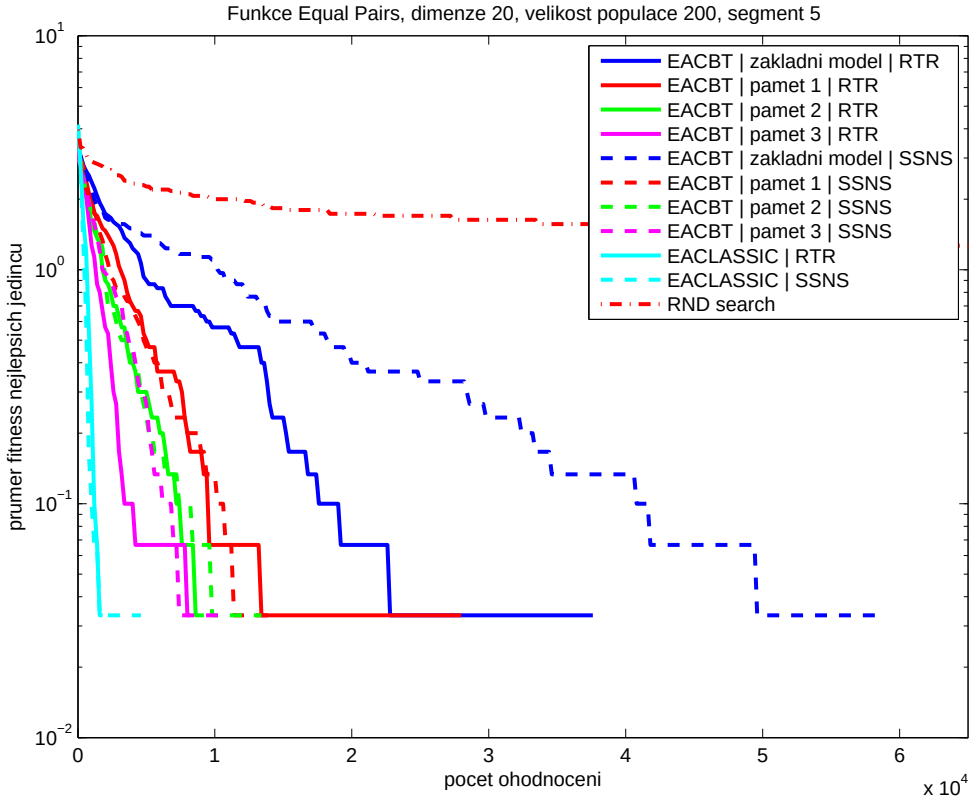


Obrázek 20: Porovnání výkonu testovaných algoritmů na One Max funkci dimenze 25

Z obrázku 19 vidíme, že jednotlivé velikosti paměti jsou si stále velice blízké i pro vyšší dimenze, ovšem ukazuje se, že přeci jen paměť velikosti 1 je málo. Na obrázku 20 jasně vidíme,

že paměť EACBT algoritmu zřetelně pomáhá, že rozdíl mezi RTR a SSNS není téměř žádný, ale také to, že to stále není dost na to, aby na unimodální funkci překonal klasický EA.

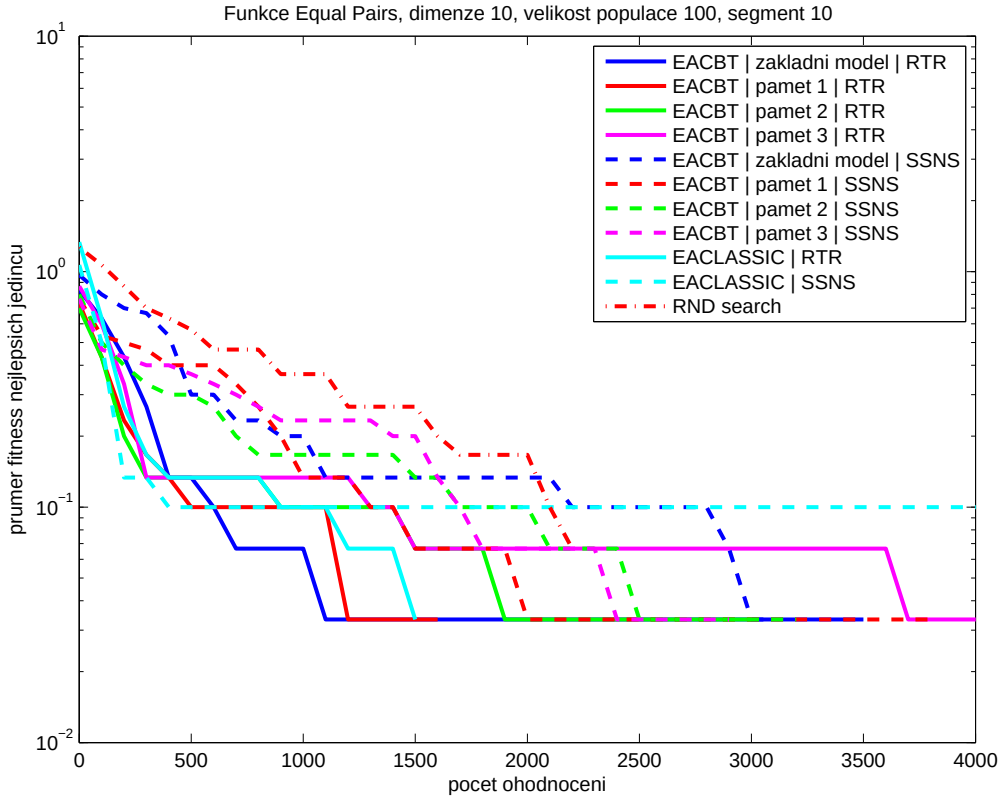
4.2.2 Grafy Equal Pairs funkce



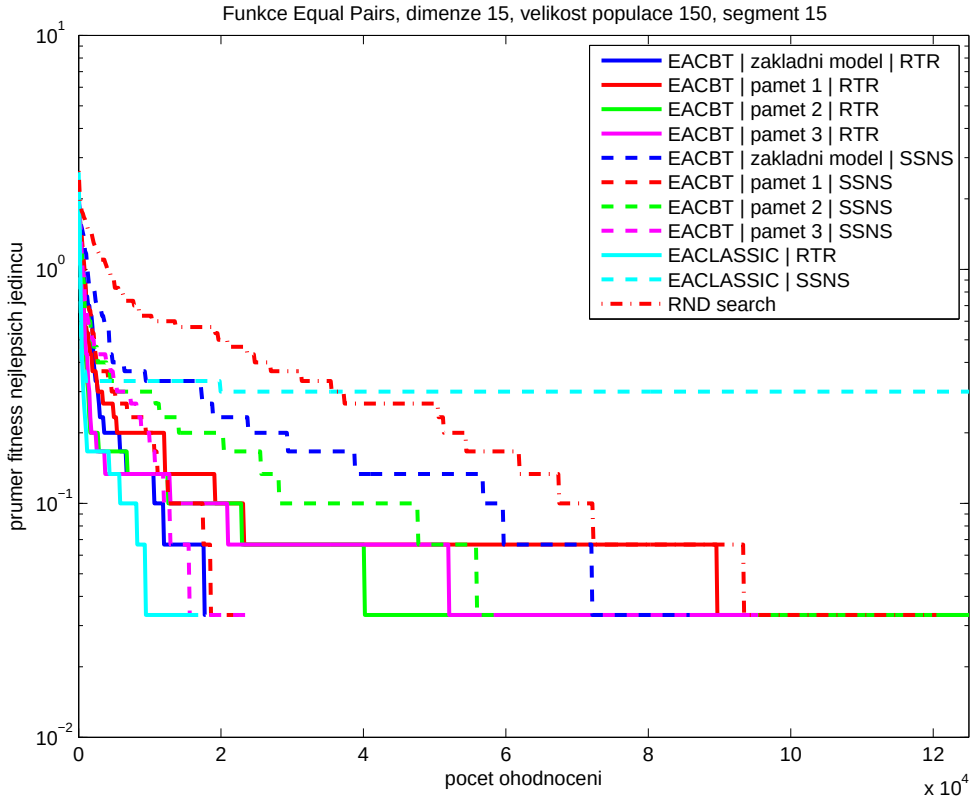
Obrázek 21: Porovnání výkonu testovaných algoritmů na Equal Pairs funkci dimenze 20

Obrázek 21 ukazuje, že výsledky jsou obdobné jako u One Max funkce (totéž platí i pro dimenzi 10 a 15) s tím, že mezi různými velikostmi paměti rozdíly jsou už opravdu velmi malé. Pro dimenzi 25 jsou rozdíly mezi pamětí 2, klasickým EA, EACBT v základním modelu a náhodným prohledáváním obdobné jako pro dimenzi 20.

Neseparované verze funkce EP



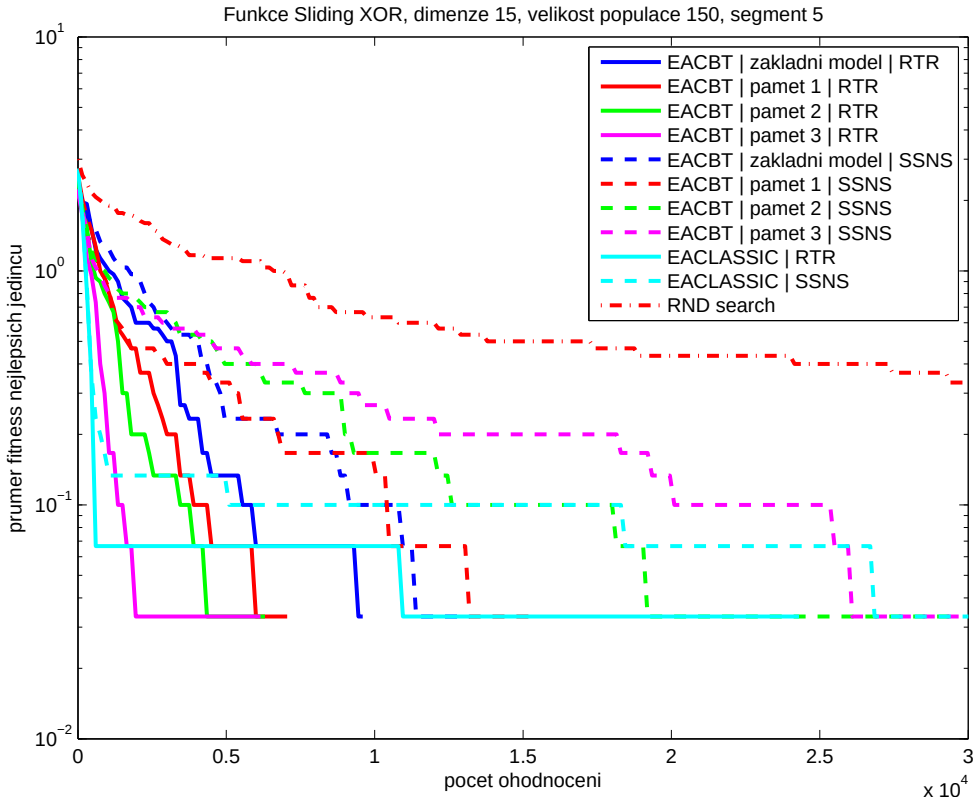
Obrázek 22: Porovnání výkonu testovaných algoritmů na neseparované verzi funkce Equal Pairs dimenze 10



Obrázek 23: Porovnání výkonu testovaných algoritmů na neseparované verzi funkce Equal Pairs dimenze 15

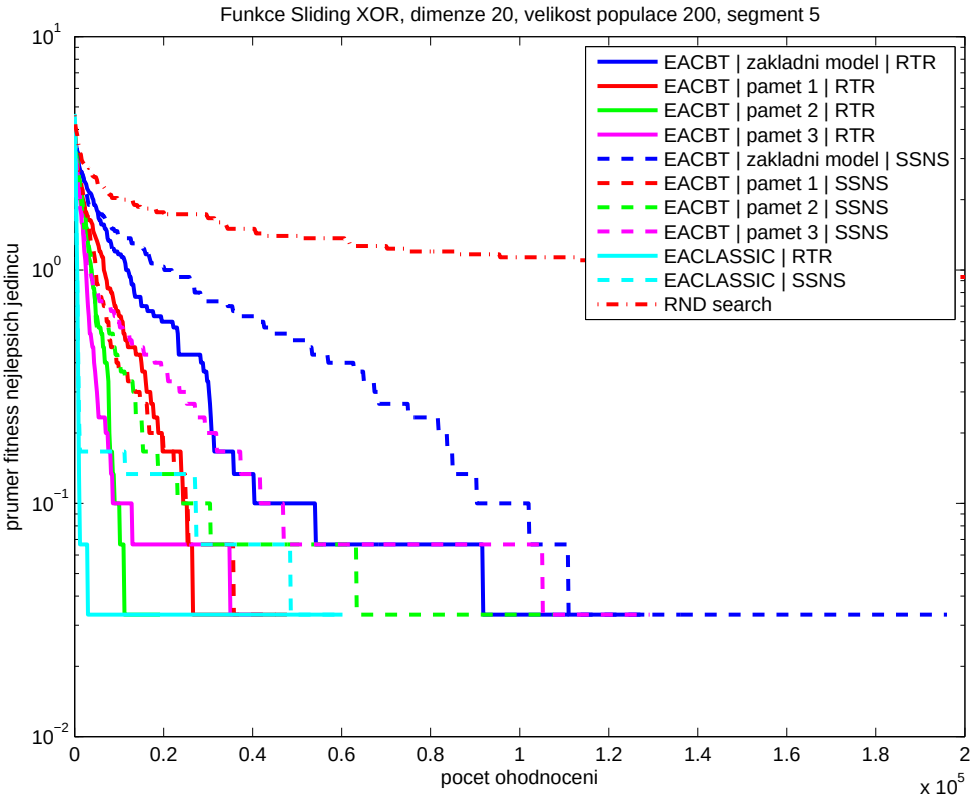
Z grafů na obrázcích 22 a 23 pozorujeme, že paměť pro EACBT s RTR je převážně na škodu a že s SSNS s pamětí je obdobně výkonný jako základní model EACBT s RTR.

4.2.3 Grafy Sliding XOR funkce



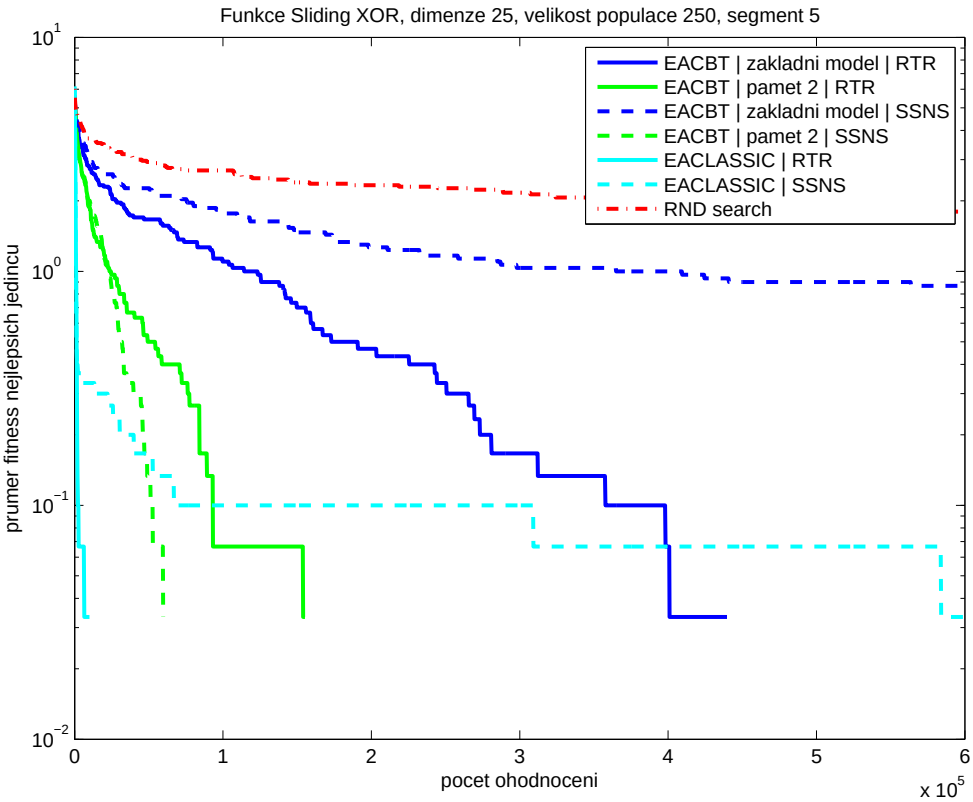
Obrázek 24: Porovnání výkonu testovaných algoritmů na funkci Sliding XOR dimenze 15

Obrázek 24 ukazuje, že u EACBT s RTR náhradou je paměť zlepšením a je skoro jedno s jakou velikostí a že pro stejný algoritmus s SSNS je paměť naopak zhoršením.



Obrázek 25: Porovnání výkonu testovaných algoritmů na funkci Sliding XOR dimenze 20

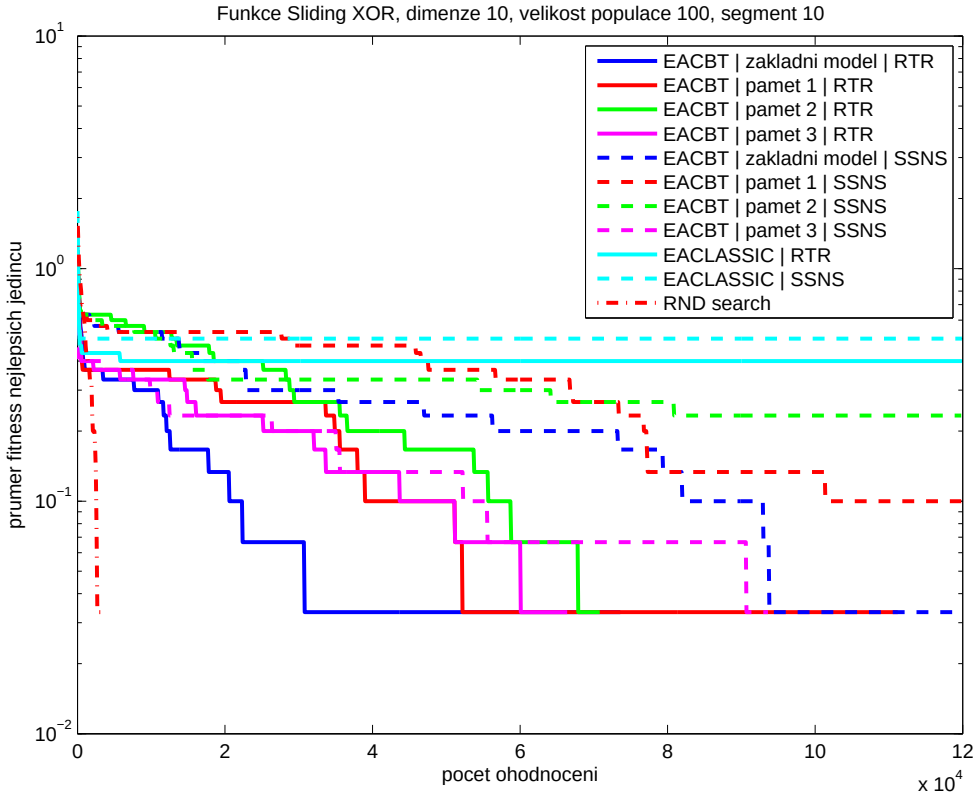
Jak vidíme z grafu na obrázku 25, pro dimenzi 20 paměť EACBT s RTR strategií přinesla takové zlepšení, že pro velikost 2 porazila i klasický EA. Pomohla i EACBT s SSNS, ale už ne tak razantně.



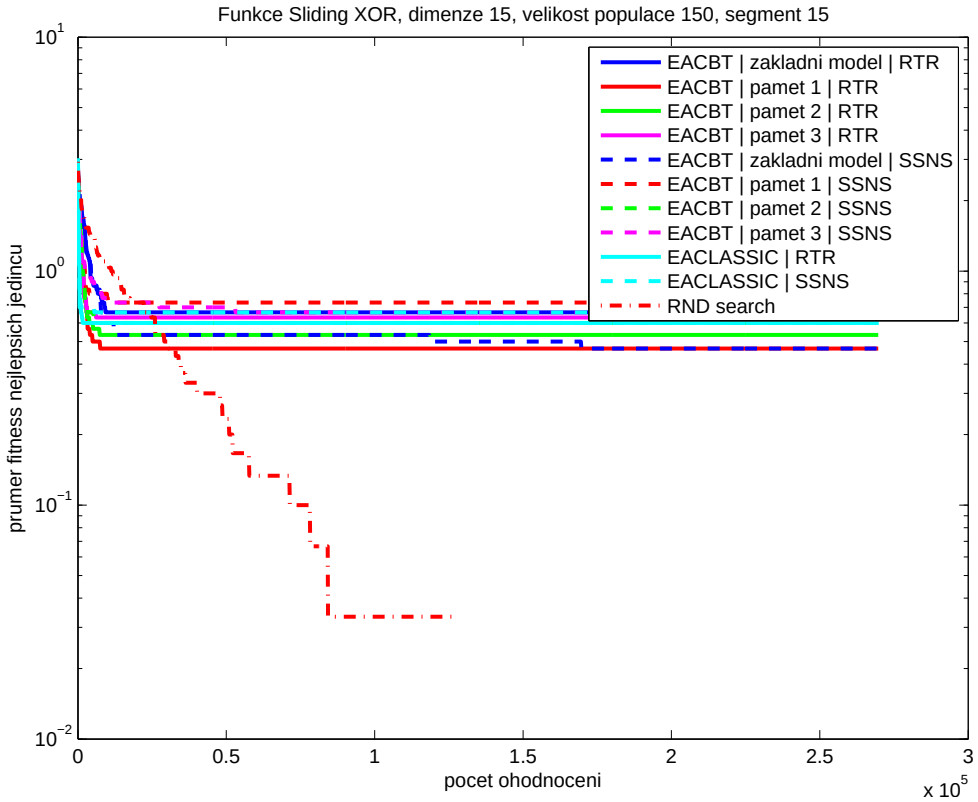
Obrázek 26: Porovnání výkonu testovaných algoritmů na funkci Sliding XOR dimenze 25

Na obrázku 26 je jasně vidět, že paměť pro EACBT není na škodu, ale že stále nepomáhá dost na to, aby porazila klasický EA s RTR náhradou.

Neseparované verze funkce SX



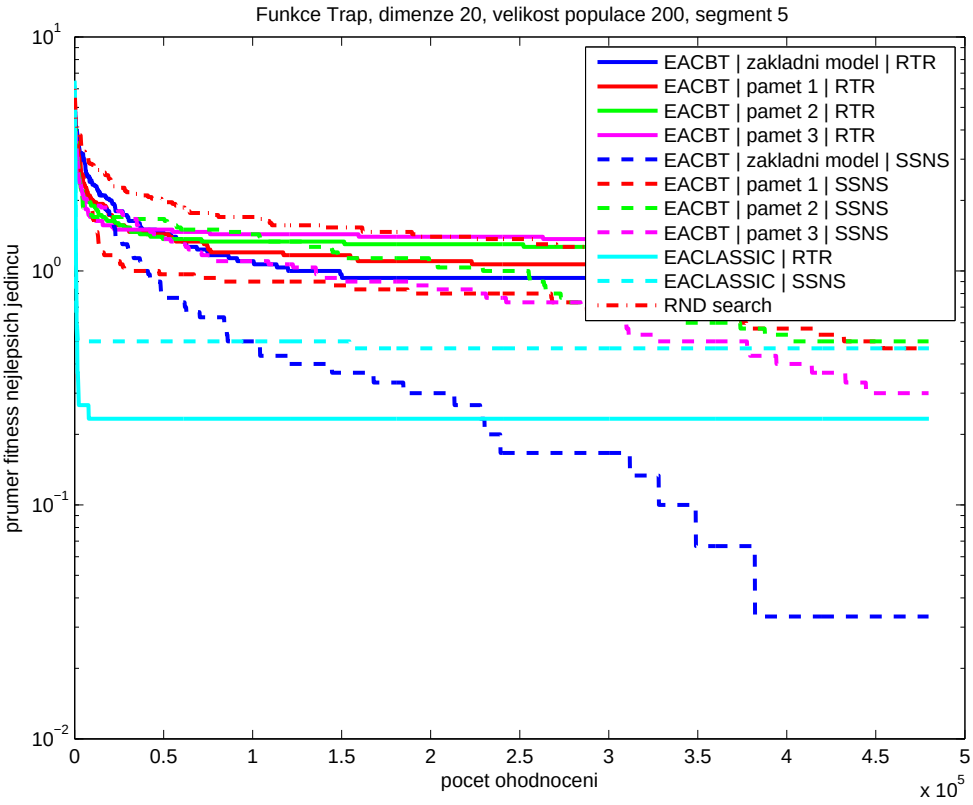
Obrázek 27: Porovnání výkonu testovaných algoritmů na neseparované verzi funkce Sliding XOR dimenze 10



Obrázek 28: Porovnání výkonu testovaných algoritmů na neseparované verzi funkce Sliding XOR dimenze 15

Z obrázků 27 a 28 vyčteme, že pro neseparované verze funkce Sliding XOR přidání paměti na optimalizaci této funkce nestačí a téměř žádnou změnu nepřineslo.

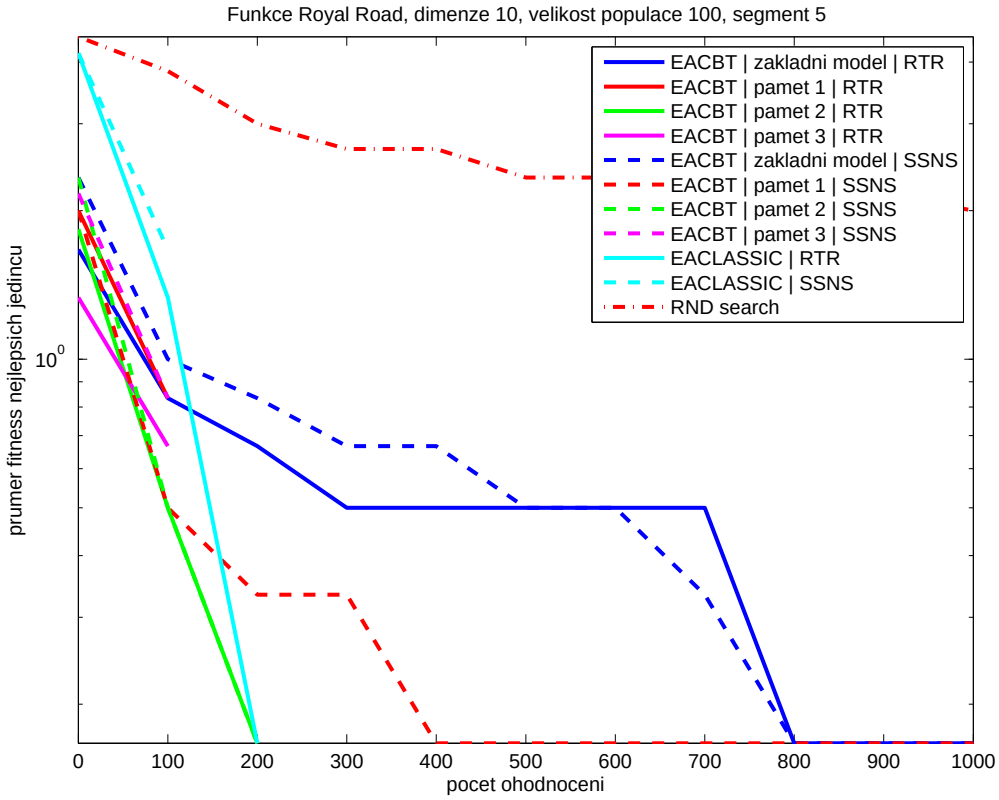
4.2.4 Grafy Trap funkce



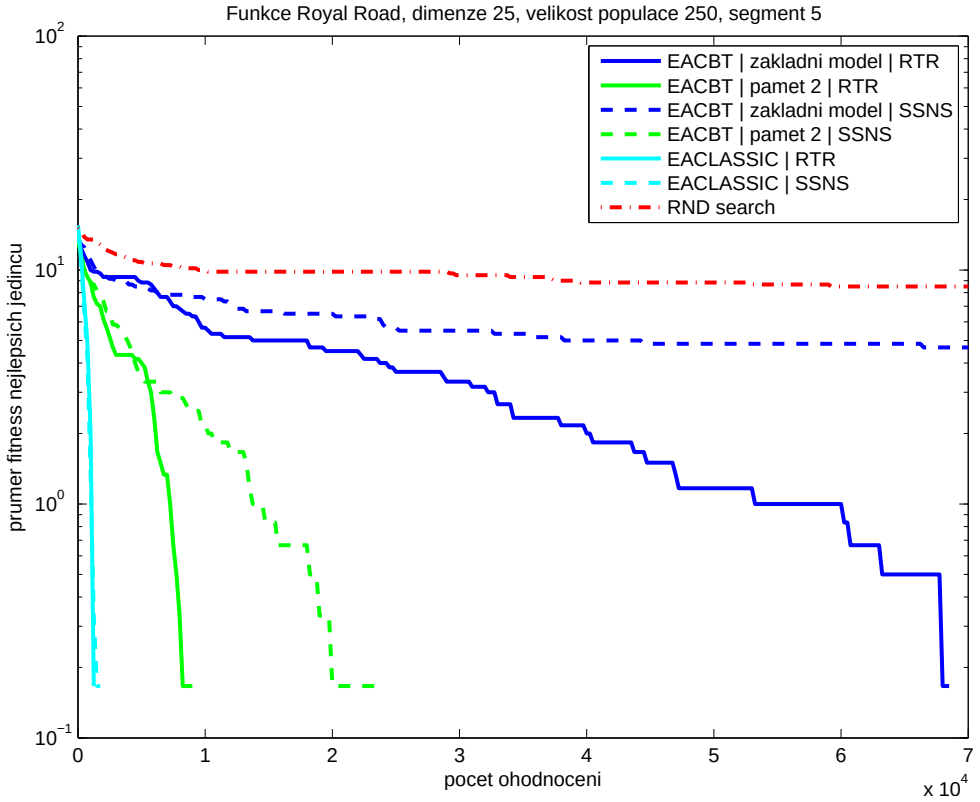
Obrázek 29: Porovnání výkonu testovaných algoritmů na funkci Trap dimenze 20

Z obrázku 29 vidíme, že na Trap funkci paměť si vede obdobně jako základní verze. Kromě EACBT s SSNS, který s pamětí ztratí schopnost rychlé konvergence. Totéž platí pro dimenze 10 i 15.

4.2.5 Grafy Royal Road funkce



Obrázek 30: Porovnání výkonu testovaných algoritmů na funkci Royal Road dimenze 10



Obrázek 31: Porovnání výkonu testovaných algoritmů na funkci Royal Road dimenze 25

Na obrázcích 30 a 31 vidíme, že průběh jednotlivých grafů je obdobný jako u One Max funkce a tedy podobný je i verdikt: na unimodálních funkcích je paměť pro EACBT rozhodně zlepšením. Velikost paměti nehraje příliš velkou roli. A stále platí, že to není dost na to, aby v rychlosti konvergence porazil klasický EA.

4.3 Diskuze výsledků

Vyvstává tedy otázka: Pomohlo to? Ano, pomohlo. Ale jak moc? Příliš ne. V některých případech přidání paměti znamená i několikanásobné zrychlení konvergence algoritmu ale nikde není zlepšení natolik výrazné, aby to znamenalo rozdíl mezi nalezením či nenalezením globálního optima. S výjimkou EACBT s SSNS, pro něž verze s pamětí pro některé případy znamená rozdíl mezi schopností optimum nalézat nebo nenalézat.

Zdá se, že paměť snižuje diverzitu populace a pomáhá lépe vyměřit slibné oblasti stavového prostoru, tedy dělá to, co dle očekávání dělat má. Proto na unimodálních funkcích vede její aplikace k výraznému zlepšení optimalizace. Na bimodálních či multimodálních funkcích ovšem výrazné zlepšení nepřinesla, někdy dokonce i naopak, protože jak popisuji v kapitole 3.4, vyšší diverzita populace může EACBT i pomáhat a zdá se, že v některých případech se zmenšuje příliš rychle.

Co se týče velikosti paměti, dle výsledků pro dimenze 10 – 20 se ukazuje, že velikost 1 bývá málo a že rozdíly mezi pamětí 2 a 3 nejsou většinou příliš výrazné. Proto když jsem se rozhodoval, jakou její hodnotu podrobím testům na dimenzi 25, nakonec jsem se rozhodl pro velikost 2.

5. Zhodnocení

5.1 Porovnání očekávání a skutečnosti

Od EACBT algoritmu se očekávalo, že by mohl být schopný překonat klasický EA, obdobně jako LEM v [1], ale tak se nestalo. Na vině může být více důvodů. LEM používá střídání módu s klasickým EA, kdežto já zde ne. LEM pracuje s reálnou reprezentací, kdežto já s binární. A hlavně LEM používá k řízení populace pravidlový model, kdežto EACBT model CART stromu. Jelikož ale EACBT algoritmus není příliš úspěšný, bylo by potřeba vyzkoušet více jeho verzí. Je taky možné, že některé vstupní parametry algoritmu mohou být nastaveny neoptimálně, ale není pravděpodobné, že by to vedlo k nějakým razantním změnám v chování algoritmu.

Taky jsem očekával, že se mi podaří provést testy i pro mnohem vyšší dimenze, než uvádím zde, ale narazil jsem na vysokou výpočetní složitost. Totiž, že stavba stromu trvá dlouho. A pokud se to má provádět každou generaci, navíc aby bylo možné podat nějaké statistické

výsledky, trvá to velice dlouho. K tomu, s rostoucí dimenzí a s rostoucí velikostí trénovací množiny se doba prodlužuje rychleji než lineárně. Získat výsledky na všech funkcích dimenze 10 mi trvalo jen pár hodin, kdežto pro dimenzi 20 a 25 už algoritmus běžel v řádech dnů. Proto jsem také pro dimenzi 25 snížil počet ohodnocení a testoval už pouze jednu velikost paměti. A také proto zde nejsou výsledky pro Trap funkci dimenze 25. Přesto si myslím, že má cenu se tímto algoritmem zabývat dále. Více v kapitole 5.2.

5.2 Návrhy na další rozšíření a vylepšení

Myslím si, že by stálo za další zkoumání pokusit se implementovat i ten druhý způsob vyžití paměti zmíněný v kapitole 4.1, tedy využití předešlých populací namísto předešlých stromů. Protože velikost trénovací množiny ovlivňuje tvorbu a kvalitu stromů. A správnost modelu ovlivňuje kvalitu nově vytvářených potomků.

S tím je spojené, že by stálo za hlubší prozkoumání i vliv velikosti populace na hloubku stromu a zároveň vliv na celý optimalizační proces.

Dalším způsobem jak algoritmu pomoci, by mohlo být pokusit se rozdělit populaci na populace dvě. Předpokládejme, že by se jednalo o minimalizaci. Jedna populace by se pokoušela nalézt globální minimum, druhá globální maximum. Na začátku by se vygenerovala jedna velká populace, ohodnotila, rozdělila v polovině a lepší část by byla označena za „OK“ kategorii, horší část za „KO“ kategorii. Na těchto datech by se vytvořil strom. Z listů stromu by se vytvořili potomci. V této části by se velká populace rozdělila na dvě. Z KO listů by se generovali potomci hledající globální maximum a z OK listů potomci hledající globální minimum. Dále by proběhla klasická mutace nových potomků a jejich následné ohodnocení. Mohla by proběhnout i mutace, že by s nějakou malou pravděpodobností pár jedinců z obou oddělených populací invertovalo všechny své bity a navzájem by si v populacích vyměnili pozice. Dále by se uplatnila náhradová strategie. Pro OK populaci by se snažila přijímat jen lepší jedince, pro KO populaci jen horší jedince. V dalších generacích by se populace spojili vždy jen pro učení nového stromu. A celý cyklus by se opakoval.

Věřím, že s vhodně nastavenou diverzitou populace by toto řešení mohlo být přínosem, protože by generovalo jak kvalitní OK jedince, tak „kvalitní“ KO jedince a teoreticky by se je CART strom od sebe měl naučit dobře rozlišovat a mohlo by mu to pomoci specifičtěji vymezit oblast s globálním optimem (být hlubší).

6. Seznam literatury

[1] MICHALSKI, Ryszard S. *Machine Learning* [online]. Springer Netherlands : Kluwer Academic Publishers, 2000 [cit. 2011-05-26]. Learnable Evolution Model, s. 9 - 40. Dostupné z WWW: <http://dx.doi.org/10.1023/A:1007677805582>.

[2] HAVLÍN, Václav. *Preferenční křížení v evolučních algoritmech*. Praha, 2009. 70 s. Bakalářská práce. ČVUT FEL.

[3] KLASCHKA, Jan; KOTRČ, Emil. Klasifikační a regresní lesy. In ANTOCH, J.; DOHNAL, G. *Sborník prací 13. letní školy JČMF* [online]. Praha : ROBUST, 2004 [cit. 2011-05-26]. Dostupné z WWW: <http://www.statspol.cz/robust/robust2004/klaschka.pdf>.

[4] MITCHELL, Melanie; FORREST, Stephanie; HOLLAND, John H. *Toward a Practice of Autonomous Systems: Pro-ceedings of the First European Conference on Articial Life*. Cambridge : MIT Press, 1992. The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance, s. 11.

1. Příloha – testované funkce

1.1 Vlastnosti funkcí

Abychom byli schopní od sebe jednotlivé funkce odlišit, je nejdříve potřeba definovat jejich jednotlivé vlastnosti. Vlastnosti, které nás budou zajímat jsou modalita, separovatelnost a klamnost.

Modalita funkce určuje kolik optim funkce má. Existují pouze unimodální, bimodální a multimodální funkce. Unimodální funkce mají pouze jedno jediné optimum, které je zároveň i globální. Zde se kvalita optimalizace určuje především dle rychlosti a přesnosti konvergence. Bimodální funkce mají optima dvě – jedno lokální a druhé globální. Dle velikosti plochy zabírané jedním optimem oproti druhému, je pro optimalizační algoritmus různě těžké požadované globální optimum nalézt. Čím je plocha globálního optima větší, tím lépe. Nakonec multimodální funkce mají složitý průběh s více lokálními optimy.

Separovatelnost funkce nám říká, zda je funkce separovatelná či nikoliv. Tedy jestli ji dokážeme rozdělit na její jednotlivé dimenze a optimalizovat jen její jednotlivé části nebo ne. Hledané d-dimenzionální optimum získáme opětovným složením všech částí.

Klamné funkce jsou takové, jejichž hodnota klesá jedním směrem do lokálního optima, ale globální optimum mají na opačné straně. Nalezení takového optima je velmi těžké.

Všechny zde použité funkce jsou též funkce binární proměnné a globální optimum je v jejich minimu – jedná se tedy o minimalizaci. Mějme tedy bitový řetězec *x* o délce *n* $x = [x_1, x_2, x_3, ..., x_n]$, potom globální minimum všech zde použitých funkcí je $f(x^*) = 0$, kde f je prohledávaná funkce a x^* je hledaný jedinec s nejlepší (a zároveň nejnižší) možnou fitness.

Jelikož některé funkce by byli příliš složité, používám v této BP obecnou rovnici, kterou používám u všech funkcí, jejíž definice je následující:

$$f_{seg}(x, s) = \sum_{i=1}^{\frac{n}{s}} f_x\left(x_{s\cdot(i-1)+1}, x_{s\cdot(i-1)+2}, ..., x_{s\cdot(i-1)+s}\right)$$

Rovnice 1

kde *s* je velikost segmentu a f_x může být jakákoliv funkce. Má několik omezení, například *n* musí být dělitelné *s* a *s* musí mít pro některé funkce stanovenu minimální velikost. Takto tedy separujeme i „neseparovatelné“ funkce, protože je rozložíme na nezávislé podproblémy délky *s* bitů.

1.2 One Max funkce

Jedná se o jednoduchou binární funkci, která je unimodální, separovatelná a neklamná. Jednoduše její fitness se lepší s počtem binárních jedniček v řetězci. V našem případě, kdy se zlepšující fitness snižuje, vypadá ohodnocovací funkce následovně:

$$f_{OneMax}(x) = n - \sum_{i=1}^{n} x_i$$

Rovnice 2

1.3 Equal Pairs funkce

Tato funkce je bimodální, neseparovatelná a neklamná. Fitness se zlepšuje, pokud jsou dva bity vedle sebe stejné. Má dvě optima, z nichž jedno je lokální pro řetězec nul a jedno globální pro řetězec složený z jedniček. Funkce Equal Pairs vypadá následovně:

$$f_{EqualPairs} = n - x_1 - \sum_{i=2}^{n} f_{EP}(x_{i-1}, x_i)$$

Rovnice 3

přičemž f_{EP} je funkce, která detekuje rovnost jejích dvou vstupních bitů[3]:

$$f_{EP}(x_1, x_2) = \begin{cases} 1, & když\ x_1 = x_2 \\ 0, & když\ x_1 \neq x_2 \end{cases}$$

Rovnice 4

Odečtením první bitu získá výhodu řetězec složený ze samých jedniček, namísto řetězce složeného z nul.

1.4 Sliding XOR funkce

Jedná se o multimodální, neseparovatelnou a neklamnou funkci, jejíž optimalizace spočívá v nalezení řetězce, kde každé tři bity vedle sebe splňují logickou funkci XOR. Rovnice funkce:

$$f_{SlidingXOR} = n - 1 - f_{AllEqual}(x) - \sum_{i=3}^{n} f_{XORWindow}(x_{i-2}, x_{i-1}, x_i)$$

Rovnice 5

kde je potřeba odečíst jedničku, aby bylo globální optimum v nule, a kde funkce $f_{AllEqual}$ vrací jedna, pokud jsou všechny bity v řetězci stejné, tedy:

[3] V podstatě se jedná o negovanou funkci XOR. Více o funkci XOR viz 1.4 a Tabulka 1.

$$f_{AllEqual}(x)=\begin{cases}1, & když\ x=0,0,...,0\ \ nebo\ 1,1,...,1,\\ 0, & jinak\end{cases}$$

Rovnice 6

Díky kombinaci logické funkce XOR a díky zvýhodnění všech stejných bitů vedle sebe, vychází globální optimum pro řetězec *n* nul. Funkce $f_{XORWindowl}$, která vyhodnocuje vazby tří bitů vedle sebe, vypadá následovně:

$$f_{XORWindow}(x_1,x_2,x_3)=\begin{cases}1, & když\ x_1\oplus x_2=x_3\\ 0, & jinak\end{cases}$$

Rovnice 7

a	b	$a\oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Tabulka 1: hodnoty logické funkce XOR

1.5 Trap funkce

Tato funkce je bimodální, neseparovatelná a jak již název napovídá – klamná. Její hodnota stoupá jedním směrem do lokálního optima, ale globální optimum je na úplně druhé straně. A jelikož lokální optimum zabírá v tomto případě mnohem více prostoru než optimum globální, je nalezené globálního optima velice obtížné. Rovnice této funkce je následující:

$$f_{Trap}(x)=\begin{cases}0, & když\ x=1,1,...,1,\\ n-1-f_{OneMax}(x), & jinak\end{cases}$$

Rovnice 8

1.6 Royal Road funkce

Royal Road funkce je unimodální, separovatelná a neklamná. V této BP používám upravenou verzi oproti její originální definici, kterou naleznete v [4], ve 3. kapitole. Upravená definice je velice jednoduchá:

$$f_{RoyalRoad}(x)=\begin{cases}0, & když\ x=1,1,...,1,\\ n, & jinak\end{cases}$$

Rovnice 9

2. Příloha – obsah přiloženého DVD

K této práci je přiloženo DVD, na kterém jsou uloženy následující soubory a složky:

- TomasTokoly-BP-2011.pdf: elektronická verze této BP
- LEM.pdf: elektronická verze [1]
- matlab-kod: Matlabovský zdrojový kód algoritmů
- matlab-vysledky: uložené výsledky v Matlabovském formátu .mat
- grafy: všechny grafy v .eps formátu, které jsem zde zobrazil i ty které ne a jen se o nich zmiňuji
- ctime.txt: popis jak spustit testy a zobrazit výsledky naprogramovaného EACBT algoritmu i ostatních zde použitých algoritmů