VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

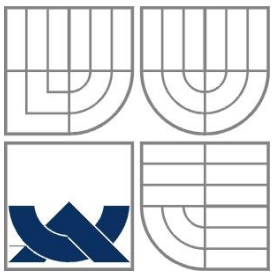
ANALÝZA RŮZNÝCH PŘÍSTUPŮ K ŘEŠENÍ OPTIMALIZAČNÍCH ÚLOH

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

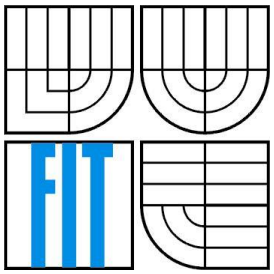
AUTOR PRÁCE
AUTHOR

BC. JAKUB KNOFLÍČEK

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA RŮZNÝCH PŘÍSTUPŮ K ŘEŠENÍ OPTIMALIZAČNÍCH ÚLOH

ANALYSIS OF VARIOUS APPROACHES TO SOLVING OPTIMIZATION TASKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. JAKUB KNOFLÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

DOC. ING. FRANTIŠEK VÍTĚZSLAV ZBOŘIL, CSC

BRNO 2013

Abstrakt

Tato práce se zabývá řešením optimalizačních úloh při použití různých přístupů. Na úvod je formálně definován pojem optimalizační úloha a následuje zavedení pojmu fitness funkce, která je společná pro všechny optimalizační metody. Poté jsou rozebrány přístupy při optimalizaci pomocí hejna částic, mravenčí kolonií, simulovaným žíháním, genetickými algoritmy a posilovaným učením.

Pro testování slouží dvě diskrétní (problém více batohů a problém pokrytí množiny) a dvě spojité úlohy (hledání globálního minima Ackleyho a Rastriginovy funkce), jež popisujeme v další kapitole.

Následuje popis implementačních detailů pro jednotlivé optimalizační metody, například způsoby reprezentace řešení či jakým způsobem jsou stávající řešení v průběhu algoritmu měněna.

Nakonec jsou prezentovány výsledky měření, které ukazují optimální nastavení parametrů zkoumaných metod vzhledem k testovacím úlohám.

Abstract

This paper deals with various approaches to solving optimization tasks. In prolog some examples from real life that show the application of optimization methods are given. Then term optimization task is defined and introducing of term fitness function which is common to all optimization methods follows. After that approaches by particle swarm optimization, ant colony optimization, simulated annealing, genetic algorithms and reinforcement learning are theoretically discussed.

For testing we are using two discrete (multiple knapsack problem and set cover problem) and two continuous tasks (searching for global minimum of Ackley's and Rastrigin's function) which are presented in next chapter.

Description of implementation details follows. For example description of solution representation or how current solutions are changed.

Finally, results of measurements are presented. They show optimal settings for parameters of given optimization methods considering test tasks.

Klíčová slova

Optimalizační úlohy, optimalizační metody, simulované žíhání, optimalizace mravenčí kolonií, optimalizace hejnem částic, genetické algoritmy, posilované učení, problém více batohů, problém pokrytí množiny, Ackleyho funkce, Rastriginova funkce.

Keywords

Optimization tasks, optimizations methods, simulated annealing, ant colony optimization, particle swarm optimization, genetic algorithms, reinforcement learning, multiple knapsack problem, set cover problem, Ackley's function, Rastrigin's function.

Citace

Knoflíček Jakub: Analýza různých přístupů k řešení optimalizačních úloh. Brno, diplomová práce, FIT VUT v Brně, 2013.

Analýza různých přístupů k řešení optimalizačních úloh

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Františka Vítězslava Zbořila, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jakub Knoflíček
20. 5. 2013

Poděkování

Chtěl bych tímto vyjádřit poděkování vedoucímu této práce, který se mnou vždy ochotně konzultoval všechny nastalé problémy.

Obsah

1 Úvod

V rámci této práce se budeme zabývat optimalizačními úlohami, konkrétně metodami pro jejich řešení. V následující kapitole si proto takovou optimalizační úlohu důkladně definujeme a dále pak popíšeme použité optimalizační metody.

Konkrétně se budeme zabývat pěti metodami a to optimalizací hejnem částic, simulovaným žíháním, genetickými algoritmy, optimalizací pomocí mravenčí kolonie a posilovaným učením.

Na úvod ale nejprve představme praktické použití a aplikace těchto metod v reálném světě (viz [1]), které budou sloužit coby motivace. Například při optimalizaci portfolia hledáme nejlepší cestu pro investování kapitálu do množiny n aktiv, kde proměnná x_i bude reprezentovat investici do i-tého aktiva, čímž dostaneme řešení ve formě vektoru $x \in \mathbb{R}^n$. Tento vektor pak popisuje celkovou alokaci portfolia napříč řadou aktiv. Samozřejmě budeme klást určité požadavky na naše rozdělení – limit na rozpočet (celkové investované prostředky) a minimální očekávaný zisk celého portfolia.

Jiným příkladem použití optimalizačních metod je tzv. device sizing, kdy při návrhu elektroniky musíme vybrat vhodnou šířku a délku každého zařízení v elektrickém obvodu. Zde máme opět dána určitá omezení a to ve formě limitů na velikosti zařízení uložených výrobním procesem nebo časová omezení, která kladou požadavky na to, aby zařízení garantovalo specifikovanou rychlost. A konečně se může jednat o limity kladené na celkovou plochu obvodu. Naším úkolem je pak nalezení takového rozložení zařízení, které splňuje výše uvedené požadavky a přitom je maximálně energeticky efektivní.

Z uvedených příkladů vidíme, že optimalizační metody mají své opodstatnění i v reálném světě. V tomto dokumentu se zaměříme na čtyři optimalizační úlohy, dvě spojité a dvě diskrétní: problém více batohů (multiple knapsack problem), problém pokrytí množiny (set cover problem), hledání globálního minima Ackleyho a Rastriginovy funkce.

V navazující části dále rozebereme implementaci zmíněných metod pro dané úlohy, uvedeme celkovou koncepci, případně zmíníme některé významné detaily.

Hlavní činností v této práci pak bude hledání nejlepšího nastavení optimalizačních metod pro naše úlohy tak, abychom dostávali co nejkvalitnější výsledky. Tak trochu nadneseně to můžeme popsat jako optimalizaci samotných optimalizačních metod.

2 Optimalizační úloha

Řešení optimalizačních úloh je ústředním tématem celé této práce, proto je naprosto esenciální si pojem optimalizační úloha exaktně definovat.

Matematickou optimalizační úlohu nebo prostě optimalizační úlohu (někdy nazývanou optimalizační problém) můžeme definovat následovně:

$$minimalizace\ f_0(x)$$
$$vzhledem\ k\ f_i(x) \leq b_i, i = 1, \dots, m.$$

Přibližme jednotlivé komponenty těchto vztahů. Nejprve vektor $x = (x_1, \dots, x_n)$ zde vystupuje jako optimalizační proměnná úlohy. Dále funkce $f_0: \mathbb{R}^n \to \mathbb{R}$ je objektivní funkcí a funkce $f_i: \mathbb{R}^n \to \mathbb{R}$, $i = 1, \dots, m$ jsou omezující funkce (nerovnice) a konečně $b_1, \dots, b_m$ jsou limity či meze pro omezující funkce. [1]

Název „optimalizační“ úloha nám jasně naznačuje, že budeme potřebovat definici jistého extrému, kterým bude vektor x^*. Vektor x^* nazýváme optimální, též řešením, pokud představuje vektor s nejmenší objektivní hodnotou mezi všemi vektory, které splňují omezení, tj. pro libovolné z, kde $f_i(z) \leq b_i, i = 1, \dots, m$, musí platit $f_0(x^*) \leq f_0(z)$.

Optimalizační úlohy můžeme dělit do různých skupin či kategorií. Z matematického pohledu se může například jednat o takzvanou lineární úlohu, tj. jestliže objektivní a omezující funkce jsou lineární:

$$f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y)$$

pro všechna $x, y \in \mathbb{R}^n$ a všechna $\alpha, \beta \in \mathbb{R}$. Pokud optimalizační úloha není lineární, pak se jedná o nelineární optimalizační úlohu. [1]

V rámci této práce budeme optimalizační úlohy dělit také z jiného pohledu a to, zda se jedná o diskrétní nebo spojitou optimalizační úlohu. Laicky můžeme říci, že diskrétní úloha je postavena na množině diskrétních hodnot či na spočetné množině hodnot či stavů, naopak spojitá úloha na nespočetné (spojité) množině hodnot. Některé metody, které dále budeme uvažovat, jsou navrženy pro řešení diskrétních úloh, proto budeme muset někdy provést diskretizaci řešené úlohy, čili převod spojitého modelu a rovnic na diskrétní.

Optimalizační úloha je tedy abstrakce problému volby nejlepšího možného vektoru v $\mathbb{R}^n$ z množiny kandidátních řešení. Proměnná x reprezentuje vybranou volbu, omezení $f_i(x) \leq b_i$ reprezentují požadavky na toto řešení a objektivní funkce $f_0(x)$ značí cenu výběru tohoto řešení x. Řešení optimalizační úlohy poté koresponduje s volbou, která má minimální cenu mezi všemi možnostmi, které splňují požadavky. [1]

3 Optimalizační metody

Pod pojmem optimalizační metody budeme rozumět metody pro řešení optimalizačních úloh, tak jak jsme si je definovali v předchozí kapitole.

V následující kapitole si rozebereme použité techniky, popíšeme jejich princip a definujeme vlastní algoritmy jednotlivých optimalizačních metod. Předem si můžeme uvést, že všechny použité metody se vyznačují tím, že negarantují nalezení globálního optima, tedy řešení, které je nejlepší v rámci všech možných řešení.

3.1 Fitness funkce

V následujících kapitolách se seznámíme s různými optimalizačními metodami, ale vyskytuje se zde jejich společný prvek, který si uvedeme předem. Tímto je hodnotící (fitness) funkce. Při běhu optimalizačních metod bývá generováno či procházeno množství řešení. Potřebuje mít operátor uspořádání, kterým tato funkce je. Fitness funkce udává kvalitu řešení. Musí pro ni platit přímá úměra: čím kvalitnější řešení, tím vyšší hodnota fitness funkce. Nebo jinak, čím blíže je řešení k optimálnímu či správnému řešení, tím vyšší hodnotu má. Tomuto přímo odpovídají maximalizační úlohy, ale pro minimalizační úlohy bývá potřeba promítnout snižující se hodnotu do jmenovatele, aby platila výše zmíněná přímá úměra. V podstatě se v našem případě jedná o objektivní funkci (viz kapitola 2), ale obecně tomu tak úplně být nemusí. Často se například používá mocnina hodnoty objektivní funkce či jinak upravená její hodnota.

3.2 Optimalizace mravenčí kolonií

Optimalizace pomocí mravenčí kolonie (angl. ant colony optimization) patří do oblasti tzv. swarm intelligence (inteligence hejna). Pojem swarm intelligence byl poprvé použit v kontextu s celulárními robotickými systémy, kde mnoho jednoduchých agentů obsazovalo jedno nebo dvoudimenzionální pole za účelem generování vzorů a samoorganizace prostřednictvím interakcí mezi sousedy. Taková definice je ovšem velmi restriktivní, proto můžeme swarm intelligence definovat jako jakýkoliv pokus o návrh algoritmů nebo problémy řešících distribuovaných systémů, které jsou inspirovány kolektivním chováním sociálních hmyzích kolonií a jiných zvířecích společností. [2]

Mravenčí kolonie, podobně jako i jiné sociální hmyzí společnosti, lze považovat za distribuované systémy, které navzdory jednoduchosti jednotlivců vykazují vysoce strukturovanou sociální organizaci. V této skupině jako celku jsou při komplexních úlohách dosahovány výsledky, které přesahují možnosti jednoho jedince.

Algoritmy vycházející z chování mravenčí kolonie jsou založeny na výsledcích studií a modelů chování, jež byly vytvořeny na základě pozorování reálných mravenců. Tyto modely jsou využívány pro návrh nových algoritmů na řešení optimalizačních úloh a úloh s distribuovaným řízením. [3]

Studium mravenčích kolonií přineslo a inspirovalo množství metod a technik, avšak nejstudovanější a nejúspěšnější je prozatím všeobecná optimalizační technika známá jako optimalizace mravenčí kolonií (angl. ant colony optimization, ACO). Pro úplnost dodejme, že pro výzkum se užívají různé či pouze některé druhy mravenců. [4]

Podstatou jsou samoorganizační principy, které dovolují vysoce koordinované chování reálných mravenců. Tyto pak lze použít k vývoji a řízení spolupráce v populaci inteligentních agentů, kteří jsou schopni společně řešit výpočetní problémy.

Algoritmy mohou sledovat rozdílné aspekty chování mravenců. Uveďme několik příkladů, které inspirovaly možné přístupy: hledání potravy, kooperativní transport, třídění dle plodů. Ve všech těchto případech mravenci využívají stigmergii, což je forma komunikace, kdy si hmyz předává informace prostřednictvím chemických značek v prostředí. Čili jedná se o nepřímou komunikaci prostřednictvím změn v prostředí. Biologové prokázali, že stigmergie je dostatečná k vysvětlení toho, jak mohou mravenci dosáhnout samoorganizovaného chování jakožto celá populace.

Jak jsme již uvedli, ACO je nejčastěji užívanou aplikací sociálního chování mravenčích druhů. Uveďme, že tato je inspirována právě chováním mravenčí kolonie při hledání potravy a je využívána pro diskrétní optimalizační úlohy. [3]

Příčinou, proč mravenci využívají pro komunikaci právě chemické látky, je pravděpodobně to, že jejich zrak je vyvinut pouze na rudimentální úrovni. Některé druhy mravenců jsou dokonce zcela slepé. Chemické látky, které mravenci produkují, jsou nazývány feromony. Obzvláště významným feromonem je feromon, který užívají některé specifické druhy mravenců ke značení cest, například mezi nalezištěm potravy a mraveništěm (angl. trail pheromone). Poznamenejme, že účinek feromonů časem vyprchá, naopak pokud je feromon na daném místě zanechán více jedinci, je jeho účinek silnější. Odtud je vidět, že intenzivnější hladina feromonu označuje cestu, která je aktuálně významnější. [3]

Pokud budeme uvažovat definici ACO algoritmu jako množinu výpočetních, konkurenčních a asynchronních agentů, pak se tito pohybují napříč stavy řešeného problému, které odpovídají jeho částečným řešením. Agenti se pohybují aplikováním stochastické politiky pro lokální rozhodnutí založené na dvou parametrech.

První z nich indikuje, jak moc byl daný stav v minulosti výhodný (tzv. trail level). Tento parametr tedy určuje *a posteriori* informaci o žádanosti (rozumějme vhodnosti) daného stavu při tvorbě lokálního rozhodování.

Druhým parametrem je atraktivita (attractiveness) pohybu či stavu. Tato *a priori* informace je tvořena nějakou, lépe řečeno vhodnou, heuristickou funkcí.

Svým pohybem mravenec inkrementálně konstruuje řešení problému. Jakmile dosáhne řešení, ohodnotí dané řešení a dle toho aktualizuje hodnotu trail level u všech prvků použitých při konstrukci řešení. Zmiňme také, že ACO obsahuje proces vypařování feromonů, který snižuje trail level u všech obsažených stavů rovnoměrně v průběhu času. Toto je důležitý mechanismus, který zabraňuje nekontrolovanému kumulování hodnot feromonů u některých komponent. Navíc může implementace obsahovat i tzv. daemon actions, které zajišťují centralizované akce, které nemůže vykonat jedinec samotný (například vyvolání globální optimalizační metody). [5]

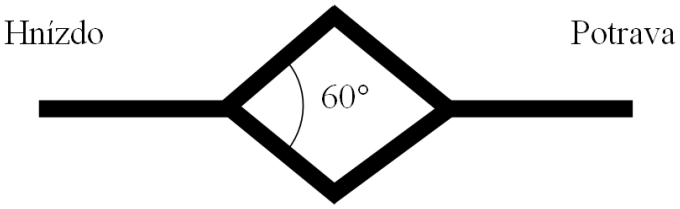
Shrňme tedy, že centrální komponentou ACO je parametrizovaný pravděpodobnostní model nazývaný feromonový model. Tento odpovídá prohledávanému prostoru při získávání řešení. Aktualizace feromonových hladin je provedena za použití předchozích dosažených řešení. [6]

3.2.1 Double bridge experiment

Nyní si ilustrujme to, že tento navržený model odpovídá reálnému chování mravenčí kolonie. Jedná se o pokus zvaný „double bridge experiment“ (tj. pokus se dvěma mosty) viz [7] a [8]. V tomto pokusu byl zkoumán vzor chování speciálního druhu argentinských mravenců Iridomyrmex humilis. Zatímco většina druhů mravenců je známa tím, že uvolňují feromonovou stopu ve chvíli, kdy se vrací do hnízda od důležitého zdroje potravy, tento druh uvolňuje feromony na cestu více či méně kontinuálně a používá tentýž feromon při cestě z hnízda i při návratu z hledání potravy.

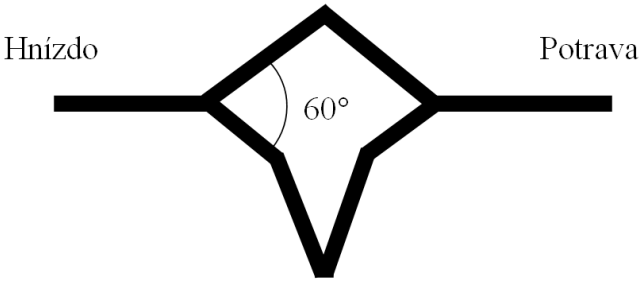
Popišme si podmínky pokusu. Mravenčí dělníci v počtu 150-1200 jedinců se pohybují ve vymezené oblasti cca 0.8 × 0.8 m, která je pokryta bílým pískem a je spojena mostem s krabicí s jídlem. Most je rozdělen vedví do tvaru diamatu, přičemž úhel mezi cestami je 60° (viz Obrázek 1).

Z obrázku je taktéž vidět, že obě cesty na mostě jsou rovnocenné, tedy stejně dlouhé. Předpokladem je, že mravenci budou využívat obě strany mostu přibližně se stejnou pravděpodobností. Je důležité poznamenat, že pro pokus je povrch zkoumané oblasti absolutně bez feromonů, které by mohly mít vliv na průběh experimentu. Tedy celá oblast je bez jakéhokoli označkování. Jen tak můžeme pozorovat vzorec chování pro prozkoumávání neznámého místa, či hledání potravy na neznámých místech.



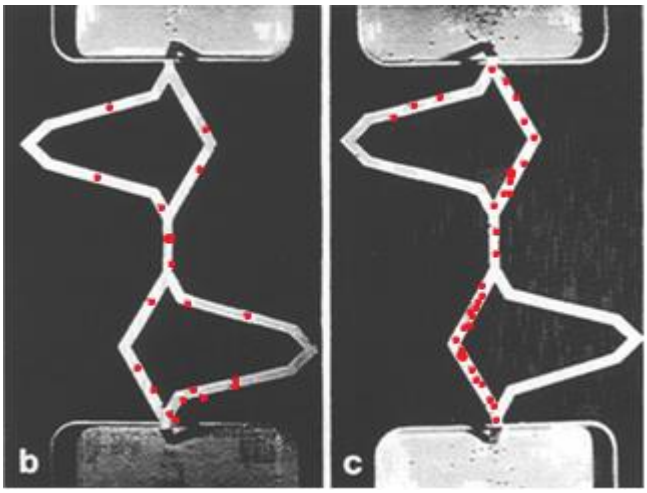
Obrázek 1: Diamantový tvar mostu při double bridge experimentu

Mravenci prohledávají neoznačkovanou oblast náhodně. Na rozděleném mostě jsou obě větve zpočátku vybírány přibližně stejnoměrně, jak bylo předpokládáno. Ale tím, jak každý mravenec prochází přes most (jednou či druhou větví), mění prostřednictvím uvolňování feromonu pravděpodobnost výběru levé či pravé strany mostu dalším mravencem. V určité chvíli převáží účinek feromonu na jedné straně a rapidně se změní poměr mravenců procházejících přes větve mostu ve prospěch jedné. Tento fakt byl také ověřen pomocí simulační metody monte carlo. Je to způsobeno náhodnými fluktuacemi v počtu mravenců procházejícími větvemi, kdy si v nějakém okamžiku náhodně vybere více mravenců stejnou cestu. Pak je intenzita jejich feromonové cesty tak velká, že s sebou strhne i následující mravence. O tom, že byly obě strany mostu rovnocenné, svědčí to, že při opakování tohoto pokusu zvítězila každá z větví v přibližně 50% případů. Tato pozitivní zpětná vazba může být mravenci použita k nalezení nejkratší cesty mezi hnízdem a potravou (zde bych poukázal na očividnou podobnost s úlohou obchodního cestujícího).



Obrázek 2: Varianta double bridge experimentu s jednou delší větví

Byla provedena také druhá varianta tohoto pokusu (viz [7]). Podmínky experimentu jsou téměř totožné. Opět jsou zde dvě místa (hnízdo a potrava), která jsou spojena mostem se dvěma cestami. Obě cesty v připojovacích bodech jsou opět odkloněny od roviny o 30°. Poznamenejme, že tato hodnota je volena kvůli minimalizaci odchylek od běžného chování. Jedna ze stran mostu je ovšem tentokráte velmi významně delší. Aby byl minimalizován vnější vliv na výběr cesty, je za most zařazen druhý, který je naprosto stejný, jen je delší cesta na druhé straně. Zde platí, že na kratší cestě dochází ke kumulaci feromonu rychleji, tedy logicky po určitém čase nastává konvergence ke kratší cestě. Zde je ještě lépe vidět schopnost mravenčí kolonie nalezení optimální, či suboptimální cesty v prostředí, v němž má pouze minimální, lokální informace. Ilustrujme průběh konvergence na dvou originálních fotografiích, kde byla pro větší názornost zvýrazněna červenou barvou poloha jednotlivých mravenců. Výsledkem je nalezení nejkratší cesty (viz Obrázek 3).



Obrázek 3: Originální fotografie zobrazující výsledek experimentu (viz [8]). Na obrázku B je vidět průběh konvergence ve 4. minutě. Obrázek C zobrazuje vybranou minimální cestu.

3.2.2 Ant colony system

V tomto okamžiku se seznámíme s formálními základy optimalizace mravenčí kolonií, konkrétně s nejpoužívanější verzí ACS (Ant Colony System), viz [9]. Tato verze je navíc i poměrně obecná, proto si principy vysvětlíme právě na ní. Připomeňme si, že výpočet optimalizace probíhá v cyklech. Na konci každé iterace dosáhnou všichni mravenci (agenti) řešení. Z nich je vybrán jeden, který nalezl nejlepší řešení dle hodnotící funkce. Touto hodnotou jsou pak aktualizovány spojnice mezi stavy, kterými prošel při hledání tohoto řešení, což reprezentuje feromonovou zpětnou vazbu. Dodejme, že hodnoty feromonových hladin mohou být při reprezentaci zaznamenány ve stavech, potažmo v uzlech (částečných řešeních), kterými agent při konstrukci či hledání řešení prochází. Též je možné například při stagnaci řešení snižovat feromonovou hladinu u řešení, která neodpovídají nejlepšímu nalezenému.

Charakteristikou ACS je dále lokální aktualizace feromonů, která je způsobem diversifikace vyhledávání v prohledávaném prostoru. Aktualizovaná hodnota feromonové hladiny leží poté mezi defaultní hodnotou a předchozí hodnotou. Tato aktualizace je prováděna při každém kroku uvnitř prostředí a to každým agentem.

Uveďme si formálně rozdíl mezi globální a lokální aktualizací feromonových hladin. Pro globální aktualizaci platí následující vztah:

$$\tau_{ij} \longleftarrow \delta \cdot \tau_{ij} + (1 - \delta) \cdot \Delta\tau_{ij}$$

kde τ_{ij} značí hodnotu feromonů mezi uzly i a j. Dále δ značí koeficient vypařování feromonů, jehož hodnota by se měla nacházet v rozmezí $\langle 0; 1)$. Konečně $\Delta\tau_{ij}$ reprezentuje přidanou hodnotu feromonu. Nárůst feromonové hladiny o $(1 - \delta) \cdot \Delta\tau_{ij}$ je uplatňován pro každý jeden krok z vítězného řešení. Tedy:

$$\Delta\tau_{ij} = \begin{cases} Add_{best}\ jestliže\ (i,j) \in globálně\ nejlepší\ řešení \\ 0\ jinak \end{cases}$$

Hodnota Add_{best} je funkcí ohodnocení doposud nejlepšího nalezeného řešení S_{best}. Obvykle se používá $Add_{best} = f(S_{best})$ pro maximalizační a $Add_{best} = (f(S_{best}))^{-1}$ pro minimalizační úlohy.

Oproti tomu, jak jsme si již uvedli, lokální optimalizace provádí všichni agenti a to při každém kroku konstrukce svého řešení. Platí pro ni vztah:

$$\tau_{ij} \longleftarrow \varepsilon \cdot \tau_{ij} + (1 - \varepsilon) \cdot \tau_0$$

Proměnná ε je nastavitelný parametr v intervalu $(0; 1)$ a τ_0 je počáteční hodnota feromonových hladin. Ostatní proměnné mají stejný význam jako v předešlé rovnici.

Objasnili jsme, jak se mění hodnoty feromonů ve feromonovém modelu. Avšak hlavní věcí, kterou je třeba vysvětlit pro činnost ACO, je výběr následujícího stavu (uzlu grafu), do kterého se agent přesune při budování svého řešení, viz [3].

Nejprve vyjdeme z prvotního Ant Systému, který je nejstarším systémem ACO. Tento je také základem pro vysvětlovaný ACS. Uveďme tedy systém výběru následujícího stavu systémem AS:

$$q_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} + \eta_{ij}^{\beta}}{\sum_{(ix)\notin tabu_k}\left(\tau_{ix}^{\alpha} + \eta_{ix}^{\beta}\right)} & jestliže\ (ij) \notin tabu_k \\ 0\ jinak \end{cases}$$

Proměnná q_{ij}^{k} je pravděpodobnost výběru spojnice stavů i a j agentem k, τ_{ij} je hodnota feromonu, která je umocňována na hodnotu parametru α. Dále pak η_{ij} je atraktivita (heuristické ohodnocení) spojnice stavů i a j umocněná na hodnotu parametru β a $tabu_k$ je množinou spojnic již vybraných pro konstrukci řešení agenta k. Pro následující krok je vybrána jedna spojnice náhodně dle spočítaných pravděpodobností.

V tomto okamžiku tedy známe základní vztah pro výpočet pravděpodobnosti výběru jednotlivých spojnic pro konstrukci řešení agentem. Tento si ještě rozšíříme, abychom dosáhli ACS:

$$s = \begin{cases} \arg\max_{(ij)\notin tabu_k} \tau_{ij}^{\alpha} + \eta_{ij}^{\beta}\ jestliže\ q \le q_0 \\ jinak\ použij\ pravidlo\ AS \end{cases}$$

Zde s je vybraná spojnice pro krok agenta, dále q je náhodně vygenerovaná hodnota z intervalu $< 0{,}1 >$ a q_0 je nastavitelný parametr pro volbu systému (také v rozmezí $< 0{,}1 >$). Tedy s pravděpodobností q je vybírán systém ACS a s pravděpodobností $1 - q$ o následujícím kroku rozhoduje systém AS. ACS pravidlo nepoužívá na rozdíl od AS pravděpodobností, ale pro krok vybere spojnici s maximálním ohodnocením.

Konečně uveďme celý výpočetní model, přesněji uveďme obecný rámec či algoritmus, který zahrnuje řešené úlohy. Proměnné τ_{ij} a η_{ij} odpovídají předešlému popisu a BS reprezentuje nejlepší nalezené řešení.

```
inicializace τij a ηij dle zadaného problému, počáteční řešení BS
until není splněna ukončující podmínka do
        nastav počáteční stav u všech mravenců
        until všichni mravenci nedosáhli řešení do
                forall mravenec k, který nedosáhl řešení do
                        spočítej ohodnocení možných kroků
                        vyber krok pro vykonání mravencem k
                        přidej vybraný krok do seznamu tabu mravence k
                        proveď lokální aktualizaci feromonů
                forall mravenec k, který právě dosáhl řešení then
                        proveď lokální prohledání okolí
        vyber mravence s nejlepším řešením
        if je nalezené řešení lepší než BS then
                aktualizuj BS
        proveď globální aktualizaci feromonů
```

Algoritmus 1: Algoritmus Ant Colony System

Seznamme se nyní s činnostmi v rámci uvedeného algoritmu, o kterých ještě nebyla zmínka. Vidíme, že se celý základní postup opakuje, dokud není splněna ukončující podmínka. Tato není algoritmem přesně specifikována, její volba obvykle záleží na zadané úloze. Často se užívá buď předem daný počet iterací, nebo se čeká, než bude nalezeno řešení, jehož ohodnocení je lepší než zadaná hraniční hodnota.

Dále se v algoritmu vyskytuje nastavení počátečního stavu u všech mravenců. Toto nastavení opět závisí na dané úloze. Může se jednat například o umístění mravence na určitý bod ve stavovém prostoru.

Částí algoritmu, o které jsme se ještě také nezmiňovali, je lokální prohledání okolí. To provádí mravenec, který právě dosáhl řešení. Mravenec ještě před tím, než porovná své řešení s ostatními mravenci, prohledá lokální okolí svého řešení, zda nenajde ještě lepší. Lokální prohledání může být představováno například snahou nahradit poslední krok řešení jeho alternativami. Uveďme, že tuto část algoritmu lze vynechat. Záleží na vhodnosti užití vzhledem ke zvolené úloze.

Jakmile všichni mravenci naleznou svá řešení, je nutné je porovnat a nalézt to nejlepší. K tomu slouží fitness funkce (viz kapitola 3.1).

3.3 Optimalizace hejnem částic

Particle swarm optimization (PSO, optimalizace hejnem částic) je evoluční výpočetní technika vyvinutá v roce 1995. Původním záměrem bylo simulovat nepředvídatelné tvary ptačího hejna, avšak počáteční simulace byly modifikovány k multidimenzionálnímu vyhledávání, eliminaci vedlejších proměnných a akceleraci dle vzdálenosti. Nakonec bylo usouzeno, že algoritmus lze považovat za optimalizační. [10]

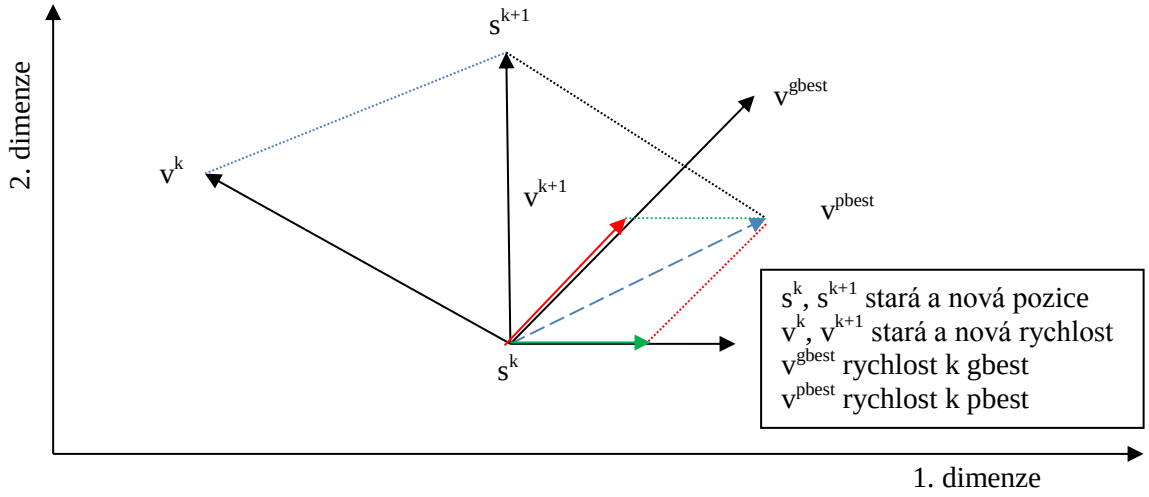
PSO algoritmus je adaptivní algoritmus, založený na sociálně psychologické metafoře, kde se populace jednotlivců, nazývaných částice, adaptuje stochastickým následováním předešle úspěšných oblastí.

V základu je možné pozorovat několik významných rysů, které odpovídají genetickým algoritmům. Shrňme je tedy (viz [11]):

- Obě metody inicializují populaci s náhodnými řešeními.
- Obě metody používají hodnotící fitness funkci k ohodnocení, jak kvalitního či dobrého řešení je momentálně dosaženo.
- Lze také považovat obě metody za generační v tom smyslu, že obě opakují tu samou množinu procesů po předem určenou jednotku času.

Jak jsme tedy uvedli, základní jednotkou výpočtu je částice. Částice jsou umístěny obecně v prostoru s libovolným počtem dimenzí. Tento prostor představuje množinu možných řešení. Každá částice si udržuje koordináty ve vymezeném n-dimenzionálním prostoru, které jsou spojeny s nejlepším řešením, kterého částice dosáhla a to včetně hodnoty tohoto řešení. Tuto hodnotu nazýváme pbest. Krom toho algoritmus udržuje ještě jedno řešení (koordináty a hodnotu) – globálně nejlepší dosažené řešení v celé populaci částic. Tato lokace je nazývána gbest.

PSO má dva primární operátory, konkrétně změnu rychlosti (velocity update) a změnu pozice (position update) částice. V každé iteraci je zjištěna nová hodnota rychlosti každé částice. Tato je vypočtena na základě rychlosti částice z předchozí iterace, vzdálenosti od předchozího nejlepšího řešení dané částice a vzdálenosti od globálního nejlepšího nalezeného řešení. Poznamenejme, že zde chápeme rychlost jako vektorovou veličinu, tedy krom velikosti je vypočítáván i směr. Jakmile známe novou velikost a směr rychlosti, můžeme spočítat i novou pozici částice v prohledávaném prostoru. Následující obrázek nám ilustruje, jak výpočet nové pozice probíhá. V této ukázce budeme uvažovat pouze dvoudimenzionální prostor.

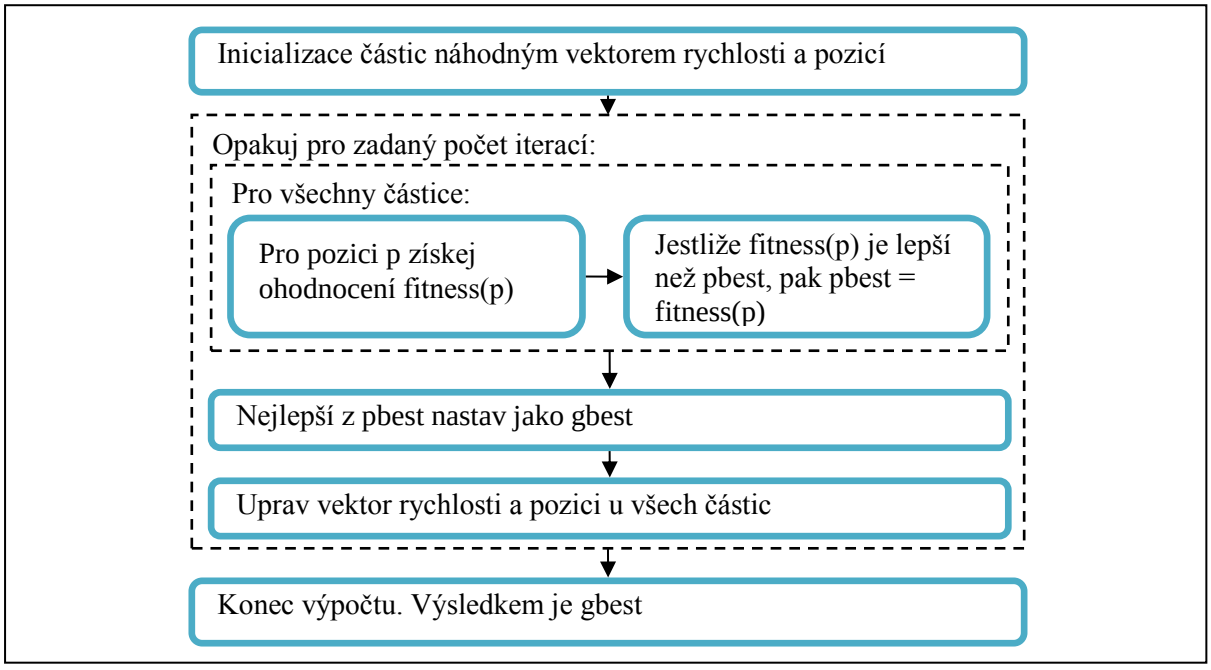


Obrázek 4: Koncept pro výpočet nové pozice částice v PSO

Standardní verze PSO má tendenci tíhnout k tomu, že výkyvy v rychlosti se stávají větší a větší, dokud není aplikována nějaká metoda na nápravu exploze rychlosti. Obvyklá metoda pro prevenci této exploze je předem definovaný parametr (označme jej například V_{max}), který zabraňuje překročení této hodnoty v každé dimenzi. [11]

Základní koncept obsahuje také náhodné váhování rychlosti pomocí oddělených náhodných hodnot, které jsou generovány za účelem akcelerace sledování pbest a gbest lokací.

Shrňme tedy obecný postup při výpočtu následujícím diagramem.



Algoritmus 2: Diagram znázorňující obecný postup PSO

Vidíme, že po prvotní náhodné inicializaci částic se po předem zadaný počet iterací vždy pro každou částici zjistí hodnota pozice, na které se nachází. Pokud je lepší než její dosavadní lokální maximum, tak si ji zapamatuje. Nejlepší z lokálních maxim v rámci propojených částic se nastaví jako globální maximum a pro každou částici se upraví vektor rychlosti a vypočítá se nová pozice. Jakmile dosáhneme zadaného počtu iterací, tak za výsledek považujeme globální maximum.

Uveďme si tedy konečně klíčovou rovnici pro výpočet nového vektoru rychlosti. Tato rovnice má následující tvar:

$$v_{id}(t) = v_{id}(t-1) + c_1 \cdot rand() \cdot (p_{id} - x_{id}(t-1)) + c_2 \cdot rand() \cdot \left(p_{gd} - x_{id}(t-1)\right)$$

Proměnná v_{id} je rychlost, t je číslo iterace, c_1 a c_2 konstanty reprezentující stochastickou akceleraci k pbest a gbest, $rand()$ náhodná hodnota s uniformním rozložením mezi 0 a 1. Dále p_{id} je pbest částice i, p_{gd} je gbest a x_{id} je současná pozice.

Jak jsme si již uvedli, budeme se snažit tuto rovnici řídit tak, aby hodnoty nepřerostly určité meze. Tedy nejjednodušším mechanismem je nastavení pevného maxima pro každou dimenzi. Pokud tedy hodnota překročí danou mez ať v kladném či záporném směru, tak je její hodnota nastavena právě na toto maximum.

$$v_{id} \in (-V_{max}; V_{max})$$

tedy

$$jestliže\ v_{id} > V_{max}\ pak\ v_{id} = V_{max}$$
$$jestliže\ v_{id} < -V_{max}\ pak\ v_{id} = -V_{max}$$

Tento mechanismus je součástí standardního PSO. Uveďme si nyní dvě nejznámější modifikace standardního algoritmu.

3.3.1 Inertia

První používanou modifikační metodou je tzv. Inertia autorů Eberharta a Shi [10]. Jedná se o inerciální váhový faktor (označme jej w), uplatňovaný při výpočtu nové rychlosti. Tento je v průběhu postupně lineárně snižován z prvotně relativně velké hodnoty směrem k malé, tak aby výpočet dával co nejpřesvědčivější výsledky v porovnání s pevným váhovým faktorem. Potom větší parametr $w(t)$ znamená lepší globální vyhledávací schopnosti a naopak menší hodnota parametru $w(t)$ lepší lokální vyhledávací schopnosti. Tedy algoritmus se snaží postupně přecházet od velkých skoků k menším a tedy od globálního prohledávání přecházet k prohledávání místnímu. Rovnice se systémem Inertia má následující tvar:

$$v_{id}(t) = w(t) \cdot v_{id}(t-1) + c_1 \cdot rand() \cdot (p_{id} - x_{id}(t-1)) + c_2 \cdot rand() \cdot \left(p_{gd} - x_{id}(t-1)\right)$$

Pro $w(t)$ platí lineární redukce. Typicky se $w(t)$ snižuje rovnoměrně od počáteční hodnoty w_{start} až po koncovou hodnotu w_{end} při každé iteraci. Obvyklé hodnoty pro tyto hraniční body jsou 0.9 pro w_{start} a 0.4 pro w_{end}. Uveďme rovnici pro výpočet $w(t)$:

$$w(t) = \frac{(T_{max} - t) \cdot (w_{start} - w_{end})}{T_{max}} + w_{end}$$

T_{max} je maximální povolený počet iterací algoritmu PSO. Tedy $w_{start} = w(0)$ a hlavně platí $w_{end} = w(T_{max})$.

3.3.2 Zužující koeficient

Druhou modifikací je PSO se zužujícím (omezujícím) koeficientem (PSO with constriction coefficient). Tento byl vyvinut Maricem Clercem v roce 1999 [10]. Ten modeloval interakce hejna částic za pomoci množiny komplikovaných rovnic. Faktem je, že amplituda oscilací částice se snižuje s tím, jak se částice zaměřuje na lokální nejlepší pozici a na předchozí nejlepší pozici v sousedství. Zužující koeficient zabraňuje kolapsu za přítomných sociálních podmínek. Částice bude oscilovat kolem váženého průměru lokálního a globálního maxima. Pokud budou blízko sebe předchozí nejlepší pozice pbest a nejlepší pozice v rámci okolí, částice bude používat lokálního prohledávání. Naopak jestliže pozice předchozí nejlepší pozice a nejlepší pozice nalezené v rámci sousedních částic jsou daleko od sebe, pak částice bude upřednostňovat expanzivní globální prohledávání. Pokud se

během prohledávání mění vzájemné rozpoložení předchozího nejlepšího stavu a nejlepšího stavu v rámci sousedních částic, posouvá se od lokálního prohledávání směrem ke globálnímu. Zužující koeficient tedy balancuje směřování algoritmu k lokálnímu či globálnímu prohledávání v závislosti na platných sociálních podmínkách. Uveďme si tedy znění rovnice pro výpočet nového vektoru rychlosti částice s touto modifikací:

$$v_{id}(t) = \chi \cdot \left[v_{id}(t-1) + c_1 \cdot rand() \cdot (p_{id} - x_{id}(t-1)) + c_2 \cdot rand() \cdot \left(p_{gd} - x_{id}(t-1)\right)\right]$$

kde χ je právě náš zužující koeficient. Ještě tedy uvedeme výpočet tohoto koeficientu:

$$\chi = \frac{2k}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}$$

přičemž $\varphi = c_1 + c_2$ a současně $\varphi > 4$. Nejčastěji jsou dále parametry nastaveny na následující hodnoty: $k = 1$, $c_1 = c_2$, $\varphi = 4.1$. Tedy obvyklá hodnota koeficientu je $\chi = 0.73$.

3.3.3 Výpočet nové pozice

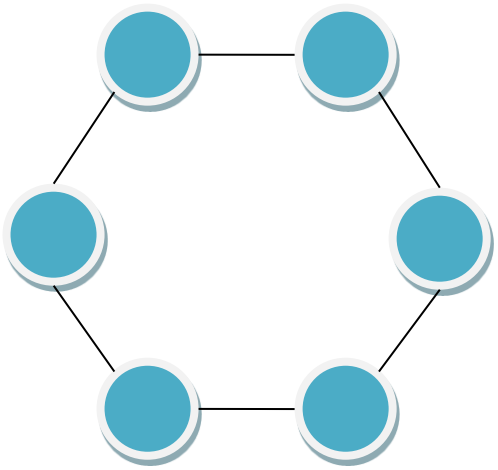
Konečně se seznamme s rovnicí pro výpočet nové pozice z nově vypočtené hodnoty rychlosti a staré pozice, která je společná pro všechny uváděné varianty:

$$x_{id}(t) = x_{id}(t-1) + v_{id}(t) \cdot \Delta t$$

Uveďme, že Δt určuje dobu letu částice, kterou když vynásobíme rychlostí částice $v_{id}(t)$, získáme uraženou vzdálenost částice. Obvykle se Δt uvažuje $1s$ (pokud uvažujeme rychlost v ms^{-1}). Proto lze někdy v literatuře nalézt i fyzikálně nepřesný vztah bez Δt.

3.3.4 Topologie

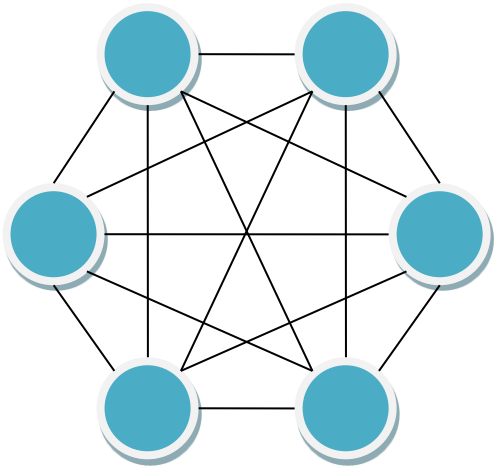
Dále se seznámíme se třemi možnými topologiemi sousedství, které lze pro PSO nejběžněji použít. Výběr topologie určuje částici, jejíž hodnota bude brána jako p_{gd}. První topologie je kruh, kdy je každá částice připojena ke k topologicky nejbližším částicím (viz obrázek). Poté p_{gd} je pro každou částici brána jako nejlepší hodnota její a k okolních částic. Typicky má k pro kruhovou topologii hodnotu 2.



Obrázek 5: Kruhová topologie pro PSO

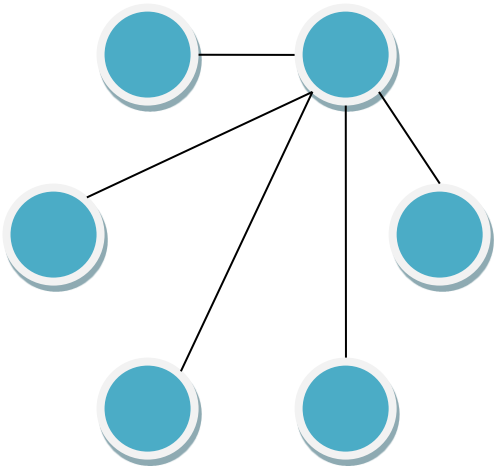
Další topologií je topologie hvězda. Je to tzv. plné propojení, kdy je každá částice propojena se všemi ostatními částicemi. Někdy bývá tato topologie ve spojené s PSO nazývána též jako globálně

nejlepší topologie (global best topology). Logicky pak p_{gd} je nejlepší hodnotou ze všech částic v populaci. Vzhled topologie ilustruje následující obrázek.



Obrázek 6: Topologie hvězda pro PSO

Poslední topologie, kterou si zmíníme, bude kolo (wheel topology). Zajímavé na této topologii je to, že efektivně vzájemně izoluje částice. Jedna částice je pak vybrána za ústřední, ohniskovou (fokální). Skrze tuto speciální částici komunikují ostatní částice. Potom p_{gd} je lepší z hodnot běžné částice a naší speciální ohniskové částice. Ilustrujme opět obrázkem. Fokální částice pracuje tak, že změní svou polohu na základě informací od ostatních částic a pokud dosáhne lepšího řešení, tak jej rozšíří k ostatním částicím.



Obrázek 7: Topologie kolo pro PSO

3.4 Genetické algoritmy

Začněme jednou definicí genetického algoritmu: Algoritmus je série kroků pro vyřešení problému. Genetický algoritmus je metoda pro řešení problémů, která používá genetiku jako model řešení problému. Je to vyhledávací technika pro nalezení aproximačních řešení optimalizačních a vyhledávacích problémů. [12]

Uveďme ještě jinou definici, která více přibližuje genetickou podstatu těchto algoritmů. Genetické algoritmy jsou rodinou výpočetních modelů, které jsou inspirovány evolucí. Tyto

algoritmy kódují potenciální řešení specifického problému do datové struktury, která je podobná chromozomu. Na tyto pak aplikují rekombinační operátory při zachování kritických informací. Na genetické algoritmy se často nazírá jako na optimalizační, ačkoliv rozsah problémů, na které lze genetické algoritmy aplikovat, je velmi široký. [13]

Jak již název napovídá, genetické algoritmy jsou inspirovány přírodními procesy. Snaží se napodobit Darwinův evoluční proces. Základními pojmy tedy jsou jedinec, kterého bude reprezentovat jeden chromozom, dále je to pojem populace, což je skupina jedinců, potažmo skupina chromozomů. Následující odstavec bude určen k seznámení se základními biologickými pojmy, které jsou vzorem pro počítačový model.

Všechny živé organizmy se skládají z buněk, kdy každá buňka obsahuje tu samou množinu jednoho či více chromozomů, které slouží jako předpis pro organizmy. Tato sekvence se nazývá DNA. Chromozom může být konceptuálně rozdělen na dílčí jednotky nazývané geny, přičemž každý kóduje jistý protein. Na gen lze pohlížet jako kódování pro jeden rys, jako je například barva očí. Různá „nastavení“ rysů (modrá, hnědá či zelená barva očí) nazýváme alely. Každý gen je umístěn na specifickém místě v chromozomu. [14]

Nyní si připomeňme ještě několik biologických pojmů, které se týkají operátorů užívaných v genetických algoritmech. Organismy v přírodě můžeme dělit na haploidní či diploidní. Haploidní organizmy mají nespárované chromozomy, naproti tomu diploidní mají chromozomy spojeny v pár. Pokud se organizmy v přírodě rozmnožují sexuálně, pak jsou to ve většině případů diploidní organizmy a při rozmnožování poskytují polovinu chromozomového páru pro vytvoření nového jedince a tedy pro svou reprodukci. Avšak v klasickém genetickém algoritmu se užívá pouze jednoho chromozomu pro jednoho jedince, kterého bychom tak mohli nazvat haploidním.

Reprodukce je právě podstatou genetických algoritmů. Při reprodukci jsou v genetických algoritmech vybíráni rodiče (selekce), kteří se spárují. Základní operací při sexuální reprodukci je křížení (crossover) rodičů, které je v genetických algoritmech také pochopitelně obsaženo. Při křížení může dojít k náhodné mutaci nějakého genu, což znamená jistou změnu v genetickém předpisu potomstva. Takto jsou vytvořeni noví jedinci (potomci svých rodičů) s novými vlastnostmi, kteří jsou určitým způsobem začleněni do stávající populace.

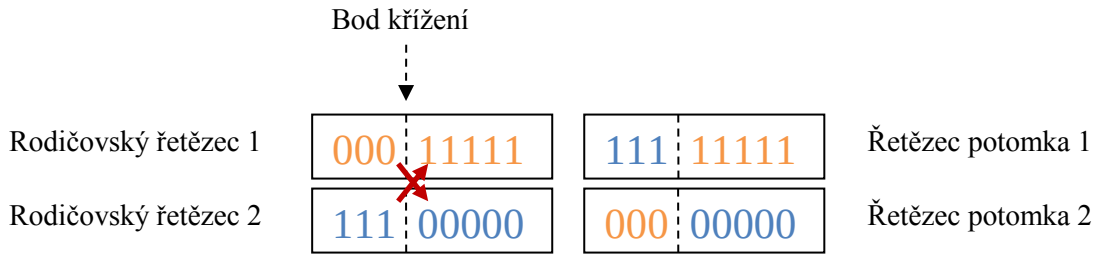
Popišme si tedy v následujících kapitolách tyto činnosti realizované během reprodukce.

3.4.1 Křížení

Rekombinace či křížení znamená následující postup: každý z rodičů promění chromozomový pár v dvojici chromozomů nazývaných gamety. Pak spojení dvou gamet vytvoří opět kompletní chromozomový pár. Při haploidní reprodukci jsou geny vyměňovány mezi dvěma rodiči prostřednictvím jednovláknových chromozomů. [14]

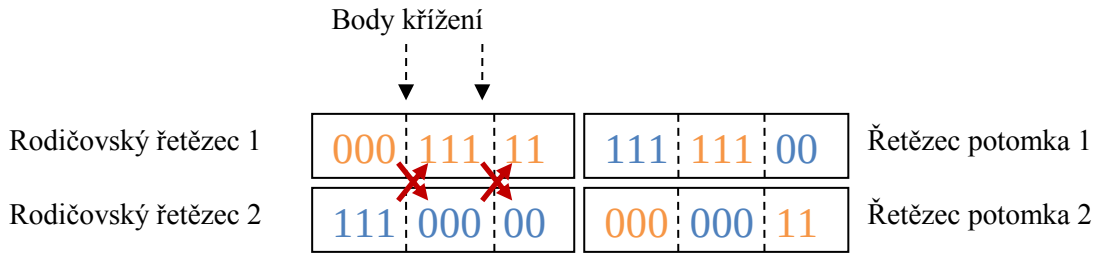
Shrňme tedy informace o křížení používané v genetických algoritmech. Pro křížení je nutné mít dva jedince, kteří si vzájemně vymění genetické informace v jisté formě chromozomu. Jinak řečeno, křížení je proces, kdy se kombinují dva řetězce za účelem dosažení lepšího (lépe ohodnoceného) řetězce. Původní řetězce nazýváme rodičovské, naopak řetězce, které jsou výsledkem, nazýváme potomky. Rekombinační proces tudíž vytváří rozdílné jedince v následujících generacích prostřednictvím kombinování materiálu dvou jedinců předchozí generace, viz [15]. Z těchto faktů můžeme vyvodit princip tohoto procesu. Pokud nám řetězec představuje genetickou informaci, pak budeme předpokládat, že jeho jisté podřetězce mají obecně odlišné složení v porovnání se stejnými podřetězci (co se týče umístění) v jiném řetězci. Tato rozdílnost znamená, že jeden z odpovídajících podřetězců dvou chromozomů má lepší ohodnocení vzhledem k našemu cíli. Pokud budeme měnit tyto subsekvence mezi řetězci (křížit je), pak obecně dostáváme nové řetězce (potomky), z nichž některé mohou mít lepší ohodnocení. Uveďme si graficky názornou ukázku, kde máme jeden bod

křížení. Na tomto obrázku je vidět, že každý potomek se skládá z podřetězců obou rodičů. Pokud by se ukázkové sekvence hodnotily jako binární čísla, pak jsme křížením dvou průměrných potomků získali potomka s vyšším a druhého s nižším ohodnocením, než jsou rodiče. Křížení nám tedy nezaručuje, že potomci vzniklí křížením budou mít požadované lepší ohodnocení. Ale mít jej mohou (viz příklad).



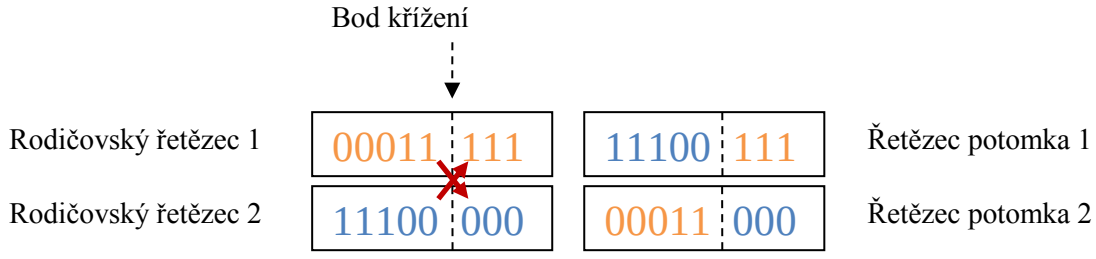
Obrázek 8: Ilustrace křížení v genetických algoritmech

Při genetických algoritmech můžeme také používat více bodů křížení. Při násobném křížení se vyměňované sekvence střídají s nevyměňovanými. Tato možnost může, ale také nemusí přinést zlepšení. Opět závisí na konkrétní situaci.



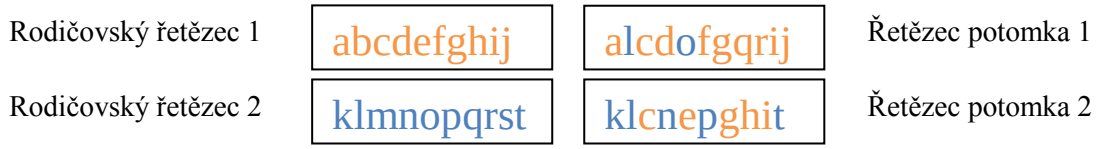
Obrázek 9: Vícebodové křížení v genetických algoritmech

Body křížení mohou být rozloženy rovnoměrně v řetězci, ale také náhodně. Úspěšnost křížení závisí také na tom, zda bod křížení je na poloze, kde je to „vhodné." Ukažme si, jak se změní náš příklad s jednobodovým křížením s jinou pozicí. Je vidět, že dosáhneme zcela rozdílných výsledků.



Obrázek 10: Vliv jiné pozice křížení v genetických algoritmech

Uveďme si ještě jeden typ křížení a to uniformní křížení. Nejedná se úplně přesně o výměnu úseků mezi dvěma řetězci jako v předchozím případě. Každý potomek se vytváří gen po genu a to tak, že každý odpovídající gen přebírá s pravděpodobností 0.5 od prvního rodiče a s pravděpodobností 0.5 od druhého. Proto například může být jeden gen zkopírován do obou potomků od jednoho z rodičů a odpovídající gen od druhého může být ztracen.



Obrázek 11: Uniformní křížení v genetických algoritmech

3.4.2 Mutace

Uveďme si další významnou součást genetických algoritmů. V přírodě napomáhá variabilitě při vývoji organizmů. Pokud si uvědomíme, že máme omezenou množinu genů, případně alel, které jsou k dispozici, pak jejich kombinacemi můžeme dosáhnout velkého, ale omezeného počtu kombinací. Navíc pokud dochází ke změnám pouze při křížení, tak tento proces může být pomalý. Ale například vliv chemických, fyzikálních či biologických mutagenů zvyšuje pravděpodobnost mutace. Právě proces mutace je tím dalším evolučním nástrojem.

Mutace zahrnuje modifikaci hodnoty každého genu s jistou pravděpodobností p_m (pravděpodobnost mutace). Mutace v genetických algoritmech má za úkol obnovit ztracený či nalézt neobjevený, neprozkoumaný genetický materiál v populaci, za účelem zabránění předčasné konvergence genetického algoritmu k suboptimálním řešením. [16]

Mutace přidává novou informaci náhodnou cestou do genetického vyhledávacího procesu a zásadně pomáhá vyhýbat se uvíznutí v lokálním optimu. Je to operátor, který vnáší rozmanitost do populace, kdykoli populace tíhne k tomu, aby se stala homogenní díky opakovanému užívání reprodukce a křížení. [15]

Pokud budeme uvažovat mutaci na úrovni počítačového modelu, pak nejběžnější je mutace na bitové úrovni. V okamžicích, kdy je bitový řetězec kopírován, je každý bit s jistou pravděpodobností invertován. Mutace tedy tvoří protiklad ke křížení, tedy k řízenému prohledávání stavového prostoru. Principem je to, že malá změna (například inverze jednoho bitu) pomáhá určit nový bod v sousedství současného bodu, čili je vhodná pro lokální prohledávání v okolí současného stavu. A samozřejmě může napomoci k nalezení stavu, který by nemusel být běžnými kombinacemi dosažen. To je právě dáno náhodností mutace. Ukažme si ilustrativní příklad na několika bitových řetězcích. Mějme tedy populaci tří jedinců reprezentovanými následujícími řetězci:

01010
00110
11110

Pokud by operátor křížení byl představován bitovou operací OR a pro získání optimálního řetězce (11111) by bylo zapotřebí, aby na poslední pozici řetězce byl bit s hodnotou 1, pak by nebylo pouhým křížením možné takové situace dosáhnout. Ale právě nenulová pravděpodobnost mutace p_m pro poslední bit nám dává šanci na dosažení optimálního řešení. Hodnota pravděpodobnosti mutace bývá většinou volena poměrně malá, řekněme například několik desetin procenta. Často se ale při malé populaci užívá větší pravděpodobnosti mutace.

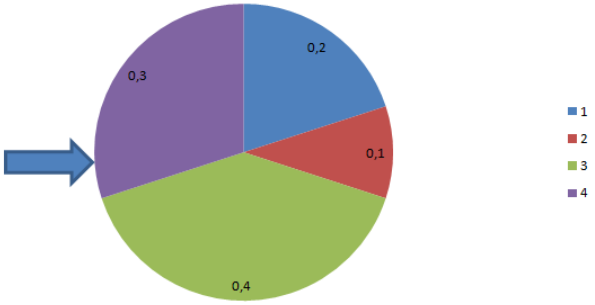
3.4.3 Selekce

Selekce (výběr) vybírá dobře hodnocené řetězce v populaci a vytváří genetický fond pro páření. Někdy je selekce v genetických algoritmech označována jako operátor selekce. Proces přirozené selekce způsobuje, že jedinci, kteří kódují úspěšné řešení, jsou vybíráni s větší frekvencí.

Pro tvorbu nové generace je reprodukce jedinců současné populace nezbytná. Abychom získali lepší jedince, tak by měli být vybírání nejlépe hodnocení jedinci předchozí populace. [15]

Selekce je v genetických algoritmech obvykle první operátor aplikovaný na populaci. Tento může mít více podob, ale základní princip zůstává stejný. Jedinci z populace (všichni či část) jsou vybráni ke křížení s daným pravděpodobnostním rozložením. Rozumějme tím, že jedinci s lepším ohodnocením budou mít větší pravděpodobnost výběru. Přehled různých modifikací můžeme nalézt například v [17]. Ukážeme si ty základní.

První metodou výběru je ruleta. Je to proporcionální reprodukční operátor, kde jedinec s vyšším ohodnocením má tím větší pravděpodobnost na výběr ke křížení, čím větší (lepší) hodnotu má fitness funkce. Název funkce bývá odvozen od ilustrace tohoto operátoru formou hracího kola, na němž jsou výseče o velikosti přímo úměrné velikosti hodnotící funkce pro jednotlivé jedince. Kolo je vybaveno ukazatelem (šipkou). Kolo se roztočí náhodnou rychlostí, a jakmile se zastaví, ke křížení je vybrán jedinec, na jehož výseč ukazuje šipka. Točit je potřeba tolikrát, jako je dvojnásobek potřebného počtu křížení (jedno otočení vybere jednoho jedince do páru potřebného pro křížení).



Jedinec	Hodnota fitness funkce	Pravděpodobnost výběru
1	10	0,2
2	5	0,1
3	20	0,4
4	15	0,3

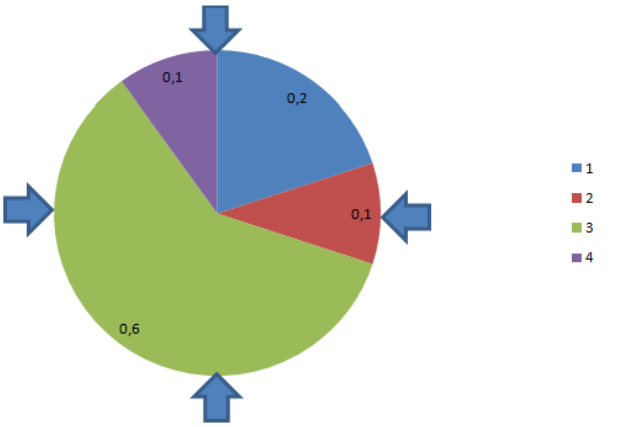
Obrázek 12: Ilustrace pro výběr pomocí operátoru ruleta

Ruletu lze programově implementovat například tak, že každému jedinci je přiřazena pravděpodobnost výběru p_i v rozmezí od 0 do 1 a to tak, že součet všech pravděpodobností je rovna jedné. Prvnímu jedinci jsou přiřazeny dolní hranice 0 a horní hranice p_1, druhému $(p_1; p_1 + p_2\rangle$ a podobně dále. Nakonec je vygenerována hodnota v rozmezí $\langle 0; 1\rangle$ a do kterého intervalu padne, ten jedinec je vybrán.

Ukažme si rovnici pro výpočet pravděpodobnosti výběru jedince i (tj. p_i) z populace o velikosti n na základě znalosti hodnoty fitness funkce F_i:

$$p_i = \frac{F_i}{\sum_{i=1}^{n} F_i}$$

Velmi podobný operátor výběru je stochastické univerzální vzorkování. To je téměř totožné s ruletou, ale s tím rozdílem, že při statickém univerzálním vzorkování se kolem rulety točí pouze jednou. Na kole je totiž rovnoměrně rozmístěno tolik ukazatelů, kolik řetězců pro křížení je třeba vybrat.



Jedinec	Hodnota fitness funkce	Pravděpodobnost výběru
1	10	0,2
2	5	0,1
3	30	0,6
4	5	0,1

Obrázek 13: Operátor výběru stochastickým univerzálním vzorkováním

Dále je vhodné si uvést algoritmus výběru elity. Elita je dána sestupným uspořádáním populace dle hodnoty fitness funkce jedinců. Z populace je vybráno pouze jisté, předem dané, množství jedinců s nejlepším hodnocením. Obvykle se jedná o 10-50%. Řetězce reprezentující tyto jedince jsou připuštěny ke křížení.

Posledním zmiňovaným operátorem výběru je algoritmus turnaje. Je náhodně vybrán předem daný počet jedinců z populace. Z těch je vybrán ten s nejlepším ohodnocením. Tento postup opakujeme, dokud nezískáme potřebný počet rodičů.

3.4.4 Reprezentace řešení

Kódovací schéma je klíčová otázka v návrhu genetických algoritmů, jelikož může významně omezit prostor informací, které je možné systémem sledovat. Pro zajištění výkonu algoritmu je nutné uložit informace o problému ve formě chromozomu. Obecně genetické algoritmy zahrnují multimnožinu chromozomů. Připomeňme, že každý chromozom reprezentuje řešení problému. Chromozom je obvykle reprezentován jako řetězec proměnných, kde každý element je nazýván genem. Proměnné mohou být vyjádřeny binárně, reálnými čísly či jinými formami, přičemž rozsah je obvykle dán problémem. [18]

Nejčastější reprezentace, se kterou je možné se setkat, jsou binární řetězce. Důvodem je jednoduchost a sledovatelnost. Přesto existují úlohy, při nichž se při užití bitového řetězce vyskytují potíže. Například při permutačních problémech je nutné zachovat počet jednotlivých genů (smí se jen přeskládat), proto se zde používají například indexové tabulky.

3.4.5 Tvorba nové populace

V okamžiku, kdy umíme vytvořit nové potomky, je nutné znát model pro vytvoření populace. Vždy v každé inkrementaci je vytvářena nová populace obecně za pomoci původních jedinců i nových potomků. Základní přístupy lze nalézt například v [19].

Prvním modelem je standardní generační model. Jak je patrné již z názvu, je kladen důraz na generace, což znamená, že všichni jedinci populace jsou nahrazeni novými potomky. Zde je snaha o maximální urychlení vývoje, nehledě na to, zda jsou noví potomci lépe hodnoceni fitness funkcí než předchozí generace.

Druhý základní model je inkrementační (steady state model). Na rozdíl od předchozího případu, zde je nahrazen pouze jeden jedinec původní populace nově vytvořeným jedincem. Evoluce je v tomto případě pomalejší, avšak lépe kontrolovatelná.

Očividně jsou oba předchozí modely krajními případy tzv. modelu s překrytím generací. Tento je jistým kompromisem, přičemž je v populaci nahrazena část jedinců novými.

S tvorbou nové populace přímo souvisí způsob zařazení nových potomků do původní populace (reinsertion), tj. jakým způsobem realizujeme výše zmíněné modely. Blíže například v [20].

Nejblíže generačnímu modelu je případ, kdy je počet nových jedinců stejný jako původních a plně je nahradí (pure reinsertion). Další možností je vygenerovat více potomků, než je v původní populaci, avšak do nové populace zařadit pouze potomky s lepší hodnotou fitness funkce (fitness-based reinsertion). Tento výběr se snaží jít pozitivním směrem vývoje vzhledem k fitness funkci.

Pro model s překrytím, či pro inkrementační model, odpovídá možnost vygenerování menšího počtu potomků, než je velikost původní populace, a vyměnit jimi náhodně vybrané rodiče s rovnoměrným rozložením pravděpodobnosti výběru (uniform reinsertion). Nebo můžeme opět přihlédnout k fitness funkci a nahradit nejhůře ohodnocené rodiče (elitist reinsertion).

Poznamenejme, že výše zmíněné případy patří mezi globální začleňování (global reinsertion). Tyto modely jsou vhodné pro globální výběrové metody.

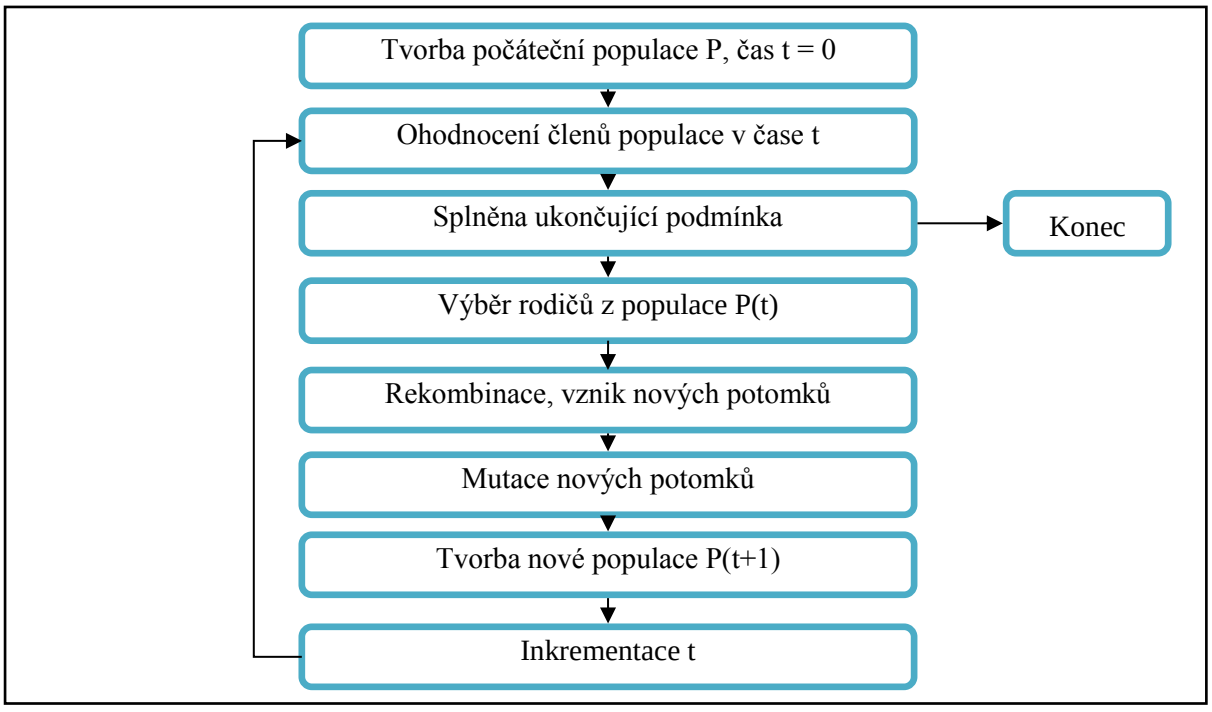
3.4.6 Algoritmus výpočtu

Nyní již známe všechny základní stavební kameny genetických algoritmů, proto si nakonec ukažme i výpočetní model, který je všechny zahrnuje.

Uveďme ještě, že obecně nemusí být jasné, zda jsme dosáhli již optimálního výsledku, proto ukončující podmínka algoritmu může buď zahrnovat určitou hodnotu fitness funkce, ale spíše se používá varianta pevného počtu iterací. Lze také použít jako ukončující podmínku počet iterací, kdy nedojde ke zlepšení fitness funkce populace, která je dána jako průměr fitness funkcí jedinců:

$$\bar{F} = \frac{\sum_{i=1}^{n} F_i}{n}$$

Pro ilustraci algoritmu můžeme použít grafické znázornění:



Algoritmus 3: Genetický algoritmus

3.5 Simulované žíhání

Algoritmy založené na simulovaném žíhání (simulated annealing) byly původně vyvinuty pro diskrétní optimalizační problémy. Jsou navrženy k překonání lokálních minim na cestě ke globálnímu minimu.

Pojem žíhání pochází z analogie k ochlazování kapalin či pevných látek. Ústředním tématem ve statistické mechanice je analýza chování látek při chladnutí. Při vysokých teplotách se molekuly hodně pohybují, ale jakmile se teplota snižuje, tento pohyb se ztrácí a molekuly mohou tíhnout k uspořádání do krystalické struktury. Toto krystalické uspořádání představuje stav systému s minimální energií. Zdůrazněme, že molekuly pouze mohou dosáhnout takové situace. Teplota samotná přímo nezaručuje, že substance dosáhla stavu s minimální energií. Abychom dosáhli tohoto stavu, ochlazování se musí odehrávat dostatečně nízkou rychlostí. Pokud je substance ochlazována příliš rychle, může být dosaženo amorfního či polykrystalického stavu, který není minimálním energetickým stavem substance. Tedy princip stojící za žíháním ve fyzických systémech je pomalé ochlazování substance za účelem dosažení minimální energie. [21]

Při simulovaném žíhání se minimalizace cenové funkce identifikuje s energií fyzikálního systému a prostor řešení je identifikován se stavovým prostorem. Prostor řešení optimalizačního problému je procházen metodou *pravděpodobnostního* vyhledávání hill climbing, kde velikost kroku je kontrolována parametrem T, který představuje teplotu ve fyzickém systému. Za pomoci pomalého snižování teploty směrem k nule dle zvoleného plánu, lze ukázat, že globálně optimální řešení se přibližují asymptoticky. V typickém vykonávaném simulovaném žíhání je počáteční teplota nastavena na dostatečně vysokou hodnotu. Nový stav (řešení) je generován inkrementálně ze současného stavu náhodným výběrem a navrhuje tak tah z množiny předdefinovaných tahů. Tah je změna, jejíž aplikace na současný stav vede k novému stavu. [22]

Současný stav označme C (Current) a následující stav po provedení kroku označme N (Next). Každému stavu X můžeme přiřadit hodnotu (cenu či energii) $value(X)$. Pro vlastní algoritmus je potom důležitá změna energie mezi současným a následujícím stavem:

$$\Delta E = value(N) - value(C)$$

Rozdíl mezi současným a novým stavem je důležitý pro rozhodnutí, zda nový stav může nahradit pro další iteraci aktuální stav. Pokud je hodnota (energie) nového stavu menší nežli současného, pak jej přijmeme, jinak o tom necháme rozhodnout náhodu. Ta je zprostředkována náhodně vygenerovaným číslem v intervalu $\langle 0; 1\rangle$ s uniformním rozložením. Pak nově vygenerovaný (horší) stav je přijat za aktuální pokud:

$$random(0; 1) < e^{-\frac{\Delta E}{T}}$$

Odtud tedy jasně vidíme, že pravděpodobnost přijetí nového stavu s větší energií, než má současný, je rovna:

$$p(N) = e^{-\frac{\Delta E}{T}}$$

Někdy bývá tento vztah pro pravděpodobnost přijetí nového stavu N formulován ještě s formálním ujištěním, že pravděpodobnost nepřekročí hodnotu 1:

$$p(N) = \min\left(e^{-\frac{\Delta E}{T}}, 1\right)$$

Parametr teplota T je tedy ústřední částí výpočetního modelu, přičemž pomáhá rozhodovat o přijímání nových stavů a také určuje ukončení výpočtu. Klesání teploty v algoritmu lze realizovat dvěma způsoby. Buďto je možné pokles teploty počítat v každé iteraci, nebo je možné „předpočítat“ jej pro každou iteraci a uložit například ve formě pole do tabulky. Parametr poklesu označme α. Potom výpočet nové teploty je:

$$T_{k+1} = T_k \cdot \alpha$$

Pro algoritmus je nutné, aby teplota klesala. Pak rozsah parametru α je následující:

$$\alpha \in (0; 1)$$

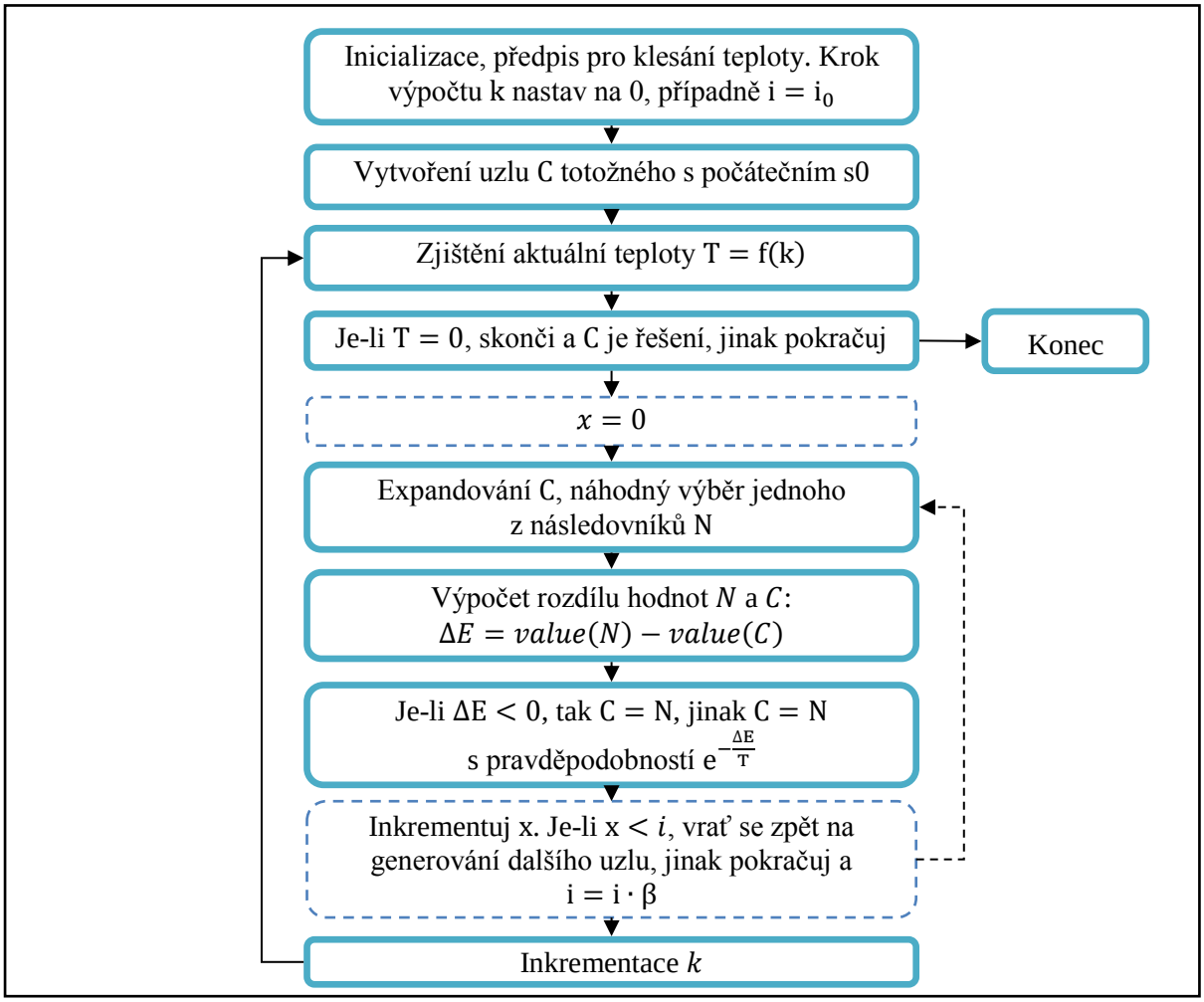
Nezbytnými kroky, které je třeba vyřešit ještě před zahájením výpočtu, jsou:

- Určit, jak se bude generovat počáteční stav S_0.
- Jaké budou hodnoty parametrů algoritmu.
- Kolik času počítače může výpočet spotřebovat.

Tyto volby mají významný dopad na kvalitu produkovaného řešení. Určit pro ně optimální nastavení není však možné, jelikož optimální nastavení nezávisí pouze na řešeném problému, ale také na instanci tohoto problému. Čas potřebný pro určení optimálního nastavení, je snad lépe použít pro delší běh algoritmu. [23]

3.5.1 Algoritmus

Na následujícím obrázku je shrnut algoritmus simulovaného žíhání. Nepřerušovaně je vyznačena základní varianta algoritmu. Z obrázku, konkrétně z části, kde se dle změny energie, případně dle zadané pravděpodobnosti, přijímá nový stav za aktuální, je vidět, že s vysokou počáteční teplotou je šance na přijetí nového stavu s horším ohodnocením (oproti současnému) vysoká. Avšak jakmile teplota dostatečně poklesne, výpočet se začne podobat metodě hill climbing, tedy se prakticky přijímají pouze následovníci, kteří mají lepší ohodnocení.



Algoritmus 4: Algoritmus simulovaného žíhání. Přerušované části nemusí být přítomny.

Základní algoritmus lze ještě upravit pro opakovaný výpočet (viz například [23]). Na obrázku jsou to části znázorněné přerušovaně. Tato změna spočívá v tom, že generování následujícího stavu a jeho přijímaní budeme opakovat po i iterací (i_0 označuje prvotní nastavení pro i). Zavedeme nový parametr β. Tento nám bude určovat nárůst počtu iterací i. Jeho hodnota musí být větší nebo rovna jedné (obvykle $\beta \in \langle 1; 2 \rangle$).

3.6 Posilované učení

Posilované učení (reinforcement learning) je interdisciplinární pojem (činnost, metoda), který se týká psychologie, statistiky, neurální a výpočetní vědy. Z našeho pohledu je posilované učení zainteresováno obzvláště v oblasti učení strojů a umělé inteligenci, konkrétně se jedná o snahu naučit umělého agenta určitým znalostem za pomoci odměn a trestů.

Posilované učení je problém agenta, který se musí naučit chování prostřednictvím interakcí s dynamickým prostředím stylem pokus omyl. [24]

Na druhou stranu ovšem můžeme nahlížet na toto učení jako na třídu úloh, spíše než na množinu technik. Potom pro řešení těchto úloh můžeme použít dva základní přístupy. Pro první přístup musíme znát množinu možných chování, přičemž se snažíme nalézt požadované řešení v tomto prostoru chování (tento postup vychází z práce genetických algoritmů a genetického programování a dalších vyhledávacích technik [24]). Druhou možností je užití statistických technik a technik dynamického programování pro odhadnutí vhodnosti (užitečnosti) jednotlivých akcí mezi stavy systému. My se zaměříme právě na tuto variantu.

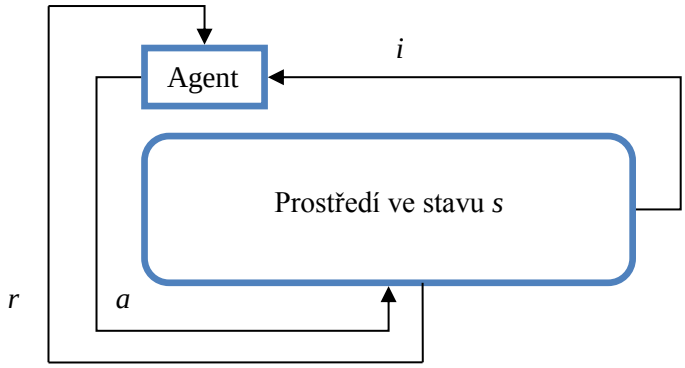
Posilované učení není jediným přístupem k získání agenta s požadovaným chováním. Dalšími metodami jsou například učení s učitelem (supervised learning) či plánování. Proto uveďme základní rozdíly, kterými se posilované učení od nich odlišuje. Na rozdíl od učení s učitelem, nejsou třeba páry stav-akce (či vstup-výstup), které jsou příklady buďto správného nebo nesprávného chování, na základě nichž by byla tvořena celková politika chování. Odměny za vykonání akcí (rozumějme přechodů mezi stavy) mohou být udělovány řídce, například až za dosažení určitého stavu (v případech, kdy nás zajímá pouze výsledek a ne to, jak jsme k němu došli). Na druhou stranu oproti plánování je možné použít posilované učení nejen pro případy, kde je třeba mít dobře (případně i zpětně) trasovatelný model. Plánovací metody totiž obecně potřebují explicitní model přechodové funkce $\delta(s, a)$, kde s stav a a je akce, jenž posilované učení nutně obsahovat nemusí.

Uveďme tedy oblasti použití. Posilované učení nabízí flexibilní přístup k návrhu inteligentních agentů v situacích, pro které nejsou plánování a učení s učitelem vhodné. Takovými situacemi jsou například problémy, v nichž významná část znalostí je nedostupná, či obtížně dosažitelná. [25]

Posilované učení lze tedy aplikovat například v oblasti hraní her (např. TD-GAMMON z roku 1992) či v oblasti kontroly robotů.

3.6.1 Model

V této kapitole popíšeme model posilovaného učení v jeho standardní variantě. Základními částmi modelu jsou prostředí a agent. Agent je s prostředím propojen pomocí vjemů (vstupů i) a následných akcí (a). Akce je tedy výstupem agenta, která mění stav (s) prostředí. Tato akce znamená změnu hodnoty, o níž je agent zpětně informován prostřednictvím signálu (r).



Obrázek 14: Ilustrace interakcí mezi agentem a prostředím při posilovaném učení

Agent se při každé iteraci rozhoduje, do jakého stavu se dostane provedením příslušné akce. Jeho snahou by mělo být, aby se snažil udržet dlouhodobý růst sumy hodnot signálů r. K dosažení takového chování slouží právě opakovaní přístupu pokus omyl, kdy signál r numericky určuje, jak byla použitá změna stavu vhodná (řekněme, že r je odměna).

Uveďme model více formálně (viz [24]). Model je složen z:

- množiny S diskrétních stavů prostředí,
- množiny A akcí agenta,
- množiny skalárních posilovacích signálů (typicky s hodnotami 0, 1 nebo s reálnými hodnotami).

Dále uveďme, že součástí modelu je vstupní funkce I, která určuje, jakým způsobem agent vnímá stav prostředí (zjednodušeně se dá říci, že se jedná o identifikační funkci pro stav).

Formálněji je úkolem agenta nalezení politiky π, která mapuje stavy prostředí na akce, které maximalizují dlouhodobý běh měření signálu r. Optimální politika π^* může být definována více způsoby, ale typicky se jedná o politiku, která má největší kumulativní součet odměn přes všechny stavy:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} V^{\pi}(s), (\forall s \in S),$$

kde $V^{\pi}(s)$ je kumulativní odměna při politice π a stavu s. [25]

Pro výpočet $V^{\pi}(s)$ se často využívá discount faktoru $\gamma \in \langle 0; 1 \rangle$. Potom je $V^{\pi}(s)$ dáno součtem vážených odměn r_t v časových okamžicích t v rámci nekonečného horizontu:

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Shrňme tedy, že úlohou posilovaného učení je použití pozorovaných odměn k naučení optimální (nebo jí blízké) politiky pro dané prostředí. [26]

Učení optimální strategie V^* pak můžeme realizovat ohodnocením stavů metodou dynamického programování, tj. iterativním řešením Bellmanových rovnic (viz dále).

3.6.2 Typy posilovaného učení

Posilované učení zahrnuje více přístupů, které můžeme dělit z několika pohledů. Základní možností je rozdělit metody na pasivní a aktivní (viz [26]).

Pasivní učení spočívá v tom, že je zadána fixní agentova politika. Z toho nám plyne, že hlavním úkolem při procesu učení je pak určení ohodnocení stavů. Tento proces může samozřejmě zahrnovat učení se modelu prostředí.

Při aktivním učení se agent musí ovšem naučit také to, co má dělat. V tomto ohledu musí maximálně využívat průzkumu prostředí.

Na druhou stranu lze metody členit z hlediska toho, zda potřebují ke své činnosti model prostředí (model-based a model-free). Dále také můžeme rozlišovat online učení (interakce s reálným prostředím) či offline učení (využívá model prostředí).

Konečně můžeme mít tzv. on-policy učení (používá k učení akci, kterou agent použil) nebo off-policy, kdy je použita nejlepší přípustná akce (nemusela být agentem užita).

3.6.3 Temporal-difference learning

Temporal-difference (TD) learning spadá do kategorie pasivního model-based učení. Kromě TD-learning metody můžeme mezi pasivní přístupy řadit například ADP learning (viz [26]).

Název temporal difference vychází z toho, že se k výpočtu ohodnocení stavu využívá rozdílu mezi dvěma následujícími stavy. Ukažme si aktualizační rovnici pro ohodnocení stavu metodou TD-learning:

$$U^{\pi}(s) = U^{\pi}(s) + \alpha\big(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s)\big),$$

kde $U^{\pi}(s)$ je ohodnocení stavu s při použití strategie π, $R(s)$ odměna za dosažení stavu s, γ discount faktor určující vliv ohodnocení následujícího stavu s' a konečně α je koeficient učení (learning rate).

Představme si, že se nacházíme ve stavu s, vybereme akci a dle politiky $\pi(s, a)$, čímž se dostaneme do stavu s' a získáme odměnu r. Navíc si představme pouze kosmetickou změnu, a to náhradu písmene V za U^{π}. Potom pozorovaná kvantita $r + V(s')$ je aktuální Monte Carlo vzorek Bellmanovy rovnice pro politiku π. Pokud bychom tento proces opakovali mnohokrát a zprůměrovali, pak bychom dostali pravou stranu Bellmanovy rovnice:

$$V^{\pi}(s) = \sum_{a} \pi(s, a) \sum_{s'} P^{a}_{s,s'} \left(R^{a}_{s,s'} + \gamma V^{\pi}(s')\right),$$

kde $\pi(s, a)$ je politika přiřazující stavu s akci a, $P^{a}_{s,s'}$ pravděpodobnost přechodu z s do s', kdy je vybrána akce a, $R^{a}_{s,s'}$ očekávaná střední hodnota odměny pro daný přechod a samozřejmě $V^{\pi}(s)$ je ohodnocení stavu s. [27]

Samozřejmě můžeme pro úplnost uvést optimální ohodnocení:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) = \max_{a} \sum_{s'} P^{a}_{s,s'} \left(R^{a}_{s,s'} + \gamma V^{\pi}(s')\right)$$

Nyní se vraťme k našemu koeficientu učení α. Jeho hodnota by se měla pohybovat v intervalu $\alpha \in (0; 1)$. Příliš malá hodnota znamená, že hodně lpíme na již spočteném ohodnocení, naopak koeficient α blížící se k 1 vede k rychlým změnám, kdy často radikálně měníme ohodnocení. Z tohoto důvodu často nebývá koeficient α fixní, nýbrž klesá s počtem průchodů stavem s.

Nyní uveďme algoritmus takového pasivního TD agenta ve formě funkce, která má na vstupu aktuální stav s' a odměnu r' a dále perzistentně uchovává tabulku ohodnocení U, fixní politiku π, tabulku návštěv stavů N_s (nastavenou na 0) a minulý stav s, akci a a odměnu r, které jsou iniciálně nastavené na $null$ (viz [26]):

function *TD_AGENT* (s', r')
    persistent π, U, N_s, s, a, r
    if s' *je nové* **then**
    $U[s'] = r'$
    if s *není null* **then**
    inkrementuj $N_s[s]$
        $U[s] = U^{\pi}(s) + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
    if s' *je terminální* **then**
        $s = null$
        $a = null$
        $r = null$
    else
        $s = s'$
        $a = \pi[s']$
        $r = r'$
    return a

Algoritmus 5: Funkce TD agenta

3.6.4 Q-learning

Alternativou k TD-learningu je Q-learning. Tento se namísto učení ohodnocení stavů učí reprezentaci akce-ohodnocení. Za tímto účelem budeme hodnotu akce a ve stavu s značit $Q(s, a)$. Zřejmě poté platí vztah vzhledem k ohodnocení U (viz [26]):

$$U(s) = \max_a Q(s, a)$$

Pokud se zamyslíme nad tím, že agent namísto stavů hodnotí akce v těchto stavech, dojdeme k závěru, že pro výpočet Q-funkce nepotřebujeme mít přechodový model. Z tohoto důvodu řadíme metodu Q-learning mezi model-free metody.

Tomu, že nepoužíváme model, musí být přizpůsobeno i aktualizační pravidlo – musí být postaveno na základě Q hodnot (viz [26]):

$$Q(s, a) = Q(s, a) + \alpha\left(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)$$

Aktualizační pravidlo je aplikováno vždy, když je vykonána akce a ve stavu s, čímž se dostane do nového stavu s'.

Jelikož se ve vzorci vyskytuje maximum, řadí se Q-learning mezi off-policy algoritmy. Pro úplnost uveďme, že ke Q-learningu má velmi blízko tzv. SARSA (viz [26]). Tento zástupce on-policy algoritmů užívá následující aktualizační pravidlo:

$$Q(s, a) = Q(s, a) + \alpha\big(R(s) + \gamma Q(s', a') - Q(s, a)\big)$$

Nakonec opět uveďme algoritmus ve formě funkce pro agenta používajícího Q-learning. Vstupem jsou aktuální stav s' a aktuální odměna r'. Perzistentními údaji jsou tabulka hodnot akcí Q, tabulka N_{sa} počtu použití akce a ve stavu s (obojí na počátku vynulované) a dále pak předchozí stav s, akce a a odměna r (počátečně nastavené na $null$):

function *Q_AGENT* (s', r')
persistent Q, N_{sa}, s, a, r
if s *je terminální* **then**
$Q[s, None] = r'$
if s *není* $null$**then**
inkrementuj $N_{sa}[s, a]$
$Q[s, a] = Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
$s = s'$
$a = \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a'])$
$r = r'$
return a

Algoritmus 6: Funkce pro Q-learning agenta

Funkce $f(u, n)$ je funkcí pro průzkum okolí (exploration funkce). Tato určuje, do jaké míry bude algoritmus tzv. greedy (tj. bude preferovat vysoké hodnoty u) oproti míře zvědavosti, což znamená, že algoritmus bude více tíhnout k tomu, aby preferoval akce, které byly méně často použité a mají malé n. Funkce $f(u, n)$ by měla být rostoucí s u a klesající s n. Existuje mnoho možností, jak takovou funkci definovat. Uveďme si proto jednoduchý příklad (viz [26]):

$$f(u, n) = \begin{cases} R^+ \text{ if } n < N_e \\ u \; otherwise \end{cases}$$

kde R^+ je optimistický odhad nejlepší možné odměny v libovolném stavu a N_e je fixní parametr, který určuje, kolikrát nejméně agent vyzkouší daný pár akce-stav.

4 Řešené úlohy

V této kapitole rozebereme optimalizační úlohy, které byly vybrány pro demonstraci přístupů zmíněných v předchozí kapitole. Nejprve se jedná o dvě diskrétní optimalizační úlohy, konkrétně problém více batohů a problém pokrytí množiny. Poté si uvedeme dvě spojité úlohy a to hledání globálního extrému u Ackleyho a Rastriginovy funkce.

4.1 Problém více batohů

Problém více batohů, anglicky multiple knapsack problem (MKP), je variací na známý knapsack problem, česky označovaný jako problém batohu nebo problém zlodějova rance, ovšem s více batohy. Jedná se tudíž o diskrétní úlohu. U MKP jsou vstupními informacemi počet zdrojů (batohů) m, počet objektů n, kde každý objekt j představuje zisk p_j. Každý zdroj má svůj rozpočet (objem batohu) c_i, přičemž je dána spotřeba r_j zdroje objektem j (objem objektu j). Úkolem je maximalizovat celkový zisk s omezeným celkovým rozpočtem.

MKP může být formulováno následovně (viz [28]):

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$$

s ohledem na:

$$\sum_{j=1}^{n} r_j x_{ij} \leq c_i, \qquad i = 1, \dots, m$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \qquad i = 1, \dots, m$$

$$x_{ij} \in \{0,1\}, \qquad i = 1, \dots, m,\ j = 1, \dots, n$$

Proměnná x_{ij} značí, zda je objekt j v batohu i obsažen.

Jestliže budeme dále definovat $I = \{1, \dots, m\}$, $J = \{1, \dots, n\}$, s tím že $c_i \geq 0$ pro všechna $i \in I$, pak dobře formulovaný MKP předpokládá $p_j > 0$ a $r_{ij} \leq c_i \leq \sum_{j=1}^{n} r_{ij}$ pro všechna $i \in I$ a $j \in J$.

V praxi může MKP znamenat například problém spediční společnosti, která má za úkol přepravit jisté množství různorodých balíků několika dopravními prostředky s omezenou kapacitou. Dalšími oblastmi aplikace MKP jsou finanční management, kontrola rozpočtu, nebo minimalizace odpadu při řezání (rozdělování) zboží. MKP může být také využito v kryptografii, kde se bezpečnost realizuje obtížností řešení MKP.

4.2 Problém pokrytí množiny

Problém pokrytí množiny, anglicky set cover problem (SCP), je klasickou diskrétní optimalizační úlohou, přičemž je dána konečná množina položek ve formě 0-1 $m \times n$ matice $A = (a_{ij})$ a n-dimenzionální celočíselný vektor $c = (c_j)$. Dále může být zavedeno značení $M = \{1, \dots, m\}$ a $N = \{1, \dots, n\}$. Hodnota $c_j (j \in N)$ reprezentuje cenu sloupce j, přičemž bez ztráty obecnosti lze uvažovat $c_j > 0$ pro $j \in N$. Řekneme, že $j \in N$ pokrývá $i \in M$, pokud $a_{ij} = 1$. Úkolem je nalezení podmnožiny $S \subseteq N$ takové, že každé $i \in M$ je pokryto přinejmenším jedním $j \in S$. Dále budeme definovat matematický model pro SCP (viz [29]):

$$v(SCP) = \min \sum_{j \in N} c_j x_j$$

vzhledem k:

$$\sum_{j \in N} a_{ij} x_j \geq 1, \qquad i \in M$$

$$x_j \in \{0,1\}, \qquad j \in N$$

Proměnná $x_j = 1$, pokud $j \in S$, jinak platí $x_j = 0$. Často se zavádí značení $J_i = \{j \in N \,|a_{ij} = 1\}$ množiny j pokrývajících i a analogicky $I_j = \{i \in M \,|a_{ij} = 1\}$ množiny i pokrytých j.

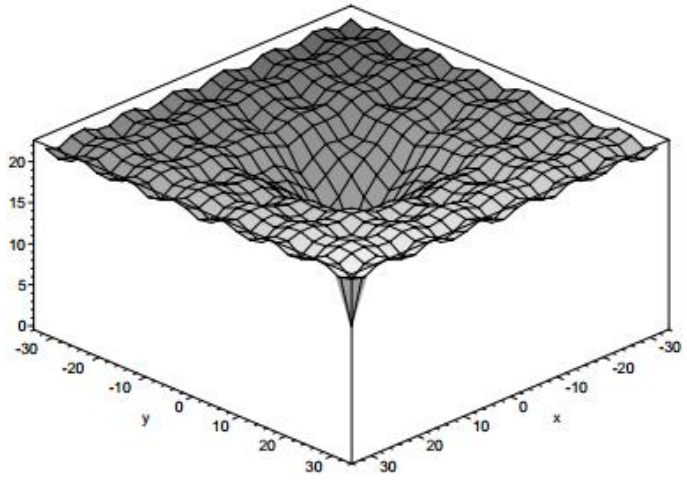
V reálu se SCP používá při plánování letového řádu, kdy je dána množina letů, která musí být pokryta při minimálních nákladech na posádku. Používá se párování, což je sekvence letů, které mohou být pokryty jednou posádkou. Samozřejmě lze tento přístup použít i jinde, např. v kamionové dopravě apod.

4.3 Nalezení globálního minima u Ackleyho funkce

Ackleyho funkce (viz [30]) je spojitou funkcí, která se často využívá jako multimodální testovací funkce, definovaná jako:

$$f(x) = -a \cdot \exp\left(-b \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(cx_i)\right) + a + \exp(1)$$

přičemž je doporučeno nastavit $a = 20$, $b = 0.2$, $c = 2\pi$ a testovací oblast omezit na hyperkostku $-32.768 \leq x_i \leq 32.768$, $i = 1, \ldots, n$, kde proměnná n označuje počet dimenzí. Globální minimum $f(x) = 0$ je dosažitelné pro $x_i = 0$, $i = 1, \ldots, n$.



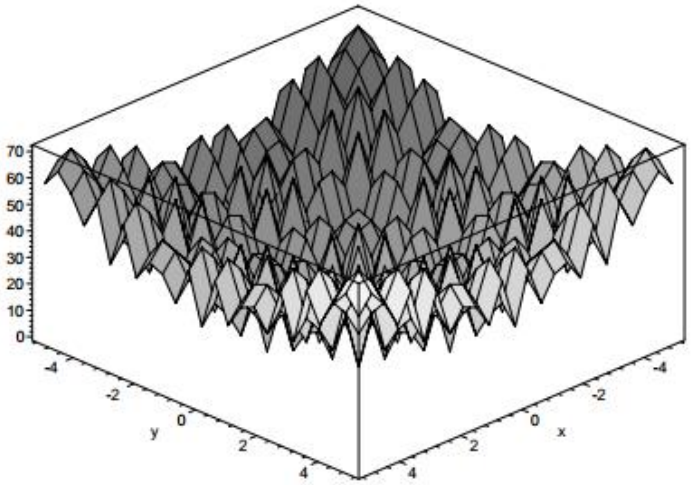
Obrázek 15: Ackleyho funkce ve 2D. [30]

4.4 Nalezení globálního minima u Rastriginovy funkce

Rastriginova funkce (viz [30]) je také n-dimenzionální spojitou funkcí. Vybral jsem ji záměrně kvůli její velké produkci lokálních minim, která je způsobena modifikací De Jongovy funkce pomocí kosinu. Je ovšem nutné počítat s tím, že minima nejsou rozložena náhodně. Definice funkce:

$$f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i)\right]$$

Obvykle se testovací oblast funkce omezuje na hyperkostku $-5.12 \leq x_i \leq 5.12,\ i = 1, \dots, n$, přičemž globální minimum $f(x) = 0$ lze nalézt při $x_i = 0,\ i = 1, \dots, n$.



Obrázek 16:Rastriginova funkce ve 2D. [30]

5 Implementace

V této kapitole se seznámíme s některými detaily implementační části této práce. Nejprve uveďme, že v souladu se zadáním je projekt implementován v jazyce Java verze 7, přičemž jej lze rozdělit na dvě pomyslné části – část, která realizuje samotný proces výpočtu jednotlivých úloh, a část grafickou, která umožňuje snadný uživatelský přístup s možností nastavení parametrů jednotlivých metod při jejich aplikaci na zvolené úlohy.

Podotkněme, že byl zvolen přístup abstraktní implementace algoritmů zmíněných v předchozích kapitolách, které jsou pak upřesněny pomocí mechanizmů polymorfismu pro účely konkrétních úloh. Ve spojení s generickými datovými typy, lze implementaci metod užít v libovolném projektu, kdy stačí definovat abstraktní třídy. Projekt může být tedy také považován za jakousi malou knihovnu pro zmíněné optimalizační metody.

V následujících kapitolách ukážeme, které komponenty jsou nezbytné pro implementaci uvedených optimalizačních metod, případně jaké byly zvoleny přístupy při jejich realizaci pro dané úlohy.

5.1 Optimalizace mravenčí kolonií

Již z názvu je patrné, že kromě vlastního algoritmu je nutné mít k dispozici nějakou realizaci mravence (agenta). Jelikož byl pro implementaci zvolen objektově orientovaný přístup, tak se přímo nabízí, že mravenec může být realizovaný jako samostatný objekt. Hlavní činností mravence je konstrukce řešení, proto jeho nejdůležitější částí je výběr následujícího kroku směřujícího ke kompletnímu řešení. Za tímto účelem každý objekt reprezentující mravence obsahuje uspořádaný seznam či vektor míst (či spojnic míst), která navštívil. Tento seznam, řekněme cesta, je tedy částečným řešením daného problému. Pokud mravenec již nemá možnost dalšího kroku, je zkonstruováno kompletní řešení.

V ACS pravidlu pro určení následujícího kroku se též vyskytuje výraz $(ix) \notin tabu_k$, který označuje krok z aktuálního místa i do místa x, jenž není v množině $tabu_k$ mravence k. Množina $tabu_k$ obsahuje spojnice míst (popřípadě jen místa, záleží na implementaci), které mravenec již použil. Z výrazu je tedy patrné, že hledáme takové spojnice, které nebyly ještě tímto mravencem navštíveny. Realizaci množiny $tabu_k$ jsme si představili již v předchozím odstavci (viz cesta). Ačkoli z matematického pohledu zde nenastává problém, z implementačního pohledu by bylo krajně nevhodné mít pouze seznam navštívených spojnic (či případně míst) a pokaždé hledat místa a spojnice k nim, které nebyly při konstrukci cesty ještě použity. To by zavádělo do činnosti algoritmu nezanedbatelnou časovou redundanci. Proto byla při implementaci použita komplementární množina, nazvěme ji $nottabu_k$, která udržuje ještě nepoužité spojnice. Společně tedy $tabu_k$ a $nottabu_k$ tvoří množinu všech možných spojnic, případně míst, na pomyslné virtuální mapě, po které se mravenec pohybuje.

Nakonec je nutné zmínit komponentu, která realizuje feromonový model, jenž ke každé spojnici dvou míst přiřazuje hodnotu označující jeho feromonovou hladinu. Uveďme, že by teoreticky bylo možné přiřazovat feromonovou hladinu pouze místům, čímž bychom snížili paměťovou náročnost algoritmu. Avšak ztrácíme informaci o propojení mezi místy, proto jsem tuto možnost nezvolil. Držel jsem se klasické reprezentace, kdy hodnoty feromonových hladin spojnic míst udržujeme ve čtvercové matici (symetrické dle hlavní osy), přičemž obě hrany této matice tvoří množina všech míst.

Výhodou při implementaci je fakt, že mravenci spolu komunikují pouze prostřednictvím prostředí a ne přímo spolu. Není nutné implementovat komunikaci či topologii jejich propojení jako je tomu například u PSO.

5.1.1 Aplikace optimalizace mravenčí kolonií na problém více batohů

Zásadní otázkou je to, jakým způsobem převedeme batohy a jim přiřazované objekty ze zadané úlohy na místa ACS algoritmu, po kterých se mravenec pohybuje. Teoreticky můžeme uvažovat v zásadě o třech možnostech. Buďto můžeme za ACS místa považovat batohy nebo naopak můžeme jako místa užít objekty, či můžeme mít množinu všech přípustných kombinací batohů a objektů coby množinu všech míst. Varianta s batohy nebyla užita z povahy úlohy. Většinou je totiž množina objektů mnohem větší než množina batohů. Například klasická úloha batohu má pouze jeden batoh. Potom bychom měli pouze jedno místo pro ACS algoritmus, čímž bychom naprosto narušili jeho funkčnost. Z tohoto pohledu by se mohla zdát nejlepší varianta s kombinacemi všech batohů a objektů. Vezměme si příklad, že bychom měli 100 batohů a 10000 objektů, z nichž je možné průměrně jen 30 přiřadit každému batohu. Pak pro výběr 30 objektů pro jediný batoh dostáváme cca $3.5 \cdot 10^{87}$ možností. Takto obrovský stavový prostor je pak velmi komplikované procházet. To ovšem není jediný problém. Jakmile se zvětší počet ACS míst, vzrostou nároky na paměť, jelikož s rostoucím počtem míst roste exponenciálně velikost matice, která reprezentuje spojnice mezi jednotlivými místy. Pro zadanou úlohu se potřebná paměť již pohybuje v řádu terabytů.

Z těchto důvodů byla zvolena možnost za místa považovat pouze objekty. Mravenec pak pomyslně nese všechny batohy a vždy, když má vykonat jeden krok, vybere jeden objekt, který přiřadí do některého z batohů, který je určen heuristikou.

Při implementaci jsem uvažoval čtyři typy heuristik. První statická heuristika uvažuje pouze hodnotu a objem objektu:

$$\eta = \frac{{p_i}^{d_1}}{{r_i}^{d_2}},$$

kde p_i představuje hodnotu objektu i a r_i jeho objem. Parametry d_1 a d_2 slouží k případnému doladění vlivu objemu a hodnoty, avšak většinou bývají obecně nastaveny na 1. Druhá heuristika, taktéž statická, uvažuje kromě hodnoty a objemu objektu také objem cílového batohu:

$$\eta = \frac{{p_i}^{d_1}}{\left(\frac{r_i}{c_k}\right)^{d_2}},$$

kde c_k je celkový objem cílového batohu k. Další dvě heuristiky jsou dynamické, jelikož jejich hodnota závisí na aktuálním přiřazení objektů k batohům. Výpočetně jsou obě heuristiky totožné, liší se pouze nastavením parametrů d_1 a d_2 v tom smyslu, že první dynamická heuristika je uvažuje shodné a druhá rozdílné. Uveďme tedy rovnici:

$$\eta = \frac{{p_i}^{d_1}}{\left(\frac{r_i}{\Delta c_k}\right)^{d_2}},$$

přičemž Δc_k představuje aktuálně volný objem v zadaném cílovém batohu.

5.1.2 Aplikace optimalizace mravenčí kolonií na problém pokrytí množiny

Podobně jako v předchozí kapitole řešíme problém, jakým způsobem reprezentovat místa v ACS algoritmu. Zde je situace o poznání jednodušší. V zásadě se nabízí pouze jediná možnost a to, že každé místo feromonového modelu reprezentuje jednu z kandidátních množin, z nichž každá pokrývá určitou část univerza. Mravenec tedy vybírá místa (kandidátní množiny) tak dlouho, dokud nepokryje celé univerzum.

Pro výběr další množiny jsem implementoval tři heuristiky. První heuristika, kterou lze též označit za tzv. greedy, preferuje ty kandidátní množiny, které pokrývají největší část nepokrytého univerza bez ohledu na to, jakou má daná množina cenu. Můžeme uvést hodnotu vrácenou heuristikou jako:

$$\eta = |\{x | x \in S \wedge x \; není \; ještě \; pokryto\}| \; ,$$

kde S je kandidátní množina. Druhou extrémní variantou je heuristika, která vybírá množiny pouze dle ceny:

$$\eta = \frac{1}{c_S},$$

kde c_S je cena za množinu S. Poslední heuristika je kompromisem mezi oběma zmíněnými variantami, která využívá poměru přínosu nových prvků k ceně za množinu:

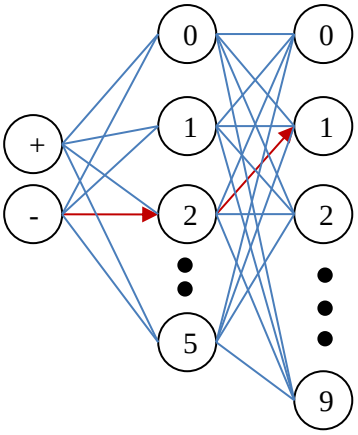
$$\eta = \frac{|\{x | x \in S \wedge x \; není \; ještě \; pokryto\}|}{c_S},$$

přičemž proměnné mají stejný význam jako v předešlých rovnicích.

5.1.3 Aplikace optimalizace mravenčí kolonií na hledání globálních minim Ackleyho a Rastriginovy funkce

Jelikož algoritmus ACS je navržen pro diskrétní problémy, nastává komplikace s realizací feromonového modelu. Jelikož si potřebujeme pamatovat hladiny feromonů na spojnicích mezi místy a těchto míst je nekonečně mnoho pro reálné hodnoty, musíme přistoupit k diskretizaci. Algoritmus je ovšem implementován tak, že je možné nastavit přesnost na libovolný počet desetinných míst. Vzhledem k zadaným úlohám, ale často i v mnoha jiných spojitých úlohách, nám stačí přesnost třeba i jen na jedno desetinné místo. Na druhou stranu, pokud zvýšíme dimenzi Ackleyho či Rastriginovy funkce, tak se úloha stává dostatečně složitou i pro přesnost jen na pár desetinných míst.

U těchto spojitých úloh jsem zvolil trochu jiný přístup k tomu, co reprezentují místa v ACS algoritmu a jak jsou propojena. Jsou totiž hierarchicky uspořádána vzhledem k jejich výběru mravencem. Například u Rastriginovy funkce si mravenec nejprve vybere místo reprezentující jednu z hodnot 0-5 pro jednotky, dále 0-9 pro desetiny, 0-9 pro setiny atd. Ovšem úplně první krok (ještě před čísly) rozhodne o tom, zda výsledné číslo bude kladné či záporné. Takto mravenec postupně zkonstruuje řešení s přesností na libovolný počet desetinných míst (viz následující obrázek).



Obrázek 17: Ilustrace konstrukce řešení s hodnotou -2.1 pro 1D Rastriginovu funkci.

Ovšem tuto úvahu ještě musíme zobecnit na libovolný počet dimenzí. Tedy například v jednom kroku mravenec vybere namísto hodnoty 0-9 vektor hodnot 0-9.

Heuristika u těchto spojitých úloh byla zvolena jako:

$$\eta = \frac{1}{v},$$

kde v představuje hodnotu dané funkce pro navrhované řešení.

5.2 Optimalizace hejnem částic

Tento algoritmus pracuje s množinou (hejnem) částic, proto musíme mít samozřejmě reprezentaci takové částice. Znovu využijeme objektové orientovanosti jazyka a každou částici implementujeme jako samostatný objekt. Částice si udržuje informace o své aktuální poloze v n-dimenzionálním prostoru. Dále zná velikost a směr rychlosti svého pohybu. A konečně si pamatuje také svou lokální nejlepší polohu. Proto při každé iteraci algoritmu každá částice přepočítá svou polohu a rychlost na základě těchto údajů a údaji o pozici s globálně nejlepším řešením. V teoretické části této práce jsme uvedli různé možnosti výpočtu nového vektoru rychlosti, které byly implementovány, tudíž jedním z parametrů algoritmu je výběr typu rovnice pro přepočet vektoru rychlosti.

Na rozdíl například od ACS, kde jednotliví agenti komunikují pouze nepřímo, zde částice komunikují přímo v rámci svého okolí, jež je dáno jistou topologií. Jak jsme si uvedli, tak pro výpočet potřebujeme znát globální maximum. Ovšem uveďme, že toto globální maximum se vztahuje pouze na sousedství dané topologií. Proto jsem musel zvolit dostatečně obecnou implementaci tak, aby byla schopná vyjádřit minimálně topologie uvedené v teoretické části tj. topologie hvězda, kolo a kruh. Nakonec byl zvolen přístup, kdy každá částice obsahuje množinu indexů částic, se kterými sousedí a se kterými sdílí informace o globálním optimu. Před samotným spuštěním algoritmu inicializujeme každou částici množinou indexů sousedů dle požadované topologie. Tato implementace je natolik obecná, že by teoreticky bylo možné mít i oddělené skupiny částic s naprosto libovolným vnitřním propojením (kromě smyčky sama na sebe).

Další komplikací, nutnou vyřešit, byla situace, kdy se částice při aktualizaci pozice dostane mimo vytyčený n-dimenzionální prostor v některé z dimenzí. Bylo by sice možné částici pevně usadit na hranici, kterou v dané dimenzi překročila, a nechat ji statickou (bez pohybu v dané dimenzi), ale tím bychom se zbytečně ochudili o možnost dalšího prohledávání příslušné části stavového prostoru. Proto částici sice umístíme na hranici vytyčeného prostoru pro danou dimenzi, ale udělíme jí rychlost s opačným směrem, než se kterým hranici překročila.

V dalších kapitolách si ukážeme reprezentaci řešení testovacích úloh v PSO a také určíme, jak velký volit prostor pro částice.

5.2.1 Aplikace optimalizace hejnem částic na problém více batohů

Hlavním úkolem při řešení problému více batohů pomocí hejna částic je reprezentace řešení problému jako bodu n-dimenzionálního prostoru, kde se částice pohybují. Pokud se zamyslíme nad omezeními, které problém více batohů přináší, dojdeme k závěru, že každý objekt může být umístěn nanejvýše v jednom batohu. Naopak každý batoh může obsahovat teoreticky až všechny objekty. Pokud by dimenze reprezentovaly batohy, pak by každý bod (interval) v dané dimenzi musel reprezentovat kombinaci objektů, které daný batoh obsahuje. Tento přístup přináší dvě komplikace. Nejprve existuje obrovské množství kombinací všech možných objektů, které lze do batohu umístit. Ty je pak obtížné spravovat. Druhou komplikací je také to, že bychom museli komplikovaně hlídat, zda kombinace objektů v jedné dimenzi nekoliduje s kombinací objektů v jiné dimenzi.

Z uvedených důvodů byl zvolen přístup, kdy pro každý objekt je vyhrazena jedna dimenze. Daná dimenze je rozdělena na intervaly shodné velikosti, přičemž každý interval reprezentuje jeden batoh. Samozřejmě je nutné počítat s jedním extra intervalem pro možnost, že objekt není umístěn v žádném batohu. Takto je zaručena podmínka, že objekt bude umístěn maximálně v jednom batohu. Při rekonstrukci řešení se pak pouze hlídá to, aby nebyl překročen rozpočet jednotlivých batohů.

Další otázkou je, jak velký prostor volit pro jednotlivé dimenze. Zvolil jsem přístup, kdy daný prostor dimenze je dynamicky dělen na potřebný počet intervalů, tudíž je naprosto irelevantní jak velký (samozřejmě nenulové velikosti) jej volíme. Můžeme si zvolit například hyperkostku o velikosti hrany 1 či třeba 100.

5.2.2 Aplikace optimalizace hejnem částic na problém pokrytí množiny

Stejně jako v předchozí kapitole nás především zajímá způsob, jakým bude n-dimenzionální prostor reprezentovat řešení problému pokrytí množiny. Úloha je dána množinou prvků univerza a množinou pokrývajících množin. Ale pouze přiřazení dimenze každému prvku univerza dává použitelný význam.

Každý prvek univerza má tedy svou dimenzi, kterou znovu dělíme na intervaly, přičemž každý interval odpovídá množině, která pokrývá příslušný prvek univerza. Pro každý prvek univerza platí, že je pokryt nějakou množinou, tedy každá pozice částice v zadaném prostoru reprezentuje jedno korektní řešení.

Znovu jsme zvolili způsob dělení dimenze na intervaly podobně jako u předešlé úlohy. Řešení je tedy nezávislé na velikosti prostoru jednotlivých dimenzí. Například jsem volil prostor jako hyperkostku o velikosti hrany 100.

5.2.3 Aplikace optimalizace hejnem částic na hledání globálních minim Ackleyho a Rastriginovy funkce

U n-dimenzionálních úloh, jakou jsou hledání globálního minima Ackleyho a Rastriginovy funkce, je reprezentace řešení pomocí n-dimenzionálního prostoru, ve kterém se pohybují částice, naprosto

zřejmá. Pokud máme například zadánu Ackleyho funkci se třemi dimenzemi, volíme také prostor se třemi dimenzemi i pro částice. Tedy prostor zadání i prostor pro částice jsou navzájem ekvivalentní.

Jediným omezením, které klademe na prostor částic, je to, že by jeho rozměry měly odpovídat rozměrům, jež jsou dány zadáním. Například pro Rastriginovu funkci uvažujeme prostor $\langle -5.12; 5.12 \rangle$ pro každou dimenzi.

5.3 Genetické algoritmy

Genetické algoritmy pracují s populací jedinců, kdy každý jedinec reprezentuje jedno kompletní řešení zadané úlohy. Populaci lze tedy snadno implementovat jako vektor jedinců. Pro výběr rodičů a obnovu populace stačí znát ohodnocení jedince, proto každý jedinec musí být schopen poskytnout ohodnocení (fitness) odpovídající jeho řešení úlohy. Poznamenejme, že pro výběr rodičů byly implementovány metody turnaj, statistické náhodné vzorkování, elita a ruletový výběr. Pro obnovu populace jsem implementoval mechanismy čisté, elitistické a uniformní náhrady a náhrady na základě fitness hodnoty.

Jak jsme si uvedli, jedinec udržuje řešení zakódované ve formě chromozomu, proto v následujících kapitolách ukážeme, jaká reprezentace byla zvolena pro jednotlivé úlohy. Formát chromozomu je také podstatný pro křížení dvou jedinců a jejich mutaci, tudíž jsem také při implementaci zvolil přístup, kdy implementace genetického algoritmu obsahuje funkce pro křížení a mutaci pouze v abstraktní formě a jejich konkrétní implementace se přenáší na implementaci konkrétní úlohy.

Dodejme, že pro mutaci můžeme zvolit dvě možnosti, co bude vyjadřovat pravděpodobnost mutace. Nejprve to může být pravděpodobnost mutace genu, tedy dále uvedené mutace jsou s danou pravděpodobností aplikovány na každý jeden gen chromozomu jedince. Často se však používá i pravděpodobnost mutace jedince, kdy se s danou pravděpodobností zmutuje jeden náhodně vybraný gen chromozomu. Realizovány jsou obě možnosti, jejichž optimální nastavení si ukážeme v kapitole 6.

Pro ukončení algoritmu jsou implementovány dvě možnosti. První možnost je pevně zadaný počet iterací, po jejichž uplynutí algoritmus končí. Druhou možností je zadaný počet iterací bez zlepšení nejlepšího řešení. To znamená, že pokud se nenalezne lepší řešení v dané iteraci, je inkrementován čítač. Jakmile čítač dosáhne zadaného počtu, algoritmus končí. Pokud ovšem v dané iteraci dojde ke zlepšení nejlepšího řešení, čítač je vynulován.

5.3.1 Aplikace genetického algoritmu na problém více batohů

Hlavním implementačním detailem při řešení problému více batohů je formát chromozomu. V tomto případě je namísto standardního binárního chromozomu použit chromozom, kdy je každý jeden gen reprezentován celým číslem. Při volbě reprezentace řešení jsem uvažoval dvě možnosti. První možností je to, že každý gen odpovídá obsahu jednoho batohu, tedy gen reprezentuje množinu objektů. Nevýhodou je, že počet možností alternativ pro jeden gen je obrovský. Také je třeba kontrolovat, zda jednotlivé geny (množiny objektů) vzájemně nekolidují. Druhou možností je to, že každý gen reprezentuje jeden objekt a jeho přiřazení k batohu. Vidíme, že tak snadno dosáhneme požadované vlastnosti, že žádný objekt není obsažen současně ve více batozích. Avšak vždy pro křížení a mutaci musíme zkontrolovat, zda nové řešení někde nepřekračuje rozpočet batohu a případně tuto situaci napravit. Je zřejmé, že každý gen musí mít možnost být nastaven na hodnotu, která reprezentuje to, že není přiřazen žádnému batohu.

Operace křížení je implementována stejně, jako jsme definovali operaci křížení v teoretické části. Jsou implementovány obě možnosti, tj. výměna částí řetězců daná body křížení a uniformní výběr genů z obou rodičů.

Operace mutace mění chromozom prostřednictvím změny hodnoty jednoho genu na jednu z přípustných hodnot. To znamená, že změníme umístění objektu z jednoho batohu do jiného. Případně změna hodnoty genu způsobí, že nezařazený objekt se umístí do jednoho z batohů nebo již nebude přiřazen žádnému batohu.

5.3.2 Aplikace genetického algoritmu na problém pokrytí množiny

Při aplikaci genetického algoritmu na problém pokrytí množiny byl použit formát lineárního chromozomu s celými čísly. Zde je vyčleněn jeden gen pro jeden prvek univerza. Gen pak obsahuje označení jedné z množin, která tento prvek univerza pokrývá. Je zaručeno (pokud to splňuje zadání úlohy), že každý prvek univerza je pokryt alespoň jednou množinou. Existuje i jiná možnost, kdy bychom měli binární reprezentaci chromozomu, přičemž každý gen by představoval jednu pokrývající množinu. Hodnota genu by pak určovala, zda tuto množinu použijeme. Tato varianta ovšem nese problém, že ne všechny binární kombinace představují korektní řešení. Některý prvek univerza by totiž nemusel být pokryt, proto by bylo nutné každý nově vytvořený chromozom zkontrolovat a případně upravit do korektního stavu či penalizovat jeho fitness hodnotu.

Operace křížení je implementována jako standardní operace křížení daná body křížení nebo jako uniformní křížení. Na křížení uvedeného formátu chromozomu nejsou kladeny žádné zvláštní požadavky, jelikož hodnoty genů nejsou vzájemně nijak závislé. Po provedení křížení dostaneme potomky, kteří představují opět korektní řešení úlohy.

Operace mutace nám změní hodnotu genu na jinou přípustnou, která označuje jinou množinu, která příslušný prvek pokrývá. Čili operace mutace nenaruší korektnost řešení.

5.3.3 Aplikace genetického algoritmu na hledání globálních minim Ackleyho a Rastriginovy funkce

Při řešení zadaných spojitých úloh jsem se snažil držet co nejvíce klasického pojetí genetických algoritmů, které v zásadě používají binárního či celočíselného zakódování. Proto jsem zvolil, podobně jako u ACS, diskretizaci hodnot dané funkce. Každý chromozom je složen z n-tic genů, přičemž každá n-tice je složena z jednoho genu pro znaménko, několika genů pro celočíselnou část a několika genů pro desetinnou část (tolika, kolik určuje požadovaná přesnost). Obecně geny pro celočíselnou a desetinnou část obsahují hodnoty 0-9. Počet n-tic v chromozomu je dán počtem dimenzí řešeného problému. Lze nahlédnout, že i při diskretizaci budou zadané úlohy při dostatečně vysoké dimenzi i při malé požadované přesnosti natolik složité, že dostatečně prověří schopnosti genetických algoritmů. Připomeňme, že často je i menší přesnost v praxi dostatečná. Teoreticky by bylo možné se mírně odchýlit od klasických genetických algoritmů a použít reálné zakódování a speciální operátory křížení a mutace. Ovšem pokud bychom uvažovali o použití evolučních algoritmů, do kterých genetické algoritmy spadají, a reálného zakódování, pak by mnohem větší smysl namísto genetických algoritmů dávaly například evoluční strategie, které přímo s reálným zakódováním počítají.

Jelikož je dán formát s pevnou délkou chromozomu, lze užít běžné operace při křížení dvou chromozomů. Dva potomci vytvoření křížením dvou korektních rodičů reprezentují opět korektní řešení.

Při operaci mutace, která mění jeden náhodně vybraný gen (či každý z genů s určitou pravděpodobností) na jinou hodnotu z intervalu celých čísel 0 až 9, je třeba dbát pouze na to, aby nová hodnota genu reprezentovala korektní řešení. Například pokud chceme, aby v dané dimenzi ležela vstupní hodnota Rastriginovy funkce v intervalu $\langle -5.12; 5.12 \rangle$, pak by gen pro jednotky měl ležet v intervalu $\langle 0; 5 \rangle$.

5.4 Simulované žíhání

Při bližším pohledu na algoritmus simulovaného žíhání zjistíme, že je poměrně velmi nezávislý na řešené úloze. Jedním implementačním detailem, který musí každý jeden objekt reprezentující kandidátní řešení splňovat, je poskytnutí ohodnocení (fitness funkce) pro dané řešení.

Druhou věcí, která je závislá na konkrétní úloze, je generování dalšího kandidátního řešení. Algoritmus totiž udržuje jedno aktuální řešení, na jehož základě pak generuje jedno kandidátní řešení. Toto kandidátní řešení pak může být přijato za aktuální.

Poznamenejme, že algoritmus byl obohacen o určitý práh, který určuje nulovou teplotu. Vždy, když teplota klesne pod tento práh, považujeme již teplotu za nulovou. Tato změna je dána čistě z praktického důvodu, jelikož teplotu upravujeme tak, že její hodnotu násobíme konstantou. Tím pádem čistě z matematického pohledu dosáhneme nuly po nekonečně mnoha úpravách teploty. Ale i když víme, že v počítači máme hodnoty diskretizovány, přesto může trvat nezanedbatelnou dobu, než je nulové teploty dosaženo.

5.4.1 Aplikace simulovaného žíhání na problém více batohů

Hlavním rysem, který je typický pro aplikaci pro danou úlohu, je funkce pro generování dalšího kandidátního řešení z toho aktuálního. První aktuální řešení je vygenerováno tak, že postupně do každého batohu jsou přidávány objekty, dokud je to jen možné.

Při generování nového kandidátního řešení vycházíme z množiny objektů, které nebyly v aktuálním řešení použity. Vždy se pokusíme postupně vkládat jeden objekt z této množiny do každého z batohů. Pokud má objekt menší objem, než je objem batohu, mohou nastat dvě možnosti. Nejprve může být v batohu ještě dostatek volného místa a objekt můžeme bez potíží vložit. Nebo v batohu není dostatek místa, tudíž z batohu odstraňujeme objekty tak dlouho, dokud není možné tam nový objekt vložit. Vždy, když se nám podaří vytvořit nové řešení pomocí vložení některého z nepoužitých objektů do některého z batohů, tak si toto řešení zaznamenáme. Jakmile vyzkoušíme všechny možnosti, ze zaznamenaných řešení vybereme to, které představuje nejlepší hodnotu. Případně můžeme vybrat jedno náhodné řešení.

5.4.2 Aplikace simulovaného žíhání na problém pokrytí množiny

Počáteční řešení je tvořeno náhodně tak, že náhodně vybíráme pokrývající množiny tak dlouho, dokud nejsou pokryty všechny prvky univerza.

Pro výběr dalšího kandidátního řešení jsou vytvořena nová řešení v rámci okolí současného řešení dvěma způsoby. Nejprve se snažíme odebrat vždy jednu pokrývající množinu ze současného řešení. Jakmile ji odebereme, vždy zkontrolujeme, zda se jedná o korektní kompletní řešení. Kompletnost je kontrolována z toho důvodu, abychom nemuseli zavádět penalizační funkci či jinou opravu a současně abychom zmenšili prostor uvažovaných řešení. Pokud není přijato ani jedno (či neexistuje) řešení s menším počtem množin, pak vytváříme nová řešení přidáváním množin. Ačkoliv

by se mohlo zdát, že je to zavádějící cesta, tak při přijetí takového řešení pak v dalším kole algoritmu může nastat situace, že lze odebrat prvek, který odebrat nešel.

5.4.3 Aplikace simulovaného žíhání na hledání globálních minim Ackleyho a Rastriginovy funkce

Jelikož při hledání extrémů daných funkcí se jedná o spojitou úlohu, musel být zvolen buďto diskrétní či spojitý přístup. Jelikož je algoritmus simulovaného žíhání velmi nezávislý na formátu řešení, není nutné hodnoty diskretizovat, a proto jsem zvolil přístup s reálnými hodnotami.

Počáteční řešení je vytvořeno jako náhodný bod v zadaném n-rozměrném prostoru daném úlohou.

Kandidátní řešení tvoříme tak, že vygenerujeme předem daný počet náhodných bodů (řešení) v určitém zadaném okolí současného řešení. Z těchto pak vybereme ten nejlepší bod, který je algoritmem následně posouzen k přijetí za nový aktuální stav.

Při tomto postupu zavádíme nové důležité parametry. Nejprve se jedná o počet bodů v okolí toho aktuálního, ze kterých vybíráme kandidátní řešení. Prozraďme, že ačkoli vyšší počet bodů lépe pokryje zkoumanou oblast, nemusí vést k lepším výsledkům. Druhým parametrem je velikost oblasti dané hranou hyperkostky (prostoru řešení), ve které generujeme řešení. Lze snadno nahlédnout, že čím větší oblast bude, tím více se algoritmus podobá algoritmu náhodného výběru.

5.5 Posilované učení

Algoritmus posilovaného učení je poměrně nezávislý na řešené úloze, proto nám pro každou úlohu stačí reprezentovat množinu stavů (objektů, které posilované učení hodnotí), které jsou standardně reprezentovány jako java objekty. Každý objekt reprezentující stav pak udržuje počitadlo použití daného stavu, odměnu za dosažení tohoto stavu a hodnotu, kterou mu přiděluje algoritmus posilovaného učení.

Uveďme, že byly implementovány obě varianty posilovaného učení, TD-learning i Q-learning. Navíc byly implementovány pro každou úlohu i variantu dva přístupy, co každý stav reprezentuje (viz následující kapitoly). Pracovně jsem tyto přístupy nazval relativní a absolutní.

5.5.1 Aplikace posilovaného učení na problém více batohů

Hlavní implementační otázkou při implementaci je to, co budou stavy reprezentovat. Do úvahy připadají dvě možnosti pro TD-learning i Q-learning, jež byly obojí implementovány. Nejprve se zaměřme na TD-learning. První (absolutní) variantou je, že každý stav může reprezentovat částečné řešení. To znamená, že máme stavy, kde je umístěn jeden objekt v jednom batohu, pak existují stavy reprezentující umístění více položek v různých batozích (v pořadí jak byly přidávány) a konečně jsou zde stavy, které reprezentují konečná řešení. Po skončení běhu algoritmu, pak zkonstruujeme řešení dle vypočtených hodnot. Lze to přirovnat k aplikaci posilovaného učení na hru šachy, kdy každý stav reprezentuje určité rozmístění figurek na šachovnici. Poznamenejme, že tato varianta může znamenat velkou expanzi stavového prostoru, což zapříčiní velké časové nároky algoritmu.

Druhou možností je to, že stav reprezentuje vždy umístění jednoho objektu v určitém batohu (relativní přístup). Znamená to, že budeme mít značně redukovaný stavový prostor oproti předchozí variantě (počet hodnocených stavů je roven součinu počtu objektů a batohů, samozřejmě pokud lze přiřadit objekt každému batohu). Tento přístup reprezentuje myšlenku ohodnocení toho, jaký přínos má přidělení určitého objektu do nějakého batohu. Je to jistý způsob, jak hodnotit akce přidání

objektu do batohu (jako v Q-learningu), ale při zachování TD-learning způsobu výpočtu. Po skončení samotného algoritmu je výsledné řešení zkonstruováno tak, že tyto stavy jsou nejprve uspořádány od nejvyšší hodnoty dané posilovaným učením, a pak se postupně snažíme vkládat objekty do batohů tak, jak nám to udává pořadí stavů. Jedná se o velmi netypickou alternativu, přesto byla implementována a vyzkoušena.

Obdobně můžeme uvažovat i pro Q-learning. Na rozdíl od TD-learning varianty, stavy musíme obohatit o akce, které jsou při daném částečném řešení vykonány. Q-learning totiž nehodnotí částečná řešení, ale akce pro uvedená částečná řešení. První možností tedy je, že stav nám reprezentuje situaci, kdy máme více objektů umístěných v určitých batozích a chceme jeden konkrétní nový objekt umístit do jednoho z batohů. Uveďme, že se nám tak stavový prostor oproti odpovídající variantě pro TD-learning dokonce ještě zvětší.

Druhou možností je hodnotit akce (přidání jedné položky do určitého batohu) stejně jako u relativní TD-learning varianty, tedy bez znalosti historie použitých akcí.

5.5.2 Aplikace posilovaného učení na problém pokrytí množiny

Podobně jako při aplikaci posilovaného učení na problém více batohu, tak i zde byly implementovány čtyři možnosti toho, co může reprezentovat stav, který uvažuje algoritmus posilovaného učení.

Pro TD-learning máme nejprve implementovánu absolutní variantu, kde každý stav reprezentuje částečné řešení problému pokrytí množiny. Obecně tak stav obsahuje množinu prvků, které částečně pokrývají prvky univerza. Samozřejmě jsou pak zastoupeny i stavy, které reprezentují kompletní řešení a právě ony pak rozhodují o výši udělované odměny. Druhou variantou je to, že jeden stav odpovídá jedné pokrývající množině (relativní stav). Pak by jeho hodnota měla vypovídat jeho přínosu pro řešení. Znovu uveďme, že takový přístup v podstatě simuluje hodnocení akcí, definovaných jako výběr jedné další pokrývající množiny.

Pro Q-learning znovu máme variantu, kdy stav reprezentuje akci, kterou použijeme pro určitý výběr pokrývajících množin. A konečně pak poslední implementovanou možností je to, že stav reprezentuje akci výběru jedné určité pokrývající množiny (relativní přístup).

5.5.3 Aplikace posilovaného učení na hledání globálních minim Ackleyho a Rastriginovy funkce

Jelikož algoritmus posilovaného učení počítá s jistými stavy, kterým přiděluje hodnotu, je nutné přistoupit k diskretizaci. Jinak by totiž těchto stavů bylo nekonečně mnoho a nebylo by prakticky možné je hodnotit.

Podobně jako u předešlých úloh, také zde byly implementovány čtyři varianty toho, co jeden stav reprezentuje. Pro TD-learning nejprve uveďme možnost, kdy jeden stav reprezentuje jeden bod v prostoru daném dimenzemi úlohy (nazvěme jej absolutním). Například pro Rastriginovu funkci začínáme se stavy, kdy jeden stav reprezentuje výběr hodnoty na místě jednotek pro všechny dimenze (například stav reprezentující $(5; 2)$). Z těchto stavů se poté dostáváme do dalších, kde stav reprezentuje zpřesnění výběrem hodnoty na pozici prvního desetinného místa (pro uvedený příklad by byl následný stav třeba $(5.0; 2.8)$). Takto pokračujeme, až dosáhneme zanoření určeného jemností diskretizace. Například pro jednodimenzionální řešení s hodnotou 1.029 by byly hodnoceny stavy 1, 1.0, 1.02 a 1.029.

Druhou variantou je, že stav reprezentuje vybrané hodnoty pouze pro jednotky, desetiny nebo setiny či obdobně (nazvěme relativním). Budeme-li se držet prvního uvedeného příkladu, pak po stavu $(5; 2)$ bude následovat stav $(0; 8)$, který reprezentuje $(0.0; 0.8)$. Budeme tak reprezentovat myšlenku, že odděleně zjistíme, jaké hodnoty ve výsledném řešení chceme mít na místě jednotek, desetin, setin atd. Předem uveďme, že toto oddělení může naprosto narušit kvalitu dosaženého řešení, neboť ztrácíme právě vztah mezi těmito úrovněmi. Přesto však vyzkoušíme i tento způsob.

Pro Q-learning postupujeme obdobně. Nejprve je uvažováno, že stav reprezentuje zpřesňující akci. Nejprve bychom měli stav odpovídající výběru hodnot $(5; 2)$ na místě jednotek. Potom uvažujeme akce, které vybírají hodnoty pro desetinná místa, ale stále v souvislosti s hodnotami na místě jednotek (např. akce vedoucí na $(5.0; 2.8)$).

Konečně pak vyzkoušíme i variantu s oddělením jednotek, desetin, setin atd. pro Q-learning. Hodnotíme tak akce pro výběr hodnot pro danou úroveň desetinného místa, nezávisle na předchozí úrovni. Redukujeme tak stavový prostor, ovšem opět za cenu ztráty informace, kterou sebou nesou vazby mezi úrovněmi.

6 Výsledky měření

Tato kapitola shrnuje naměřené výsledky při provádění testovacích úloh pomocí uvedených přístupů. Výsledky jsou uskupeny dle optimalizačních metod a jejich parametrů, jejichž nastavení zkoumáme.

Pro každý měřený parametr byly provedeny vždy alespoň desítky měření, z nichž byl vytvořen průměr. To je nezbytné z toho důvodu, jelikož všechny uvedené algoritmy využívají při řešení úloh náhodně vygenerované hodnoty a tudíž je nutné provádět průměrování, abychom zohlednili fluktuace způsobené vlivem náhodných hodnot.

Každá metoda a každý její parametr je zkoušen na konkrétních testovacích příkladech (instancích) vycházejících ze zadaných úloh. Pro problém více batohů (značíme MKP) to byly postupně instance s 5 batohy a 30 objekty, 5 batohy a 100 objekty, 10 batohy a 30 objekty a 10 batohy a 100 objekty, které mají pracovní označení popořadě „5 K + 30 P“, „5 K + 100 P“, „10 K + 30 P“ a „10 K + 100 P“. Pro problém pokrytí množiny (SCP) byly zvoleny také čtyři úlohy. Postupně s 10 prvky univerza a 10 pokrývajícími množinami, s 10 prvky univerza a 50 pokrývajícími množinami, s 20 prvky univerza a 50 pokrývajícími množinami a s 30 prvky univerza a 100 pokrývajícími množinami. Znovu pro ně zaveďme pro jednoduchost následující pracovní označení: „10 U + 10 M“, „10 U + 50 M“, „20 U + 50 M“ a „30 U + 100 M“.

Pro hledání minim Ackleyho a Rastriginovy funkce (značíme AF a RF) jsem zvolil počet dimenzí 1, 5 a 10 se značením D1, D5 a D10. U optimalizace mravenčí kolonií a pomocí posilovaného učení jsou však zvoleny dimenze 1 až 3 (D1, D2, D3).

Uveďme, že nastavení parametrů metod bylo zkoumáno tak, že pro každou metodu bylo zvoleno konvenční nastavení, jehož parametry byly postupně měněny tak, aby byly nalezeny co nejvhodnější hodnoty pro jednotlivé parametry.

Vysvětleme nyní použité popisy u svislých os následujících grafů. Pokud je uvedena *dosažená hodnota*, jedná se o průměr z dosažených objektivních ohodnocení pro danou úlohu. Často však nepotřebujeme znát přímo průměrnou dosaženou hodnotu, ale spíše nás zajímá změna, kterou dosáhneme jiným nastavením příslušného parametru. Tedy pod pojmem *zlepšení oproti nejhorší průměrné hodnotě* si představíme to, že byla vybrána nejhorší dosažená průměrná hodnota pro daná nastavení parametru (v grafu pak má nulovou hodnotu). Ostatní naměřené hodnoty pro jiné nastavení parametru pak vyjadřují přínos oproti této nejhorší hodnotě. Tedy jedná se o vyhodnocení výrazu $(\overline{x_{curr}} - \overline{x_{min}})$ pro maximalizační úlohy a $(\overline{x_{max}} - \overline{x_{curr}})$ pro minimalizační úlohy, kde $\overline{x_{curr}}$ je průměrná hodnota pro dané nastavení zkoumaného parametru a $\overline{x_{min}}$ (resp. $\overline{x_{max}}$) je minimální (resp. maximální) dosažená průměrná hodnota ze všech zkoumaných nastavení parametru. Tento přístup nám také umožňuje porovnávat v jednom grafu různé instance úloh. Při běžném vynesení hodnot by totiž pro některé instance bylo nutné vynášet hodnoty např. až 20000 a u jiných jen třeba do 2000, tudíž by bylo obtížné z grafu vyčíst malé změny pro instance s nižšími dosahovanými hodnotami.

6.1 Optimalizace mravenčí kolonií

Optimalizace mravenčí kolonií se osvědčila pro diskrétní úlohy, ale navržená reprezentace pro spojité úlohy nebyla vůbec úspěšná. Ukázalo se, že s rostoucí dimenzí spojité úlohy rostou nároky na výpočetní čas a na paměť natolik, že bylo možné naměřit hodnoty pouze pro dimenze 1-3.

Navíc je nutno poukázat na další problém, který se u spojitých úloh vyskytl. Pro standardní nastavení vykazuje metoda až nezvykle precizní výsledky. To je dáno velkým vlivem heuristického ohodnocení. Navržená reprezentace, kdy nejprve vybereme jednotky, pak desetiny, atd. totiž velmi dobře funguje pro zkoumané úlohy. Platí pro ně, že již odhad na pouhé jednotky nám dobře udá

oblast s globálním extrémem. To by ovšem zcela určitě neplatilo pro průběh libovolné funkce. Uvažme klamný problém takový, že pro jednotky by bylo nejvhodnější (dáno fitness funkcí) vybrat například hodnotu 8 a nejméně výhodné hodnotu 2, ale globální řešení by se nacházelo na hodnotě 2,2. Algoritmus zkoumá nejprve jednotky, tudíž by s velkou pravděpodobností byla prohledávána řešení v oblasti kolem hodnoty 8. Z toho důvodu jsem zvolil pro zkoumání parametrů nastavení, které klade mnohem méně důraz na deterministické řešení a vnáší do algoritmu více náhody. Tento přístup je při použité reprezentaci mnohem vhodnější pro obecnou funkci s nám neznámým průběhem.

Shrňme tedy, že pro spojité úlohy je optimalizace vhodná pouze pro malé dimenze, ale při volbě vhodné heuristiky a realizace vykazuje výborné výsledky. Zůstává ovšem otázkou, zda by člověk pro malé dimenze nepoužil raději jiné konvenční metody.

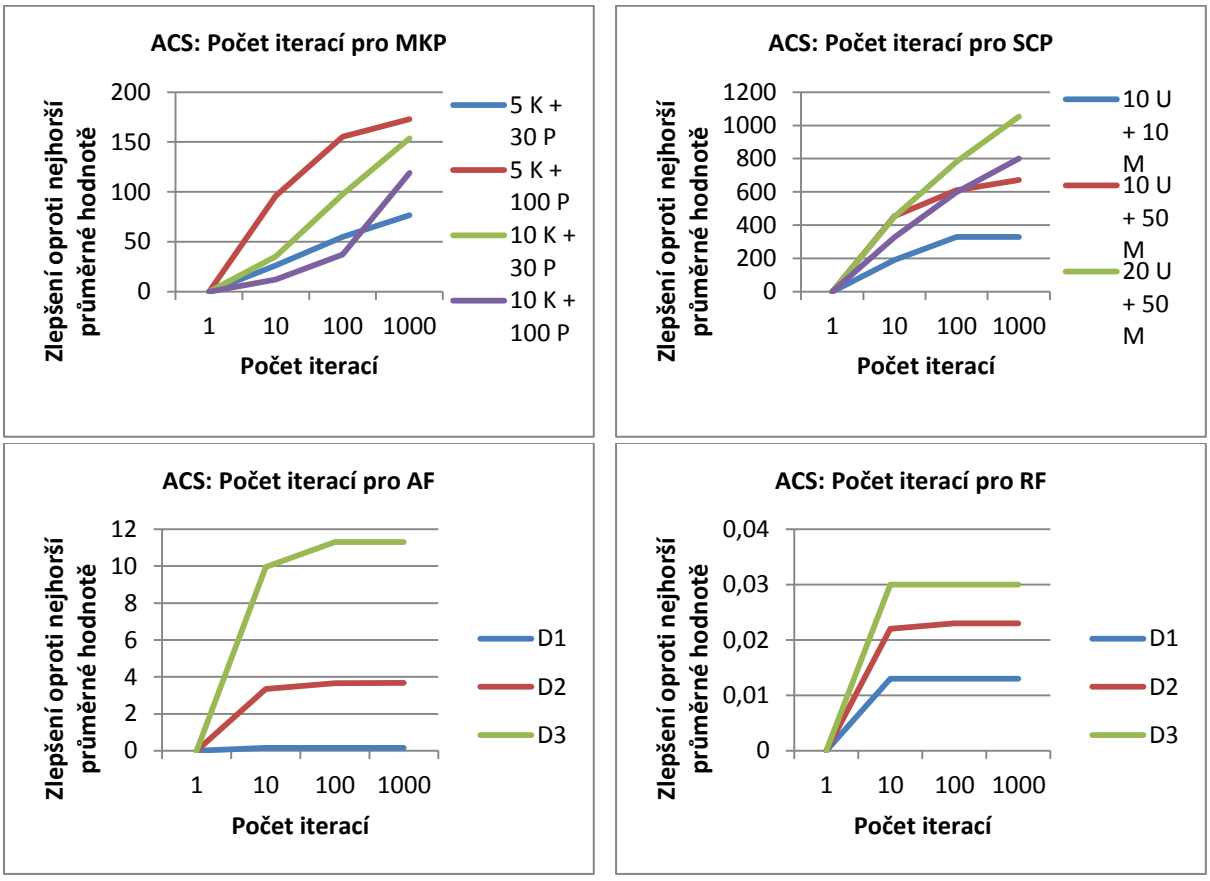
Dodejme, že pro optimalizaci mravenčí kolonií jsem kromě standardních parametrů také zkoumal různé heuristiky pro problém více batohů a pro problém pokrytí množiny (viz dále).

6.1.1 Počet iterací algoritmu

Lze předpokládat, že počet iterací algoritmu je značně specifický pro každou instanci řešených úloh. Nelze proto vyvozovat žádná pravidla o pevném počtu těchto iterací. Ovšem u uvedených problémů často stačilo použít řádově desítky iterací, jak ukazují uvedené grafy.

Poznamenejme, že pokud necháme působit při výpočtu více náhodu, je logicky lepší zvýšit počet iterací, abychom minimalizovali vliv náhodných fluktuací.

U diskrétních úloh naznačují grafy podobnost s lineární závislostí, tudíž lze říci, že s větším počtem iterací dosahujeme lepších výsledků. Jelikož bylo možné vyzkoušet jen malé dimenze pro spojité úlohy, dostačovaly i desítky iterací.

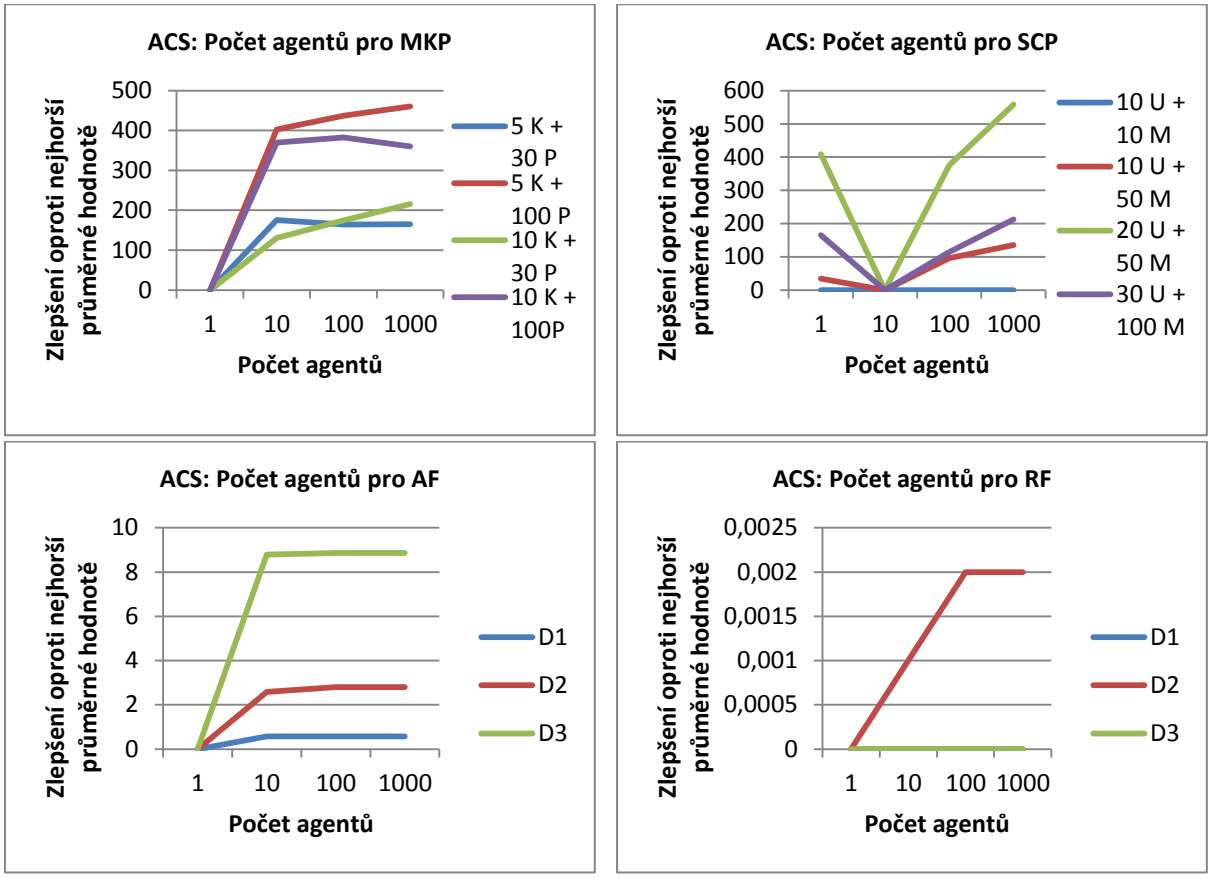


Grafy 1-4: Vliv počtu iterací na ACS.

6.1.2 Počet agentů

Počet agentů (mravenců) vyšší než jeden umožňuje (pseudo-)paralelizaci algoritmu, kdy všichni agenti současně konstruují řešení a vzájemně se tak prostřednictvím feromonového modelu ovlivňují. Z toho důvodu je počet agentů důležitý parametr.

U většiny úloh platí, že vyšší počet agentů znamená vyšší pravděpodobnost nalezení kvalitnějšího řešení. Často však stačí mít jen řádově desítky agentů pro získání dobrého výsledku. Zajímavá anomálie se vyskytla u problému pokrytí množiny, kdy nejlepší výsledky jsou dosahovány od stovek agentů. Paradoxně deset agentů vykázalo výsledky horší než agent jeden nebo agentů sto.



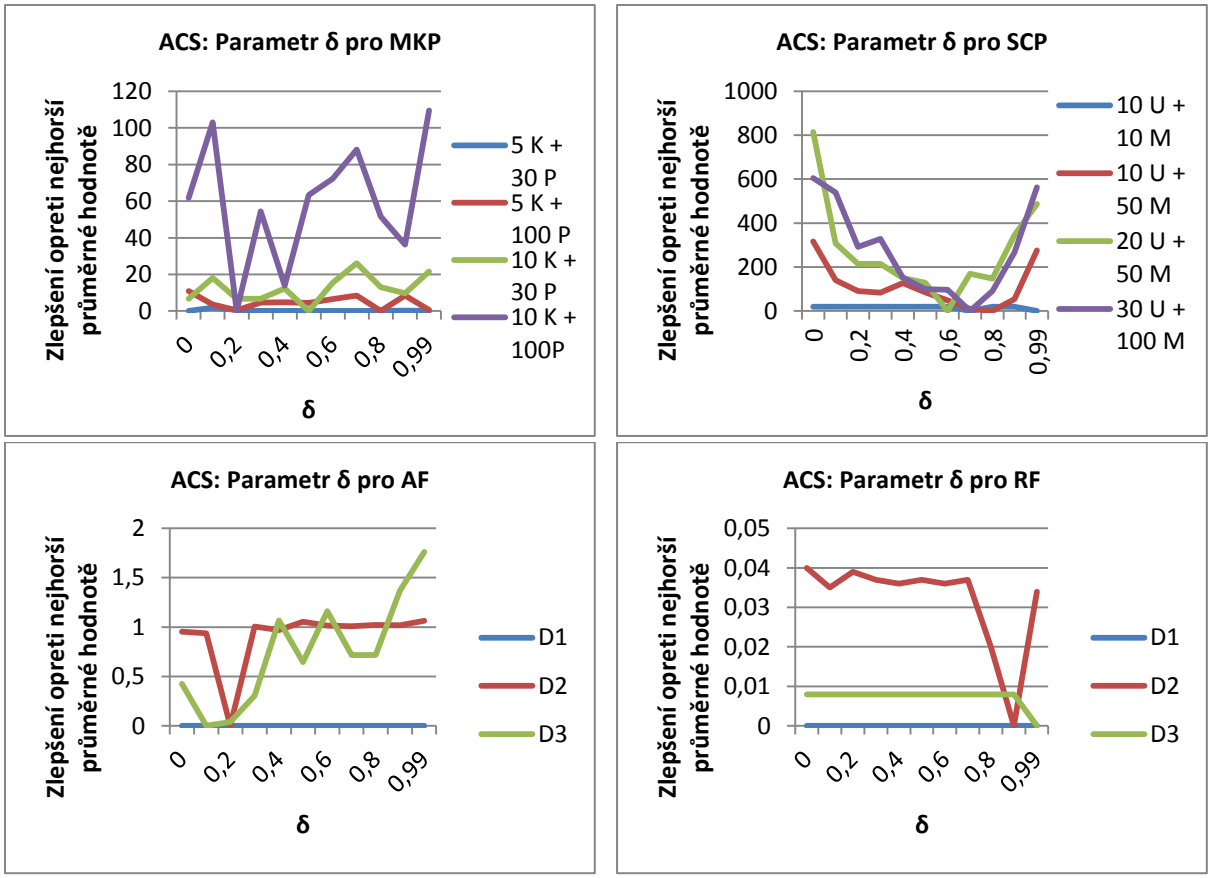
Grafy 5-8: Vliv počtu agentů na ACS.

6.1.3 Parametr δ

Parametr δ ovlivňuje rychlost vypařování feromonů ve feromonovém modelu. Jeho vhodné nastavení zajišťuje, aby nedošlo k velké kumulaci feromonů pro jednu určitou výpočetní posloupnost.

U spojitých úloh nebylo vlivem použití malých dimenzí možno naměřit signifikantní rozdíly v kvalitě dosažených výsledků při použití různé hodnoty tohoto parametru. Podobně u problému více batohů nelze stanovit nějakou pevnou hodnotu.

Pro problém pokrytí množiny se však ukázalo, že nejlepších řešení je dosahováno buď při velmi nízkém koeficientu vypařování, nebo naopak při velmi vysoké hodnotě koeficientu vypařování. Směrem ke středním hodnotám se kvalita výsledků snižuje.



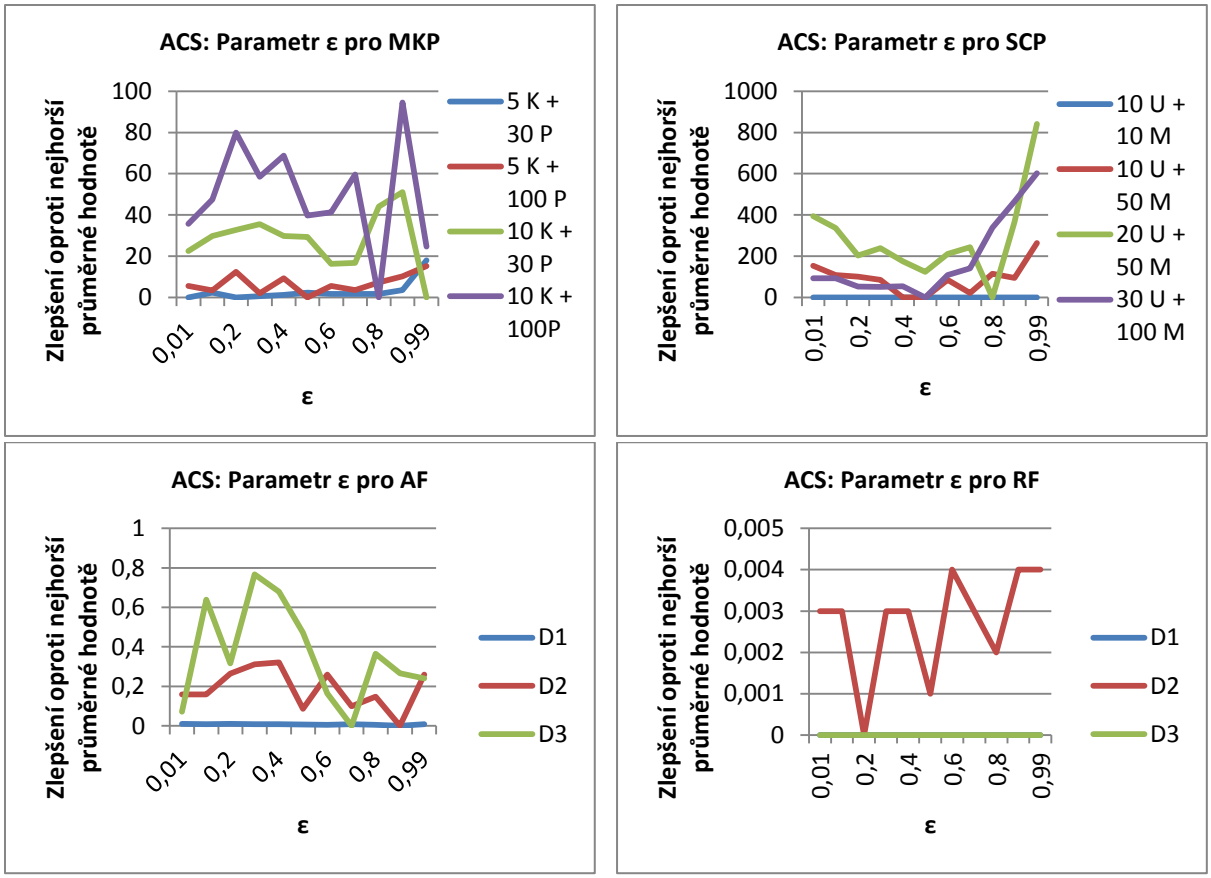
Grafy 11-12: Vliv parametru δ na ACS.

6.1.4 Parametr ε

Nastavení koeficientu ε ovlivňuje, zda při lokální aktualizaci feromonové hladiny budeme klást větší vliv na původní hodnotu, či na přidávanou defaultní hodnotu feromonu.

Pro řešení problému pokrytí množiny se ukázalo, že nejlepší řešení leží v oblasti blížící se hodnotě 1. Také u řešení problému více batohů jsem zaznamenal rozdíly v kvalitě dosaženého řešení při různé hodnotě parametru ε, ale nutno podotknout, že nebyly až tak velké jako u problému pokrytí množiny. Zde však křivky grafu nenaznačují tak jasně optimální nastavení. Lze pouze odhadnout ze složitějších instancí problému, že nejlepší hodnota by se mohla pohybovat kolem 0.2 nebo 0.9.

Pro spojité úlohy nebyly naměřeny signifikantní rozdíly v kvalitě, jelikož úlohy s malými dimenzemi nejsou natolik závislé na nastavení parametrů. Přesto pro hledání minima Ackleyho funkce bychom mohli odhadovat, že by se optimální nastavení mohlo pohybovat v intervalu 0.3-0.4.



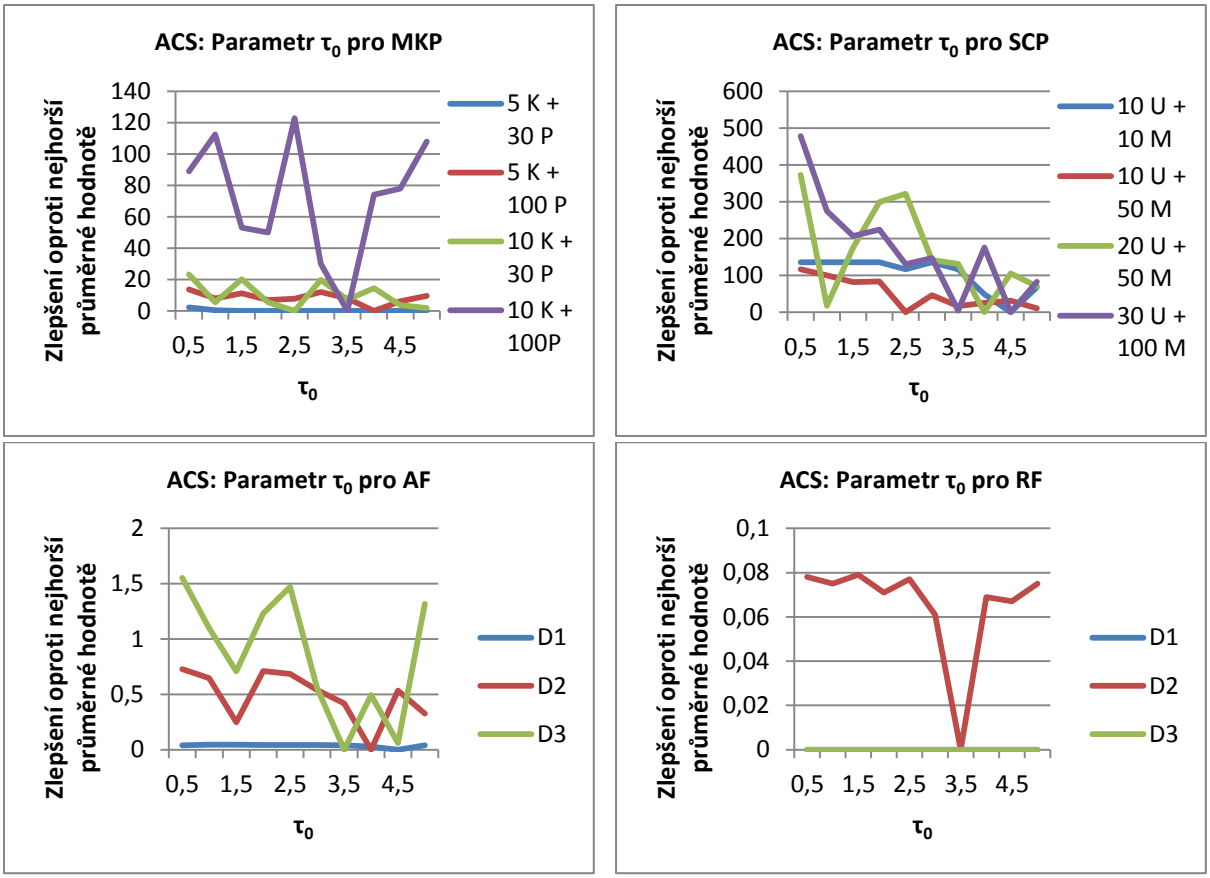
Grafy 15-16: Vliv parametru ε na ACS.

6.1.5 Parametr τ_0

Defaultní hodnota feromonu je při lokální aktualizaci feromonové hladiny zastoupena tímto parametrem. Jeho hodnoty pro řešené úlohy bývají různorodé, volí se například 1. Proto nás při tomto měření nebude zajímat jen samotná cílová hodnota, ale i schopnost ovlivnit kvalitu výsledků.

Obzvláště u diskrétních úloh vidíme, že vhodné nastavení je skutečně schopno ovlivnit dosaženou kvalitu výsledků, avšak pouze pro problém pokrytí množiny lze stanovit z naměřených dat konkrétní hodnotu a to přibližně 0.5. U problému více batohů pak nejsložitější instance ukazuje vliv tohoto parametru, ovšem z naměřených dat jsem nebyl jednoznačně schopen určit nejvhodnější hodnotu. Je též možné, že se optimálnější hodnota pohybuje mimo měřený interval, jelikož jeho rozmezí se jen těžko odhaduje, přesto však bylo dosahováno velmi kvalitních řešení.

U diskrétních úloh je takřka nemožné o optimálním nastavení ze získaných výsledků cokoli říci, jelikož jsme nebyli schopni obsáhnout i složitější dimenze, tudíž se výsledky pohybovaly velmi blízko optimálním hodnotám a vykazovaly buďto jen nepatrné odchylky či nejednoznačné chování.



Grafy 19-20: Vliv parametru τ_0 na ACS.

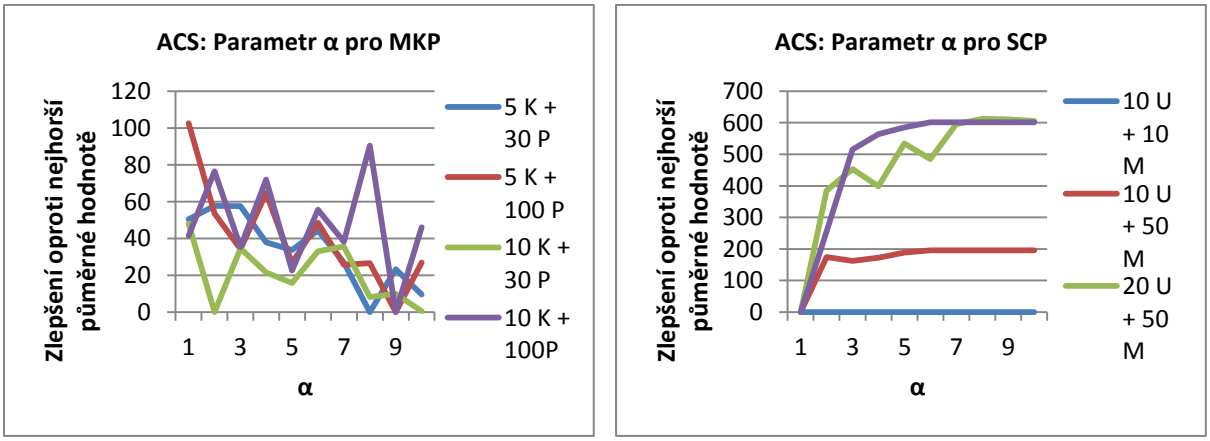
6.1.6 Parametr α

Parametr α umožňuje umocnit vliv feromonů při výběru následujícího kroku mravence. Jeho hodnota se běžně uvažuje jako 1, ale přesto se přesvědčíme o jeho vlivu na kvalitu řešení.

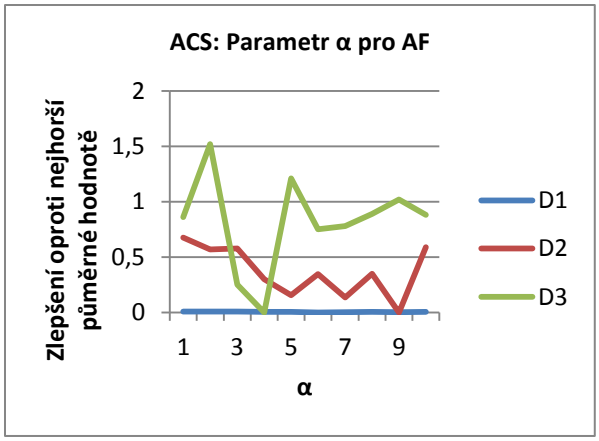
Při pohledu na graf pro problém více batohů si můžeme všimnout, že pro jednodušší instance se kvalita dosaženého řešení s rostoucí mocninou spíše snižuje, je tedy dosahováno nejlepších výsledků s hodnotou kolem 1. Pro vyšší instance nelze toto tvrzení ani potvrdit ani vyvrátit.

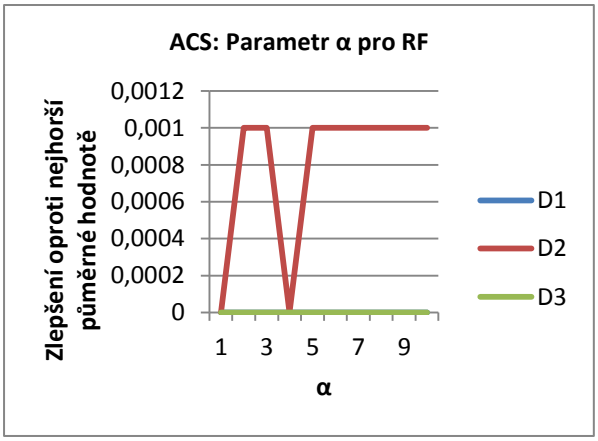
Zcela opačná je situace u problému pokrytí množiny. Vidíme, že jasně pozitivně reaguje na zvýšení mocniny feromonu. Nejlepších výsledků bylo dosahováno s mocninami od 2 až 3 a více.

Pro spojité úlohy nebylo opět možné z provedených měření stanovit nějaký jednoznačný závěr.



Grafy 21-22: Vliv parametru α na ACS.





Grafy 23-24: Vliv parametru α na ACS.

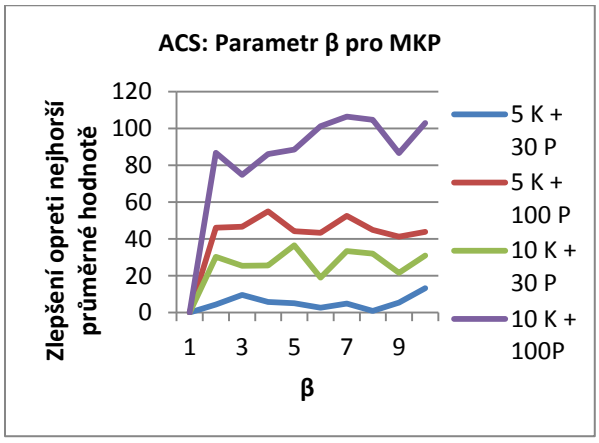
6.1.7 Parametr β

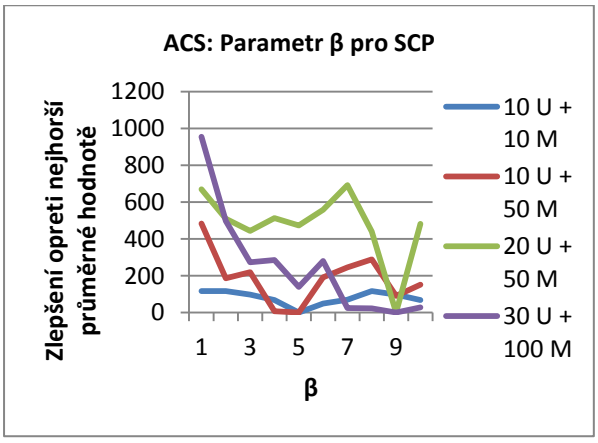
Tento parametr představuje mocninu hodnoty dané heuristickou funkcí. Umožňuje tak zvýšit její vliv. Lze předpokládat, že kde se bude lépe osvědčovat vyšší hodnota parametru α, tam bude lépe nastavit nižší hodnotu parametru β a naopak, což potvrzují i měření.

Pro problém více batohů se jasně ukazuje, že heuristika má velmi pozitivní vliv na kvalitu řešení, proto nejlepších řešení je dosahováno při nastavení parametru na hodnotu 2 a vyšší.

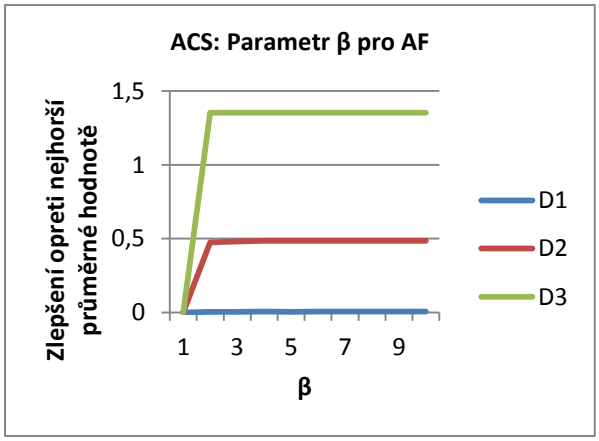
U problému pokrytí množiny je tomu přesně obráceně. Nejlepších výsledků bylo dosaženo za požití hodnoty 1.

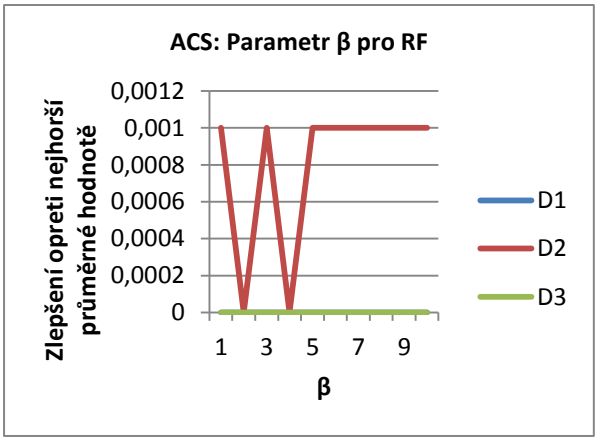
U hledání minima Ackleyho funkce se osvědčily hodnoty 2 a vyšší. U Rastriginovy funkce se nám nepodařilo měřením získat data s relevantními odchylkami.





Grafy 25-26: Vliv parametru β na ACS.





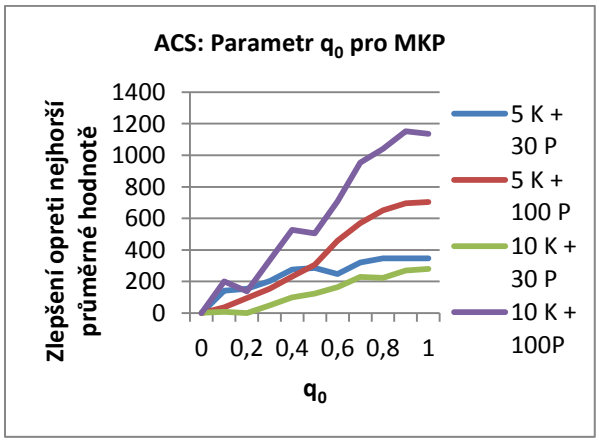
Grafy 27-28: Vliv parametru β na ACS.

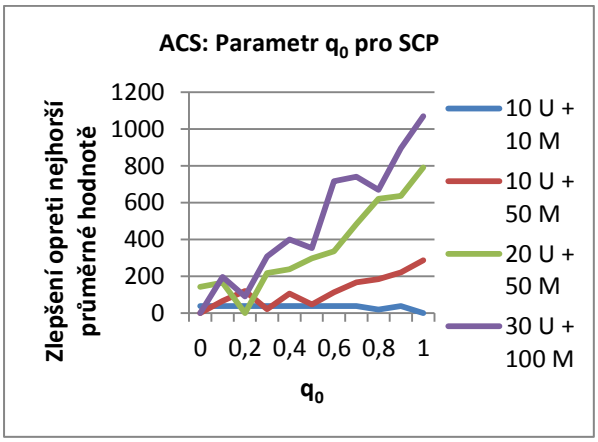
6.1.8 Parametr q_0

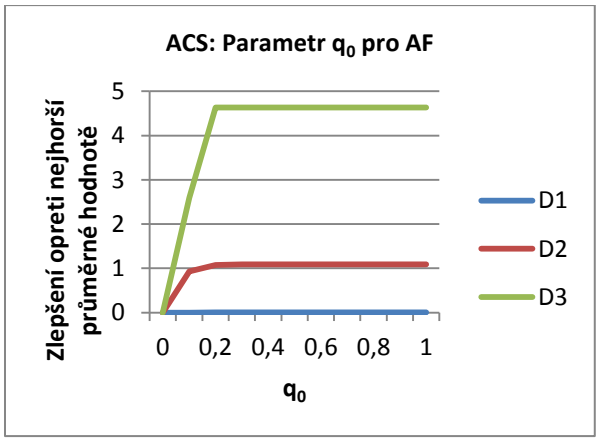
Tento parametr velmi významně ovlivňuje chování algoritmu, jelikož určuje pravděpodobnost, s jakou bude použito výběrové pravidlo ACS namísto AS (viz kapitola 3).

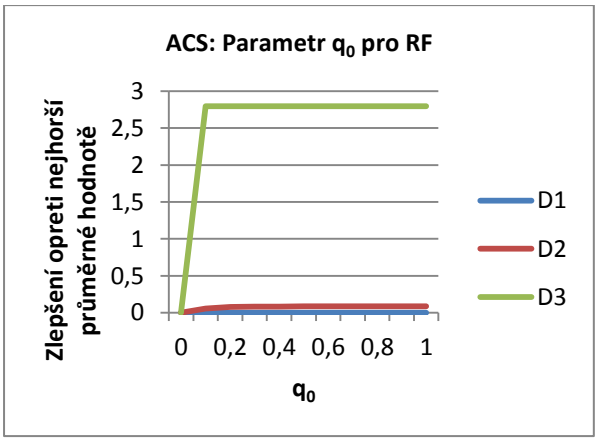
Z výsledků pro všechny úlohy je jasně vidět, že je vhodné užívat vyšší hodnotu parametru q_0, tedy více upřednostňovat pravidlo ACS.

Poznamenejme, že obecně není však vhodné nastavit hodnotu přímo na 1, ale nechat ji o něco málo menší (často například 0.9), jelikož výběrové pravidlo AS vnáší do algoritmu jistou míru náhodnosti, což někdy může vést k objevení zajímavých řešení.









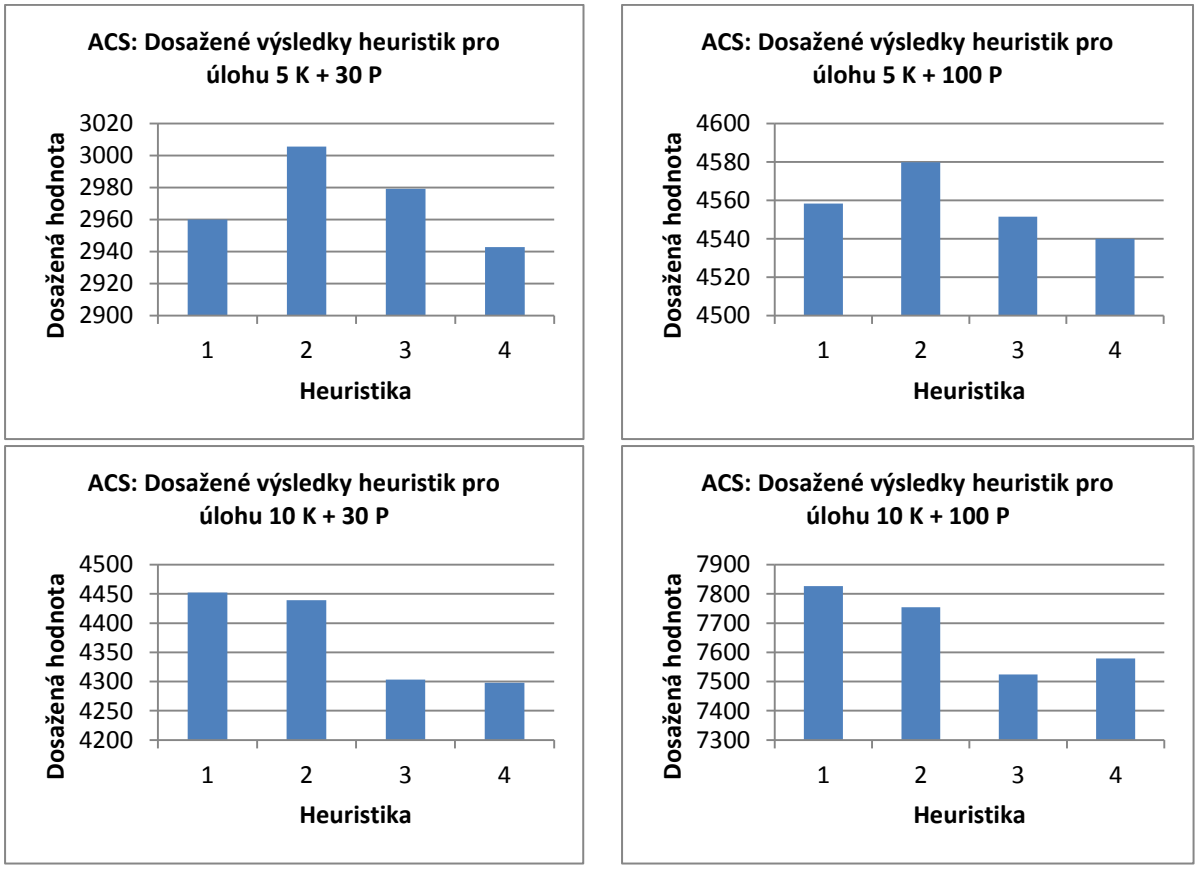
Grafy 29-32: Vliv parametru q_0 na ACS.

6.1.9 Heuristiky pro řešení problému více batohů

Jelikož algoritmus optimalizace mravenčí kolonií vyžaduje použití heuristiky, pro problém více batohů byly vyzkoušeny čtyři uvedené heuristiky.

Dosažené výsledky naznačují, že nejvhodnějšími heuristikami jsou heuristiky statické: heuristika zkoumající pouze objem a cenu objektu (heuristika 1) a heuristika, která zkoumá objem a cenu objektu a celkovou kapacitu batohu (heuristika 2). Naproti tomu se tolik neosvědčily dynamické heuristiky, které využívají informace o aktuálním zaplnění batohu. Tento trend se pak zvláště projevuje pro úlohy s větším počtem objektů a batohů.

Podobné výsledky pro problém multidimenzionálního batohu, který je principiálně velmi příbuzný problému více batohů, lze najít například v [31].

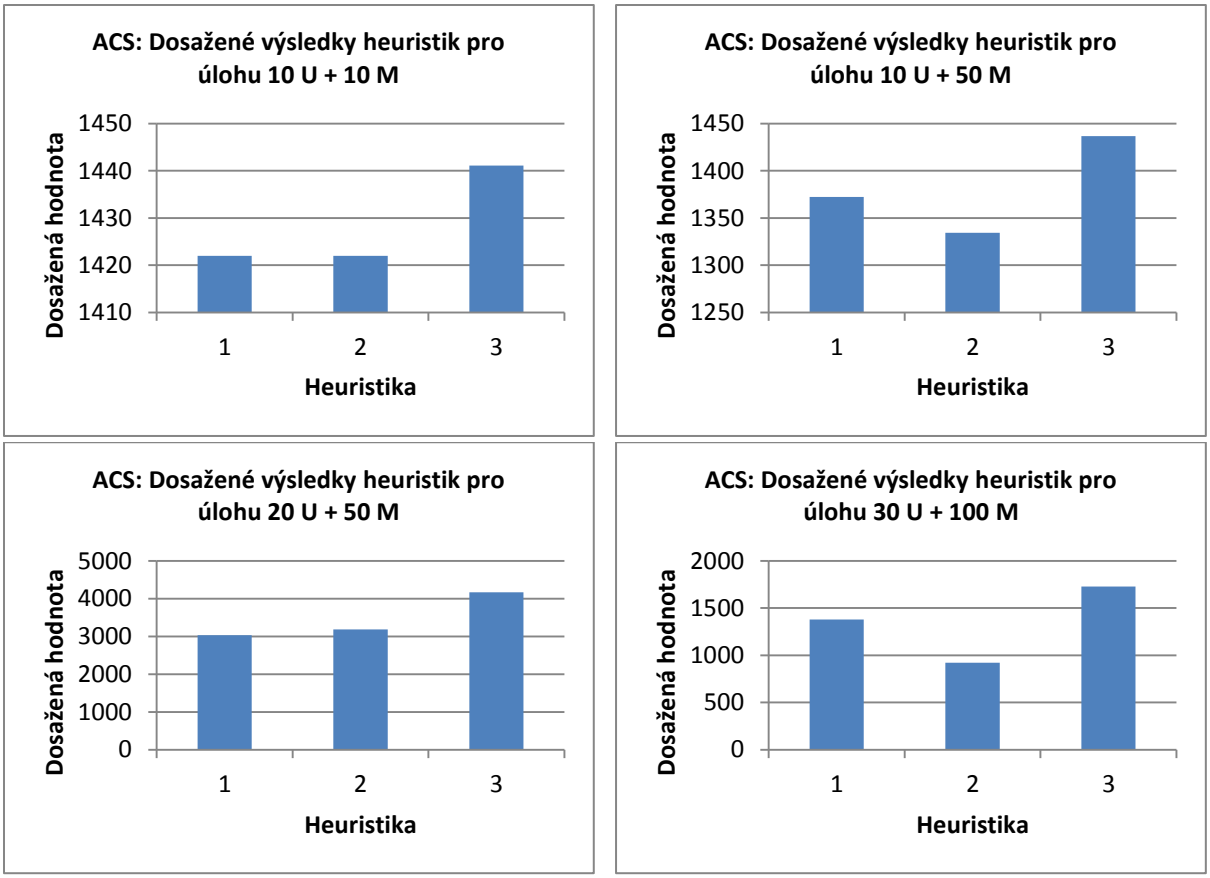


Grafy 33-36: Heuristiky pro problém více batohů.

6.1.10 Heuristiky pro řešení problému pokrytí množiny

Pro problém pokrytí množiny jsem při optimalizaci mravenčí kolonií vyzkoušel tři různé heuristiky. Všechny heuristiky vycházejí z dostupných údajů. Tedy dle toho, kolik množina pokrývá doposud nepokrytých prvků a z ceny této množiny. První heuristika (označme ji 1) využívá právě informaci o počtu nově pokrytých prvků univerza. Jelikož tato informace závisí na konkrétním výběru pokrývajících množin, jedná se o dynamickou heuristiku. Druhá heuristika, také dynamická, danou hodnotu dělí cenou. Poslední heuristika (heuristika 3) pak upřednostňuje množiny s nižší cenou. Jedná se tedy o statickou heuristiku.

Naměřené výsledky ukazují, že většinou se nejlépe osvědčuje heuristika 2, která je následována heuristikou 1. Třetí heuristika pak vždy vykazuje výrazně horší výsledky při dané minimalizační úloze.



Grafy 37-40: Heuristiky pro problém pokrytí množiny.

6.2 Optimalizace hejnem částic

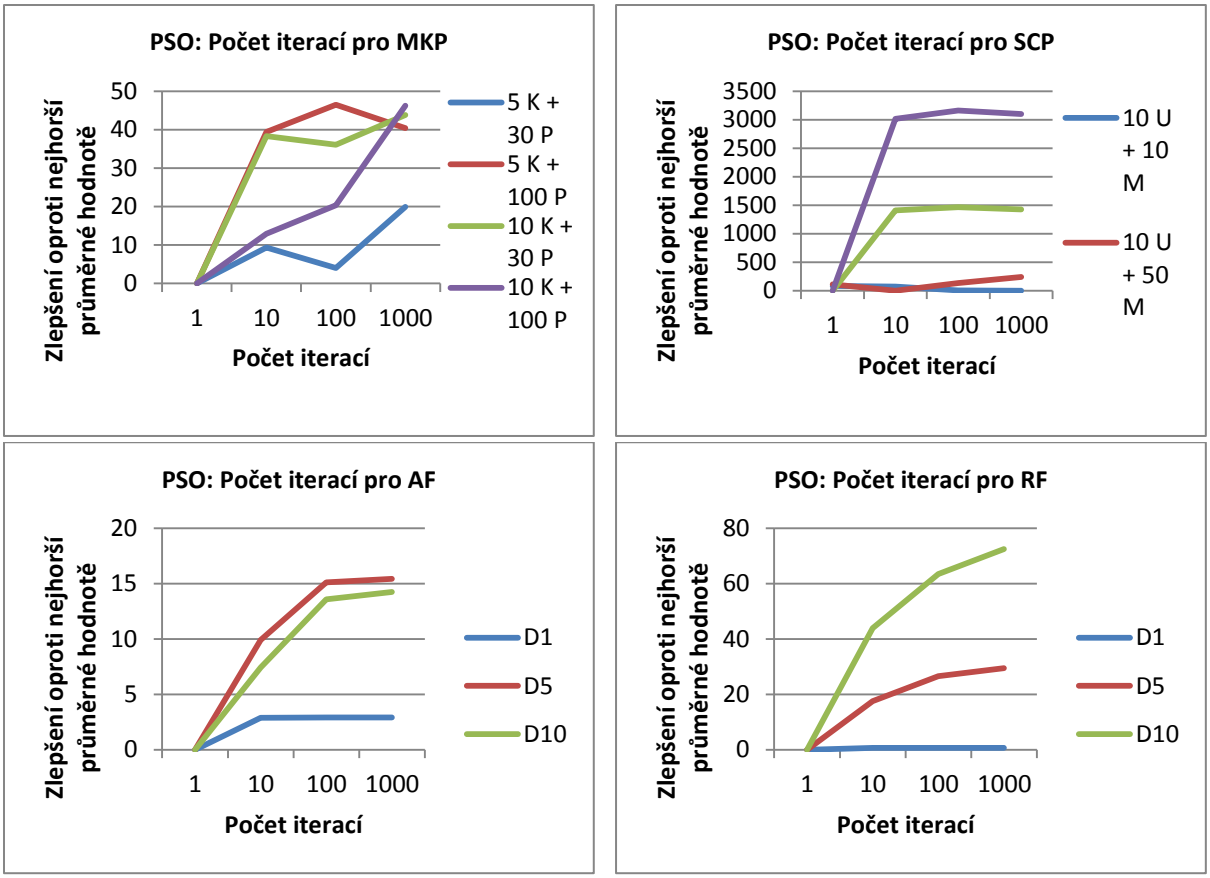
Optimalizace hejnem částic primárně pracuje se spojitými prostory, proto není divu, že se hodí pro řešení spojitých úloh. Tato metoda se však rovněž osvědčila pro diskrétní úlohy, přičemž bylo dosahováno kvalitních výsledků.

6.2.1 Počet iterací algoritmu

Pro počet iterací algoritmu lze logicky předpokládat, že čím vyšší tento počet bude, tím větší bude pravděpodobnost toho, že bude nalezeno kvalitnější řešení.

Dle naměřených výsledků předpoklad platí pro všechny zvolené úlohy, přičemž grafy taktéž názorně ilustrují, že nutnost většího počtu iterací roste se složitostí instance dané úlohy.

Zatímco u problému pokrytí množiny dostačovalo i 10 iterací, v ostatních úlohách bylo nutné volit alespoň stovky až tisíce iterací.



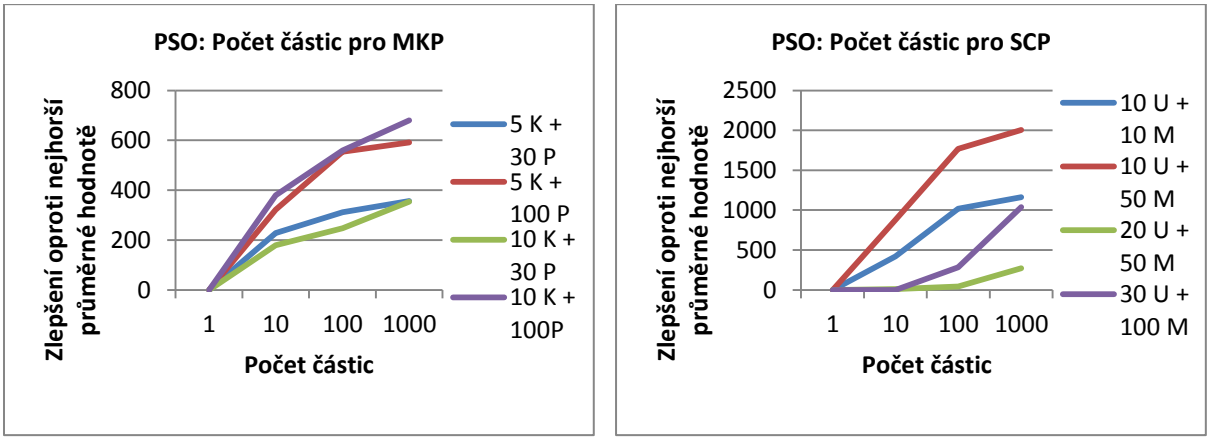
Grafy 41-44: Vliv počtu iterací na PSO.

6.2.2 Počet částic

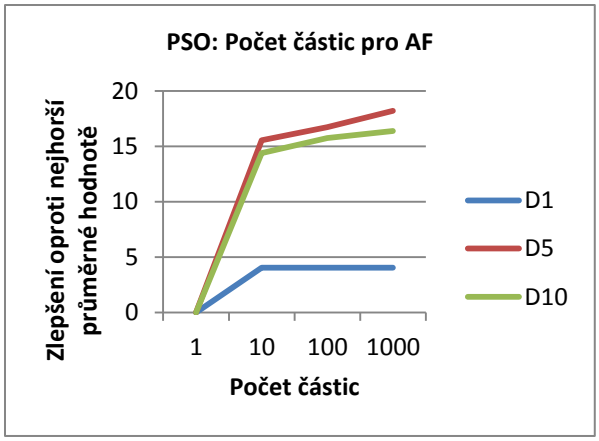
Počet částic je velmi důležitým parametrem algoritmu. Nelze předpokládat, že si například ponecháme jen jednu částici a adekvátně zvýšíme počet iterací a dosáhneme stejného výsledku. Částice mezi sebou totiž komunikují dle zvolené topologie. A právě komunikace mezi částicemi vytváří přidanou hodnotu, na níž je činnost algoritmu založena.

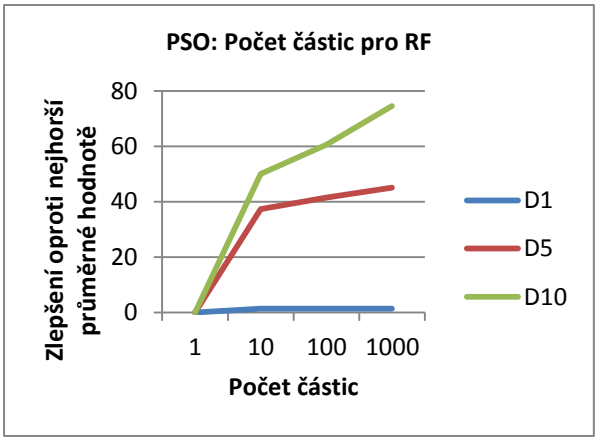
Vliv počtu částic potvrzují naměřené hodnoty. Z grafů lze vyčíst pozitivní vliv většího počtu částic. Pro úlohu hledání minima Ackleyho funkce se ukázalo, že již od desítek částic jsou vykazovány kvalitní výsledky. U ostatních úloh bylo zapotřebí spíše stovek až tisíců částic.

Také se ukázalo, že lze předpokládat, že u složitějších instancí problémů je třeba více částic.



Grafy 45-46: Vliv počtu částic na PSO.



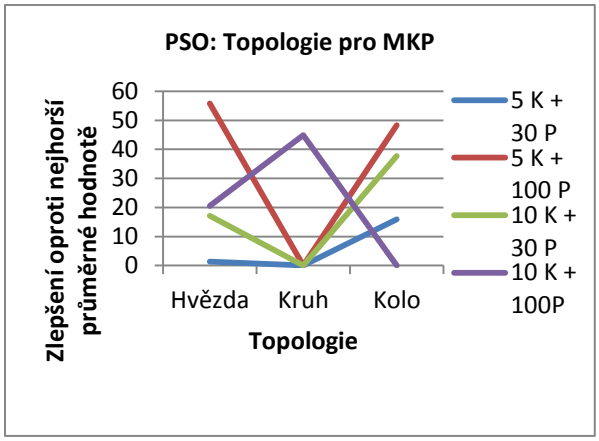


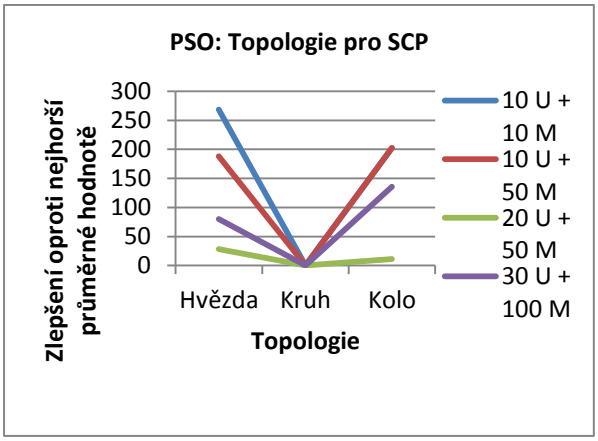
Grafy 47-48: Vliv počtu částic na PSO.

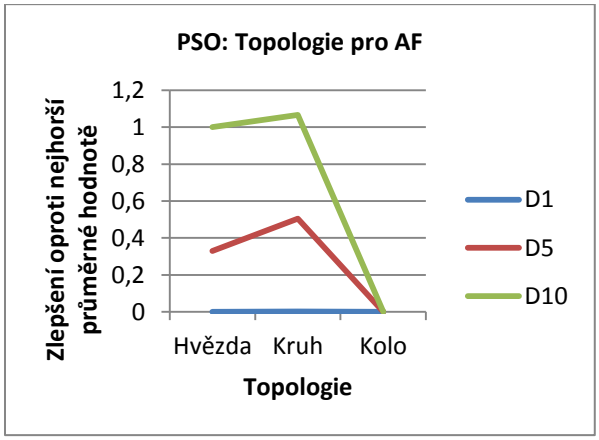
6.2.3 Topologie

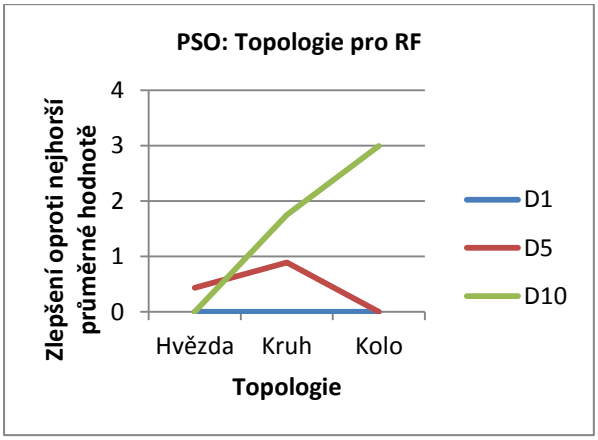
Pokud přihlédneme k předchozímu měření, tak víme, že spolupráce částic je důležitá. Z toho pohledu je nasnadě vyzkoušet běžně užívané topologie propojení těchto částic, kterými jsou topologie hvězdy, kruhu a kola.

Z výsledných grafů vidíme, že nejen globálně, ale i pro jednotlivé úlohy je obtížné určit nejlepší topologii. Pro problém pokrytí množiny lze odhadnout, že by nejlepšími topologiemi mohly být hvězda a kolo. Podobně by tomu mohlo být i u problému více batohů, ale u nejsložitější instance se nejlépe osvědčila kruhová topologie. Obdobně obtížná je i volba u hledání minima Rastriginovy funkce. U hledání minima Ackleyho funkce se jeví jako nejlepší kruhová topologie.









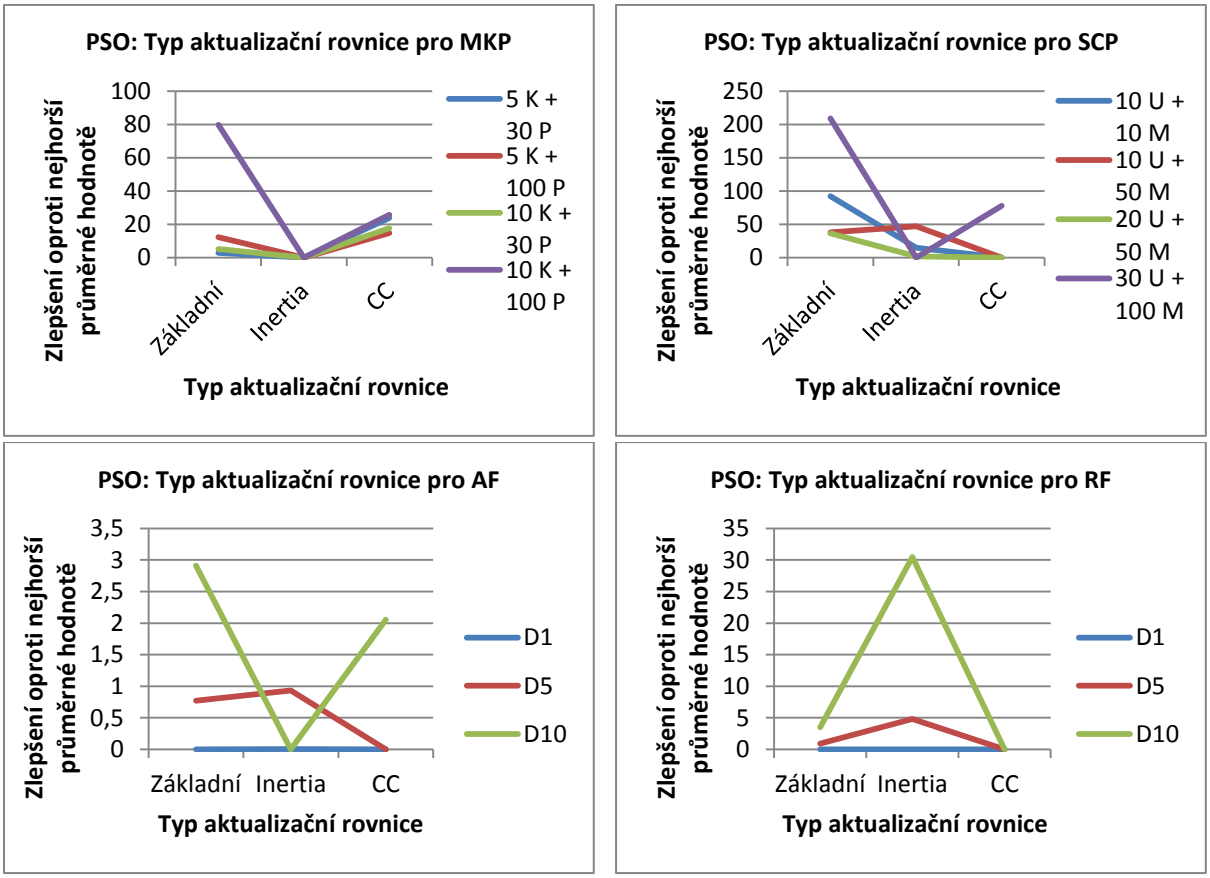
Grafy 49-52: Vliv topologie částic na PSO.

6.2.4 Aktualizační rovnice

V teoretické části této práce jsme uvedli tři typy aktualizační rovnice pro rychlost částic. Kromě té základní se jednalo o rovnici s názvem Inertia a rovnici s omezující podmínkou, či omezujícím koeficientem (značíme ji CC).

Výsledky naznačují, že pro problém více batohů by mohly být vhodné základní a CC rovnice. U problému pokrytí množiny se většinou nejlépe osvědčilo použití základní rovnice. Ačkoliv pro hledání minima Rastriginovy funkce bych volil rovnici typu Inertia, u Ackleyho funkce není dána žádná jasná volba.

Podobně jako u volby topologie, zde je také dobré pro každou instanci úlohy vyzkoušet více typů aktualizační rovnice.

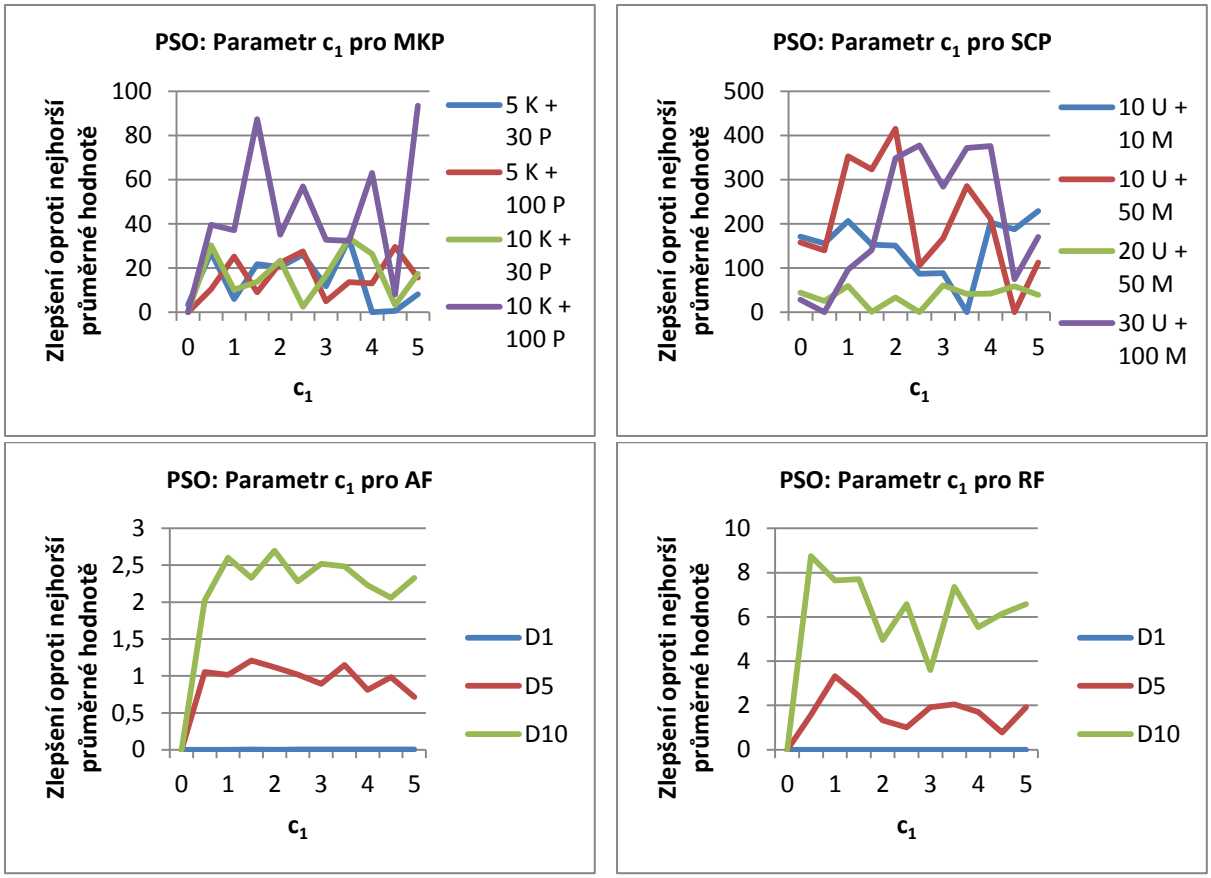


Grafy 53-56: Vliv volby aktualizační rovnice na PSO.

6.2.5 Parametr c_1

Parametr c_1 nám určuje akceleraci k lokálnímu maximu, neboli jakou váhu mu přidělujeme. Často se lze setkat s variantou, že jeho hodnota je jedna, proto jsem zkoumal hodnoty v nedalekém okolí.

Ukázalo se, že pro každou konkrétní instanci každé úlohy je optimální jiné nastavení, proto je těžké dát určité doporučení pro tento parametr. Můžeme si však všimnout, že pro spojité hodnoty se celkem osvědčila hodnota kolem 0.5-1.5. A také je pro řešení problému více batohů nezbytné, aby nebyl tento parametr nulový, jelikož vliv lokálního maxima je zřejmě pro kvalitu řešení důležitý.



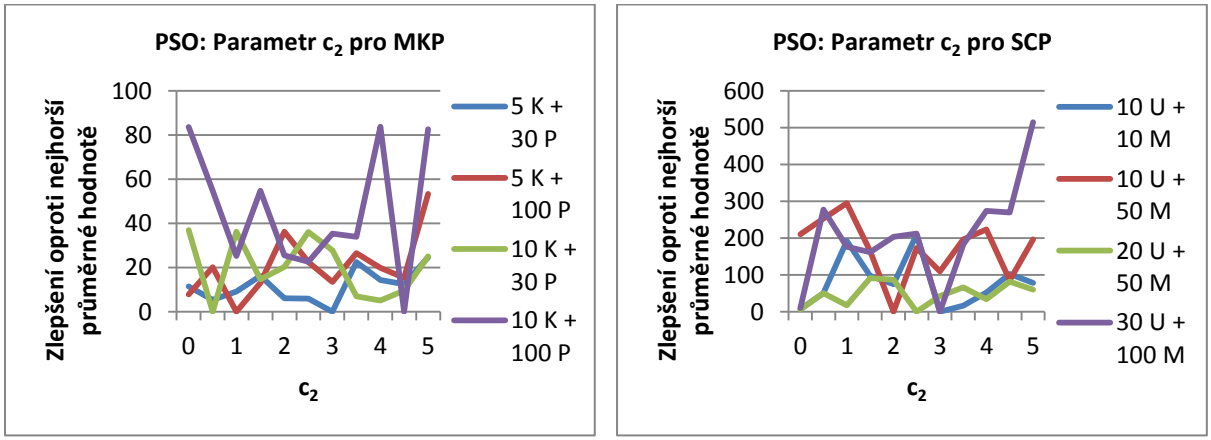
Grafy 57-60: Vliv parametru c_1 na PSO.

6.2.6 Parametr c_2

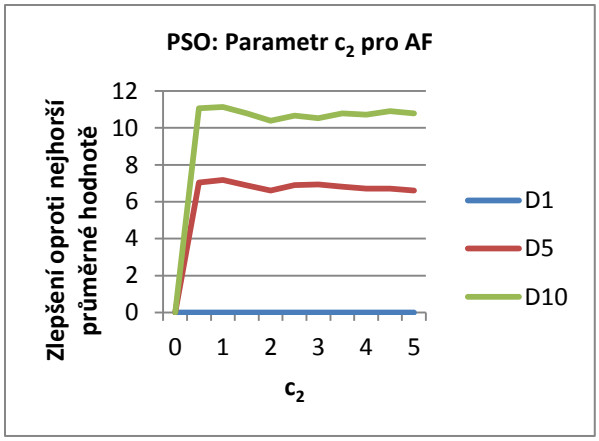
Parametr c_2 určuje, jak velký vliv má na novou rychlost globálně nejlepší pozice, přesněji určuje akceleraci k této pozici. Podobně jako u parametru c_1 jsem i zde zkoumal hodnoty v okolí blízkém hodnotě 1.

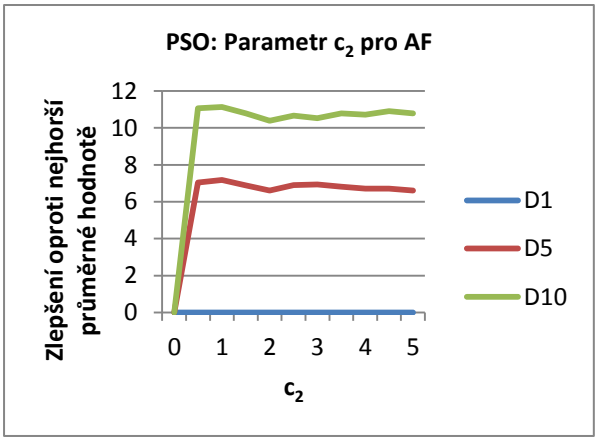
Pro spojité úlohy měření vykazují podobné výsledky jako u parametru c_1, tedy nejlépe se osvědčily hodnoty větší než 0.5. U hledání minima Rastriginovy funkce se od hodnoty 1 a výše u nejsložitější instance objevuje kolísání a náznak klesání kvality řešení.

Pro diskrétní problémy sice měření prokázala, že různé nastavení hodnoty tohoto parametru přináší různě kvalitní výsledky, ale nepodařilo se určit nějakou konkrétní hodnotu.



Grafy 61-62: Vliv parametru c_2 na PSO.





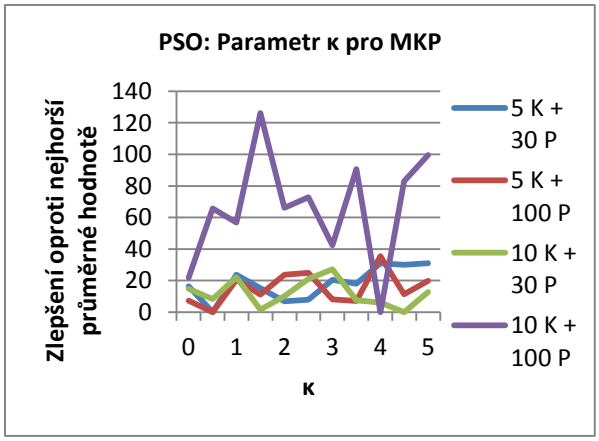
Grafy 63-64: Vliv parametru c_2 na PSO.

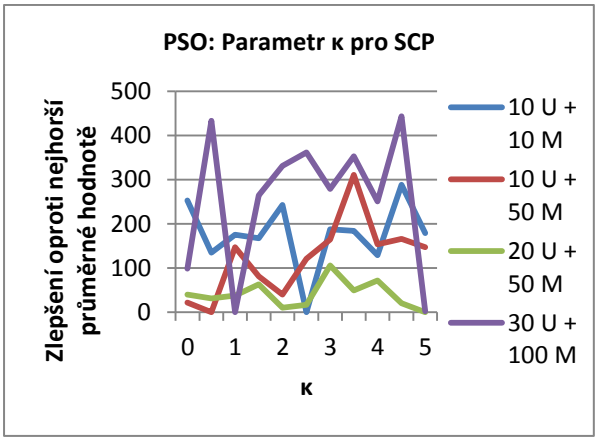
6.2.7 Parametr κ

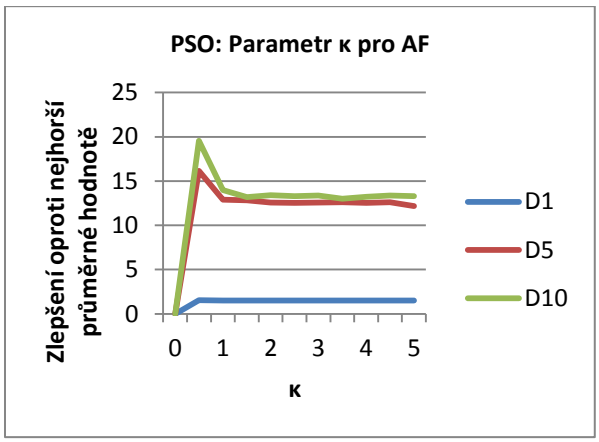
Parametr κ (někdy značený jen k) je parametrem, který se vyskytuje v aktualizační rovnici s omezující podmínkou. Obvykle je nastavován na hodnotu 1, přesto se však pokusíme nalézt jinou hodnotu, která by byla pro zvolené úlohy lepší.

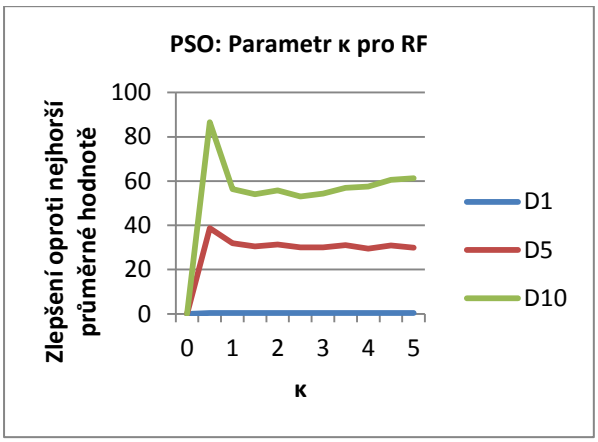
Pro zvolené spojité úlohy se ukázalo, že je lépe použít hodnotu lehce nižší, než je 1, konkrétně se lépe osvědčila hodnota 0.5.

Pro diskrétní úlohy jsme sice dosáhli lepších výsledků, než s hodnotou 1, ale pro každou jednotlivou instanci se ukázala být lepší jiná hodnota. Tím pádem se nepodařilo určit jednu konkrétní hodnotu.









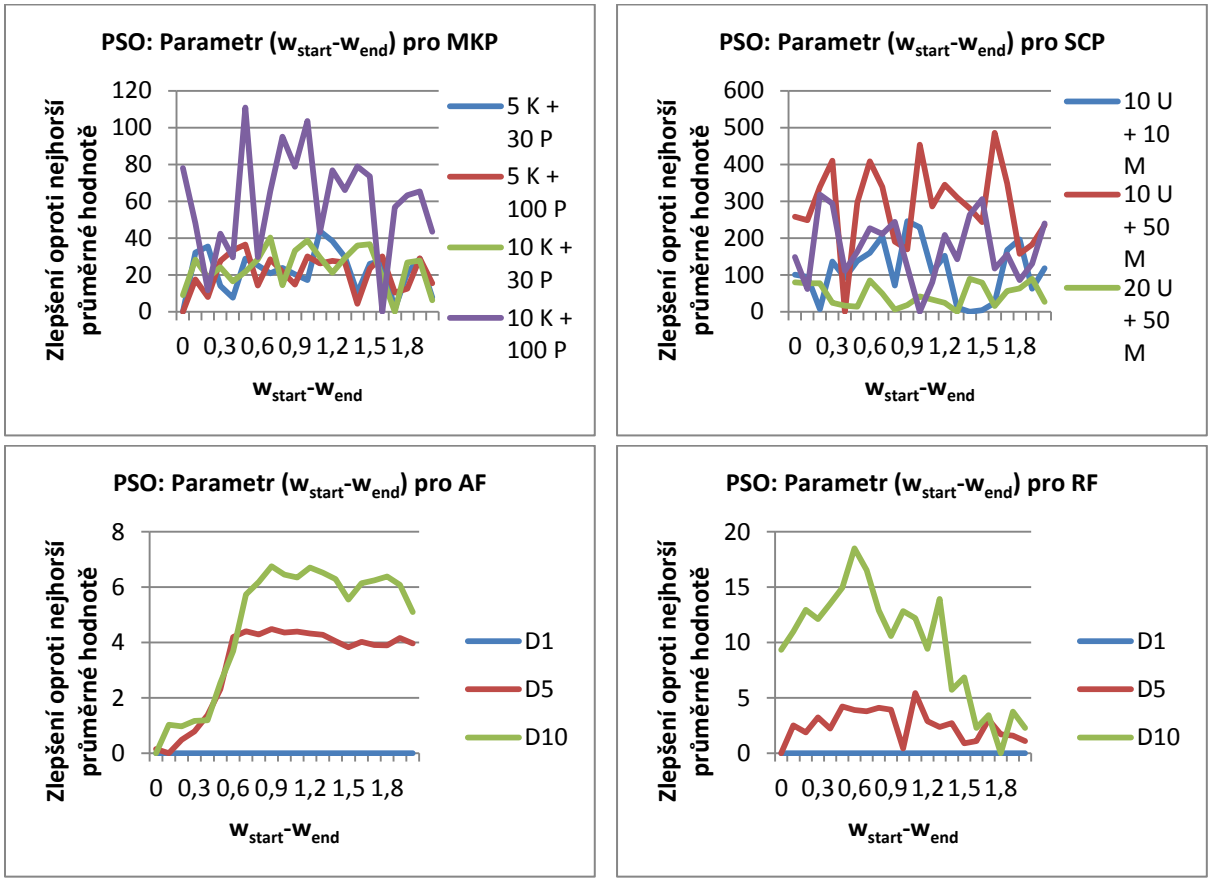
Grafy 65-68: Vliv parametru κ na PSO.

6.2.8 Parametr (w_{start}-w_{end})

V aktualizační rovnici s názvem Inertia se vyskytuje pro jeden běh algoritmu konstantní výraz $(w_{start} - w_{end})$. Nebudeme zkoumat jednotlivé prvky tohoto výrazu zvlášť, ale nyní nás bude zajímat jeho hodnota jako celku.

Podobně jako u předešle uvedeného parametru, tak i zde nelze provést jednoznačný závěr pro diskrétní úlohy. Výsledky vykazují signifikantní rozdíly v kvalitě dosaženého řešení při různé hodnotě zadaného rozdílu, avšak na základě naměřených dat nelze předem předpovědět pro konkrétní instanci jeho optimální hodnotu.

Pro hledání minima Rastriginovy funkce výsledný graf naznačuje, že by se vhodná hodnota mohla pohybovat kolem 0.5. U Ackleyho funkce se kvalitní výsledky vyskytují s hodnotou 0.5 a vyšší.

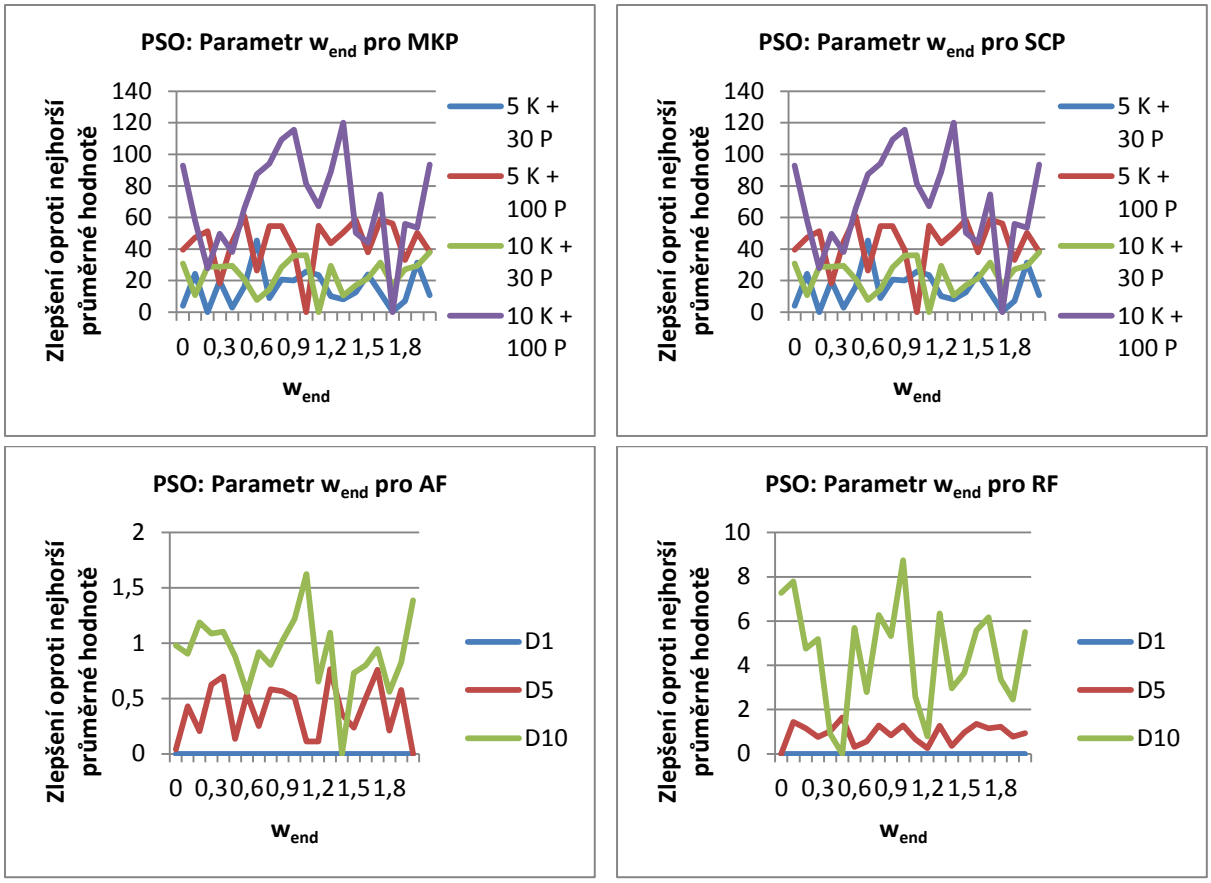


Grafy 69-72: Vliv hodnoty výrazu (w_{start}-w_{end}) na PSO.

6.2.9 Parametr w_{end}

Předešlý zkoumaný parametr představoval rozdíl $(w_{start} - w_{end})$, avšak je nutné ještě určit alespoň w_{end}, abychom věděli, jaké hodnoty jsou v tomto rozdílu zastoupeny.

Ačkoli grafy jasně ukazují signifikantní rozdíly při různé volbě různé hodnoty w_{end}, není možné ani pro jednu úlohu určit nějakou obecnou optimální hodnotu.



Grafy 73-76: Vliv parametru w_{end} na PSO.

6.3 Optimalizace genetickými algoritmy

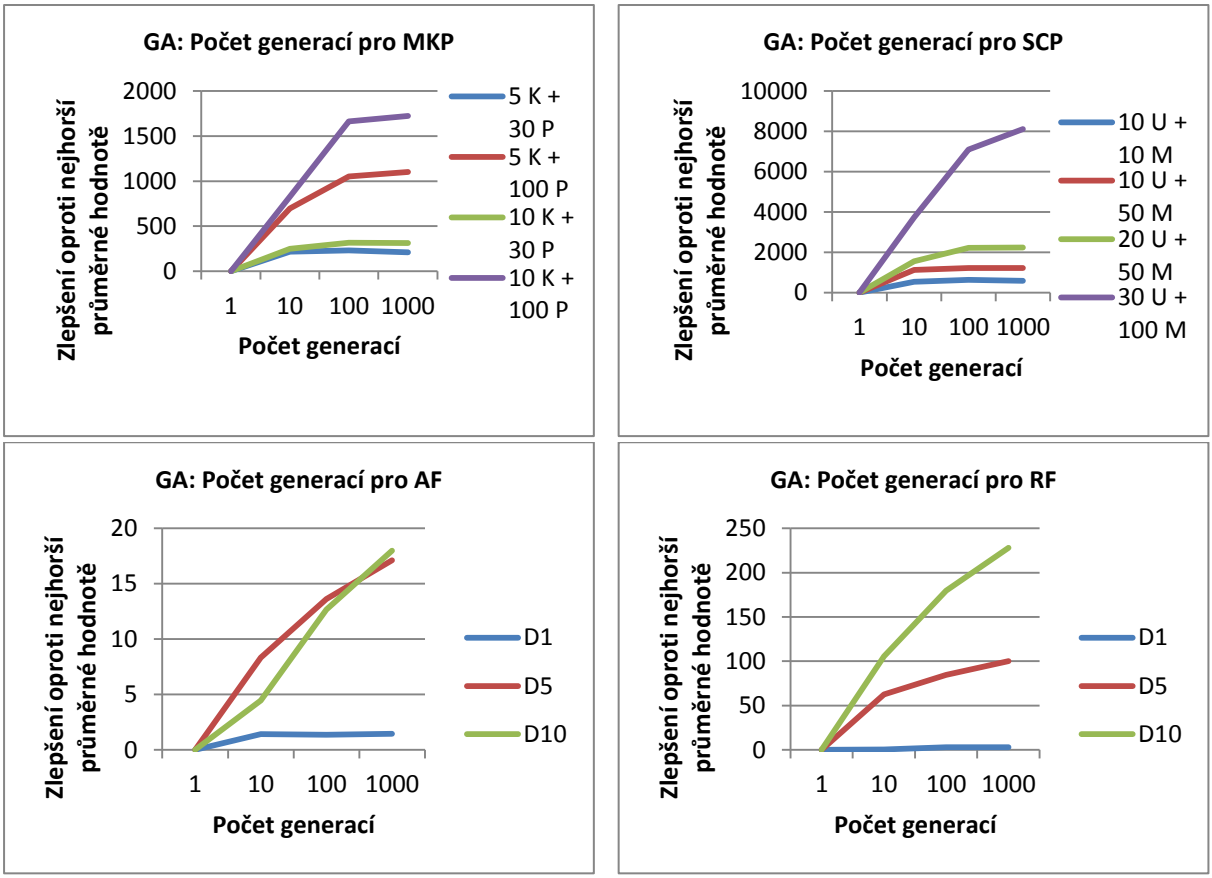
Genetické algoritmy bývají často vyzdvihovány pro svou univerzálnost. To potvrdila provedená měření. Ukázalo se, že jsme schopni dosahovat s genetickými algoritmy výborných výsledků jak pro diskrétní, tak i pro (diskretizované) spojité úlohy.

6.3.1 Počet generací

Je naprosto přirozené, že pro algoritmus založený na náhodě, znamená vyšší počet generací (iterací) větší šanci na objevení globálního extrému, tudíž je logické nastavit tuto hodnotu na co nejvyšší možnou přijatelnou. Avšak snažil jsem se nalézt co nejmenší počet generací, který by nám pro dané úlohy nacházel dostatečně kvalitní řešení.

Pokud pohlédneme na grafy s výsledky, můžeme odhalit, že nejvýznamnější růst kvality dosažených řešení pro diskrétní problémy je do okamžiku použití řádově stovek generací. Dalším přechodem na řádově tisíce generací dosáhneme pochopitelně kvalitnějších řešení, ale rozdíl již není tak významný. Proto i s řádově stovkami iterací jsme schopni dosáhnout poměrně kvalitního řešení.

U spojitých úloh lze vypozorovat, že čím vyšší je dimenze, tak tím více generací je třeba. To je dáno tím, že prohledávaný prostor roste s počtem dimenzí.



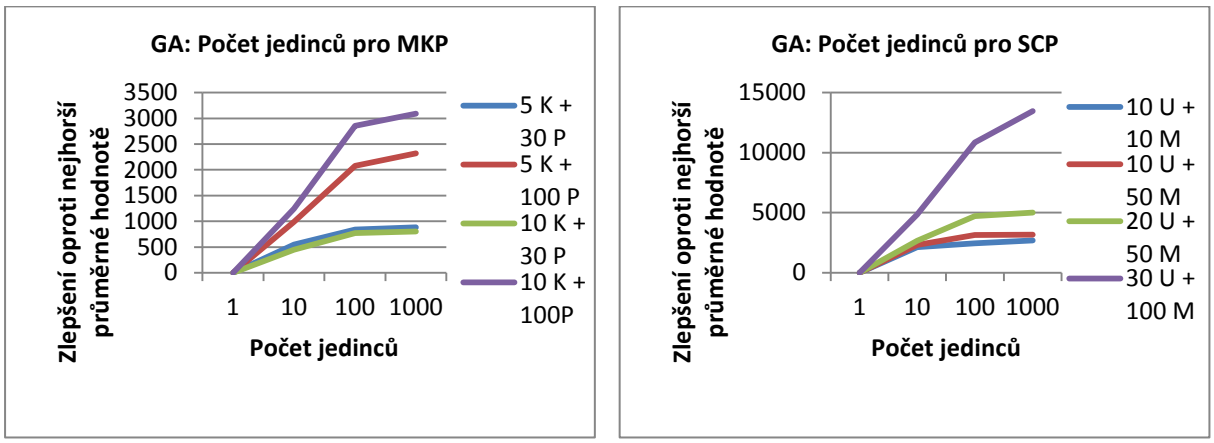
Grafy 77-80: Vliv počtu generací na GA.

6.3.2 Počet jedinců v populaci

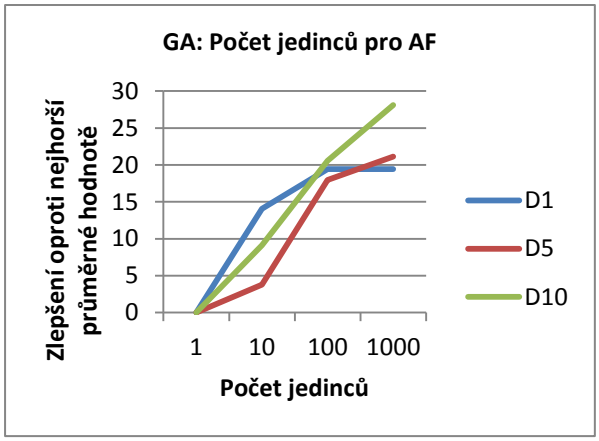
Počet jedinců v populaci velmi významně ovlivňuje kvalitu dosaženého řešení. Je-li jedinců příliš malý počet, metoda vykazuje i poměrně špatné výsledky. Naopak s větším počtem jedinců jsou dosahovány velmi kvalitní výsledky. Podobně jako u počtu iterací, zde lze též říci, že větší počet jedinců znamená vyšší šanci na kvalitní výsledek.

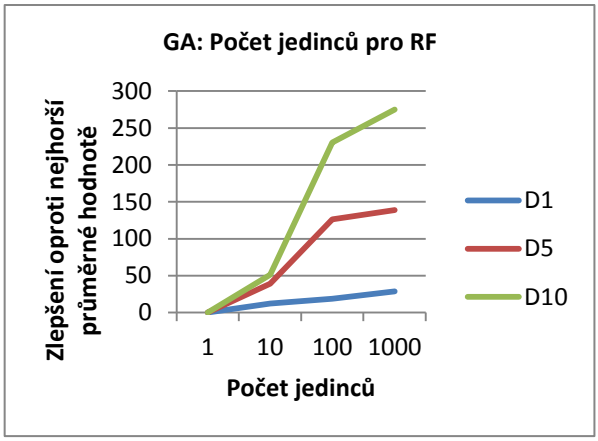
Pokud porovnáme závislosti kvality řešení na počtu jedinců a na počtu iterací, počet jedinců je dokonce významnější než počet iterací.

Za vhodný počet jedinců pro dané úlohy lze považovat hodnotu alespoň v řádu stovek jedinců.



Grafy 81-82: Vliv počtu jedinců v populaci na GA.





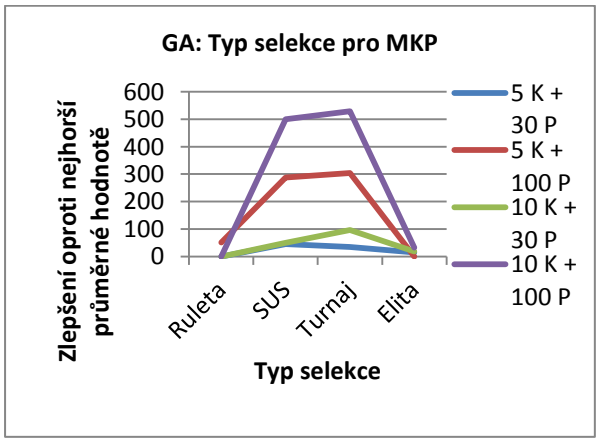
Grafy 83-84: Vliv počtu jedinců v populaci na GA.

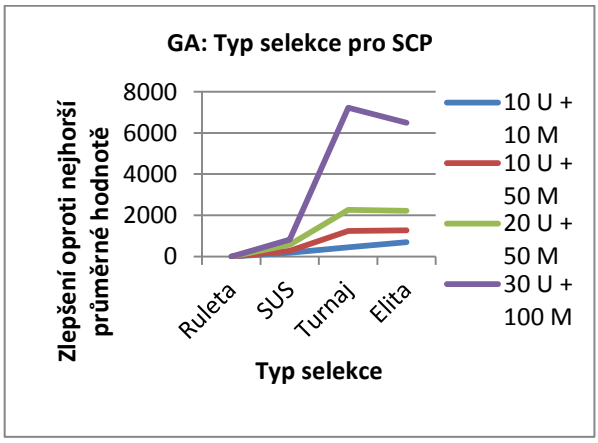
6.3.3 Typ selekce

Selekce vybírá jedince pro křížení, má proto potenciálně velmi velký vliv na dosaženou kvalitu řešení. U všech úloh lze vypozorovat přímou úměru mezi složitostí řešené úlohy a významem výběru selekce.

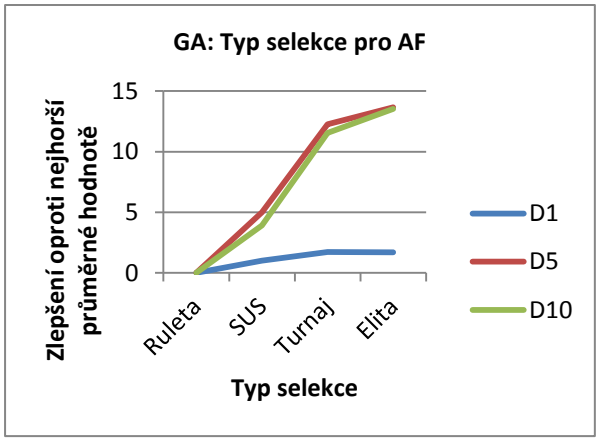
Nejlepších výsledků pro problém více batohů, pro problém pokrytí množiny a problém nalezení minima Rastriginovy funkce bylo dosahováno s použitím selekce typu turnaj. Podobně dobré výsledky vykazovala i elitistická selekce, která dosahovala nejlepších výsledků pro problém nalezení minima Ackleyho funkce.

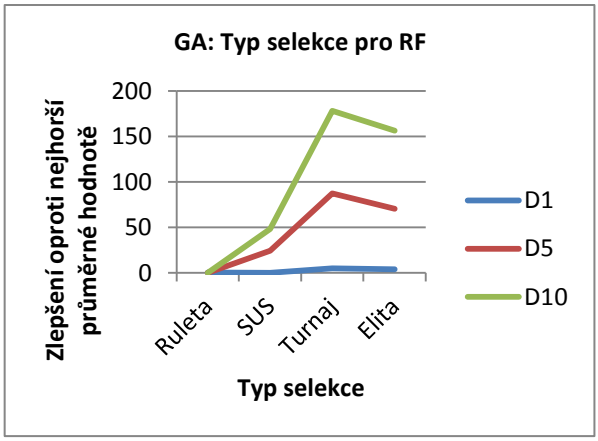
Pokud srovnáme výsledky uvedené v následujících grafech s výsledky pro typ obnovy a křížení, tak zjistíme, že volba typu selekce má mnohem vyšší dopad na kvalitu dosaženého řešení.





Grafy 85-86: Vliv volby selekce na GA.





Grafy 87-88: Vliv volby selekce na GA.

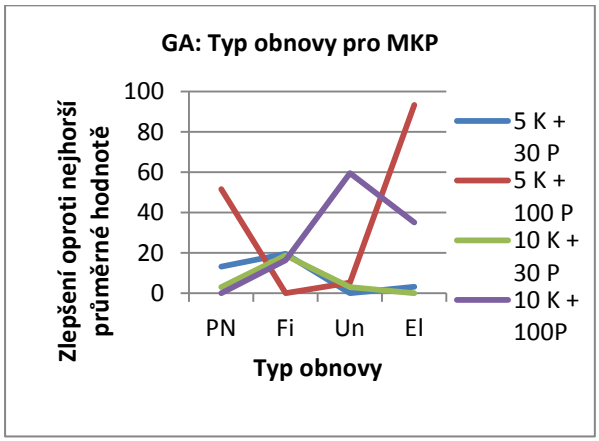
6.3.4 Typ obnovy

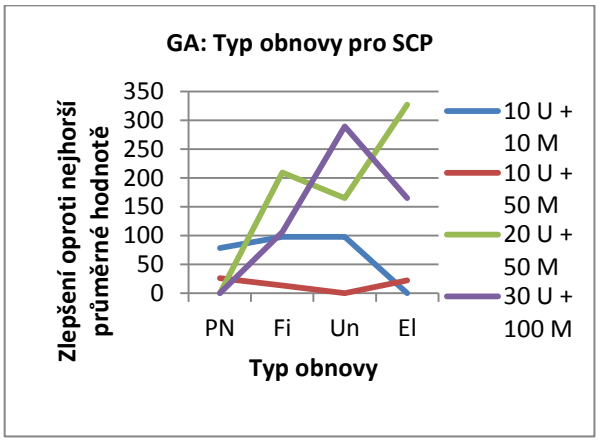
Obnova populace vytváří z původní populace a vytvořených potomků novou populaci. Proto je důležité volit vhodný typ obnovy pro zadanou úlohu. Výsledky ukázaly měřitelné rozdíly v kvalitě dosaženého řešení při použití různých typů obnovy, avšak pokud srovnáme výši těchto rozdílů s rozdíly při volbě typu selekce, tak nebyly natolik velké. Navíc se výsledky liší i pro jednotlivé instance úloh, je proto komplikované vyvodit závěr, který by jednoznačně určil nejlepší typ obnovy.

Snad jen pro hledání minima Rastriginovy funkce můžeme říci, že výsledky naznačují, že je pro ni vhodné použít obnovu na základě fitness případně elitistickou.

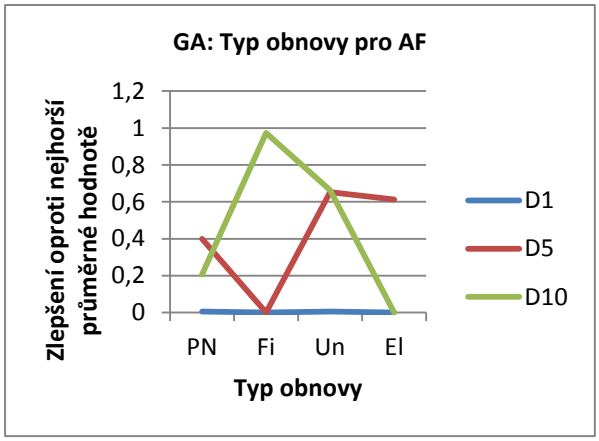
Pro každou úlohu a i pro každou instanci úlohy je tudíž vhodné vyzkoušet více typů obnovy populace.

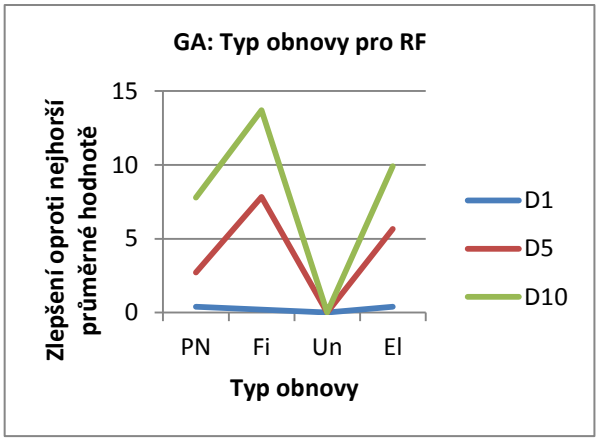
Poznamenejme, že v uvedených grafech je zavedeno označení PN, Fi, Un, El postupně pro plnou náhradu, obnovu na základě fitness, uniformní obnovu a konečně elitistickou.





Grafy 89-90: Vliv volby typu obnovy na GA.





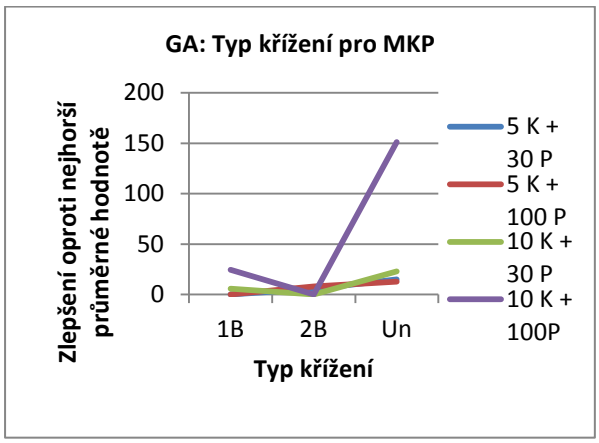
Grafy 91-92: Vliv volby typu obnovy na GA.

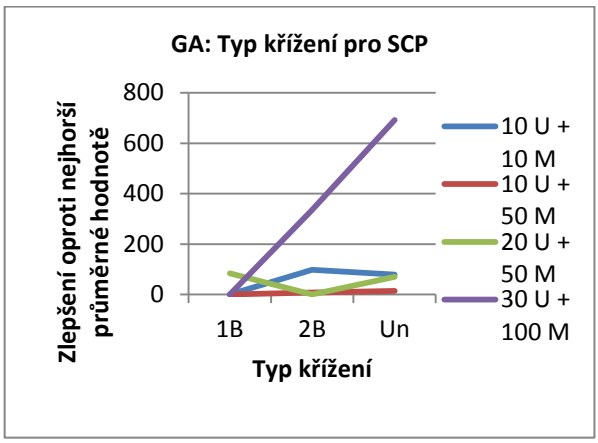
6.3.5 Typ křížení

V praxi se nejběžněji pro úlohy, které nejsou permutační, využívá tří typů křížení. Jedná se o jednobodové, dvoubodové a uniformní křížení (v grafech značené jako 1B, 2B,Un).

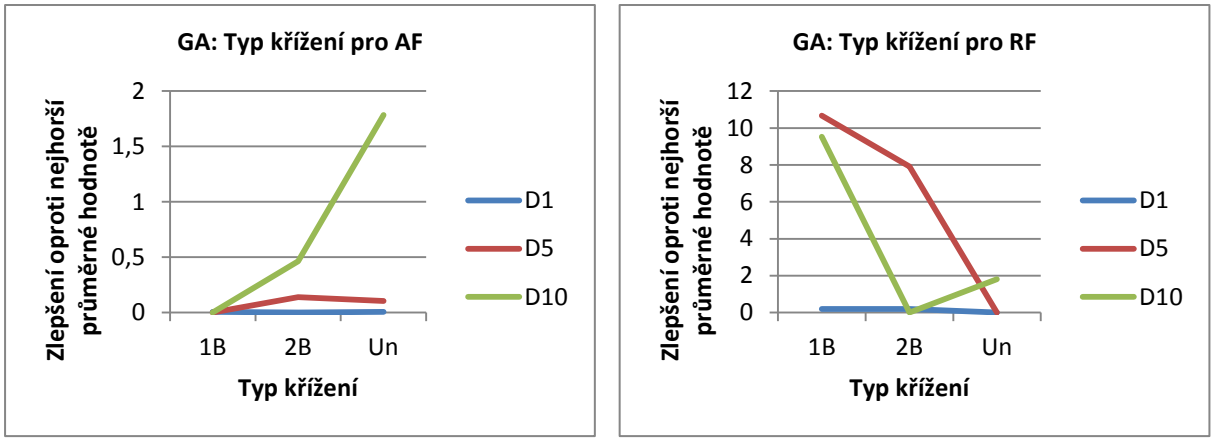
Při pohledu na grafy s výsledky lze vyvodit, že rozdíly v kvalitě dosažených výsledků nejsou extrémně velké (obzvláště pro instance s malou složitostí), přesto lze říci, že pro problém více batohů, problém pokrytí množiny a hledání minima Ackleyho funkce se nejvíce osvědčilo uniformní křížení a to obzvláště při složitějších instancích daných úloh. Pro hledání minima Rastriginovy funkce se pak nejlepších řešení dosahovalo při použití jednobodového křížení (znovu více patrné při pohledu na instance této úlohy s vyšší dimenzí).

Podobně jako u typu obnovy nelze vyvodit obecný závěr, který by jednoznačně vyzdvihoval jediný typ křížení. Přesto je možné nalézt nejlepší typ křížení pro danou úlohu.





Grafy 93-94: Vliv volby typu křížení na GA.



Grafy 95-96: Vliv volby typu křížení na GA.

6.3.6 Pravděpodobnost mutace

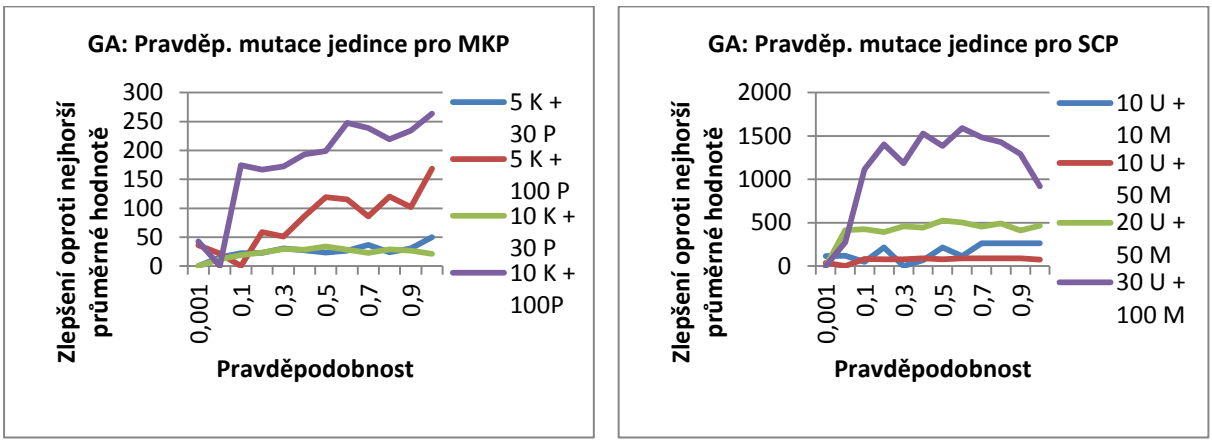
Zkoumání pravděpodobnosti mutace přineslo asi nejzajímavější zjištění. Standardně se nastavuje mutace s pravděpodobností kolem 0.001, ale při pohledu na uvedené grafy vidíme, že nejlepších výsledků bylo dosahováno s mnohem vyšší pravděpodobností mutace.

Začněme nejprve s variantou, že budeme uvažovat pravděpodobnost mutace jako pravděpodobnost mutace jedince, u něhož mutujeme jeden náhodně vybraný gen.

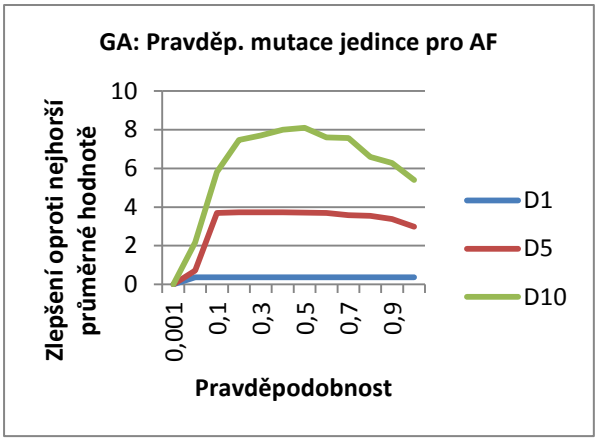
Pro problém více batohů se nejvíce osvědčilo nastavení pravděpodobnosti mutace dokonce na hodnoty blížící se 1. Avšak kvalitnějších výsledků je dosahováno často již od hranice 0.1-0.2. Ale hodnoty nižší jak 0.1 nevykazují až tak dobré výsledky.

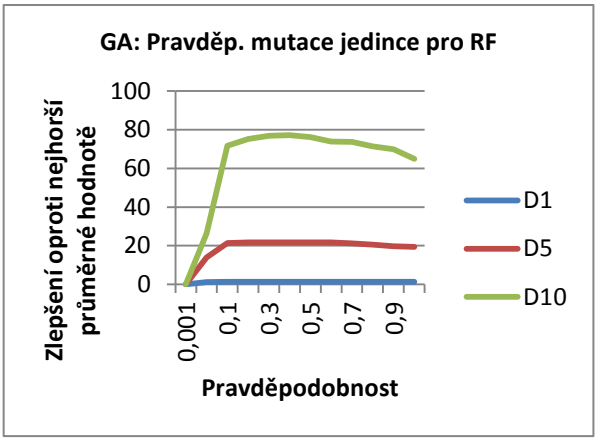
Problém pokrytí množiny také vykazuje lepší výsledky pro pravděpodobnost mutace vyšší než 0.1. Poté nejsložitější zkoumaná instance této úlohy vykazovala mírný sestup kvality řešení pro hodnoty vyšší než 0.8. V ostatních případech potom zůstávala kvalita dosahovaných řešení poměrně stálá až pro pravděpodobnost 1.

Pro úlohy hledání minima Ackleyho a Rastriginovy funkce pak také nejlepších výsledků dosahuje nastavení pravděpodobností mutace na hodnoty mezi 0.1-0.9. V tomto intervalu jsou dosahované výsledky poměrně konstantní, přičemž pokud bychom chtěli nalézt užší interval, tak bychom za něj prohlásili interval 0.3-0.5.



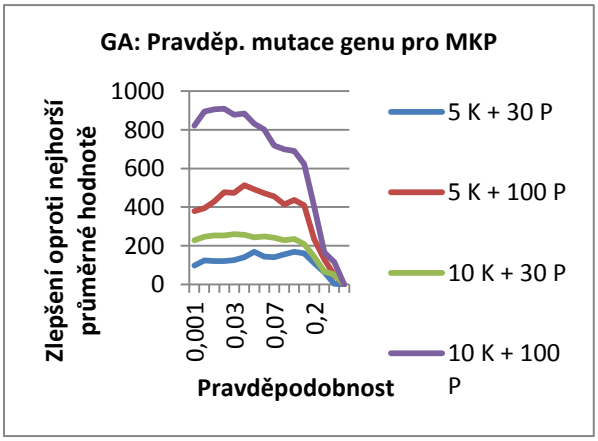
Grafy 97-98: Vliv pravděpodobnosti mutace jedince na GA.

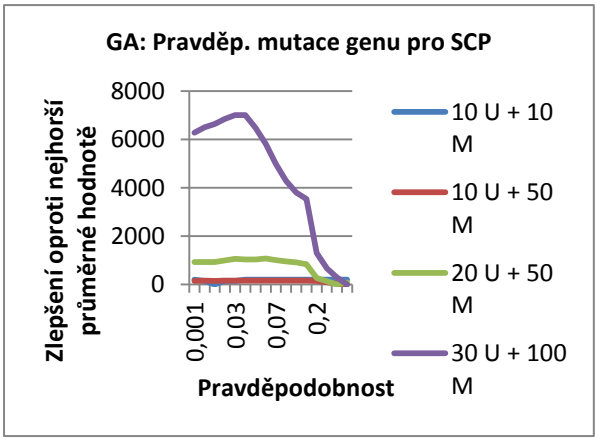


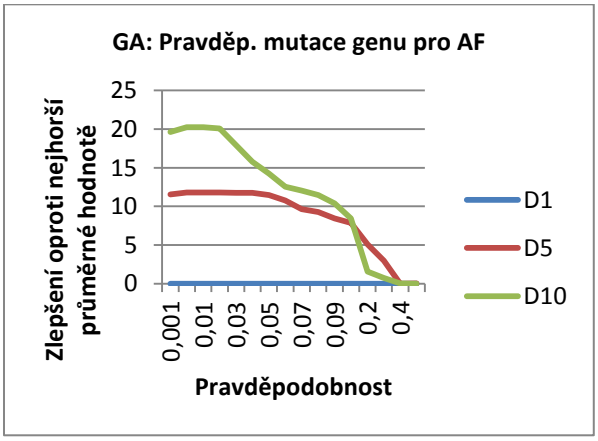


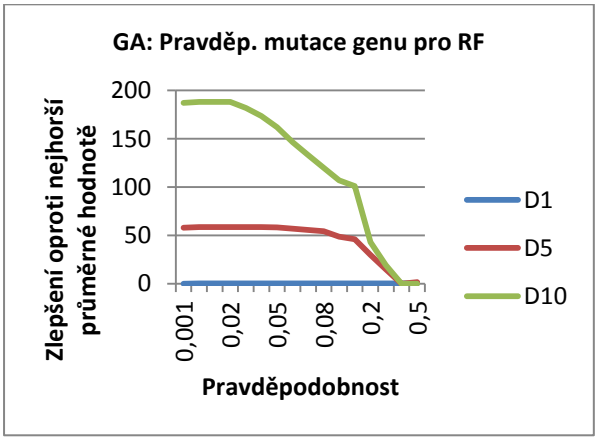
Grafy 99-100: Vliv pravděpodobnosti mutace jedince na GA.

Nyní si přibližme výsledky pro pravděpodobnost mutace genu, která je standardně uvažována. Vidíme, že i zde je vhodné volit vyšší hodnotu pravděpodobnosti. Pro diskrétní úlohy se jeví vhodná hodnota v řádu procent, přibližně 0.01 až 0.05. U spojitých úloh není třeba zvětšit mutaci tak výrazně, optimální je hodnota v rozmezí 0.005 až 0.02.









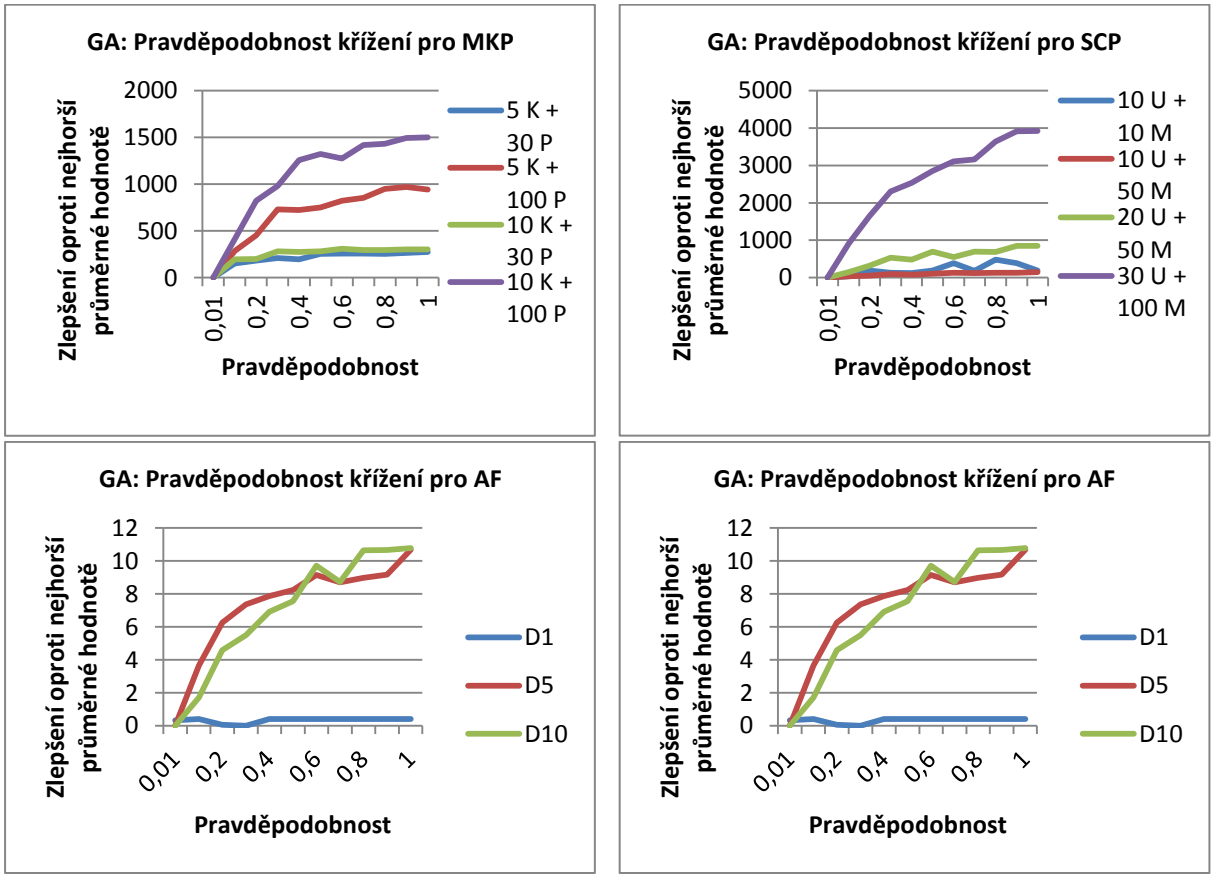
Grafy 101-104: Vliv pravděpodobnosti mutace genu na GA.

Pro zadané úlohy lze tedy vyvodit závěr, že je velmi vhodné volit větší (v případě pravděpodobnosti mutace jedince i velmi velké) pravděpodobnosti mutace.

6.3.7 Pravděpodobnost křížení

Pro pravděpodobnost křížení nebyly objeveny žádné překvapivé závěry. V praxi se často volí hodnoty kolem 0.7-1 a pro zvolené úlohy se také osvědčily. Pro zkoumané úlohy může být použito bez potíží i varianty, že křížení je prováděno vždy.

Naopak pokud bychom snížili pravděpodobnost křížení pod hodnotu přibližně 0.5, pak by nám velmi klesla kvalita dosažených řešení.



Grafy 105-108: Vliv pravděpodobnosti křížení na GA.

6.3.8 Počet generovaných potomků

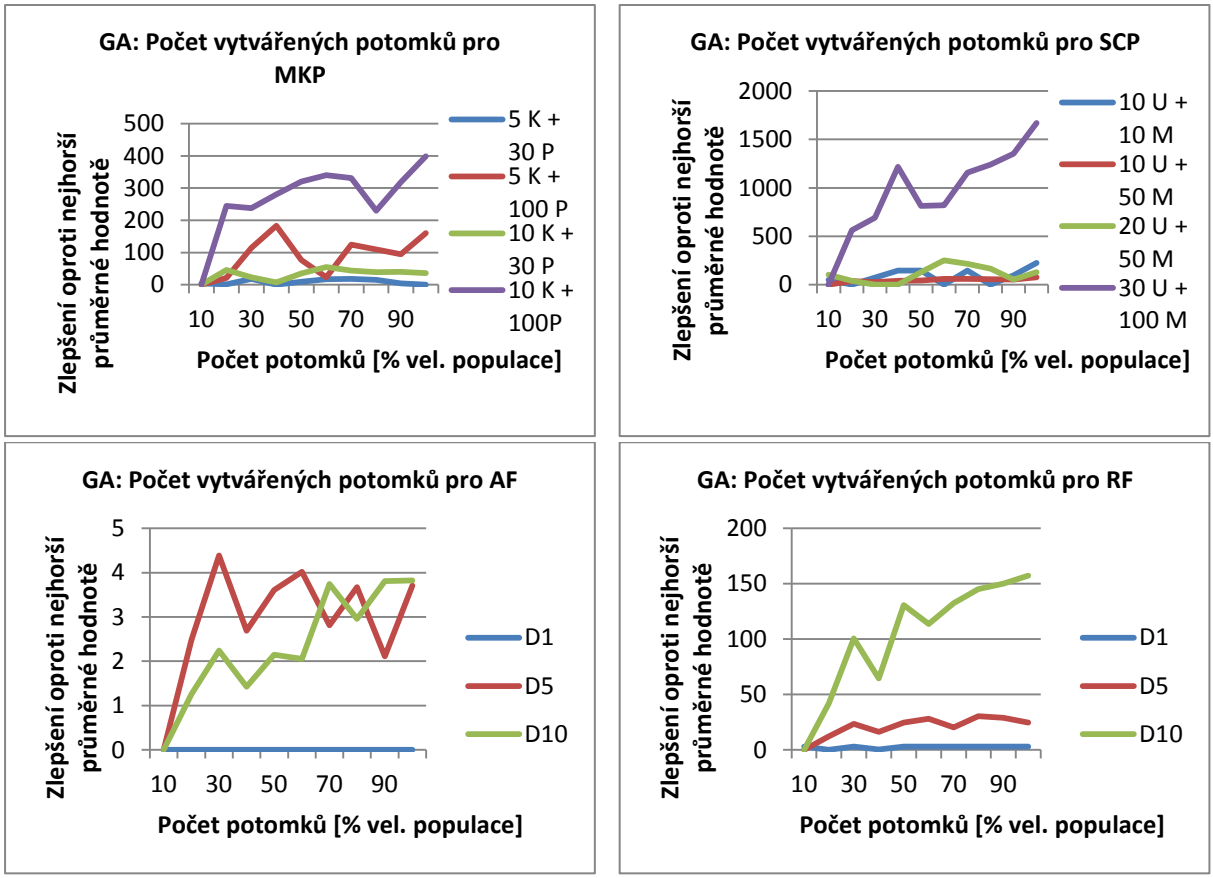
Pro genetické algoritmy lze například při volbě uniformní či elitistické obnovy populace generovat menší počet potomků, než je v původní generaci. Motivací pro snížení počtu potomků by mohlo být snížení časové náročnosti algoritmu.

Při pohledu na výsledné grafy si lze povšimnout toho, že čím je instance zvolené úlohy složitější, tím více potomků je nutné generovat pro kvalitnější výsledky.

Avšak při zachování iterací je pak pro jednodušší instance úloh možné volit počet potomků nižší než je velikost populace, přičemž kvalita řešení nijak výrazně nepoklesne. Odhaduji, že je to způsobeno tím, že počet iterací je pořád dostatečný na to, aby byl algoritmus schopen dosáhnout kvalitního řešení.

Uveďme, že pokud při provedených pokusech poklesl počet potomků pod 30-40%, tak již kvalita řešení prudce klesla.

Snížení počtu generovaných potomků tak může být jistou alternativou ke snižování počtu iterací, ale je nutno dodržet jistou mez.

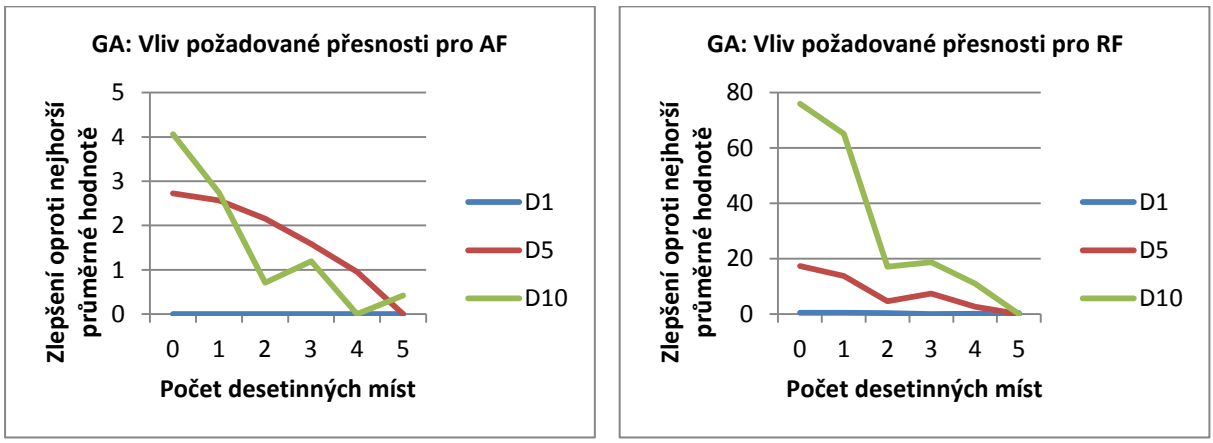


Grafy 109-112: Vliv různého počtu vytvářených potomků na GA.

6.3.9 Volba cílové přesnosti pro spojité úlohy

Jak jsme si již uvedli, tak byla pro spojité úlohy užita diskretizace. Je samozřejmé, že pokud budeme klást vyšší požadavky na přesnost, tak se nám prodlouží i délka chromozomu a poklesne i kvalita dosaženého řešení (pro stejný počet generací a jedinců). Pro zajímavost tedy ukažme, jak rychle klesá kvalita pro rostoucí počet desetinných míst.

Pro zadané instance úloh můžeme přirovnat pokles kvality k lineárnímu sestupu, ale grafy vykazují náznaky, že by pro volby velké dimenze mohla nastat situace, že se bude kvalita dosažených řešení až exponenciálně snižovat.



Grafy 113-114: Vliv různé požadované přesnosti na GA.

6.4 Simulované žíhání

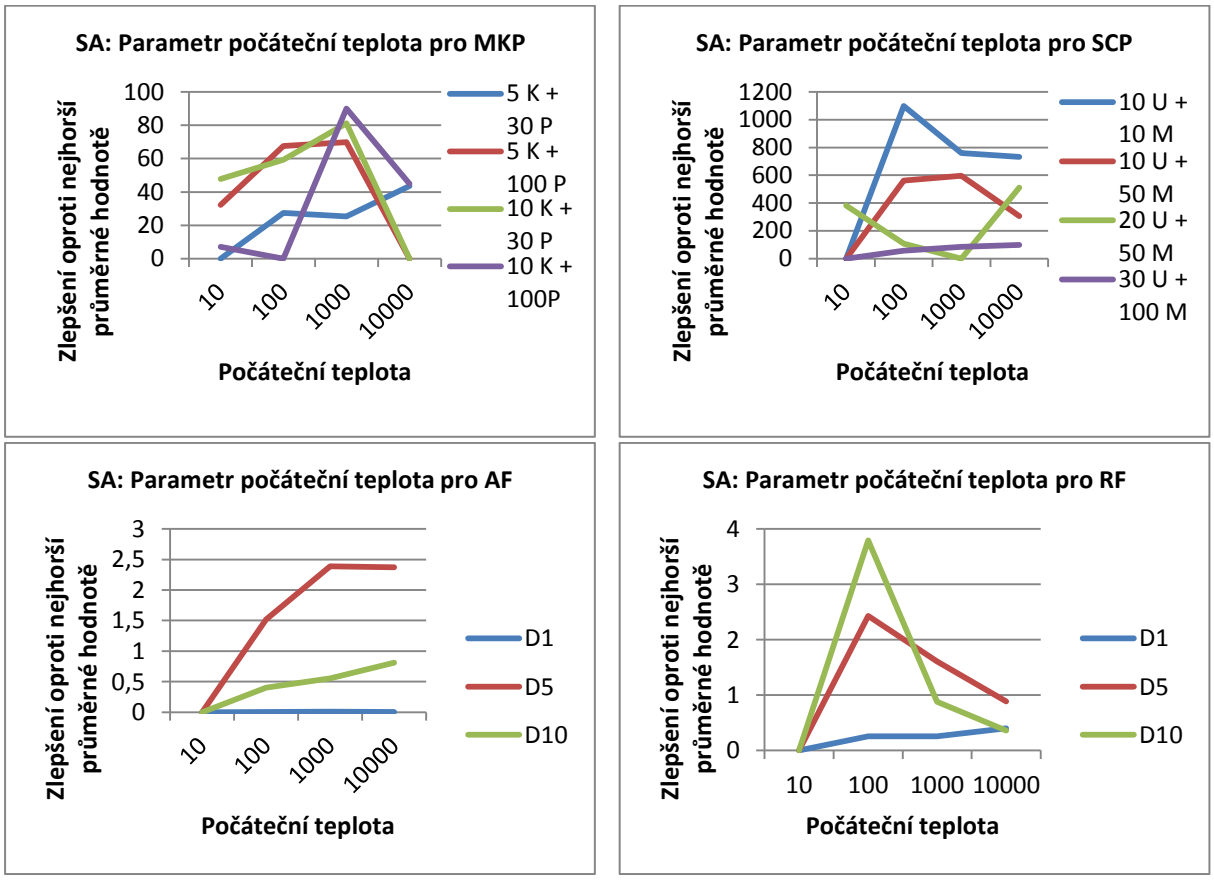
Jelikož je simulované žíhání poměrně nezávislé na řešené úloze a ukládá si pouze aktuální stav, nevyskytl se žádný problém s řešením ani diskrétních ani spojitých úloh. Simulované žíhání také poskytuje solidní výsledky pro všechny řešené úlohy.

6.4.1 Počáteční teplota

Počáteční teplota je jedním ze základních parametrů simulovaného žíhání. Připomeňme, že při vyšší teplotě algoritmus přijímá s větší pravděpodobností i horší stav, než je ten aktuální. Z tohoto důvodu obecně neplatí, že čím vyšší teplota, tím lepšího řešení je automaticky dosaženo.

Pro problém více batohů výsledky naznačují, že optimální teplota by se mohla pohybovat v řádu několika set až kolem jednoho tisíce. Podobně tomu je i u problému pokrytí množiny, avšak povšimněme si průběhu pro instanci 20 U + 50M, která ukazuje, že se mohou vyskytnout odchylky od tohoto chování.

Pro hledání minima Ackleyho funkce lze usoudit, že vhodná počáteční teplota by se mohla pohybovat v řádu tisíců. Naproti tomu hledání minima Rastriginovy funkce nejspíše vyžaduje nastavení počáteční teploty na hodnotu kolem 100.



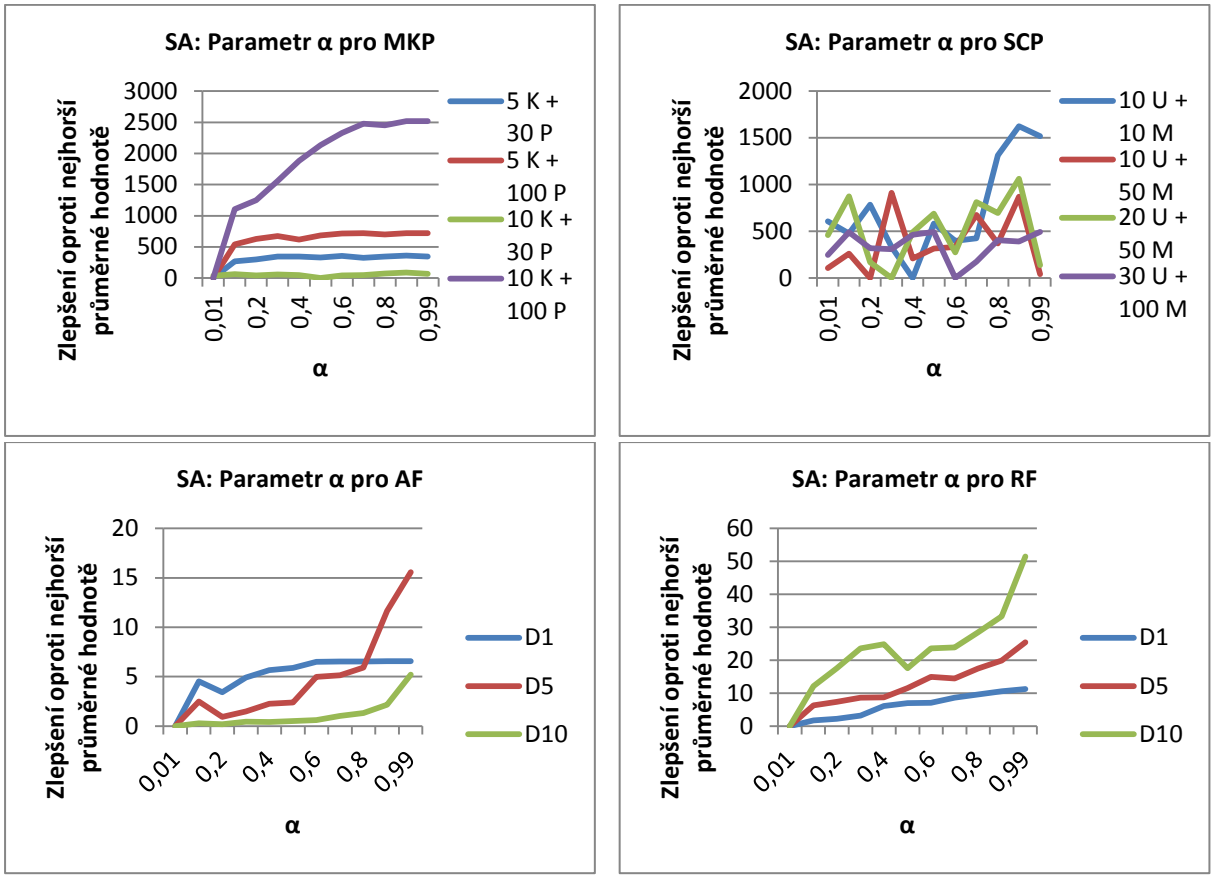
Grafy 115-118: Vliv počáteční teploty na SA.

6.4.2 Koeficient klesání teploty α

Koeficient α nám určuje, jak rychle klesá teplota. Lze na to nahlížet ovšem i z jiného pohledu: tento koeficient určuje počet iterací algoritmu. Dosažené výsledky ukazují, že volba nastavení parametru α má dokonce větší dopad, než volba počáteční teploty.

Standardně se volí hodnota kolem 0.9-0.95. Pro problém více batohů a hledání minima Ackleyho a Rastriginovy funkce lze s tímto nastavením souhlasit. Příslušné grafy totiž jasně naznačují, že čím je hodnota vyšší, tím kvalitnějšího řešení je dosaženo.

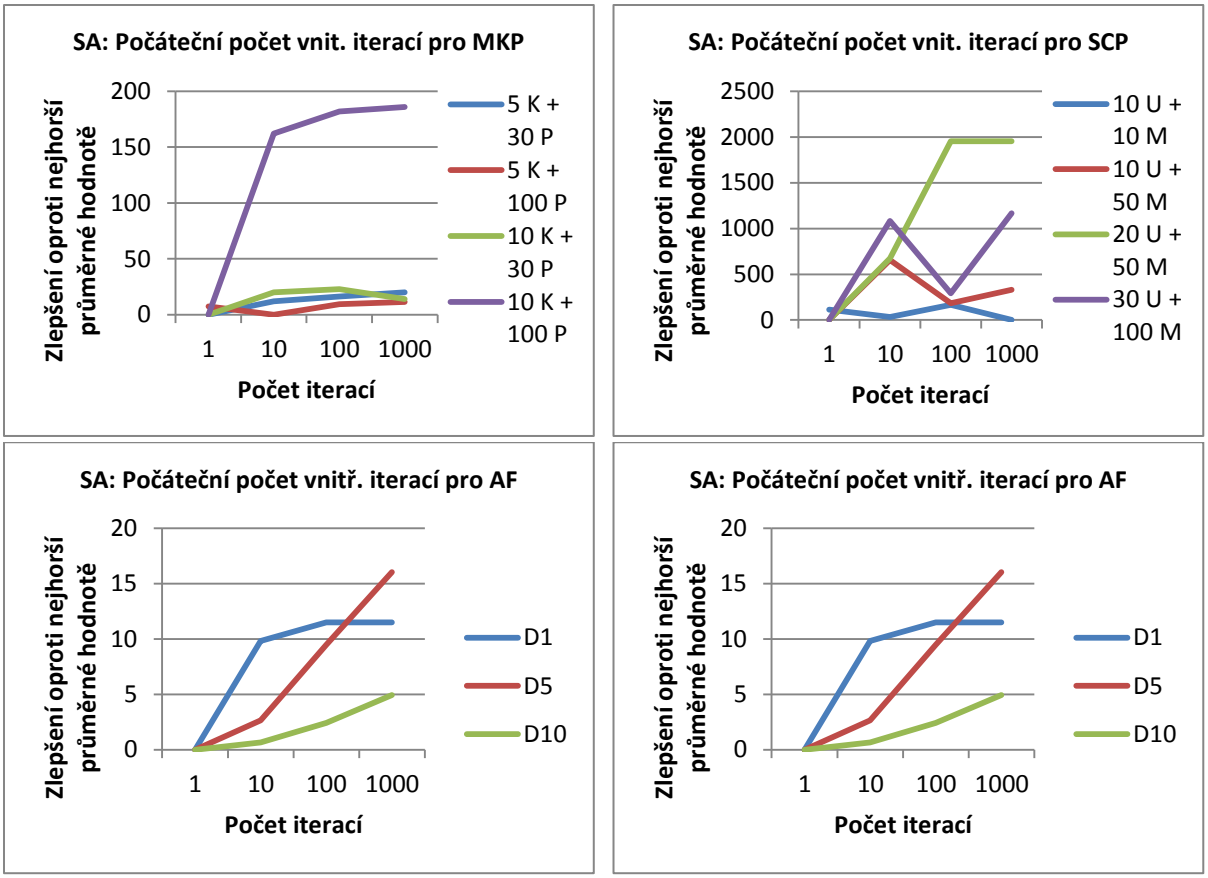
U problému pokrytí množiny se nepodařilo jasně určit nejlepší hodnotu. Kvalita řešení se pro jednotlivé instance této úlohy nepředvídatelně mění. Ovšem výsledky ukázaly, že i pro standardní interval 0.9-0.95 je stále dosahováno poměrně slušných hodnot.



Grafy 119-122: Vliv parametru α na SA.

6.4.3 Počáteční počet vnitřních iterací

Můžeme samozřejmě říci, že čím vyšší bude počet vnitřních iterací, tím větší bude pravděpodobnost nalezení lepšího řešení. Je si ale třeba uvědomit, že pokud budeme volit následující zkoumaný parametr koeficient růstu β větší než jedna, pak se tento počet v průběhu algoritmu mnohonásobně zvýší. Proto můžeme říci, že je vhodné začít s hodnotou 10, která i vzhledem k naměřeným hodnotám dosahuje solidních výsledků, a pokud nám to čas dovolí, tak tuto hodnotu zvýšit.



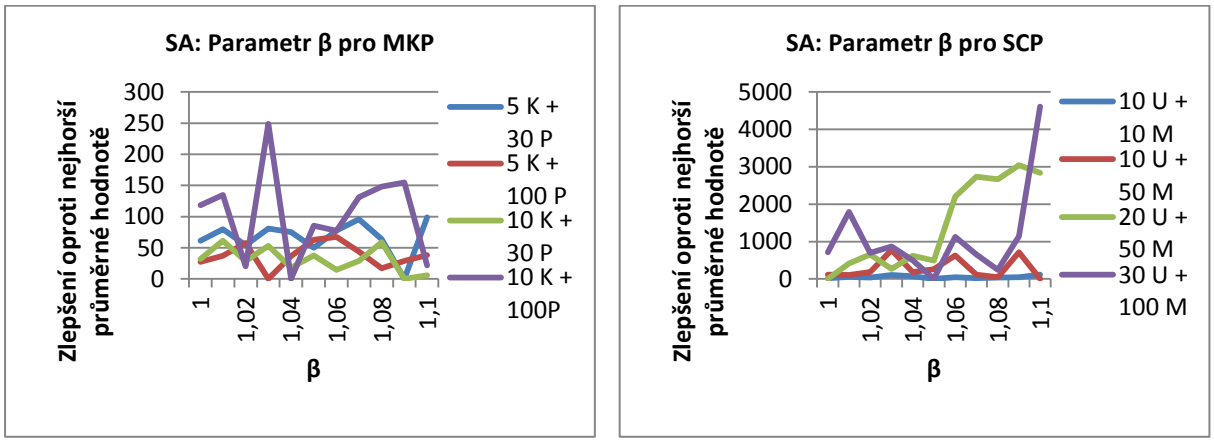
Grafy 123-126: Vliv počátečního počtu vnitřních iterací SA.

6.4.4 Koeficient růstu počtu vnitřních iterací β

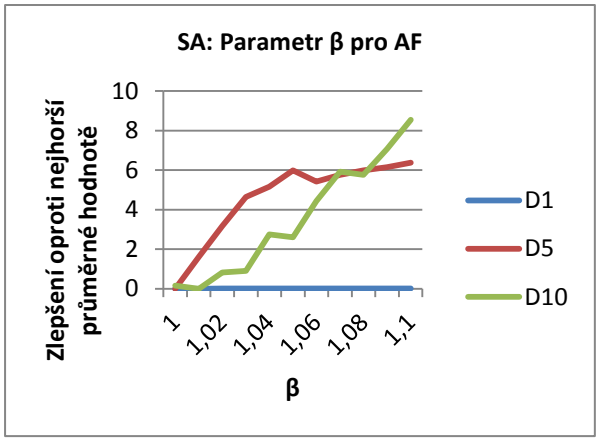
Koeficient β ovlivňuje růst vnitřních iterací. To znamená, že provedeme více iterací algoritmu se stejnou teplotou. Zvýšeným počtem iterací chceme dosáhnout lepšího výsledku.

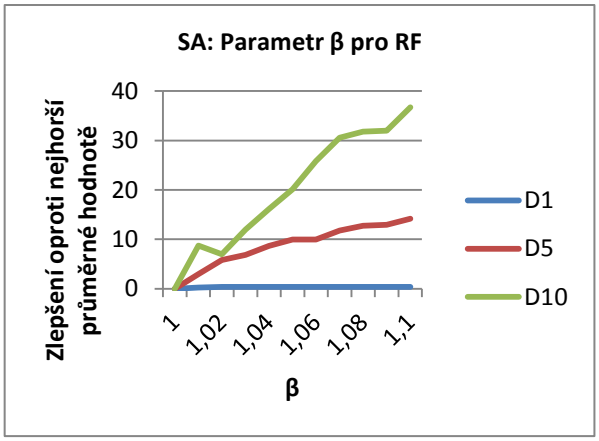
Výsledky pro spojité úlohy jasně ukazují téměř přímou úměru mezi zvýšením počtu iterací a kvalitou dosaženého řešení.

Naopak pro diskrétní problémy výsledky sice ukazují, že je možné dosáhnout lepšího řešení při větším počtu vnitřních iterací algoritmu, ale není možné odhadnout přesnější hodnotu pro koeficient β. Můžeme si povšimnout, že u problému pokrytí množiny se nám pozitivní trend pro zvyšující se vnitřní počet cyklů algoritmu projevil až pro složitější instance této úlohy.



Grafy 127-128: Vliv parametru β na SA.



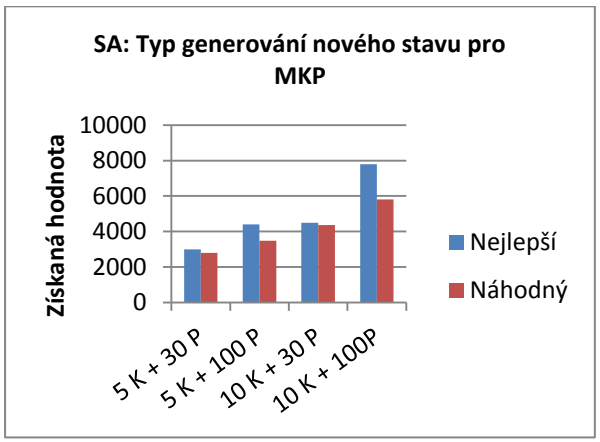


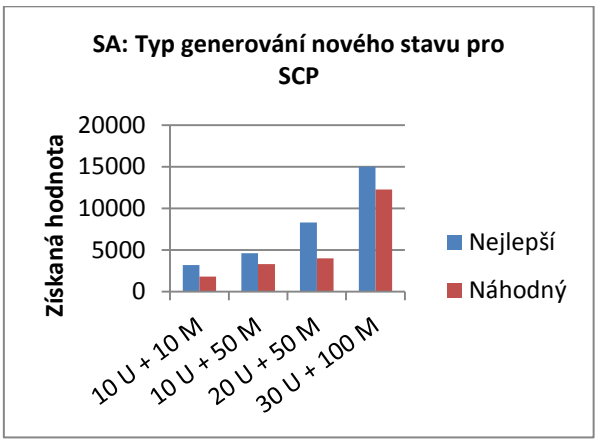
Grafy 129-130: Vliv parametru β na SA.

6.4.5 Typ generování nového stavu pro diskrétní úlohy

Pro zvolené diskrétní úlohy se nabízí dva přístupy generování kandidátního stavu. Můžeme ze všech možných následujících stavů vybrat ten nejlepší z hlediska fitness funkce nebo můžeme zvolit náhodně jeden z nich.

Výsledky ukazují, že pro problém více batohů se více vyplatí vybírat vždy nejlepší následující kandidátní stav. Naopak pro minimalizační problém pokrytí množiny graf jasně vykazuje kvalitnější řešení při použití náhodného stavu z množiny všech možných stavů.



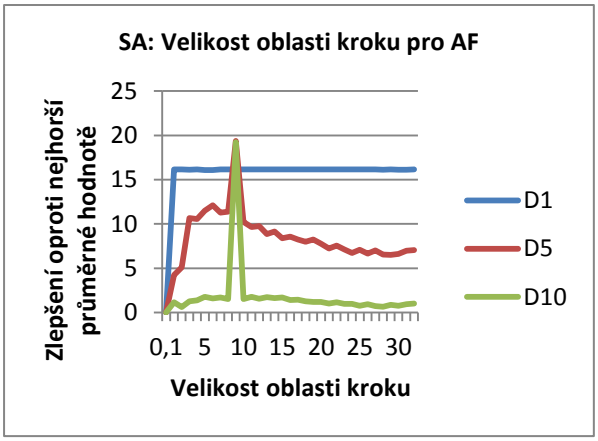


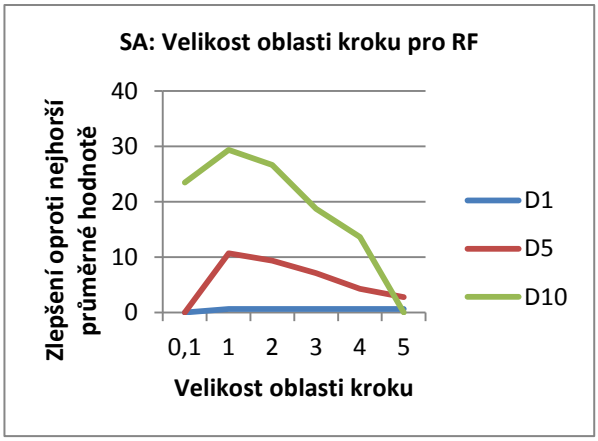
Grafy 131-132: Vliv volby typu generování nového stavu na SA.

6.4.6 Velikost oblasti kroku pro spojité úlohy

Při simulovaném žíhání vybíráme jako další stav bod v prostoru řešení, které je omezeno na určitou oblast. Jako tuto oblast jsem volil hyperkostku se zvolenou délkou její hrany. A právě nastavení délky této hrany je schopno velmi ovlivnit dosahovanou kvalitu výsledků. Je naprosto logické, že při volbě velikosti této hrany musíme přihlédnout k celkové velikosti prohledávaného prostoru.

Pro hledání minima Ackleyho funkce se ukázalo být nejlepší nastavení délky hrany na hodnotu cca 9. Naopak pro hledání minima Rastriginovy funkce graf ukazuje, že je nejvhodnější volit hodnotu kolem 1.



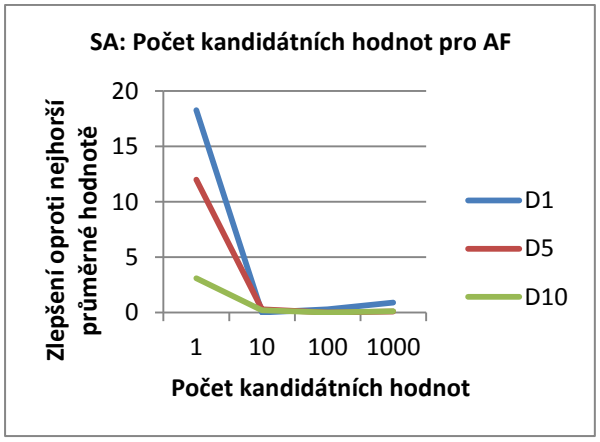


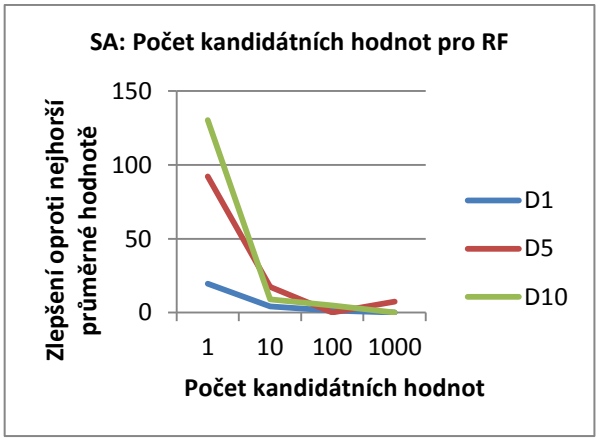
Grafy 133-134: Vliv volby velikosti kroku spojité úlohy na SA.

6.4.7 Počet kandidátních hodnot pro spojité úlohy

V tomto měření jsem se pokoušel najít optimální počet kandidátních hodnot, ze kterých je vybrána jedna nejlepší, která je brána jako nabízený další stav pro algoritmus simulovaného žíhání. Ačkoliv jsem předpokládal, že nárůst kandidátních hodnot by mohl přinést zvýšení kvality dosahovaného řešení, tak tomu bylo právě naopak.

Jednoznačně nejlepších výsledků je dosaženo, pokud vygeneruji jen jedno náhodné řešení a to je hned předloženo k posouzení algoritmem, zda bude přijato za nový stav. Předpokládám, že je to způsobeno tím, že když vygeneruji více kandidátních hodnot a z nich vyberu nejlepší, algoritmus bude mít více snahu tíhnout k lokálním extrémům, z nichž se pak nebude již schopen dostat.





Grafy 135-136: Vliv počtu kandidátních hodnot na SA.

6.5 Posilované učení

U optimalizace založené na posilovaném učení budeme měření dělit vždy do dvou skupin a to měření pro Q-learning (značíme Q-RL) a TD-learning (značíme TD-RL).

V kapitole 5 jsme si řekli, že budeme uvažovat dvě implementace stavů, které budeme hodnotit. Pracovně jsme je nazvali absolutní a relativní. Pro diskrétní úlohy připadá pro praxi v úvahu pouze tzv. relativní implementace stavů. Je to způsobeno tím, že kdybychom chtěli ukládat a procházet všechny možné stavy reprezentující částečná řešení, tak bychom toho nebyli schopni pro složitější instance dosáhnout. Relativní stavy dosahují konstantního (pro danou instanci úlohy)

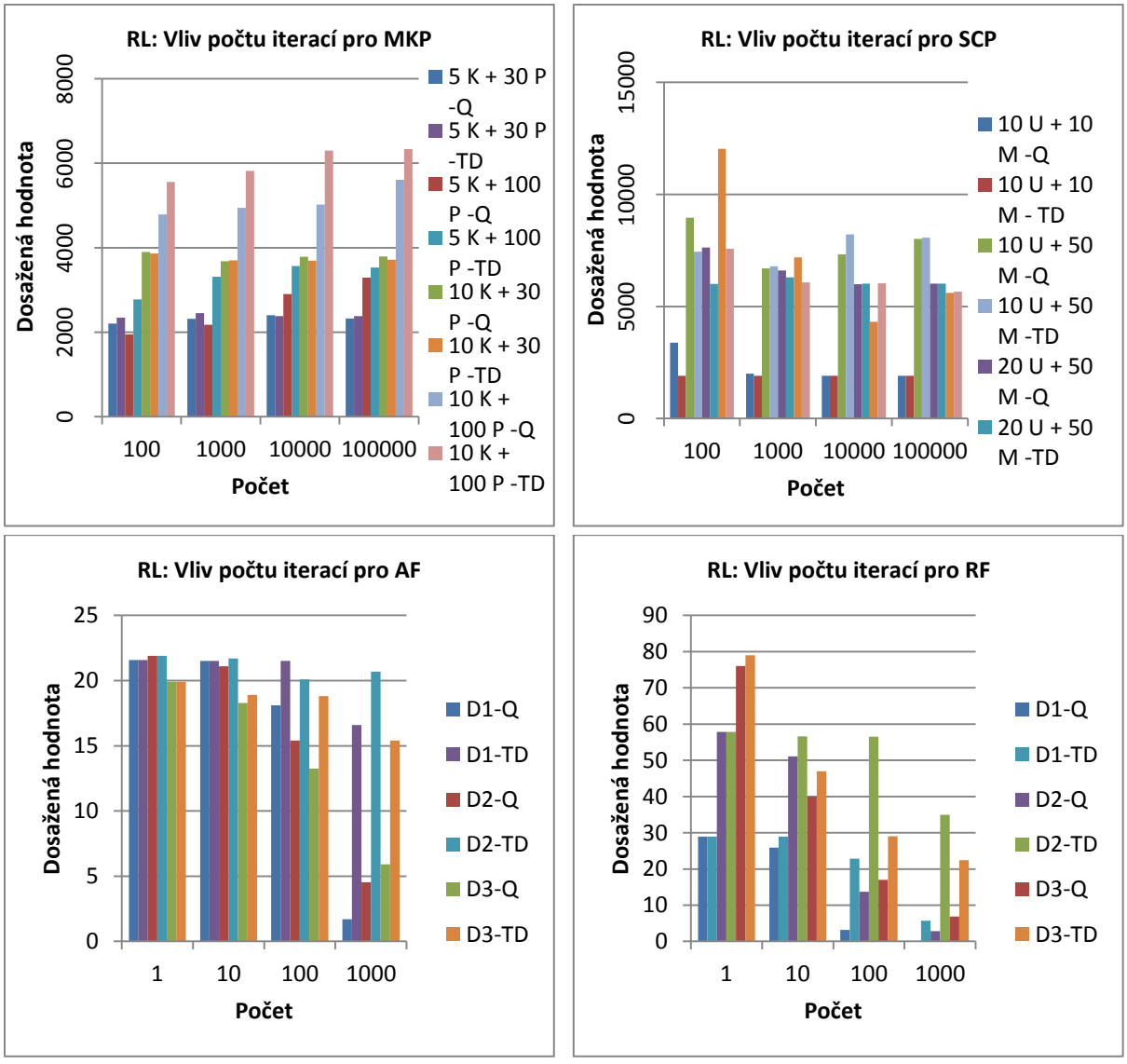
a poměrně malého počtu a i přes absenci informace o přesné návaznosti stavů jsou s relativními stavy dosahovány přijatelné výsledky. Proto uvažujeme pro diskrétní úlohy jen tuto reprezentaci.

U spojitých úloh je problém s ukládáním částečných řešení ještě větší. Pro absolutní stavy je obtížné uložit a procházet takové množství stavů. Avšak i pro relativní stavy nebylo možné dosáhnout přijatelných řešení ani při extrémně velkém počtu iterací (řešení nekonvergovala). Jelikož tedy relativní stavy nelze použít, uvedl jsem alespoň výsledky pro spojité úlohy s absolutními stavy s malými dimenzemi.

6.5.1 Počet iterací

Je logické, že pro použití posilovaného učení pro optimalizační úlohy je nutné užít mnohem vyššího počtu iterací, než je tomu u ostatních zmiňovaných metod. Z naměřených hodnot je vidět, že u zkoumaných úloh bylo třeba vykonat alespoň tisíce iterací.

Pro diskrétní úlohy rychleji konverguje spíše TD-learning varianta, naopak pro spojité úlohy Q-learning varianta. Při vyšším počtu iterací se kvalita dosažených řešení pro obě varianty u diskrétních úloh srovnává. U některých instancí se dokonce objevilo přeučení a pro velké počty iterací se kvalita spíše zhoršuje.



Grafy 137-140: Vliv počtu iterací na RL.

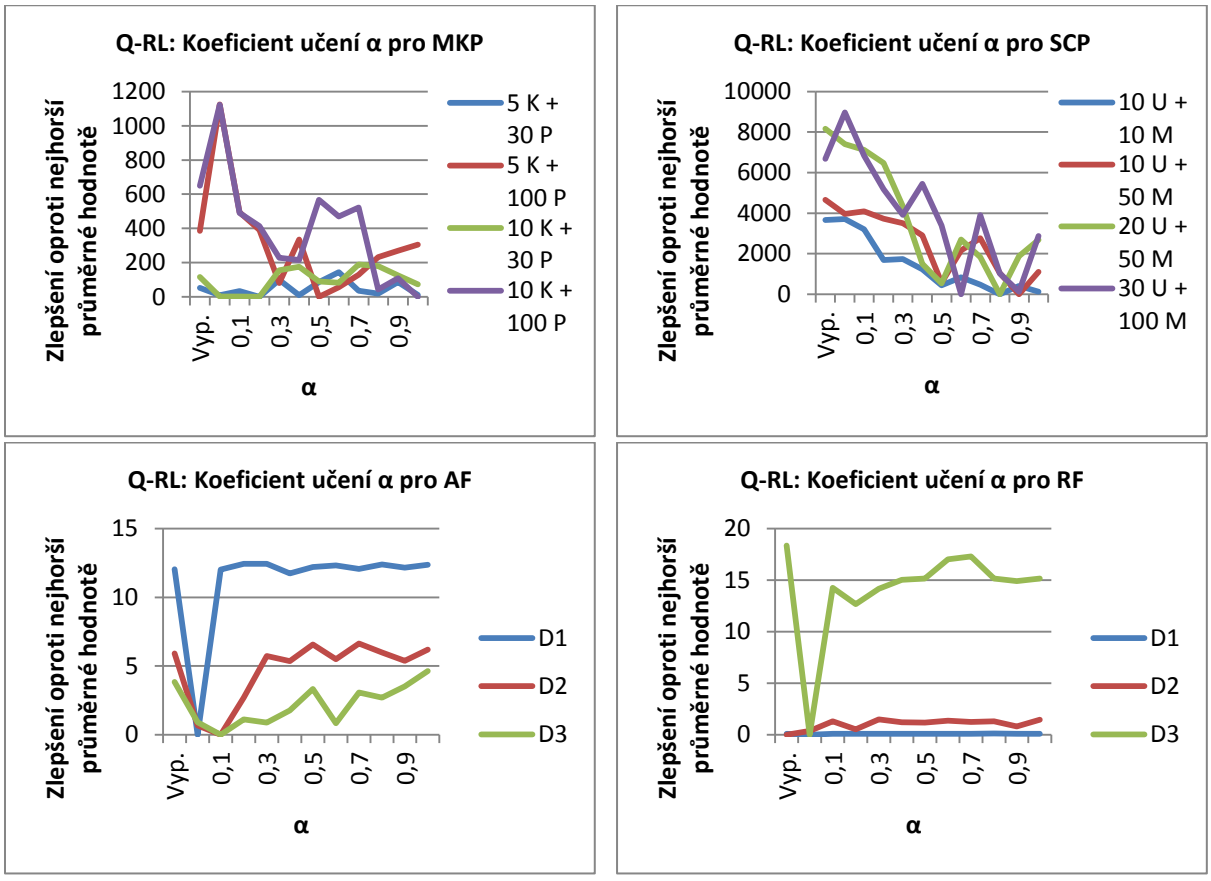
6.5.2 Koeficient učení

Koeficient učení α vyjadřuje, jak rychle (jak moc) hodláme měnit ohodnocení stavu při každé aktualizaci jeho hodnoty.

Při měření jsem uvažoval konstantní hodnoty koeficientu učení a vypočtenou hodnotu, která klesá s počtem návštěv daného stavu (v grafu značená jako Vyp.).

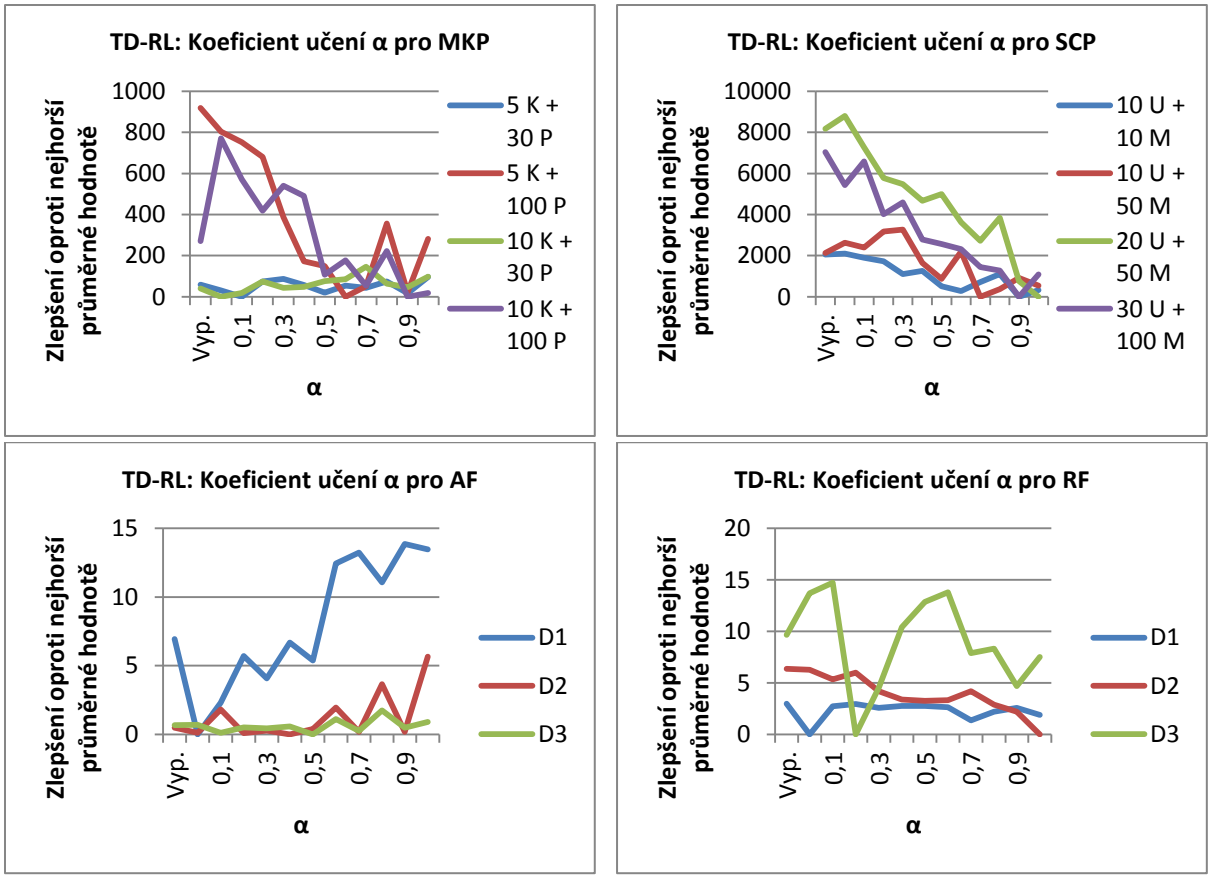
Pro Q-learning platí, že ačkoliv vypočtený koeficient ne vždy poskytuje zcela nejlepší řešení, většinou jej lze s úspěchem aplikovat. U spojitých úloh pak platí, že pokud použijeme konstantní hodnotu, měla by být alespoň 0.1 či vyšší.

Problém pokrytí množiny na druhou stranu vykazuje spíše klesající charakter se zvyšující se hodnotou koeficientu učení. Totéž platí pro problém více batohů s větším množstvím zkoumaných objektů, ale pro instance této úlohy s nižším počtem objektů nelze určit vhodnou hodnotu, lze se jen dohadovat, že by mohla ležet někde v intervalu 0.3 až 0.9.



Grafy 141-144: Vliv koeficientu učení α na Q-learning.

Pro TD-learning aplikovaný na diskrétní úlohy platí totéž jako pro Q-learning. Pro úlohu nalezení minima Ackleyho funkce vykazuje graf spíše rostoucí trend pro rostoucí koeficient učení. U Rastriginovy funkce se nepodařilo zjistit nějaký společný trend pro instance této úlohy.



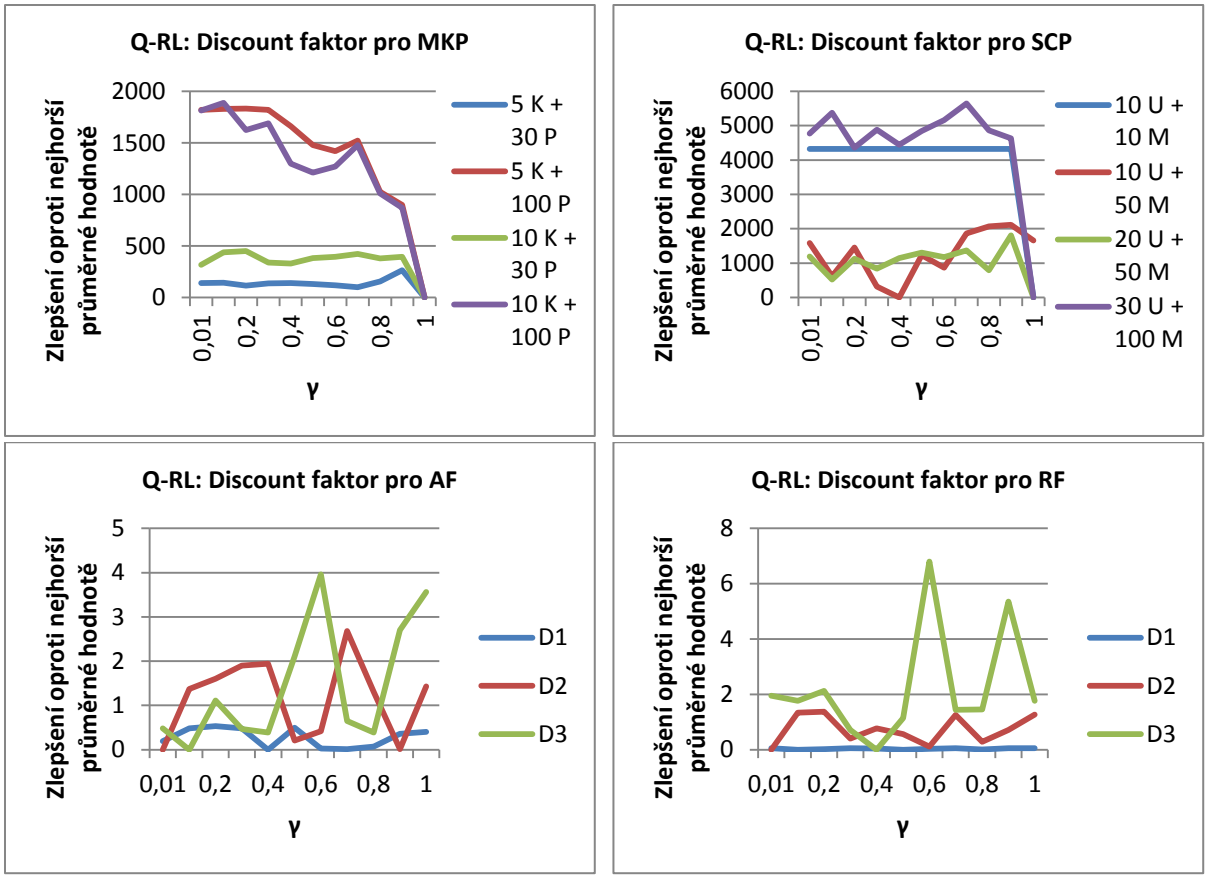
Grafy 145-148: Vliv koeficientu učení α na TD-learning.

6.5.3 Discount faktor

Discount faktor γ určuje vliv ohodnocení následujícího stavu. Z tohoto pohledu se jedná o parametr, který je schopen velmi ovlivnit výsledek.

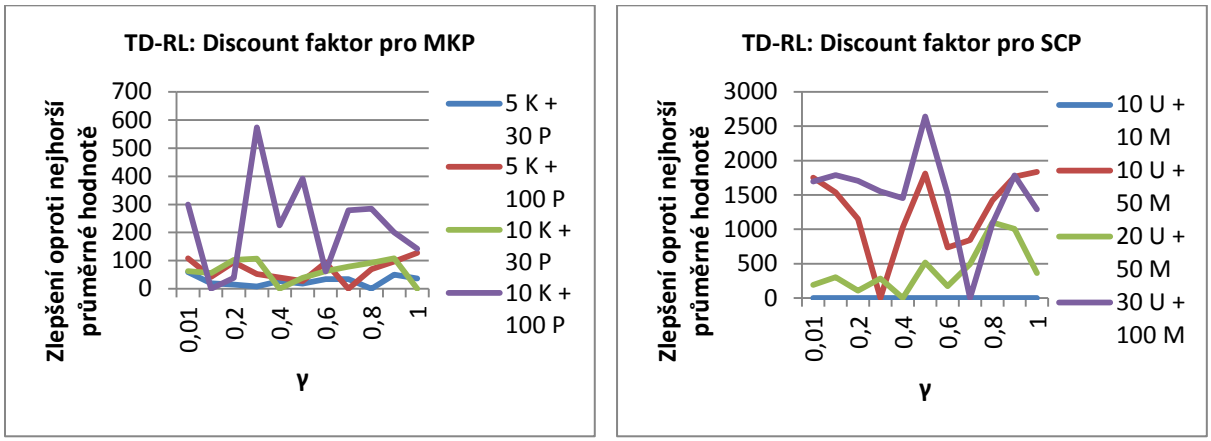
Začněme s variantou Q-learningu, kde pro spojité úlohy lze jen těžko z naměřených výsledků odvodit konkrétní hodnotu pro výsledek. I když byly zkoumány pouze malé dimenze, průběhy grafů jsou pro každou dimenzi rozdílné.

U diskrétních úloh jsem zaznamenal zajímavý pokles kvality výsledků pro nastavení na hodnotu 1. U většiny instancí úloh se kvalita dosaženého řešení významně neměnila pro různá nastavení tohoto parametru (krom uvedené hodnoty 1), přičemž se dobrou volbou zdála být hodnota kolem 0.8 či 0.9. U problému více batohů se pro instance s větším množstvím nabízených objektů kvalita řešení snižovala s rostoucí hodnotou discount faktoru.

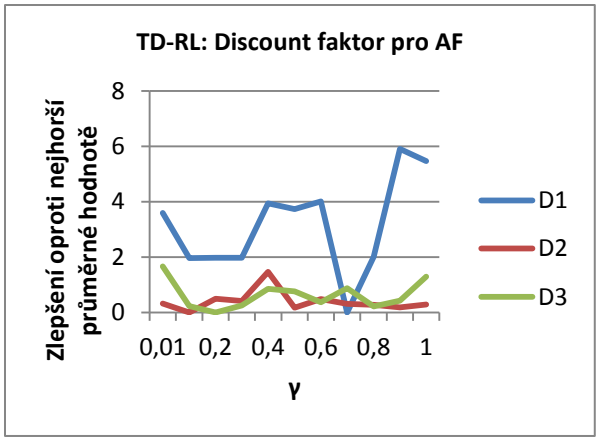


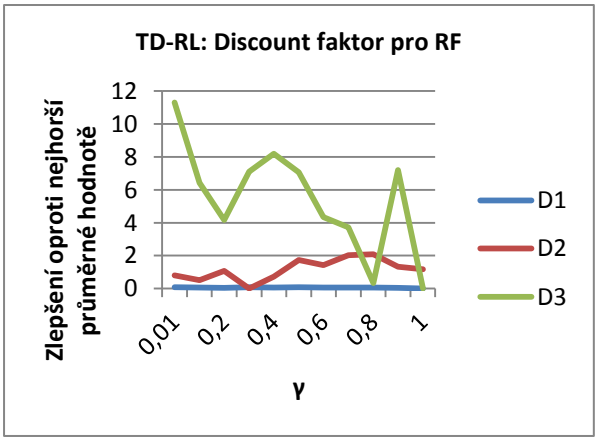
Grafy 149-152: Vliv discount faktoru γ na Q-learning.

Pro TD-learning je ještě obtížnější vyvodit nějaký závěr s konkrétní hodnotou pro discount faktor. Lze si však všimnout, že u každé z úloh se vyskytuje minimálně jedna měřená instance, která poskytuje nejlepší výsledky s nastavením discount faktoru na hodnoty kolem 0.9, 0.01 nebo přibližně 0.5, což je velmi zajímavé, ale nelze z toho vyvodit žádný závěr vzhledem k ostatním instancím.



Grafy 153-154: Vliv discount faktoru γ na TD-learning.



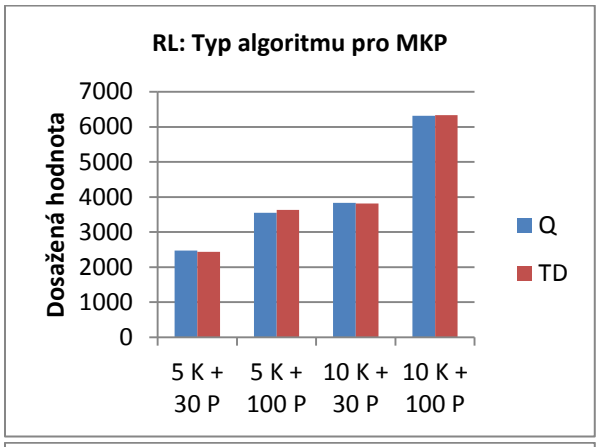


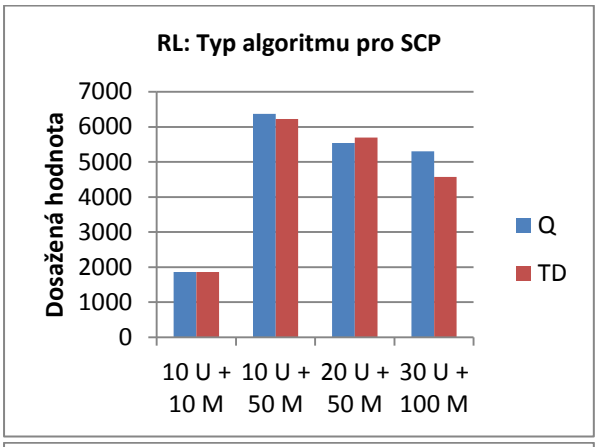
Grafy 155-156: Vliv discount faktoru γ na TD-learning.

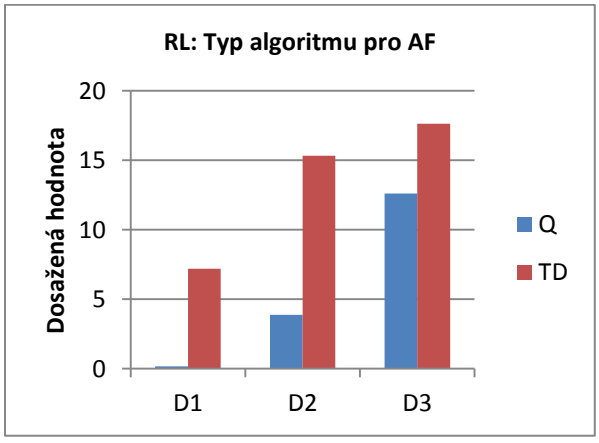
6.5.4 Typ posilovaného učení

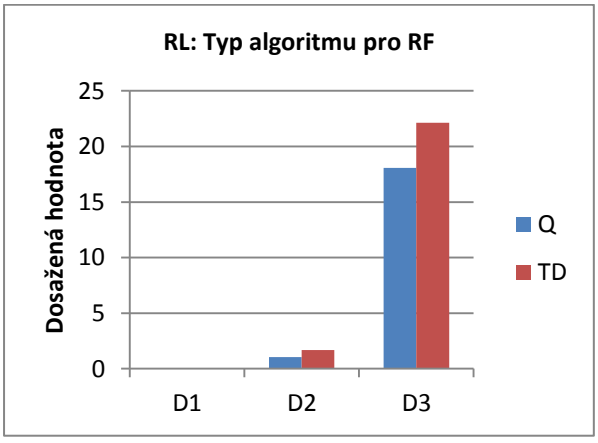
Poměrně zásadní otázkou je, jaký typ posilovaného učení vybrat pro konkrétní úlohu. Pro diskrétní úlohy platí, že pro oba typy posilovaného učení, Q-learning i TD-learning, lze nalézt nastavení parametrů, při nichž dosahují přibližně stejně kvalitního nejlepšího řešení při stejném počtu iterací.

Pro spojité úlohy se jasně vyplácí použití typu Q-learning, jelikož pro dané úlohy konverguje ke správným řešením mnohem rychleji než TD-learning.









Grafy 157-160: Vliv volby typu RL.

7 Závěr

Tato práce se zabývala optimalizačními metodami, obzvláště pak jejich vztahem k optimalizačním úlohám. V úvodu jsme kromě popisu jednotlivých kapitol uvedli příklady ze skutečného světa, kterými byly optimalizace portfolia a device sizing, jež ilustrovaly použití optimalizačních metod.

Dále jsme formálně definovali, co chápeme pod pojmem optimalizační problém či optimalizační úloha, což bylo vzhledem k zaměření práce naprosto esenciální.

V následující kapitole bylo uvedeno a rozebráno pět použitých přístupů k řešení optimalizačních úloh. Konkrétně se jednalo o optimalizaci pomocí hejna částic, mravenčí kolonií, simulovaným žíháním, genetickými algoritmy a posilovaným učením. Popis jednotlivých metod byl rámcově velmi podobný, vždy byly uvedeny základní myšlenky metody, neformální i formální popis činnosti a samozřejmě algoritmus v textové formě či ve formě diagramu.

Teoretickou část uzavírá definice čtyř testovacích optimalizačních úloh, které jsou vhodné pro zkoumání nastavení parametrů optimalizačních metod. Byly vybrány dvě spojité a dvě diskrétní úlohy. Těmi spojitými jsou hledání globálního minima Ackleyho a Rastriginovy funkce, těmi diskrétními jsou pak problém více batohů a problém pokrytí množiny.

Následně byly zmíněny principy implementace optimalizačních metod a hlavně některé důležité detaily při jejich aplikaci na uvedené úlohy, jakou je například reprezentace řešení úlohy.

Nakonec byly prezentovány zjištěné výsledky při zkoumání optimálního nastavení parametrů optimalizačních metod. Uveďme nyní jejich shrnutí.

Optimalizace mravenčí kolonií se velmi osvědčila pro diskrétní úlohy. Platí, že ačkoliv zvyšování počtu iterací přinášelo pozitivní efekt, tak co se týče počtu agentů (mravenců), tak výsledky ukazují, že jich stačí mít jen několik desítek, například i jen 10, pro dosažení kvalitního řešení. Pro vypařování feromonů ve feromonovém modelu se jasné výsledky podařilo získat jen pro problém pokrytí množiny, v ostatních případech se bylo optimální nastavení specifické pro každou instanci daných úloh. Podobně tomu je i u parametrů pro aktualizaci feromonových hladin. Poté jsme zkoumali, zda je výhodnější upřednostňovat při výběru více feromonovou či heuristickou informaci. Ukázalo se, že je tomu pro každou úlohu jinak. Například pro problém více batohů je lépe upřednostňovat heuristickou informaci, pro problém pokrytí množiny naopak. Zásadním parametrem pro optimalizaci mravenčí kolonií ve variantě ACS je pravděpodobnost volby výběrového pravidla ACS namísto pravidla AS, které vnáší více náhody do procesu výběru následujícího kroku mravence. Ukazuje se, že je vhodné nastavit tuto pravděpodobnost na vysokou hodnotu (např. 0.9).

Pro aplikaci optimalizace mravenčí kolonií na diskrétní úlohy bylo zkoumáno použití více heuristických funkcí. Pro problém více batohů se osvědčila statická heuristika založená na podílu ceny a objemu objektu a pro problém pokrytí množiny dynamická heuristika založená na podílu počtu dosud nepokrytých prvků univerza, které daná množina pokrývá, a ceny této množiny.

Pro spojité úlohy se optimalizace mravenčí kolonií příliš neosvědčila. Ačkoli řešení úloh s nízkou dimenzí vykazuje, obzvláště při volbě vhodné heuristiky, výborné výsledky, tak pro vyšší dimenze vyvstává problém expanze počtu míst feromonového modelu, které je třeba uložit. To pak znamená nesmírnou paměťovou, ale i časovou náročnost, která může být jen stěží akceptovatelná.

Základními parametry pro optimalizaci hejnem částic jsou počet iterací algoritmu a počet částic. Pozitivně samozřejmě působí zvyšování hodnot u obou parametrů, avšak u diskrétních úloh je lépe klást důraz na počet částic namísto iterací, kterých může stačit jen několik desítek. Právě naopak je tomu pak u zkoumaných spojitých úloh. Pokud se zaměříme na topologii částic, pak můžeme říci, že se pro diskrétní úlohy osvědčila volba topologie hvězdy či kola a u spojitých kruhová topologie, ale celkově většinou nebyly zaznamenány velké rozdíly v kvalitě dosaženého řešení.

U optimalizace hejnem částic jsme si také položili otázku, která aktualizační rovnice vektoru rychlosti je nejvhodnější pro dané úlohy. Většinou bylo však těžké vybrat jednu konkrétní rovnici pro všechny instance jedné úlohy, snad jen pro hledání minima Rastriginovy funkce se nejlépe osvědčila rovnice typu Inertia. U ostatních úloh připadaly v úvahu klasická rovnice případně rovnice s omezujícím koeficientem. Obecnými parametry jsou při aktualizaci rychlosti také dva parametry, které vyjadřují váhu lokálního a globálního maxima. Pro činnost algoritmu je důležité, aby nebyl žádný z těchto parametrů nastaven na nulu, mimoto lze často užít hodnoty v rozsahu 0.5 až 1.5. Dodejme, že krom standardních parametrů jsme též hledali optimální natavení parametrů specifických pro aktualizační rovnici s omezující podmínkou a rovnici typu Inertia.

Pro genetické algoritmy začněme s počtem generací a velikostí populace, kdy se ukázalo, že je vhodné mít alespoň řádově stovky jedinců a stovky iterací. Další zajímavou informací je, že oproti obnově mnohem více záleží na typu selekce, přičemž můžeme říci, že velmi vhodná je selekce typu turnaj. Také pro různou volbu typu křížení se výsledky dramaticky neliší, ale často dominovalo uniformní křížení nad bodovým. A zatímco u pravděpodobnosti křížení se potvrdila běžně užívaná hodnota kolem 0.7-1, tak pro pravděpodobnost mutace se zjištěné hodnoty oproti standardně užívaným lišily. Pokud uvažujeme pravděpodobnost mutace jedince, kdy měníme jeden náhodný gen, tak se ukázalo, že by se měla pravděpodobnost takové mutace pohybovat v rozmezí 0.2 až 0.8. U klasické pravděpodobnosti mutace genu pak můžeme volit hodnotu v řádu procent.

U simulovaného žíhání se na první pohled nevyskytuje počet iterací, který jsme tradičně u zadaných metod zkoumali. Tento je skryt v několika jiných parametrech. Nejprve se jedná o počáteční teplotu, kterou je vhodné nastavit řádově na stovky případně tisíce. V nejzákladnější variantě je pak počet iterací dán tak, že teplotu násobíme koeficientem klesání teploty, který také ovlivní svým působením na teplotu, jak rychle bude klesat pravděpodobnost přijetí horšího stavu. Ukazuje se, že parametr klesání by neměl být nižší než 0.9, ale je třeba si dát pozor na přibližování se k hodnotě 1. Osobně považuji za poslední rozumně použitelnou hodnotu 0.99. Počet iterací lze však také navýšit počtem vnitřních iterací, kdy je vícekrát generován nový stav při stejné teplotě. Zahájit algoritmus s počtem alespoň 10 vnitřních iterací se zdá být rozumné. Konečně pak můžeme ještě navýšit počet iterací koeficientem růstu vnitřních cyklů, kdy vždy při nové teplotě zvětšíme počet vnitřních iterací vynásobením tímto koeficientem. Udává se, že by se tento koeficient měl pohybovat v rozmezí 1 až 2, ale pokud vezmeme v úvahu nastavení předchozích parametrů, tak překročení hodnoty 1.1 znamená, že počet vnitřních iterací enormně naroste. Tím se nám sice zvyšuje pravděpodobnost nalezení lepšího řešení, ale algoritmus nemusí skončit v rozumném čase.

Při simulovaném žíhání musíme určitým způsobem vybírat další kandidátní řešení (stav). Pro problém více batohů se osvědčilo vybírat nejlepší ze všech možných sousedních stavů. Naopak u problému pokrytí množiny se zdá být nejlepší vybrat jeden stav náhodně ze všech možných. U spojitých úloh je nejlepší vygenerovat pouze jednu náhodnou hodnotu v určitém okolí. Pro hledání minima Ackleyho funkce volíme okolí o velikosti 9 a u Rastriginovy funkce o velikosti 1 v každé dimenzi.

Pro optimalizaci založenou na posilovaném učení předem uveďme, že se osvědčila spíše pro diskrétní úlohy. Přesněji řečeno se posilované učení hodí pro úlohy, pro které jsme schopni uložit všechny hodnocené stavy. U spojitých úloh se nám tudíž dařilo řešit úlohy pouze do dimenze velikosti 3. Pro ty se osvědčilo použití klasických stavů, které reprezentují částečná řešení, a to ve spojení s Q-learning variantou posilovaného učení.

U diskrétních úloh se nám podařilo množství generovaných stavů obejít tím, že jsme použili stavy, které jsme pracovně nazvali relativní a které pouze vyjadřovali buďto umístění objektu v určitém batohu nebo užití určité pokrývající množiny, bez informace o předchozích volbách. Jelikož nás u diskrétních úloh ani tak nezajímá posloupnost voleb, kterými bylo konečné řešení utvořeno, ale zajímá nás jen finální podoba, tak jsme byli schopni i s těmito relativními stavy dosahovat solidních

výsledků. U diskrétních úloh jsme také zjistili, že pro obě varianty posilovaného učení, Q-learning i TD-learning, jsme schopni dosahovat přibližně stejně kvalitních řešení.

Pro volbu koeficientu učení můžeme říci, že absolutně nejlepší nastavení je pro každou úlohu individuální, přesto s postupně klesajícím vypočteným koeficientem učení jsme dosahovali vždy nadprůměrných výsledků, proto se jeví jako vhodné počáteční nastavení pro úlohy, o nichž nic v tomto ohledu nevíme. Podobně pro volbu discount faktoru jsme nebyli schopni určit konkrétní hodnotu, ale pro diskrétní úlohy se osvědčilo nepoužít hodnotu vyšší než 0.9.

Nyní uveďme možné pokračování a rozšíření této práce. Samozřejmě by bylo možné tuto práci v budoucnu rozšířit o další optimalizační metody a přístupy, například neuronové sítě. Také připadá v úvahu použití více testovacích optimalizačních úloh. Též připadá v úvahu důkladnější zkoumání vzájemného vlivu více parametrů na kvalitu dosaženého řešení. Co se týče implementační části, tak by bylo možné urychlit výpočet například implementací rychlejších vyhledávacího algoritmu pro hledání mezi hodnocenými stavy u posilovaného učení. Také by nebylo špatné pro vytvořený applet zprovoznit podporu vykreslování i 2D spojitých úloh.

Literatura

[1] Boyd, S. Vandenberghe, L.: Convex Optimization. Cambridge University Press, 2004. ISBN 0521833787

[2] Banabeau, E., Doringo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. New York, Oxford University Press, 1999. ISBN 0-19-513158-4

[3] Doringo, M., Stützle, T.: Ant Colony Optimization. Bradford Books, 2004. ISBN 0-262-04219-3

[4] Doringo, M., Birattari, M, Stützle, T.: Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique. Iridia, 2006, dostupné na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.9532&rep=rep1&type=pdf. ISSN 1781-3794

[5] Maniezzo, V., Gambardella, L. M., Luigi, F.: Ant Colony Optimization. 2004, dostupné na: http://www.idsia.ch/~luca/aco2004.pdf

[6] Doringo, M., Blum, C.: Ant colony optimization theory: A survey. Theoretical Computer Science. 2005, 344, stránky 243-278. ISSN 0304-3975

[7] Deneubourg, J. L., Aron, S., Goss, S., Pasteels, J.M.: The Self-Organizing Exploratory Pattern of the Argentine Ant. Journal of Insect Behavior, 1990. ISSN 0892-7553

[8] Goss, S., Aron, S., Deneubourg, J. L., Pasteels, J. M.: Self-organized shortcuts in the Argentine ant. Naturwissenschaften, 1989. ISSN 0028-1042

[9] Doringo, Marco a Gambardella, Luca Maria. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. Evolutionary Computations, 1997, sv. 1, stránky 53-66

[10] Eberhart, R. C., Shi, Y.: Particle swarm optimization: developments, applications and resources. Proceeding Congress on Evolutionary Computation, 2001, stránky 81-86

[11] Settles, M.: An Introduction to Particle Swarm Optimization [online]. 2005, cit. 2013-01-11, dostupné na: http://www2.cs.uidaho.edu/~tsoule/cs504/particleswarm.pdf

[12] Sivanandam, S. N., Deepa, S. N.: Introduction to Genetic Algorithms. Springer, 2008. ISBN 978-3-540-73189-4

[13] Whitley, D.: A Genetic Algorithm Tutorial. Statistics and Computing, 1994, stránky 65-85. dostupné na: http://www.cs.colostate.edu/~genitor/MiscPubs/tutorial.pdf

[14] Mitchell, M.: An Introduction to Genetic Algorithms. A Bradford Book The MIT Press, 1999. ISBN 0-262-63185-7

[15] Mathew, T. V.: Genetic Algorithm. Bombay [online]. 1996, cit. 2013-01-11, dostupné na: http://www.civil.iitb.ac.in/tvm/2701_dga/2701-ga-notes/gadoc.pdf

[16] Srinivas, M., Patnaik, L. M.: Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. Systems. Man and Cybernetics, 1994

[17] Sivaraj, R., Ravichandran, T.: A Review of Selection Methods in Genetic Algorithm. International Journal of Engineering Science and Technology, 2011, sv. 3. ISSN 0975-5462

[18] Tang, K. S., Man, K.F., Kwong, S., He, Q.: Genetic algorithms and their applications. IEEE Signal Processing Magazine. 1996, 13, stránky 22-37

[19] Vavak, F., Fogarty, T. C.: Comparison of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments. Evolutionary Computation, 1996, stránky 192-195

[20] Pencheva, T., Atanassov, K., Shannon, A.: Generalized Nets Model of Offspring Reinsertion in Genetic Algorithm. Annual of "Informatics" Section, Union of Scientists in Bulgaria, 2011, sv. 4, stránky 29-35

[21] Spall, J. C.: Introduction to Stochastic Search and Optimization. Estimation, Simulation, and Control. John Wiley & Sons, 2003. ISBN 0-471-33052-3

[22] Lam, J.: An Efficient Simulated Annealing Schedule. Yale University, 1988

[23] Nahar, S., Sahni, S., Shragowitz, E.: Simulated Annealing and Combinatorial Optimization. 23rd Design Automation Conference, 1986, stránky 293-299

[24] Kaelbling, L. P., Littman, M. L., Moore, A. W.: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, 1996, 4, stránky 237-285

[25] Moriarty, D. E., Schultz, A. C., Grefenstette, J. J.: Evolutionary Algorithms for Reinforcement Learning. Journal of Artificial Intelligence Research. 1999, sv. 4, 11, stránky 241-276

[26] Russell, S., Norvig, P.: Arfiticial Intelligence: A Modern Approach. Prentice Hall, 2010. ISBN 0-13-207148-7

[27] Sims, C. R. Reinforcement Learning: Model-free [online]. 2012, cit. 2013-01-11, dostupné na: http://www.bcs.rochester.edu/people/robbie/jacobslab/cheat_sheet/ModelFreeRL.pdf

[28] Pisinger, D.: Algorithms for Knapsack Problems. Copenhagen, University of Copenhagen, 1995

[29] Caprara, A., Fischetti, M., Toth, P.: A Heuristic Method for the Set Covering Problem. 1995, dostupné na: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.1922

[30] Molga, M., Smutnicki, C.: Test functions for optimization needs. 2005, dostupné na: http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf

[31] Fidanova, S.: Heuristics for Multiple Knapsack Problem. 2005. International Conference on Applied Computing. dostupné na: http://www.iadis.net/dl/final_uploads/200501c042.pdf

Seznam příloh

Příloha 1: Systémové požadavky

Applet vyžaduje použití nejlépe webového prohlížeče s nainstalovanou podporou Javy minimálně verze 7. Pokud budeme spouštět applet z lokálního PC, bude třeba mít nainstalovaný též JDK ve verzi alespoň 1.7.

Pro pohodlnou práci s appletem je vhodné mít k dispozici monitor s rozlišením alespoň 1024x768, není to však podmínkou.

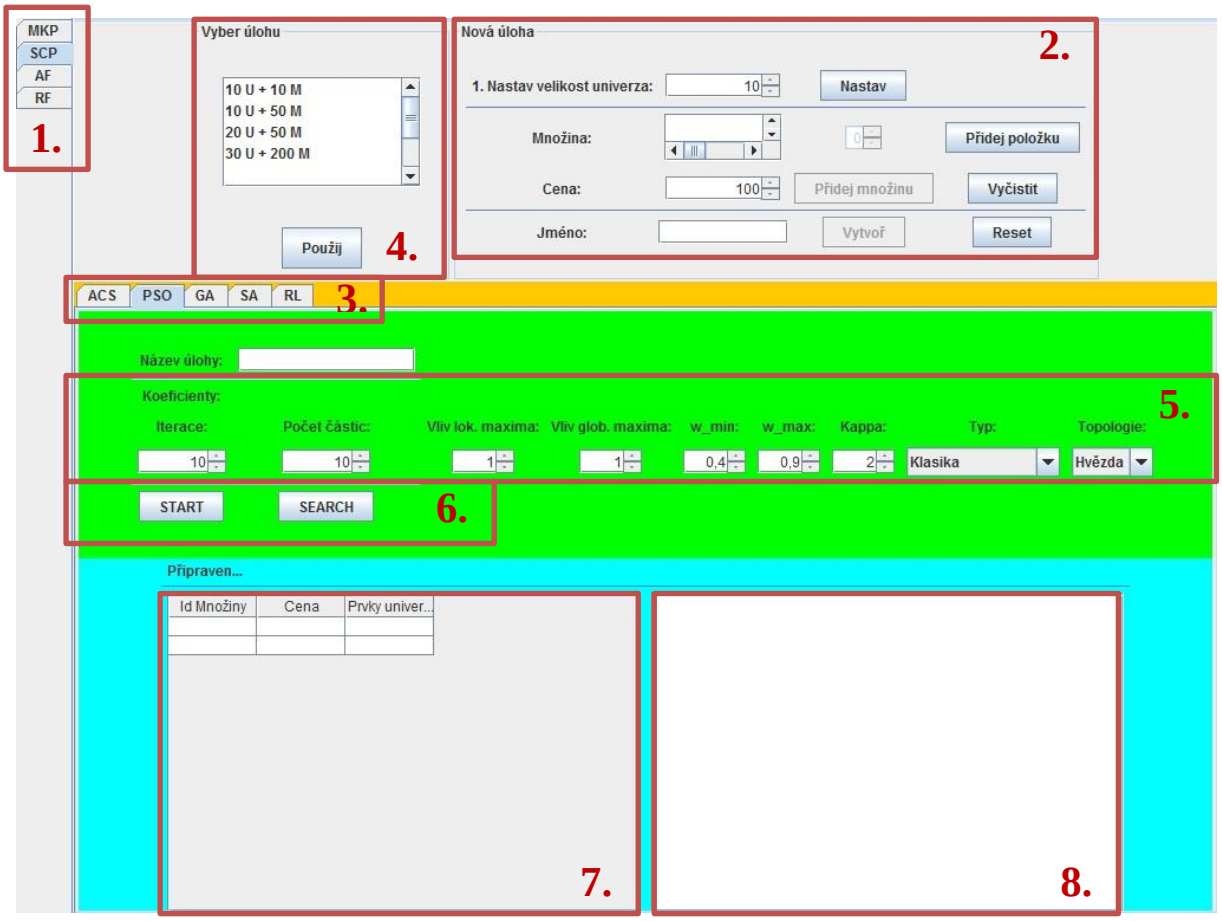
Procházení projektem a zdrojovými soubory usnadní například framework NetBeans ve verzi 7.2.1 či 7.3 nebo novější, který byl pro vývoj použit.

Projekt byl testován na operačním systému Windows 7, ale měl by fungovat i na jiných operačních systémech, které splňují výše zmíněné požadavky.

Příloha 2: Manuál

Pro optimalizaci vybrané úlohy postupujeme následovně:

1. Vybereme řešenou úlohu kliknutím na příslušnou záložku (MKP = problém více batohů, SCP = problém pokrytí množiny, AF = hledání globálního minima Ackleyho funkce, RF = hledání globálního minima Rastriginovy funkce).
2. Pokud je to žádoucí, vytvoříme nové zadání vybrané úlohy.
3. Vybereme optimalizační metodu, kterou hodláme úlohu řešit (ACS = optimalizace mravenčí kolonií, PSO = optimalizace hejnem částic, GA = genetický algoritmus, SA = simulované žíhání, RF = optimalizace na principu posilovaného učení).
4. Vybereme konkrétní úlohu a klikneme na tlačítko *Použij* (pojmenování úlohy se objeví v poli *Název úlohy* u vybrané optimalizační metody).
5. Nastavíme parametry optimalizační metody na požadované hodnoty.
6. Klikneme na tlačítko *START* a počkáme, dokud úloha není dokončena. Tlačítko *SEARCH* slouží pro hledání optimálního nastavení parametrů metody pro danou úlohu. Po jeho stisku jsou vypisovány průměrné dosažené fitness hodnoty pro různá nastavení parametrů.
7. Výsledky jsou zobrazovány v polích 7. a 8.



Obrázek P2.1: Použití appletu.

.

Příloha 3: Obsah přiloženého CD

Adresář či soubor	Popis
Applet	Složka se spustitelným appletem
Applet/index.html	HTML soubor pro spouštění appletu
Project	Složka s celým projektem
Project/src	Složka se zdrojovými soubory
text.pdf	Tento text ve formátu pdf
text.docx	Tento text ve formátu docx