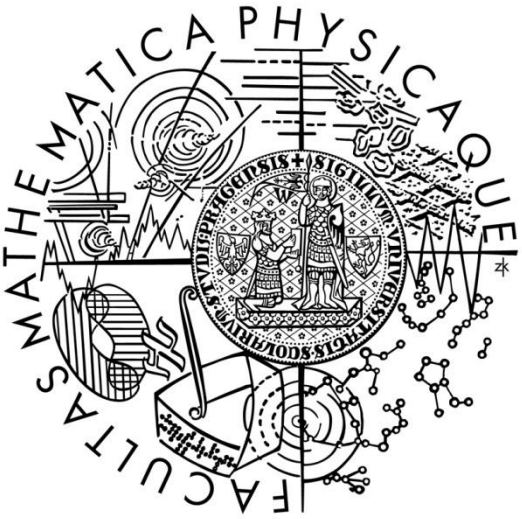
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Igor Kulman

Samoupravující seznamy

Katedra distribuovaných a spolehlivých systémů

Vedoucí diplomové práce: RNDr. Alena Koubková, CSc.

Studijní program: Informatika

Studijní obor: Softwarové systémy

Praha 2011

Rád by som poďakoval RNDr. Alene Koubkové, CSc., za cenné rady poskytnuté v priebehu tvorby tejto práce a za čas, ktorý vedeniu tejto práce venovala.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …….. dne.....

Název práce: Samoupravující seznamy

Autor: Bc. Igor Kulman

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí diplomové práce: RNDr. Alena Koubková, CSc.

Abstrakt:

Samoupravující seznamy jsou datové struktury sloužící k rychlému vyhledávání za předpokladu, že některé prvky v nich uložené jsou vyhledávány častěji než jiné, přičemž pravděpodobnosti přístupu k jednotlivým prvkům obecně nejsou předem známy. Efektivnějšího vyhledávání je dosaženo použitím různých permutačních pravidel, která průběžně mění uspořádání seznamu tak, aby častěji vyhledávané prvky byly blíže k jeho začátku. V této práci je uveden přehled známých algoritmů pro řešení tohoto problému (s uvedením teoretických výsledků o jejich složitosti, jsou-li známy) a experimentální studie o jejich chování (s využitím vlastních nebo volně dostupných implementací a programových prostředků pro generování vstupních dat, testování algoritmů a zpracování výsledků experimentů).

Klíčová slova: vyhledávání, lineární seznam, samoupravující seznam

Title: Self-organizing linear lists

Author: Bc. Igor Kulman

Department: Department of Distributed and Dependable Systems

Supervisor: RNDr. Alena Koubková, CSc.

Abstract:

Self-organizing linear lists are data structures for fast search, provided that certain elements stored in them are searched more frequently than others, while the probability of access to individual elements is generally not known in advance. Efficient search is achieved using different permutation rules that keep changing the list structure so that the more frequently searched elements are closer to the beginning. This thesis gives an overview of known algorithms for solving this problem (with the theoretical results about their complexity, if they are known), and experimental study of their behavior (using its own or freely available implementations and software for generating input data, testing algorithms and processing the results of experiments).

Keywords: search, linear list, self-organizing list

Obsah

1 Úvod

V programátorských úlohách sa na uloženie údajov často používajú lineárne spojové zoznamy. Takýto zoznam obsahuje prvky vyhľadávané na základne svojho kľúča. Cieľom vyhľadávania je zistiť, či sa daný prvok v zozname nachádza a ak áno, získať jeho dáta. Tieto zoznamy nebývajú väčšinou usporiadané žiadnym spôsobom, ktorý by urýchľoval vyhľadávanie.

Predpokladajme, že niektoré prvky sú vyhľadávané častejšie, ako iné. Samoupravujúci zoznam môže nejakým spôsobom permutovať poradie týchto prvkov po každom vyhľadaní prvku. Cieľom tejto permutácie je umiestnenie často vyhľadávaných prvkov bližšie k začiatku zoznamu, čím sa dosiahne zníženie času potrebného na ich ďalšie vyhľadávanie. Jednotlivé algoritmy sa líšia práve stratégiou používanou na vykonanie spomínanej permutácie .

Štúdium samoupravujúcich zoznamov začal svojou prácou J. McCabe [**24**] už v roku 1965. Vo svojej práci predstavil dve základné stratégie, a to Move-to-front, ktorá presúva nájdené prvky na začiatok zoznamu a Transpose, ktorá presúva nájdený prvok pred svojho predchodcu. Stratégiu Move-to-front charakterizuje rýchla konvergencia do stabilného stavu a schopnosť rýchlo sa prispôsobiť meniacim sa podmienkam. Stratégia Transpose naopak vďaka svojmu konzervatívnejšiemu správaniu dosahuje lepšie asymptotické usporiadanie zoznamu, na druhej strane však konverguje pomalšie a aj schopnosť reagovať na meniace sa podmienky je výrazne nižšia.

Viacero nasledujúcich prác sa pokúšalo skombinovať tieto dve stratégie a ťažiť tak z rýchlosti stratégie Move-to-front a presnosti stratégie Transpose. V [**12**], [**30**], [**32**] sú popísané hybridné stratégie, ktoré kombinujú stratégie Move-to-front a Transpose, objavujú sa aj dávkové stratégie, ktoré vykonávajú permutačné pravidlo vždy po k krokoch [**24**] alebo po k prístupoch k prvku [**21**].

V novších prácach ako [**19**], [**28**], [**2**] sa objavujú nedeterministické (pravdepodobnostné) stratégie, ktorých prístup k presunu nájdeného prvku je konzervatívnejší ako v prípade stratégie Move-to-front.

V práci [**20**] nájdeme štúdiu očakávanej zložitosti stratégie Move-to-front v závislých prostrediach s uvážením Markovovho modelu závislých prístupov, práca [**17**] ukazuje očakávaný čas vyhľadávania stratégie Move-to-front v prostrediach s lokalitou. Tieto štúdie boli neskôr v roku 2002 potvrdené prácou [**10**], ktorá sa zaoberala experimentom, ktorý otestoval skoro všetky dovtedy známe stratégie samoupravujúcich zoznamov. V tejto práci dosiahla stratégia Move-to-front lepšie výsledky ako všetky ostatné stratégie.

Z hľadiska praktického použitia samoupravujúcich zoznamov v reálnych aplikáciách je zaujímavý prípad popisovaný v [**11**]. V práci je popísaný rozsiahly simulačný systém, ktorého štart trval 5 minút a väčšinu tohto času zaberalo vyhľadávanie v symbolických tabuľkách. Po použití samoupravujúcej dátovej štruktúry sa tento čas skrátil na 30 sekúnd.

V tejto práci sa najprv oboznámime so základnými pojmami, uvedieme kritéria, podľa ktorých budeme samoupravujúce zoznamy hodnotiť. Ďalej uvedieme prehľad najznámejších samoupravujúcich stratégií s popisom známych teoretických výsledkov.

V experimentoch sa zameriame na meranie času a počtu porovnaní pri vyhľadávaní na samoupravujúcich zoznamoch. Porovnáme samoupravujúce stratégie vyhľadávania v spojovom zozname s klasickou stratégiou, medzi sebou, zistíme či a za akých podmienok konvergujú. Bude nás zaujímať závislosť správania samoupravujúcich stratégií na použitom pravdepodobnostnom rozdelení a vhodnosť ich použitia na testovaných pravdepodobnostných rozdeleniach.

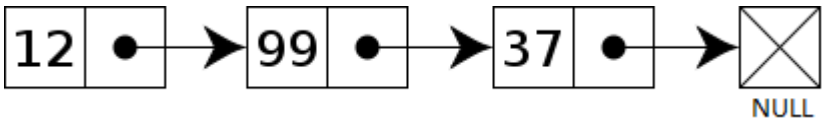
V závere sformulujeme získané poznatky a porovnáme ich (ak to bude možné) s teoretickými výsledkami.

2 Vyhľadávanie

V tejto kapitole stručne definujeme spojový zoznam a operácie, ktoré nás pri práci so samoupravujúcimi zoznamami budú zaujímať.

2.1 Spojový zoznam

Jednosmerný lineárny spojový zoznam je definovaný ako acyklická dátová štruktúra pozostávajúca z uzlov, kde každý uzol ukazuje na nasledujúci uzol a posledný uzol ukazuje na Null.



Obrázok 1 Jednosmerný spojový zoznam (zdroj: Wikipedia)

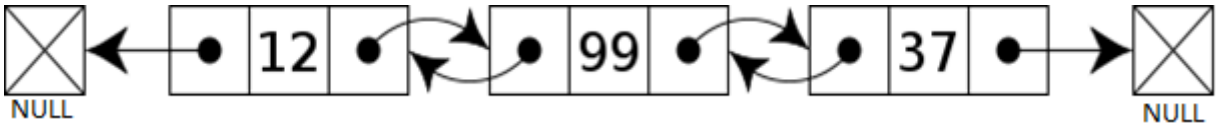
Uzol U obsahuje:

Data - dáta uzlu (kľúč, prípadne ďalšie údaje spojené s kľúčom)

Next - ukazovateľ na nasledujúci uzol

Naviac je daný ukazovateľ na začiatok zoznamu.

V obojsmernom spojovom zozname obsahuje každý uzol na viac ukazovateľ na predchádzajúci uzol.



Obrázok 2 Obojsmerný spojový zoznam (zdroj: Wikipedia)

Uzol U obsahuje:

Data - dáta uzlu (kľúč, prípadne ďalšie údaje spojené s kľúčom)

Next - ukazovateľ na nasledujúci uzol

Prev - ukazovateľ na predchádzajúci uzol

Naviac sú dané ukazovatele na začiatok a koniec zoznamu.

V tejto práci budeme pod pojmami spojový zoznam a lineárny spojový zoznam rozumieť vždy obojsmerný lineárny spojový zoznam a prvky reprezentovanej množiny stotožníme s ich kľúčmi.

Operácie

Budeme brať do úvahy nasledujúce operácie.

Member(x) – vyhľadá *x* v zozname *S*
Insert(x) – vloží *x* do zoznamu *S*, ak sa *x* v zozname nenachádza
Delete(x) – odstráni *x* zo zoznamu S

Ak je *S* nezotriedený zoznam, môžu byť tieto operácie implementované nasledujúcim spôsobom. Pri vyhľadávaní prvku musí algoritmus lineárne prejsť všetky prvky od začiatku zoznamu až po vyhľadávaný prvok alebo koniec zoznamu. Pri obojsmernom spojovom zozname je možné aj vyhľadávanie od konca smerom k začiatku.

Pri vkladaní prvku do zoznamu musí algoritmus najprv prejsť celý zoznam a overiť, že sa v ňom vkladaný prvkom nenachádza a následne ho vložiť na koniec zoznamu (vloženie prvku je možné aj na začiatok zoznamu).

Zmazanie prvku znamená jeho vyhľadanie v zozname, prvok sa zmaže na mieste a upravia sa ukazovatele predchádzajúceho a nasledujúceho uzla.

Zložitosť týchto operácií v najhoršom aj v očakávanom prípade je úmerná dĺžke zoznamu, teda veľkosti reprezentovanej množiny.

2.2 Samopravujúci spojový zoznam

Predpokladajme, že niektoré prvky sú vyhľadávané častejšie, ako iné. Samoupravujúci zoznam môže nejakým spôsobom permutovať poradie týchto prvkov po každom vyhľadaní prvku. Cieľom tejto permutácie je umiestnenie často vyhľadávaných prvkov bližšie k začiatku zoznamu, čím sa dosiahne zníženie času potrebného na ich ďalšie vyhľadávanie. Jednotlivé algoritmy sa líšia práve stratégiou používanou na vykonanie spomínanej permutácie .

Niekedy sa v literatúre pri samoupravujúcich spojových zoznamoch uvádza použitie danej permutácie aj pri vkladaní prvku. V tejto práci sa pod pojmom vloženie prvku do samoupravujúceho zoznamu myslí vždy jeho vloženie na koniec zoznamu, teda vloženie bez použitia daných permutačných pravidiel.

Skôr ako si popíšeme niektoré konkrétne stratégie, zavedieme všeobecné charakteristiky, podľa ktorých bude kvalita a efektivita týchto stratégií hodnotená.

3 Hodnotenie stratégií

V tejto kapitole vysvetlíme rôzne typy zložitosti algoritmov, zavedieme pojmy konvergencia a stabilný stav a popíšeme tzv. optimálny algoritmus.

3.1 Základné pojmy

Prvky zoznamu budeme pre jednoduchosť označovať *1* až *n*. Nech σ je vyhľadávacia sekvencia taká, že $\sigma[k]$ je prvok vyhľadávaný pri k-tej operácií Member. Označme $\Lambda(\sigma, a)$ konfiguráciu zoznamu (v zmysle usporiadania prvkov zoznamu) takú, že $\Lambda(\sigma, a)_r$ je pozícia prvku *r* v zozname po prvých *a* dotazoch z σ (a permutácií nasledujúcej po každom dotaze). $\Lambda(\sigma, 0)$ potom označuje počiatočnú konfiguráciu, pre zjednodušenie označovanú λ.

Samoupravujúci zoznam môže počas obsluhovania vyhľadávacej sekvencie svoje prvky preusporiadať. Ihneď po úspešnom vyhľadaní prvku alebo vložení nového prvku môže byť tento prvok presunutý na ľubovoľnú pozíciu bližšie k začiatku. Takéto výmeny prvkov nazývame bezplatné výmeny a k celkovej zložitosti vôbec neprispievajú. V dôsledku toho sa u algoritmov, ktoré pri preusporiadaní používajú len neplatené výmeny, redukuje ich zložitosť iba na zložitosť vyhľadávania. Pomocou bezplatných výmen môže samoupravujúci zoznam znížiť zložitosť nasledujúcich požiadaviek. Dva susedné prvky môžu byť vymenené v ľubovoľnom čase za cenu 1. Takéto výmeny prvkov nazývame platené výmeny. Tento model nazývame štandardný model. V prácach [**24**] a [**29**] autori predstavili P^dmodel. V tomto modele nie sú žiadne bezplatné výmeny a cena každej platenej výmeny je *d*. V tejto práci predpokladáme použitie štandardného modelu.

3.2 Zložitosť

Podľa [**13**] budeme zložitosťou permutačného algoritmu A pre dané λ a σ rozumieť priemernú zložitosť operácie Member v zmysle počtu prístupov k prvkom zoznamu potrebných na nájdenie vyhľadávaného prvku. Množstvo práce potrebnej na následnú permutáciu prvkov v zozname sa v štandardnom modeli do zložitosti nezapočítava.

Definícia 1: Označme $\Lambda(\sigma, k-1)_{\sigma[k]}$ umiestnenie k-tého vyhľadávaného prvku (zodpovedá počtu prístupov do zoznamu potrebných na jeho nájdenie). Potom $C_A(\lambda, \sigma)$

(zložitosť vyhľadania) je suma tejto hodnoty pre všetky prvky v σ v pomere k celkovému počtu prvkov v σ :

$$C_A(\lambda, \sigma) = \frac{\sum_{k=1}^{|\sigma|} \Lambda(\sigma, k-1)_{\sigma[k]}}{|\sigma|}$$

Predpokladáme však, že na začiatku vyhľadávania σ nepoznáme, preto môže byť permutačný algoritmus hodnotený iba pomocou predpokladov o vyhľadávacej postupnosti.

Zložitosť v najhoršom prípade

Zložitosť permutačného algoritmu A v najhoršom prípade je maximum $C_A(\lambda, \sigma)$ cez všetky λ a σ .

Očakávaná zložitosť

Vo všeobecnosti je očakávaná zložitosť permutačného algoritmu očakávaná zložitosť cez všetky λ a σ. Použitie permutačných algoritmov však implikuje predpoklad, že k niektorým prvkom sa pristupuje s vyššou pravdepodobnosťou ako k iným. Bez tohto predpokladu by sme museli predpokladať, že ku všetkým prvkom sa pristupuje úplne náhodne (ale s rovnakou pravdepodobnosťou) a že žiadny počet preusporiadaní nezvýši šancu nasledujúceho prvku nachádzať sa bližšie k začiatku zoznamu. Práve preto analýzy očakávanej zložitosti zvyčajne predpokladajú nejaké obmedzenia vyhľadávacej sekvencie σ. Zvyčajne sa predpokladá, že pravdepodobnosť prístupu k prvkom je pevná a riadi sa nejakým známym pravdepodobnostným rozdelením.

Podľa [**5**] môžeme očakávanú zložitosť formálne definovať nasledujúcim spôsobom. Pri rozdelení pravdepodobnosti $\vec{p} = (p_1 \dots p_n)$ a vyhľadávacej postupnosti vygenerovanej podľa $\vec{p}$ sa konfigurácia zoznamu pre algoritmus A riadi Markovovim reťazcom s n! stavmi konvergujúcim do stacionárneho stavu. Pre každý z n! stavov $S_i, 1 \leq i \leq n!$ označme q_i ako stacionárnu pravdepodobnosť S_i.

Definícia 2: Pre každý prvok $x_j, 1 \leq j \leq n$, označme $pos(x_j, S_i)$ ako pozíciu x_j v konfigurácií reprezentovanej S_i . Potom očakávaná zložitosť algoritmu A pri vyhľadávaní prvku x_j v postupnosti vygenerovanej podľa $\vec{p}$ je

$$e_A(x_j) = \sum_{1 \le i \le n!} q_i pos(x_j S_i)$$

a očakávaná zložitosť algoritmu A pri obsluhe jedného dotazu vo vyhľadávacej postupnosti vygenerovanej podľa $\vec{p}$ je

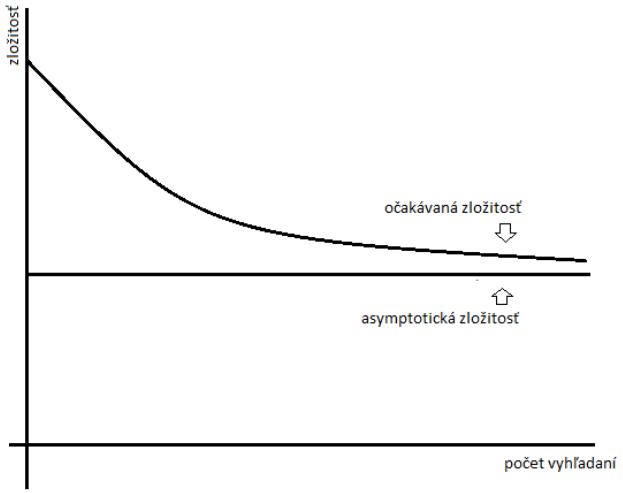
$$E_A(\vec{p}) = \sum_{j=1}^{n} p_j e_A(x_j)$$

Amortizovaná zložitosť

Analýzy najhoršieho prípadu série operácií rovnakého typu často uvažujú zložitosť jednej operácie v najhoršom prípade vynásobenú ich počtom. Vo väčšine prípadov je však nemožné, aby tento najhorší prípad nastal v každom kroku. Amortizovaná zložitosť berie tento fakt do úvahy a počíta priemernú cenu operácie ako najhorší prípad celej sekvencie operácií vydelený ich počtom.

3.3 Konvergencia a stabilný stav

Predkladáme, že použitie permutačných algoritmov spôsobí presun prvkov, ktoré sú častejšie vyhľadávané, smerom k začiatku zoznamu. Nie je však rozumné predpokladať, že usporiadanie prvkov v zozname bude konvergovať do perfektného usporiadania vzhľadom k pravdepodobnostiam vyhľadávania a že v tomto stave zostane.



Obrázok 3 Priebeh konvergencie

Predpokladáme však, že usporiadanie zoznamu dosiahne stabilný stav, po ktorého dosiahnutí ďalšie permutácie už nijak výrazne neovplyvnia rýchlosť vyhľadávania. Tento

stabilný stav nemusí tvoriť jediné usporiadanie zoznamu, môže isť o viacero usporiadaní, ktorých zložitosť je blízka hodnote funkcie $C_A(\lambda, \sigma)$ pre veľké | σ | (na obrázku 3 označená ako asymptotická zložitosť). Čas vyžadovaný na dosiahnutie stabilného stavu (prípadne počet prístupov) nazývame konvergencia algoritmu.

Lokalita

Ako už bolo uvedené, predpokladáme, že pravdepodobnosť prístupu ku každému prvku je pevná pre každý prvok σ. Keďže sa často taktiež predpokladá, že dotazy sú navzájom nezávislé, analýza očakávanej zložitosti predpokladá, že všetky vyhľadávacie sekvencie vygenerované podľa daného rozdelenia sú rovnako pravdepodobné. Tieto predpoklady však nemodelujú vlastnosť vyhľadávacích sekvencií nazvanú lokalita, keď subsekvencie σ môžu mať relatívne pravdepodobnosti prístupov výrazne odlišné od celkových relatívnych frekvencií.

Vyhľadávacia postupnosť vykazuje lokalitu, ak v ľubovoľnom čase vyhľadáva len malú podmnožinu prvkov. Ak je prvok vyhľadávaný, je veľká šanca, že bude znovu vyhľadávaný v blízkej budúcnosti. Na základe toho je v modeli definovanom v práci [**3**] zavedený pojem beh definovaný ako podpostupnosť požiadaviek na rovnaký prvok. V najlepšom prípade, pri vysokom stupni lokality, je prvok vyhľadávaný veľakrát za sebou predtým, ako je vyhľadávaný iný prvok. V praxi mávajú vyhľadávacie sekvencie iba veľmi málo takýchto dlhých behov. Dlhé behy sa však môžu objaviť, ak sa zameriame na malé množiny prvkov, špeciálne na dvojice. Ak je v ľubovoľnom čase prvok *x* významnejší ako prvok *y,* je veľká šanca, že tento vzťah bude platiť aj v blízkej budúcnosti a pred požiadavkou na prvok *y* sa stretneme s viacerými požiadavkami na prvok *x*. Vyhľadávacia sekvencia sa teda vyznačuje vysokým stupňom lokality, ak značná časť jej požiadavkou patrí do dlhých behov.

V práci [**8**] a vo viacerých starších prácach ([**10**], [**20**]) je model s lokalitou definovaný na základe Markovových reťazcov. V tomto modeli je zoznam s *n* prvkami rozdelený na *k* disjunktných podmnožín s *m* prvkami, kde *n = km*. Tieto podmnožiny sa nazývajú lokálne podmnožiny. Prvky jednotlivých lokálnych podmnožín sú závislé, to znamená, že ak je v čase *t* vyhľadávaný prvok z lokálnej podmnožiny *i*, je veľká pravdepodobnosť, že v čase *t+1* bude vyhľadávaný prvok z rovnakej lokálnej podmnožiny.

Lokalitu v našej práci do experimentov nezahrnieme.

3.4 Optimálny algoritmus

Ťažkosti pri nachádzaní absolútnych kritérií na porovnávanie algoritmov podmienili vznik relatívnych mier. Relatívne porovnanie algoritmov je užitočné najmä ak nie je možné pre daný algoritmus získať žiadnu absolútnu mieru a ak nie je porovnanie mier algoritmov vôbec možné. Zložitosť algoritmu sa preto často porovnáva s optimálnym algoritmom.

Permutačné pravidlo je optimálne v rámci nejakej množiny pravidiel ak má z týchto pravidiel najmenšiu zložitosť cez všetky rozdelenia pravdepodobnosti vyhľadávania a cez všetky úvodné usporiadania zoznamu [**30**]. Stratégia Transpose je podľa tejto definície optimálny algoritmus [**30**].

Častejšie sa však za optimálny algoritmus pokladá off-line algoritmus, teda algoritmus, ktorý dopredu pozná všetky požiadavky na vyhľadávanie a dokáže ich teda vykonať s minimálnou zložitosťou [**31**]. V literatúre sa zvykne označovať ako optimálne statické poradie (OSO). Pri vyhľadávacej postupnosti σ označme $C_A(\sigma)$ zložitosť algoritmu A a $C_{OPT}(\sigma)$ zložitosť optimálneho algoritmu OPT.

Definícia 3: Algoritmus A sa nazýva c-kompetitívny, ak existuje konštanta *k* taká, že pre zoznamy všetkých veľkostí a pre všetky vyhľadávacie postupnosti σ a platí

$$C_A(\sigma) \leq c.C_{OPT}(\sigma) + k$$

Faktor *c* sa nazýva kompetitívny pomer.

Kompetitivita nedeterministického algoritmu

Kompetitivita nedeterministického (pravdepodobnostného) algoritmu je definovaná vzhľadom k tzv. protivníkovi. V [**7**] sa uvádzajú traja takýto protivníci:

- Nevedomý protivník (anglicky Oblivious Adversary) : musí dopredu vygenerovať celú vyhľadávaciu postupnosť ešte predtým, ako je akákoľvek požiadavka obslúžená on-line algoritmom. Zložitosť tohto protivníka je optimálna zložitosť off-line algoritmu pre túto postupnosť.
- Adaptívny on-line protivník (anglicky Adaptive On-line Adversary): môže pozorovať on-line algoritmus a generovať nasledujúcu požiadavku na základe odpovedí na predchádzajúce požiadavky algoritmu. Tento protivník musí obslúžiť každú

požiadavku on-line, teda bez znalostí náhodných rozhodnutí algoritmu pre súčasnú alebo akúkoľvek budúcu požiadavku

- Adaptívny off-line protivník (anglicky Adaptive Off-line Adversary): generuje vyhľadávanú postupnosť adaptívne, jeho zložitosť je optimálna off-line zložitosť pre túto postupnosť.

Definícia 4: Nedeterministický algoritmus A sa nazýva c-kompetitívny proti nevedomému ľubovoľnému protivníkovi ak existuje konštanta b taká, že pre všetky dĺžky zoznamu a všetky vyhľadávané postupnosti σ vygenerované nevedomým protivníkom platí

$$E[C_A(\sigma)] \leq c.C_{OPT}(\sigma) + b$$

Ide o odhad cez náhodne rozhodnutia algoritmu A.

Ak je daný nedeterministický algoritmus A a adaptívny on-line protivník ADV, označme $E[C_A]$ a $E[C_{ADV}]$ očakávané zložitosti A a ADV pri obslúžení vyhľadávacej postupnosti vygenerovanej ADV.

Definícia 5: Nedeterministický algoritmus A sa nazýva c-kompetitívny proti ľubovoľnému adaptívnemu on-line (adaptívnemu off-line) protivníkovi, ak existuje konštanta b, taká, že pre všetky dĺžky zoznamu a všetkých adaptívnych on-line (adaptívnych off-line) protivníkov ADV platí

$$E[C_A] \leq c.E[C_{ADV}] + b$$

Ide o odhad cez náhodné rozhodnutia algoritmu A.

4 Známe stratégie

V tejto kapitole popíšeme najznámejšie stratégie samoupravujúcich zoznamov. Pri každej stratégií vysvetlíme jej myšlienku a uvedieme aj zápis v pseudokóde, prípadne aj teoretické poznatky, ak sú pre danú stratégiu známe. Na záver uvedieme porovnanie jednotlivých stratégií na základe ich teoretických charakteristík.

Niekedy sa v literatúre pri samoupravujúcich zoznamoch uvádza použitie danej permutácie aj pri vkladaní prvku. V tejto práci sa pod pojmom vloženie prvku do samoupravujúceho zoznamu myslí vždy jeho vloženie na koniec zoznamu, teda vloženie bez použitia daných permutačných pravidiel.

V nasledujúcom texte budeme pod pojmom prístup k prvku rozumieť jeho vyhľadanie operáciou Member.

4.1 Stratégia OSO

Stratégia optimálneho statického poradia (OSO) je off-line stratégia, teda stratégia, ktorá dopredu pozná všetky požiadavky na vyhľadávanie a dokáže ich teda vykonať s minimálnou zložitosťou. Táto stratégia sa zvykne pokladať za optimálny algoritmus, všetky samoupravujúce stratégie teda budeme porovnávať najmä s touto stratégiou.

Na rozdiel od stratégie OSO sú všetky ďalej popisované stratégie on-line stratégiami, nepoznajú teda dopredu žiadne požiadavky na vyhľadávanie.

4.2 Zotriedený zoznam

K samoupravujúcim zoznamom môžeme zahrnúť aj tento jednoduchý a dobre známy algoritmus. Na rozdiel od iných stratégií dochádza v tomto algoritme k presunu prvkov pri vkladaní nového prvku do zoznamu. Jednotlivé prvky zoznamu sú vždy zotriedené podľa kľúča vzostupne a novo vkladaný prvok je zaradený na miesto, kam patrí, teda bezprostredne za všetky prvky s menším kľúčom. Pri vyhľadávaní tak nie je potrebné prehľadať vždy celý zoznam, ak algoritmus narazí na prvok s väčším kľúčom ako ten vyhľadávaný je jasné, že prvok sa v zozname nenachádza.

V testoch označujeme zotriedený zoznam ako stratégiu Sorted.

4.3 Stratégia Move-to-front

Popis stratégie

Najjednoduchšou a zároveň najradikálnejšou stratégiou používanou pri samoupravujúcich zoznamoch je stratégia Move-to-front. Pri každej úspešnej operácií Member je nájdený prvok presunutý úplne na začiatok zoznamu.

Táto stratégia rýchlo konverguje, má však veľkú asymptotickú zložitosť. Dôvodom je, že ak sa pristupuje k prvku s nízkou pravdepodobnosťou prístupu, ten je presunutý na začiatok zoznamu, čo zvýši čas potrebný na prístup ku všetkým ostatným prvkom v ďalších krokoch.

Pseudokód

```
Pri nájdení prvku x:

      Presuň x na začiatok zoznamu
```

Presun prvku na začiatok zoznamu znamená jeho zaradenie pred prvý prvok zoznamu, nie výmenu s aktuálne prvým prvkom zoznamu.

Teoretické poznatky

Veta 1: Asymptotická očakávaná zložitosť stratégie Move-to-front pre vyhľadanie jedného prvku z vyhľadávacej sekvencie $\boldsymbol{\sigma}$ je

$$1 + 2 \sum_{1 \le i \le j \le n} \frac{p_i p_j}{p_i + p_j}$$

kde $\vec{p} = (p_1 \dots p_N)$ je rozdelenie pravdepodobnosti prístupov k prvkom

Dôkaz: uvedený v [**12**]

Veta 2: Stratégia Move-to-front je 2-kompetitívna.

Dôkaz: uvedený v [**7**]

4.4 Stratégia Transpose

Popis stratégie

Ako už napovedá názov tejto stratégia, jej cieľom je výmena dvoch prvkov. Pri každej úspešnej operácií Member, ak nájdený prvok nie je úplne na začiatku zoznamu, vymení si pozíciu s prvkom, ktorý je jeho bezprostredným predchodcom. Prvok sa tak dostane na začiatok zoznamu len v prípade, že sa k nemu často pristupuje.

Táto stratégia konverguje pomalšie, pri malej lokalite prístupov však dosahuje nižšiu asymptotickú zložitosť.

Stratégiu Transpose je možné efektívne implementovať aj v poliach.

Pseudokód

```
Pri nájdení prvku x:

      Ak x nie je na začiatku zoznamu
            vymeň x a x.prev
```

Teoretické poznatky

Veta 3: Priemerný čas vyhľadávania jedného prvku z vyhľadávacej sekvencie $\boldsymbol{\sigma}$ je

$$\frac{\sum_{\pi}\left[\left(\prod_{i=1}^{n} p_i^{\delta(i,\pi)}\right)\sum_{j=1}^{n} p_j \pi(j)\right]}{\sum_{\pi}\prod_{i=1}^{n} \mathrm{p_i}^{\delta(i,\pi)}}$$

kde π označuje možné usporiadania prvkov, $\pi(j)$ označuje j-tý prvok v danom usporiadaní a $\delta(i,\pi)$ označuje kvantitu $i-\pi(i)$ alebo vzdialenosť, o ktorú bude prvok *i* posunutý v π oproti svojej pôvodnej pozícií.

Dôkaz: uvedený v [**30**]

Veta 4: Stratégia Transpose nie je c-kompetitívna pre žiadne c.

Dôkaz: uvedený v [**31**]

4.5 Stratégia Count

Popis stratégie

Pri každom prvku je udržiavaná hodnota Count udávajúca koľkokrát už bol prvok vyhľadávaný. Táto hodnota je po každom úspešnom vyhľadaní prvku inkrementovaná. Vyhľadaný prvok je následne presunutý smerom k začiatku zoznamu pred všetky prvky s menšou hodnotou Count.

Všetky prvky v zozname sú vždy usporiadané rastúcim spôsobom podľa hodnoty Count. Nevýhodou tejto stratégia však je vyššia pamäťová náročnosť.

Pseudokód

```
Inicializácia:

      Pre každý prvok x v zozname nastav x.count = 0

Pri nájdení prvku x:

      Inkrementuj x.count o 1
      Kým x.prev != null a x.prev.count < x.count
            vymeň x a x.prev
```

Teoretické poznatky

Veta 5: Stratégia Count nie je c-kompetitívna pre žiadne c.

Dôkaz: uvedený v [**31**]

4.6 Stratégia Move-ahead-k

Popis stratégie

Stratégia Move-ahead-k je akýsi kompromis medzi relatívne extrémnymi stratégiami Move-to-front a Transpose. Vyhľadaný prvok je presunutý o *k* pozícií smerom k začiatku zoznamu. Podľa definície tejto stratégie, ak je *f* vzdialenosť k začiatku zoznamu, tak Move-to-front zodpovedá Move-ahead-f a Transpose zodpovedá Move-ahead-1.

Implementácia tejto metódy je priamočiara pre konštantné hodnoty *k*, posun o *k* pozícií späť smerom k začiatku zoznamu je možný, ak ide o implementáciu v obojsmernom

spojovom zozname. Implementácia v jednosmernom spojovom zozname vedie k zbytočnému druhému prechodu zoznamom a počítaniu pozícií jednotlivých prvkov.

Parameter *k* nemusí byť pevná číselná konštanta, môže závisieť na dĺžke zoznamu a teda byť v dlhších zoznamoch väčší ako v kratších alebo na pozícií vyhľadávaného prvku v zozname, kedy sa budú vzdialenejšie prvky posúvať o väčšiu vzdialenosť ako bližšie.

Pseudokód

```
Pri nájdení prvku x:

      Opakuj k-krát
            Ak x.prev!=null
            Vymeň x a x.prev
```

4.7 Stratégia Jump

Popis stratégie

Táto stratégia navrhuje náhodnú metódu zanechania spätného ukazovateľa na prvok počas vyhľadávania s tým, že tento ukazovateľ bude neskôr použitý na presun nájdeného prvku. Táto metóda odstraňuje nevyhnutnosť druhého prechodu zoznamom popísaného pri stratégii Move-ahead-k.

Posun spätného ukazovateľa postupne dopredu počas vyhľadávania by zodpovedal lineárnemu prechodu na konci vyhľadávania, ukazovateľ preto musí príležitostne preskočiť na pozíciu práve testovaného prvku. Spätný ukazovateľ je posunutý na pozíciu testovaného prvku práve ak testovaný prvok nie je vyhľadávaný prvok a booleovská funkcia Jump má hodnotu 1. Funkcia Jump môže byť funkcia premenných ako aktuálna pozícia spätného ukazovateľa, aktuálna pozícia testovaného prvku, počet vykonaných skokov a pod.

Pseudokód

```
Inicializácia

      Nastav backPointer na prvý prvok v zozname

Pri nájdení prvku x:

      Ak práve testovaný prvok y != x a JUMP(...)==true
            Nastav backPointer = y

      Vlož x pred backPointer
```

Definujeme JUMP(y,backPointer) ako triedu funkcií akceptujúcich ako parametre polohu aktuálne testovaného prvku a aktuálny spätný ukazovateľ. Stratégia Move-To-Front je potom vlastne stratégia JUMP s funkciou JUMP vracajúcou vždy false a stratégia Transpose je stratégia JUMP s funkciou JUMP vracajúcou vždy true. S použitím nekonštantnej funkcie JUMP je možné dosiahnuť posun prvok o premenlivú vzdialenosť smerom k začiatku zoznamu.

Funkcia JUMP môže byť fixná alebo pravdepodobnostná. Príkladom fixnej funkcie JUMP môže byť funkcia

$$JUMP(y, backPointer) = (y \geq backPointer + 2c)$$

pre ľubovoľné celé číslo c popísaná v [**14**]. Príkladom pravdepodobnostnej funkcie JUMP je

$$P(JUMP(y) = true) = \begin{cases} \frac{c}{y} & pre\ y \geq c \\ 1 & pre\ y < c \end{cases}$$

pre konštantné c > 0 popísaná taktiež v [**14**].

4.8 Stratégia Timestamp

Popis stratégie

Deterministická verzia

Táto stratégia má dve verzie, deterministickú a nedeterministickú (pravdepodobnostnú). Deterministická verzia je niekedy označovaná aj ako Timestamp(0). Stratégia Timestamp(0) premiestni vyhľadávaný prvok pred prvého predchodcu v zozname, ktorý nebol vyhľadávaný od posledného prístupu k vyhľadávanému prvku. Ak takýto prvok neexistuje, alebo vyhľadávaný prvok je vyhľadávaný prvýkrát, jeho poloha v zozname sa nezmení.

Pseudokód

```
Pri nájdení prvku x v čase t:

 Ak prvok x nebol vyhľadávaný v čase [1, t-1]
      Skonči
 Inak
      Nech t'∈[1, t-1] je čas posledného vyhľadávania prvku x
      Nech v_x(t) je prvok najbližšie k začiatku zoznamu a
      zároveň je predchodcom x a
           nebol vyhľadávaný  v čase <t',t-1>
      alebo
            bol vyhľadávaný v čase <t',t-1> práve raz

      Ak takýto prvok neexistuje, potom v_x(t)= x

      Vlož x pred v_x(t)
```

Teoretické poznatky

Veta 6: Očakávaná asymptotická zložitosť deterministickej stratégie Timestamp pri vyhľadávaní prvku $x_j, 1 \leq j \leq n$ z vyhľadávacej sekvencie $\boldsymbol{\sigma}$ je

$$\frac{1}{2} + \sum_{i=1}^{n} \frac{{p_i}^3 + 3{p_i}^2 p_j}{\left(p_i + p_j\right)^3}$$

kde $\vec{p} = (p_1 \ldots p_N)$ je rozdelenie pravdepodobnosti prístupov k prvkom.

Dôkaz: uvedený v [**5**]

Veta 7: Očakávaná asymptotická zložitosť deterministickej stratégie Timestamp pri vyhľadávaní jedného prvku z vyhľadávacej sekvencie $\boldsymbol{\sigma}$ vygenerovanej podľa rozdelenie pravdepodobnosti prístupov k prvkom $\vec{p} = (p_1 \dots p_N)$ je

$$\sum_{1 \le i \le j \le n} \frac{p_i p_j}{p_i + p_j} \left(2 - \frac{(p_i - p_j)^2}{(p_i + p_j)^2} \right)$$

Dôkaz: uvedený v [**4**]

Veta 8: Stratégia Timestamp(0) je 2-kompetitívna.

Dôkaz: uvedený v [**2**]

Nedeterministická (pravdepodobnostná) verzia

Nedeterministická (pravdepodobnostná) verzia stratégie Timestamp, označovaná Timestamp(p), sa od deterministickej verzie líši tým, že s pravdepodobnosťou *p* presunie vyhľadávaný prvok rovno na začiatok zoznamu a s doplnkovou pravdepodobnosťou *1-p* vykoná rovnaké operácie, ako deterministická verzia Timestamp(0). Timestamp(0) je vlastne špeciálnym prípadom Timestamp(p) s nulovou šancou presunu vyhľadávaného prvku priamo na začiatok zoznamu.

Pseudokód

```
Pri nájdení prvku x v čase t:

      S pravdepodobnosťou p

            (a) Presuň x na začiatok zoznamu

      S pravdepodobnosťou 1-p

            (b) Ak prvok x nebol vyhľadávaný v čase [1, t-1]
                  Skonči
                Inak
                 Nech t'∈[1, t-1] je čas posledného vyhľadávania prvku x
                 Nech v_x(t) je prvok najbližšie k začiatku zoznamu a
                  zároveň je predchodcom x a
                       nebol vyhľadávaný  v čase <t',t-1>
                  alebo
```

```
                    bol vyhľadávaný v čase <t',t-1> práve raz a toto
                    vyhľadávanie bolo obslúžené podľa (b)

            Ak takýto prvok neexistuje, potom v_x(t)= x

            Vlož x pred v_x(t)
```

Teoretické poznatky

Veta 9: Pre každé reálne číslo $p \in [0,1]$ je algoritmus Timestamp(p) c-kompetitívny, kde $c = \max\{2-p, 1+p(2-p)\}$ proti ľubovoľnému nevedomému protivníkovi.
Dôkaz: uvedený v [**2**]

Na základe tejto vety je ideálne zvoliť *p* tak, aby *c* bolo minimálne, teda *p*=0.382, čím dosiahne stratégia kompetitívny faktor *c*=1.6181.

4.9 Stratégia Split

Popis stratégie

Stratégia Split je nedeterministická pravdepodobnostná stratégia. Jej fungovanie je zložitejšie ako fungovanie predchádzajúcich stratégií, uvádzame preto len popis v pseudokóde.

Pseudokód

```
Inicializácia:

      Pre každý prvok x v zozname nastav x.split = x

Pri nájdení prvku x:

      Pre každý prvok y taký, že y.split = x nastav y.split = x.next

      S pravdepodobnosťou ½:
            Presuň x na začiatok zoznamu

      S pravdepodobnosťou ½:
            Vlož x pred prvok x.split
            Ak je y predchodcom x a x.split == y.split nastav y.split = x

      Nastav x.split na prvý prvok v zozname
```

Teoretické poznatky

Veta 10: Stratégia Split je 15/8-kompetitívna proti ľubovoľnému nevedomému protivníkovi.

Dôkaz: uvedený v [**7**]

4.10 Stratégia Bit

Popis stratégie

Pri každom prvku je udržiavaný bit, ktorý je nahradený svojim komplementom pri každom vyhľadaní tohto prvku. Ak vyhľadanie spôsobí zmenu tohto bitu na 1, je prvok presunutý na začiatok zoznamu. Bity jednotlivých prvkov sú inicializované na náhodne a nezávisle.

Pseudokód

```
Inicializácia

      Pre všetky x nastav x.bit na náhodný bit (0 alebo 1)

Pri nájdení prvku x:

      Ak x.bit == 0
            Nastav x.bit = 1
            Presuň x na začiatok zoznamu
      Inak
            Nastav x.bit = 0
```

Teoretické poznatky

Veta 11: Stratégia Bit je 1.75-kompetitívna proti ľubovoľnému nevedomému protivníkovi.

Dôkaz: uvedený v [**7**]

4.11 Stratégia List on List

Popis stratégie

Stratégia List on List (LOL) vznikla na použitie v modeloch s lokalitou (popísaná v kapitole 3.3), kedy vyhľadanie prvku znamená veľkú pravdepodobnosť vyhľadania iného prvku z jeho okolia. Myšlienkou tejto stratégia je rozdelenie prvkov zoznamu do podzoznamov

podľa lokality. Pri nájdení prvku je následne vykonané nielen presuporiadanie prvkov v podzozname daného prvku ale aj presuporiadanie podzoznamov v rámci zoznamu.

Stratégia List on List býva označená ako LOL X-Y, kde X je stratégia preusporiadania prvkov vrámci podzoznamu a Y stratégia presuporiadania podzoznamov. Napríklad pri stratégií LOL MTF-MTF rozdelíme zoznam dĺžky *n* na *k* podzoznamov dĺžky *m*. Pri nájdení prvku je tento prvok presunutý na začiatok svojho podzoznamu a celý tento podzoznam je presunutý na začiatok zoznamu.

Dôležitou súčasťou stratégie LOL je efektívne rozdelenie zoznamu na podzoznamy. Deliaci algoritmus je spustený na aktuálne vyhľadávanom prvku (v čase t) a na predchádzajúcom vyhľadávanom prvku (v čase t-1). Ak tieto dva prvky patria do rovnakej priehradky internej reprezentácie deliaceho algoritmu, je táto operácia ,,odmenená“ a stratégia LOL vykoná preskupenie. V opačnom prípade dôjde k ,,penalizácií“ a namiesto preskupenia je vykonané nové rozdelenie prvkov do podzoznamov.

Pseudokód

```
Značenie

    q: dotaz na prvok
    Rq: prvok zodpovedajúci dotazu q
    L(Rq): podzoznam, do ktorého patrí Rq
    X: stratégia použitá na úrovni prvkov (v podzoznamoch)
    Y: stratégia použitá na úrovni podzoznamov
    Z: stratégia na rozdelenie prvkov do podzonamov

Opakuj

     dotaz na prvok q
     q1 = q2 //posledný dotaz na prvok
     q2 = q //aktuálny dotaz na prvok
     Vyhľadaj Rq2
     Ak nejde o prvé vyhľadanie
           Spusti Z(q1,q2)
           ak Odmena
                 Spusti X(Rq2,L(Rq2))
                 Spusti Y(L(Rq2))
```

```
inak
      Preskup Rq1 alebo Rq2 podľa rozdelenia daného Z
```

Teoretické vlastnosti pre túto triedu stratégií nie sú známe (autori sa v práci [**8**] venovali výhradne experimentom), zložitosť LOL je však závislá od zložitosti použitého deliaceho algoritmu. Autori pri experimentoch použili deliace algoritmy object migration automaton (OMA) [**26**] a modified linear reward-penalty reinforcment rule (ML_{RP}) [**27**].

4.12 Ďalšie stratégie

Okrem popísaných stratégií existujú aj ďalšie stratégie, ktoré sa zvyknú označovať ako meta-stratégie. Jedná sa väčšinou o základné stratégie kombinované s pravdepodobnostnými alebo inými pravidlami, ktorých cieľom je spomaliť konvergenciu permutačného algoritmu kvôli zníženiu efektov pri jednorazovom prístupe k prvku. Príkladom môže byť stratégia Move-Every-kth-Access navrhnutá v [**24**], ktorá vyvoláva permutačné pravidlo po každom k-tom prístupe alebo K-in-a-row navrhnutá v [**21**], kde sa permutačné pravidlo spustí po *k* prístupoch k prvku.

Hybridné stratégie sú stratégie, ktoré kombinujú základné popísané stratégie. Príkladom môže byť stratégia navrhovaná v [**12**], keď je použitá stratégia Move-to-front až do priblíženia sa stabilného stavu a následne prepnutie na stratégiu Transpose. Problémom však je určiť moment, kedy má k tomuto prepnutiu dôjsť. Podobne v [**22**] je navrhnutá kombinácia stratégií Transpose a Count, a to použitie stratégie Transpose do dosiahnutia stabilného stavu a následne použitie stratégie Count.

V [**6**] nájdeme stratégiu Comb, ktorá kombinuje stratégie Bit a Timestamp na základe pravdepodobnosti. S pravdepodobnosťou 4/5 je použité permutačné pravidlo stratégie Bit a s pravdepodobnosťou 1/5 permutačné pravidlo stratégie Timestamp. O stratégií Comb je dokázané, že je 1.6 kompetitívna.

4.13 Teoretické porovnanie

Porovnať jednotlivé stratégie samoupravujúcich zoznamov na základe ich teoretických výsledkov nie je vôbec jednoduché, je to často možné len na základe kompetitívneho faktoru.

Kompetitívny faktor sa však pre stratégie Jump a Move-ahead-k v literatúre vôbec neuvádza a pre stratégie Transpose a Count bolo dokázané, že nie sú c-kompetitívne pre žiadne *c*. Zvyšné stratégie je možné podľa kompetitívneho faktoru zoradiť do nasledujúcej tabuľky.

Stratégia	**Kompetitívny faktor**
Timestamp(0.382)	1.6181
Bit	1.75
Split	1.875
Move-to-front, Timestamp(0)	2

Tabuľka 1 Zoradenie stratégií podľa komeptitívneho faktoru

4.14 Ďalšie úlohy pre samoupravujúce zoznamy

Klasické samoupravujúce stratégie vykonávajú presuny prvkov v zozname v momente, keď je prvok úspešne nájdený (niekedy sa toto pravidlo aplikuje aj v okamžiku vloženia prvku).

Existujú algoritmy, ktoré vykonávajú nejaké úpravy v zozname i po neúspešnej operácií Member, Insert alebo Delete. Ich cieľom je zrýchliť neúspešné vyhľadávanie, príkladom je algoritmus MP popísaný v [**18**].

Práca [**1**] sa zaoberá možnosťou samoupravujúcich zoznamov vidieť dopredu a dokazuje, že takto upravená stratégia Count dosahuje menší ako 2-kompetitívny faktor.

Albers sa v práci [**1**] zaoberá otázkou zlepšenia kompetitívneho faktoru samoupravujúcich stratégií za predpokladu slabého a silného výhľadu. Slabý výhľad o veľkosti *l* definuje ako schopnosť stratégie vidieť nielen aktuálny dotaz na prvok ale aj nasledujúcich *l* dotazov, silný výhľad veľkosti *l* ako schopnosť stratégie vidieť nasledujúcich *k* dotazov na *l* rôznych prvkov. Práca dokazuje zlepšenie kompetitívneho faktoru *c* stratégie Count so silným výhľadom $l \leq n-1$ na $c \leq 2 - \frac{2}{3}\frac{l+2}{2n-l}$.

Schulz v práci [**31**] predstavuje dve nové triedy deterministických stratégií nazvaných Sort-By-Rank(α) a Sort-By-Delay(k) a dokazuje, že Sort-By-Rank(0) a Sort-By-Delay(1) sú

ekvivalentné stratégií Move-To-Front, Sort-By-Rank(1) a Sort-By-Delay(2) stratégií Timestamp a Sort-By-Delay(k) je k-kompetitívna stratégia pre $k \geq 2$.

5 Metodika testovania

V tejto kapitole uvedieme prehľad predchádzajúcich experimentov, stanovíme cieľ našich experimentov. Vysvetlíme princíp testovania, teda spôsob generovania testovacích dát a spôsob merania času. Popíšeme jednotlivé pravdepodobnostné rozdelenia, ktoré sme v práci zvolili.

5.1 Prechádzajúce experimenty

Zatiaľ čo teoretická práca v oblasti samoupravujúcich zoznamov je pomerne rozsiahla, experimentom bola venovaná relatívne menšia pozornosť. Niekoľko experimentov nájdeme v prácach [**30**], [**11**], [**34**], [**3**], [**8**] a [**9**].

V práci [**30**] Rivest vykonal experiment, kedy sa na prvky v zozname spustilo vyhľadávanie postupnosti prvkov riadiacej sa Zipfovym zákon. V tomto experimente dosiahla stratégia Transpose lepšie výsledky[1] ako stratégia Move-to-front.

Bentley a McGeoh v práci [**11**] testovali vyhľadávanie postupnosti prvkov zodpovedajúcej sekvencii slov v textových súboroch a zdrojových súboroch jazyka Pascal. V ich experimente však stratégia Transpose dosiahla naopak horší výsledok[2] ako stratégia Move-to-front.

Trochu iný prístup zvolili Zobel, Heinz a Williams v [**34**]. Na experimenty použili namiesto zoznamu hešovaciu tabuľku s metódou separovaných reťazcov a namiesto malých súborov použili veľké súbory anglických slov. Distribúcia však taktiež zodpovedala Zipfovmu zákonu. V ich experimentoch dosiahla stratégia Move-to-front výrazne lepšie výsledky ako klasická stratégia pri tabuľkách s vysokým faktorom naplnenia. Pri nižšom faktore naplnenia už nebol rozdiel taký výrazný, stratégia Move-to-front bola však stále rýchlejšia.

V práci [**9**] vykonali Bachrach a El-Yaniv rozsiahly experiment, kde na zozname a vyhľadávacej postupnosti z rozsiahleho korpusu porovnávali samoupravujúce stratégie. Rovnako ako v [**11**] dosiahla stratégia Move-to-front lepšie výsledky ako stratégia Transpose.

[1] Pod pojmom lepšie výsledky rozumieme, že stratégia bola rýchlejšia
[2] Pod pojmom horšie výsledky rozumieme, že stratégia bola pomalšia

Práca [**3**] skúma správanie samoupravujúcich stratégií na zoznamoch s lokalitou, Albers a Laurer sa okrem teoretických zložitostí zaoberajú aj experimentmi, ktoré vykonali s použitím známych knižníc na testovanie výkonu. Ich zistením je, že v praxi, najmä pri výskyte lokality, dosahujú jednotlivé samoupravujúce stratégie nižšie kompetitívne faktory ako tie, ktoré sú pre nich teoreticky dokázané.

Amer a Oommen v novšej práci [**8**] navrhujú nový prístup lokalite, a to použitie stratégie List on List (LOL). Podľa ich experimentov dosahuje tento prístup, konkrétne stratégie MTF-MTF-OMA a MTF-MTF-MLRP, výrazne lepšie výsledky ako stratégia Move-to-front.

Práca [**16**] predstavuje všeobecnú techniku na zrýchlenie neúspešného vyhľadávania. Táto technika vyžaduje málo miesta (dva bity) a môže byť použitá spojové zoznamy ale aj vyhľadávacie stromy. Hui a Martel potom v práci [**15**] rozširujú túto techniku aj na mazanie a predstavujú stratégiu MP, ktorá je 6-kompetitívna pri operáciách úspešného aj neúspešného vyhľadávania, vkladania aj mazania prvkov.

5.2 Cieľ experimentov

Predchádzajúce experimenty prebiehali prevažne na textových súboroch obsahujúcich texty anglického jazyka, my pracujeme s číselnými množinami, pretože v praxi sa nepracuje iba s charakterovými kľúčmi, ale aj s ich číselnými kódmi. Príkladom môžu byť databázové aplikácie, kde je značná časť operácií vykonávaná s umelými číselnými kľúčmi namiesto s nimi asociovanými dátami.

Všetky experimenty sme vykonávali na simulovaných dátach, čo nám umožnilo použiť rôzne typy pravdepodobnostných rozdelení dotazov, rôzne dĺžky zoznamov a vyhľadávacích množín a pod.

Zipfovo rozdelenie používané v predchádzajúcich experimentoch pri našich testoch používať nebudeme, pretože sa hodí prevažne na prácu s textom.

5.3 Voľba typu kľúča a dĺžky zoznamu

Testovanie na celočíselných množinách má ešte jeden dôvod. Porovnanie celočíselných kľúčov je časovo oveľa menej náročné, ako porovnávanie slov (textových reťazcov). Na samotnom vyhľadávaní sa teda na spotrebovanom čase prejavuje porovnávanie

celočíselných kľúčov menej a väčšiu šancu prejaviť sa majú ostatné operácie, ktoré daná stratégia vykonáva. Zatiaľ čo pri vyhľadávaní slov je výhodné použiť algoritmy, ktoré znižujú počet porovnaní aj za cenu pridania ďalších časovo menej náročných operácií, pri vyhľadávaní celočíselných kľúčov to už nemusí byť pravda. Vo výsledkoch si preto budeme všímať, či skutočne spotrebovaný čas zodpovedá počtu porovnaní alebo má odlišný priebeh.

Pri teste konvergencie a vyhľadávania sme najprv zvoli zoznamy dĺžky 1000 prvkov. Experimenty s dlhšími zoznamami by síce boli lepšie, problém však je, aby použité pravdepodobnostné rozdelenie pokrývalo väčšiu časť testovanej množiny a nebolo príliš ploché. Splniť túto požiadavku bolo pri zoznamoch väčších dĺžok veľmi ťažké a musel by byť použitý len minimálny počet pravdepodobnostných rozdelení. Zvolili sme preto radšej kratší zoznam a viacero pravdepodobnostných rozdelení.

Pri druhom teste vyhľadávania na dlhých zoznamoch sme použili vyhľadávacie množiny vygenerované pre predchádzajúci experiment na zoznamoch dĺžky 10 000. Chceli sme simulovať situáciu, kedy je vyhľadávaná len časť prvkov zoznamu (v tomto prípade desatina). Takáto situácie môže občas nastať v praxi, hlavne sme však jednotlivým stratégiám chceli dať možnosť, aby sa rozdiely medzi nimi prejavili výraznejšie, ako je to možné na krátkych zoznamoch.

5.4 Testované operácie

Testovali sme samotnú operáciu úspešný Member. Pri teste sme jednotlivé zoznamy vždy najprv naplnili testovacou množinou (operácia Insert, záznamy vkladané vždy na koniec zoznamu) a následne spustili vyhľadávanie prvkov (operácia Member). Pri oboch operáciách sme merali počet vykonaných porovnaní a čas.

5.5 Potrebné dáta

Jedným zo základných predpokladov úspešného testovania je dostatočný počet dostatočne rôznorodých dát. Rozhodli sme sa testovacie dáta vygenerovať v programe Matlab a uložiť ich do súborov, aby bolo možné ich opakované použitie.

Pri teste konvergencie a teste vyhľadávania budeme potrebovať dve množiny dát: testovaciu množinu, t.j. zoznam prvkov, na ktorý bude aplikovaná postupnosť vyhľadávacích operácií a vyhľadávaciu množinu, t.j. množinu prvkov, ktoré budeme brať (v danom poradí) ako argumenty operácie Member. Predvolená testovacia množina pre všetky stratégie bude náhodná permutácia čísel 1..1000 (pre testy na dlhých zoznamoch to bude náhodná permutácia 1..10000). Vyhľadávacia množina bude obsahovať tie isté prvky ako testovacia množina (alebo bude jej podmnožina), ale bude vygenerovaná z požadovaného pravdepodobnostného rozdelenia. Jej veľkosť bude obvykle rádovo väčšia ako veľkosť testovacej množiny, takže jednotlivé prvky sa v nej budú vyskytovať opakovane s početnosťami zodpovedajúcimi pravdepodobnostiam.

5.6 Pravdepodobnostné rozdelenia

Dôležitou súčasťou experimentov bola voľba použitých pravdepodobnostných rozdelení. Rovnomerné, normálne, geometrické, binomické a Poissonovo rozdelenie patria medzi základné pravdepodobnostné rozdelenia, bolo teda prirodzené začať práve nimi.

Vzhľadom na princíp fungovania samoupravujúcich stratégií sme do experimentov chceli zahrnúť hlavne pravdepodobnostné rozdelenia, ktoré majú výrazne nerovnomerný priebeh, doplnili sme preto beta rozdelenie a logaritmicko-normálne rozdelenie. Rovnomerné rozdelenie je použité hlavne na porovnanie.

Každé použité pravdepodobnostné rozdelenie by malo pokrývať určitú časť testovacej množiny.

Rovnomerné rozdelenie

Rovnomerné rozdelenie je rozdelenie pravdepodobnosti, v ktorom má každé číslo daného intervalu rovnakú pravdepodobnosť. Má dve verzie, a to diskrétnu a spojitú. Pri experimentoch sme použili spojitú verziu.

Hustota rovnomerného spojitého rozdelenia je

$$f(x) = \begin{cases} \frac{1}{|I|} & pre\ x \in I \\ 0 & inak \end{cases}$$

Kde *I* je interval *R*.

Normálne rozdelenie

Normálne rozdelenie so strednou hodnotou $\mu \in R$ a rozptylom σ^2, kde $\sigma > 0$ je spojité pravdepodobnostné rozdelenie s hustotou

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \; pre \; x \in R$$

Geometrické rozdelenie

Geometrické rozdelenie s parametrom $p \, \epsilon \, (0,1)$ je diskrétne pravdepodobnostné rozdelenie s pravdepodobnostnou funkciou

$$P(k) = p(1-p)^k \; pre \; k = 0,1 \ldots$$

Logitmicko-normálne rozdelenie

Logaritmicko-normálne rozdelenie s parametrami $\mu \in R$ a σ^2, kde $\sigma > 0$ je spojité pravdepodobnostné rozdelenie s hustotou

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x-\mu)^2/2\sigma^2} \; pre \; x > 0$$

Beta rozdelenie

Beta rozdelenie s parametrami $A > 0$, $B > 0$ je spojité pravdepodobnostné rozdelenie s hustotou

$$f(x) = \frac{x^{A-1}(1-x)^{B-1}}{\int_0^1 u^{A-1}(1-u)^{B-1}du} \; pre \; 0 < x < 1$$

Poissonovo rozdelenie

Poissonovo rozdelenie s parametrom $\lambda > 0$ je diskrétne rozdelenie na množine celých nezáporných čísel s pravdepodobnostnou funkciou

$$P(k) = \begin{cases} \frac{e^{-\lambda}\lambda^k}{k!} & pre\ k = 0,1 \dots n \\ 0 & inak \end{cases}$$

Binomické rozdelenie

Binomické rozdelenie s parametrami $n \in N$a $p \in (0,1)$ je diskrétne rozdelenie na množine {0,1,…,n} s pravdepodobnostnou funkciou

$$P(k) = \begin{cases} \binom{n}{k} p^k (1-p)^{n-k} & pre\ k = 0,1 \dots n \\ 0 & inak \end{cases}$$

5.7 Generovanie dát

Pre každé z použitých rozdelení sme vygenerovali 16 dvojíc testovacia množina a vyhľadávacia množina, celkovo teda 80 množín. Tieto testovacie súbory boli použité pri všetkých testoch a všetkých stratégiách.

Voľba použitých pravdepodobnostných rozdelení sa ukázala náročnejšia, ako sme si pôvodne mysleli. Bolo potrebné zvoliť také pravdepodobnostné rozdelenia, aby podľa nich vygenerované prvky mali dostatočný rozsah hodnôt. Pri Poissonovom a binomickom rozdelení nebol tento spôsob možný, pretože naťahovaním dát na požadovaný rozsah zväčšovaním parametrov sa strácajú vlastnosti daného rozdelenia a dáta sa približujú rozdeleniu normálnemu, preto sme ich v experimentoch nepoužili.

Pre rovnomerné rozdelenie sme pri generovaní použili pseudonáhodný generátor. Tento generátor generuje čísla zo spojitého intervalu (0,1), označme R(0,1). Ak chceme tento interval previesť na spojitý interval interval r=(a,b), môžeme použiť nasledujúci prevod:

```
r = a + (b-a).* R(0,1)
```

Pre geometrické rozdelenie, normálne rozdelenie, beta rozdelenie a logaritmické rozdelenie boli parametre jednotlivých rozdelení volené tak, aby vygenerované dáta pokryli množinu prvkov 1..1000.

Vygenerované testovacie množiny boli pri všetkých pravdepodobnostných rozdeleniach spojité, bolo ich ešte potrebné previesť na diskrétny interval 1..1000. K tomu sme použili zaokrúhlenie smerom nadol. Všetky dáta boli vygenerované programom Matlab [**33**].

Pri testoch konvergencie a vyhľadávania na krátkych zoznamoch sme použili vždy množiny veľkosti 1000 prvkov a počet vyhľadávaní 500 000 (vygenerované podľa daného pravdepodobnostného rozdelenia). Pri teste vyhľadávania na dlhých zoznamoch bola ako testovacia množina vždy použitá náhodná permutácia čísel 1..10000.

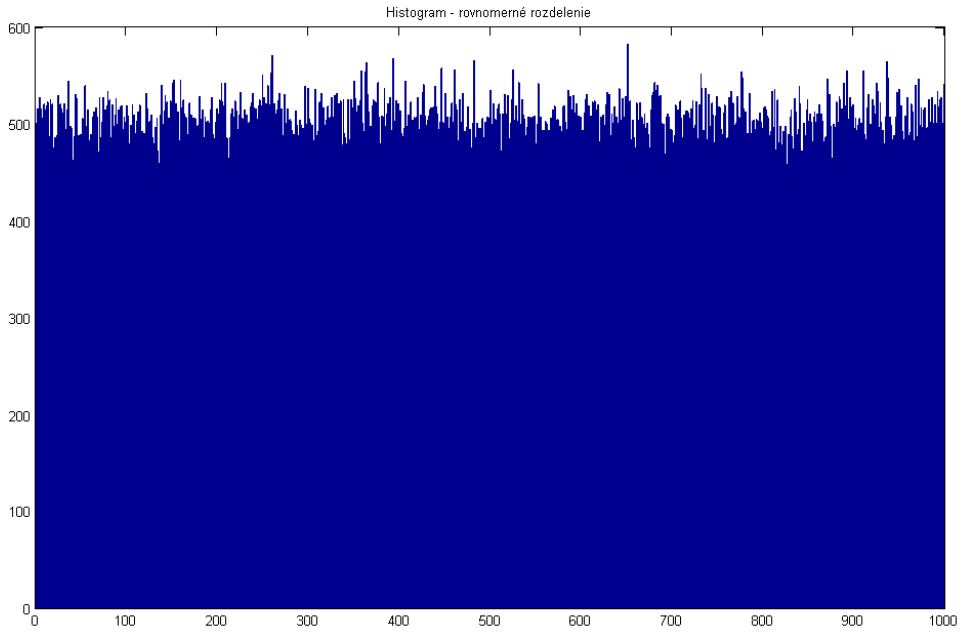
Všetky testovacie dáta boli uložené do súborov kvôli možnosti jednoduchého opakovania ktoréhokoľvek testu.

5.8 Vygenerované dáta

Pre lepšiu predstavu o vygenerovaných dátach uvádzame histogramy vyhľadávaných množín a na porovnanie aj hustoty daných pravdepodobnostných rozdelení. Vygenerované vyhľadávané množiny obsahovali vždy presne 500 000 prvkov.

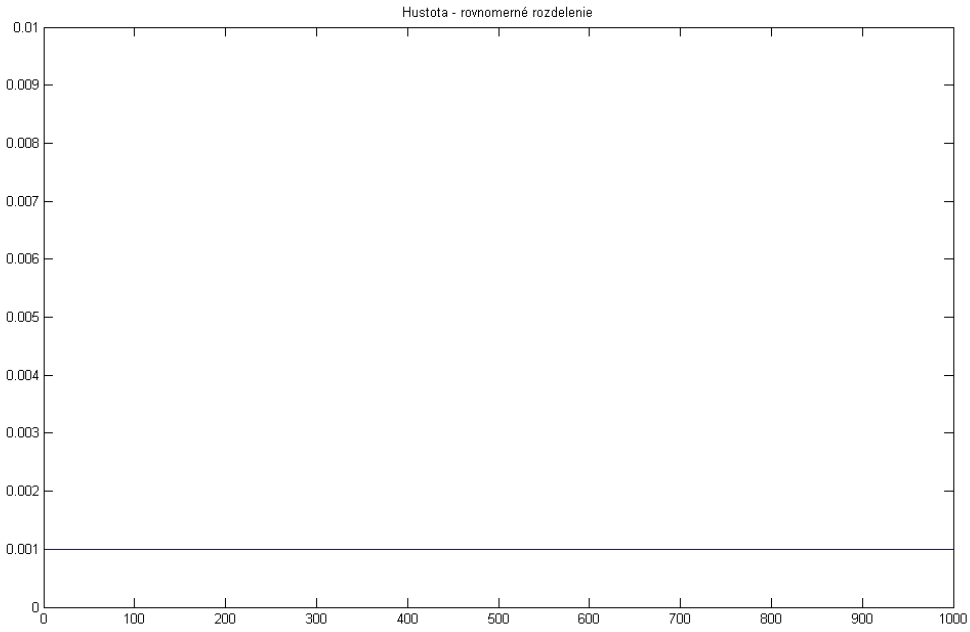
Rovnomerné rozdelenie

Testovacie dáta pre rovnomerné rozdelenie sme vygenerovali pomocou pseudonáhodného generátora.



Obrázok 4 Histogram rovnomerného rozdelenia

Z histogramu je vidieť, že vygenerované dáta zodpovedajú základnej charakteristike rovnomerného rozdelenia. Početnosť vyhľadávaní je pre všetky prvky testovacej množiny približne rovnaká, kolíše s malými odchýlkami okolo hodnoty 500.



Obrázok 5 Hustota rovnomerného rozdelenia

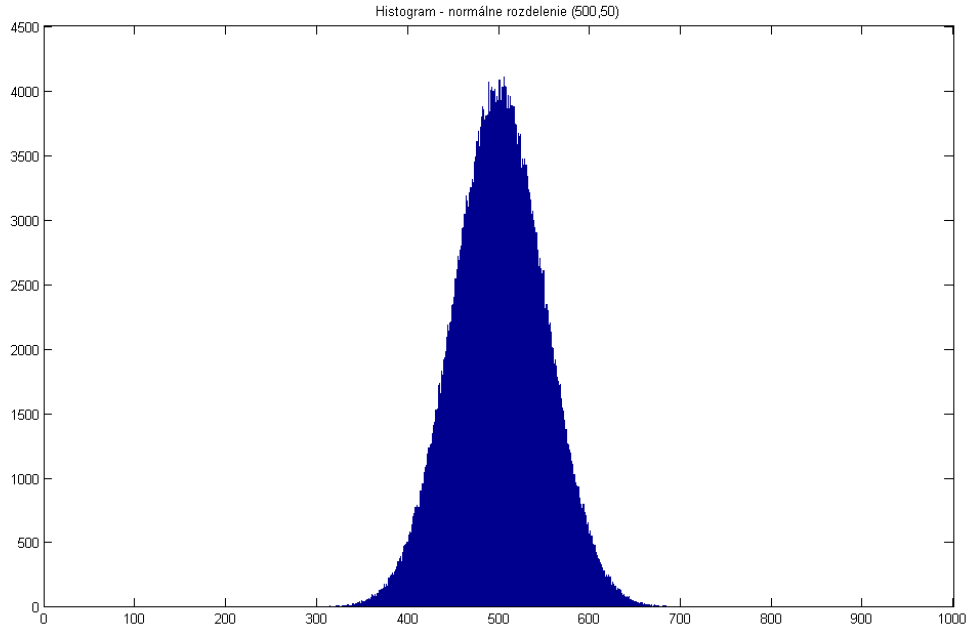
Hustota rovnomerného rozdelenia potvrdzuje fakt, že všetky prvky intervalu sú v tomto pravdepodobnostnom rozdelení generované s rovnakou pravdepodobnosťou. Hodnota pravdepodobnosti 0.001 pri 500 000 vygenerovaných prvkoch zodpovedá početnosti výskytu jedného prvku 500.

Normálne rozdelenie

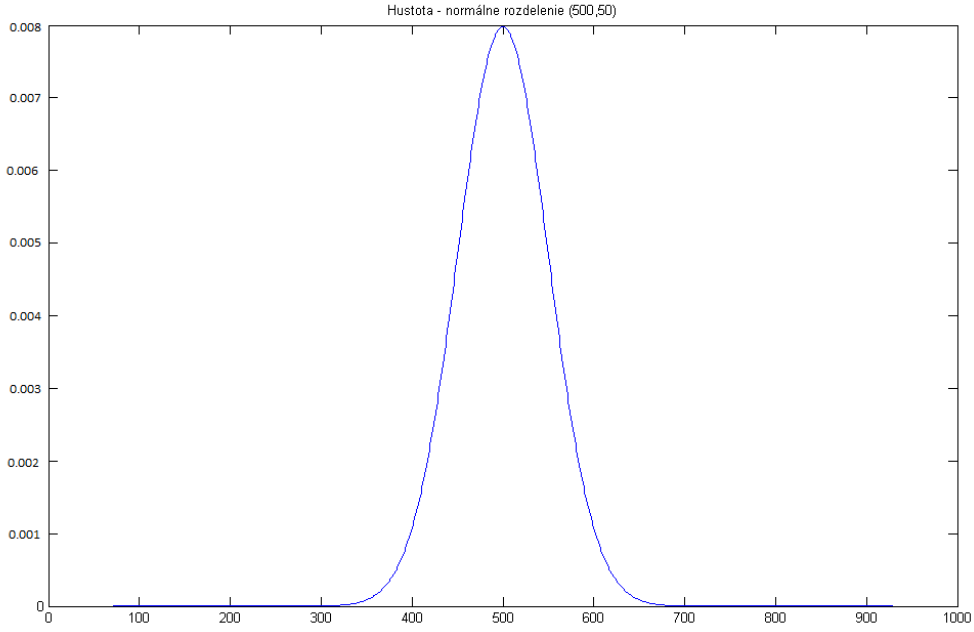
Testovacie dáta pre normálne rozdelenie sme vygenerovali s parametrami $\mu = 500$ a $\sigma = 50$.

Pri danej voľbe parametrov pokrývajú vygenerované dáta celý rozsah testovacej množiny. S najväčšou početnosťou, približne 4000, sú zastúpené prvky okolo bodu 500. Smerom k okrajom intervalu početnosti symetricky klesajú až takmer k nulovým hodnotám, teda najväčšie a najmenšie prvky budú v experimentoch vyhľadávané len minimálne.

Hustota normálneho rozdelenia udáva, že prvky v strede intervalu (okolo bodu 500) sú generované s pravdepodobnosťou 0.008, čo pri 500 000 prvkoch zodpovedá 4000 prvkom v histograme.



Obrázok 6 Histogram normálneho rozdelenia



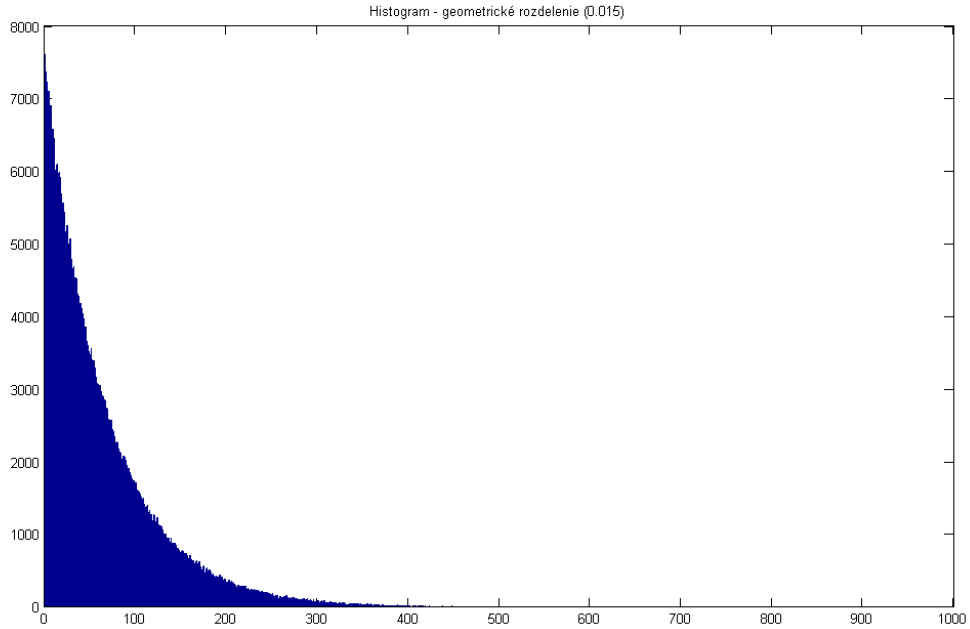
Obrázok 7 Hustota normálneho rozdelenia

Geometrické rozdelenie

Testovacie dáta pre geometrické rozdelenie sme vygenerovali s parametrom p=0.013.

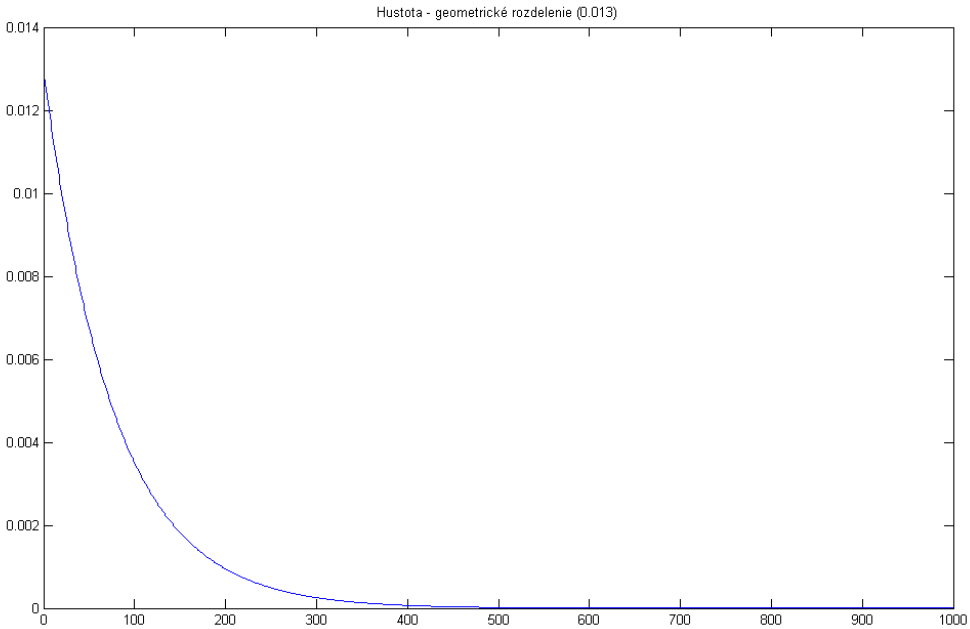
Aj pri geometrickom rozdelení voľba hodnoty parametrov zaručuje, že vygenerované dáta pokrývajú celý zvolený rozsah. Na rozdiel od normálneho rozdelenia budú s najväčšou početnosťou, približne 7000, zastúpené najmenšie prvky s hodnotami menšími ako 10.

Tieto početnosti budú ďalej strmo klesať, takže prvky s hodnotami väčšími ako 500 už budú pri experimentoch vyhľadávané len minimálne.



Obrázok 8 Histogram geometrického rozdelenia

Hustota geometrického rozdelenia má klesajúci priebeh, najväčšiu pravdepodobnosť ocitnúť sa vo vygenerovanej množine, okolo 0.014, majú prvky zo začiatku intervalu, čo pri 500 000 prvkoch zodpovedá 7000 prvkom v histograme.

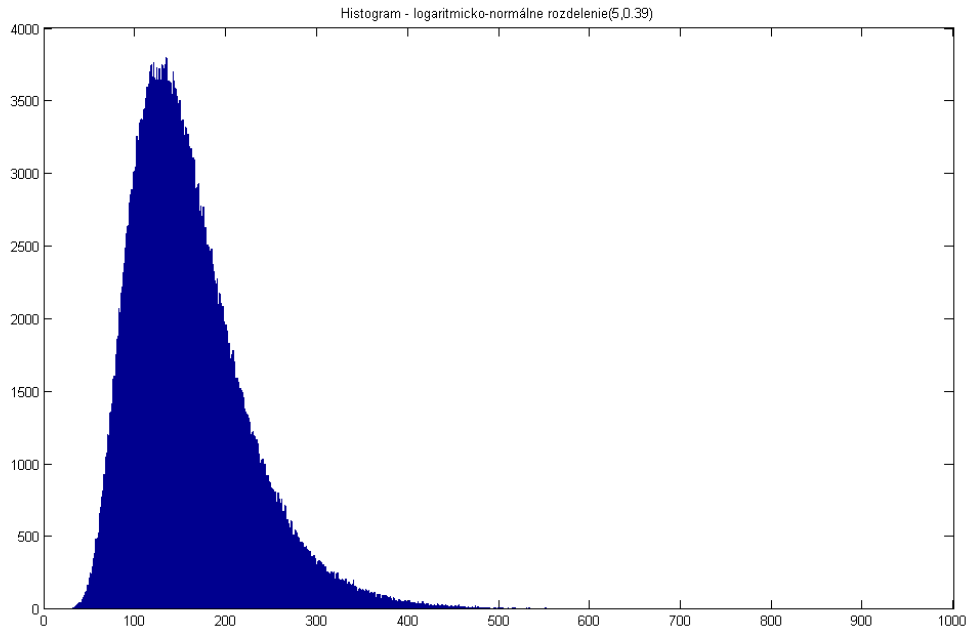


Obrázok 9 Hustota geometrického rozdelenia

Logaritmicko-normálne rozdelenie

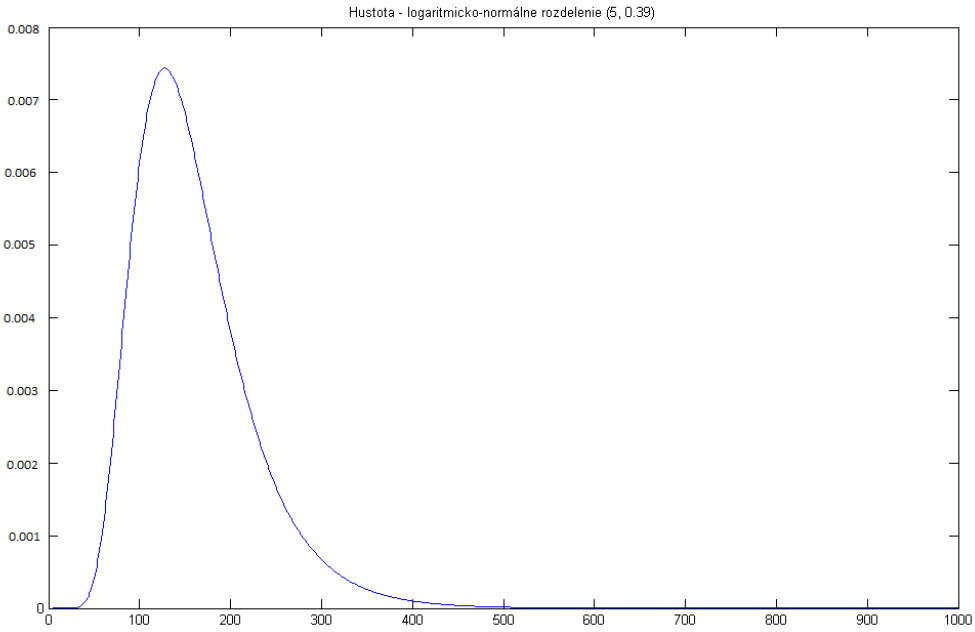
Testovacie dáta pre logaritmicko-normálne rozdelenie sme vygenerovali s parametrami $\mu = 5$ a $\sigma = 0.39$.

Vhodná voľba parametrov zaručovala podobne ako pri geometrickom rozdelení pokrytie celého rozsahu základnej testovacej množiny aj pri logaritmicko-normálnom rozdelení. S najväčšou početnosťou sa budú vyhľadávať taktiež prvky menšie ako 500. Rozdiel je však v tom, že najväčšie početnosti nebudú mať úplne najmenšie prvky, ale vrchol rozdelenia je posunutý do intervalu 100 až 200. Od normálneho rozdelenia sa toto pravdepodobnostné rozdelenie líši tým, že je nesymetrické.



Obrázok 10 Histogram logaritmického rozdelenia

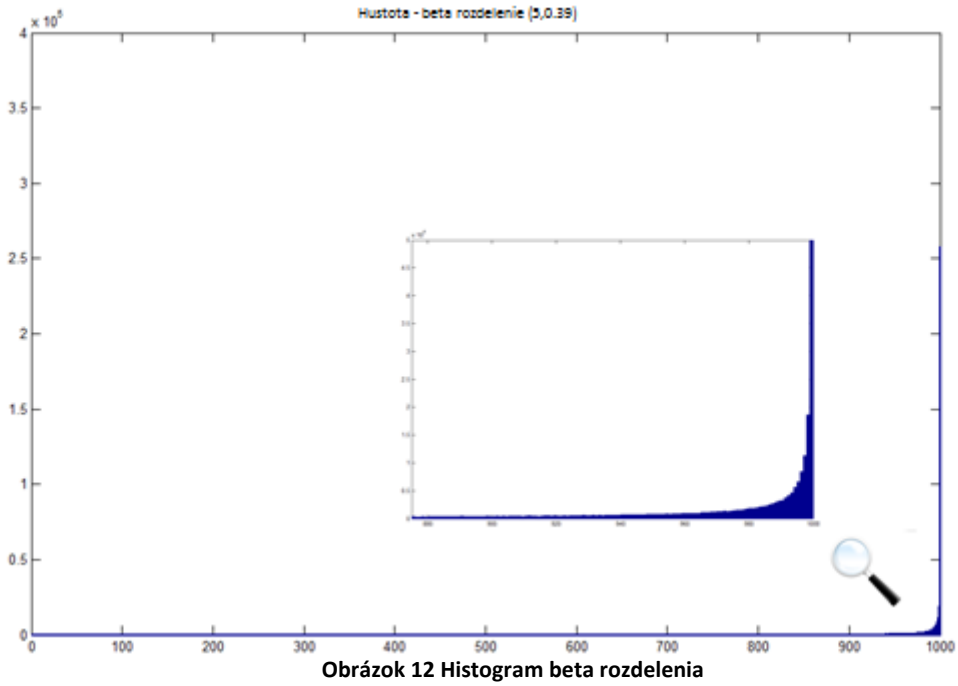
Hustota logaritmicko-normálneho rozdelenia určuje, že prvky z intervalu 100 až 200 budú vygenerované s pravdepodobnosťou 0.0075, čo pri 500 000 prvkoch zodpovedá 3750 prvkov v histograme.



Obrázok 11 Hustota logaritmicko-normálneho rozdelenia

Beta rozdelenie

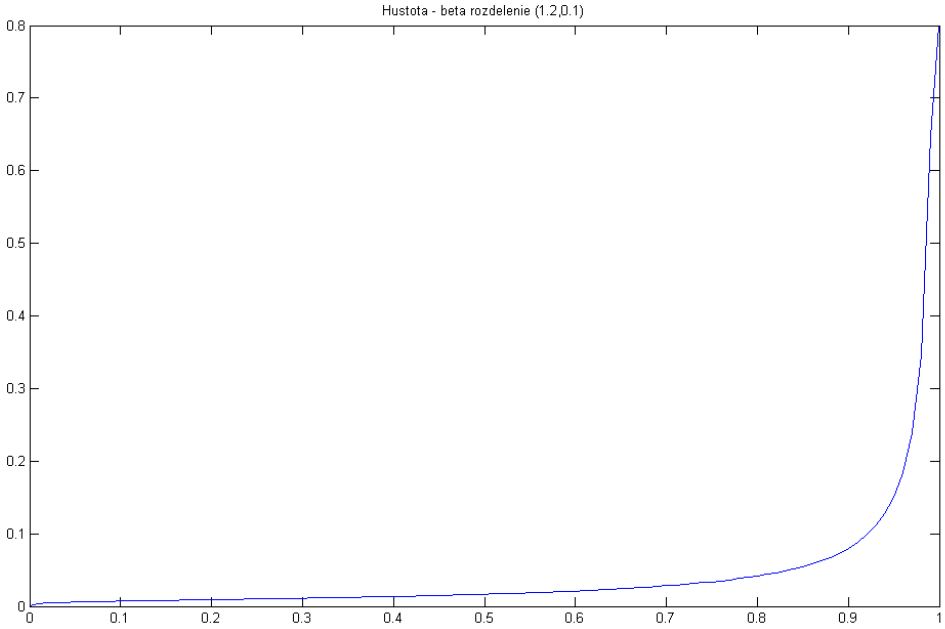
Testovacie dáta pre beta rozdelenie sme vygenerovali s parametrami A=1.2 a B=0.1 a vynásobili konštantou 1000.



Obrázok 12 Histogram beta rozdelenia

Priebeh beta rozdelenia je opačný ako priebeh geometrického rozdelenia. S najväčšou početnosťou sú generované a vyhľadávané najväčšie prvky v rozsahu 900 až 1000. Pokles

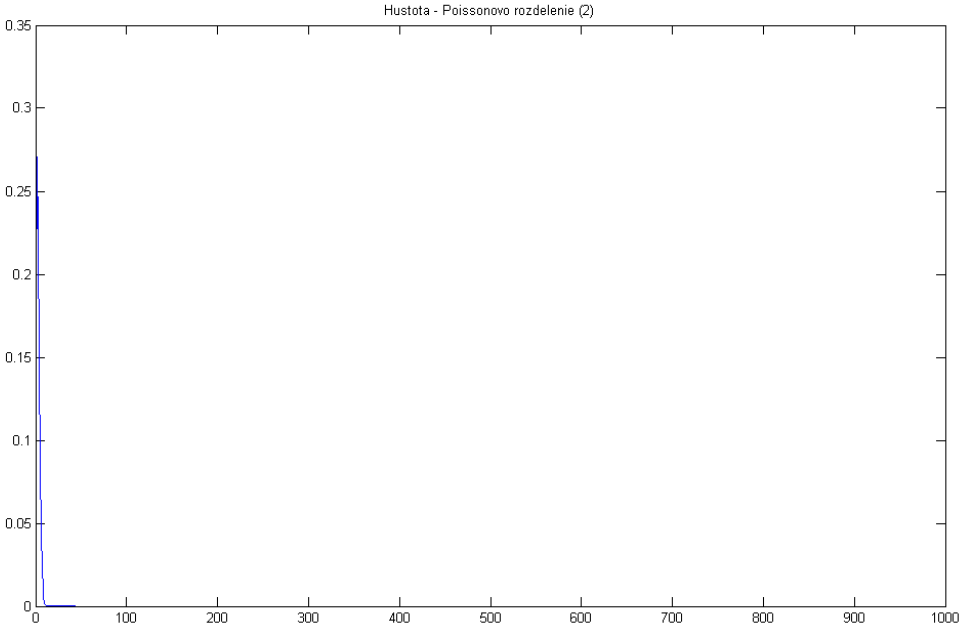
početnosti smerom k menším hodnotám je tu však ešte výraznejší. Na Obrázku 12 je v histograme zobrazený naviac detail konca intervalu zväčšený 10x.



Obrázok 13 Hustota beta rozdelenia

S najväčšou pravdepodobnosťou sú pri beta rozdelení generované prvky z konca intervalu (0,1) s pravdepodobnosťou 0.8, čo po prevode na interval (0,1000) a pri 500 000 prvkoch zodpovedá 400 000 prvkov v histograme.

Poissonovo rozdelenie

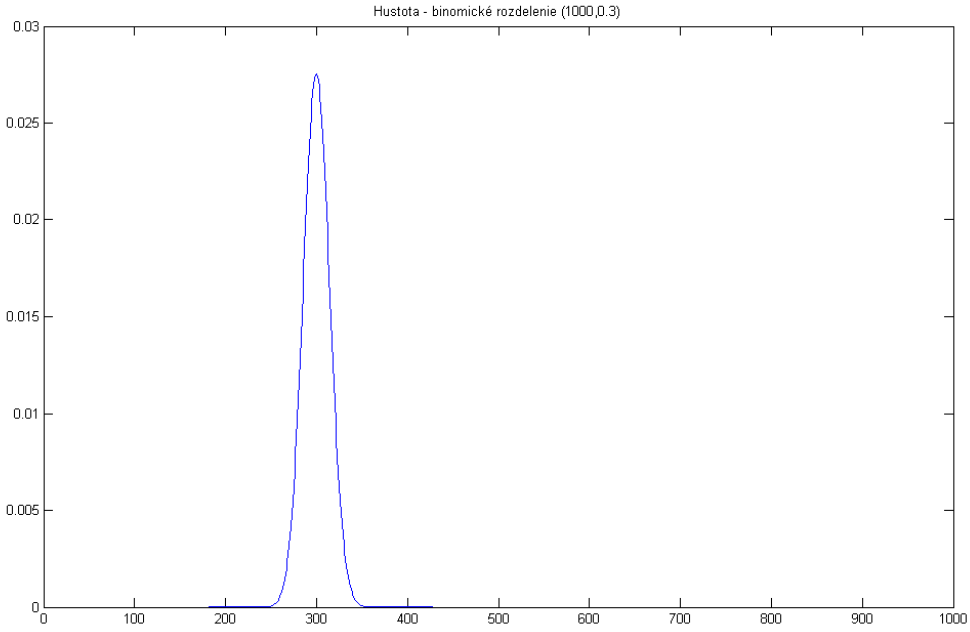


Obrázok 14 Hustota Poissonovho rozdelenia

Pri Poissonovom rozdelení nebolo možné vygenerovať dáta zo zvoleného rozsahu, pretože pri veľkých hodnotách parametra λ sa toto rozdelenie blíži normálnemu rozdeleniu, pri malých hodnotách λ sa generuje len veľmi málo rôznych hodnôt. Poissonovo rozdelenie preto nakoniec v testoch použité nebolo. Na Obrázku 14 je hustota Poissonovho rozdelenia s parametrom $\lambda = 2$.

Binomické rozdelenie

Z toho istého dôvodu pri binomickom rozdelení nebolo možné vygenerovať dáta zo zvoleného rozsahu, čo vyplýva z vlastností tohto rozdelenia (Moivre-Laplaceova lokálna veta hovorí, že pre binomické rozdelenie je pre veľké hodnoty *n* možné aproximovať normálnym rozdelením). Binomické rozdelenie preto nakoniec v testoch použité nebolo. Na Obrázku 15 je hustota binomického rozdelenia s parametrami n=1000, p=0.03.



Obrázok 15 Hustota binomického rozdelenia

5.9 Meranie času

Problematika merania času behu algoritmu nie je až taká jednoduchá, aká by sa mohla zdať. Na tento čas má vplyv viacero faktorov, od paralelne spustených procesov a aplikácií v operačnom systéme, až po aktivitu sieťovej karty, nestačí teda jednoducho zmerať začiatočný a konečný čas a urobiť ich rozdiel.

Je veľmi dôležité, aby boli všetky testy vykonávané vždy na rovnakom počítači (s rovnakým hardvérom a operačným systémom), okrem testov nebežali žiadne iné používateľské procesy a počítač bol odpojený od siete.

Keďže sme sa rozhodli implementovať jednotlivé algoritmy a aj testy v jazyku C# na platforme .NET, na meranie času sme využili triedu StopWatch. Táto trieda meria čas počítaním tikov časovača. Ak nainštalovaný hardvér a operačný systém podporujú časovač s vysokým výkonom, StopWatch použije tento časovač, v opačnom prípade použije časovač operačného systému [**25**]. Na počítači, ktorý sme použili na testy, bol využitý časovač s vysokým výkonom.

5.10 Hardvérová konfigurácia

Procesor: AMD Athlon64 X2 3600+, 2 GHz, FSB 1000 MHz, 256KB L2 cache

Operačná pamäť: 4 GB DDR2 667 MHz

Operačný systém: Windows 7 x64 Professional SP1

Prekladač: Visual Studio 2008 SP (.NET 3.5 SP1)

6 Testovanie

V tejto kapitole uvedieme popis testov, ktoré sme vykonávali a spôsob merania jednotlivých veličín.

6.1 Testované stratégie

Rozhodli sme sa otestovať a porovnať nasledujúce deterministické stratégie:

- Move-to-front
- Transpose
- Count
- Move-ahead-k
- Timestamp(0)

a nedeterministickú stratégiu

- Timestamp(p)

doplnené o zotriedený zoznam a klasický obojsmerný zoznam.

Stratégie Move-to-front a Transpose sme zvolili, pretože ide o najznámejšie a najdetailnejšie preskúmané stratégie.

Stratégia Move-ahead-k je v podstate kompromis oboch týchto stratégií a je teda zaujímavé ju týmito stratégiami porovnať. Pri tejto stratégií sme zvolili parameter závislý na pozícií vyhľadávaného prvku v zozname, pretože táto verzia sa ukázal efektívnejší ako parameter vypočítaný na základe dĺžky testovacej množiny. Na CD však nájdete výsledky stratégie Move-ahead-k s oboma parametrami.

Stratégia Count nesmela v teste chýbať, pretože sa zo všetkých stratégií svojím princípom najviac približuje optimálnemu algoritmu.

Z nedeterministických stratégií sme na testy zvolili najznámejšiu stratégiu Timestamp(p) a na porovnanie otestovali aj jej deterministickú verziu Timestamp(0). V prípade stratégie Timestamp(p) sa do popredia dostáva otázka, ako daný parameter p čo najlepšie zvoliť.

Podľa Vety 9 je ideálne zvoliť p tak, aby c bolo minimálne, teda p=0.382, čím dosiahne stratégia kompetitívny faktor c=1.6181.

Bolo by určite zaujímavé sledovať výsledky nedeterministickej stratégie Timestamp(p) v závislosti na parametri *p*, na to však už v tejto práci nezvýšil čas.

Do testov vyhľadávania bola ešte zahrnutá stratégia OSO, ktorá predstavuje optimálny algoritmus (viď 3.4).

6.2 Typy testov

Konvergencia

Ako základný test sme zvolili test konvergencie danej stratégie. Jeho účelom bolo zistiť, ako rýchlo jednotlivé stratégie konvergujú do stabilného stavu (ak vôbec) a za akých podmienok vstupných dát.

Vyhľadávanie

Po tom, ako sme u jednotlivých stratégií našli stabilný stav, zamerali sme sa na testy vyhľadávania. Pri vyhľadávaní išlo vždy o operáciu úspešný Member. Testovať neúspešný Member nemá zmysel, pretože pri tejto operácií je pri klasickom zozname aj samoupravujúcich stratégiách vždy potrebné prejsť celý zoznam.

Vyhľadávanie v zozname sme chceli testovať od bodu dosiahnutia stabilného stavu, aby bolo možné jednotlivé stratégie priamo porovnať, testovali sme od maxima bodov dosiahnutia stabilného stavu jednotlivých stratégií, teda od bodu, v ktorom sú už všetky stratégie v stabilnom stave.

Vyhľadávanie na dlhých zoznamoch

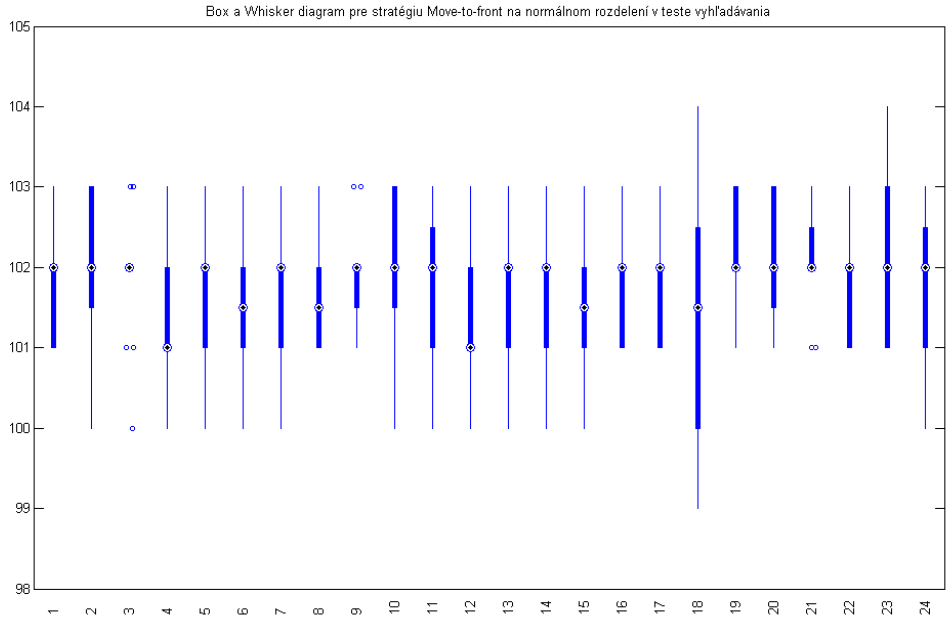
Posledným testom, ktorý sme zvolili, bol test vyhľadávania na dlhých zoznamoch. Rozdiel oproti testu vyhľadávania je v tom, že testovacia množina nebola tvorená náhodnou permutáciou čísel 1..1000, ktoré sa vyskytujú vo vyhľadávacej množine, ale náhodnou permutáciou čísel 1..10000. V tomto prípadne vykonávame rovnako ako v teste vyhľadávania operáciu úspešný Member, nevyhľadávame však všetky prvky, ale len niektoré. Rozdiely medzi jednotlivými samoupravujúcimi stratégiami by sa v tomto prípade mali prejaviť výraznejšie, pretože presun vyhľadávaných prvkov na začiatku zoznamu by mal trvať dlhšie.

6.3 Merané veličiny

Pri všetkých testoch sme merali uplynulý čas a počet porovnaní. Pod počtom porovnaní sa myslí počet porovnaní kľúčov potrebných k nájdeniu požadovaného prvku v zozname.

Počet porovnaní kľúča na nájdenie jedného prvku sme zvolili kvôli porovnaniu s teoretickými výsledkami. Čas by mal vystihovať správanie sa jednotlivých stratégií v praxi, pretože odráža aj náročnosť ďalších operácií, ako sú presuny prvkov v zozname, vyhľadávanie podľa početnosti alebo časových známok atď. Priebeh týchto dvoch charakteristík teda môže byť niekedy odlišný.

Pre každé rozdelenie sme potom zobrali priemer z nameraných hodnôt 16 testovacích dvojíc; testovacia množina a vyhľadávacia množina. Šestnásť dvojíc sa ukázalo ako dostatočné množstvo, pretože rozdiely medzi jednotlivými dvojicami sa v prípade merania počtu porovnaní pohybovali rádovo v jednotkách percent, čo potvrdzuje aj Box a Whisker diagram na Obrázku 16 zostavených z jednotlivých meraní (šestnásť experimentov) stratégie Move-to-front na normálnom rozdelení v teste vyhľadávania na základe mediánov a 25 a 75 percentilu.



Obrázok 16 Box a Whisker diagram pre stratégiu Move-to-front na normálnom rozdelení v teste vyhľadávania

Všetky namerané hodnoty nie sú inkrementálne ale prepočítané na jednu operáciu.

7 Výsledky testov

V siedmej kapitole uvedieme a zanalyzujeme výsledky jednotlivých experimentov. Výsledky testov prezentujeme pre prehľadnosť graficky, namerané hodnoty sú na priloženom CD.

Stratégiu Count budeme v obrázkoch označovať ako Count, klasický zoznam ako General, stratégiu Move-ahead-k vo verzii s k=i/2, kde i je pozícia prvku v zozname, ako MoveAheadK(i/2), stratégiu Move-to-front ako MoveToFront, zotriedený zoznam ako Sorted, deterministickú verziu stratégie Timestamp ako Timestamp(0), nedeterministickú (pravdepodobnostnú) verziu ako Timestamp(0.382), stratégiu Transpose ako Transpose, optimálne statické usporiadanie ako OSO. Stratégie General, OSO a Sorted neboli použité v teste konvergencie ale až v testoch vyhľadávania.

7.1 Konvergencia

Prvým testom, ktorý sme vykonali, bol test konvergencie. Jeho cieľom bolo zistiť, ako rýchlo (a či vôbec) jednotlivé stratégie konvergujú do stabilného stavu.

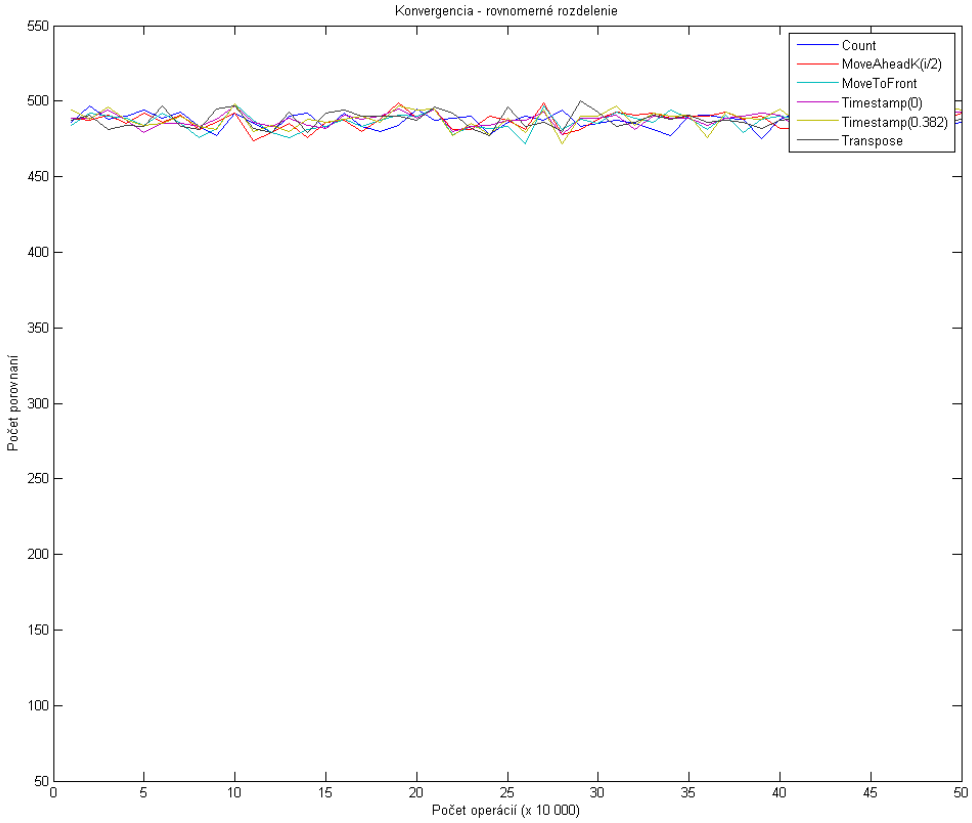
Konvergenciu sme posudzovali na základe počtu porovnaní kľúčov potrebných na nájdenie jedného prvku. Pre zaujímavosť ku každému pravdepodobnostnému rozdeleniu prikladáme aj graf času potrebného na nájdenie jedného prvku.

Namerané hodnoty nájdete na CD v adresári results\convergence a grafy v adresári graphs\convergence.

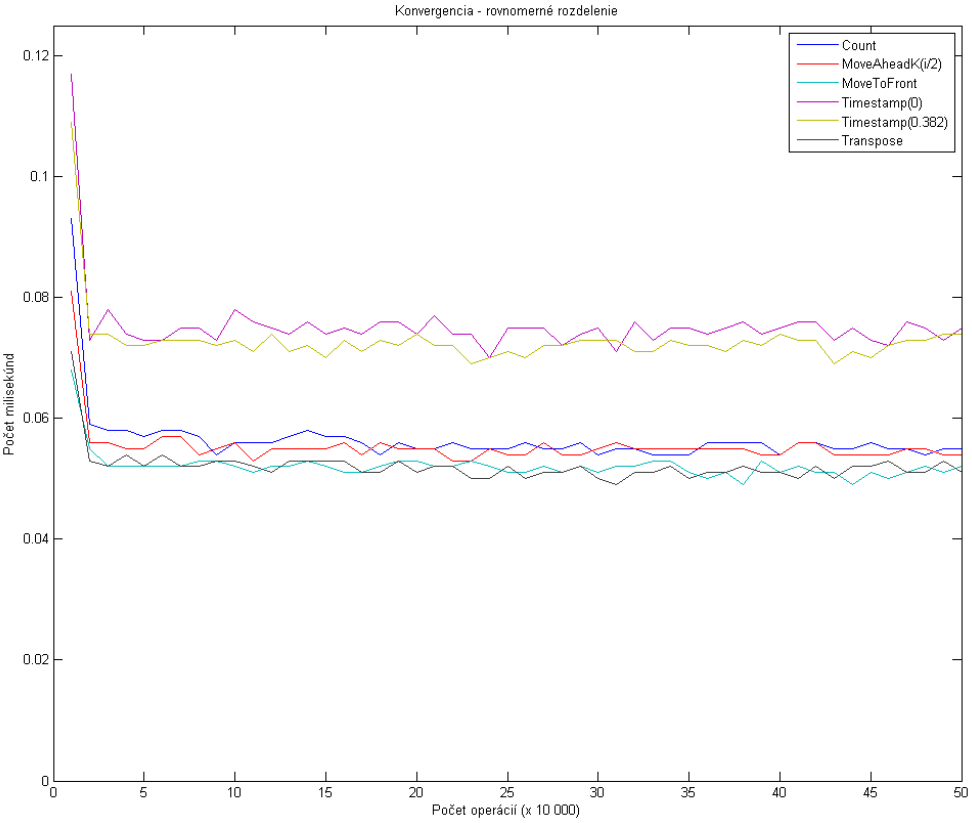
Rovnomerné rozdelenie

Prvým testovaným rozdelením pri teste konvergencie bolo rovnomerné rozdelenie. Pri rovnomernom rozdelení je každý prvok vo vyhľadávacej postupnosti zastúpený rovnako, výhody samoupravujúcich zoznamov by sa teda nijako prejaviť nemali a nemalo by dôjsť ani ku konvergencii žiadnej stratégie.

Všetky stratégie vykonávali stabilne okolo 500 porovnaní pri vyhľadávaní jedného prvku v zozname dĺžky 1000, čo sa zhoduje s očakávanou zložitosťou úspešného vyhľadávania v neusporiadaných zoznamoch pri rovnomernom rozdelení.



Obrázok 17 Konvergencia - rovnomerné rozdelenie - počet porovnaní



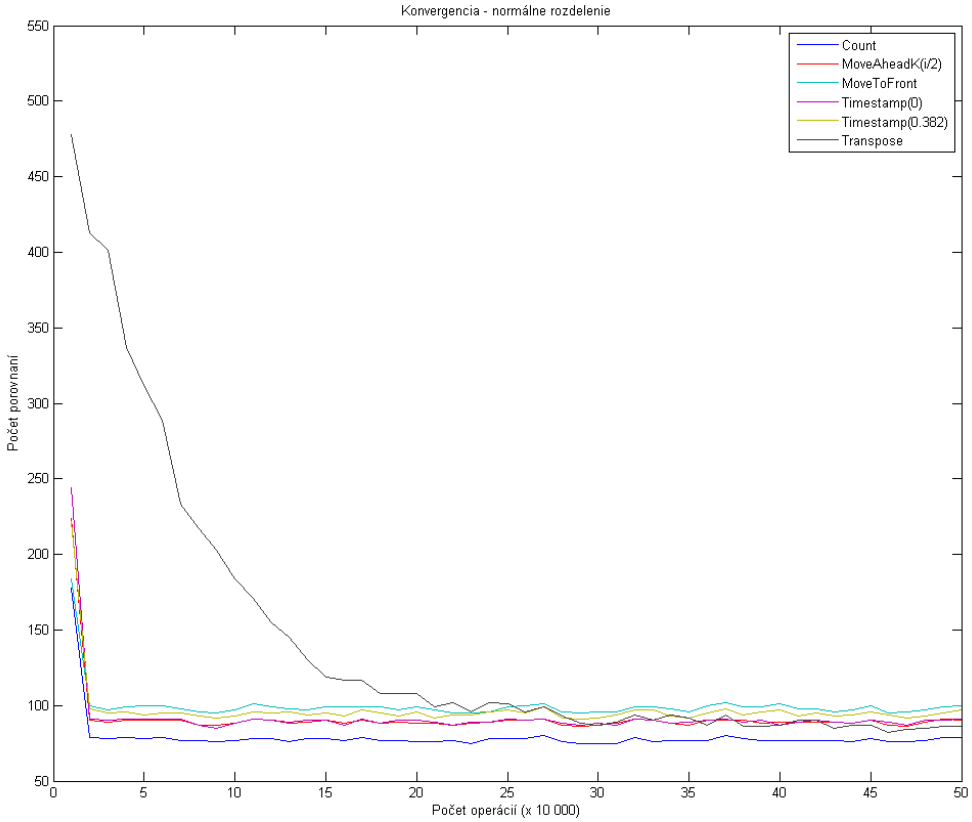
Obrázok 18 Konvergencia - rovnomerné rozdelenie – čas

Pokles stratégií v rozmedzí okolo dvoch stotín milisekundy môže byť spôsobený programovým prostredím. Jednotlivé stratégie sú totiž implementované pomocou dedičnosti a používajú virtuálne metódy, ktorých prvé zavolanie je vždy o niečo pomalšie.

Horší výsledok stratégie Timestamp v oboch svojich verziách je možné odôvodniť jej implementačnou zložitosťou, pri permutácii zoznamu po nájdení prvku totiž vykonáva výrazne časovo náročnejšie operácie ako ostatné samoupravujúce stratégie.

Normálne rozdelenie

Pri normálnom rozdelení poli použité parametre $\mu = 500$ a $\sigma = 50$.

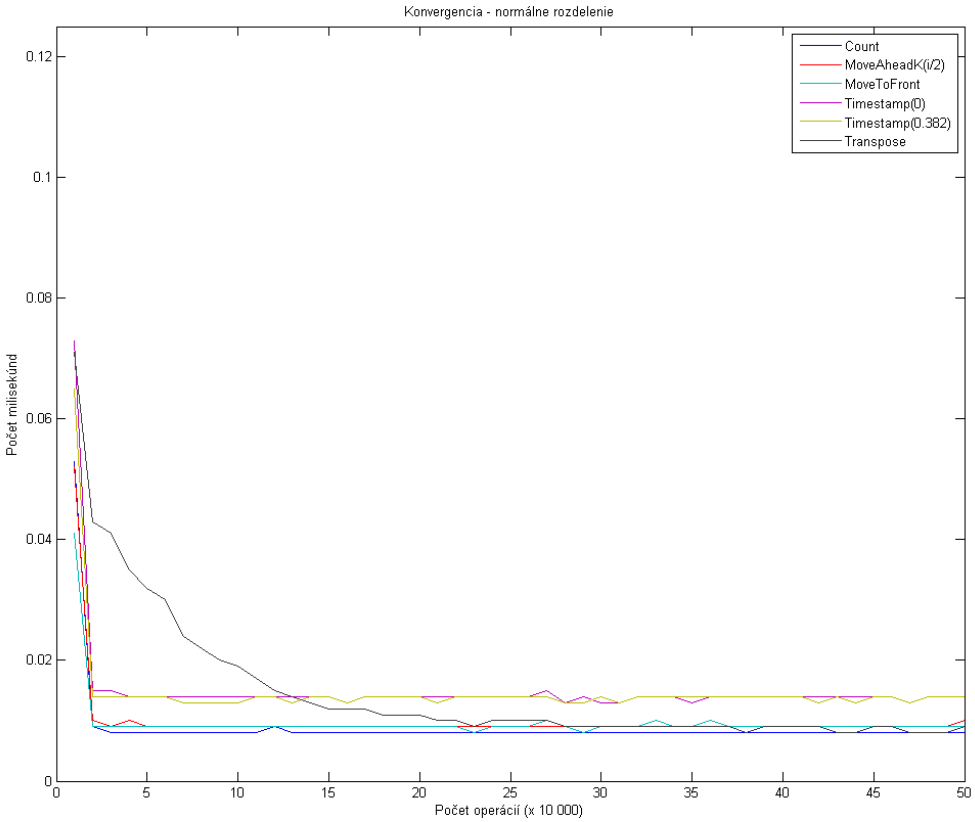


Obrázok 19 Konvergencia - normálne rozdelenie - počet porovnaní

Druhým testovaným rozdelením v teste konvergencie samoupravujúcich zoznamov bolo normálne rozdelenie. Pri tomto pravdepodobnostnom rozdelení konvergovali všetky testované samoupravujúce stratégie. Stratégie Count, Move-to-front, Move-ahead-k vo verzii s k=i/2 a Timestamp reagovali okamžite, už po pár tisíc operáciách Member.

Stratégia Transpose konvergovala ako úplne posledná po výrazne väčšom počte operácií Member ako ostatné stratégie. Na obrázku splýva priebeh stratégie Move-ahead-k vo verzii s k=i/2 s priebehom deterministickej verzie stratégie Timestamp.

Tieto výsledky nie sú nijak prekvapujúce vzhľadom k tomu, že početnosti prvkov vo vyhľadávacej postupnosti sú výrazne odlišné. Tento fakt dokázali využiť všetky samoupravujúce stratégie a postupne znížiť počet porovnaní na nájdenie jedného prvku v zozname až na zhruba desatinu dĺžky zoznamu (to je približne 5x menej ako pri rovnomernom rozdelení). To, že samoupravujúca stratégia Transpose konverguje najpomalšie je dané tým, že v zozname vykonáva relatívne najmenšie zmeny.



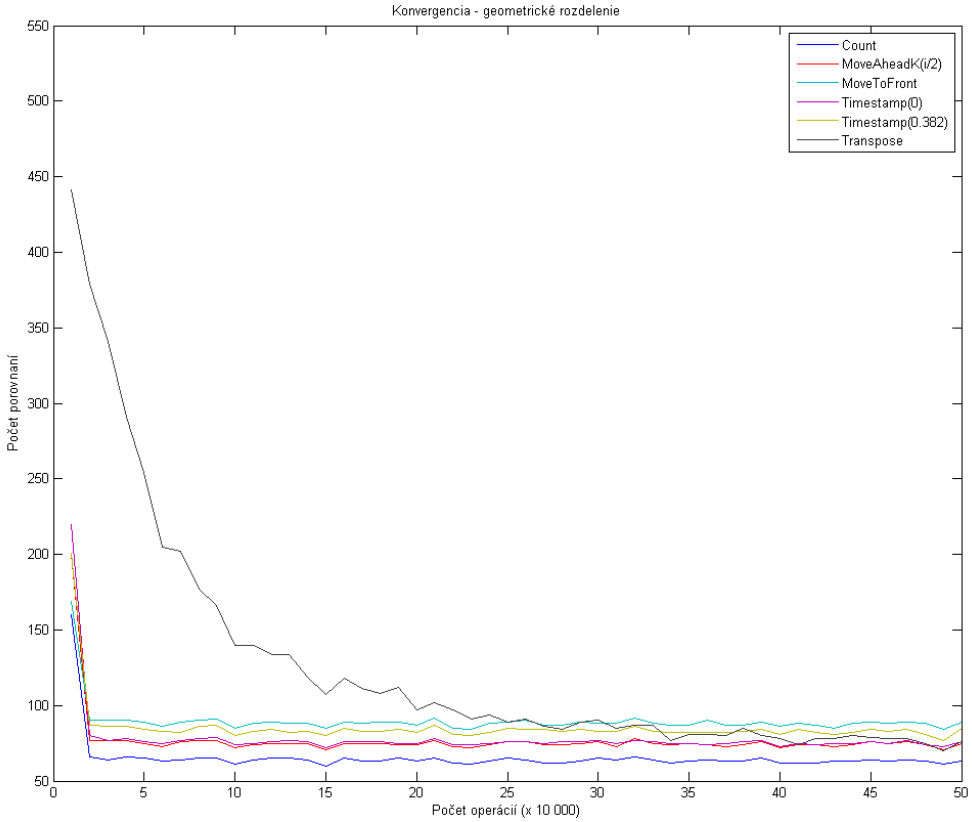
Obrázok 20 Konvergencia - normálne rozdelenie – čas

Z hľadiska času potrebného na nájdenie jedného prvku zodpovedajú namerané výsledky výsledkom z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku. Stratégie Count, Move-to-front a Timestamp reagovali okamžite, stratégia Transpose konvergovala ako úplne posledná.

Pri hodnotení času potrebného na nájdenie jedného prvku môžeme dobre vidieť vyššiu implementačnú náročnosť stratégie Timestamp v oboch svojich verziách. Na obrázku splývajú priebehy oboch verzií stratégie Timestamp a priebeh stratégie Move-To-Front s priebehom stratégie Move-ahead-k vo verzii s k=i/2.

Geometrické rozdelenie

Pri geometrickom rozdelení pol použitý parameter p=0,013.

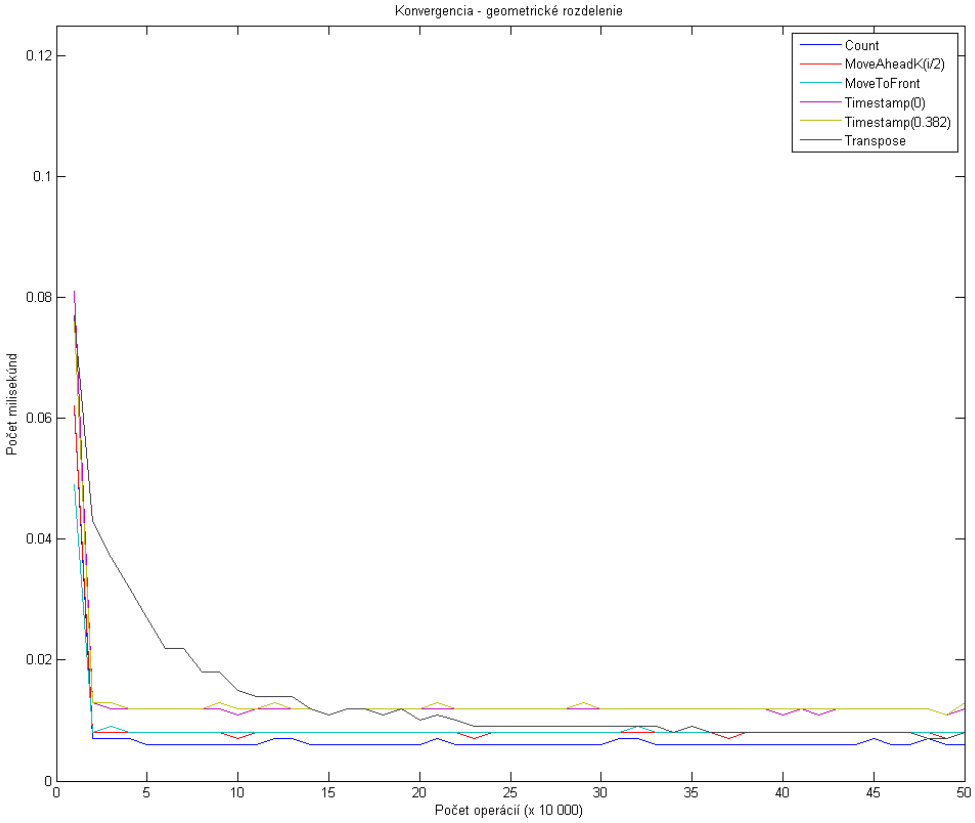


Obrázok 21 Konvergencia - geometrické rozdelenie - počet porovnaní

Výsledky testu konvergencie pri geometrickom rozdelení z hľadiska počtu porovnaní v poradí konvergencie zodpovedajú výsledkom pri predchádzajúcom normálnom rozdelení. Rozdielom je len fakt, že všetky testované samoupravujúce stratégie konvergovali na geometrickom rozdelení trochu neskôr ako na normálnom rozdelení.

Je ale potrebné si všimnúť, že zaznamenané hodnoty sú o niečo menšie ako v prípade normálneho rozdelenia a jednotlivé samoupravujúce stratégie konvergujú k menšej asymptote. Mohlo by to byť spôsobené tým, že rozdiely v početnosti v geometrickom rozdelení sú výraznejšie ako v normálnom rozdelení.

Stratégie Count, Move-to-front a Timestamp reagovali okamžite, už po pár tisíc operáciách Member. Stratégia Transpose konvergovala ako úplne posledná po výrazne väčšom počte operácií Member ako ostatné stratégie. Na obrázku splýva priebeh stratégie Move-ahead-k vo verzii s k=i/2 s priebehom deterministickej verzie stratégie Timestamp.



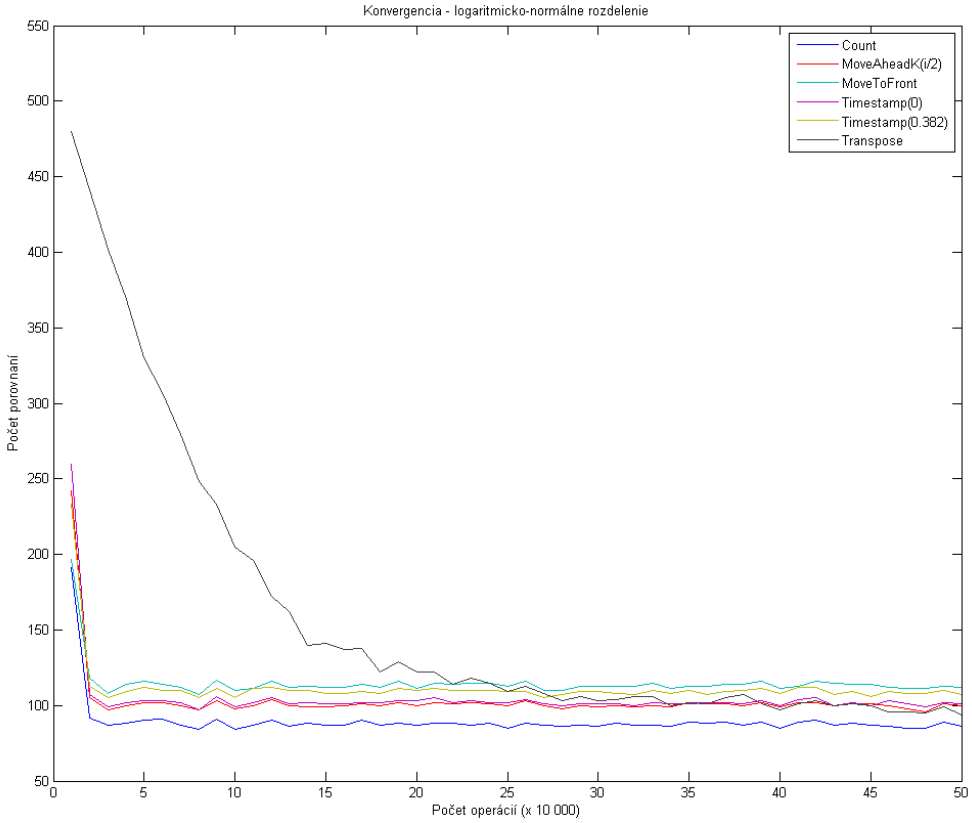
Obrázok 22 Konvergencia - geometrické rozdelenie – čas

Rovnako ako pri normálnom rozdelení tak aj pri geometrickom rozdelení zodpovedajú výsledky testu konvergencie z hľadiska času potrebného na nájdenie jedného prvku výsledkom z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku. Taktiež je vidno implementačnú zložitosť stratégie Timestamp v oboch verziách. Na obrázku splývajú priebehy oboch verzií stratégie Timestamp a priebeh stratégie Move-To-Front s priebehom stratégie Move-ahead-k vo verzii s k=i/2.

Logaritmicko-normálne rozdelenie

Pri logaritmicko-normálnom rozdelení boli použité parametre $\mu = 5$ a $\sigma = 0.39$.

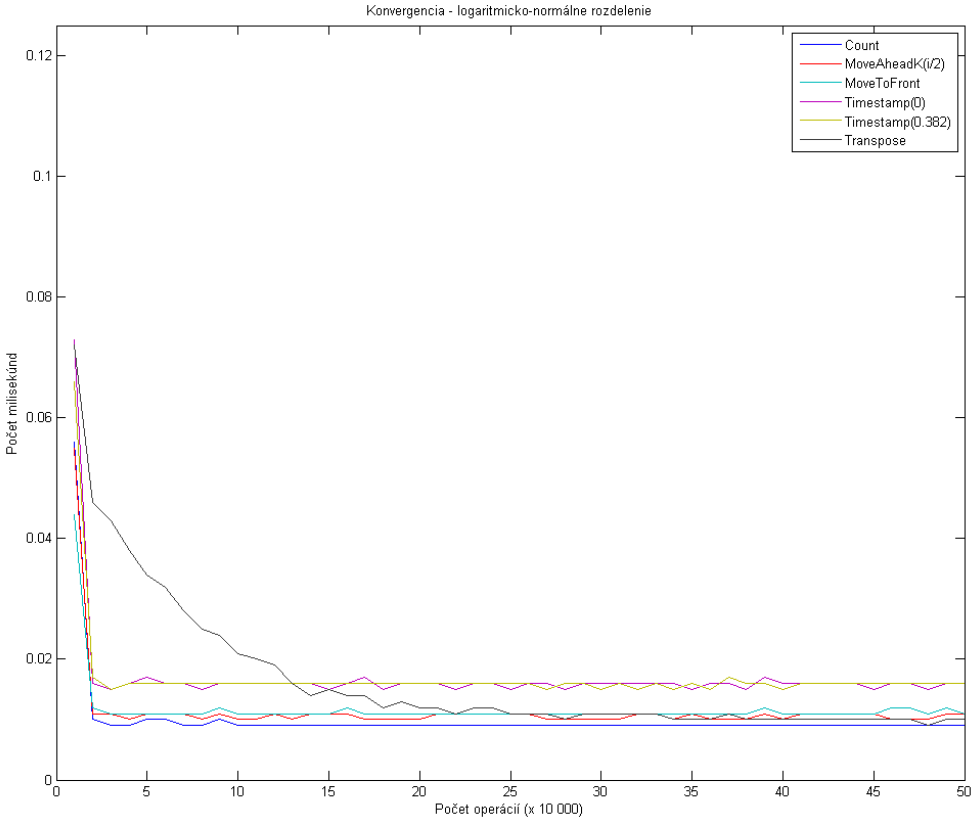
Logaritmicko-normálne rozdelenie je rozdelenie náhodnej veličiny, ktorého logaritmus sa riadi normálnym rozdelením, podobnosť výsledkov logaritmicko-normálneho a normálneho rozdelenia preto nebola nijak prekvapivá.



Obrázok 23 Konvergencia - logaritmicko-normálne rozdelenie - počet porovnaní

Stratégie Count, Move-to-front a Timestamp reagovali okamžite, už po pár tisíc operáciách Member. Stratégia Transpose konvergovala ako úplne posledná po výrazne väčšom počte operácií Member ako ostatné stratégie.

Môžeme si všimnúť, že pri tomto rozdelení sú hodnoty vyššie ako pri normálnom rozdelení a rozdiely medzi jednotlivými samoupravujúcmi stratégiami väčšie ako pri geometrickom rozdelení.



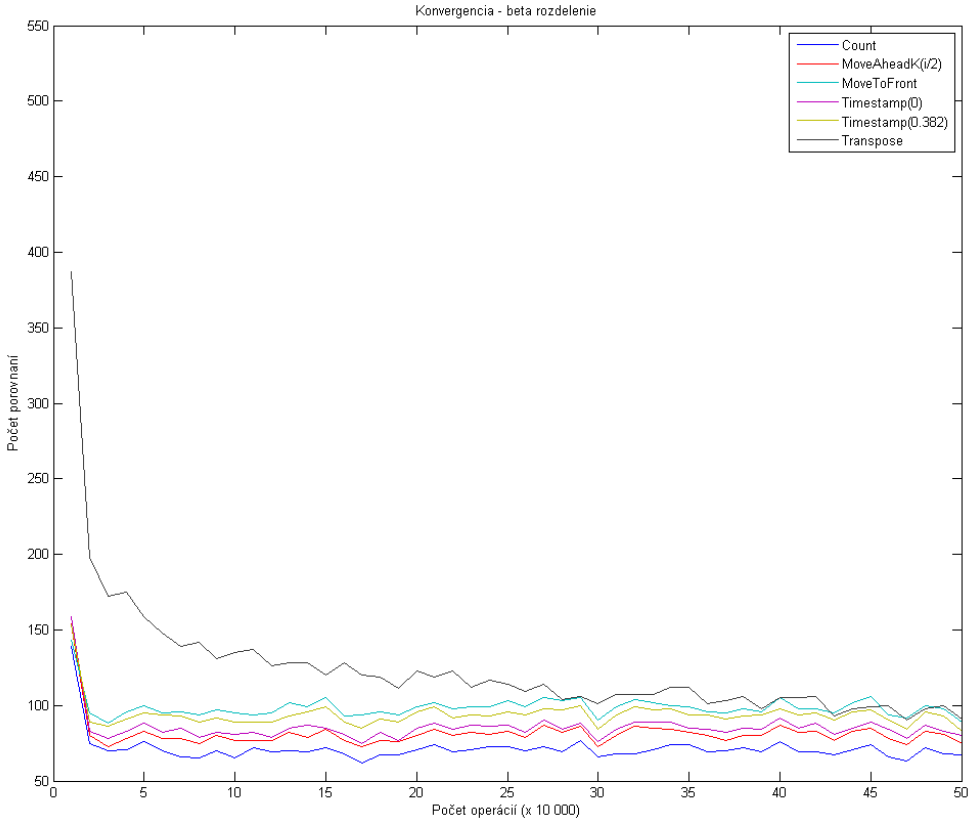
Obrázok 24 Konvergencia - logaritmicko-normálne rozdelenie – čas

Z hľadiska času potrebného na nájdenie jedného prvku zodpovedajú výsledky výsledkom z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku. Stratégie Count, Move-to-front a Timestamp reagovali okamžite, stratégia Transpose konvergovala ako úplne posledná.

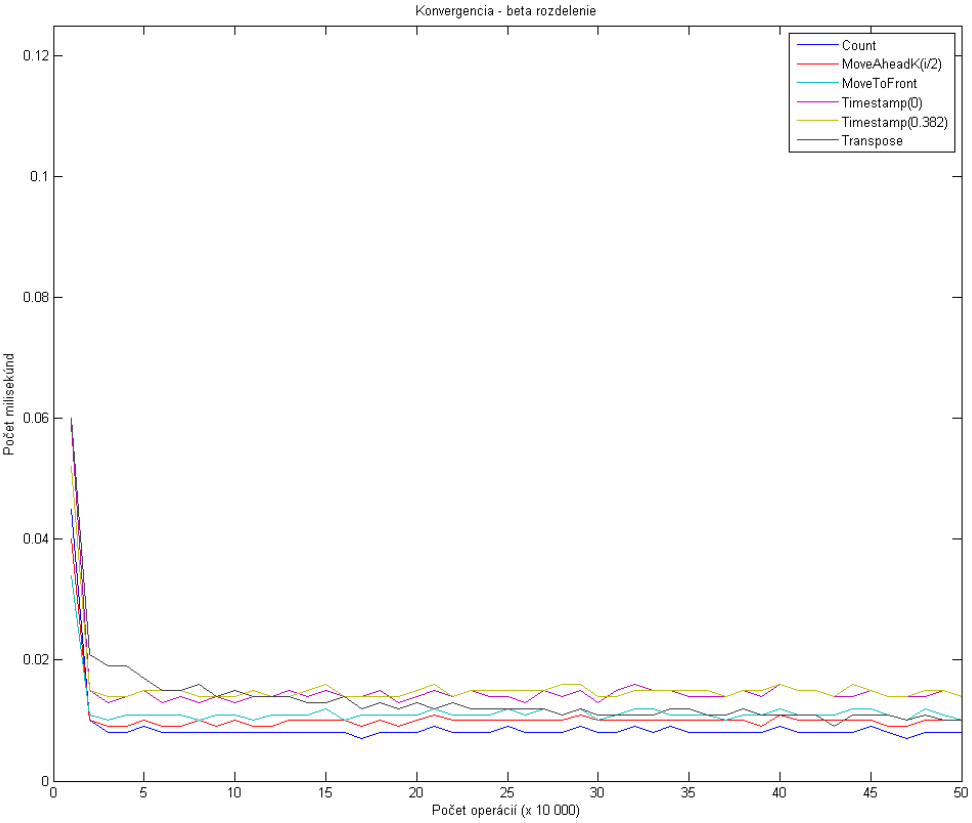
Beta rozdelenie

Pri beta rozdelení boli použité parametre A=1.2 a B=0.1 a výsledné dáta vynásobené konštantou 1000.

Rozdiely medzi jednotlivými samoupravujúcimi stratégiami Count, Move-to-front a Timestamp boli pri beta rozdelení výraznejšie, najrýchlejšie konvergovala stratégia Move-to-front. Stratégia Transpose konvergovala rovnako ako pri predchádzajúcich pravdepodobnostných rozdeleniach najpomalšie, jej konvergencia však mala viac strmý priebeh.



Obrázok 25 Konvergencia - beta rozdelenie - počet porovnaní



Obrázok 26 Konvergencia - beta rozdelenie – čas

Z hľadiska času potrebného na vyhľadanie jedného prvku sú pri beta rozdelení rozdiely medzi jednotlivými stratégiami výraznejšie ako z hľadiska počtu porovnaní potrebných na vyhľadanie jedného prvku. Poradie konvergencie však zostalo zachované.

Vyhodnotenie

Pri testoch konvergencie sa potvrdil predpoklad, že rýchlosť konvergencie záleží od vyhľadávacej postupnosti. Samoupravujúce stratégie sú vhodné v prípadoch, že sa vo vyhľadávacej postupnosti opakujú jednotlivé prvky s nerovnakou početnosťou a stratégie sa podľa toho prispôsobujú, záleží teda na použitom pravdepodobnostnom rozdelení. Pri rovnomernom rozdelení, kde sú všetky prvky vo vyhľadávacej postupnosti zastúpené rovnako jednotlivé stratégie nekonvergujú, pri ostatných pravdepodobnostných rozloženiach konvergujú.

Výsledky z hľadiska počtu porovnaní

Z hľadiska počtu porovnaní na nájdenie jedného prvku najrýchlejšie konvergovali stratégie Count a Move-To-Front. Tento výsledok je očakávaný, pretože stratégia Count sa najviac približuje optimálnemu algoritmu a stratégia Move-To-Front robí v zozname pri vyhľadávaní najväčší posun nájdeného prvku.

Výsledky z hľadiska času

Z hľadiska času potrebného na nájdenie jedného prvku bola najrýchlejšia stratégia Move-to-front. Táto stratégia dosiahla veľmi dobrý výsledok už pri hodnotení počtu porovnaní potrebných na nájdenie jedného prvku a vzhľadom na svoju úplne jednoduchú implementáciu bola jej rýchlosť očakávaná. Niektoré stratégie, ako napríklad stratégia Timestamp, kvôli náročnejšiemu preusporiadavaniu zoznamu strácajú na efektivite.

Vplyv pravdepodobnostného rozdelenia na výsledky

Z experimentov vyplýva, že na rýchlosť konvergencie má vplyv najmä použité pravdepodobnostné rozlíšenie, ktorým sa testovacie dáta riadia. Najlepšie to bolo viditeľné na rovnomernom rozdelení, na ktorom žiadna samoupravujúca stratégia nekonvergovala. Použitie samoupravujúcich stratégií sa najlepšie hodí na pravdepodobnostné rozdelenia, ktoré generujú vysoké početnosti malej skupiny prvkov, ako je napríklad rozdelenie geometrické alebo beta rozdelenie.

7.2 Vyhľadávanie

Po tom, ako sme v teste konvergencie zistili, že všetky stratégie konvergujú, zaujímalo nás, ako dané samoupravujúce stratégie obstoja v porovnaní s klasickým a zotriedeným zoznamom v teste vyhľadávania.

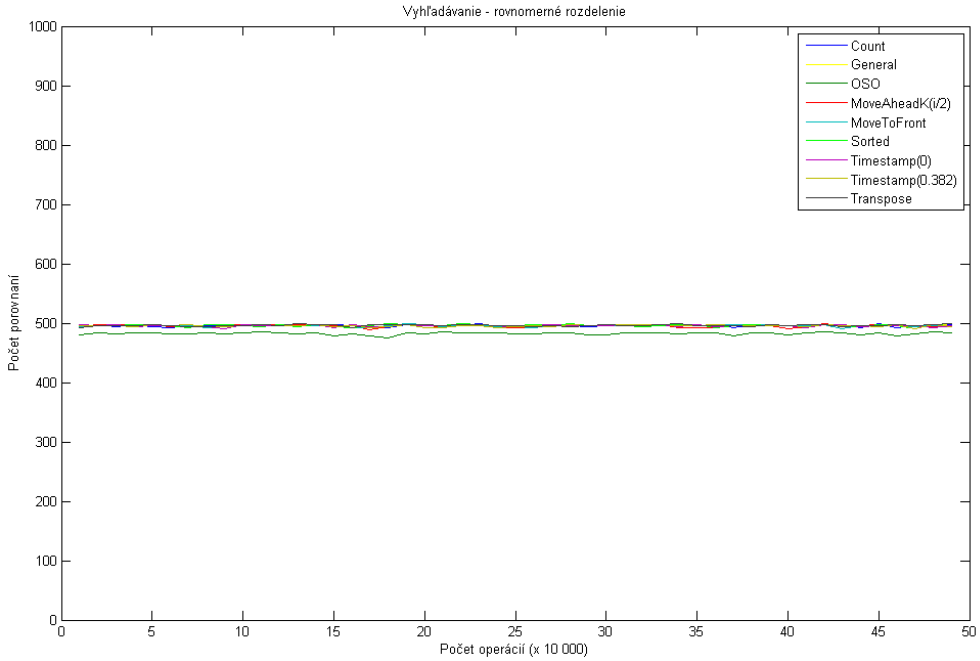
Všetky merania boli vykonané po dosiahnutí stabilného stavu, teda od bodu, v ktorom už väčšina stratégií konvergovala, čo bolo zvyčajne po 100 000 operáciách (preto sú na jednotlivých grafoch rozdielne počty operácií).

Pri každom pravdepodobnostnom rozdelení uvádzame tabuľku s percentuálnym zlepšením počtov porovnaní a časov oproti klasickému zoznamu a stratégií OSO, ktorá sa najviac približuje ideálnemu algoritmu (za 100 % sa vždy berú výsledky stratégie, voči ktorej sa porovnáva).

Ďalšia tabuľka uvádzaná pri každom pravdepodobnostnom rozdelení pre počty porovnaní a časy je relatívny pomer výsledkov jednotlivých stratégií ku klasickému zoznamu a stratégií OSO. Keďže sa stratégia OSO približuje ideálnemu algoritmu, mal by pomer výsledkov jednotlivých stratégií k výsledkom stratégie OSO v tejto tabuľke približne zodpovedať kompetitívnemu pomeru zavedenému v kapitole 3.4. Kompetitívne faktory sme sa snažili presnejšie určiť pomocou lineárnej regresie, tá však pri prekladaní priamky nameranými hodnotami uprednostnila veľmi nízku multiplikatívnu konštantu (menšiu než 1) a vysokú aditívnu konštantu (napr. y=0.52x+50). Takéto preloženie priamkou bolo síce z hľadiska regresie optimálne, ako odhad kompetitívneho faktoru sa však nehodilo.

Namerané hodnoty nájdete na CD v adresári results\member a grafy v adresári graphs\member.

Rovnomerné rozdelenie



Obrázok 27 Vyhľadávanie - rovnomerné rozdelenie - počet porovnaní

Prvým testovaným rozdelením v teste vyhľadávania bolo rovnomerné rozdelenie. Pri tomto rozdelení sme na základe jeho vlastností nepredpokladali výrazne lepšie výsledky samoupravujúcich stratégií oproti klasickému zoznamu.

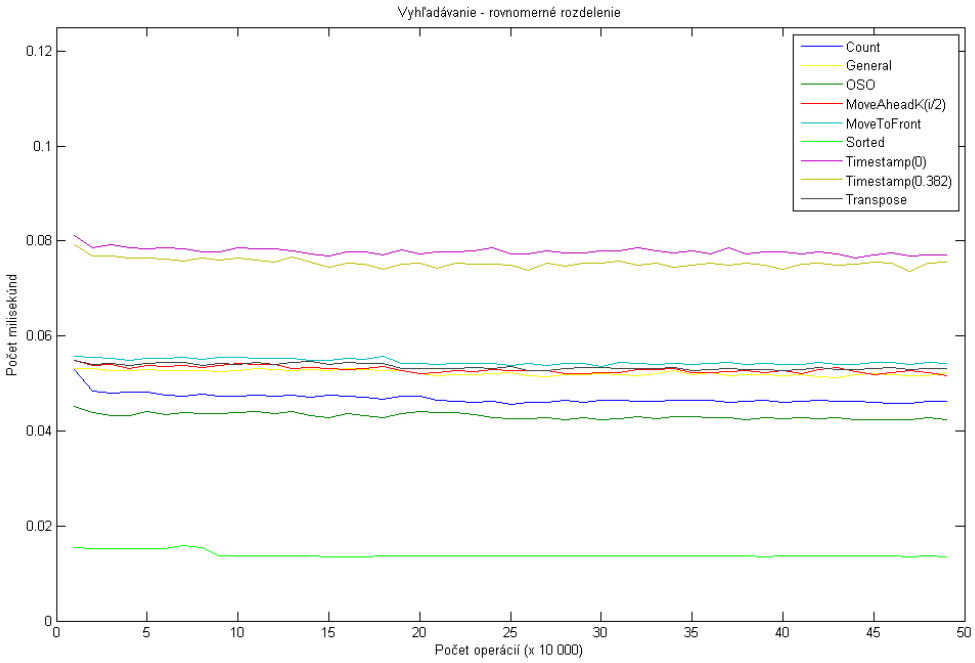
Stratégia	vs. General	vs. OSO
Count	+ 0.5 %	- 2.75 %
General	+ 0 %	- 2.80 %
OSO	+ 2.73 %	+0 %
Move-ahead-k(i/2)	+ 0.12 %	- 2.68 %
Move-To-Front	+ 0.11 %	- 2.69 %
Sorted	+ 0.37 %	- 2.77 %
Timestamp(0)	+ 0.78 %	- 2.72 %
Timestamp(0.382)	+ 0.10 %	- 2.70 %
Transpose	- 0.02 %	- 2.82 %

Tabuľka 2 Rovnomerné rozdelenie - počty porovnaní - zlepšenie stratégií oproti klasickému zoznamu a stratégii OSO

Všetky stratégie vykonávali stabilne okolo 500 porovnaní pri vyhľadávaní jedného prvku v zozname dĺžky 1000, čo sa zhoduje s očakávanou zložitosťou úspešného vyhľadávania v neusporiadaných zoznamoch pri rovnomernom rozdelení. Mierne lepší výsledok ako

ostatné testované stratégie dosiahla len stratégia OSO, ktorá predstavuje optimálny algoritmus.

Z Tabuľky 2 je jasne vidieť, že jednotlivé stratégie sú o viac ako 2% horšie ako stratégia OSO a rozdiely medzi týmito stratégiami sú naozaj veľmi malé a pohybujú sa v desatinách percenta. Mierne zlepšenie niektorých samoupravujúcich stratégií oproti klasickému zoznamu sa dajú zdôvodniť tým, že početnosti prvkov vo vyhľadávacej postupnosti nie sú úplne rovnaké.



Obrázok 28 Vyhľadávanie - rovnomerné rozdelenie – čas

Z hľadiska času potrebného na nájdenie jedného prvku je horší výsledok stratégie Timestamp v oboch svojich verziách možné odôvodniť jej implementačnou zložitosťou, pri permutácií zoznamu po nájdení prvku totiž vykonáva výrazne časovo náročnejšie operácie ako ostatné samoupravujúce stratégie.

Naopak stratégia Sorted a klasický zoznam nevykonávajú žiadnu ďalšiu prácu. Pri stratégií Sorted sa do počtu porovnaní rátajú nielen porovnania na zhodu ale aj porovnania na usporiadanie (relácia menší ako). Tieto porovnania na usporiadanie na dlhých číslach trvajú kratší čas ako porovnania na zhodu, stratégia Sorted je preto rýchlejšia ako klasický zoznam napriek skoro zhodnému výsledku vzhľadom na počty porovnaní.

Z Tabuľky 3 je vidieť výrazne lepší výsledok stratégie Sorted aj horšie výsledky stratégie Timestamp v oboch verziách.

Stratégia	vs. General	vs. OSO
Count	+ 10.26 %	- 8.71 %
General	+ 0 %	- 21.14 %
OSO	+ 5.67 %	+ 0 %
Move-ahead-k(i/2)	- 1.29 %	- 22.72 %
Move-To-Front	- 4.47 %	- 26.57 %
Sorted	+ 73.38 %	+ 67.76 %
Timestamp(0)	- 48.99 %	- 80.50 %
Timestamp(0.382)	- 44.39 %	- 74.93 %
Transpose	- 2.47 %	- 24.1513 %

Tabuľka 3 Rovnomerné rozdelenie - čas - zlepšenie stratégií oproti klasickému zoznamu a stratégií OSO

V Tabuľke 4 uvádzame pomer výsledkov testovaných samoupravujúcich stratégií k výsledkom stratégie OSO pre počty porovnaní aj čas.

Stratégia	Porovnania	Čas
Count	1.0276	1.0871
Move-ahead-k(i/2)	1.0269	1.2272
Move-To-Front	1.0269	1.2657
Timestamp(0)	1.0273	1.8050
Timestamp(0.382)	1.0270	1.7494
Transpose	1.0283	1.2415

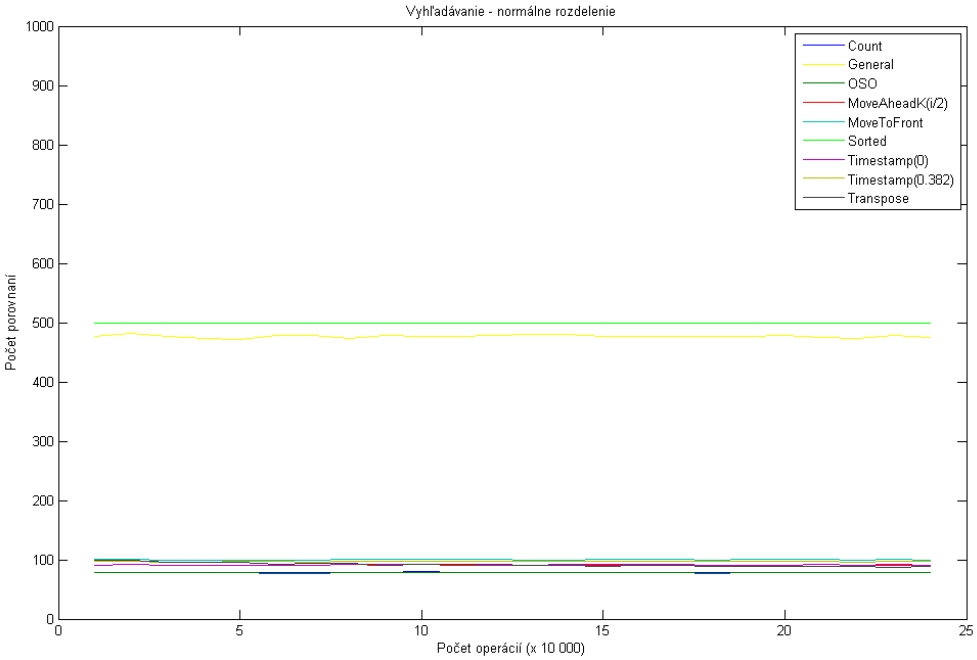
Tabuľka 4 Rovnomerné rozdelenie - Pomer samoupravujúcich stratégií k stratégií OSO

Normálne rozdelenie

Pri normálnom rozdelení boli použité parametre $\mu = 500$ a $\sigma = 50$.

Pri normálnom rozdelení sme na rozdiel od rovnomerného rozdelenia očakávali lepšie výsledky samoupravujúcich stratégií oproti klasickému zoznamu. Toto očakávanie sa potvrdilo. Na obrázku sa priebeh stratégie OSO zlieva s priebehom stratégie Count,

rovnako ako priebehy stratégií Transpose, Move-ahead-k vo verzii s k=i/2 a deterministickej verzie stratégie Timestamp.



Obrázok 29 Vyhľadávanie - normálne rozdelenie - počet porovnaní

Z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku dosiahla najlepší výsledok stratégia OSO, ktorá predstavuje optimálny algoritmus, tesne nasledovaná stratégiou Count, ktorá sa má optimálnemu algoritmu najviac približovať. Ostatné samoupravujúce stratégie dosiahli lepší výsledok ako klasický zoznam aj stratégia Sorted, ktorá dosiahla najhorší výsledok.

Klasický zoznam nemení nijak poradie prvkov v zozname, jeho priemerný výsledok by mal pri každom pravdepodobnostnom rozdelení zodpovedať približne polovici počtu prvkov zoznamu, teda 500 porovnaní na vyhľadanie jedného prvku, čo sa pri normálnom rozdelení potvrdilo.

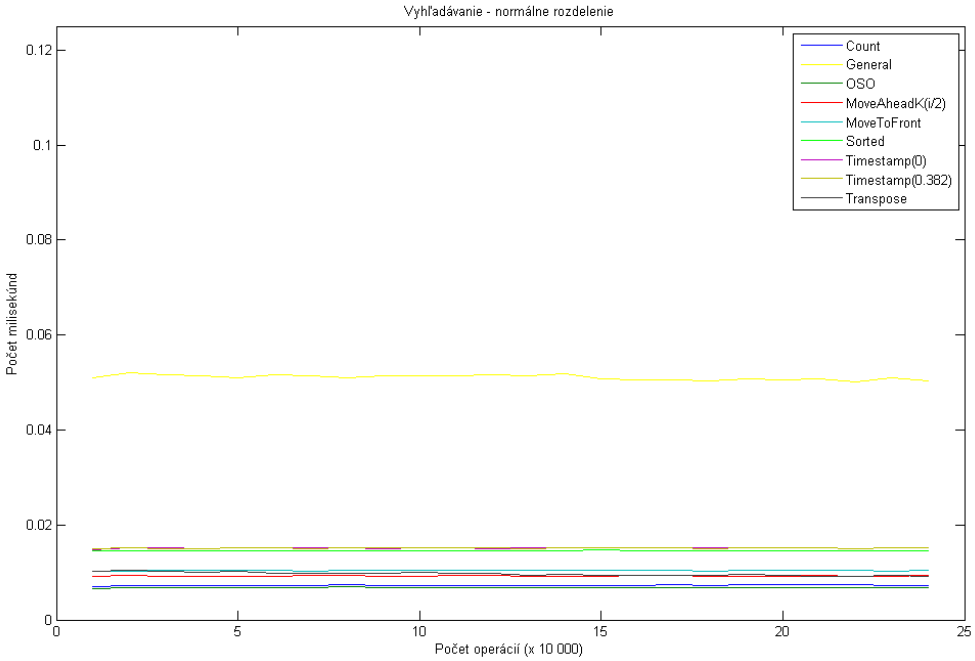
Stratégia Sorted dosiahla taktiež výsledok zodpovedajúci približne polovici počtu prvkov zoznamu, teda 500 porovnaní na vyhľadanie jedného prvku. Zdôvodnenie je možné na základe vlastností normálneho rozdelenia. Pri dátach riadiacich sa normálnym rozdelením sú najčastejšie vyhľadávané prvky zo stredu intervalu, teda v našom prípade stredne veľké prvky (Obrázok 6).

Stratégia Sorted teda najčastejšie pristupuje k prvkom okolo pozície 500. K menším a väčším prvkov pristupuje vďaka symetrickosti normálneho rozdelenia približne rovnako. Z týchto vlastností vyplýva, že stratégia Sorted potrebuje na vyhľadanie jedného prvku priemerne okolo 500 porovnaní na normálnom rozdelení.

Stratégia	vs. General	vs. OSO
Count	+ 83.43 %	- 1.39 %
General	+ 0 %	- 512.19 %
OSO	+ 83.64 %	+ 0 %
Move-ahead-k(i/2)	+ 80.92 %	- 16.80 %
Move-To-Front	+ 78.91 %	- 29.10 %
Sorted	- 4.72 %	- 541.09 %
Timestamp(0)	+ 80.81 %	- 17.44 %
Timestamp(0.382)	+ 79.63 %	- 24.66 %
Transpose	+ 80.78 %	- 17.65 %

Tabuľka 5 Normálne rozdelenie - počty porovnaní - zlepšenie stratégií oproti klasickému zoznamu a stratégií OSO

Z Tabuľky 5 je vidieť horšie výsledky klasického zoznamu a stratégie Sorted a aj veľmi podobné výsledky ostatných stratégií.



Obrázok 30 Vyhľadávanie - normálne rozdelenie – čas

Z hľadiska času potrebného na vyhľadanie jedného prvku boli najrýchlejšie stratégie OSO a Count, ktorých časy sa líšili len minimálne. Všetky samoupravujúce stratégie boli rýchlejšie ako klasický zoznam. Stratégia Sorted, ktorá dosiahla z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku najhorší výsledok bola z hľadiska času rýchlejšia, ako klasický zoznam, aj ako obe verzie samoupravujúcej stratégie Timestamp, ktorých priebeh na obrázku splýva.

Stratégia	vs. General	vs. OSO
Count	+ 85.72 %	- 7.62 %
General	+ 0 %	- 653.65 %
OSO	+ 84.68 %	+ 0 %
Move-ahead-k(i/2)	+ 81.92 %	- 36.20 %
Move-To-Front	+ 79.64 %	- 53.41 %
Sorted	+ 71.48 %	- 114.93 %
Timestamp(0)	+ 70.42 %	- 122.92 %
Timestamp(0.382)	+ 70.43 %	- 122.86 %
Transpose	+ 81.06 %	- 42.71 %

Tabuľka 6 Normálne rozdelenie - čas - zlepšenie stratégií oproti klasickému zoznamu a stratégií OSO

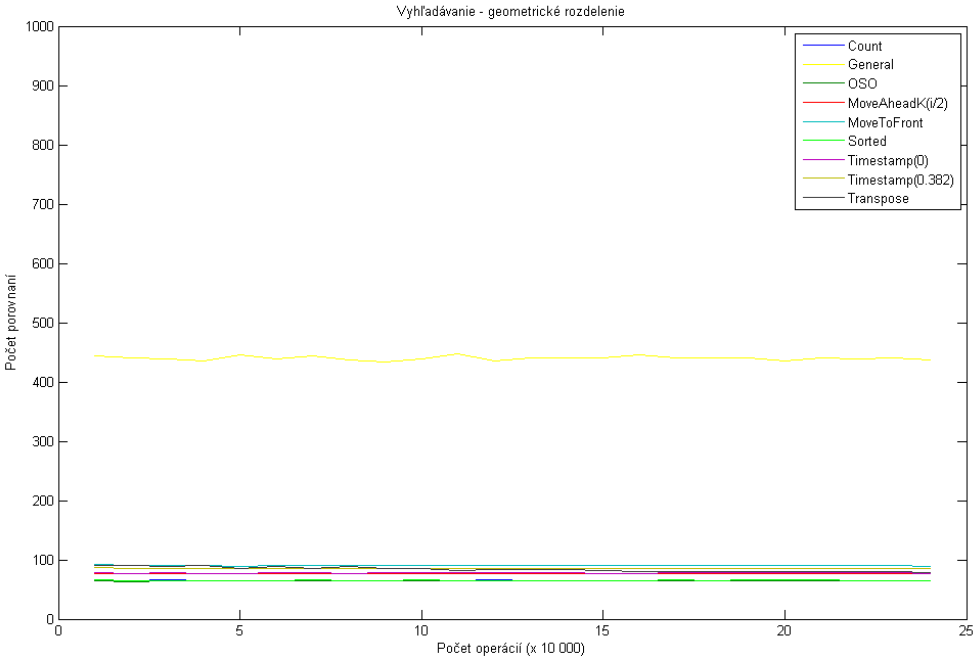
V Tabuľke 7 uvádzame pomer výsledkov testovaných samoupravujúcich stratégií k výsledkom stratégie OSO pre počty porovnaní aj čas.

Stratégia	Porovnania	Čas
Count	1.0139	1.0762
Move-ahead-k(i/2)	1.1680	1.3620
Move-To-Front	1.2911	1.5341
Timestamp(0)	1.1744	2.2293
Timestamp(0.382)	1.2467	2.2286
Transpose	1.1766	1.4272

Tabuľka 7 Normálne rozdelenie - Pomer samoupravujúcich stratégií k stratégii OSO

Geometrické rozdelenie

Pri geometrickom rozdelení pol použitý parameter p=0,013.



Obrázok 31 Vyhľadávanie - geometrické rozdelenie - počet porovnaní

Tretím testovaným pravdepodobnostným rozdelením v teste vyhľadávania bolo geometrické rozdelenie. Na obrázku sa zlievajú priebehy stratégií Sorted, Count a OSO, čiastočne aj priebehy stratégií Move-to-front, Transpose a nedeterministickej (pravdepodobnostnej) verzie stratégie Timestamp a deterministickej verzie stratégie Timestamp a stratégie Move-ahead-k s parametrom k=i/2.

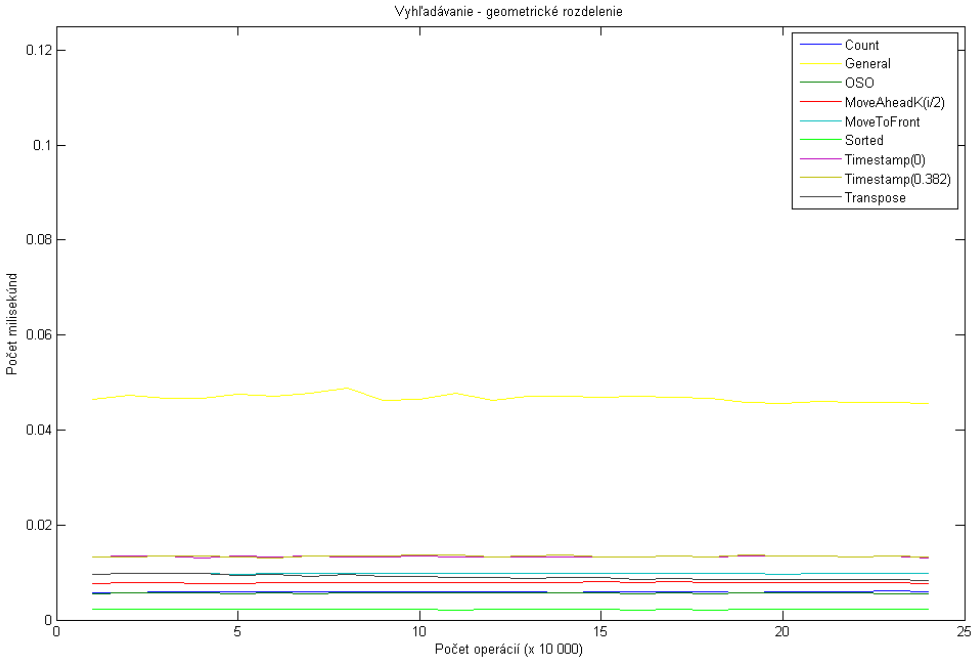
Z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku bola najrýchlejšia stratégia OSO veľmi tesne nasledovaná stratégiami Count a Sorted. Najhorší výsledok dosiahol klasický zoznam.

Dobré výsledky stratégie Sorted je možné odôvodniť pomocou vlastností geometrického rozdelenia. Pri dátach riadiacich sa týmto pravdepodobnostným rozdelením sú najviac vyhľadávané prvky zo začiatku intervalu, teda v našom prípade najmenšie prvky (Obrázok 9). Stratégia Sorted udržiava zoznam usporiadaný od najmenších prvkov na jeho začiatku po najväčšie na jeho konci.

V Tabuľke 8 vidno naozaj vyrovnané výsledky jednotlivých samoupravujúcich stratégií.

Stratégia	vs. General	vs. OSO
Count	+ 85.14 %	- 1.68 %
General	+ 0 %	- 584.57 %
OSO	+ 85.39 %	+ 0 %
Move-ahead-k(i/2)	+ 82.63 %	- 18.85 %
Move-To-Front	+ 79.48 %	- 40.44 %
Sorted	+ 85.16 %	- 1.55 %
Timestamp(0)	+ 82.32 %	- 20.99 %
Timestamp(0.382)	+ 80.56 %	- 33.05 %
Transpose	+ 80.92 %	- 30.58 %

Tabuľka 8 Geometrické rozdelenie - počty porovnaní - zlepšenie stratégií oproti klasickému zoznamu a stratégií OSO



Obrázok 32 Vyhľadávanie - geometrické rozdelenie – čas

Z hľadiska času potrebného na nájdenie jedného prvku bola najrýchlejšia stratégia Sorted, rýchlejšia ako stratégia Count (na obrázku splýva s priebehom stratégie OSO), s ktorou dosiahla takmer identické výsledky z hľadiska počtu porovnaní, aj ako ostatné samoupravujúce stratégie. Najpomalší bol klasický zoznam. Priebehy oboch verzií stratégie Timestamp v obrázku splývajú.

Stratégia	vs. General	vs. OSO
Count	+ 87.30 %	- 4.63 %
General	+ 0 %	- 724.33 %
OSO	+ 85.61 %	+ 0 %
Move-ahead-k(i/2)	+ 83.22 %	- 38.30 %
Move-To-Front	+ 79.14 %	- 71.91 %
Sorted	+ 95.31 %	+ 61.39 %
Timestamp(0)	+ 71.48 %	- 135.07 %
Timestamp(0.382)	+ 71.34 %	- 136.25 %
Transpose	+ 82.69 %	- 59.11 %

Tabuľka 9 Geometrické rozdelenie - čas - zlepšenie stratégií oproti klasickému zoznamu a stratégií OSO

V Tabuľke 9 si môžeme všimnúť najmä výrazne horší výsledok klasického zoznamu oproti ostatným stratégiám.

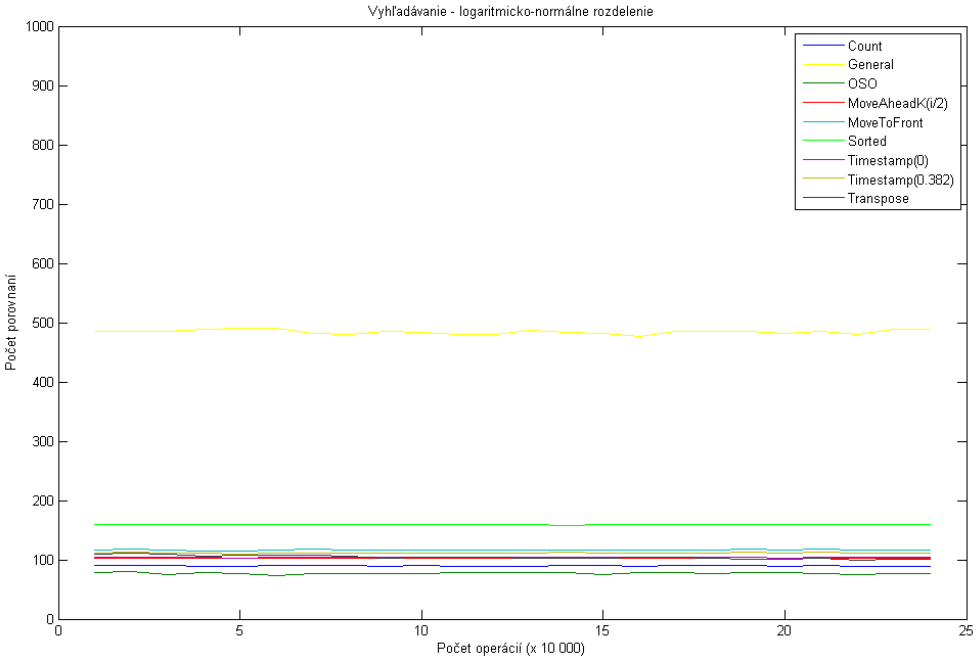
V Tabuľke 10 uvádzame pomer výsledkov testovaných samoupravujúcich stratégií k výsledkom stratégie OSO pre počty porovnaní aj čas.

Stratégia	Porovnania	Čas
Count	1.0169	1.0463
Move-ahead-k(i/2)	1.1886	1.3831
Move-To-Front	1.4044	1.7191
Timestamp(0)	1.2100	2.3507
Timestamp(0.382)	1.3305	2.3625
Transpose	1.3059	1.5912

Tabuľka 10 Geometrické rozdelenie - Pomer samoupravujúcich stratégií k stratégií OSO

Logaritmicko-normálne rozdelenie

Pri logaritmicko-normálnom rozdelení boli použité parametre $\mu = 5$ a $\sigma = 0.39$.



Obrázok 33 Vyhľadávanie - logaritmicko-normálne rozdelenie - počet porovnaní

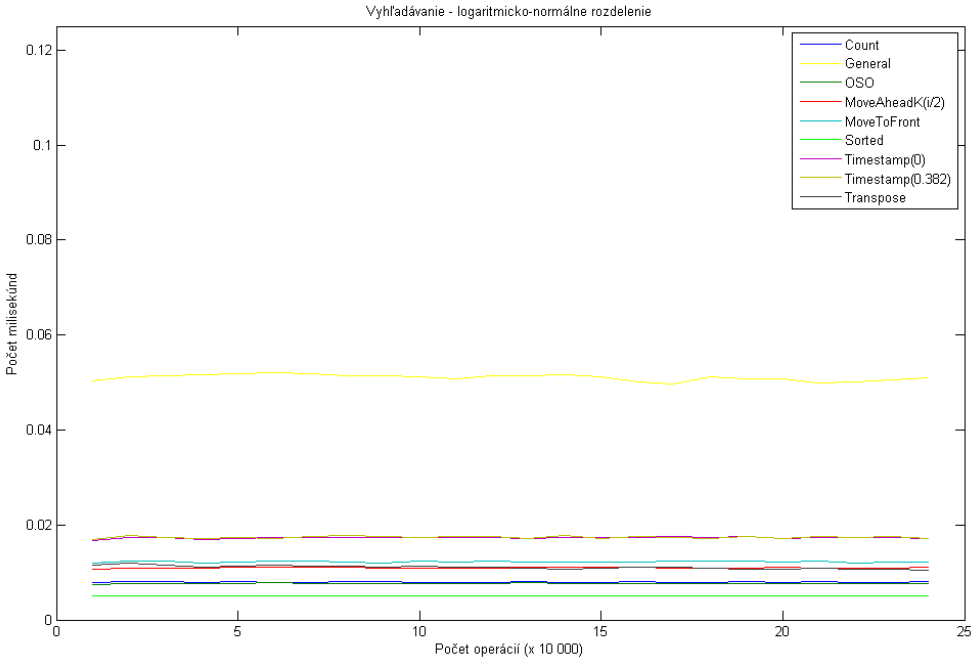
Výsledky testu vyhľadávania boli z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku pri logaritmicko-normálnom rozdelení odlišné od výsledkov pri geometrickom rozdelení. Na obrázku sa priebeh stratégie Transpose zlieva s priebehom deterministickej verzie stratégie Timestamp a so stratégiou Move-ahead-k vo verzii s k=i/2.

Stratégia OSO potrebovala rovnako ako pri predchádzajúcom testovanom pravdepodobnostnom rozdelení najmenší počet porovnaní na nájdenie jedného prvku, nasledovala stratégia Count. Stratégia Sorted, ktorá tieto dve stratégie pri geometrickom rozdelení v teste vyhľadávania nasledovala však dosiahla pri logaritmicko-normálnom rozdelení horší výsledok ako všetky samoupravujúce stratégie. Najhorší výsledok dosiahol klasický zoznam.

V tabuľke 11 dobre vidno horší výsledok stratégie Sorted oproti ostatným samoupravujúcim stratégiám a rovnaké výsledky stratégií, ktorých priebehy sa na obrázku zlievajú.

Stratégia	vs. General	vs. OSO
Count	+ 81.52 %	- 15.98 %
General	+ 0 %	- 527.91 %
OSO	+ 84.07 %	+ 0 %
Move-ahead-k(i/2)	+ 77.81 %	- 33.04 %
Move-To-Front	+ 74.93 %	- 51.07 %
Sorted	+ 67.15 %	- 106.26 %
Timestamp(0)	+ 78.49 %	- 35.04 %
Timestamp(0.382)	+ 76.91 %	- 44.92 %
Transpose	+ 78.44 %	- 35.36 %

Tabuľka 11 Logaritmicko-normálne rozdelenie - počty porovnaní - zlepšenie stratégií oproti klasickému zoznamu a stratégii OSO



Obrázok 34 Vyhľadávanie - logaritmicko-normálne rozdelenie - počet porovnaní

Výsledky stratégie Sorted môžu byť podobne ako pri geometrickom rozdelení zdôvodnené z vlastností logaritmicko-normálneho rozdelenia. Pri dátach riadiacich sa týmto rozdelením sú najviac vyhľadávané prvky zo začiatku intervalu, teda najmenšie prvky, ich početnosti sú však usporiadané inak ako geometrickom rozdelení, najviac sú vyhľadávané prvky vyskytujúce sa približne v jednej pätine intervalu (Obrázok 11). To približne zodpovedá hodnote 200 porovnaní potrebných na vyhľadanie jedného prvku. Logaritmicko-normálne rozdelenie je nesymetrické s výrazne prevládajúcimi prvkami na

začiatku intervalu, výsledok stratégie Sorted je preto posunutý pod hodnotu 200 porovnaní potrebných na vyhľadanie jedného prvku.

Z hľadiska času potrebného na nájdenie jedného prvku bola najrýchlejšia stratégia Sorted, nasledovaná päticou samoupravujúcich stratégií OSO, Count, Move-Ahead-K vo verzii s k=i/2, Transpose a Move-To-Front. Najpomalší bol klasický zoznam. Priebehy oboch verzií stratégie Timestamp na obrázku čiastočne splývajú.

Stratégia	vs. General	vs. OSO
Count	+ 84.43 %	- 3.58 %
General	+ 0 %	- 565.59 %
OSO	+ 83.88 %	+ 0 %
Move-ahead-k(i/2)	+ 78.61 %	- 42.33 %
Move-To-Front	+ 76.10 %	- 59.07 %
Sorted	+ 90.20 %	+ 34.78 %
Timestamp(0)	+ 66.13 %	- 125.43 %
Timestamp(0.382)	+ 65.93 %	- 126.73 %
Transpose	+ 78.32 %	- 44.29 %

Tabuľka 12 Logaritmicko-normálne rozdelenie - čas - zlepšenie stratégií oproti klasickému zoznamu a stratégií OSO

V Tabuľke 12 vidno hlavne väčšie rozdiely medzi jednotlivými stratégiami ako pri predchádzajúcom pravdepodobnostnom rozdelení.

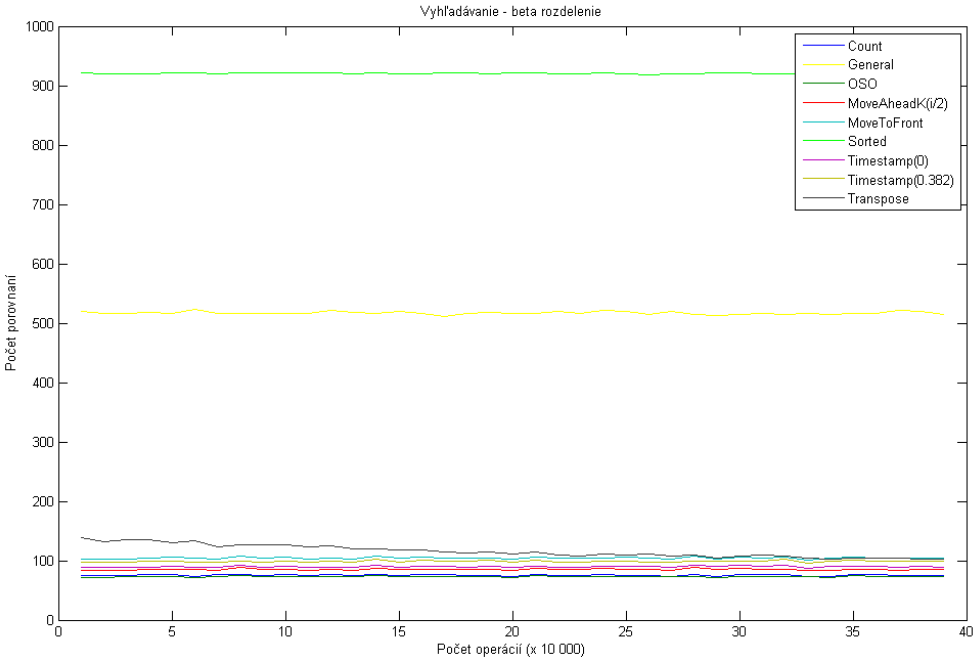
V Tabuľke 13 uvádzame pomer výsledkov testovaných samoupravujúcich stratégií k výsledkom stratégie OSO pre počty porovnaní aj čas.

Stratégia	Porovnania	Čas
Count	1.1598	1.0359
Move-ahead-k(i/2)	1.3305	1.4234
Move-To-Front	1.5108	1.5908
Timestamp(0)	1.3504	2.2543
Timestamp(0.382)	1.4492	2.2674
Transpose	1.3537	1.4429

Tabuľka 13 Logaritmicko-normálne rozdelenie - Pomer samoupravujúcich stratégií k stratégií OSO

Beta rozdelenie

Pri beta rozdelení boli použité parametre A=1.2 a B=0.1 a výsledné dáta vynásobené konštantou 1000.



Obrázok 35 Vyhľadávanie - beta rozdelenie - počet operácií

Posledným testovaným pravdepodobnostným rozdelením v teste vyhľadávania bolo beta rozdelenie. Test bol vykonaný po 100 000 operáciách, kedy už konvergovali všetky stratégie okrem stratégie Transpose. Z hľadiska počtu porovnaní potrebných na vyhľadanie jedného prvku dosiahli najlepší výsledok stratégie OSO a Count, ktorých priebehy na obrázku splývajú. Najhorší výsledok dosiahla stratégia Sorted, jej výsledok bol dokonca horší ako výsledok klasického zoznamu.

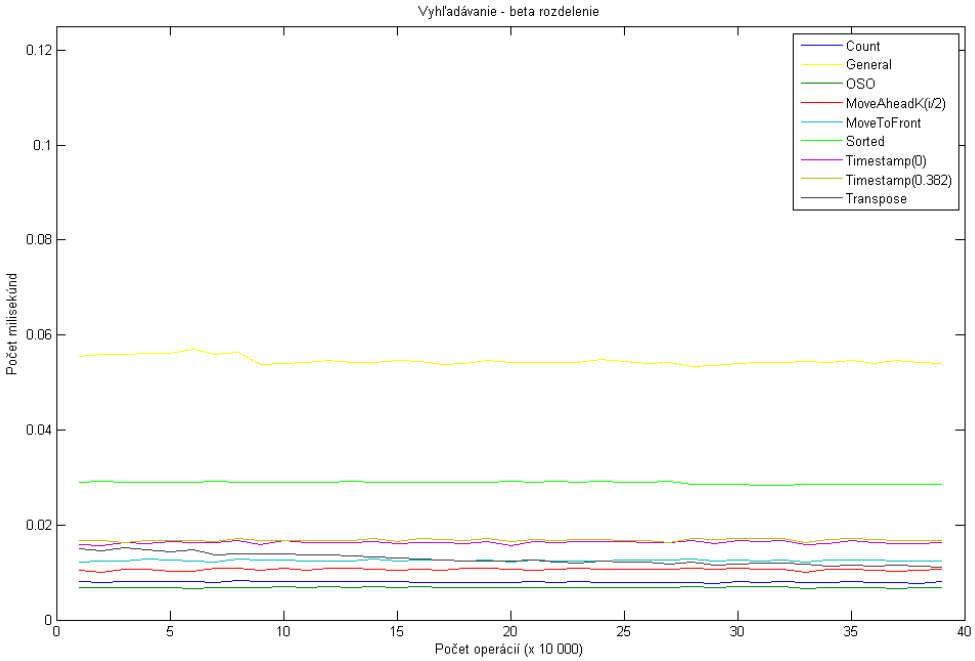
Výsledky stratégie Sorted je možné odôvodniť pomocou vlastností beta rozdelenia. Pri dátach riadiacich sa týmto pravdepodobnostným rozdelením sú najviac vyhľadávané prvky konca intervalu, teda v našom prípade najväčšie prvky (Obrázok 12). Stratégia Sorted udržiava zoznam usporiadaný od najmenších prvkov na jeho začiatku po najväčšie na jeho konci. Väčšina vyhľadávaní pri beta rozdelení smeruje na najväčšie prvky, teda pri stratégií Sorted na koniec zoznamu, čo spôsobuje výrazne horšie výsledky tejto stratégie oproti ostatným samoupravujúcim zoznamom aj klasickému zoznamu.

V Tabuľke 14 je viditeľná hlavne nevhodnosť použitia stratégie Sorted pri beta rozdelení.

Stratégia	vs. General	vs. OSO
Count	+ 85.42 %	- 2.79 %
General	+ 0 %	- 605.13 %
OSO	+ 85.81 %	+ 0 %
Move-ahead-k(i/2)	+ 83.50 %	- 16.29 %
Move-To-Front	+ 79.77 %	- 42.58 %
Sorted	- 77.96 %	- 1154 %
Timestamp(0)	+ 82.63 %	- 22.48 %
Timestamp(0.382)	+ 80.84 %	- 35.06 %
Transpose	+ 77.56 %	- 58.18 %

Tabuľka 14 Beta rozdelenie - počty porovnaní - zlepšenie stratégií oproti klasickému zoznamu a stratégií OSO

Z hľadiska času potrebného na nájdenie jedného prvku sa tiež prejavili spomínané vlastnosti beta rozdelenia a stratégia Sorted bola pomalšia ako všetky samoupravujúce stratégie, ale rýchlejšia ako klasický zoznam. Najrýchlejšie boli stratégie OSO a Count.



Obrázok 36 Vyhľadávanie - beta rozdelenie – čas

Pri porovnaní výsledok stratégie Sorted v Tabuľke 14 a Tabuľke 15 vidno, že výhoda tejto stratégia je v rýchlosti nie v počte porovnaní.

Stratégia	vs. General	vs. OSO
Count	+ 85.43 %	- 17.23 %
General	+ 0 %	- 704.72 %
OSO	+ 84.59 %	+ 0 %
Move-ahead-k(i/2)	+ 80.58 %	- 56.19 %
Move-To-Front	+ 77.04 %	- 84.73 %
Sorted	+ 47.15 %	- 325.28 %
Timestamp(0)	+ 70.22 %	- 139.64 %
Timestamp(0.382)	+ 69.21 %	- 147.69 %
Transpose	+ 76.62 %	- 88.13 %

Tabuľka 15 Beta rozdelenie - čas - zlepšenie stratégií oproti klasickému zoznamu a stratégií OSO

V Tabuľke 16 si môžeme všimnúť horšie výsledky stratégie Timestamp z hľadiska času spôsobené implementačnou zložitosťou tejto stratégie v determnistickej aj nedeterministickej (pravdepodobnostnej) verzii.

Stratégia	Porovnania	Čas
Count	1.0280	1.1723
Move-ahead-k(i/2)	1.1628	1.5620
Move-To-Front	1.4259	1.8473
Timestamp(0)	1.2248	2.3964
Timestamp(0.382)	1.3507	2.4769
Transpose	1.5818	1.8813

Tabuľka 16 Beta rozdelenie - Pomer samoupravujúcich stratégií k stratégií OSO

Vyhodnotenie

Cieľom testu vyhľadávania bolo zistiť, či sú samoupravujúce zoznamy na jednotlivých pravdepodobnostných rozdeleniach pri operácií Member efektívnejšie ako klasický zoznam a porovnať výkonnosť jednotlivých samoupravujúcich stratégií na rôznych pravdepodobnostných rozdeleniach.

Výsledky z hľadiska počtu porovnaní

Z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku sa ukázalo, že samoupravujúce stratégie sú až na stratégiu Sorted výhodnejšie ako klasický zoznam na

všetkých pravdepodobnostných rozdeleniach okrem rozdelenia rovnomerného, na ktorom sa nedokážu prejaviť ich výhody (kvôli rovnakým početnostiam vyhľadávania jednotlivých prvkov).

Najnižšie počty porovnaní potrebných na nájdenie jedného prvku dosahovala stratégia OSO, ktorá predstavuje optimálny algoritmus a stratégia Count, ktorá sa najviac približuje optimálnemu algoritmu. Zaujímavé výsledky dosiahla stratégia Sorted. Jej výsledky veľmi záležali na pravdepodobnostnom rozdelení, ktorým sa riadili dáta. Ukázalo sa, že je vhodná na pravdepodobnostné rozdelenia, ktoré generujú väčšie početnosti prvkov na začiatku intervalu (geometrické rozdelenie) a nevhodná na pravdepodobnostné rozdelenia, ktoré generujú väčšie početnosti prvkov na konci intervalu (beta rozdelenie).

Výsledky z hľadiska času

Aj z hľadiska času potrebného na nájdenie jedného prvku sa ukázalo, že použitie samoupravujúcich stratégií je na všetkých pravdepodobnostných rozdeleniach okrem rovnomerného rozdelenia výhodnejšie ako použitie klasického zoznamu.

Z klasických samoupravujúcich stratégií boli na všetkých pravdepodobnostných rozdeleniach najrýchlejšie stratégie OSO a Count, ktorá dosiahli najlepší výsledok už z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku. Výsledok stratégie Sorted dosť silno závisel od použitého pravdepodobnostného rozdelenia. Na pravdepodobnostných rozdeleniach s veľkou početnosťou vyhľadávaní prvkov na začiatku intervalu bola stratégia Sorted rýchlejšia ako stratégia Count, na pravdepodobnostných rozdeleniach s veľkou početnosťou vyhľadávaní prvkov na konci intervalu však bola pomalšia ako ostatné samoupravujúce stratégie.

Na rovnomernom rozdelení je najvýhodnejšie použiť stratégiu Sorted, ktorá je na tomto pravdepodobnostnom rozdelení veľmi rýchla.

Vplyv pravdepodobnostného rozdelenia na výsledky

Rovnako ako pri teste konvergencie malo aj pri teste vyhľadávania najväčší vplyv na výsledky jednotlivých stratégií rovnomerné rozdelenie, na ktorom sa nemohli prejaviť ich výhody. Použité pravdepodobnostné rozdelenie malo najväčší vplyv na stratégiu Sorted, u ktorej ovplyvňovalo výsledky veľmi výrazne. Pre túto stratégiu sú najvýhodnejšie pravdepodobnostné rozdelenia generujúce najväčšie početnosti prvkov na začiatku

intervalu a najnevýhodnejšie pravdepodobnostné rozdelenia generujúce najväčšie početnosti prvkov na konci rozdelenia. Na ostatné samoupravujúce stratégie malo použité pravdepodobnostné rozdelenie menší vplyv. Najlepšie výsledky dosiahli samoupravujúce stratégie na pravdepodobnostných rozdeleniach, ktoré generujú vysoké početnosti malej skupiny prvkov, ako je napríklad rozdelenie geometrické alebo beta rozdelenie.

Porovnanie s teoretickými výsledkami

Pri každom testovanom pravdepodobnostnom rozdelení sme uviedli tabuľku, ktorá obsahovala relatívny pomer výsledku každej stratégie k výsledku stratégie OSO pre počty porovnaní aj čas. Keďže sa stratégia OSO predstavuje optimálny algoritmus, mal by tento pomer približne zodpovedať kompetitívnemu faktoru. Tieto výsledky sa pre jednotlivé rozdelenia líšia, ak však zoberieme ich priemery, nezahrnieme rovnomerné rozdelenie, dostaneme nasledujúcu tabuľku usporiadanú podľa pomeru k stratégií OSO pre počty porovnaní a čas.

Stratégia	Porovnania	Čas
Count	1.0852	1.1407
Move-ahead-k(i/2)	1.1754	1.3915
Timestamp(0)	1.1974	2.2071
Timestamp(0.382)	1.2808	2.2170
Transpose	1.2893	1.5168
Move-To-Front	1.3318	1.5914

Tabuľka 17 Pomer samoupravujúcich stratégií k stratégii OSO

V Tabuľke 1 sú uvádzané kompetitívne faktory pre stratégie Timestamp(0.382), Timestamp(0) a Move-to-front s hodnotami 1.6181, 2 a 2. Tieto kompetitívne faktory sú vyššie ako nami namerané hodnoty pre počty porovnaní v Tabuľke 17 a stratégia Move-to-front sa v našom prípade viac približuje stratégií Timestamp(0.382) ako stratégií Timestamp(0).

7.3 Vyhľadávanie na dlhých zoznamoch

Posledným testom, ktorý sme vykonali, bol test vyhľadávania na dlhých zoznamoch. Rozdiel oproti prechádzajúcemu testu vyhľadávania je v tom, že testovacia množina

nebola tvorená náhodnou permutáciou čísel 1..1000, ktoré sa vyskytujú vo vyhľadávacej množine, ale náhodnou permutáciou čísel 1..10000. V tomto prípadne vykonávame rovnako ako v teste vyhľadávania operáciu úspešný Member, nevyhľadávame však všetky prvky zoznamu, ale len niektoré. Rozdiely medzi jednotlivými samoupravujúcimi stratégiami by sa v tomto prípade mali prejaviť výraznejšie, pretože presun nájdených prvkov na začiatku zoznamu by mal trvať dlhšie.

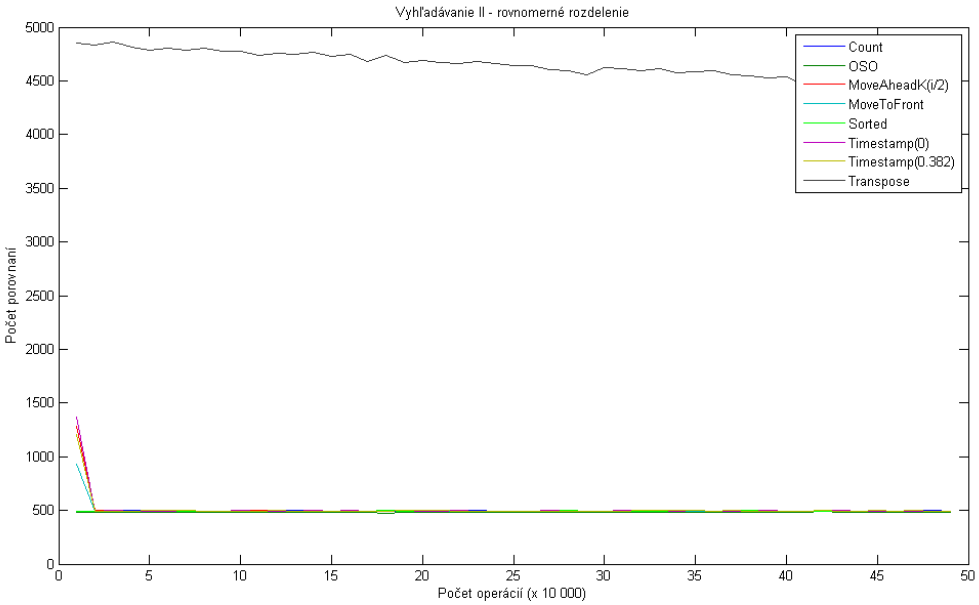
Na rozdiel od prechádzajúceho testu vyhľadávania uvádzame pre lepšie porovnanie priebeh jednotlivých stratégií od úplného začiatku, nie až od bodu konvergencie. Výsledky klasického zoznamu neuvádzame.

Tabuľka uvádzaná pri každom pravdepodobnostnom rozdelení pre počty porovnaní a časy je relatívny pomer výsledkov jednotlivých stratégií ku klasickému zoznamu a stratégií OSO. Keďže sa stratégia OSO približuje ideálnemu algoritmu, mal by pomer výsledkov jednotlivých stratégií k výsledkom stratégie OSO v tejto tabuľke približne zodpovedať kompetitívnemu faktoru zavedenému v kapitole 3.4. Pre stratégiu Transpose približné kompetitívne faktory neuvádzame, pretože v tomto teste nestihla nekonvergovať na žiadnom pravdepodobnostnom rozdelení.

Namerané hodnoty nájdete na CD v adresári results\member2 a grafy v adresári graphs\member2.

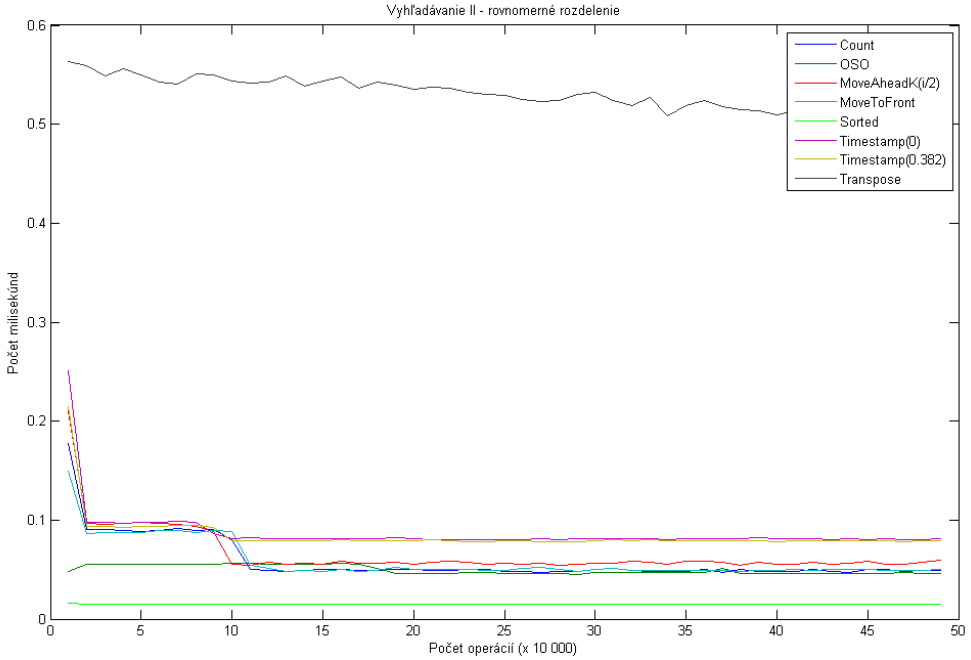
Rovnomerné rozdelenie

Prvým testovaním rozdelením bolo rovnako ako v predchádzajúcich testoch rovnomernom rozdelenie. Pri tomto pravdepodobnostnom rozdelení sú síce všetky prvky vyhľadávané s rovnakou početnosťou, v zozname sa však nachádzajú aj prvky, ktoré vyhľadávané vôbec nie sú. Predpokladali sme preto aj na rovnomennom rozdelení úvodnú konvergenciu spôsobenú tým, že si jednotlivé stratégie presunú vyhľadávaciu podmnožinu na začiatok testovacej množiny.



Obrázok 37 Vyhľadávanie na dlhých zoznamoch - rovnomerné rozdelenie - počet operácií

Najhorší výsledok z hľadiska počtu porovnaní na nájdenie jedného prvku dosiala stratégia Transpose kvôli svojmu výrazne konzervatívnemu prístupu, ktorý spôsobuje veľmi pomalé presúvanie vyhľadávaných prvkov smerom k začiatku zoznamu. Všetky ostatné testované stratégie dosiahli po úvodnej konvergencii skoro rovnaké výsledky okolo hodnoty 500, ich priebehy preto na obrázku splývajú.



Obrázok 38 Vyhľadávanie na dlhých zoznamoch - rovnomerné rozdelenie – čas

Z hľadiska času potrebného na nájdenie jedného prvku dosiahla najhorší výsledok taktiež stratégia Transpose. Najlepší výsledok dosiahla stratégia Sorted. Táto stratégia ma oproti ostatným stratégiám výhodu v tom, že si testovaciu množinu zotriedi vzostupne už pred začiatkom testu a vyhľadávanie teda prebieha vždy len v prvej desatine zoznamu (prvky 1..1000 na množine 1..10000). Horšie výsledky dosiahla stratégia Timestamp v deterministickej aj nedeterministickej (pravdepodobnostnej) verzii kvôli svojej implementačnej zložitosti. Ostatné testované stratégie dosiahli veľmi podobné výsledky a ich priebehy preto v obrázku splývajú.

V Tabuľke 18 si môžeme všimnúť výrazne horší výsledok stratégie Transpose a veľmi podobné výsledky ostatných samoupravujúcich stratégií.

Stratégia	Porovnania	Čas
Count	1.0419	1.1474
Move-ahead-k(i/2)	1.0575	1.2834
Move-To-Front	1.0414	1.1435
Timestamp(0)	1.0622	1.6918
Timestamp(0.382)	1.0548	1.6252

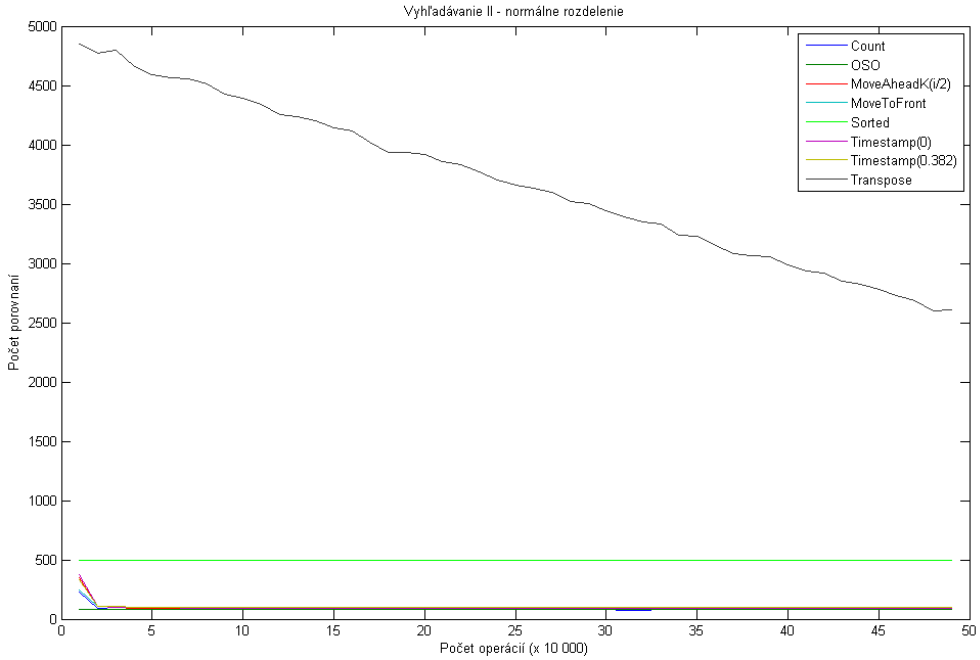
Tabuľka 18 Rovnomerné rozdelenie - Pomer samoupravujúcich stratégií k stratégií OSO na dlhom zozname

Normálne rozdelenie

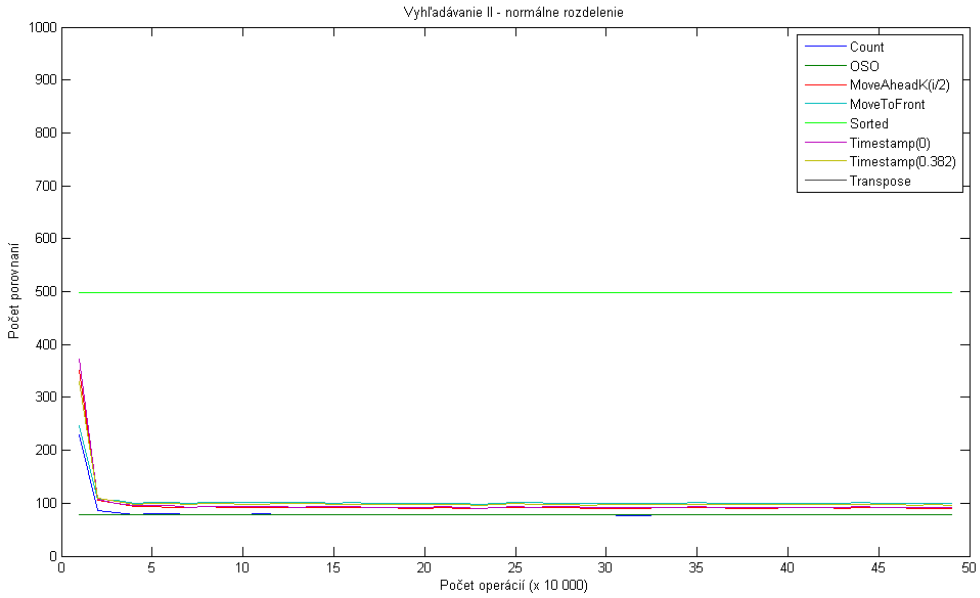
Pri normálnom rozdelení poli použité parametre $\mu = 500$ a $\sigma = 50$.

Druhým testovaným rozdelením v teste vyhľadávania na dlhých zoznamoch bolo normálne rozdelenie.

Na tomto pravdepodobnostnom rozdelení jednotlivé testované stratégie podľa očakávaní konvergovali rýchlejšie ako na rovnomernom rozdelení. Stabilný priebeh okolo 500 porovnaní mala stratégia Sorted z dôvodu predusporiadania zoznamu spomínaného pri rovnomernom rozdelení a faktu, že pri normálnom rozdelení sú vyhľadávané prevažne prvky okolo hodnoty 500. Výsledky ostatných stratégií na obrázku splývajú, uvádzame preto Obrázok 40 vo vyššom rozlíšení, na ktorom nie je viditeľná stratégia Transpose.

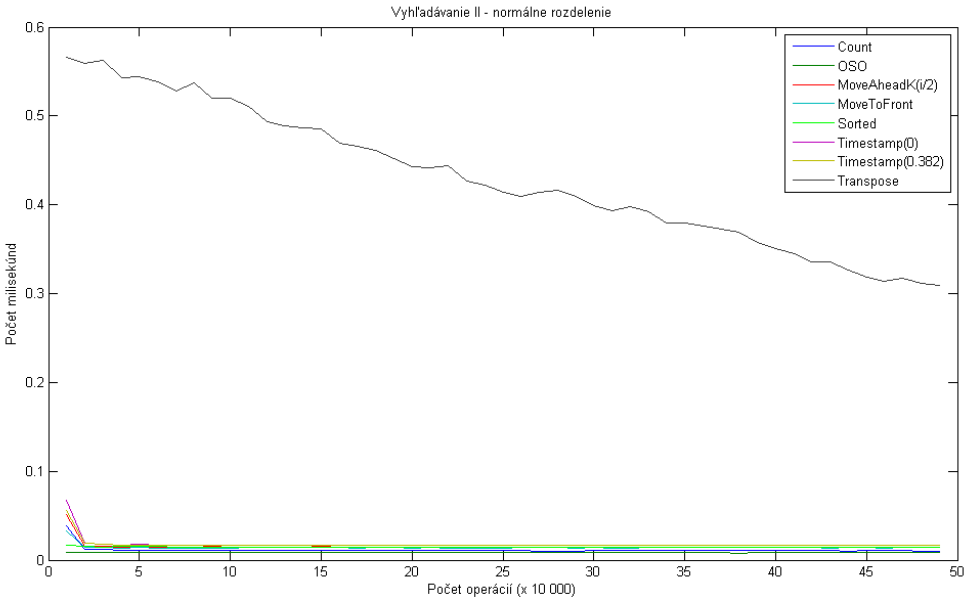


Obrázok 39 Vyhľadávanie na dlhých zoznamoch - normálne rozdelenie - počet operácií



Obrázok 40 Vyhľadávanie na dlhých zoznamoch - normálne rozdelenie - počet porovnaní pre 1..1000

Z Obrázku 40 je vidno, že najmenej porovnaní na nájdenie jedného prvku potrebovala na normálnom rozdelení stratégia OSO nasledovaná stratégiou Count. Stratégie Move-to-front a nedeterministická verzia stratégie Timestamp v oboch verziách dosiahli veľmi podobné výsledky, ktoré aj na tomto obrázku splývajú, rovnako ako priebehy stratégie Move-ahead-k vo verzii s k=i/2 a deterministickej verzie stratégie Timestamp.



Obrázok 41 Vyhľadávanie na dlhých zoznamoch - normálne rozdelenie – čas

Z hľadiska počtu času na nájdenie jedného prvku bola najrýchlejšia stratégia OSO nasledovaná stratégiami Count a Sorted. Najpomalšia bola stratégia Transpose kvôli svojej pomalej konvergencii. Priebehy ostatných stratégií v obrázku splývajú.

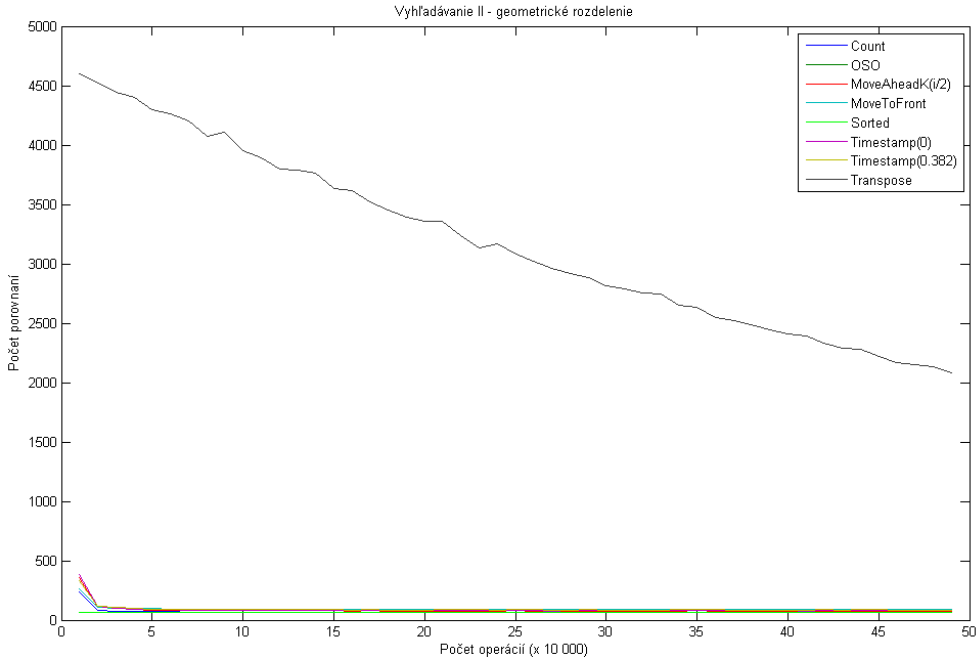
Stratégia	Porovnania	Čas
Count	1.0363	1.5818
Move-ahead-k(i/2)	1.2257	2.1294
Move-To-Front	1.3087	2.0127
Timestamp(0)	1.1648	2.3411
Timestamp(0.382)	1.2335	2.3540

Tabuľka 19 Normálne rozdelenie - Pomer samoupravujúcich stratégií k stratégii OSO na dlhom zozname

V Tabuľke 19 je viditeľný horší výsledok stratégie Transpose a veľmi podobné výsledky stratégie Timestamp v oboch verziách a stratégie Move-to-front vzhľadom na počty porovnaní.

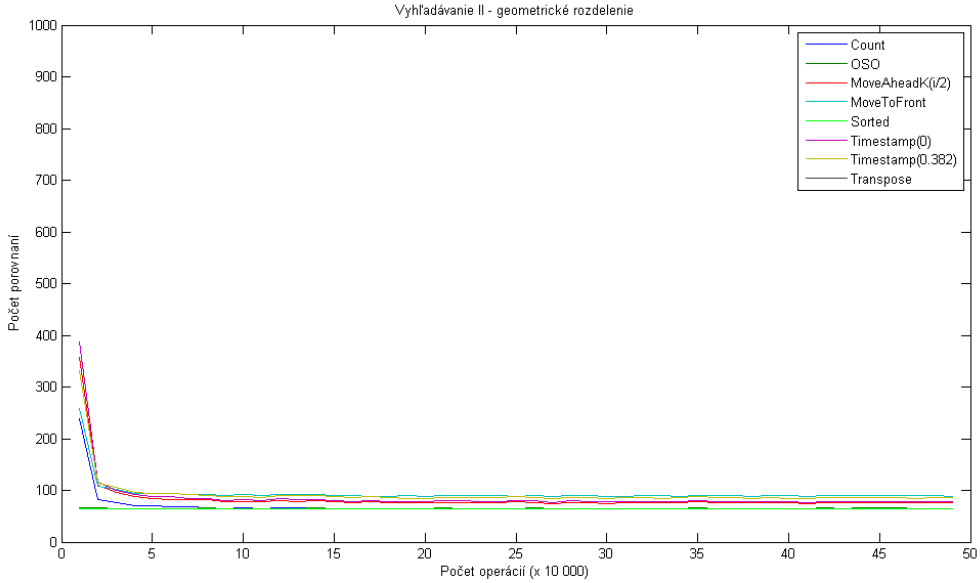
Geometrické rozdelenie

Pri geometrickom rozdelení pol použitý parameter p=0,013.



Obrázok 42 Vyhľadávanie na dlhých zoznamoch - geometrické rozdelenie - počet porovnaní

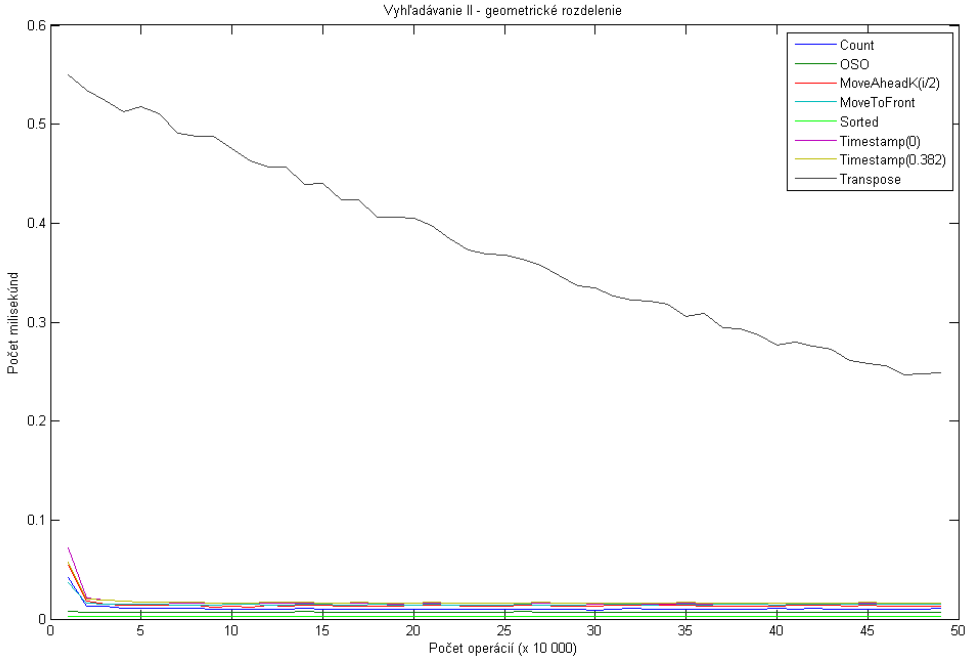
Tretím testovaným pravdepodobnostným rozdelením v teste vyhľadávania na dlhých zoznamoch bolo geometrické rozdelenie. Najhorší výsledok dosiahla stratégia Transpose, na porovnanie ostatných stratégií uvádzame rovnako ako v predchádzajúcom prípade výsek na obrázku 43.



Obrázok 43 Vyhľadávanie na dlhých zoznamoch - geometrické rozdelenie - počet porovnaní pre 1..1000

Najlepší výsledok z hľadiska počtu porovnaní na nájdenie jedného prvku dosiahli stratégie OSO, Sorted a Count, ktorých priebehy čiastočne splývajú. Rovnako ako pri normálnom

rozdelení dosiahli veľmi podobné výsledky stratégie Move-ahead-k vo verzii s k=i/2 a deterministická verzia stratégie Timestamp, ktorých priebehy taktiež splývajú. Celkovo dosiahli všetky stratégie na geometrickom rozdelení lepšie výsledky ako na normálnom.



Obrázok 44 Vyhľadávanie na dlhých zoznamoch - geometrické rozdelenie – čas

Z hľadiska času potrebného na nájdenie jedného prvku bola najrýchlejšia stratégia Sorted, pre ktorú je geometrické rozdelenie veľmi výhodné vzhľadom na to, že väčšina vyhľadávaných prvkov je zo začiatku intervalu. Nasledovala stratégia OSO, Count a ostatné stratégie, ktorých priebehy na obrázku čiastočne splývajú.

V Tabuľke 20 si môžeme všimnúť, že stratégia Timestamp dosahuje v deterministickej verzii lepšie výsledky v nedeterministickej.

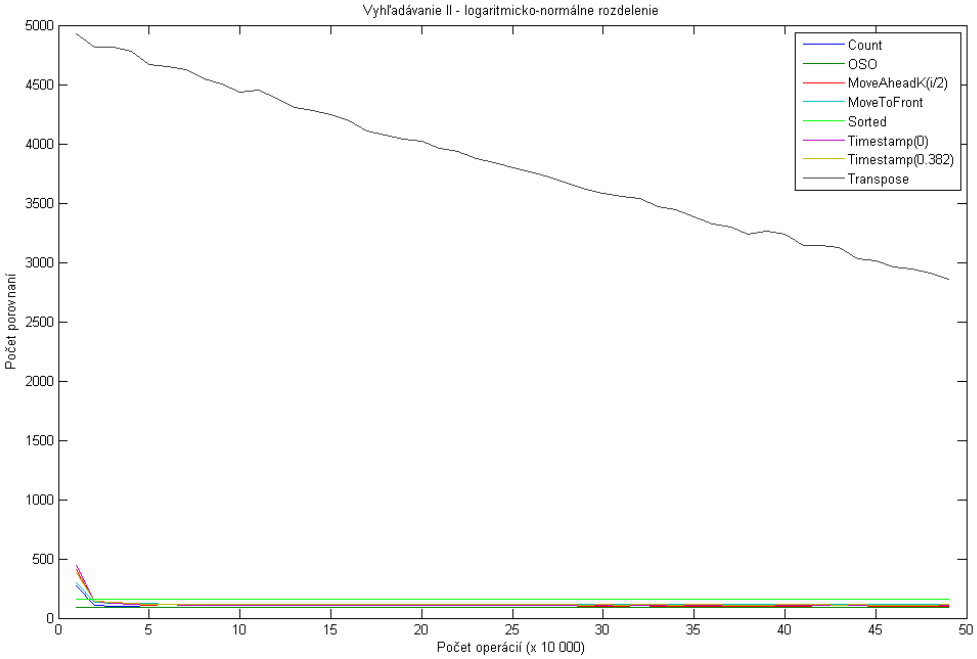
Stratégia	Porovnania	Čas
Count	1.0669	1.6791
Move-ahead-k(i/2)	1.2954	2.2181
Move-To-Front	1.4497	2.3102
Timestamp(0)	1.2058	2.4520
Timestamp(0.382)	1.3201	2.4984

Tabuľka 20 Geometrické rozdelenie - Pomer samoupravujúcich stratégií k stratégii OSO na dlhom zozname

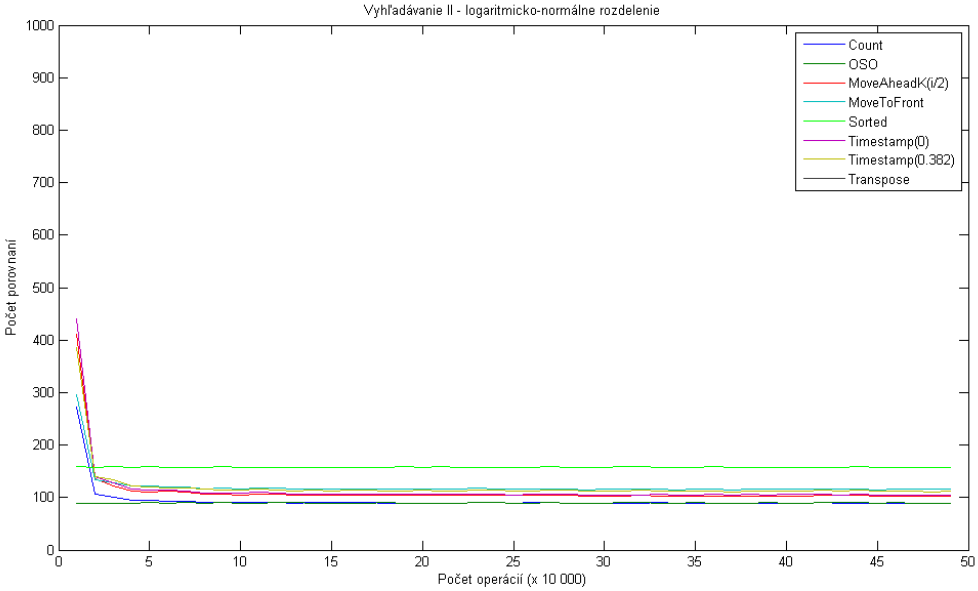
Logaritmicko-normálne rozdelenie

Pri logaritmicko-normálnom rozdelení boli použité parametre $\mu = 5$ a $\sigma = 0.39$.

Pri predposlednom testovanom logaritmicko-normálnom rozdelení dosiahla opäť z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku najhoršie výsledky stratégia Transpose, na porovnanie ostatných stratégií uvádzame rovnako ako v predchádzajúcich prípadoch výsek na obrázku 46.

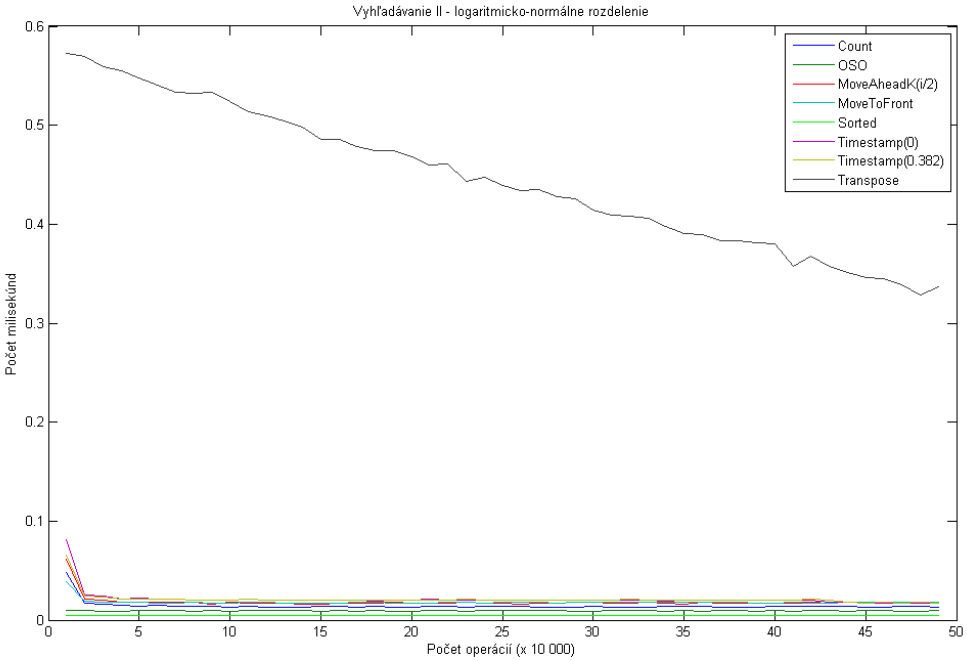


Obrázok 45 Vyhľadávanie na dlhých zoznamoch - logaritmicko-normálne rozdelenie - počet porovnaní



Obrázok 46 Vyhľadávanie na dlhých zoznamoch - logaritmicko-normálne rozdelenie - počet porovnaní pre 1..1000

Najmenej porovnaní na nájdenie jedného prvku potrebovali na logaritmicko-normálnom rozdelení stratégie OSO a Count, ktorých priebehy na obrázku splývajú. Rovnako pri dvoch predchádzajúcich testovaných pravdepodobnostných rozdeleniach dosiahli veľmi podobné výsledky stratégie Move-ahead-k vo verzii s k=i/2 a deterministická verzia stratégie Timestamp v oboch verziách, ktorých priebehy taktiež splývajú.



Obrázok 47 Vyhľadávanie na dlhých zoznamoch - logaritmicko-normálne rozdelenie – čas

Z hľadiska času potrebného na nájdenie jedného prvku bola najrýchlejšia stratégia Sorted, logaritmicko-normálne rozdelenie je pre ňu výhodné z rovnakého dôvodu ako geometrické rozdelenie, a to že najviac vyhľadávaní smeruje na prvky na začiatku intervalu. Najpomalšia bola stratégia Transpose. Priebehy ostatných testovaných stratégií a obrázku čiastočne splývajú.

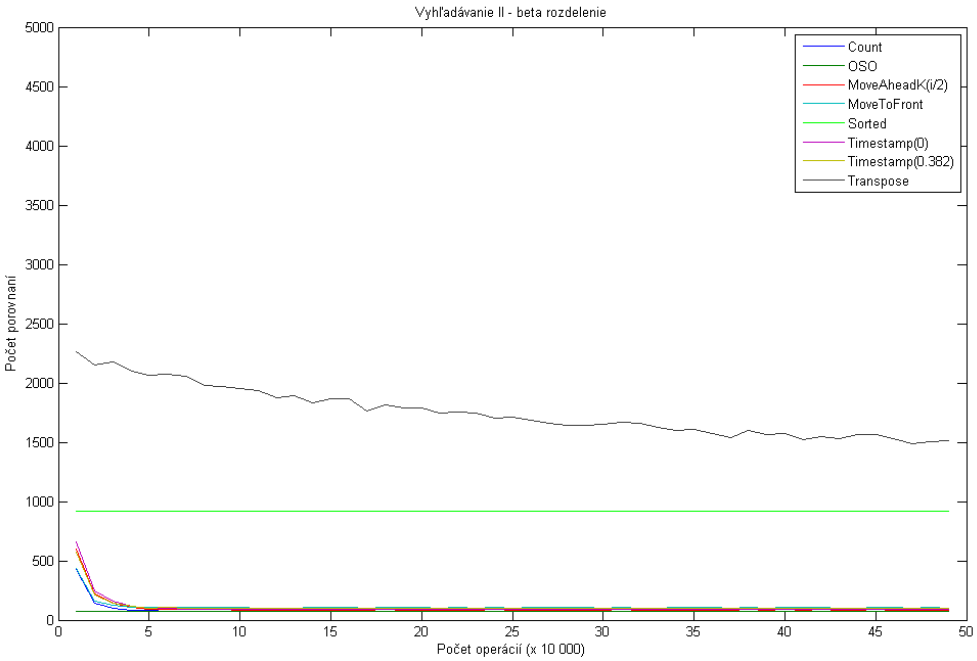
V Tabuľke 21 je viditeľný horší výsledok stratégie Transpose a veľmi podobné výsledky stratégie Timestamp v oboch verziách a stratégie Move-to-front vzhľadom na počty porovnaní.

Stratégia	Porovnania	Čas
Count	1.0527	1.5066
Move-ahead-k(i/2)	1.2520	1.9326
Move-To-Front	1.3563	1.8999
Timestamp(0)	1.1797	2.0372
Timestamp(0.382)	1.2622	2.0472

Tabuľka 21 Logaritmicko-normálne rozdelenie - Pomer samoupravujúcich stratégií k stratégií OSO na dlhom zozname

Beta rozdelenie

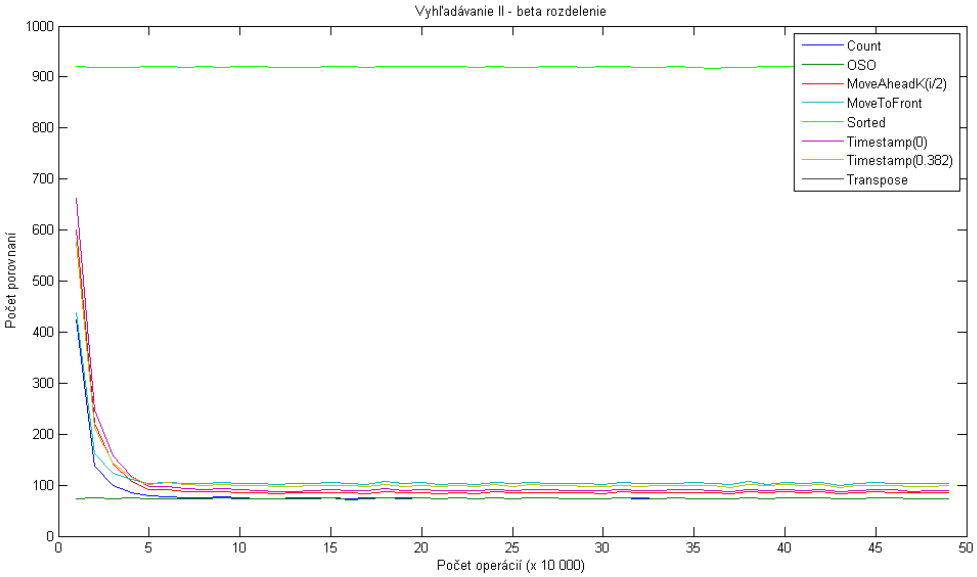
Pri beta rozdelení boli použité parametre A=1.2 a B=0.1 a výsledné dáta vynásobené konštantou 1000.



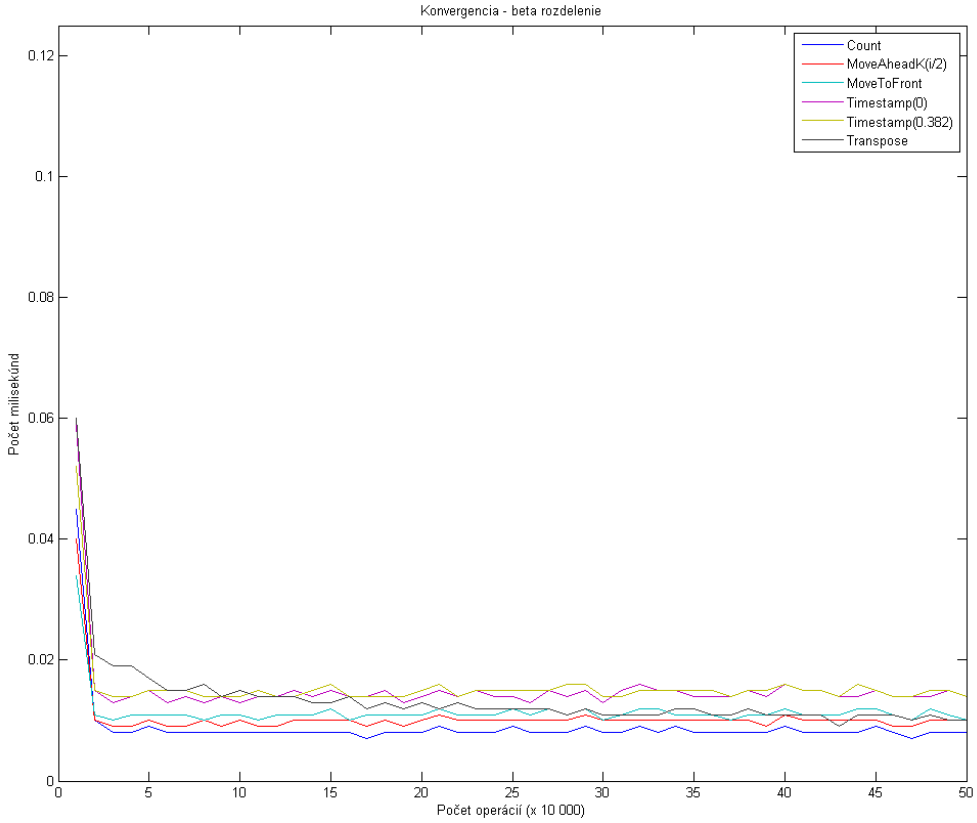
Obrázok 48 Vyhľadávanie na dlhých zoznamoch - beta rozdelenie - počet porovnaní

Posledným testovaním rozdelením v teste vyhľadávania na dlhých zoznamoch bolo beta rozdelenie. Najhorší výsledok z hľadiska počtu porovnaní na nájdenie jedného prvku dosiahla stratégia Transpose. Horší výsledok ako ostatné stratégie dosiahla ešte stratégia Sorted, pre ktorú toto pravdepodobnostné rozdelenie nie je výhodné, pretože generuje najväčšiu početnosť prvkov na konci intervalu (okolo hodnoty 1000), počty porovnaní sa pri tejto stratégií preto pohybovali okolo 1000. Na porovnanie ostatných stratégií

uvádzame rovnako ako v predchádzajúcich prípadoch výsek na obrázku 46. Najlepší výsledok dosiahla stratégia OSO nasledovaná stratégiou Count.



Obrázok 49 Vyhľadávanie na dlhých zoznamoch - beta rozdelenie - počet porovnaní pre 1..1000



Obrázok 50 Vyhľadávanie na dlhých zoznamoch - beta rozdelenie – čas

Z hľadiska času na nájdenie jedného prvku dosiahla najlepší výsledok rovnako ako z hľadiska počtu porovnaní na nájdenie jedného prvku stratégia OSO nasledovaná stratégiou Count. Oba obrázku splývajú priebehy stratégií Move-to-front a Move-ahead-k vo verzii s k=i/2 a jednotlivých verzií stratégie Timestamp.

V Tabuľke 22 si môžeme všimnúť horší výsledok stratégie Transpose a veľmi dobrý výsledok stratégie Move-ahead-k vo verzii s k=i/2.

Stratégia	Porovnania	Čas
Count	1.1284	1.4949
Move-ahead-k(i/2)	1.3544	1.9911
Move-To-Front	1.5108	1.9120
Timestamp(0)	1.2097	2.3014
Timestamp(0.382)	1.3336	2.3635

Tabuľka 22 Beta rozdelenie - Pomer samoupravujúcich stratégií k stratégií OSO na dlhom zozname

Vyhodnotenie

Cieľom testu vyhľadávania na dlhých zoznamoch bolo overiť správanie samoupravujúcich stratégií na zozname, ktorý okrem vyhľadávaných prvkov obsahuje na viac aj prvky, ktoré sa nikdy nevyhľadávajú, čo zodpovedá praxi viac ako podmienky pri predchádzajúcom teste vyhľadávania.

Potvrdil sa predpoklad, že v tomto teste sa medzi sebou jednotlivé stratégie odlíšia viac ako v predchádzajúcom teste vyhľadávania. Je to viditeľné hlavne z tabuliek kompetitívnych faktorov, ktorých hodnoty sú od seba pre jednotlivé stratégie viac vzdialené. Najvýraznejšie sme to pozorovali na stratégií Transpose, ktorá kvôli svojmu konzervatívnemu správaniu konverguje veľmi pomaly. Zaujímavé je, že po úvodnej konvergencii ostatné stratégie dosahovali len mierne horšie výsledky ako pri teste vyhľadávania na krátkych zoznamoch (s vynechaním prvkov, ktoré sa nevyhľadávajú).

Pre úplnosť v Tabuľke 23 uvádzame rovnako ako pri teste vyhľadávania priemerné kompetitívne faktory pre jednotlivé stratégie vypočítane ako priemer cez jednotlivé pravdepodobnostné rozdelenia (rovnako ako v predchádzajúcom teste bez zahrnutia

rovnomerného rozdelenia) zoradené vzostupne podľa komeptitívneho faktora vypočítaného na základe počtu porovnaní.

Oproti testu vyhľadávania je vidno zmenu poradia stratégií, vymenili sa jednotlivé verzie stratégie Timestamp. Poradie Timestamp(0.382), Move-to-front, Timestamp(0) teraz zodpovedá poradiu podľa teoretických výsledkov uvedených v Tabuľke 1.

Stratégia	**Porovnania**	**Čas**
Count	1.0711	1.5656
Timestamp(0)	1.1900	2.2829
Move-ahead-k(i/2)	1.2819	2.0678
Timestamp(0.382)	1.2874	2.3158
Move-To-Front	1.4064	2.0337

Tabuľka 23 Kompetitívne faktory samoupravujúcich stratégií na dlhom zozname

8 Záver

Cieľom tejto práce bolo experimentálne porovnať samoupravujúce stratégie vyhľadávania v spojovom zozname s klasickou stratégiou, kedy sú prvky vyhľadávané vždy lineárne od začiatku zoznamu smerom ku koncu zoznamu, porovnať výsledky experimentov s teoretickými predpokladmi, porovnať jednotlivé stratégie medzi sebou, zistiť či a za akých podmienok konvergujú, zanalyzovať závislosť ich správania na použitom pravdepodobnostnom rozdelení, vhodnosť ich použitia na testovaných pravdepodobnostných rozdeleniach a porovnať získané poznatky so známymi teoretickými výsledkami.

V tejto práci nájdete obsiahly teoretický popis jednotlivých samoupravujúcich stratégií, vrátane pseudokódu a popisu teoretických výsledkov.

Väčší dôraz ako na teoretickú časť je kladený na praktickú časť a experimenty. Vybrané samoupravujúce stratégie sme podrobili testom zameraným na overenie ich teoretických výsledkov (v teste konvergencie), na vzájomné porovnanie rýchlosti týchto stratégií a s rýchlosťou klasického zoznamu (test vyhľadávania a test vyhľadávania na dlhých zoznamoch), čo určuje ich skutočnú hodnotu v praktickom použití.

8.1 Vyhodnotenie

Prvým testom, ktorý sme vykonali, bol test konvergencie. Jeho cieľom bolo zistiť, ako rýchlo (a či vôbec) jednotlivé stratégie konvergujú do stabilného stavu. Zistili sme, že samoupravujúce stratégie sú vhodné v prípadoch, keď sa vo vyhľadávacej postupnosti opakujú jednotlivé prvky a stratégie sa podľa toho prispôsobujú. Pri rovnomernom rozdelení, kde sú všetky prvky vo vyhľadávacej postupnosti zastúpené rovnako, jednotlivé stratégie nekonvergujú, pri ostatných pravdepodobnostných rozdeleniach konvergujú.

Z experimentov vyplýva, že na rýchlosť konvergencie má vplyv najmä použité pravdepodobnostné rozdelenie, ktorým sa testovacie dáta riadia. Použitie samoupravujúcich stratégií sa najlepšie hodí na pravdepodobnostné rozdelenia, ktoré generujú vysoké početnosti malej skupiny prvkov, ako je napríklad rozdelenie geometrické alebo beta rozdelenie.

Cieľom testu vyhľadávania bolo zistiť, či sú samoupravujúce zoznamy na jednotlivých pravdepodobnostných rozdeleniach pri operácií Member efektívnejšie ako klasický zoznam a porovnať výkonnosť jednotlivých samoupravujúcich stratégií na rôznych pravdepodobnostných rozdeleniach. Stratégia OSO predstavuje optimálny algoritmus, ktorý vopred pozná jednotlivé požiadavky na vyhľadávanie, slúži teda len na teoretické porovnanie a v praxi veľmi použiteľná nie je, nebudeme sa touto stratégiou preto vo výsledkoch zaoberať. Najnižšie počty porovnaní potrebných na nájdenie jedného prvku dosahovala stratégia Count, ktorá sa najviac približuje optimálnemu algoritmu.

Z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku sa ukázalo, že samoupravujúce stratégie sú až na stratégiu Sorted výhodnejšie ako klasický zoznam na všetkých pravdepodobnostných rozdeleniach okrem rozdelenia rovnomerného.

Z klasických samoupravujúcich stratégií bola na všetkých pravdepodobnostných rozdeleniach najrýchlejšia stratégia Count, ktorá dosiahla najlepší výsledok už z hľadiska počtu porovnaní potrebných na nájdenie jedného prvku. Výsledok stratégie Sorted dosť silno závisel od použitého pravdepodobnostného rozdelenia. Na pravdepodobnostných rozdeleniach s veľkou početnosťou vyhľadávaní prvkov na začiatku intervalu bola stratégia Sorted rýchlejšia ako stratégia Count, na pravdepodobnostných rozdeleniach s veľkou početnosťou vyhľadávaní prvkov na konci intervalu však bola pomalšia ako ostatné samoupravujúce stratégie aj ako klasický zoznam.

Najlepšie výsledky dosiahli samoupravujúce stratégie na pravdepodobnostných rozdeleniach, ktoré generujú vysoké početnosti malej skupiny prvkov, ako je napríklad rozdelenie geometrické alebo beta rozdelenie.

Cieľom testu vyhľadávania na dlhých zoznamoch bolo overiť správanie samoupravujúcich stratégií na zozname, ktorý okrem vyhľadávaných prvkov obsahuje na viac aj prvky, ktoré sa nikdy nevyhľadávajú, čo zodpovedá praxi viac ako podmienky pri predchádzajúcom teste vyhľadávania. Rozdiely medzi jednotlivými samoupravujúcimi stratégiami sa v tomto prípade prejavili výraznejšie, pretože presun nájdených prvkov na začiatku zoznamu trval dlhšie. Výsledky jednotlivých testovaných stratégií boli len mierne horšie ako pri teste vyhľadávania na krátkych zoznamoch obsahujúcich iba vyhľadávané prvky, použitie

samoupravujúcich stratégií je teda vhodné aj na zoznamy, na ktorých sa vyhľadáva len časť ich prvkov.

Celkovo môžeme zhodnotiť, že použitie samoupravujúcich stratégií má zmysel, jednotlivé stratégie konvergujú a dokážu oproti použitiu klasického zoznamu ušetriť počty porovnaní aj čas potrebný na nájdenie jedného prvku. Odporúčame hlavne stratégiu Count.

8.2 Námety na budúcu prácu

V tejto práci sme pokryli problematiku samoupravujúcich zoznamov na základných pravdepodobnostných rozdeleniach. Okrem týchto základných pravdepodobnostných rozdelení by bolo zaujímavé otestovať správanie samoupravujúcich stratégií na ďalších pravdepodobnostných rozdeleniach, prípadne meniť parametre testovaných pravdepodobnostných rozdelení.

V práci sme použili relatívne krátke zoznamy, automaticky sa teda núka možnosť použitia dlhších zoznamov.

Zo samoupravujúcich stratégií by bolo zaujímavé otestovať niektoré ďalšie stratégie, najmä ich kombinácie v podobe tzv. meta-stratégií. Namiesto klasického prístupu, kedy je pri operácií Insert vložený prvok na koniec zoznamu, by bolo možné po vložení aplikovať dané permutačné pravidlo a tieto dva prístupy porovnať.

Bolo by určite zaujímavé preskúmať jednotlivé stratégie na reálnych dátach, napríklad na reťazcoch a hlavne sledovať použitie samoupravujúcich stratégií na nejakom reálnom projekte v praxi.

9 Zoznam použitej literatúry

[1] Susanne Albers, "A competitive analysis of the list update problem with lookahead," *Theoretical Computer Science*, vol. 197, no. 1-2, pp. 95-109, 1998.

[2] Susanne Albers, "Improved randomized on-line algorithms for the list update problem," *SIAM Journal on Computing*, vol. 27, no. 3, pp. 670–681, 1998.

[3] S. Albers and S. Laurer, "On list update with locality of reference," *LNCS*, vol. 5125, pp. 96-107, 2008.

[4] S. Albers and M. Mitzenmache, "Average case analyses of list update algorithms, with applications to data compression," *Algorithmica*, vol. 21, no. 3, pp. 312-329, 1998.

[5] S. Albers and M. Mitzenmacher, "Average case analyses of list update algorithms, with applications to data compression," *Algorithmica*, vol. 21, no. 3, pp. 312-329, 1998.

[6] S. Albers, B. von Stengel, and R. Werchner, "A combined BIT and TIMESTAMP algorithm for the list update problem," *Information Processing Letters*, vol. 56, no. 3, pp. 135-139, 1995.

[7] S. Albers and J. Westbrook, "Self-Organizing Data Structures," *Online Algorithms: The State of the Art*, pp. 31-51, 1998.

[8] A. Amer and B. J. Oommen, "List on Lists: A Framework for Self-organizing Lists in Environments with Locality of Reference," *LNCS*, vol. 4007, pp. 109-120, 2006.

[9] R. Bachrach and R. El-Yaniv, "Online list accessing algorithms and their applications: Recent empirical evidence," in *SODA '97 Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, 1997, pp. 53 - 92.

[10] R. Bachrach, R. El-Yaniv, and M. Reinstadtler, "On the competitive theory and practice of online list accessing algorithms," *Algorithmica*, vol. 32, no. 2, pp. 201–

245, 2002.

[11] J. L. Bentley and C. C. McGeoch, "Amortized analyses of self-organizing sequential search heurstics," *Communications of the ACM*, vol. 28, no. 4, pp. 404-411, 1985.

[12] J. R. Bitner, "Heuristics that dynamically organize data structures," *SIAM J. Comput.*, vol. 8, no. 1, pp. 82-110, 1979.

[13] J. H. Hester and D. S. Hirschberg, "Self-organizing linear search," *Computing Surveys*, vol. 17, no. 3, pp. 295-311, 1985.

[14] J. H. Hester and D. S. Hirschberg, "Self-Organizing Search Lists Using Probabilistic Back-Pointers," *Communications of the ACM*, vol. 30, no. 12, pp. 1074 - 1079, 1987.

[15] L. Hui and C. Martel, "Analyzing self-adjusting linear list algorithms with deletions and unsuccessful searches," *Information Processing Letters*, vol. 58, no. 5, pp. 231-236, 1996.

[16] L. Hui and C. Martel, "Unsuccessful search in self-adjusting data structures," *Journal of Algorithms*, vol. 15, no. 3, pp. 447-481, 1993.

[17] Philippe Chassaing, "Optimality of move-to-front for self-organizing data structures," *Annals of Applied Probability*, vol. 3, no. 4, pp. 1219–1240, 1993.

[18] Lucas Chi Kwong Hui and Charles U. Martel, "Randomized Competitive Algorithms for Successful and Unsuccessful Search on Self-adjusting Linear Lists," *Lecture Notes in Computer Science*, vol. 762, no. 199, pp. 426-435, 1993.

[19] Sandy Irani, "Two results on the list update problem," *Information Processing Letters*, vol. 38, no. 6, pp. 301–306, 1991.

[20] K. Kam, M. Y. Leung, and M. K. Siu, "Self-organizing files with dependent accesses," *Journal of Applied Probability*, vol. 21, pp. 343–359, 1984.

[21] Y. C. Kan and S. M. Ross, "Optimal list under partial memory constraints," *J.*

Appl.Probl., vol. 17, no. 4, pp. 1004-1015, 1980.

[22] K. Lam, M. K. Siu, and C. T. Yu, "A generalized counter scheme," *Theor. Comput. Sci.*, vol. 16, no. 3, pp. 271-278, 1981.

[23] K. Lam, M. K. Siu, and C. T. Yu, "A generalized counter scheme," *Theor. Comput. Sci.*, vol. 16, no. 3, pp. 271-278, 1981.

[24] M.S Manasse, L.A McGeoch, and D.D Sleator, "Competitive algorithms for on-line problems," in *Proc. 20th Annual ACM Symposium on Theory of Computing*, 1988, pp. 322-333.

[25] John McCabe, "On serial files with relocatable records," *Oper. Res.*, vol. 13, no. 4, pp. 609-618, 1965.

[26] Microsoft. (2010) Stopwatch Class (System.Diagnostics). [Online]. http://msdn.microsoft.com/en-us/library/system.diagnostics.stopwatch.aspx

[27] B.J. Oommen and D.C.Y Ma, "Deterministic learning automata solutions to the equipartitioning problem," *IEEE Transactions on Computers*, vol. 37, no. 1, pp. 2-14, 1988.

[28] B.J. Oommen and D.C.Y. Ma, "Stochastic automata solutions to the object partioning problem," *Computer Journal*, vol. 35, pp. A105–A120, 1992.

[29] N. Reingold, J. Westbrook, and D. D. Sleator, "Randomized competitive algorithmsfor the list update problem," *Algorithmica*, vol. 11, no. 1, pp. 15–32, 1994.

[30] R. Rivest, "On self-organizing sequential search heuristics," *Communications of the ACM*, vol. 19, no. 2, pp. 63 - 67, 1976.

[31] F. Schulz, "Two new families of list update algorithms," *Lecture Notes in Computer Science*, vol. 1533, pp. 100-109, 1998.

[32] Daniel D. Sleator and Robert E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, no. 2, pp. 202-208, 1985.

[33] Aaron M. Tenenbaum and Richard M. Nemes, "Two spectra of self-organizing sequential searchalgorithms," *SIAM Journal on Computing*, vol. 11, no. 3, pp. 557–566, 1982.

[34] The MathWorks, Inc. (2002) The MathWorks. [Online]. http://www.mathworks.com/access/helpdesk/help/toolbox/simevents/ref/eventbasedrandomnumber.html

[35] Justin Zobel, Steffen Heinz, and Hugh E. Williams, "In-memory hash tables for accumulating text vocabularies," *Information Processing Letters*, vol. 80, no. 6, pp. 271 - 277, 2001.

10 Zoznam tabuliek

11 Zoznam obrázkov

12 Príloha A – Obsah priloženého CD

Priložené CD obsahuje vygenerované testovacie dáta, výsledky jednotlivých experimentov, binárne súbory na spúšťanie testov, knižnicu s implementáciou samoupravujúcich zoznamov, grafy obsiahnuté v práci a samotný text tejto diplomovej práce.

Štruktúra CD:

- bin – adresár s binárnymi súbormi
- data – vygenerované testovacie dáta
- lib – knižnica s implementáciou
- src – zdrojové kódy
- results – výsledky experimentov
- graphs – graficky spracované výsledky
- diplomka.pdf – text diplomovej práce

13 Príloha B – Spúšťanie testov

V adresári bin na CD nájdete súbor diplomka.exe, pomocou ktorého je možné vykonávať jednotlivé testy. Na správny beh je potrebný operačný systém Microsoft Windows XP / Vista / 7 a Microsoft .NET Framework 3.5 alebo novší.

Spúšťanie testov má nasledujúcu syntax:

```
diplomka.exe testType algorithm inputSetFile inputTestsFile outputFile [measureAfterMembersCount]
```

kde

- testType – typ testu (convergence – test konvergencie, member – test vyhľadávania
- algorithm – stratégia (mtf – Move-To-Front, tsp – Transpose, tim0 – Timestamp(0.382), tim – Timestamp(0), k8 – MoveAheadK(i/2), k32 – MoveAheadK(N/32), cnt – Count, gen – klasický zoznam, srt – Sorted, oso-OSO)
- inputSetFile –súbor s testovacou množinou (súbory typu set.sav, set2.sav z adresára data)
- inputTestsFile – súbor s vyhľadávanou množinou (súbory typu test.sav z adresára z data)
- outputFile – výstupný súbor
- [measureAfterMembersCount] – počet operácií, po ktorých vykonaní ma začať meranie (pre test vyhľadávania)

Napríklad spustenie testu konvergencie na druhej testovacej množine z beta rozdelenia pre stratégiu Timestamp a uloženie výsledkov do súboru results.txt by vyzeralo

```
diplomka.exe convergence tsp data\beta-distribution\2\set.sav data\beta-distribution\2\test.sav results.txt
```

Výsledkom je súbor s poľom časov a poľom porovnaní odstupňovaný po 10 000 operáciách, tieto polia je možné ihneď vložiť do Matlabu a ďalej s nimi pracovať.

14 Príloha C – Použitie samoupravujúcich stratégií vo vlastných programoch

V adresári lib na CD nájdete implementáciu samoupravujúcich stratégií vo forme DLL knižnice pre platformu Microsoft .NET, ktorú môžete používať vo vlastných programoch. Ich použitie je možné akýmkoľvek typom implementujúcim rozhranie IComparable.

K dispozícií máte generické triedy CountList, MoveAheadKList, MoveToFrontList, SortedList, TimestampList, TransposeList, ktoré môžete používať vo svojich programoch namiesto klasického zoznamu.

14.1 Príklad použitia

Ak chcete použiť napríklad stratégiu MoveToFront pracujúcu s celými číslami, musíte pridať do projektu referenciu na SelfOrganizingLists.dll, pridajte do kódu odkaz na namespace

```
using SelfOrganizingLists;
```

a novú inštanciu zoznamu stratégie MoveToFront vytvoríte ako

```
MoveToFrontList list = new MoveToFrontList<int>()
```

Vytvorený zoznam môžete ďalej používať ako klasický zoznam.