Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava
Katedra Informatiky

Optimalizácia reálnych funkcií pomocou diskretizácie

(bakalárska práca)

Matúš Petruľák

Vedúci: Martin Pelikán, Ph.D.

Bratislava, 2008

Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Optimalizácia reálnych funkcií pomocou diskretizácie

(bakalárska práca)

Matúš Petruľák

Odbor: 9.2.1 Informatika
Vedúci: Martin Pelikán, Ph.D.

Bratislava, 2008

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Chcem sa poďakovať môjmu školiteľovi Martinovi Pelikánovi. Vysoko si cením jeho čas a ďakujem mu za všetko čo som sa mohol zo spolupráce s ním naučiť. Ďalej by som rád poďakoval rodine a mojim blízkym za podporu.

Abstrakt

Autor:	Matúš Petruľák
Názov práce:	Optimalizácia reálnych funkcií pomocou diskretizácie
Škola:	Univerzita Komenského
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katerda:	Katedra informatiky
Školiteľ:	Martin Pelikán, Ph.D.
Miesto:	Bratislava
Rok:	2008
Rozsah práce:	47 strán

Práca sa zaoberá riešením optimalizačných problémov definovaných na reálnych premenných použitím diskterizačných metód v evolučných algoritmoch. Pozornosť sa venuje histogramom s pevnou šírkou a výškou a problémom *Sphere* a *Two-peaks*. Medzi veľmi zaujímavé výsledky práce patrí vzťah medzi počom evaluácií potrebných na dosiahnutie dostatočne presného riešenia a veľkosťou populácie. Práca sa tiež zaoberá efektivitou rôznych druhov kríženia a vzťahom medzi veľkosťou problému a počtom potrebných evaluácii na jeho vyriešenie.

Kľúčové slová: diskretizácia, evolučný algoritmus, optimalizácia

Abstract

Autor:	Matúš Petruľák
Title:	Optimization of real functions by discretization
University:	Comenius University
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Department of Computer Science
Supervisor:	Martin Pelikán, Ph.D.
Place:	Bratislava
Year:	2008
Pages:	47

The thesis deals with solving the problems of optimization defined on the interval of real variables by means of methods of discretization in genetic algorithms. It focuses on fixed-width and fixed-height histograms and *Sphere* and *Two-peaks* problems. Among the most interesting results of the thesis is the relation between the number of evaluations needed for a precise enough solution and the population number. The thesis also takes up the issue of effectiveness of various crossing methods and the relation between the number of evaluations needed for its solving.

Key words: discretization, genetic algorithm, optimization

Obsah

Kapitola 1

Úvod

Táto práca opisuje moje úsilie, ktoré bolo venované skúmaniu riešenia optimalizačných problémov definovaných na reálnych premenných pomocou evolučných algoritmov. Evolučné algoritmy sú optimalizačné metódy inšpirované evolúciou a genetikou ktorú poznáme z bežnej prírody. Pre svoju použiteľnosť na rôzne problémy a jednoduchosť sa stávajú dôležitou oblasťou optimalizácie pomocou počítačov.

Existujú dva základné prístupy k riešeniu reálnych optimalizačných problémov pomocou evolučných algoritmov.

- Transformovať problém do diskrétnej reprezentácie a použiť evolučné algoritmy pre diskrétnu reprezentíciu.
- Problém ponechať v reálnej reprezentácii a všetky operátory evolučného algoritmu ako sú kríženie a mutácie aplikovať priamo na populáciu v reálnej reprezentácii.

Oba prístupy majú svoje výhody aj nevýhody. Výhodou prvého prístupu je, že z nekonečného priestoru získame konečný priestor s ktorým sa ľahšie pracuje. Rovnako v oblasti evolučných algoritmov, ktoré pracujú s diskrétnou

reprezentáciou sa urobilo veľa výskumu čo sa týka multimodálnych problémov a taktiež existuje niekoľko veľmi dobrých algoritmov ako napríklad *hierarchical Bayesian optimization algorithm* ktoré vedia pracovať len s diskrétnou reprezentáciou. Avšak nevýhodou je, že diskretizácia vedie ku strate presnosti a vzniká tak medzera medzi reprezentáciou a skutočným priestorom možných riešení, ktorý prehľadávame.

Základnou výhodou druhého prístupu je, že spomínaná medzera medzi reprezentáciou a skutočným priestorom možných riešení nie je. Podobne sa nemusíme obmedzovať rôznymi obmedzeniami a stratou presnosti, ktoré z diskretizácie vyplývajú.

1.1 Cieľ práce

Veľa výskumníkov a užívateľov môže mať záujem riešiť ich optimalizačný problém definovaný na reálnych premenných pomocou evolučných algoritmov. Avšak existuje málo práce, ktorá by dávala návod, či základ aký druh diskretizácie pre daný problém použiť a ako jednotlivé parametre ovplyvňujú kvalitu riešenia, prípadne aký druh kríženia použiť pre daný optimalizačný problém či spôsob diskretizácie.

V mojej práci budem hľadať odpovede na tieto otázky. Venovať sa budem konkrétne prístupu, ktorý pracuje s diskrétnou reprezentáciou problému. V kapitole 2 popíšem jednoduchý evolučný algoritmus riešiaci problémy v diskrétnej reprezentácii, ktorý budem používať. Následne v kapitole 3 sa zameriam na jednoduché prístupy k diskretizácii - konkrétne fixed width a height histogramy. V kapitole 4 dám dokopy poznatky z kapitol 2 a 3 a dostanem algoritmus, ktorý rieši optimalizačné problémy definované na reálnych premenných. Kapitola 5 ukazuje návrh experimentov, ktorých výsledky budú prezentované v kapitole 6. Nakoniec v kapitole 7 zhrniem vyhodnotenie výsledkov.

Kapitola 2

Jednoduchý evolučný algoritmus

Evolučný algoritmus je simulácia genetického stavu populácie pozostávajúcej z jedincov. Pričom každý jedinec reprezentuje nejaké riešenie problému a celá populácia je vzorkou priestoru riešení. Simulujeme hlavne operácie, o ktorých si myslíme, že sú najviac rozhodujúce v prírode. Sú to:

- prirodzený výber - selekcia
- mutácia
- kríženie - crossover, alebo v prírode párenie

My sa zameriame na evolučné algoritmy ako na prostriedok na riešenie optimalizačných problémov. Evolúcia v prírode vedie k jednotlivcom, ktoý sú lepšie prispôsovený na ich prostredie. Ak teda vieme vyrátať, ako je konkrétny jedinec vhodný pre dané prostredie, evolučné algoritmy nám môžu pomôcť vyvinúť jedinca vhodného pre zadanú úlohu.

Cieľom tejto kapitoly je vysvetliť evolučný algoritmus, ktorý bude následne použitý v experimentoch. Najprv uvediem algoritmus a neskôr prezentujem jednotlivé modifikácie algoritmu.

2.1 Jedonduchý evolučný algoritmus

Najprv si musíme uvedomiť, že algoritmus nemá o štruktúre problému skoro žiadne informácie. Jediným prostriedkom ako sa niečo dozvedieť je vygenerovať populáciu a každého jedinca ohodnotiť podľa toho, ako vyhovyje zadaným podmienkam. Algorimus následne pokračuje v iterovaní, kde v každej iterácii vygeneruje nových jedincov a výsledky vyhodnotenia použije na to, aby daľšia generácia bola lepšia. V iteálnom prípade sa kvalita týchto jedincov časom zlepší a po vhodnom počte iterácií dosiahne globálne optimum. Udržiavanie si populáciu jedincov má v kontraste s jedným jedincom viacero výhod. Umožňuje napríklad súčasné skúmanie vo viacerých smeroch prehľadávaného priestoru a dovoľuje použitie učiacich techník na zistenie zákonitostí problému.

Evolučný algoritmus nestavia žiadne obmedzenia na reprezentáciu jedincov, alebo spôsob akým budú jedinci ohodnotení. Jedinec môže byť reprezentovaný rôznymi spôsobmi. Od binárnych vektorov, vektorov reálnych čísiel až po kusy programového kódu. Ohodnotenie môže byť založené na funkcii v počítači, simulácii, interakcii s človekom alebo kombinácii predchádzajúcich. Rovnako môžene použiť operátor čiastočného usporiadania na priestor všetkých riešení problému, čím dostaneme relatívne porovnanie kvality dvoch jedincov. Avšak pre jednoduchosť budeme v zbytku práce predpokladať, že jedinci sú reprezentovaní binárnymi vektormi a že ohodnotenie určuje jedno reálne číslo nazvané *fitness*.

Zvyšok kapitoly popisuje konkrétne algorimus a jeho procedúry. Základné procedúry jednoduchého evolučného algortimu pozostávajú z:

- **Inicializácia.** Evolučný algoritmus obyčajne vygeneruje prvú populáciu rovnomerne náhodne zo všetkých možných jedincov, čiže riešení problému.

- **Selekcia.** Na začiatku každej iterácie algoritmus vyberie sľubné rieše-

nia z pôvodnej populácie v závislosti na ohodnotení jednotlivých riešení. Napriek tomu, že môžeme použiť rôzne spôsoby selekcie, všetky majú jednu spoločnú ideu - vybrať viac dobrých riešení do novej populácie na úkor tých čo sú horšie ohodnotené.

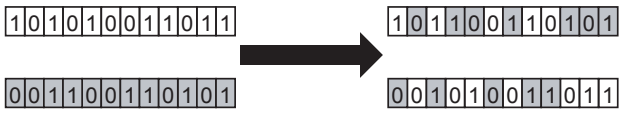
V práci budem používať Tournament selection. Ak chceme dostať novú populáciu veľkosti N, zorganizujeme turnaj s N kolami. Z každého kola vyjde jeden víťaz, ktorý sa dostane do novej populácie. V každom kole sa vyberie náhodných k riešení z pôvodnej populácie a z nich najlepšie sa stane víťazom. Jedno riešenie sa môže zúčastniť aj viacerých turnajov a v jednom prehrať a v inom vyhrať. Velkosť turnaja určuje kvalitu víťaza - tzv. *selection pressure*. Čím je totiž väčšia veľkosť turnaja, tým je väčšia šanca, ze víťaz bude lepšie ohodnotený. V mojom prípade budem používať Binary tournament selection, čiže turnajov sa budú zúčastňovať práve dve riešenia.

Existujú aj ďalšie spôsoby selekcie, ktoré však nebudem používať. Ako príklad uvediem Roulette wheel selection, alebo Truncation selection, kde sa do novej populácie vyberie najlepších 1/s riešení z pôvodnej populácie. Kde parameter s určuje *selection pressure*.

- **Variácia** Následne sa aplikovaním kríženia a mutácie vytvoria zo sľubných riešení nové riešenia. Kríženie (rekombinácia) kombinuje jednotlivé riešenia tak, že vymieňa niektoré ich časti, obyčajne s nejakou zadanou pravdepodobnosťou. Mutácia mierne pozmení nové riešenia tak aby sa preskúmali aj riešenia v ich blízkom okolí.

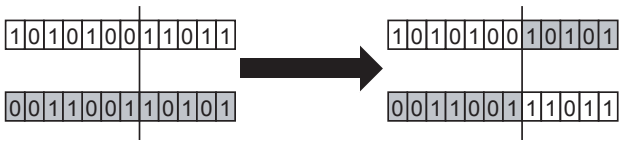
  V mojej práci budem používat kríženia One-point a Uniform.

  - **One-point.** One-point kríženie náhodne vyberie miesto v 2 vektoroch a vymení všetky bity ktoré sa nachádzajú za týmto miestom medzi vektormi.



Obr. 2.1: Kríženie Uniform

– **Uniform.** Kríženie Uniform pre každú pozíciu vymení bit na danej pozícii medzi 2 vektormi s pravdepodobnosťou 50%.



Obr. 2.2: Kríženie One-point

Pre binárne vektory sa obyčajne používa bit-flip mutácia. Funguje tak, že s určitou pravdepodobnosťou "prepne" bit. Pravdepodobnosť je ovyčajne malá, aby nenastali veľké zmeny.

Evolučným algoritmom, kde je hlavným zdrojom variácie kríženie (rekombinácia) za zvykne hovoriť aj selektorekombinatívne evolučné algoritmy.

- **Nahradenie.** Po variácii populácia nových riešení nahradí pôvodnú populáciu a vykoná sa ďalšia iterácia. Čiže znovu selekcia, varíácia, nahradenie. Iterácie sa opakujú pokiaľ nenastane nejaká ukončujúca podmienka. Napríklad populácia konverguje k jednému riešeniu, alebo populácia obsahuje vyhovujúce riešenie, prípadne populácia pozostáva z veľkej väčšiny len z vyhovujúcich riešení, alebo pri prekročení maximálneho počtu iterácií.

  – **Elitizmus.** Spôsob nahradenia starej populácie novou sa dá mierne modifikovať. Môže sa totiž stať, že krížením a mutáciou stratíme

najlepšie riešenie, ktoré sme dovtedy našli. Elitizmus je meno spôsobu, ktorý pri vytváraní novej populácie skopíruje nové riešenia spolu s niekoľkými najlepšími riešeniami z pôvodnej populácie. Tento spôsob som použil aj ja v mojej práci.

Doteraz som zhrnul ako funguje jednoduchý evolučný algoritmus, ktorý použijem v mojej práci. Na nasledovných riadkoch uvádzam pseudo-kód algoritmu.

Algorithm 1 Jednoduchý evolučný algoritmus

procedure evolučný algoritmus

1: $old \leftarrow$ generatePopulation();
2: **while** StopConditionNotMet **do**
3: $promising \leftarrow$ selection(old);
4: $i \leftarrow 0$
5: **while** $i \leq populationSize$ **do**
6: cross($individuum[i], individuum[i+1]$);
7: $i \leftarrow i + 2$;
8: **end while**
9: $new \leftarrow$ mutation (new);
10: $old \leftarrow$ mergeElite(new, old);
11: **end while**

Text voľne podľa [Pel05, Har99].

Kapitola 3

Diskretizácia

Diskretizácia sa používa, keď chceme previesť reálny problém na diskrétny problém. Diskretizácia môže často znížiť zložitosť problému, či zjednodušiť zložité problémy. Typickým spôsobom zvyknú diskretizačné metódy fungovať je, že rozdelia rozsah nejakej premennej na 2^k intervalov. Jednotlivé reprezentácie reálnej premennej môžu byť potom reprezentované k bitmi, ktoré určujú interval v ktorom sa nachádza dané reálne číslo.

V tejto kapitole prezentujem dve diskretizačné metódy. Sú to:

- **Histogram s pevnou šírkou - fixed width**
- **Histogram s pevnou výškou - fixed height**

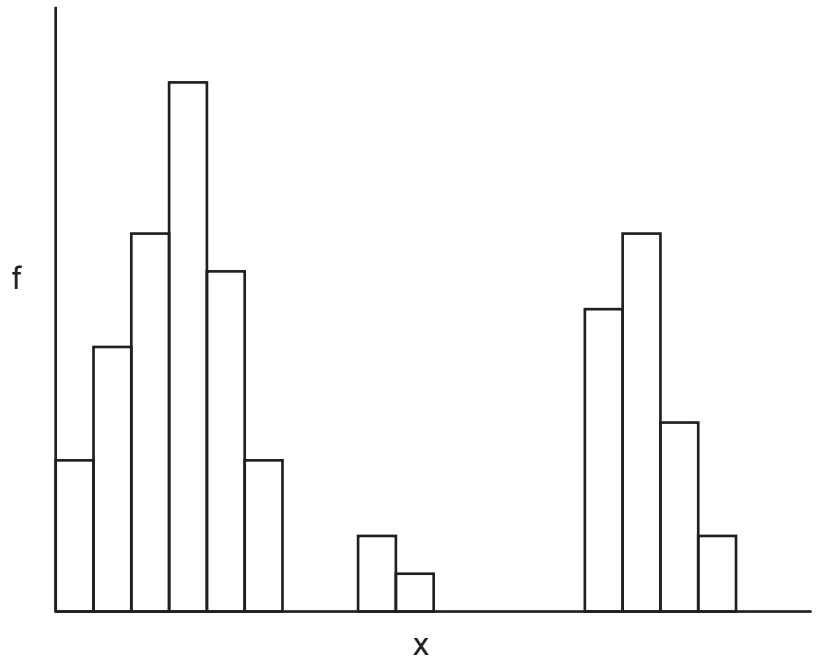
Obe patria do kategórie *unsupervised* diskterizačných metód. Naproti tomu existujú *supervised* metódy. Rozdiel spočíva v tom, že *supervised* metódy využívajú tzv. *class labels*. Tieto *class labels* sú priraďované jednotlivým riešeniam na základe toho, či boli alebo neboli vybraté do novej populácie. [CP]

3.1 Histogram s pevnou šírkou

Táto metóda je veľmi jednoduchá a je ľahko implementovateľná. Základná idea spočíva v rozdelení intervalu na $n = 2^k$ podintervalov, ktoré majú rovnakú šírku. Predpokladajme teda, že máme interval $[a, b)$. Potom i-ty podinterval je

$$\left[a + \frac{i(b-a)}{n}, a + \frac{(i+1)(b-a)}{n}\right)$$

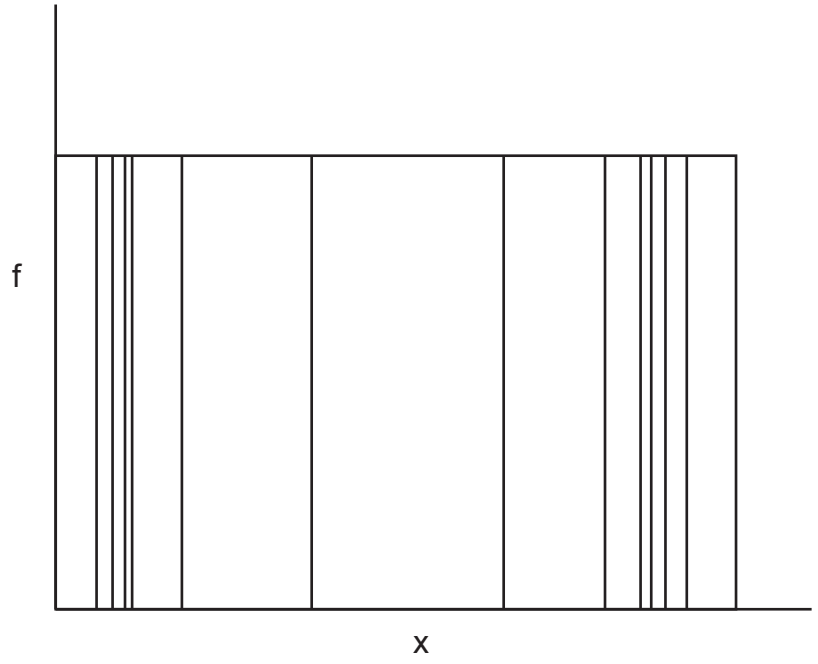
kde $i \in \{0, ..., n-1\}$. Nevýhoda histogramov s pevnou šírkou sa prejaví v prípade, že body sú zoskupené okolo pár regiónov. Vtedy iba pár podintervalov nie je prázdnych. Rovnako nám táto metóda nič nepovie o rozložení bodov v celom intervale.



Obr. 3.1: Rozdelenie intervalov - Histogram s pevnou šírkou

3.2 Histogram s pevnou výškou

Podobná s predchádzajúcou metódou je táto metóda, ktorá rozdelí interval na n podintervalov tak, aby v každom podintervale bolo zhruba rovnako veľa bodov. Teda v regiónoch, kde sú body hustejšie budú aj podintervaly bližšie k sebe, naopak v oblastiach, kde je menej bodov budú podintervaly redšie. Toto rozdelenie podintervalov reprezentuje lepšie rozloženie bodov v celom intervale.



Obr. 3.2: Rozdelenie intervalov - Histogram s pevnou výškou

V texte som uviedol metódy diskterizácie, ktoré budem používať. Obrázky znázorňujú rozdelenie podintervalov a počet bodov v jednotlivých intervaloch. Ukazujú, že v prípade histogramu s pevnou šírkou môže byť veľa

podintervalov nevyužitých a rovnako, že histogram s pevnou výľkou dáva informáciu o rozdelení populácie.

Text voľne podľa [PGT01].

Kapitola 4

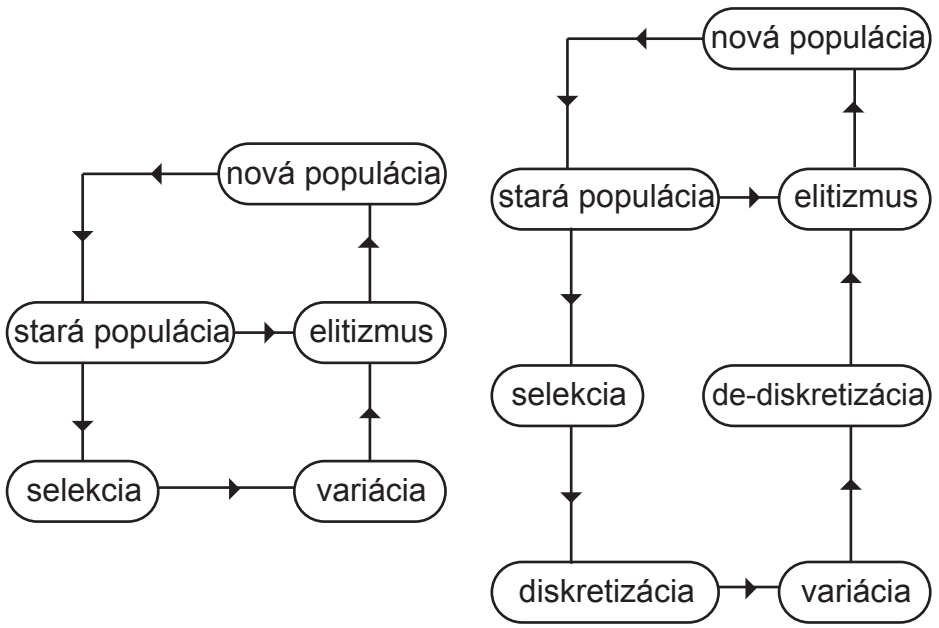
Jednoduchý evolučný algoritmus + Diskretizácia

V kapitole 2 som prezentoval jednoduchý evolučný algoritmus a v kapitole 3 naznačil ako bude fungovať diskretizácia. V tejto kapitole ukážem, ako pomocou jednoduchého evolučného algoritmu, ktorý rieši diskrétne problémy dostaneme pomocou diskretizácie algoritmus na riešenie reálnych problémov. Najprv porovnám schému jednoduchého evolučného algoritmu s diskretizáciou a bez diskretizácie, následne sa zameriam na jeho podrobnejší popis a ku koncu ukážem jednotlivé modifikácie.

4.1 Algoritmus

Jednotlivé kroky algoritmu pozostávajú z:

- **Inicializácia**
- **Selekcia**
- **Diskretizácia**



(a) (b)

Obr. 4.1: Jednoduchý evolučný algoritmus na obázku (a) v porovnaní s jeho verziou, ktorá rieši aj reálne problémy na obrázku (b).

- **Variácia**
- **De-diskretizácia**
- **Nahradenie**

Tieto kroky sú skoro identické s jednoduchým evolučným algoritmom pre diskrétne problémy. Hlavný rozdiel je v tom, že populáciu netvoria binárne vektory, ale sú to vektory reálnych čísiel. Jediným miestom, kde záleží na tom ako je reprezentovaná populácia je **variácia**. Vyššie uvedené operácie ako kríženie a mutácia totiž vyžadujú diskrétnu reprezentáciu riešení. Preto je potrebné vektory reálnych čísiel pred variácou diskretizovať a po variácii tieto binárne vektory znovu namapovať spať na reálne vektory. Toto zabezpečujú kroky diskretizácia a de-diskretizácia.

4.2 Diskretizácia

Majme reálny problém o veľkosti n reálnych premenných. Tieto premenné sú normálne reprezentované reálnymi číslami. Úlohou diskretizácie je z tejto reálnej reprezentácie dostať reprezentáciu diskrétnu. Budeme postupovať pre každú premennú zvlášť. Musíme totiž pre každú premennú nezávisle spustiť diskretizačnú metódu. V našom prípade to bude metóda používajúca histogramy s pevnou šírkou alebo výškou. Táto metóda dostane na vstup všetky reprezentácie nejakej konkrétnej premennej, ktoré sa nachádzajú v populácii. Na základe toho vyráta ako budú jednotlivé podintervaly vyzerať. Každému reálnemu číslu priradí číslo podintervalu v ktorom sa nachádza a to zakóduje do k-bitových reťazcov. Uvedomme si, že jednotlivých podintervalov je 2^k a teda k bitov na kódovanie čísla podintervalu postačuje. Výsledkom diskretizácie napokon je populácia binárnych vektorov, každý s dĺžkou nk, pričom bity $0, ..., k-1$ reprezentujú podinterval v ktorom sa nachádza 0. reálne čislo,

až po bity $nk-k,..,,nk-1$ reprezentujúce podinterval v ktorom sa nachádza $n-1$. reálne číslo.

4.3 De-diskretizácia

Tento proces je opačný k diskretizácii. Jedná sa o mapovanie binárnych vektorov späť do reálnych. Majme nk bitový vektor. Bity $ik, ..., i(k+1)-1$ kde $i \in 0, ..., n-1$ reprezentujú podinterval, v ktorom sa i-te reálne číslo nachádza. V tomto momente sa rovnomerne náhodne zvolí číslo z daného podintervalu. Opakovaním tohoto postupu sa vyráta reálna reprezentácia celej populácie.

4.4 Modifikácie

V nasledujúcom texte popíšem jednotlivé modifikácie, ktoré môžu mat na beh algoritmu vplyv. Modifikácie sa týkajú krížení a aj samotných metód diskretizácie.

4.4.1 Kríženia

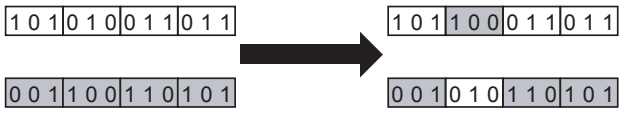
V doterajšom texte som uviedol dva spôsoby kríženia. Sú to:

- **Uniform crossover**
- **One-point crossover**

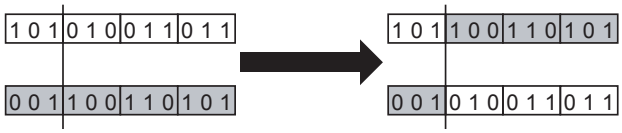
Kríženie sa aplikuje na binárne vektory. Avšak predchádzajúci text nám umožnil spoznať štruktúru binárnych vektorov. Binárny vektor má dĺžku nk, teda pozostáva z n k-bitových blokov. Každý k-bitový blok prislúcha jednému reálnemu číslu. Môžeme si teda dovoliť uviesť daľšie dva spôsoby kríženia, ktoré sú odvodené od pôvodných dvoch. Sú to:

- **Building-block wise Uniform crossover**
- **Building-block wise One-point crossover**

Základnou vlastnosťou týchto krížení je to, že nerozbíjajú jednotlivé bloky bitov. Môžeme sa na to pozerať akoby sme krížili spolu dva vektory ktoré pozostávajú z n blokov dĺžky k bitov. Tieto kríženia budem na grafoch označovať ako UniformBlock a OnePointBlock.



Obr. 4.2: Kríženie Building-block wise Uniform - veľkosť bloku 3 bity



Obr. 4.3: Kríženie Building-block wise One-point - veľkosť bloku 3 bity

4.4.2 Diskretizačné metódy

V nasledujúcom texte navhnem dva spôsoby ako môže byť diskretizačná metóda používajúca histogramy s pevnou šírkou či výskou implementovaná.

1. Na začiatku behu algoritmu sa pevne zvolí interval z ktorého môžu byť reprezentácie danej reálnej premennej. Tento interval sa ďalej počas behu algorimu nemení. Z toho napríklad vyplýva, že poditervaly v histograme s pevnou šírkou majú v každej iterácii algorimu rovnakú veľkosť.

2. Interval, z ktorého sú reprezentácie danej reálnej premennej sa mení počas behu algoritmu. Konkrétne sa mení pri každej iterácii a to tak, aby daný interval obsahoval všetky reprezentácie danej reálnej premennej ale aby bol čo najmenší. Teda pokiaľ bude populácia konvergovať k nejakému bodu, tak tento interval bude čoraz menší. Výhoda tejto implementácie sa prejaví hlavne pri histograme s pevnou šírkou. Ak populácia totiž konverguje k nejakému bodu a interval sa zmenšuje, tak aj podintervaly sa zmenšujú a je možné dosiahnuť lepšiu granularitu a presnejšie sa priblížiť k optimálnemu riešeniu. Tento spôsob však monotónne zmenšuje priestor možných riešení. Ale keď algoritmus príliš rýchlo zúži priestor riešení (napríklad môže zúžit priestor na oblasť kde je nejaké lokálne optimum daného problému), tak sa nikdy nemusí dopracovať k riešeniu problému. [CP]

Spoločnou nevýhodou oboch spôsobov je to, že neumožňujú reprezentovať nesúvislé oblasti sľubných riešení. Algoritmus musí totiž diskretizovať súvislý interval čo môže zahrňovať aj oblasti, kde sa nenachádza žiadne riešenie. Toto môže spôsobovať problémy hlavne pri použití histogramu s pevnou šírkou.

Text voľne podľa [PGT01, CP].

Kapitola 5

Návrh experimentov

V nasledujúcom texte uvediem reálne optimalizačné problémy, ako aj konkrétne parametre evolučného algoritmu, ktoré boli použité pri experimentoch.

5.1 Reálne optimalizačné problémy

V mojej práci som sa pri testovaní zameral na dve funkcie: funkcia two-peaks a funkcia sphere. Obe funkcie sú zložené z menších funkcií, ktorých príspevky k celkovej $fitness$ sa spolu zrátajú a dostaneme celkovú $fitness$. Jednotlivé dunkcie sú definované nasledovne:

- **Sphere**

$$sphere(x_0, ..., x_{n-1}) = \sum_{i=0}^{n-1} x_i^2$$

- **Two-peaks**

$$twoPeaks(x_0, ..., x_{n-1}) = \sum_{i=0}^{n-1} f_{two-peaks}(x_i)$$

  Každá premenná z two-peaks funkcie prispieva do celkovej $fitness$ na-

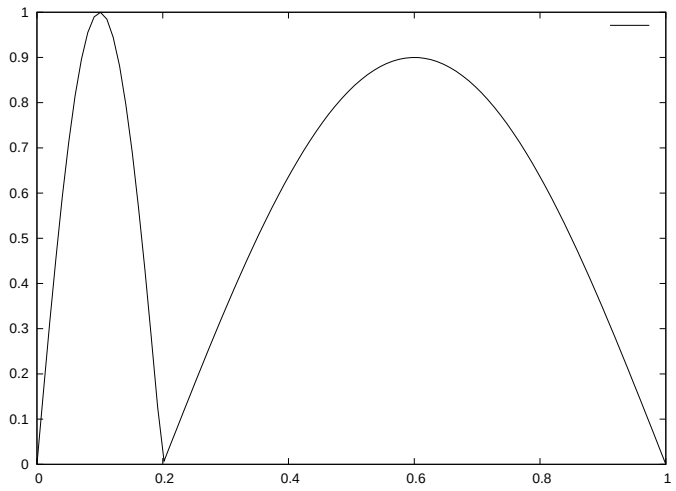
sledovne:

$$f_{two-peaks}(x) = \begin{cases} f_{peak}(x/0.2, 1) & \text{ak } x < 0.2, \\ f_{peak}((x-0.2)/0.8, 0.9) & \text{inak} \end{cases}$$

kde f_{peak} je jednoduchá funkcia pre jeden "vrchol" definovaná takto:

$$f_{peak}(x, h) = h\cos(\pi(x - 0.5))$$

Na obrázku je znázornená funkcia $f_{two-peaks}(x)$. Funkcia má jedno lokálne optimum a jedno globálne optimum. Z toho vyplýva 2^n optím pre problém o veľkosti n reálnych premenných, pričom len jedno optimum je globálne. Navyše lokálne optimá sú o mnoho širšie a skoro rovnako dobré ako globálne optimum, čo ešte sťažuje problém. [PGT01]



Obr. 5.1: približné zobrazenie funkcie $f_{two-peaks}(x)$

5.2 Experimenty

Boli otestované nasledovné problémy

- funkcia Sphere
  - veľkosť problému 10,20,30,40 a 50 reálnych premenných pri použití histogramov s pevnou výškou
  - veľkosť problému 10,12,14,16 a 18 reálnych premenných pri použití histogramov s pevnou šírkou
- funkcia Two-peaks
  - rovnaké veľkosti problému a ako v prípade Sphere

Jednotlivé nastavenia evolučného agoritmu boli nasledovné:

- problém som považoval za vyriešený, ked algorimus bol 30x po sebe schopný vyriešiť daný problém (teda po menej ako 40000 iteráciách sa v populácii objavilo dostatočne dobré riešenie)
- riešenie som považoval za dostatočne dobré, keď sa každá jeho premenná priblížila k hodnote optimálneho riešenia na menej ako 1/64.
- skúšal som štyri metódy kríženia: Uniform, Building-block wise Uniform, One-point a Building-block wise One-point
- jednotlivé reálne premenné som pri diskretizácii kódoval 5,6,8 a 10 bitmi
- metóda $cross(x, y)$ spravila kríženie jedinca x s y s pravdepodobnosťou pc=0.6
- počet evaluácii som rátal ako priemer zo všetkých 30 behov algoritmu, pričom 3 najhoršie a 3 najlepšie som nezarátal.

- skúšal som problémy riešiť s veľkosťou populácie od 4 do 1500
- pravdepodobnosť mutácie jedného bitu je pm=1/(veľkosť problému v bitoch)

Kapitola 6

Výsledky

Táto kapitola prezentuje výsledky, ktoré som experimentálne nameral podľa postupu popísaného v kapitole 5.

6.1 Histogramy s pevnou výškou

Uvediem výsledky, ktoré sa mi podarili namerať pri počte 5 bitov na reálnu premennú.

6.1.1 Počet evaulácii vs. veľkosť populácie

Meral som počet evauluácii a dával som to do súvisu s veľkosťou populácie, pričom všetky ostatné parametre ostali zachované. Ukazuje sa lineárny rast počtu evaluácii v závislosťi od veľkosti populácie, rovnako pre problém *Sphere* ako aj *Two-peaks*. Pekne je to znázornené na obrázkoch A.1, alebo A.2. Všimol som si jednu zaujímavú vec. Najmenší počet evaluácii nie je dosiahnutý pri minimálnej populácii s ktorou algoritmus dokáže úspešne vyriešiť problém. Ukazuje to napríklad graf na obrázku A.3 pre problém *Sphere* ako aj graf na obrázku A.4 pre problém *Two-peaks*.

6.1.2 Počet evaulácii vs. veľkosť problému

Algoritmus pre funkciu *Shpere* nájde riešenie s polynomiálnou časovou zložitosťou v závislosti od veľkosti problému. Graf zobrazujúci závislosť počtu evaluácii od veľkosti problému pre funkciu *Sphere* je na obrázku A.11.

To, či algoritmus pre funkciu *Two-peaks* nájde riešenie tiež s polynomiálnou časovou zložitosťou v závislosti od veľkosti problému nie je z grafu na obrázku A.12 úplne jasné. Ak však pripustíme chybu pri meraní počtu evaluácii keď veľkosť problému bola 60, tak graf ukazuje tiež polynomiálnu závislosť rovnako ako pre funkciu *Sphere*

6.1.3 Počet evaluácii vs. operátor kríženia

Uvediem graf, ktorý ukazuje závislosť počtu evaluácii od veľkosti problému pre jednotlivé spôsoby kríženia. Pre funkciu *Sphere* je na obrázku A.13 a pre funkciu *Two-peaks* je to obrázok A.14. Zhodnotiť, ktorý operátor kríženia najviac vyhovuje danému problému nebude až také jednoznačné. Pre problém *Sphere* sa ukazuje najvýhodnejší spôsob kríženia *Uniform*, ktorý je obzvlášť vhodný pre problémy o veľkosti do 50 reálnych premenných.

Pre problém *Two-peaks* sa javí najvhodnejšie zvoliť metódu nazvanú *Building-block wise Uniform crossover*. Naopak metóda *Uniform* vyšla ako najmenej vhodná. Aspoň pre prípad, že použijeme 5 bitov na reálnu premennú. V nasledujúcej sekcii ukážem, že pri použití viac bitov na premennú tomu tak byť nemusí a metóda *Uniform* je najvhodnejšia pre problém *Two-peaks*, keď pre jednu reálnu premennú použijeme bitov 8, či 10. Vidno to na obrázku A.15.

Výsledky taktiež poukazujú na to, že kríženia, ktoré zohľadňujú bloky bitov tvoriace jednotlivé reálne premenné prinášajú žiadne, alebo nevýrazné zlepšenie efektivity.

6.1.4 Počet evaluácií vs. počet bitov na premennú

Skúmal som, ako vplýva pošet bitov na riešenie. Nenazbieral som dostatok meraní k vysloveniu a overeniu vzťahov medzi počtom evaluácii a počtom bitov na premennú. Avšak podľa obrázku A.15 a obrázkov A.16 až A.23 môžem povedať, že pri zväčšovaní počtu bitov viac záleží na metóde kríženia. Pre väčší počet bitov na premennú najmenší počet evaluácii bol dosiahnutý pri použití kríženia *Uniform*.

6.2 Histogramy s pevnou šírkou

Výsledky boli namerané pri počte 5 bitov na reálnu premennú

6.2.1 Počet evaulácii vs. veľkosť populácie

V prípade histogramov s pevnou šírkou závislosť počtu evaluácii od veľkosti populácie je lineárna pre problém *Sphere* ako aj *Two-peaks*. Aj v tomto prípade platí, že najmenší počet evaluácii nie je dosiahnutý pri minimálnej populácii s ktorou algoritmus dokáže úspešne vyriešiť problém. Grafy možno vidieť napríklad na obrázku A.24 či A.25.

6.2.2 Počet evaulácii vs. veľkosť problému

Závislosť počtu evaluácii od veľkosti problému je zobrazená na obrázku A.34 pre problém *Sphere* a A.35 pre problém *Two-peaks*. Kvôli nedokonalosti mojich meraní a malému počtu výsledkov neviem určiť, či ide o polynomiálnu závislosť, alebo až o exponenciálnu.

6.2.3 Počet evaluácii vs. operátor kríženia

Pre funkciu *Two-peaks* bol najmenší počet evaluácii dosiahnutý s metódou *Building-block wise Uniform crossover*. Pre funkciu *Sphere* dosahovali všetky testované metódy približne rovnaké počty evaluácii. Náhľad umožňujú obrázky A.24 až A.33.

Kapitola 7

Záver

V práci som sa venoval optimalizácii funkcií *Sphere* a *Two-peaks*. Urobil som nemalé množstvo experimentov a meraní a výsledky prezentoval v kapitole 6.

Ako veľmi zaujímavý výsledok považujem vzťah medzi počtom evaluácii potrebných na dosiahnutie dostatoťne presného výsledku a veľkosťou populácie. Ukázalo sa totiž, že od istej veľkosti populácie zmenšovanie veľkosti populácie zväčšuje počet evaluácií. Očakávaný bol výsledok, že zmenšovanie populácie povedie k zmenšovaniu počtu evualuácií, alebo povedie k neschopnosti algorimu úspešne vyriešit problém s takou malou populáciou.

Výsledky, ktoré ukázali, že závislosť medzi počtom evaluácii a veľkosťou problému je poynomiálna boli očakávané. Práca navyše zobrazuje efektivitu jednotlivých druhov krížení. Poukazuje na to, že kríženia, ktoré zohľadňujú bloky bitov tvoriace jednotlivé reálne premenné prinášajú žiadne, alebo nevýrazné zlepšenie efektivity.

Zajímavým smerom sa javí skúmať ako vplýva počet bitov, ktoré reprezentujú jednotlivé reálne premenné na efektivitu algoritmu. Týmto smerom som v práci urobil príliš malý počet experimentov. Odporúčaním môže byť skúsiť väčšie populácie a urobiť viac experimentov. Výsledok, ktorý som v

tejto oblasti skúmania dosiahol je, že pri väčšom počte bitov efektívnosť algoritmu viac ovlyvňuje zvolená metóda kríženia.

Dodatok A

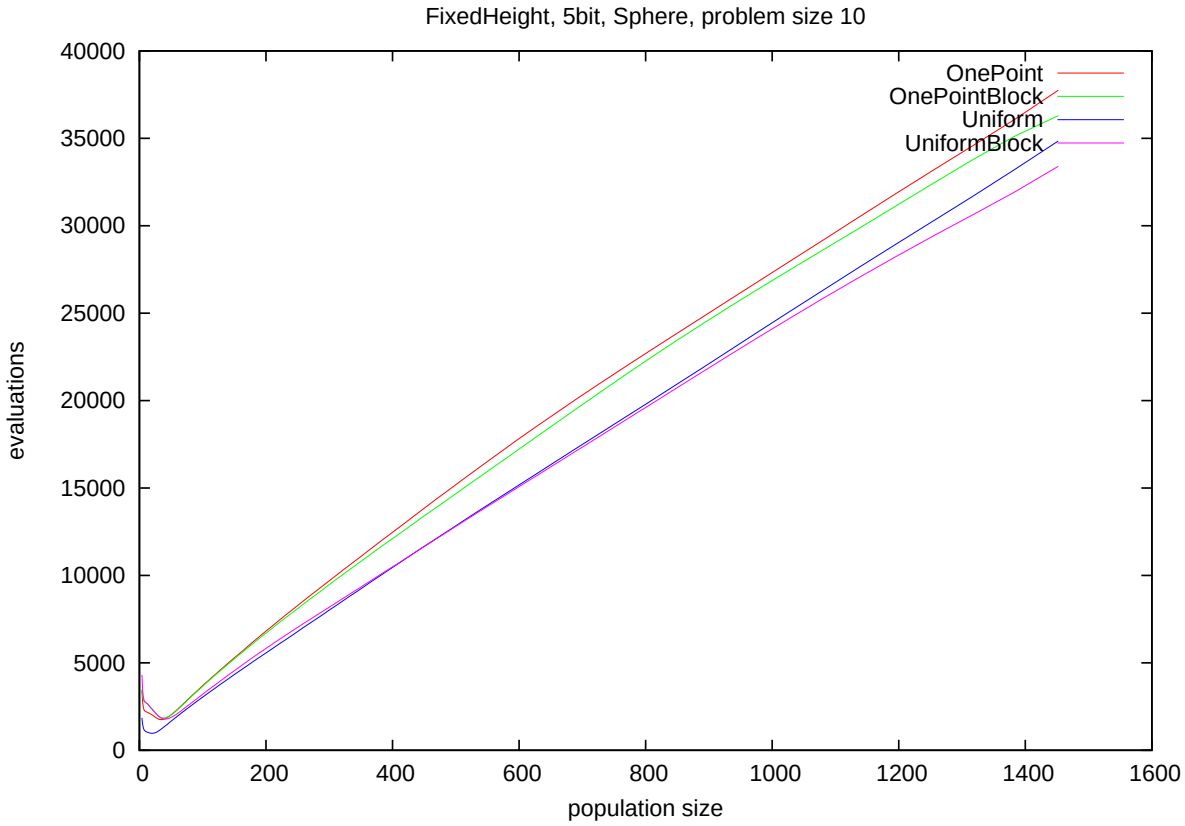
Obrázky

V tejto sekcii sú uvedené grafy s nameranými výsledkami. Každý obrázok má v sebe popis, ktorý stručne vysvetľuje k akému meraniu a s akými parametrami patrí. Grafy na obrázkoch A.1 až A.10 dávajú do súvisu počet evaluácií a veľkosť populácie, použitá metóda histogramov s pevnou výškou pre funkcie *Sphere* a *Two-peaks* a veľkosti problému 10 až 50 reálnych premenných a 5 bitov na premennú.

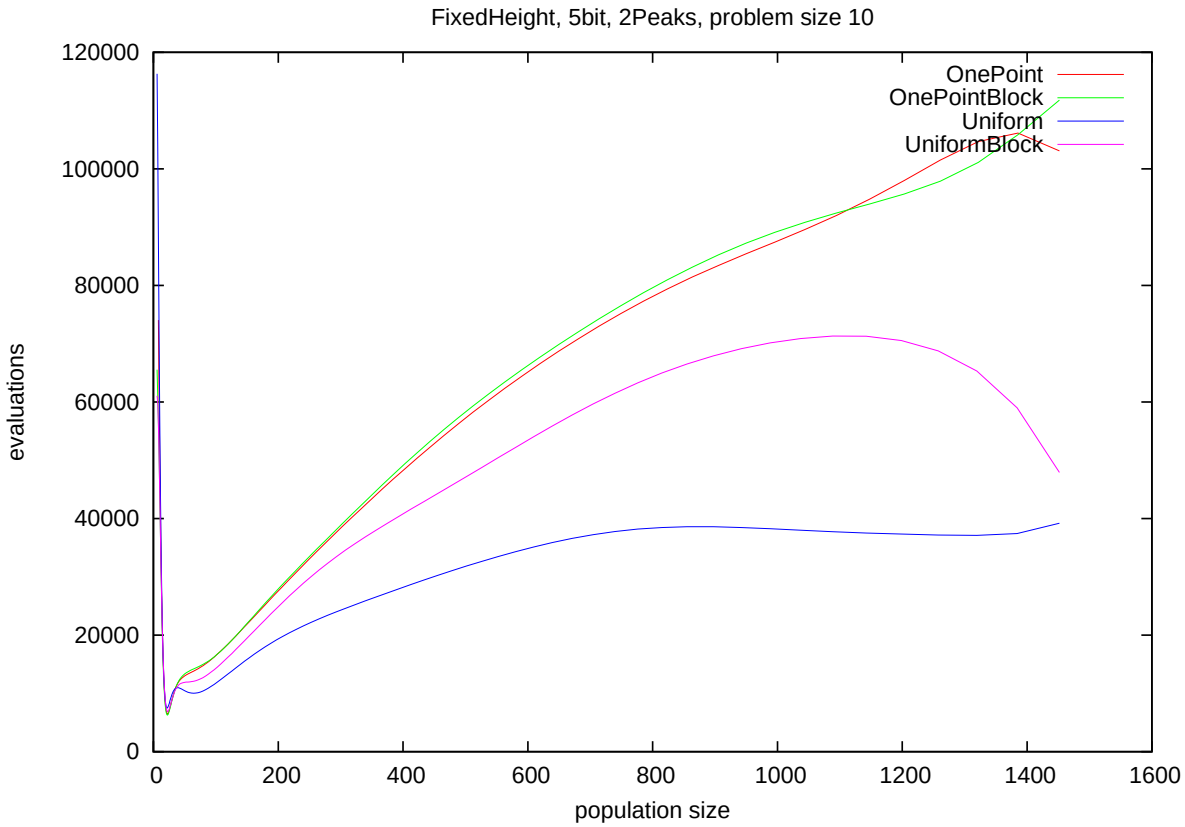
Grafy na obrázkoch A.11 až A.14 dávajú do súvisu počet evaluácii a veľkosť populácie. Merania boli použité tie isté ako pri grafoch na obrázkoch A.1 až A.10. Graf na obázku A.15 popisuje merania pri použití 10 bitov na premennú.

Grafy na obrázkoch A.16 až A.23 znázoňujú vzťah medzi počtom evaluácii a veľkosťou problému. Výsledky sú namerané pri nasledovnom počte bitov na premennú: 5, 6, 8 a 10.

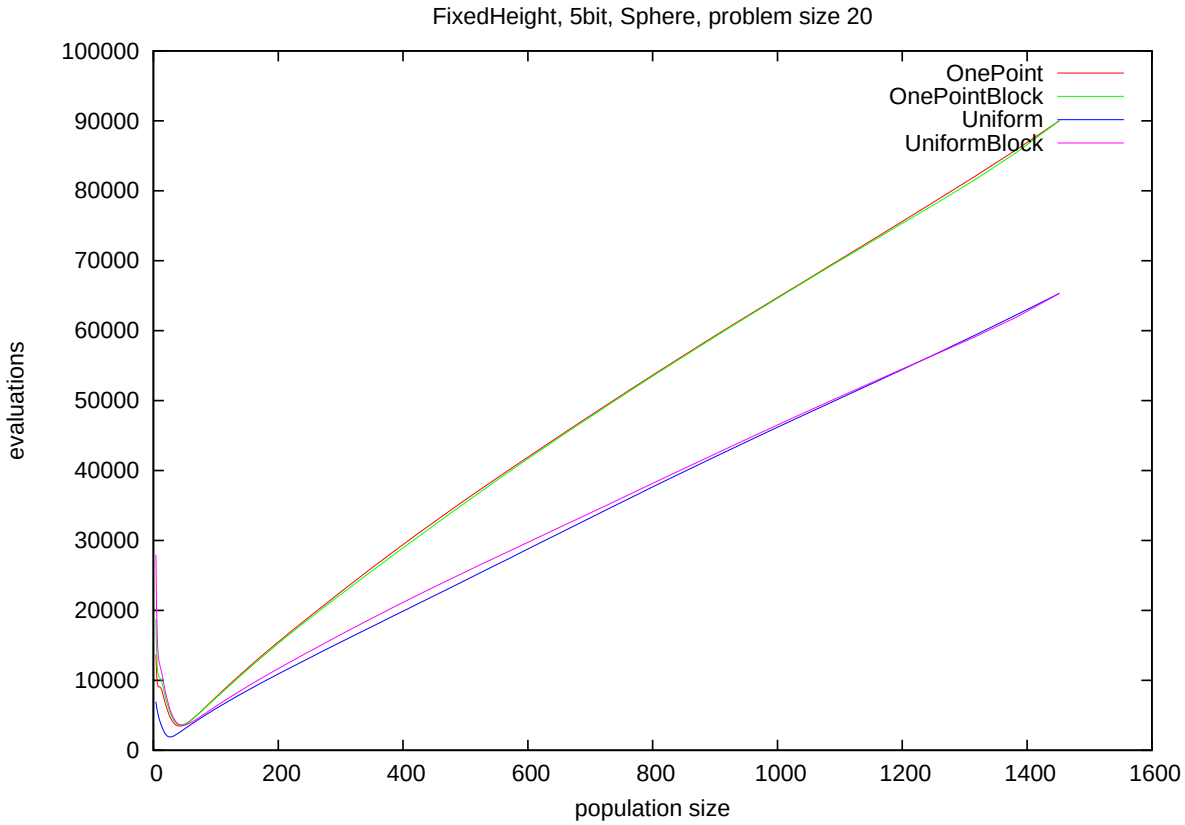
Grafy na obrázkoch A.24 až A.35 patria k meraniam, pri ktorých bola použitá metóda histogramov s pevnou šírkou.


FixedHeight, 5bit, Sphere, problem size 10
OnePoint
OnePointBlock
Uniform
UniformBlock
evaluations
population size
0
5000
10000
15000
20000
25000
30000
35000
40000
0
200
400
600
800
1000
1200
1400
1600

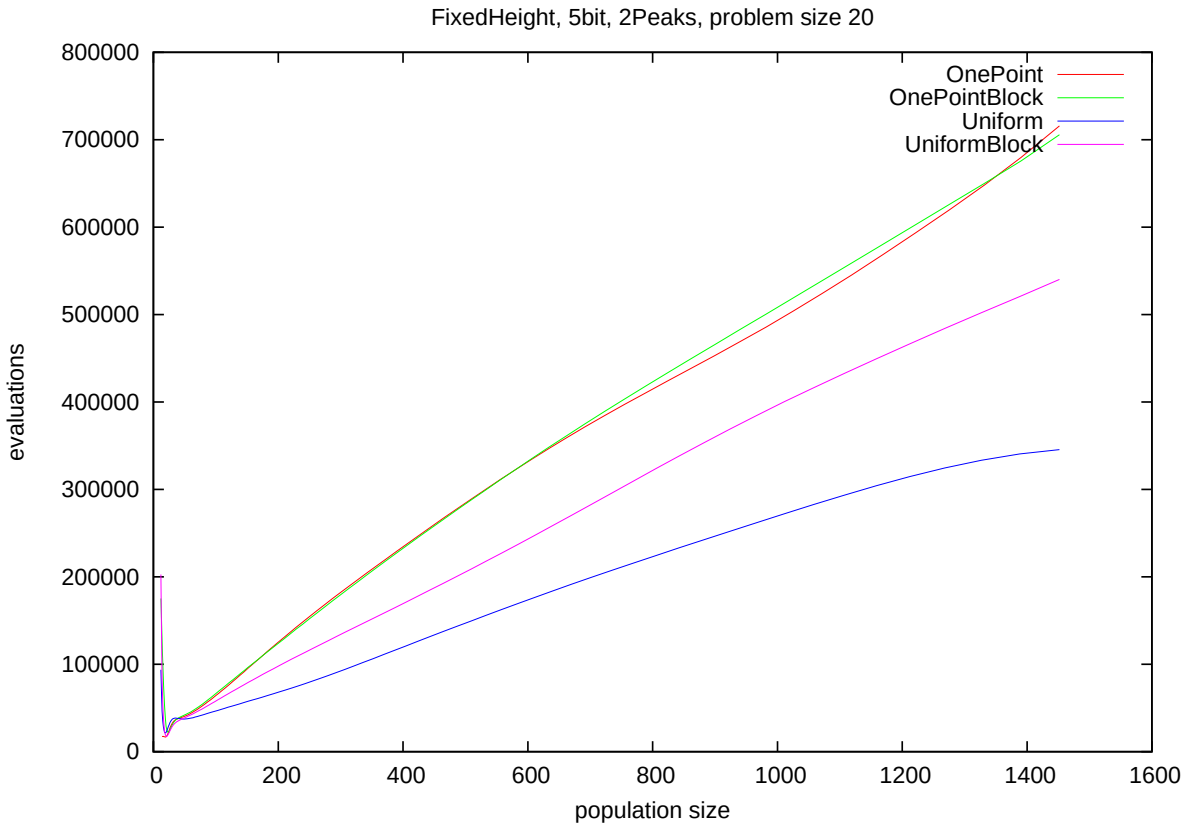
Obr. A.1:


FixedHeight, 5bit, 2Peaks, problem size 10
OnePoint
OnePointBlock
Uniform
UniformBlock
evaluations
population size
0
20000
40000
60000
80000
100000
120000
0
200
400
600
800
1000
1200
1400
1600

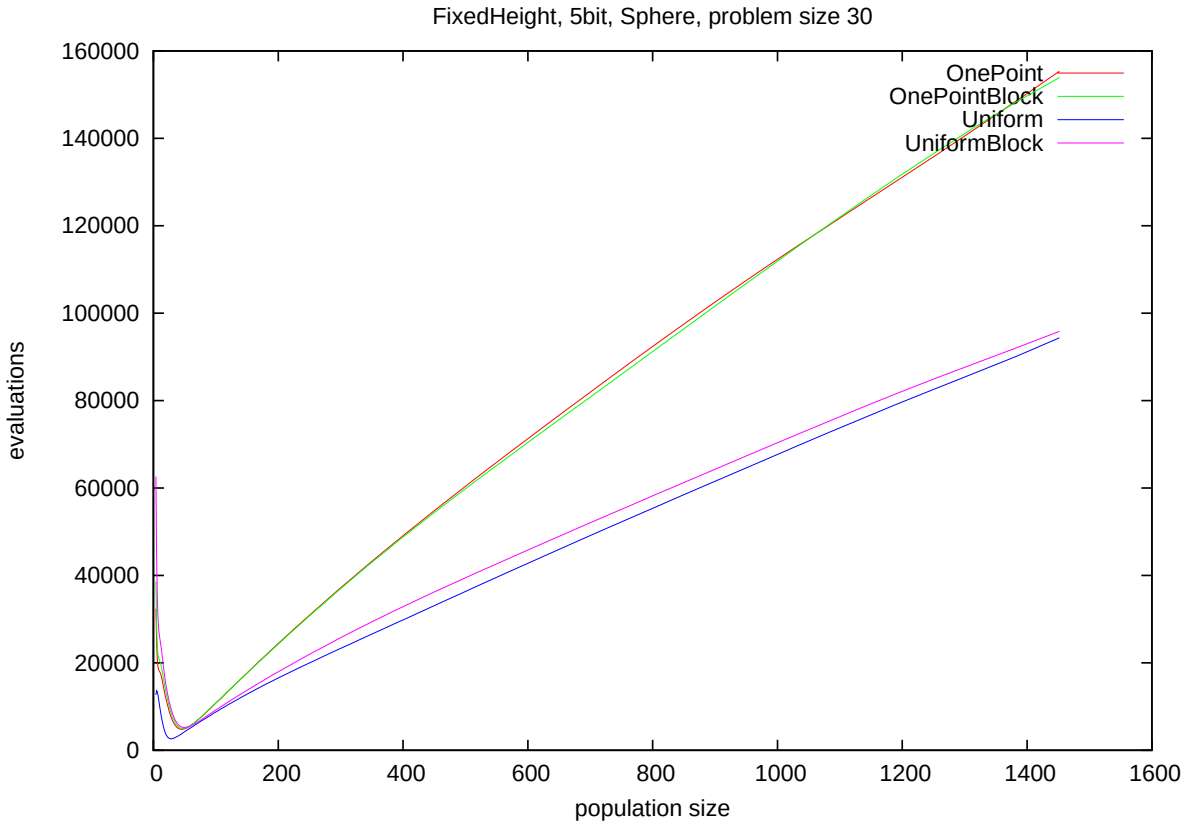
Obr. A.2:


FixedHeight, 5bit, Sphere, problem size 20
evaluations
population size
OnePoint
OnePointBlock
Uniform
UniformBlock
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
0
200
400
600
800
1000
1200
1400
1600

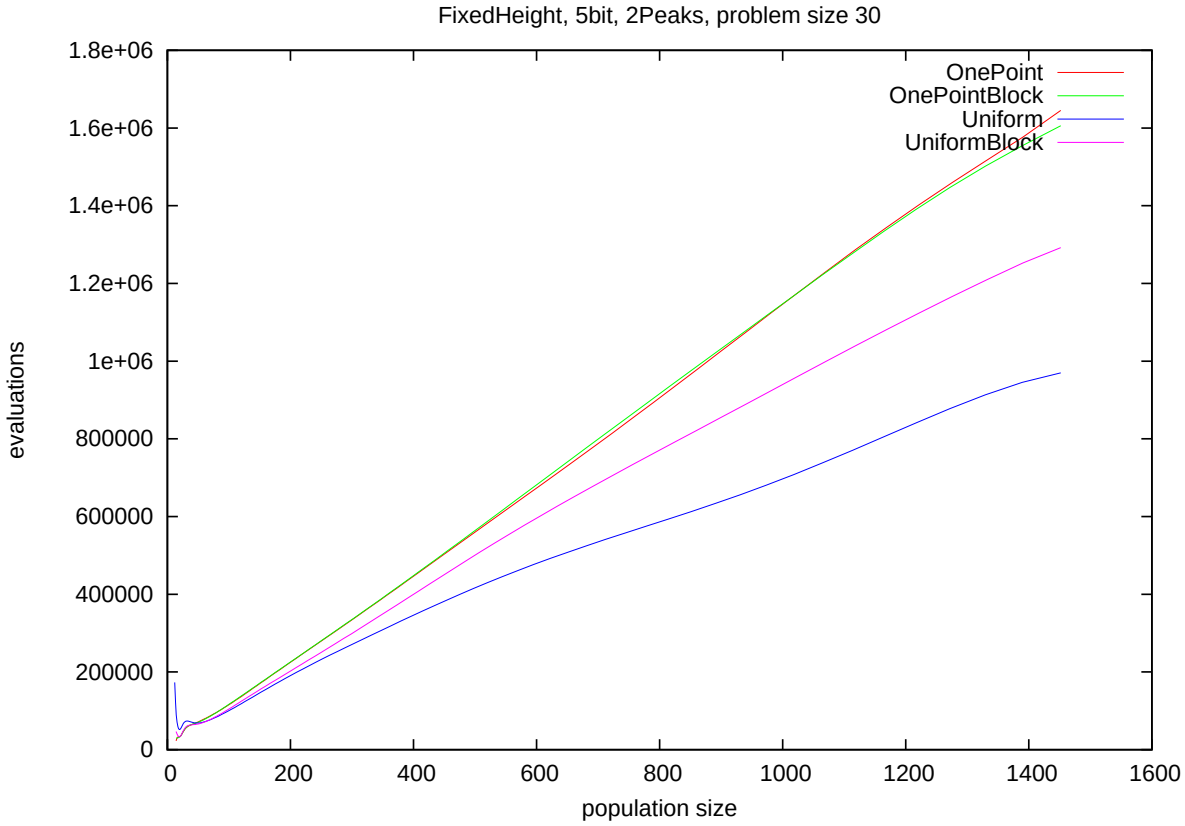
Obr. A.3:


FixedHeight, 5bit, 2Peaks, problem size 20
evaluations
population size
OnePoint
OnePointBlock
Uniform
UniformBlock
0
100000
200000
300000
400000
500000
600000
700000
800000
0
200
400
600
800
1000
1200
1400
1600

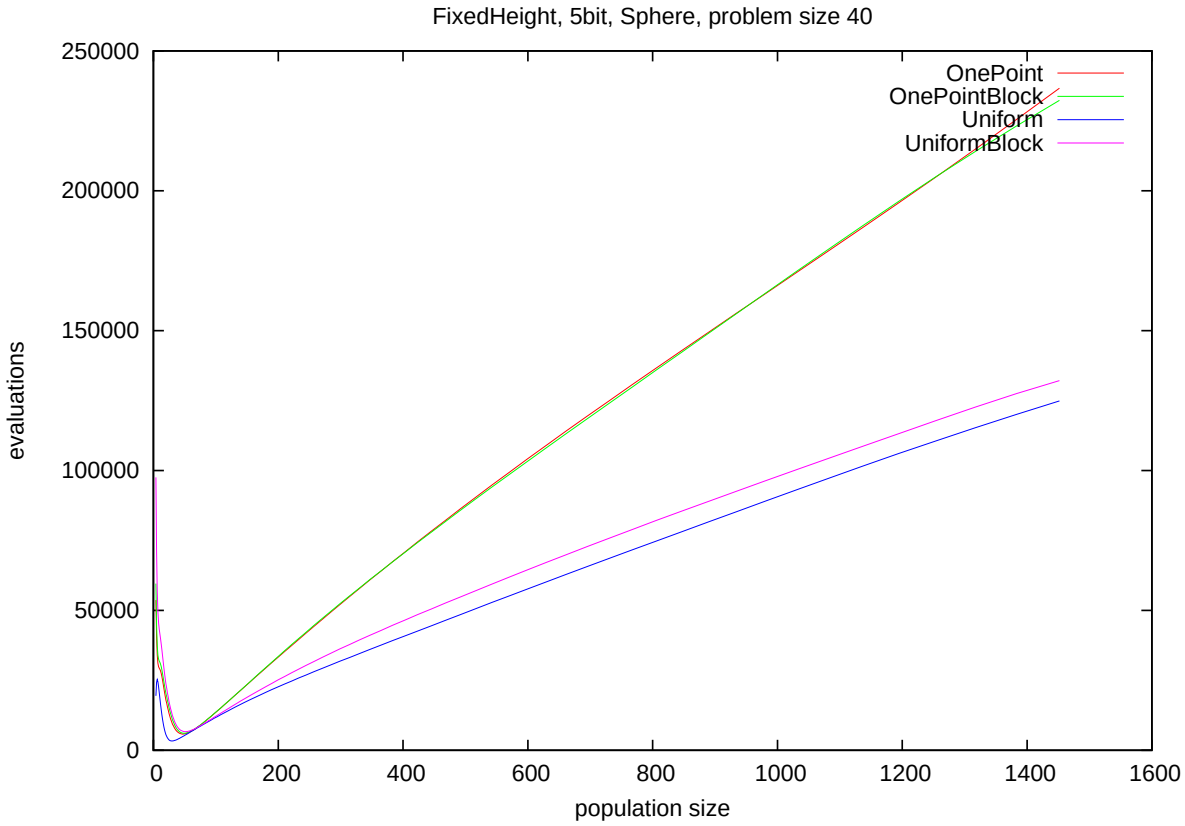
Obr. A.4:


FixedHeight, 5bit, Sphere, problem size 30
evaluations
population size
OnePoint
OnePointBlock
Uniform
UniformBlock
0
20000
40000
60000
80000
100000
120000
140000
160000
0
200
400
600
800
1000
1200
1400
1600

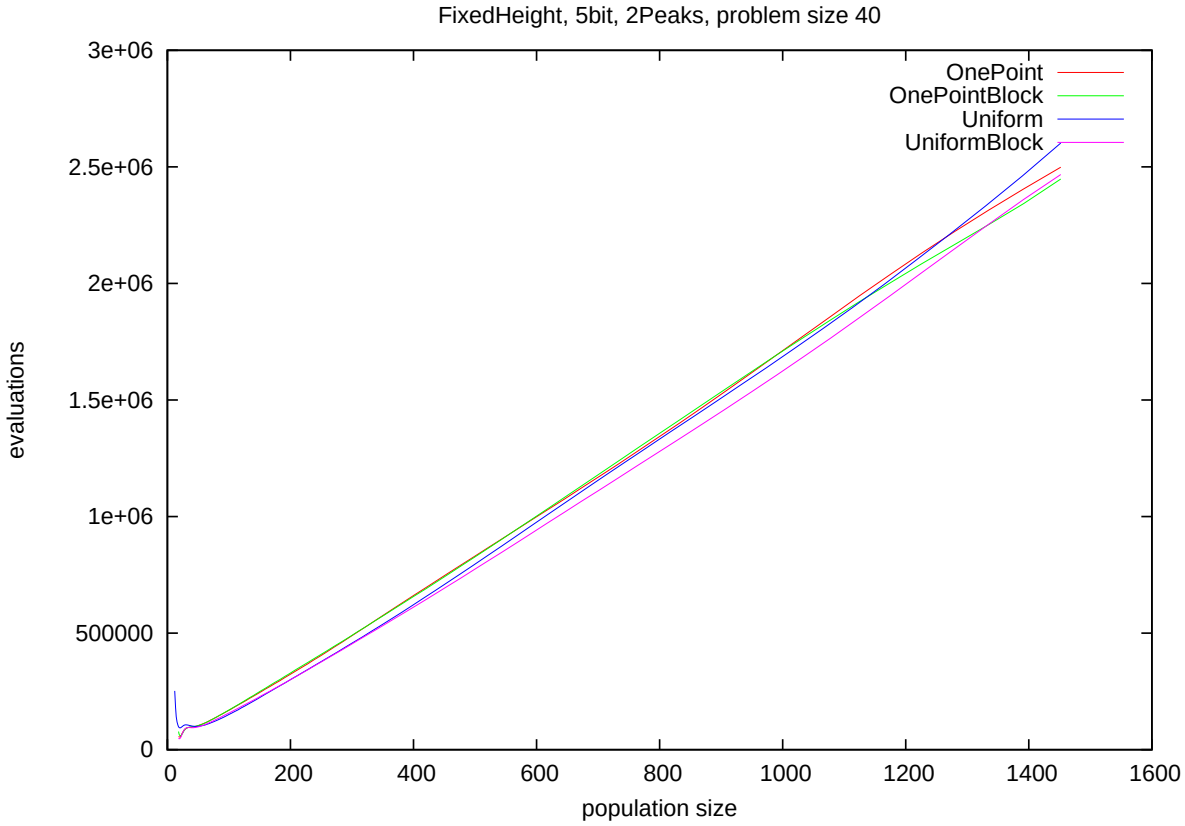
Obr. A.5:


FixedHeight, 5bit, 2Peaks, problem size 30
evaluations
population size
OnePoint
OnePointBlock
Uniform
UniformBlock
0
200000
400000
600000
800000
1e+06
1.2e+06
1.4e+06
1.6e+06
1.8e+06
0
200
400
600
800
1000
1200
1400
1600

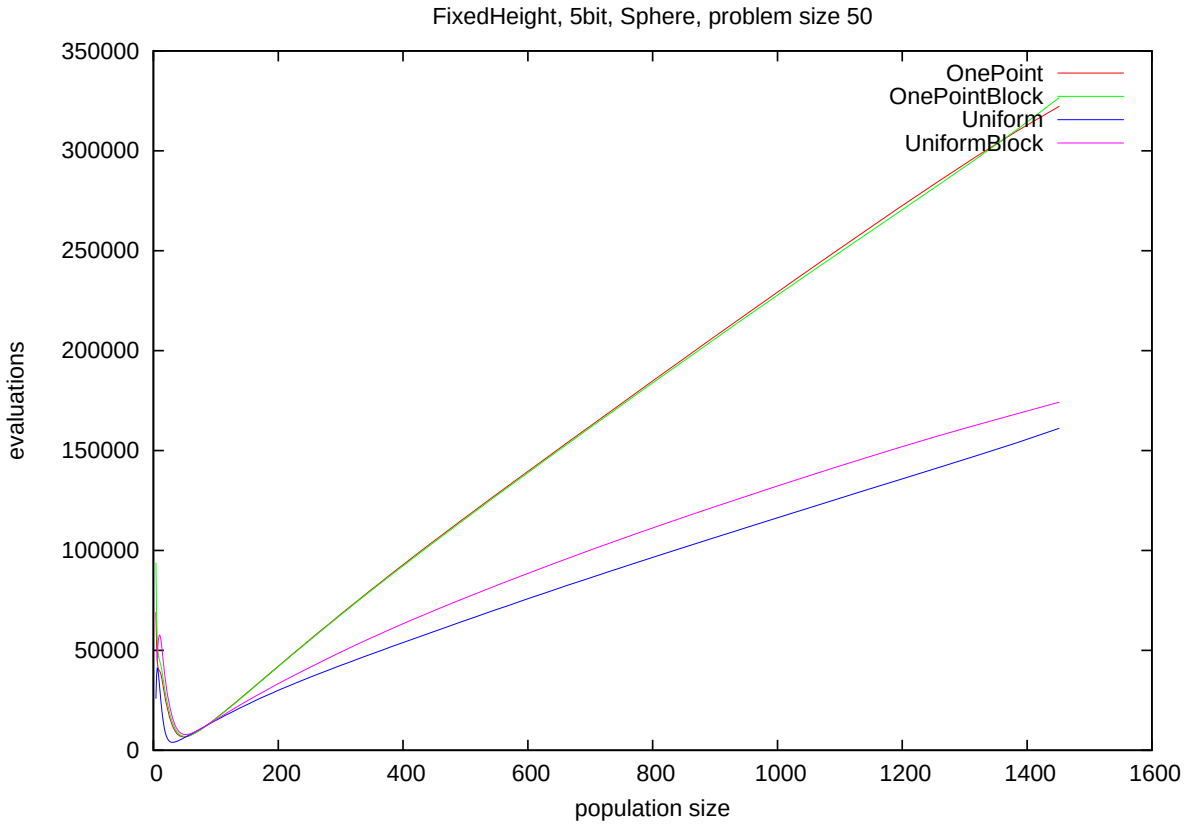
Obr. A.6:


FixedHeight, 5bit, Sphere, problem size 40
OnePoint
OnePointBlock
Uniform
UniformBlock
evaluations
population size
250000
200000
150000
100000
50000
0
0
200
400
600
800
1000
1200
1400
1600

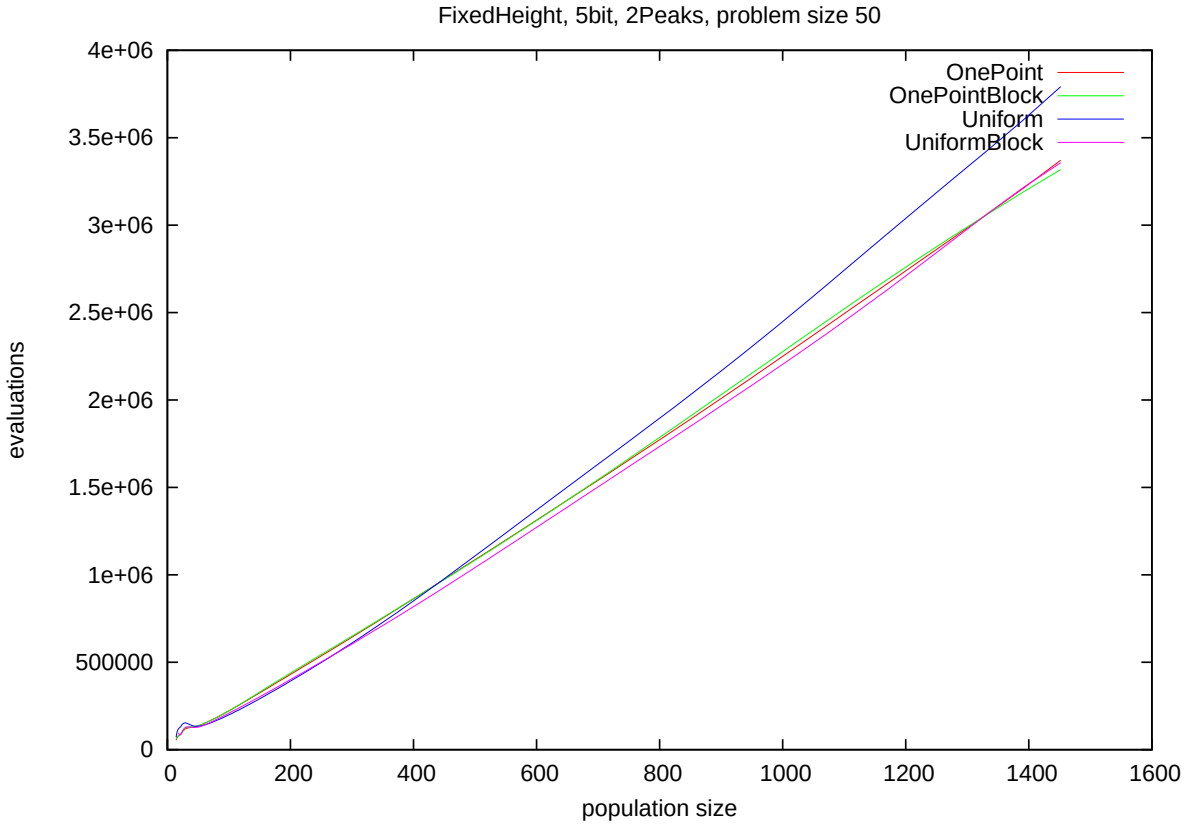
Obr. A.7:


FixedHeight, 5bit, 2Peaks, problem size 40
OnePoint
OnePointBlock
Uniform
UniformBlock
evaluations
population size
3e+06
2.5e+06
2e+06
1.5e+06
1e+06
500000
0
0
200
400
600
800
1000
1200
1400
1600

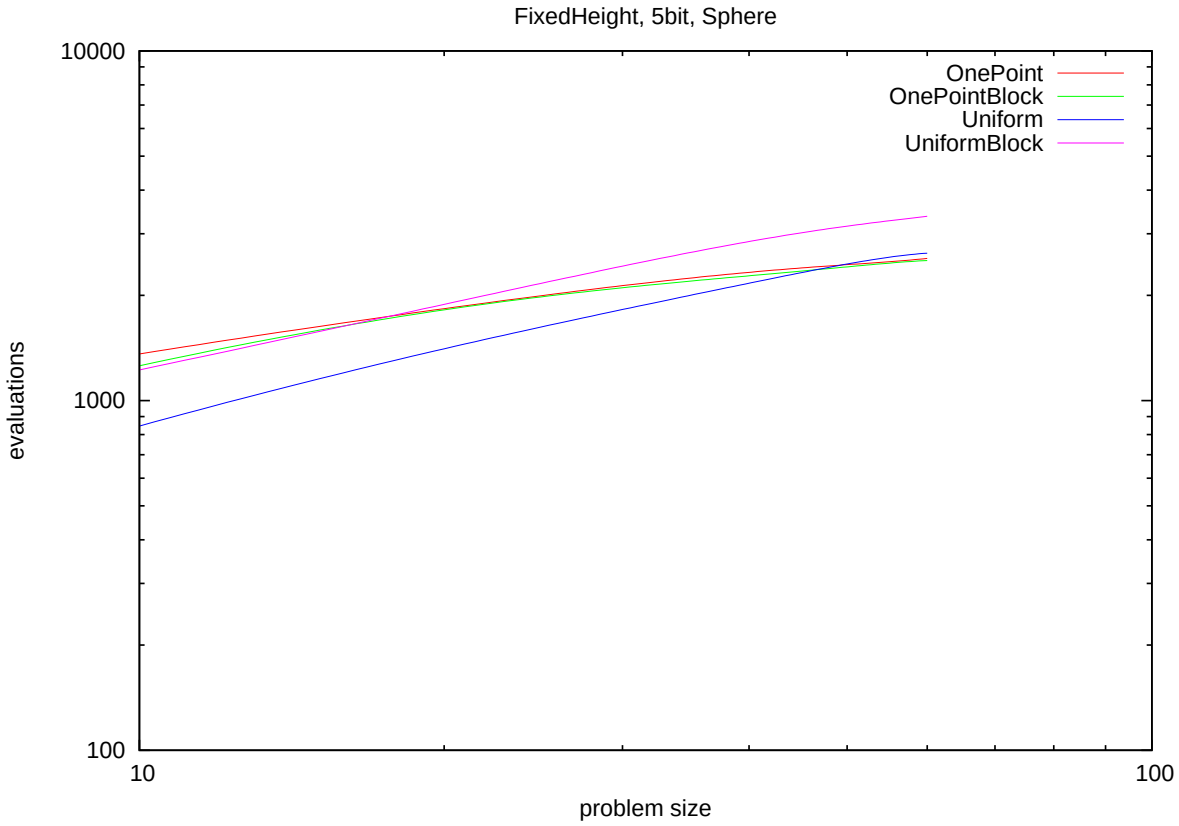
Obr. A.8:



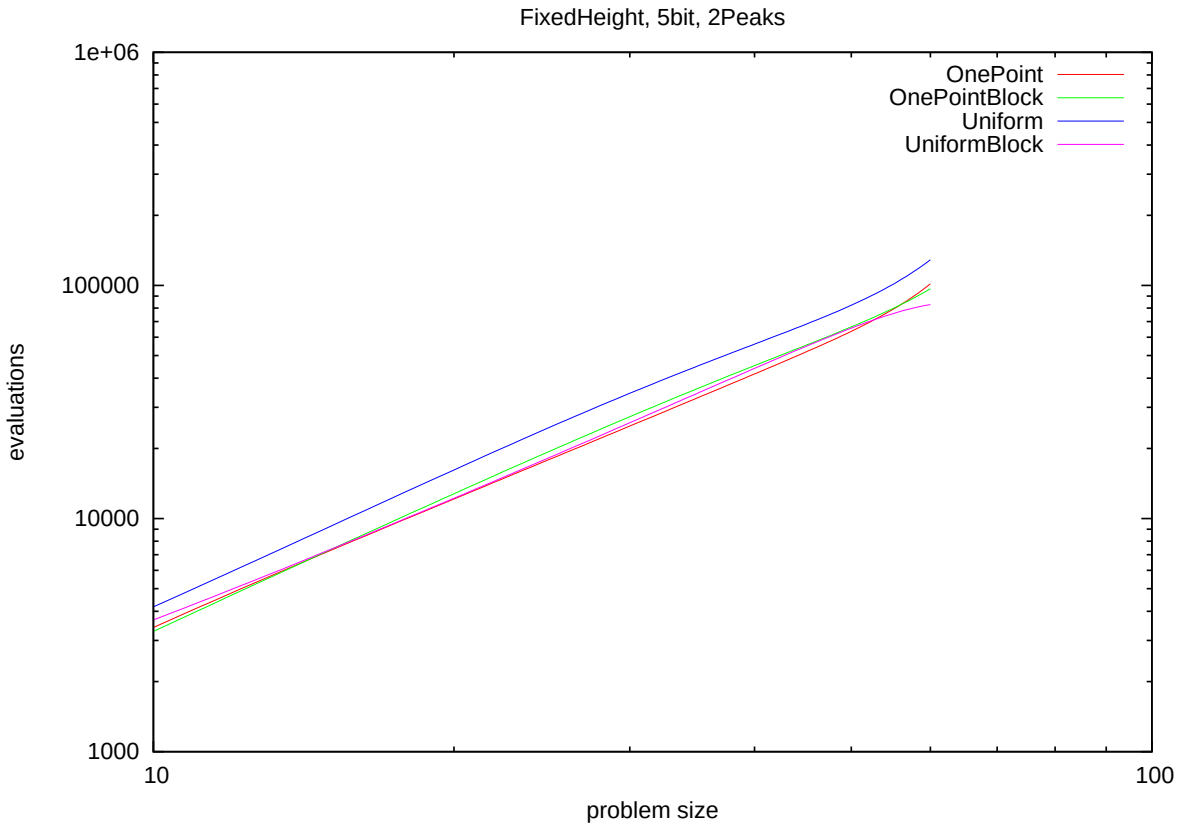
Obr. A.9:



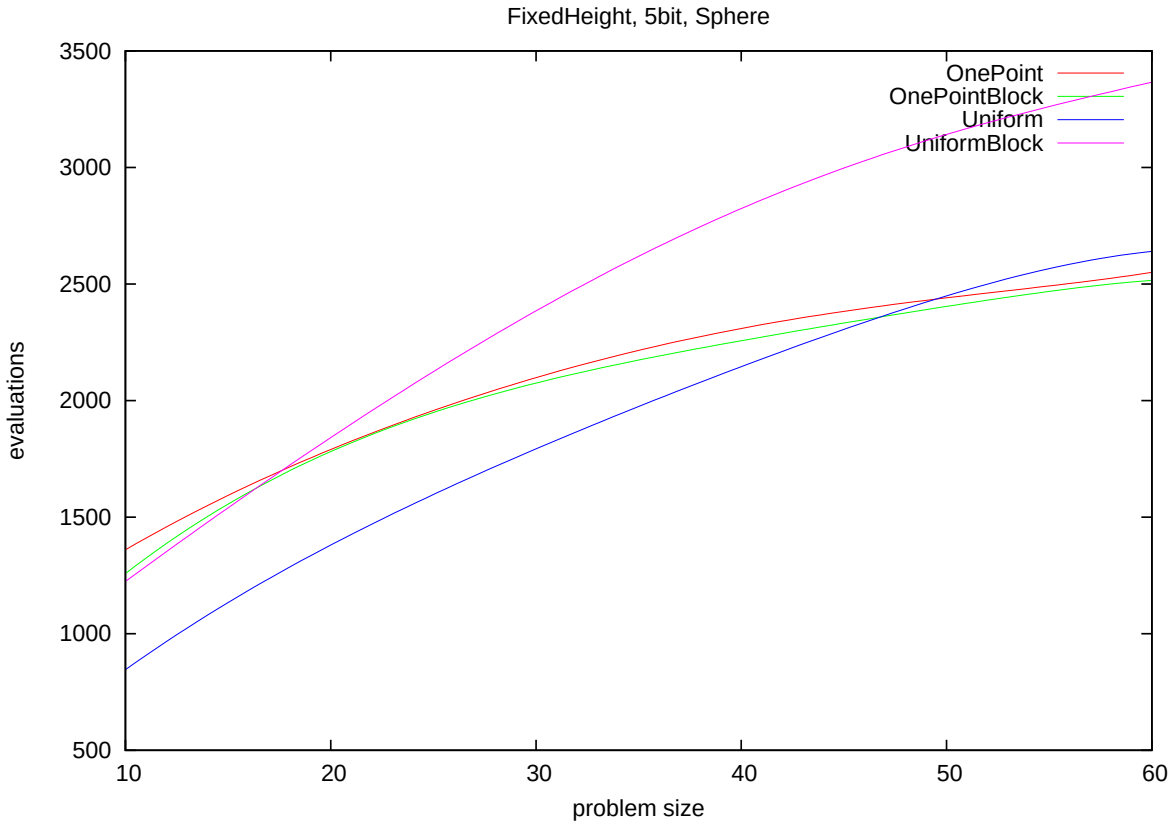
Obr. A.10:



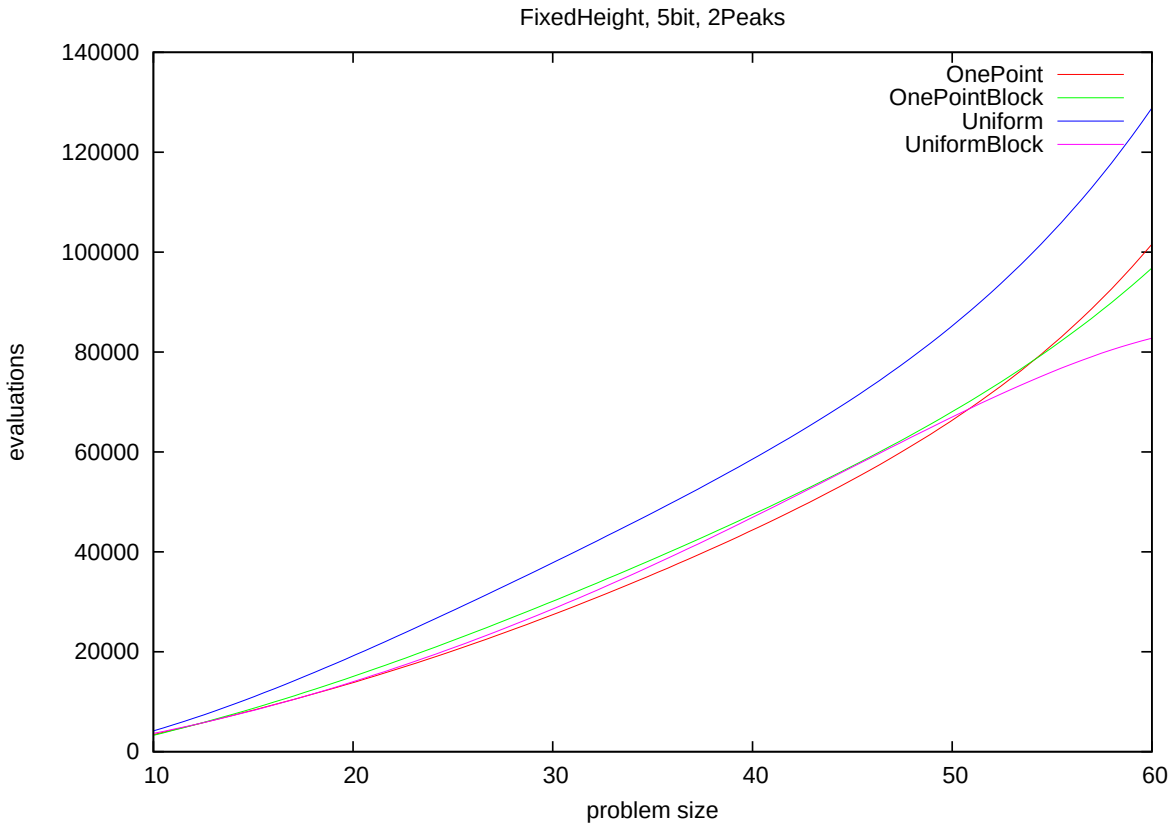
Obr. A.11:



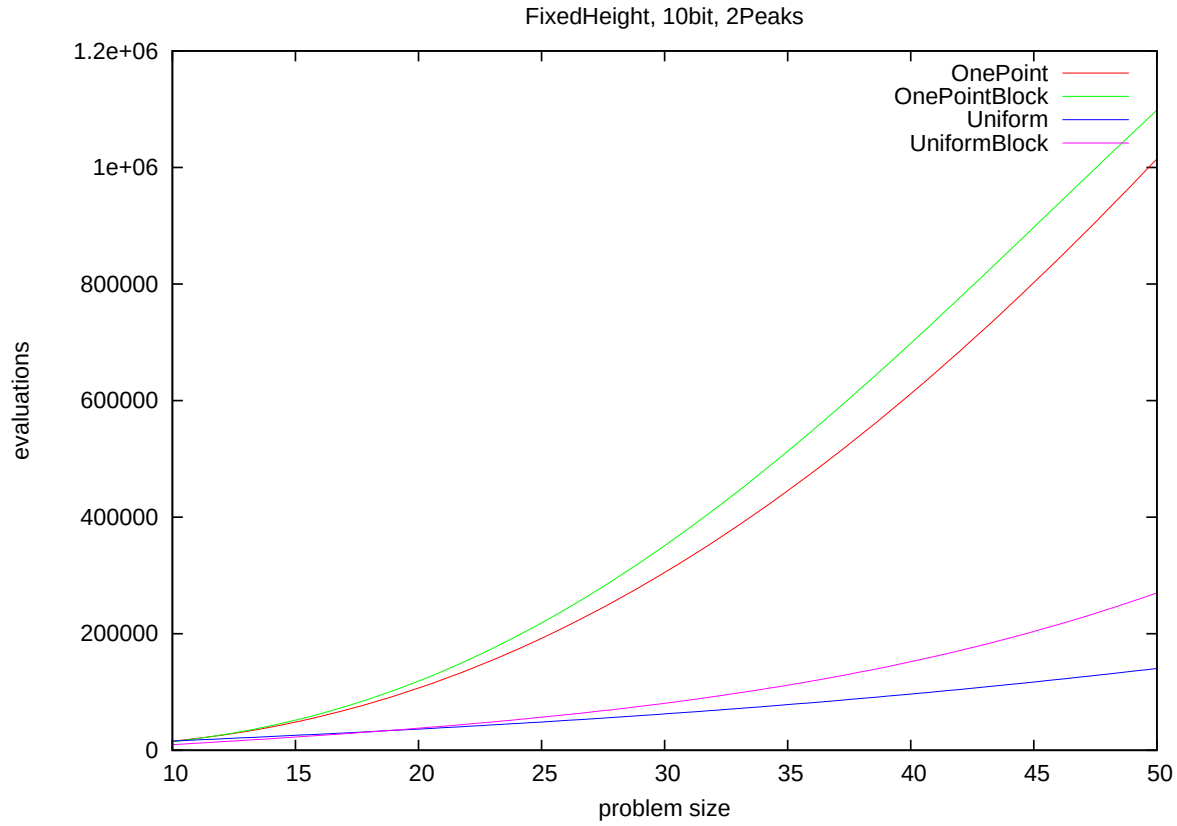
Obr. A.12:



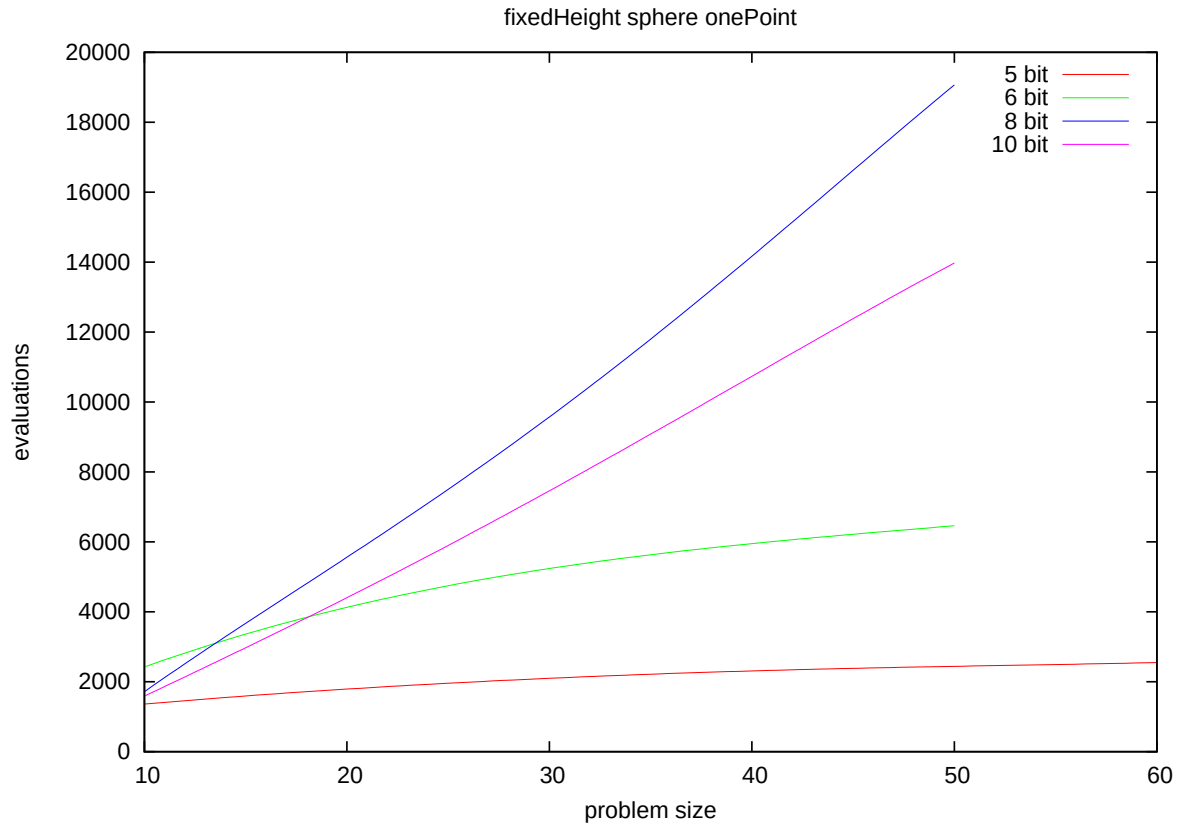
Obr. A.13:



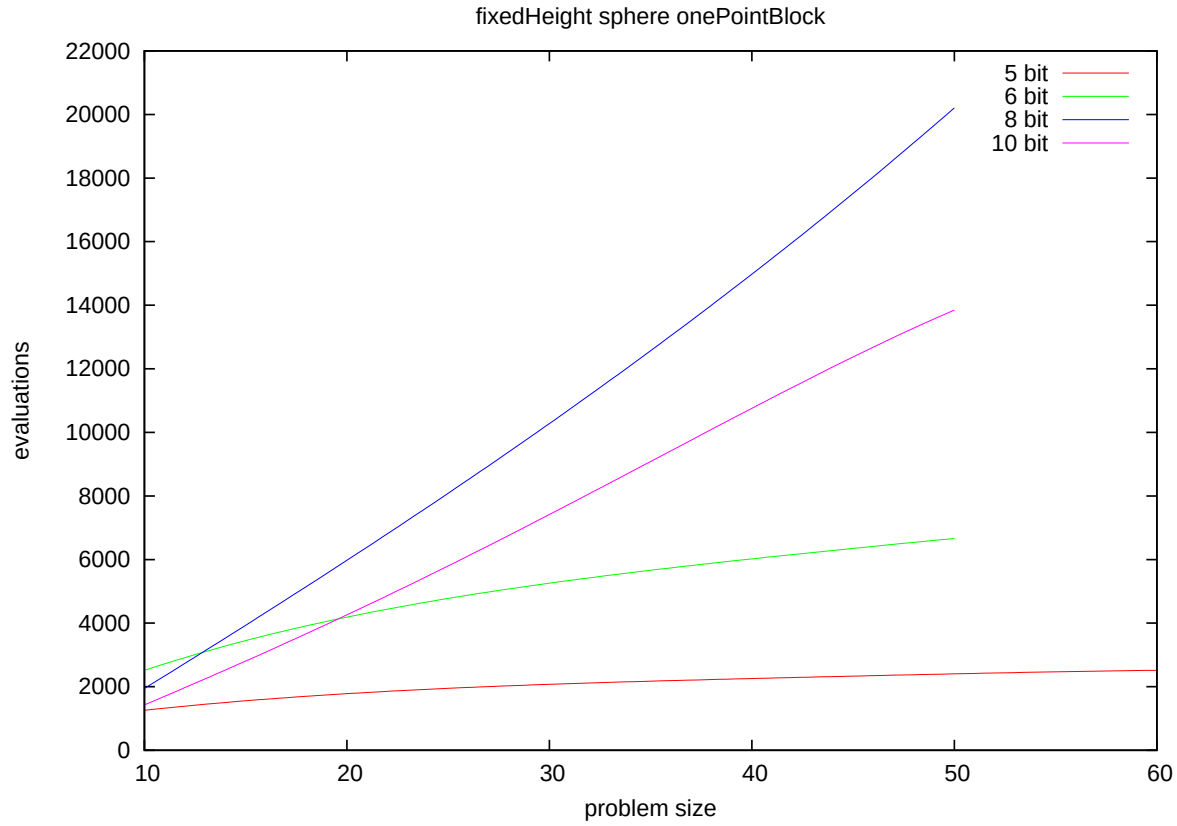
Obr. A.14:



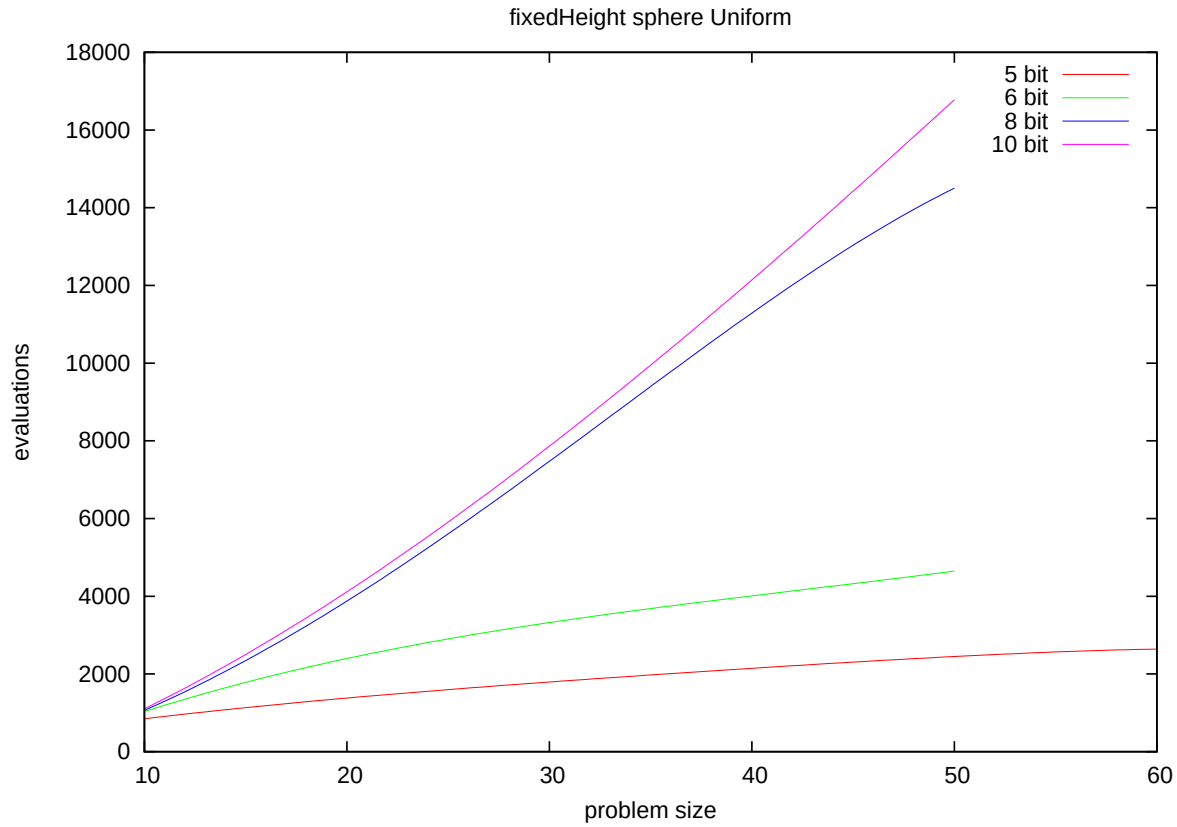
Obr. A.15:



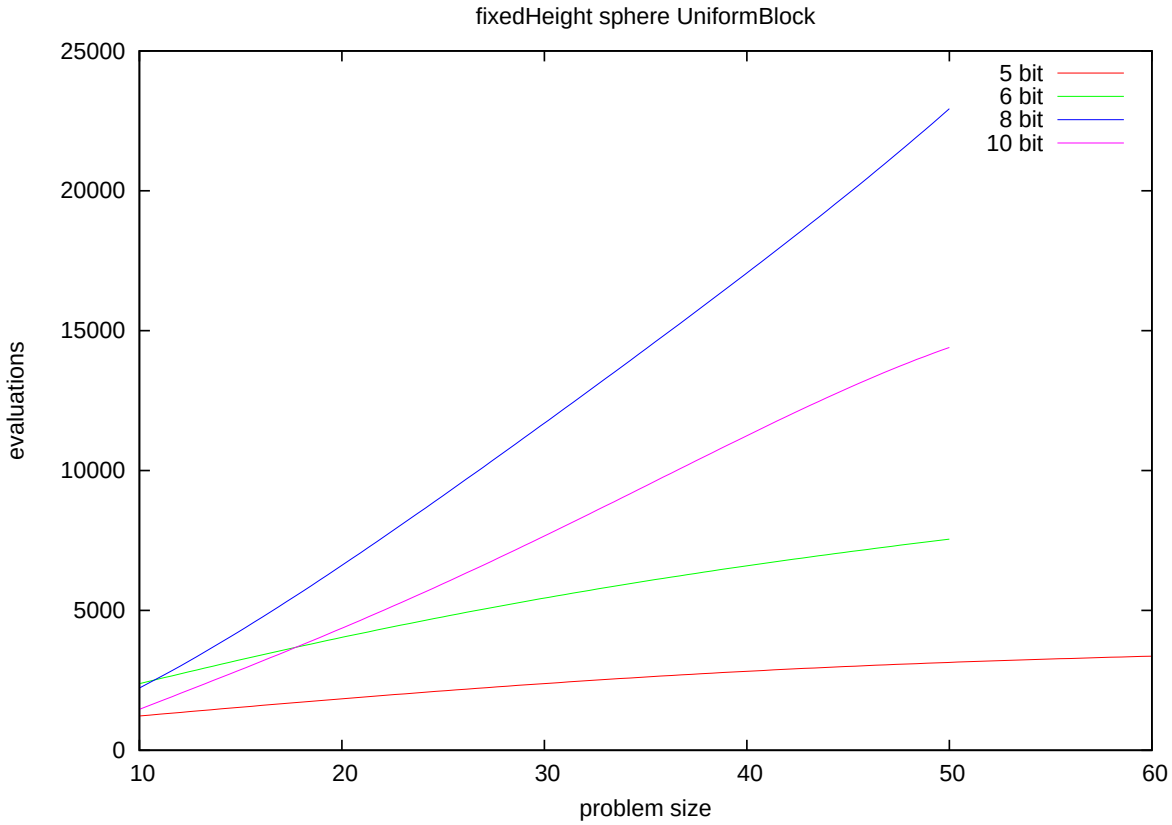
Obr. A.16:



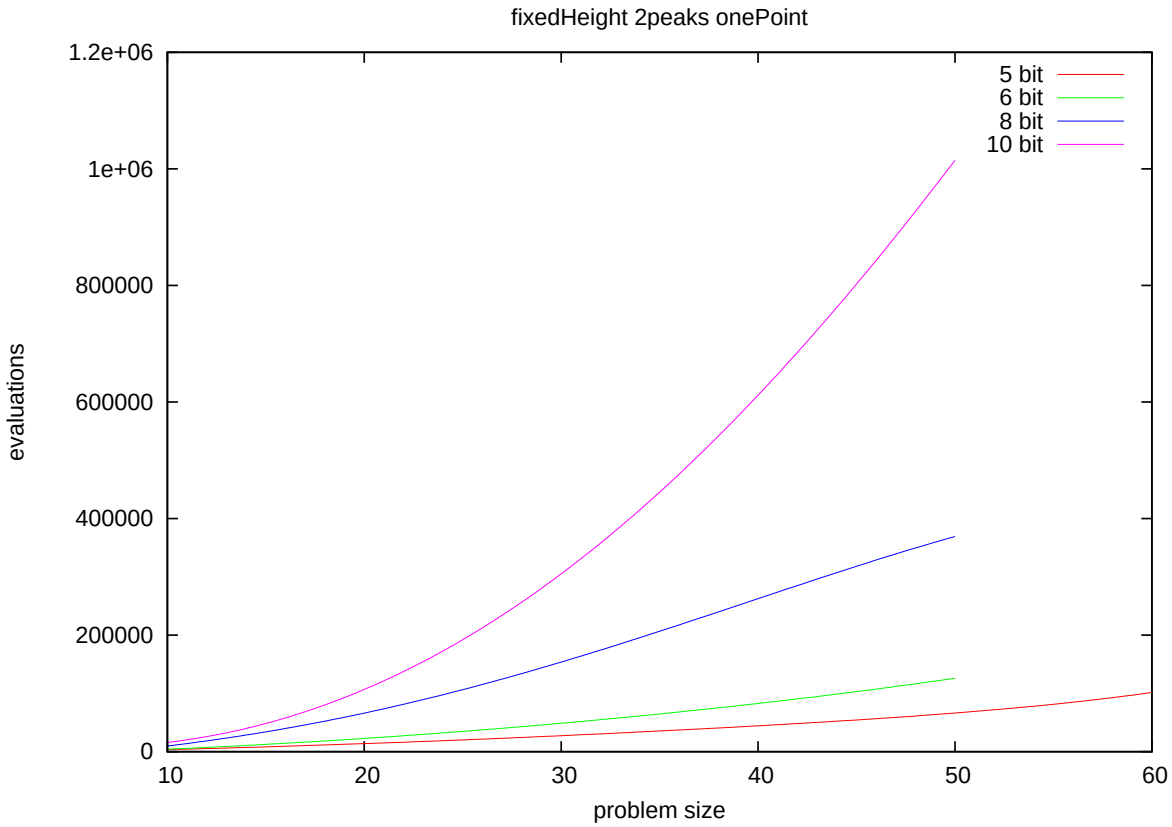
Obr. A.17:



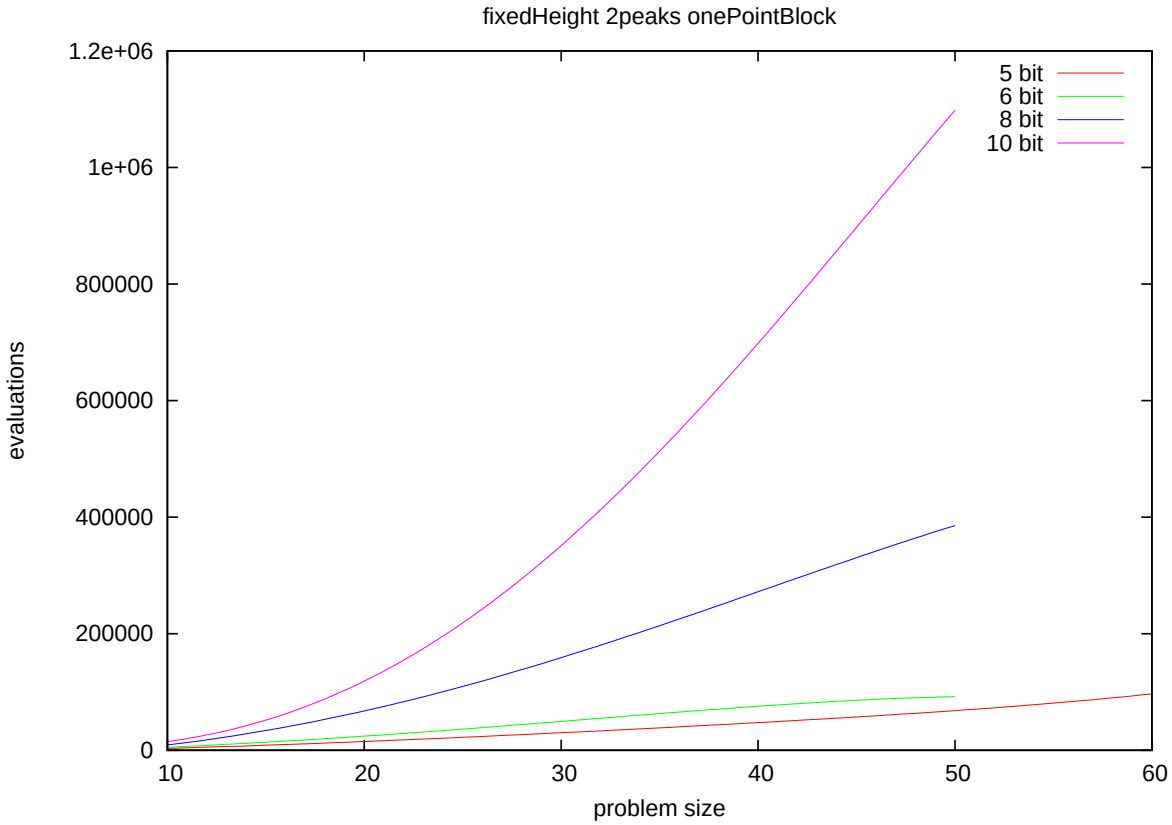
Obr. A.18:



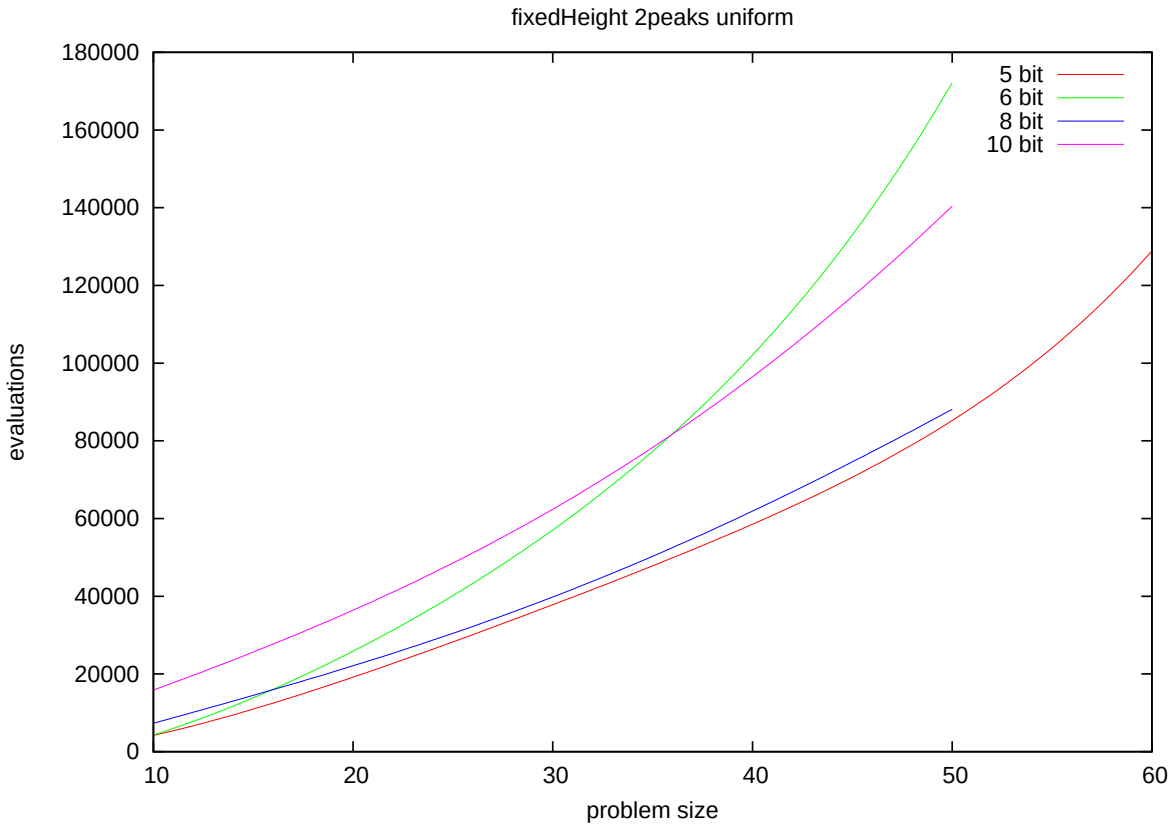
Obr. A.19:



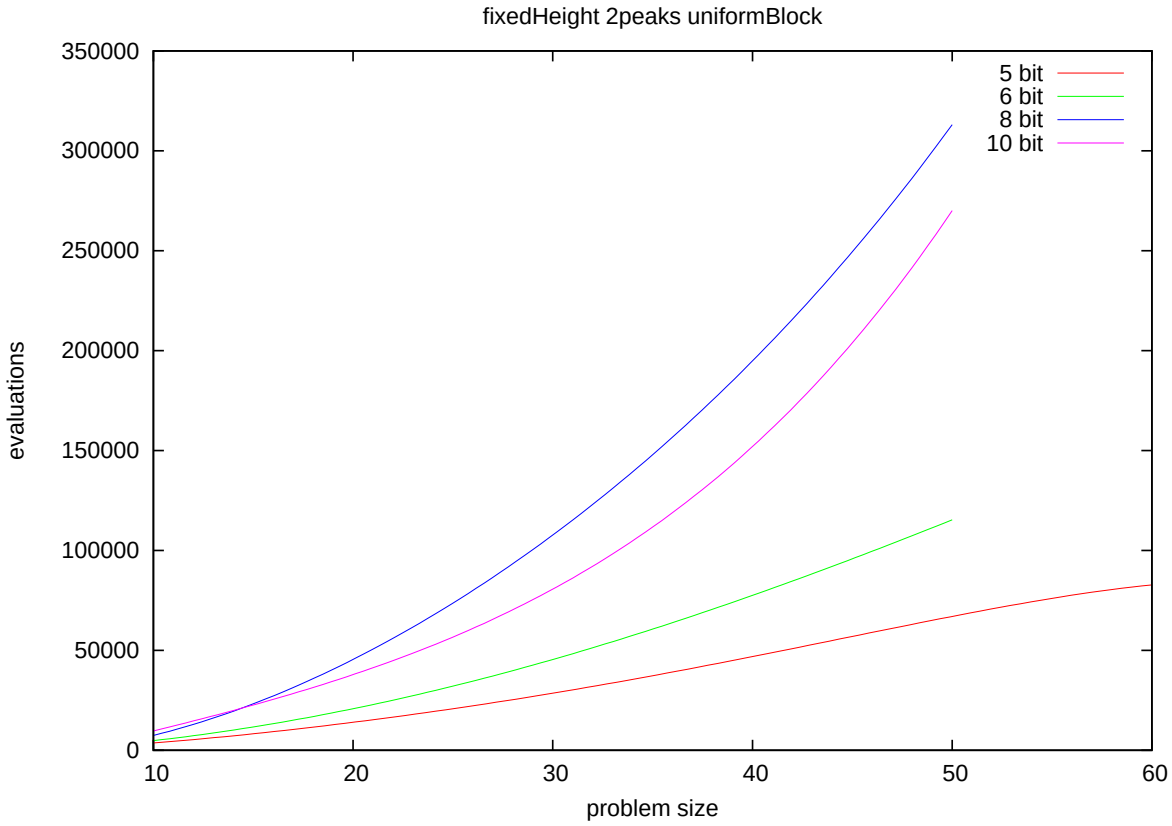
Obr. A.20:



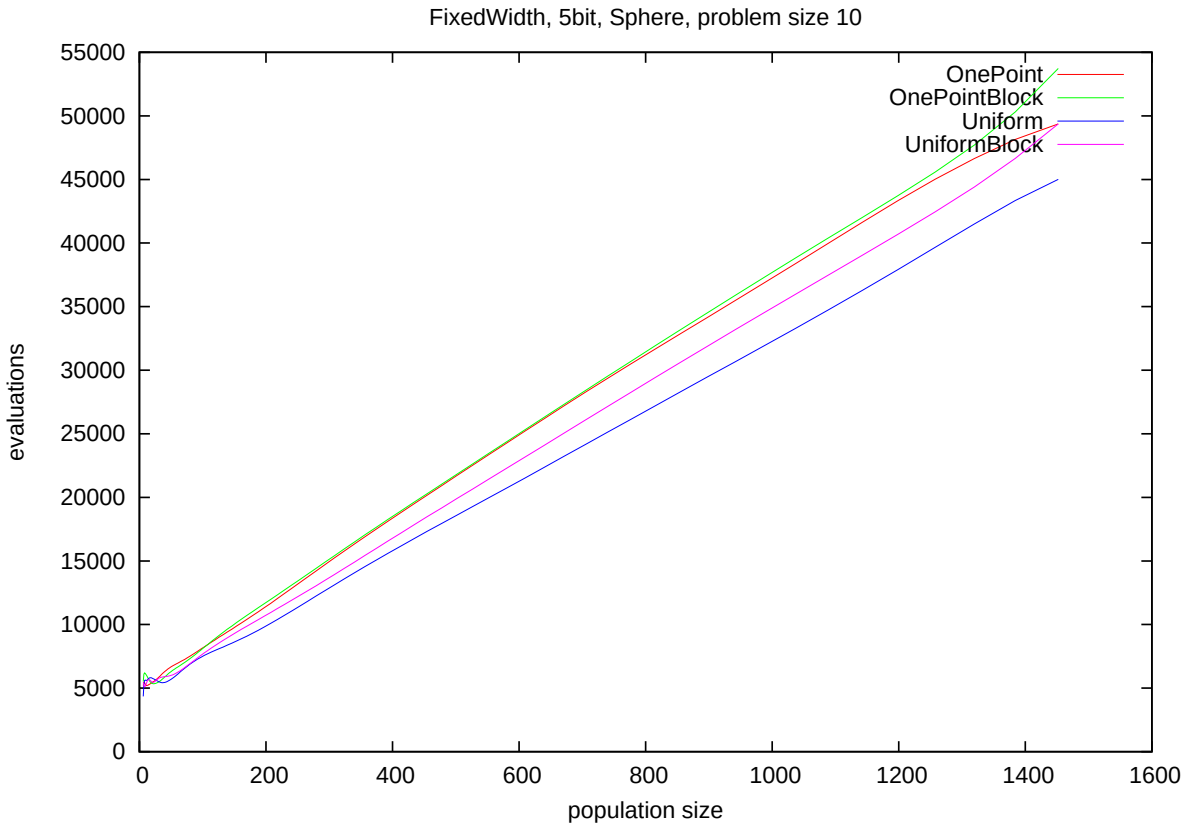
Obr. A.21:



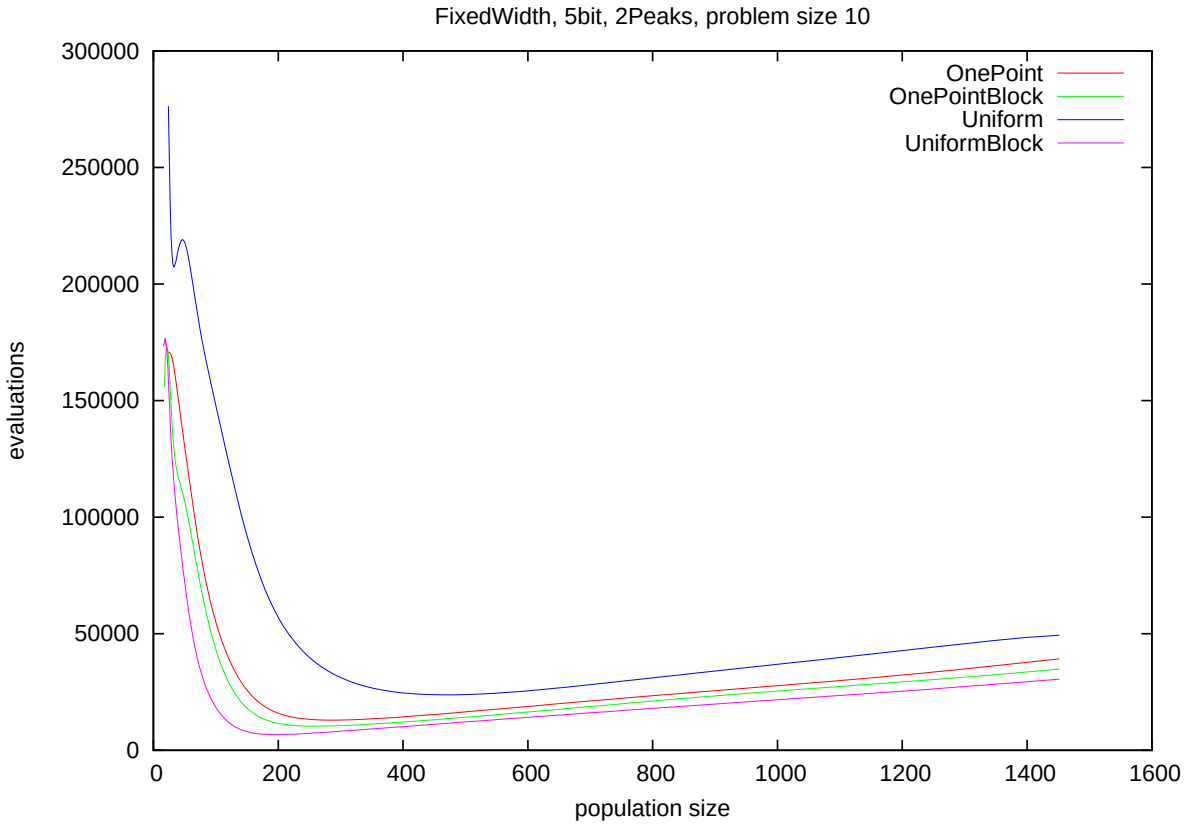
Obr. A.22:



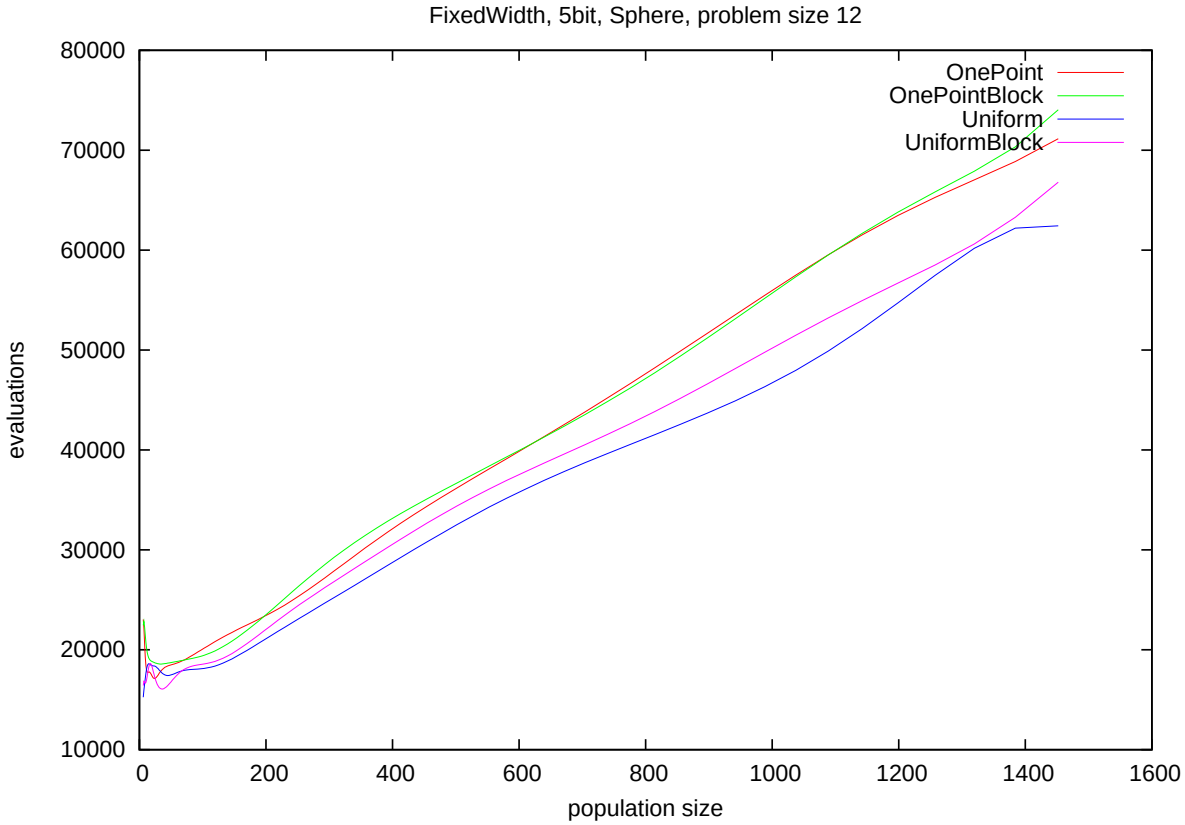
Obr. A.23:



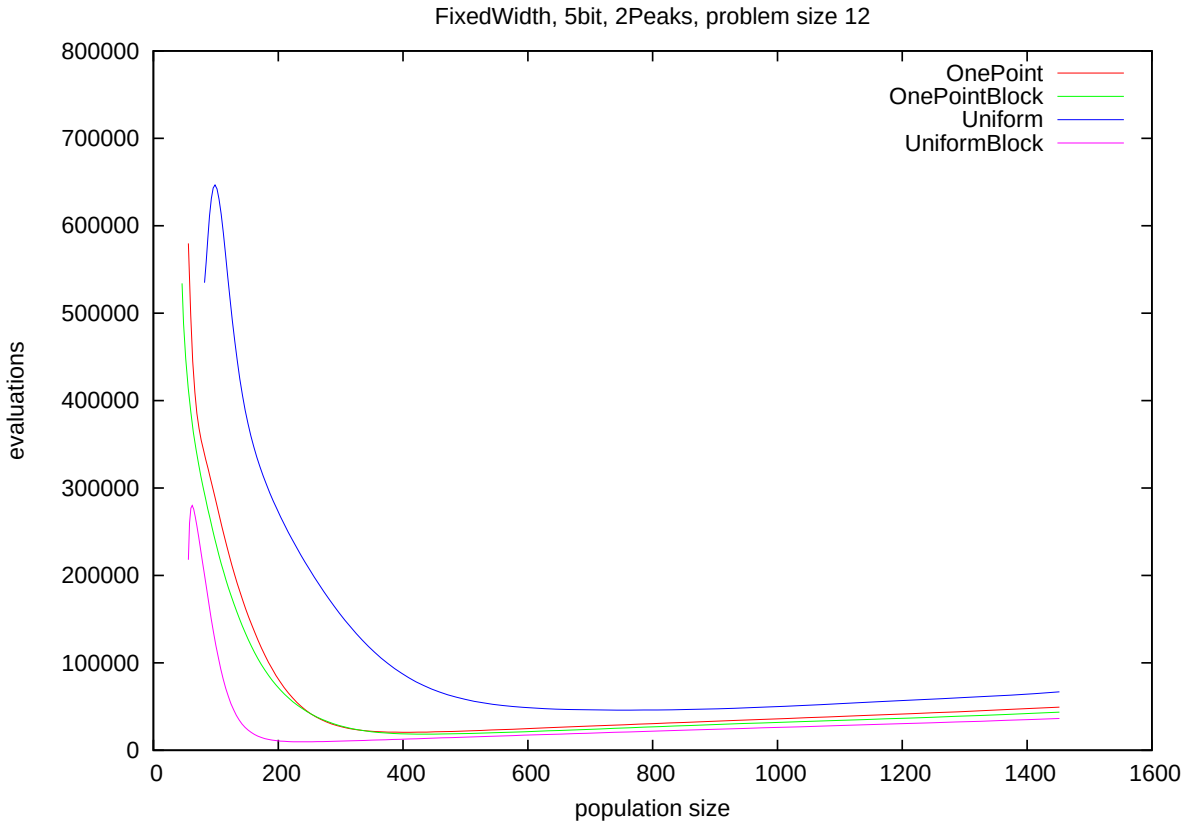
Obr. A.24:



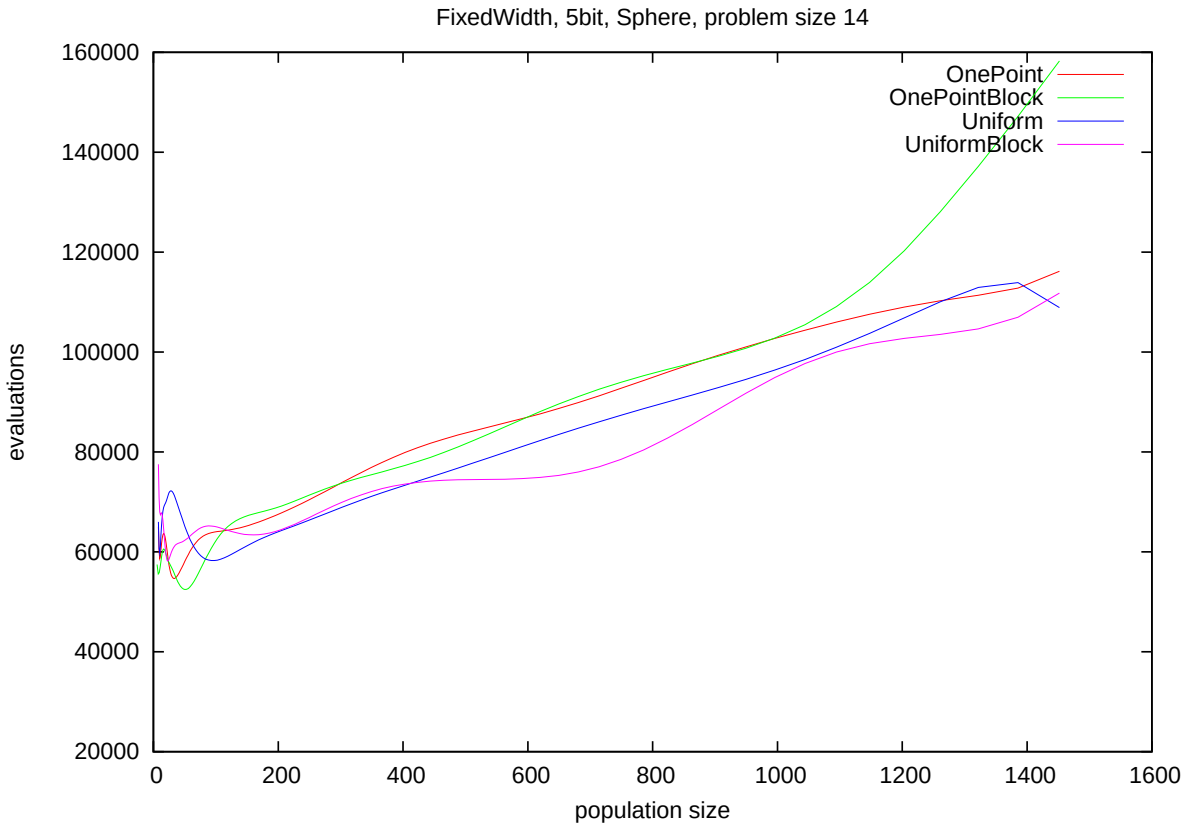
Obr. A.25:



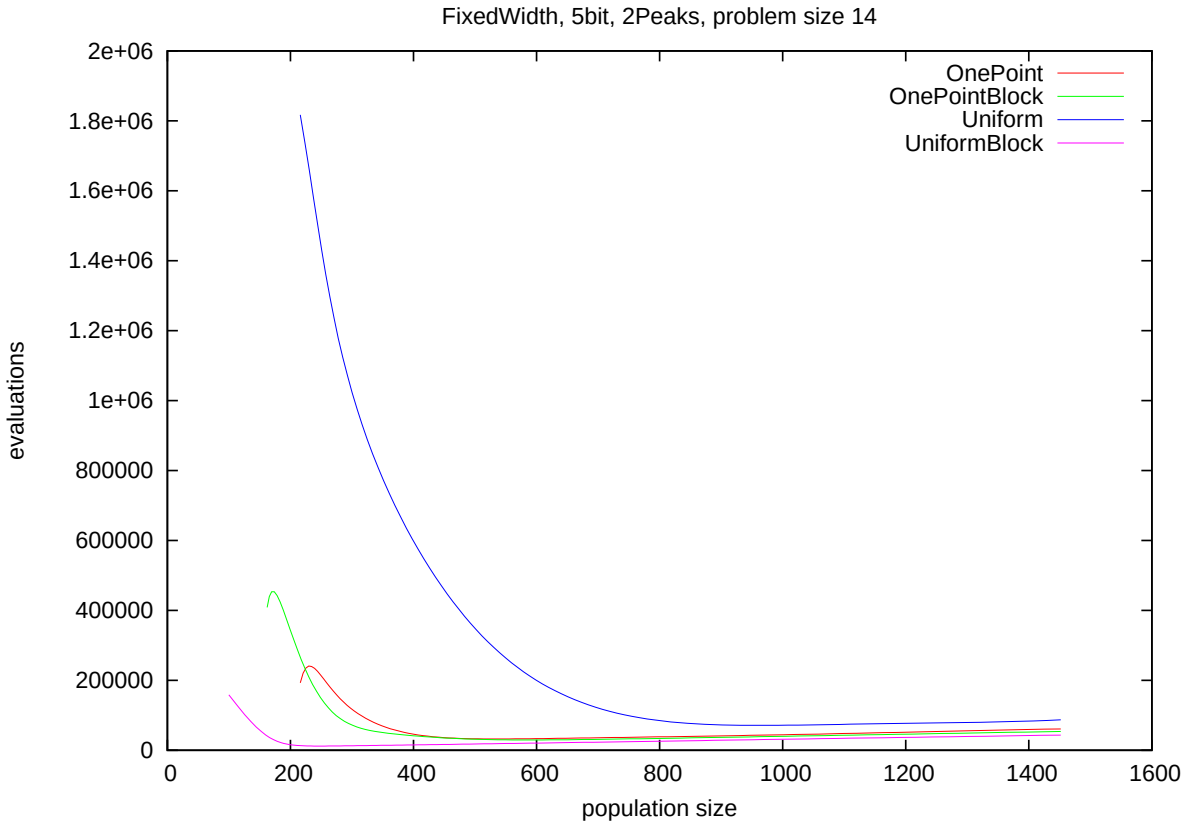
Obr. A.26:



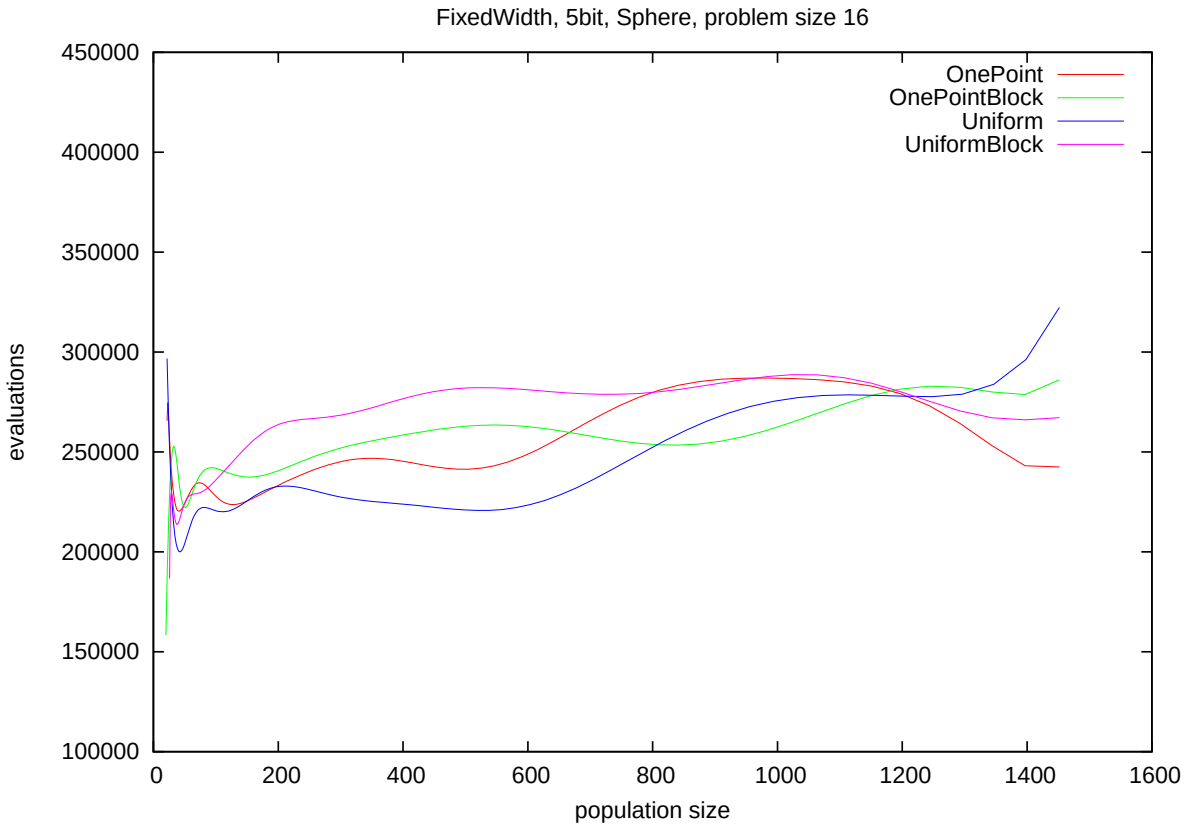
Obr. A.27:



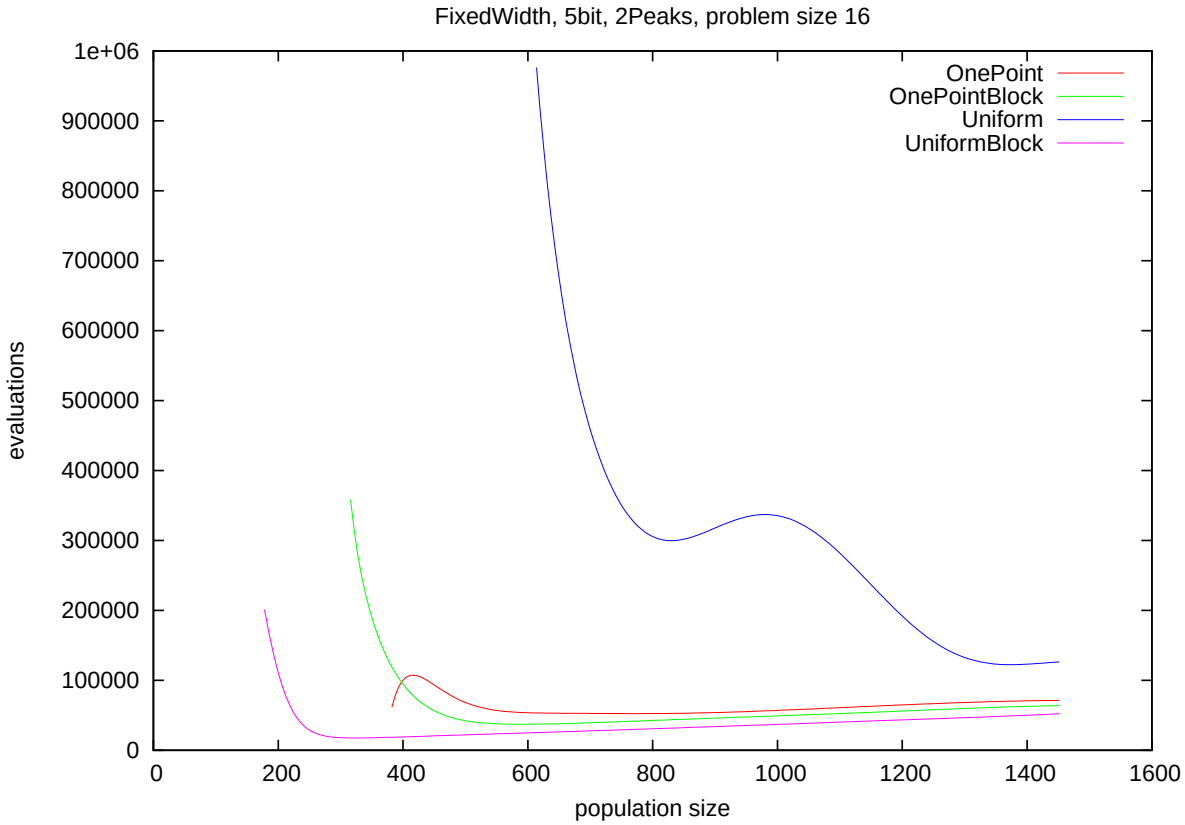
Obr. A.28:



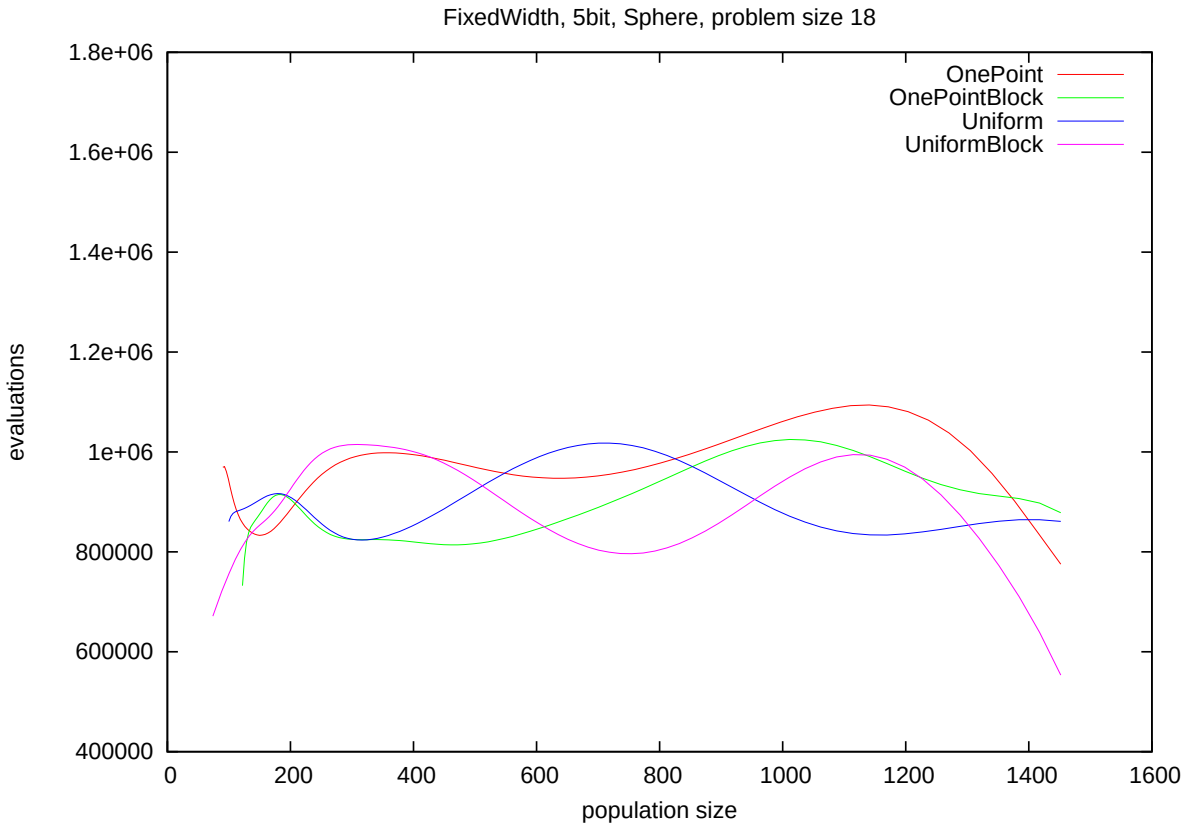
Obr. A.29:



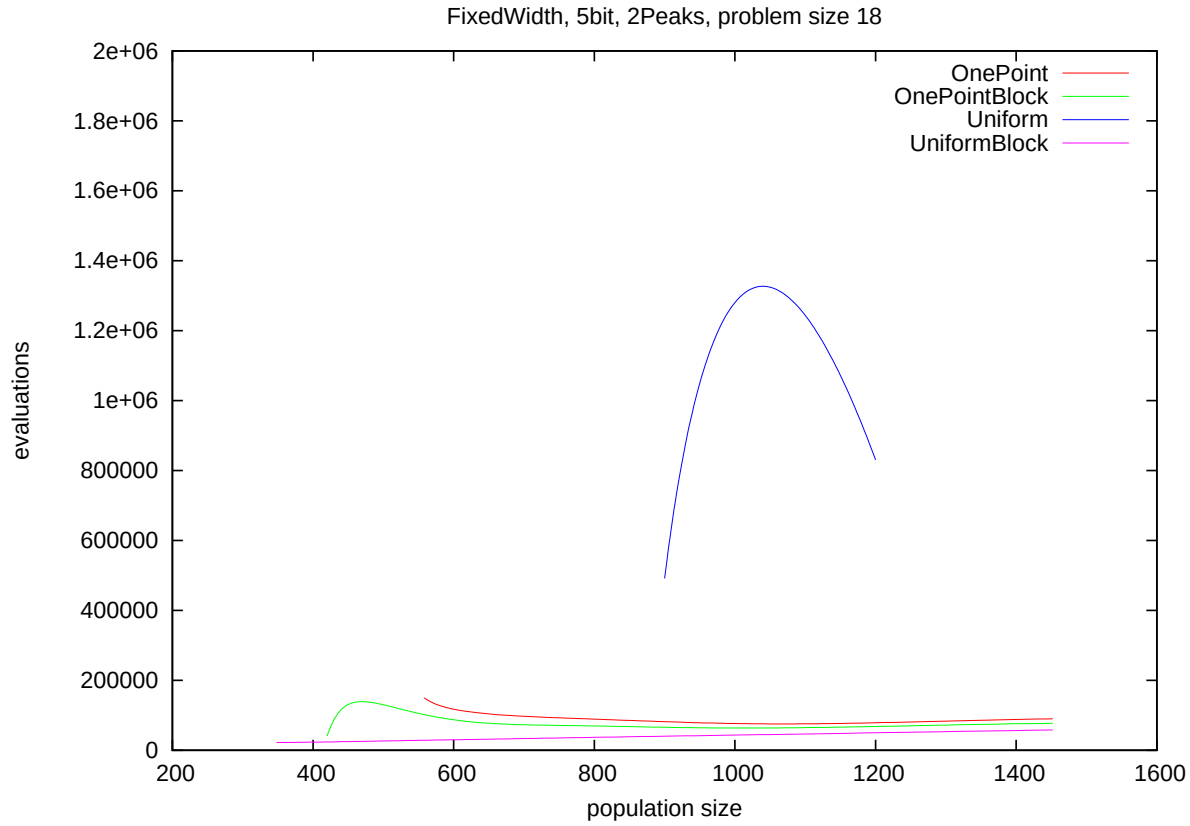
Obr. A.30:



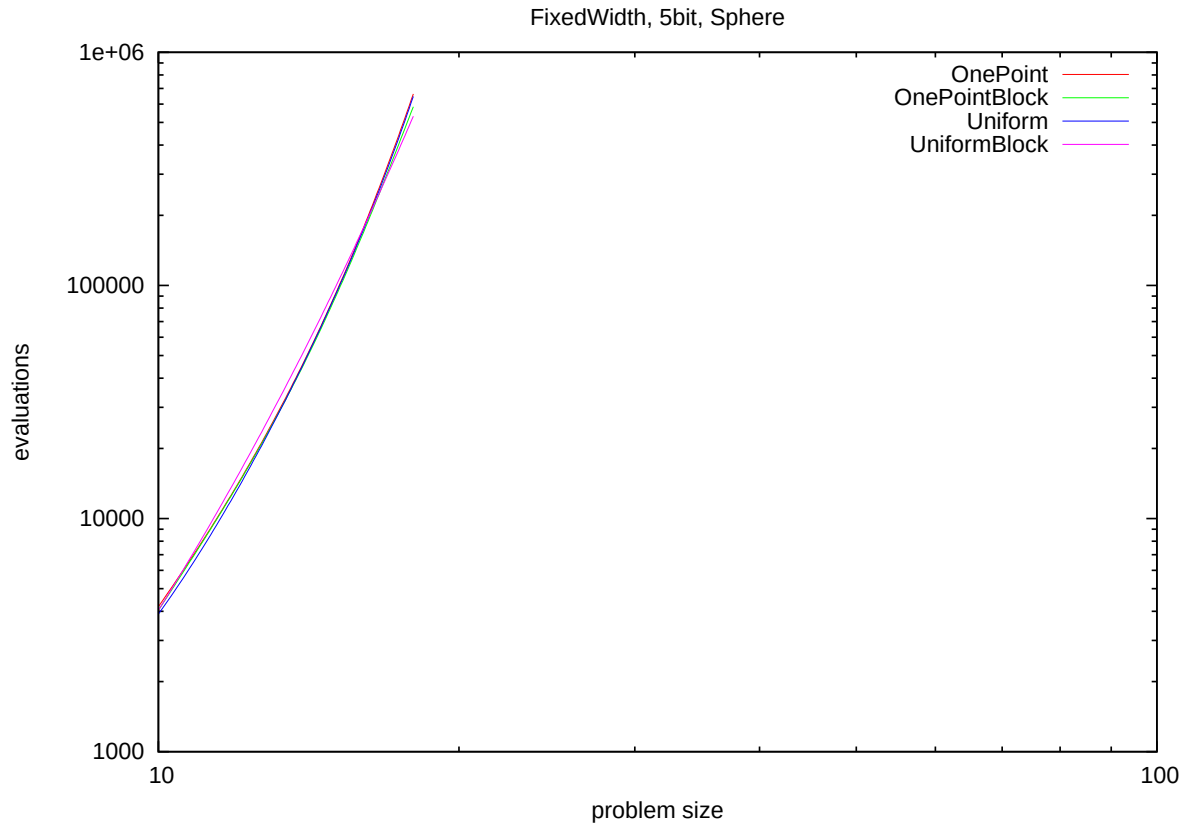
Obr. A.31:



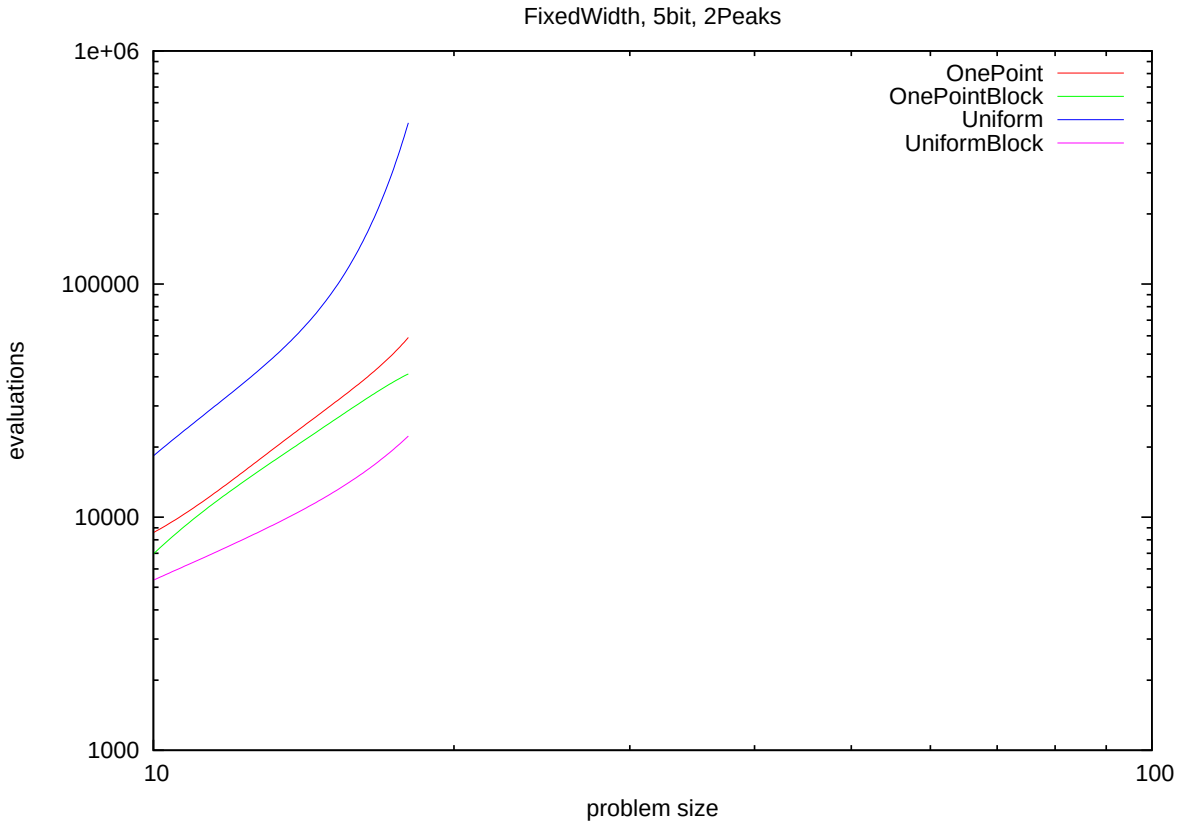
Obr. A.32:



Obr. A.33:



Obr. A.34:


FixedWidth, 5bit, 2Peaks
1e+06
100000
10000
1000
evaluations
10
100
problem size
OnePoint
OnePointBlock
Uniform
UniformBlock

Obr. A.35:

Literatúra

[CP] Erick Cantú-Paz. Supervised and unsupervised discretization methods for evolutionary algorithms. http://www.evolutionaria.com/publications/histo.pdf.

[Gol99] David E. Goldberg. Technical writinf for fun & profit. Technical Report 99020, Illionis Genetic Algorithms Laboratory, October 1999.

[Har99] Georges Harik. Linkage learning via probabilistic modeling in the ecga. Technical Report 99010, Illionis Genetic Algorithms Laboratory, January 1999.

[Pel05] Martin Pelikan. *Hierarchical Bayesian Optimization Algorithm*, chapter 1. Springer, 2005.

[PGT01] Martin Pelikan, David E. Goldberd, and Shigeyoshi Tsutsui. Combining the strengths of the bayessian optimization algorithm and adaptive evolution strategies. Technical Report 2001023, Illionis Genetic Algorithms Laboratory, June 2001.