Artur Zochniak 184725 12.11.12
grupa: pn TP 1515

projektowanie efektywnych algorytmów

branch & bound

metoda podziałów i ograniczeń

szeregowanie zadań z terminem wykonania

Wstęp

Zadanie polegało na implementacji algorytmu branch & bound w dowolnym języku w celu rozwiązania problemu szeregowania zadań z określonym terminem wykonania (i naliczanej karze w przypadku opóźnienia).

Jako	należy rozumieć
częściowe rozwiązanie	takie ułożenie, które ma jeszcze możliwość rozbudowy, tj. ma dzieci. Innymi słowy nie jest to końcowe rozwiązanie
rozwiązanie całkowite	liść w drzewie rozwiązań. Etap przeszukiwania, kiedy nie można zejść niżej.

Implementacja w programie

Algorytmy został zaimplementowany jako klasa Solver.

Klasa Solver

```
class Solver {
    protected:
    public:
    list<Zad> unordered;
    list<Zad> ordered;
    size_t bestTillNow;
    int N;
    size_t visitedVerticles;
    size_t visitedChecked;
    list<Zad> bestSol;
      size_t sequenceLen;
      double time;
//#ifdef _QT
      //void (SimulationViewer::*fun)(int,int);
//#else
      cbPtr fun;
//#endif
    Solver(const list<Zad> &jobs, const list<Zad>
&ordered=list<Zad>());
    virtual bool shouldEliminate(list<Zad> &unord, list<Zad> &ord,
size_t time, size_t extraCost, size_t nthExec);
    virtual size_t enterFunction(list<Zad> &unord, list<Zad> &ord,
size_t time, size_t extraCost, size_t nthExec);
    size_t solve(list<Zad> unord, list<Zad> ord, size_t time, size_t
extraCost, size_t nthExec, int x, int y, int parentX);
    virtual const char* getAlgoName();
      Solver& solveInterface();
      Solver& printWinningSequence();
      Solver& saveRaportToFile(ostream &cout, const char
thisTestName[]);
      void setFun(cbPtr);
};
```

Konstruktor egzemplarza klasy przyjmuje listę obiektów `Zad`, które są wczytywane z pliku.

Pozostałe implementacje metod eliminacyjnych nadpisują tylko wirtualną metodę `shouldEliminate`, która w przypadku zwrócenia wartości `PRAWDA` podczas sprawdzania danego wierzchołka informuje algorytm o tym, aby zaprzestał dalszego przeszukiwania tej gałęzi.

Przykładowa implementacja pierwszej metody eliminacyjnej

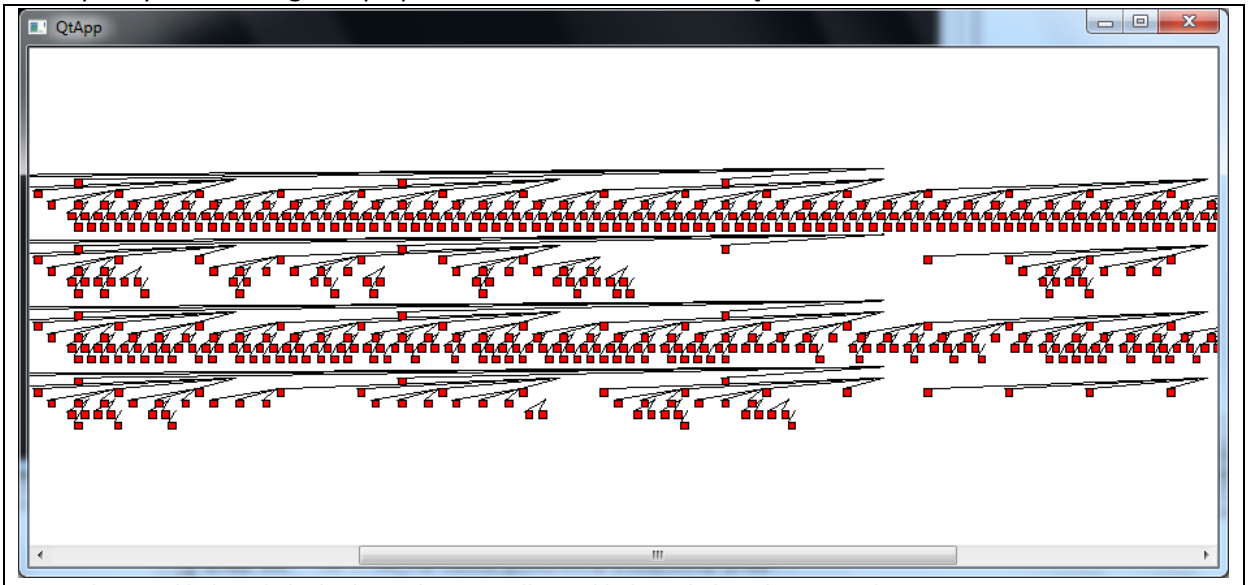
dzięki polimorfizmowi sprowadza się tylko do:

```
class SolverRemoveActualIfWorseThan:public Solver {
public:
    const char* getAlgoName() {
        return "OdetnijJesliGalazGorszaNizNajlepszeRozwiazanieDoTejPory";
    }
    SolverRemoveActualIfWorseThan(const list<Zad> &jobs, const list<Zad>
&ordered=list<Zad>()):Solver(jobs, ordered) {
    }
    virtual bool shouldEliminate(list<Zad> &unord, list<Zad> &ord, size_t time,
size_t extraCost, size_t nthExec) {
        if(extraCost>Solver::bestTillNow) return true;
        return false;
    }
};
```

Pogrubiony fragment odpowiada za porównanie najlepszego rozwiązania z aktualnym i w razie, kiedy rozpatrywana gałąź rokuje mniej korzystnymi warunkami – jest ona odcinana.

Wizualizacja

Dodatkowo, w celu prostej wizualizacji dodano nakładkę na klasę Solver, tj. klasę SimulationViewer, która wykorzystując bibliotekę Qt 4.8.3 realizuje wizualizację działania algorytmu.
Każdy z czterech algorytmów (przegląd zupełny + 3 algorytmy oparte o procedury eliminacyjne) jest uruchamiany w osobnym wątku i przekazuje stan swojego działania do okna reprezentującego działanie programu.
Dzięki możliwości zdefiniowania funkcji callback cbPtr możliwe jest wyrysowywanie w czasie rzeczywistym aktualnego etapu przeszukiwania drzewa rozwiązań.



Rysunek 1. Przykład wyglądu działania algorytmu dla przykładowych danych testowych 5.txt.

Pomiar czasu

czas liczony z wykorzystaniem klasy licznika opartego na WinApi.

```
class Timer {
       __int64 t1,t2,freq;
       public:
    Timer() {
        QueryPerformanceCounter( (LARGE_INTEGER*)&t1 );
    }
    double stop() {
        QueryPerformanceCounter( (LARGE_INTEGER*)&t2 );
        QueryPerformanceFrequency( (LARGE_INTEGER*)&freq );
        double time = (double)( t2 - t1 ) / freq;
        return time;
    }
};
```

Przegląd zupełny

Przegląd zupełny polega na przejrzeniu wszystkich możliwych rozwiązań oraz wybraniu najlepszego.

Takie podejście daje 100% pewność, że odnajdziemy optymalne rozwiązanie, ale trwa bardzo długo.

Implementacja obejmuje podejście bazujące na rekurencyjnej funkcji `rozwiąż`, której parametrami przekazywanymi rekurencyjnie są:

- lista uporządkowanych (wykorzystanych) wierzchołków,
- lista nieuporządkowanych (pozostałych) zadań.

Te dwa parametry pozwalają określić ile wierzchołków pozostało (długość listy nieuporządkowanych zadań) oraz ile już wykorzystano (długość listy uporządkowanych wierzchołków).

Dodając odpowiedni warunek można przekształcić algorytm przeglądu zupełnego w algorytm eliminujący pewne rozwiązania.

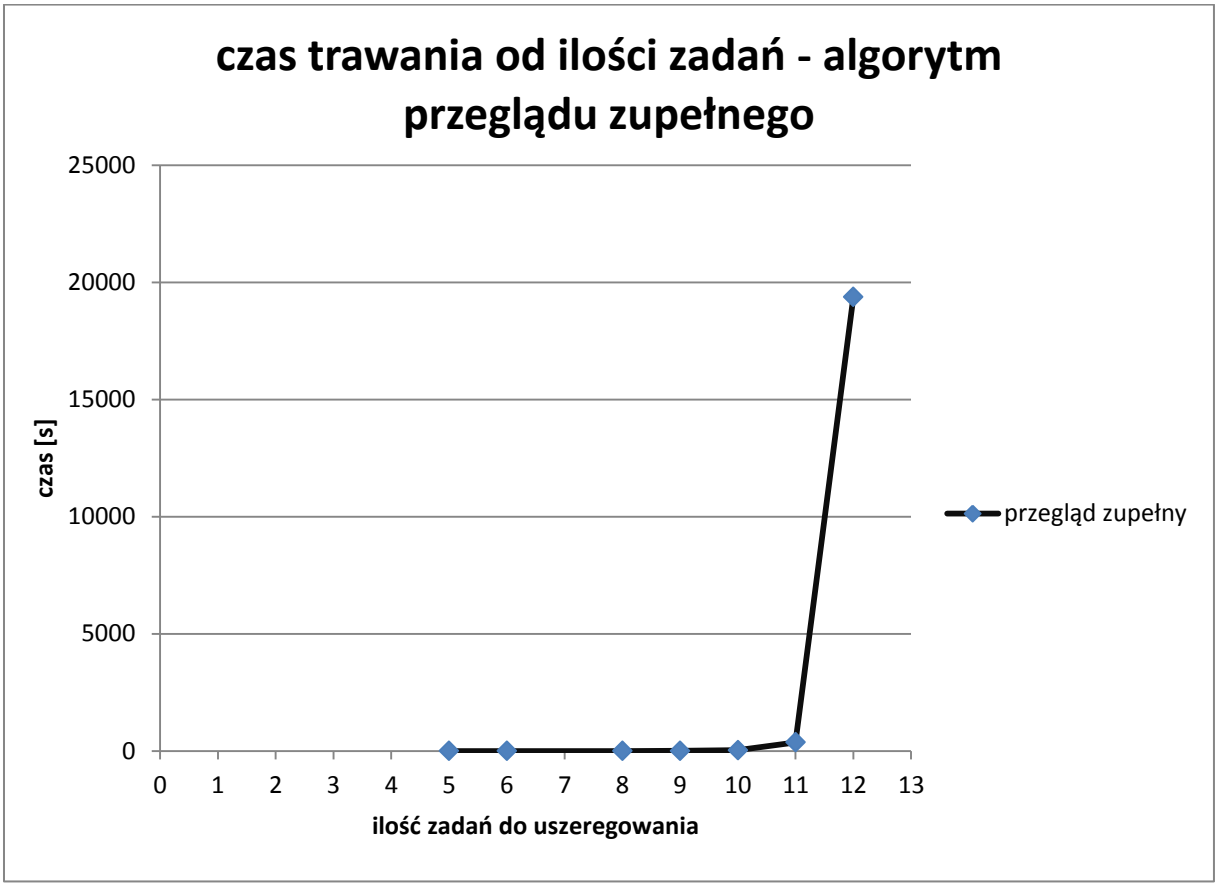
Pseudokod

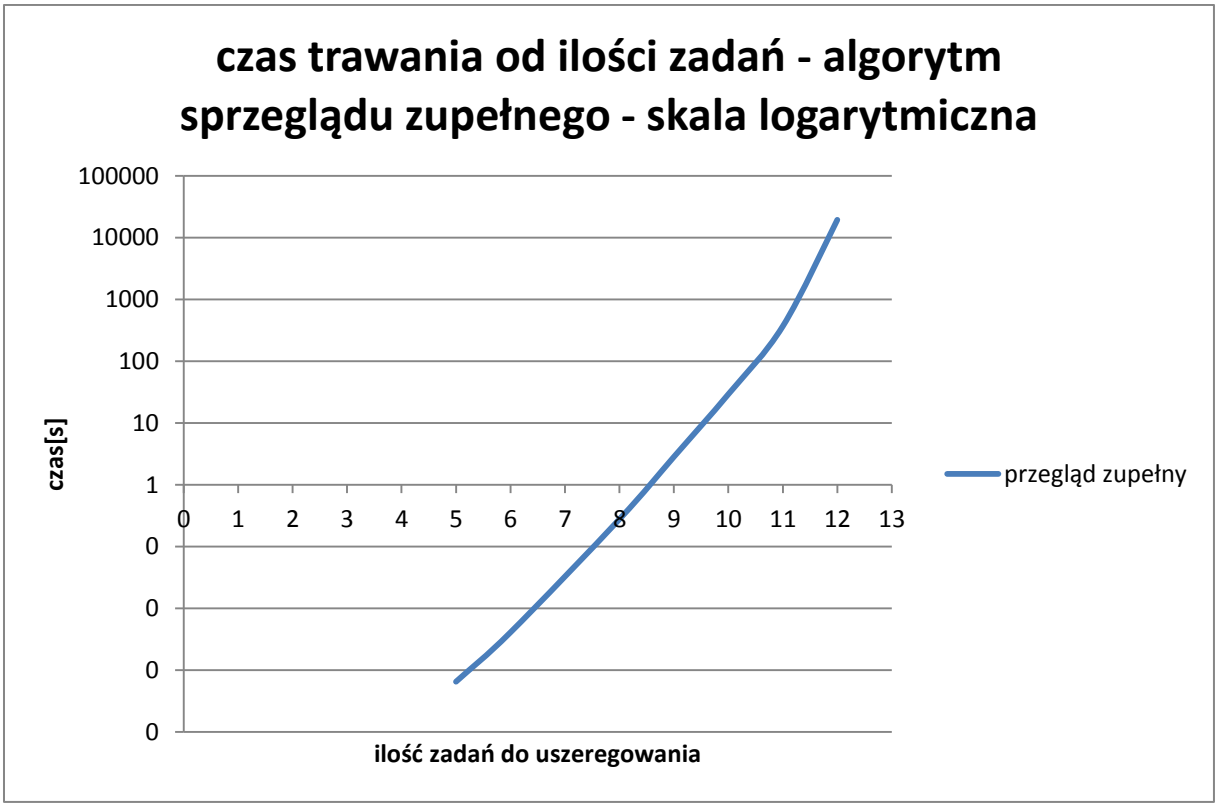
1. Wczytaj listę zadań.
2. uruchom procedurę rekurencyjną z dwoma lista, gdzie jedna jest pusta, a druga zawiera wczytane zadania,
    a. dla każdego zadania na liście nieuporządkowanej wywołuj kolejny raz funkcję z tym, że przełóż zadanie z listy zadań nieuporządkowanej na początek listy uporządkowanych zadań,
    b. po każdym wywołaniu funkcji rekurencyjnej odstaw dodane zadanie na koniec listy nieuporządkowanej,
    c. jeśli lista nieuporządkowana jest pusta – oblicz kryterium i – w razie jeśli jest korzystniejsze zapamiętaj je w celu dalszego porównywania z innymi rozwiązaniami.
3. Wypisz najlepsze rozwiązanie otrzymane dzięki przeglądowi zupełnemu.

Pomiary czasu działania algorytmu

test	czas średni[s]	odwiedzone wierzchołki	ilość zad.	najkrótsze uszeregowanie (wartość kryt.)	naj. odp.	pierwszy pomiar [s]	drugi pomiar - czas [s]
5	0,001	326	5	42	E B A D C	0,000662	0,000641014
6	0,004	1957	6	82	E F B D A C	0,004415	0,00377134
8	0,277	109601	8	223	G E F D H B A C	0,289455	0,265216
9	2,833	986410	9	313	G E F I D H B A C	2,69945	2,96689
10	29,247	9864101	10	370	G E F J D I B H A C	27,8249	30,6696
11	368,302	108505112	11	454	G E F J D I K B H A C	352,292	384,312
12	19376,500		12			19376,5	

Wykresy


czas trawania od ilości zadań - algorytm przeglądu zupełnego
25000
20000
15000
10000
5000
0
czas [s]
0 1 2 3 4 5 6 7 8 9 10 11 12 13
ilość zadań do uszeregowania
przegląd zupełny


czas trawania od ilości zadań - algorytm sprzeglądu zupełnego - skala logarytmiczna
100000
10000
1000
100
10
1
0
0
0
0
czas[s]
0 1 2 3 4 5 6 7 8 9 10 11 12 13
ilość zadań do uszeregowania
przegląd zupełny

Wnioski

Najprostsza metoda wyszukania najlepszego rozwiązania, ale już dla 12 zadań jej czas działania to aż 5,5 godziny. Obserwując tabelę z wynikami można dojść do wniosku, że zwiększenie ilości zadań do 13 wydłużyłoby czas działania do ponad 50h.
Bardzo prosta implementacja ale bardzo niska wydajność.

„Gorszy" - pierwsza metoda eliminacyjna – „eliminuj tylko jeśli to pewne, że jest lepsze rozwiązanie w innej gałęzi"

Pierwsza optymalizacja algorytmu polega na wyeliminowaniu tych rozwiązań, które na pewno nie będą lepsze w przyszłości niż aktualnie znalezione rozwiązanie. Polega na policzeniu kary, którą generuje dane częściowe ułożenie i jeśli już w momencie przechodzenia po drzewie kara będzie większa niż do tej pory znalezione rozwiązanie, to nie ma sensu dalej kontynuować poszukiwań w tym kierunku (dlatego, że kara z biegiem dobierania kolejnych zadań może tylko wzrosnąć).

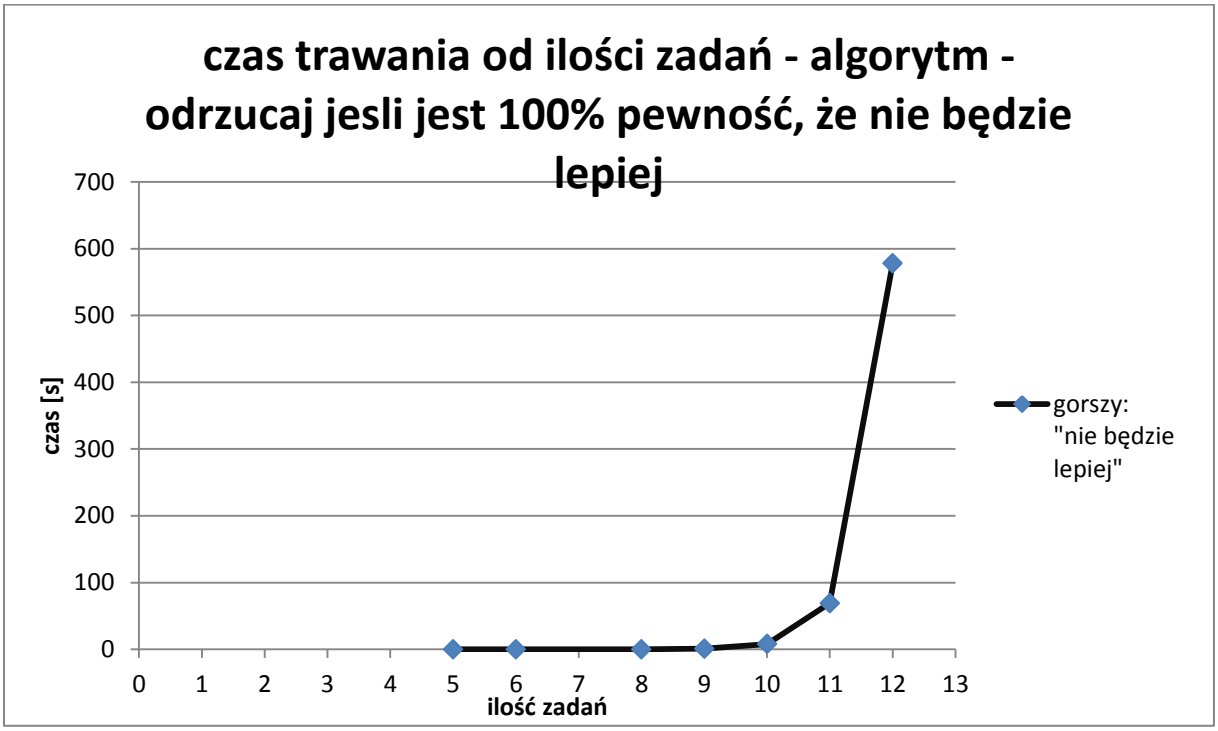
Przykład

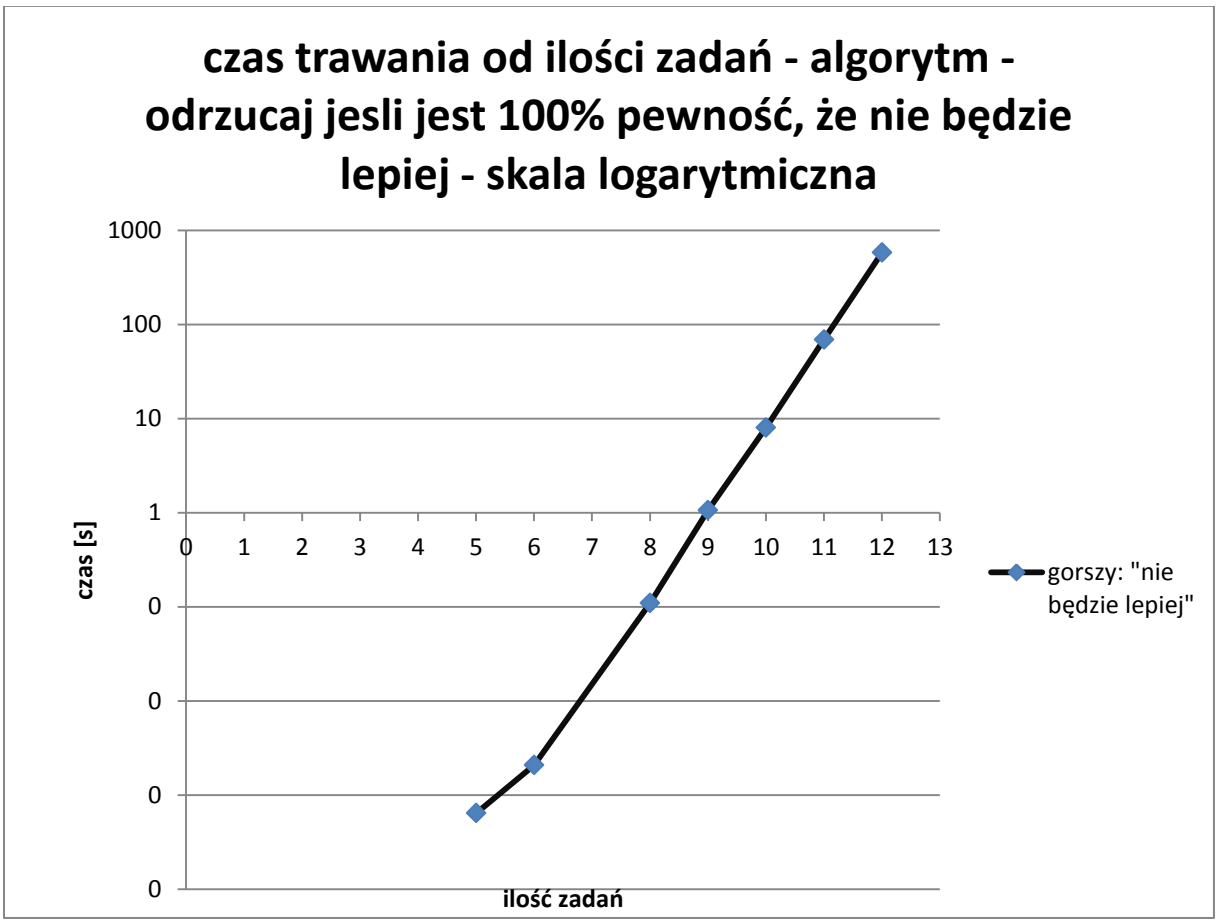
1. Rozpocznij przeszukiwanie metodą podziałów i ograniczeń.
2. Zejdź do najniższego liścia po lewej stronie.
3. Oblicz kryterium oceny rozwiązania.
4. Rozpoczęcie dalszego przeszukiwania w głąb, z tym, że:
    a. za każdym razem, kiedy algorytm wchodzi do dowolnego wierzchołka (gałęzi) sprawdź, czy sekwencja reprezentowana przez dany wierzchołek nie jest mniej korzysta niż najlepsze znalezione do tej pory rozwiązanie,
    b. jeśli jest, to zaprzestań przeszukiwania tej gałęzi,
    c. kontynuuj przeszukiwanie w głąb aż do wyczerpania wszystkich możliwych rozwiązań, pamiętając o sprawdzaniu warunku (**a.**).

Pomiary czasu działania algorytmu

test	czas średni[s]	wierzchołki odwie.	ilość zad.	najkrótsze uszeregowanie	1. pomiar [s]	2. pomiar [s]	najlepsza odp.
5	0,000	167	5	42	0,000627703	0,000653814	E B A D C
6	0,002	875	6	82	0,0020664	0,00207408	E F B D A C
8	0,109	37786	8	223	0,10943	0,109363	G E F D H B A C
9	1,059	317643	9	313	1,12649	0,991893	G E F I D H B A C
10	8,004	2080289	10	370	8,19929	7,80908	G E F J D I B H A C
11	68,937	14906059	11	454	67,6851	70,1881	G E F J D I K B H A C
12	578,065	122523804	12	599	578,065	578,065	G L E J D F K I B H A C

Wykresy





Wnioski

Procedura eliminująca wydajnie instancje do kilku zadań. Powyżej 10 zadań czas rośnie bardzo szybko.

„Zamień” – druga metoda eliminacyjna.

Druga optymalizacja polega na zamienieniu elementu ostatniego tak, aby zamienił się kolejno z wszystkimi pozostałymi i, w konsekwencji, stał się pierwszym.

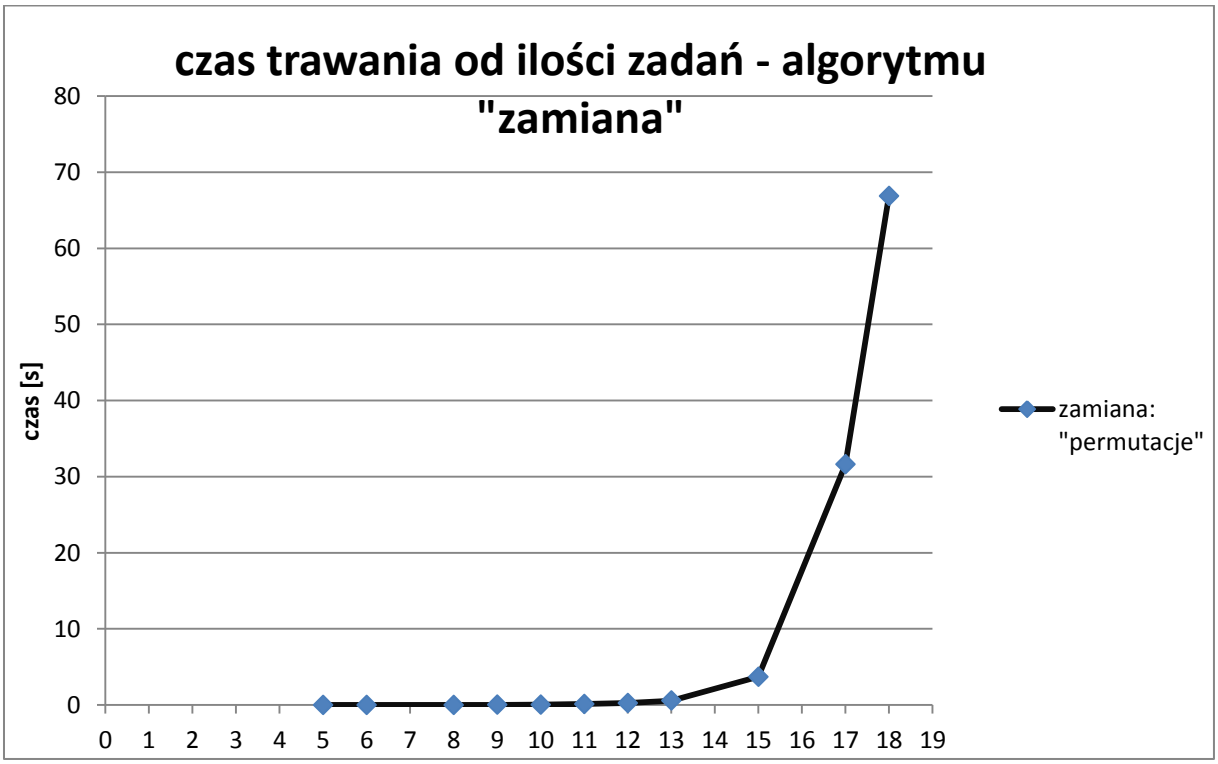
Przykład

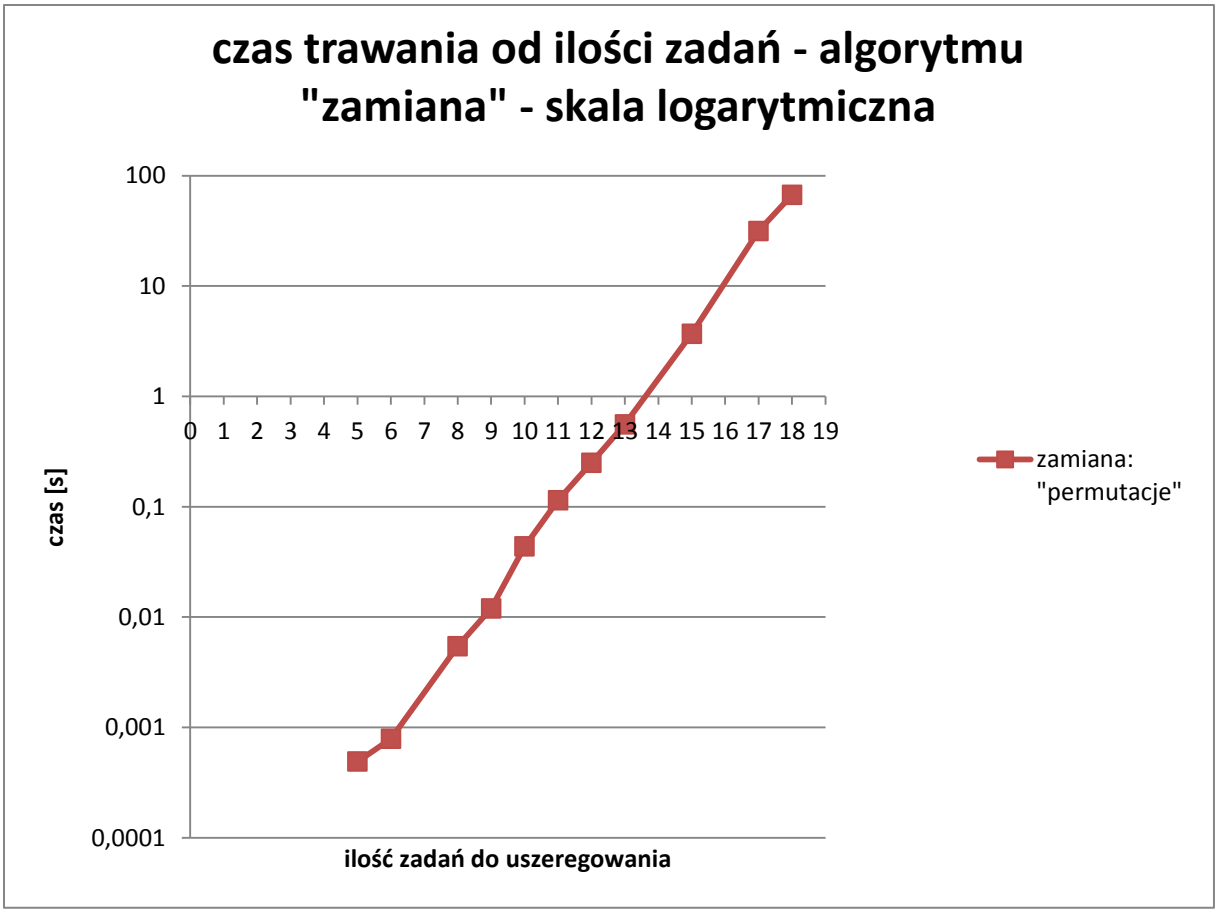
1. W początkowej fazie ułożenie to ABCDE,
2. Jeśli którekolwiek ułożenie z poniższych kroków jest korzystniejsze niż w kroku 1. – należy zaprzestać przeszukiwania tej gałęzi rozwiązań, bo lepsze rozwiązanie znajduje się w innej gałęzi.
    a. następuje zamiana dwóch ostatnich elementów - D z E, otrzymano: ABCED,
    b. następnie zamiana na E na C – otrzymana sekwencja ABECD,
    c. w kolejnych krokach następuje zamiana coraz bardziej oddalonych od końca elementów,
    d. w końcowym etapie otrzymana sekwencja to EABCD. Ostatni element (E) stał się pierwszym.
3. sprawdzenie kolejnego rozwiązania – wejście do kolejnej gałęzi - wróć do kroku 2. – powtarzaj aż do wyczerpania możliwych rozwiązań.

Pomiary czasu działania algorytmu

test	czas średni [s]	liczba odw. wierzchołków	ilość zad.	najkrótsze uszeregowanie	1. pomiar [s]	2. pomiar [s]	naj. odp.
5	0,0005	45	5	47	0,000489465	0,000543224	E B D A C
6	0,0008	80	6	92	0,000787956	0,000991729	E F D B A C
8	0,0050	357	8	223	0,00544043	0,00456288	G E F D H B A C
9	0,0116	690	9	313	0,0119392	0,0112674	G E F D I B H A C
10	0,0448	3065	10	382	0,0436801	0,0460609	G F J E D I B H A C
11	0,1134	8366	11	502	0,11432	0,112625	G F J E K D I B H A C
12	0,2611	15620	12	657	0,250369	0,271965	L G J F K E D I B H A C
13	0,5625	32770	13	855	0,5589	0,566117	L G J F K E D I B H M A C
15	3,9244	191152	15	1103	3,70017	4,14881	L G J F K E D I N B H M A C O
17	32,049	1307824	17	1162	31,6051	32,4929	L G J E D F P K N I B H M A C O Q
18	67,405	2615809	18	1346	66,882	67,928	L G J E D F P K N I B H M A C O R Q

Wykresy





Wnioski

Szybka procedura eliminująca dająca zadowalające efekty zarówno dla małych jak i dużych danych wejściowych.

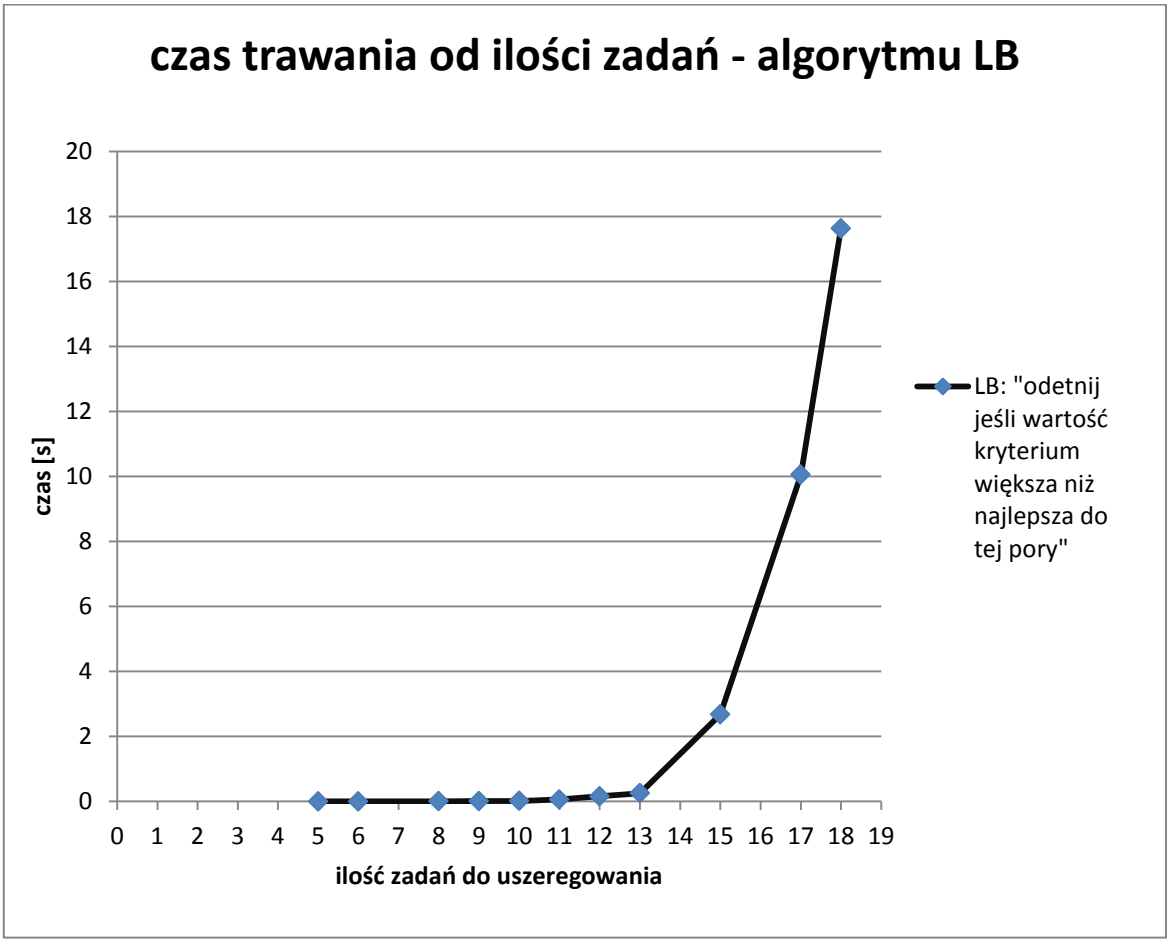
„Dolna granica" – trzecia procedura eliminacyjna.

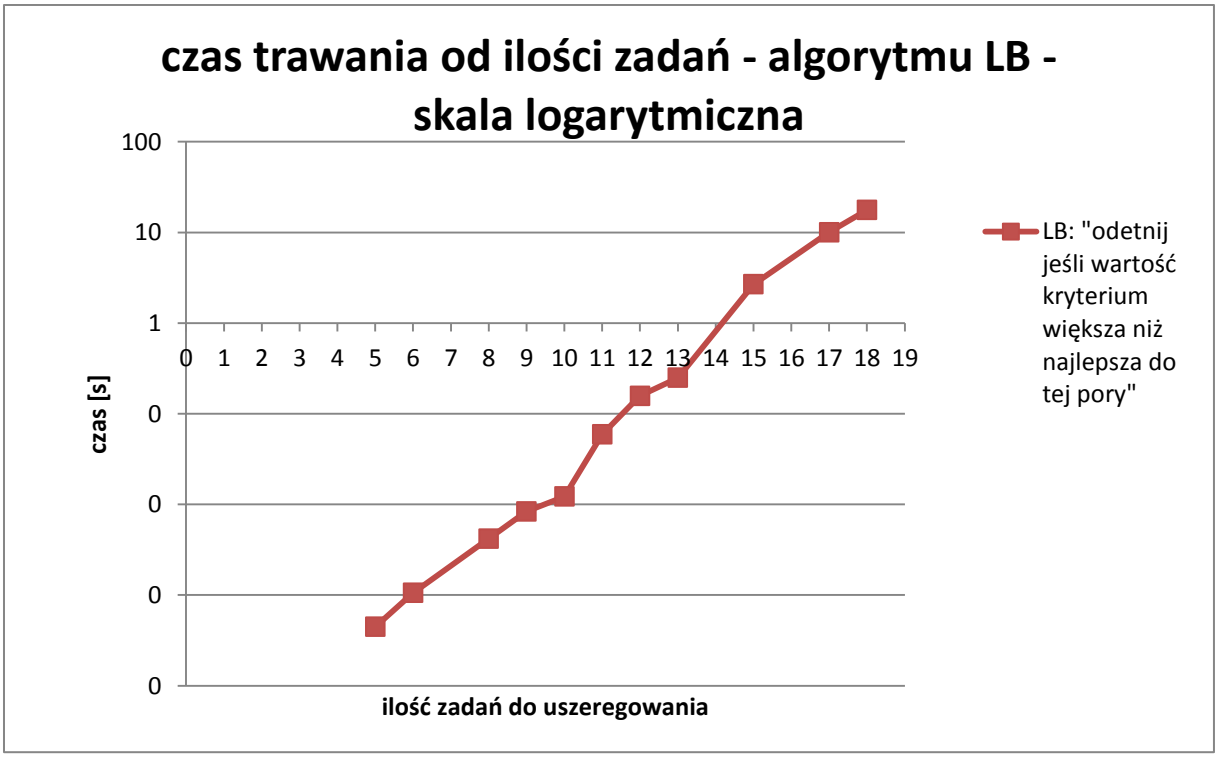
Trzecia metoda optymalizacyjna polega na obliczeniu dolnej granicy kryterium, które musi spełnić częściowe rozwiązanie, aby algorytm podjął próbę sprawdzania rozwiązań wywodzących się z niego. Najkorzystniejsze rozwiązanie całkowite uzyskane w miarę czasu działania algorytmu dyktuje również minimalną wartość kryterium oceny rozwiązania, które algorytm jest w stanie zaakceptować.

Pomiary czasu działania algorytmu

test	czas średni [s]	odwiedzone wierzchołki	ilość zad.	najkrótsze uszeregowanie	1. pomiar [s]	2. pomiar [s]	
5	0,0004	61	5	46	0,000473081	0,00041881	E A B D C
6	0,0010	139	6	89	0,00113458	0,000973298	E F A D B C
8	0,0041	450	8	224	0,00417632	0,00421626	G E F D H B C A
9	0,0083	733	9	314	0,00833012	0,00842074	G E F D I B H C A
10	0,0121	857	10	386	0,011634	0,0127317	E F G J D I B C H A
11	0,0590	4390	11	509	0,0594904	0,0587081	E F G J K I D B C H A
12	0,1573	4507	12	633	0,249598	0,0650266	E G L J D F I K B C H A
13	0,2495	14442	13	909	0,240298	0,258782	E G J L K F I D M B C H A
15	2,6724	143947	15	1147	2,5743	2,77055	D L E J G K F I N M B C H O A
17	10,0514	430740	17	1238	9,90287	10,2001	L G J D K P F I E N M B C H O Q A
18	17,6323	755807	18	1386	17,7439	17,5208	E L G J K N P F I D M A B C H O R Q

Wykresy


czas trawania od ilości zadań - algorytmu LB
czas [s]
20
18
16
14
12
10
8
6
4
2
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
ilość zadań do uszeregowania
LB: "odetnij jeśli wartość kryterium większa niż najlepsza do tej pory"


czas trawania od ilości zadań - algorytmu LB - skala logarytmiczna
czas [s]
100
10
1
0
0
0
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
ilość zadań do uszeregowania
LB: "odetnij jeśli wartość kryterium większa niż najlepsza do tej pory"

Wnioski

Algorytm uzyskujące nieco gorsze wyniki w porównaniu z pozostałymi, ale radzący sobie doskonale z większymi ilościami zadań.

Porównanie

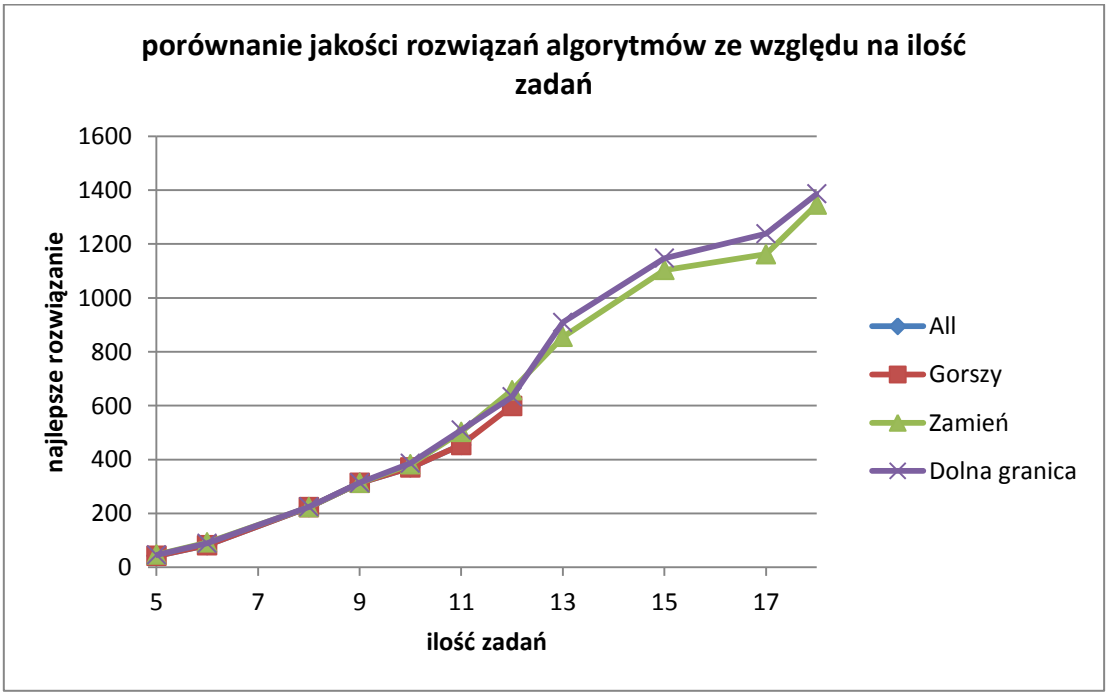
Poniżej przedstawiono zestawienie wyników uzyskanych przez wszystkie algorytmy.

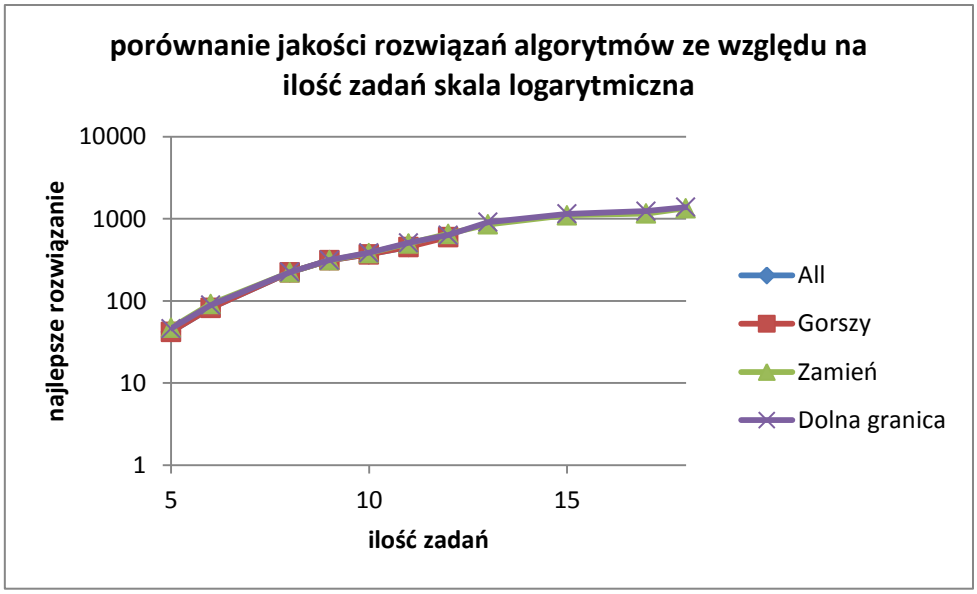
- A – przegląd zupełny,
- B – „gorszy”,
- C – „zamiana”,
- D – „dolna granica”,

błąd dla poszczególnych rozwiązań liczony jest ze wzoru (otrzymane_rozwiązanie/najlepsze rozwiązanie-1) [%].

Jakość rozwiązań

test	najlepszy wynik	A	błąd [%]	B	błąd [%]	C	błąd [%]	D	błąd [%]
5	42	42	0%	42	0%	47	12%	46	10%
6	82	82	0%	82	0%	92	12%	89	9%
8	223	223	0%	223	0%	223	0%	224	0%
9	313	313	0%	313	0%	313	0%	314	0%
10	370	370	0%	370	0%	382	3%	386	4%
11	454	454	0%	454	0%	502	11%	509	12%
12	599			599	0%	657	10%	633	6%
13	855					855	0%	909	6%
15	1103					1103	0%	1147	4%
17	1162					1162	0%	1238	7%
18	1346					1346	0%	1386	3%

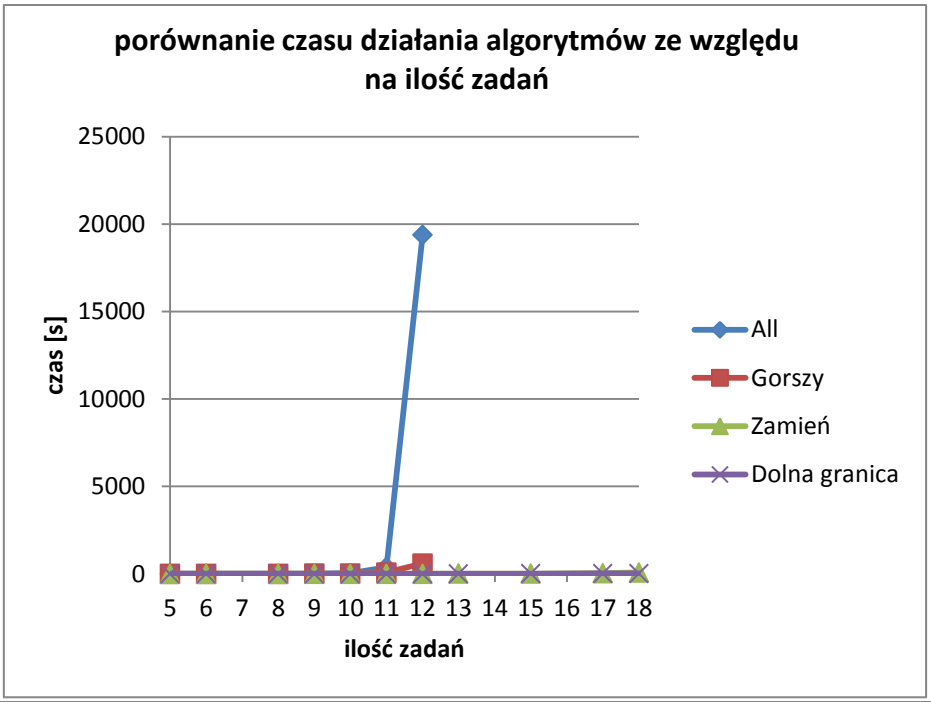


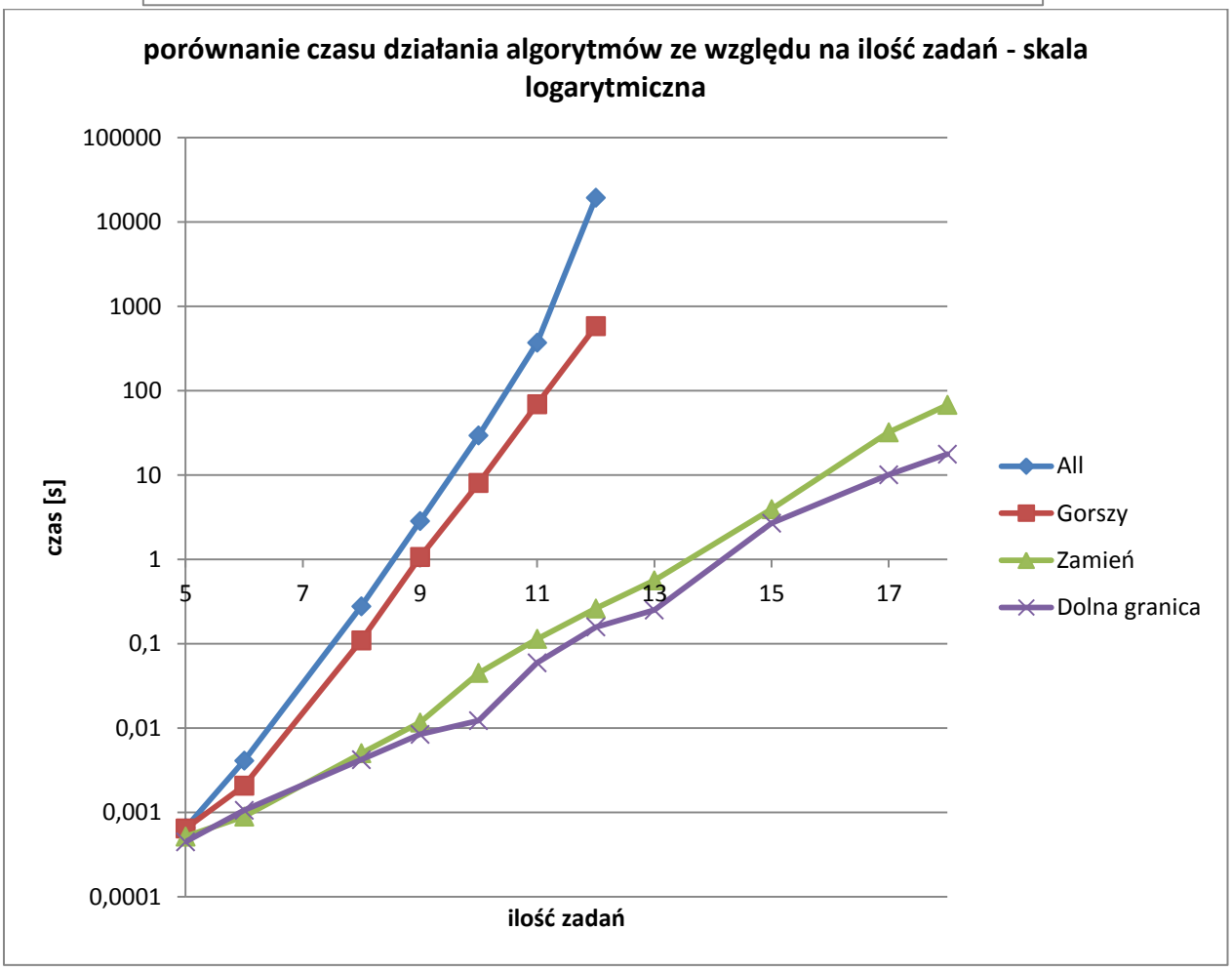


Jakości rozwiązań dla wszystkich algorytmów są, przy bardzo ogólnej ocenie, bardzo podobne. Nie ma algorytmu dającego znacząco gorsze wyniki.

Szybkość działania

test	najlepszy wynik [s]	A	błąd [%]	B	błąd [%]	C	błąd [%]	D	błąd [%]
5	0,00	0,00	46%	0,00	44%	0,00	16%	0,00	0%
6	0,00	0,00	360%	0,00	133%	0,00	0%	0,00	18%
8	0,00	0,28	6509%	0,11	2507%	0,01	19%	0,00	0%
9	0,01	2,83	33727%	1,06	12546%	0,01	39%	0,01	0%
10	0,01	29,25	239969 %	8,00	65600%	0,04	268%	0,01	0%
11	0,06	368,30	623092 %	68,94	116545 %	0,11	92%	0,06	0%
12	0,16	19376,50		578,0 7	367363 %	0,26	66%	0,16	0%
13	0,25					0,56	125%	0,25	0%
15	2,67					3,92	47%	2,67	0%
17	10,05					32,05	219%	10,05	0%
18	17,63					67,41	282%	17,63	0%

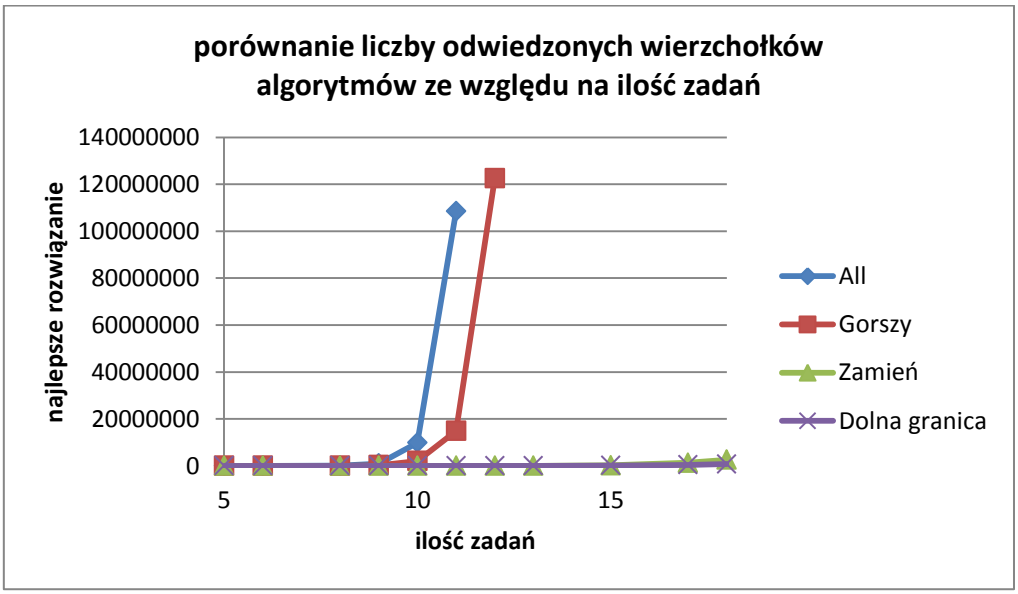


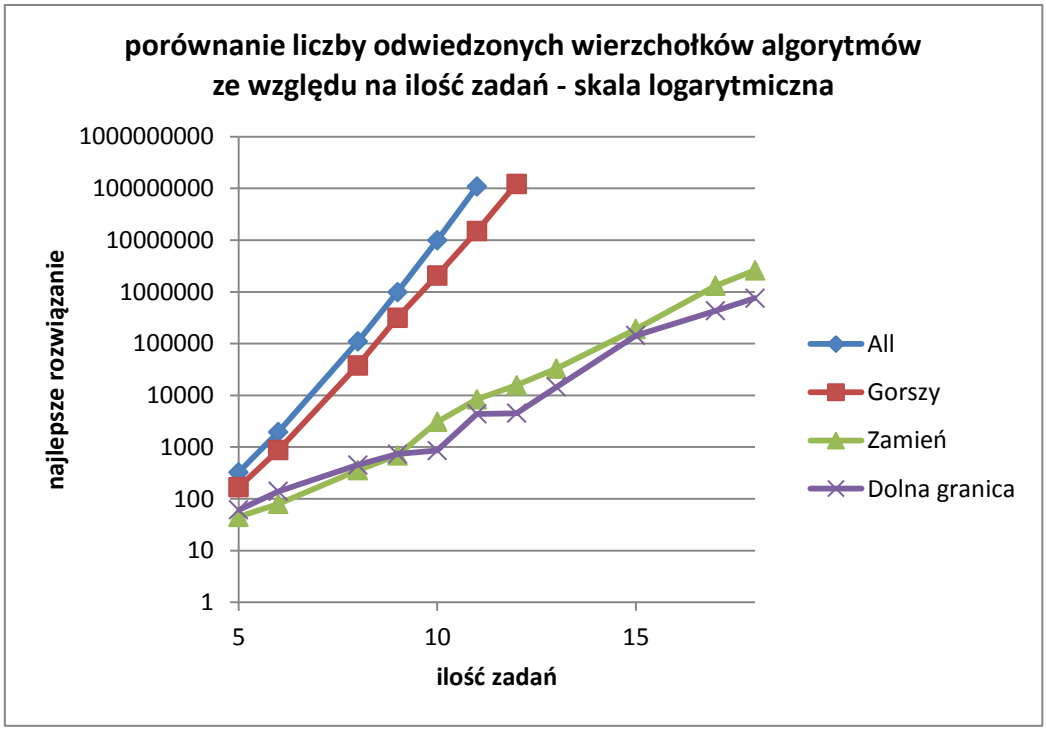


Najszybciej rosnącym algorytmem jest przegląd zupełny. Najbardziej stabilnym – algorytm dolna granica liczący kryterium i akceptujący wierzchołki z oszacowanym kryterium z wartością korzystniejszą niż najlepsze całkowite rozwiązanie znalezione do tej pory.

Ilość odwiedzonych wierzchołków

test	najlepszy wynik [s]	A	błąd [%]	V	błąd [%]	C	błąd [%]	D	błąd [%]
5	45	326	624%	167	271%	45	0%	61	36%
6	80	1957	2346%	875	994%	80	0%	139	74%
8	357	109601	30601%	37786	10484%	357	0%	450	26%
9	690	986410	142858%	317643	45935%	690	0%	733	6%
10	857	9864101	1150904%	2080289	242641%	3065	258%	857	0%
11	4390	109000000	2471543%	14906059	339446%	8366	91%	4390	0%
12	4507			122523804	2718422%	15620	247%	4507	0%
13	14442					32770	127%	14442	0%
15	143947					191152	33%	143947	0%
17	430740					1307824	204%	430740	0%
18	755807					2615809	246%	755807	0%





Podsumowanie

Wygląda na to, że algorytm oparty na trzeciej metodzie eliminacyjnej działa najwydajniej na dużych instancjach problemu, natomiast metoda eliminacyjna polegający na zamienianiu ostatniego elementu z pozostałymi daje sobie doskonale radę przy rozwiązywaniu mniejszych instancji (mniej niż 10 zadań).

Bibliografia

1. plik „B&B.pdf” otrzymany drogą elektroniczną na zajęciach,
2. ftp://sith.ict.pwr.wroc.pl/Informatyka/PEA/bnb_example_1_i_P_sumWCA.pdf – slajd 13. – trzeci algorytm eliminacyjny.