Projektowanie Efektywnych Algorytmów

Zadanie projektowe nr 2

Mikołaj Pastuszek 259138

Czwartek 11:15

Prowadzący: Mgr inż. Antoni Sterna

1. Cel projektu

   Zadanie projektowe opierało się na implementacji oraz analizie efektywności algorytmu Tabu Search (TS) dla problemu komiwojażera (TSP).

2. Wstęp teoretyczny

   2.1. Opis problemu komiwojażera

   Biorąc pod uwagę zestaw miast i odległość między każdą parą miast, problem polega na znalezieniu najkrótszej możliwej trasy, która odwiedza każde miasto dokładnie raz i wraca do punktu początkowego. W projekcie rozwiązywaliśmy asymetryczny problem komiwojażera. Jest to trudniejsza wersja tego problemu gdyż długość ścieżki z punktu A do B i z B do A mogą się od siebie różnić.

   2.2. Opis algorytmu TS

   Tabu Search (TS) jest metaheurystyk, która szuka rozwiązania problemu poprzez nadzorowanie innych procedur heurystycznych, w celu eksploracji przestrzeni rozwiązań poza lokalne minimum.

   Algorytm w krokach:

   Krok 1: Najpierw zaczynamy od rozwiązania początkowego . Może to być dowolne rozwiązanie spełniające kryteria akceptowalnego rozwiązania.

   Krok 2: Wygeneruj zestaw rozwiązań sąsiadujących z aktualnym rozwiązaniem. Z tego zestawu rozwiązań usuwane są rozwiązania znajdujące się na Liście Tabu, z wyjątkiem rozwiązań spełniających kryteria aspiracji.

   Krok 3: Wybierz najlepsze rozwiązanie spośród rozwiązań sąsiadujących. Jeśli rozwiązanie jest lepsze niż aktualnie najlepsze rozwiązanie, zaktualizuj aktualne najlepsze rozwiązanie. Elementy algorytmu.

   Krok 4: Zaktualizuj Listę Tabu usuwając wszystkie ruchy, które wygasły po okresie Tabu Tenure, dekrementuj kadencje pozostałych ruchów i dodaj nowe ruchy do Listy Tabu.

   Krok 5: Wykonaj dywersyfikacje jeśli kryterium zostało spełnione.

   Krok 6: Jeśli kryteria zakończenia zostaną spełnione, wyszukiwanie zostanie zatrzymane lub przejdzie do następnej iteracji. Kryteria zakończenia zależą od problemu, ale niektóre możliwe przykłady to:

   - maksymalna liczba iteracji
   - jeśli najlepsze znalezione rozwiązanie jest lepsze niż jakiś próg
   - miną określny czas pracy algorytmu

2.3. Elementy algorytmu TS

a) Sąsiedztwo

Sąsiedztwo to zestaw możliwych rozwiązań otrzymanych przez wykonanie ruchu lub ruchów na aktualnym rozwiązaniu.

b) Lista ruchów zakazanych (Tabu List)

Lista Tabu przechowuje ostatnio wykonane ruchy (w niektórych implementacjach zamiast ruchów używane są kompletne rozwiązania), które przez pewną kadencje będą zabronione do wykonania.

c) Kadencja tabu

Kadencja tabu (Tabu Tenure) to liczba iteracji, przez które ruch pozostaje na Liście Tabu. Ruchy, które znajdują się na Liście Tabu, to ruchy, których nie można wykonać ponownie, ponieważ zostały już niedawno odwiedzone.

d) Kryterium aspiracji

Kryterium aspiracji jest pewnym warunkiem, który pozwala na wykonanie ruchu, nawet jeżeli ruch ten jest zakazany.

e) Dywersyfikacja

Jest to zmiana obecnego rozwiązania na nowe rozwiązanie startowe, pod pewnym warunkiem np.:

- przez K kolejnych iteracji nie zostało znalezione lepsze rozwiązanie
- algorytm wykonał K iteracji od wygenerowania nowego rozwiązania startowego
- przez K kolejnych iteracji były przeglądane rozwiązania “bliskie” rozwiązaniu startowemu (pojęcie “bliskie” zależy od konkretnego problemu).

3. Opis zaimplementowanego algorytmu w krokach

Krok 1: Znalezienie początkowego rozwiązania zachłannym algorytmem.

Krok 2: Wygenerowanie zestawu rozwiązań sąsiadujących z aktualnym rozwiązaniem poprzez funkcje **generateNeighbourhood()** i wprowadzenie ich do kolejki priorytetowej segregującej rozwiązania po koszcie. Koszt nowego rozwiązania to różnica między nowym kosztem ścieżki a aktualnym kosztem.

Krok 3: Sprawdzeni czy rozwiązanie o najmniejszym koszcie znajduje się w Liście Tabu. Jeśli rozwiązanie się znajduje wykonaj Krok 4, w przeciwnym wypadku wykonaj Krok 5.

Krok 4: Sprawdź czy rozwiązanie spełnia kryterium aspiracji. Jeśli tak to zaktualizuj Listę Tabu usuwając wszystkie ruchy, które wygasły po okresie Tabu Tenure, dekrementuj kadencje pozostałych ruchów i zaktualizuj ruchy w Liście Tabu. Wybierz rozwiązanie jako aktualne. Jeśli dane rozwiązanie jest lepsze niż aktualnie najlepsze rozwiązanie, zaktualizuj aktualne najlepsze rozwiązanie.

W przypadku gdy kryterium aspiracji nie jest spełnione usuń rozwiązanie z kolejki priorytetowej i wróć do Kroku 3. Wyjątkiem jest gdy dane rozwiązanie jest ostatnim rozwiązaniem. W tym przypadku wybierz je jako aktualne i zaktualizuj Listę Tabu usuwając wszystkie ruchy, które wygasły po okresie Tabu Tenure, dekrementuj kadencje pozostałych ruchów i zaktualizuj ruchy w Liście Tabu. Omiń Krok 5.

(Wybranie ostatniego ruchu jest to mechanizm aby nie wpaść w pętle)

Krok 5: Zaktualizuj Listę Tabu usuwając wszystkie ruchy, które wygasły po okresie Tabu Tenure, dekrementuj kadencje pozostałych ruchów i dodaj nowy ruch do Listy Tabu oraz do listy **tabuHelp**. Wybierz rozwiązanie jako aktualne. Sprawdź czy koszt ruchu był ujemny. Jeśli tak to porównaj obecną drogę z aktualnie najlepszą. Jeśli obecna jest lepsza to zaktualizuj aktualne najlepsze rozwiązanie.

(**tabuHelp** jest to struktura pomocnicza, która zawiera wszystkie zabronione ruchy znajdujące się w **tabuMatrix.** Powstała ona aby nie musieć przeszukiwać całej macierzy.)

Krok 6: Wykonaj dywersyfikacje jeśli najlepsze rozwiązanie nie zmieniło się przez 20000 iteracji.

Krok 7: Jeśli kryteria zakończenia zostaną spełnione, wyszukiwanie zostanie zatrzymane, w przeciwnym wypadku przejdź do kroku 2. W tym przypadku kryterium to czas działania algorytmu.

Krok 8: Wypisz wynik algorytmu.

4. Opis najważniejszych klas

Najważniejszy kod programu, odpowiadający za logikę zaimplementowanego algorytmu, znajduję się w klasie TabuSearch.

4.1. Zmienne oraz struktury

```
int** matrix; //graf
int size; //wielkość grafu
unsigned long long stopTime; //czas w sekundach

vector <int > bestPath; //najlepsza ścieżka
int bestPathLength; //koszt najlepszej ścieżki

vector <int > currentPath; //obecna ścieżka
int currentPathLength; //koszt obecnej ścieżki

int** tabuMatrix; //Tabu List
list<vector<int>> tabuHelp; //tu znajduja się krawędzie dodane do tabu (struktura pomocnicza)

long int whenDiversification; //po ilu cyklach wykonac dywersyfikacje
long int countNotFind; //licznik do dywersyfikacji

long long int frequency, start, elapsed; //zmienne potrzebne do wyliczenia najlepszego czasu

//prawdopodobna zamia wierzchołków i jej koszt
struct Move
{
    int vIndex1;
    int vIndex2;
    int cost;

    bool operator<(const Move& o) const
    {
        return cost > o.cost;
    }
};
```

4.2. Funkcje

```
    void coutFirstPath(); //znalezienie pierwszej drogi greedy algorymem
    priority_queue<Move> generateNeighbourhood(vector<int> path); //tworzenie sąsiedztwa
    void decrementTabuMatrix(); //zmiejszenie kadencji zakazanych krawedzi i usunięcie z tabuHelp wketorów
    void checkIfBestPath(); //porównanie najlepszej i obecnej drogi i zamina jesli obecna jest mniejsza
    void diversification(); //dywersyfikacja
    void clearQueue(priority_queue<Move>& q);//usuniecie wszystkich wartości z kolejki
    long long int read_QPC();
    int tabuTenure(); //wyznaczenie kadencji w tabuMatrix

public:
    TabuSearch(int** matrixCon, int sizeCon, unsigned long long stopTimeCon);
    ~TabuSearch();

    void tabuSearch(bool isDiversification); //główna metoda
    void writeResult(); //wypisanie wyników
```

4.3. Opisy funkcji

Nazwa funkcji	Opis
coutFirstPath	Generuje aktualną i najlepszą ścieżkę (są takie same) oraz ich koszty wykorzystując do tego zachłanny algorytm.
generateNeighbourhood	Generuje sąsiedztwo dla aktualnego rozwiązania.
decrementTabuMatrix	Dekrementuje wszystkie wartości w tabuMatrix oraz usuwa ruchy z tabuHelp, jeśli kadencja się skończyła.
checkIfBestPath	Porównuje czy obecna ścieżka jest mniejsza od najlepszej. Jeśli tak to zamienia najlepszą wartość, resetujemy licznik do dywersyfikacji oraz przypisujemy do zmiennej wartość dzięki której obliczymy najlepszy czas.
diversification	Miesza miejscami wierzchołki grafu (drogę).
clearQueue	Czyszczenie z pamięci kolejki.
read_QPC	Funkcja pomocnicza do wyliczania czasu.
tabuSearch	Główna funkcja.
writeResult	Wypisanie wyniku.

5. Plan i wyniki eksperymentu

5.1. Przebieg eksperymentu

Testy efektywności zostały przeprowadzone dla trzech plików z grafami: br17.astp, ftv47.astp, rgb403.astp.

Pliki zostały przekonwertowane przez program znajdujący się na stornie: http://staff.iiar.pwr.wroc.pl/antoni.sterna/pea/tests/tsp_conv.cpp,tak aby pasowały do założeń projektowych. Każdy plik został przetestowany 2 razy po 10 min. Za pierwszym razem z wyłączoną dywersyfikacją, a za drugim razem z dywersyfikacją. Za każdym razem, gdy najlepsza ścieżka się zmieniała program wypisywał ją. W tabelach zostały zapisane wszystkie koszty dróg oraz czasy znalezienia owych dróg wypisane przez algorytm. Dodatkowo pod tabelami zostały wypisane najlepsze drogi. Po za tabelami zostały stworzone wykresy na podstawie danych z tabel. Osi X zostały dopasowane do czasu znajdowanych rozwiązań.

5.2. Wyniki eksperymentu

5.2.1. Wyniki dla pliku „br17.astp"

Plik zawiera przykład grafu z 17 wierzchołkami. Najkrótsza droga dla rozpatrywanego problemu to 39.

a) Wynik w postaci tabeli dla algorytmu z wyłączona dywersyfikacją

Droga	92	83	7	76	75	68	67	66	63	60	58	55	50	47	44	42	**41**
Czas[ms]	0	4	6	8	10	11	22	27	35	39	42	55	60	67	70	73	**87**
Błąd względny	1,359	1,128	1,000	0,949	0,923	0,744	0,718	0,692	0,615	0,538	0,487	0,410	0,282	0,205	0,128	0,077	**0,051**

Najlepsza ścieżka obliczona przez algorytm:

0 11 13 2 10 9 1 12 7 16 8 15 5 14 4 3 6 0

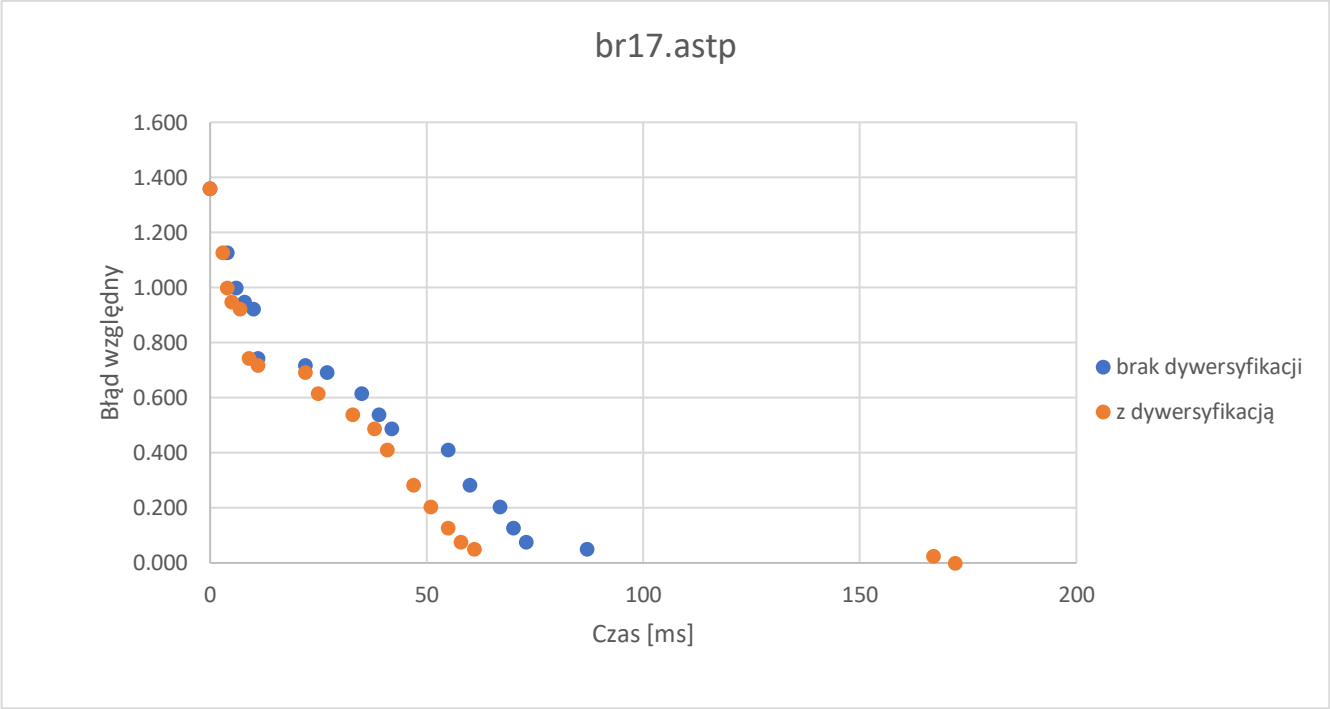
b) Wynik w postaci tabeli dla algorytmu z włączoną dywersyfikacją

Droga	92	83	78	76	75	68	67	66	63	60	58	55	50	47	44	42	41	40	**39**
Czas[ms]	0	3	4	5	7	9	11	22	25	33	38	41	47	51	55	58	61	167	**172**
Błąd względny	1,359	1,128	1,000	0,949	0,923	0,744	0,718	0,692	0,615	0,538	0,487	0,410	0,282	0,205	0,128	0,077	0,051	0,026	**0,000**

Najlepsza ścieżka obliczona przez algorytm:

0 2 13 1 10 12 9 8 7 16 4 3 15 14 6 5 11 0

c) Przedstawienie wyniku w postaci wspólnego wykresu dla algorytmu z oraz bez dywersyfikacji



5.2.2. Wyniki dla pliku „ftv47.astp”

Plik zawiera przykład grafu z 48 wierzchołkami. Najkrótsza droga dla rozpatrywanego problemu to 1776.

a) Wynik w postaci tabeli dla algorytmu z wyłączona dywersyfikacją

Droga	2374	2347	2318	2300	2293	2235	2229	2197	2174	2131	2112	2081	2080	2078	2057	2043	2035	2013	**2007**
Czas[ms]	0	4	157	161	164	169	175	180	180	959	1748	1752	15226	15230	15235	15241	49338	56050	**129637**
Błąd względny	0,337	0,322	0,305	0,295	0,291	0,258	0,255	0,237	0,224	0,200	0,189	0,172	0,171	0,170	0,158	0,150	0,146	0,133	**0,130**

Najlepsza ścieżka obliczona przez algorytm:

0 25 39 21 40 47 26 1 10 11 8 9 4 29 30 5 31 7 23 34 46 36 35 14 15 45 16 18 17 13 12 32 6 24 3 33 27 28 2 41 43 42 22 44 19 20 38 37 0

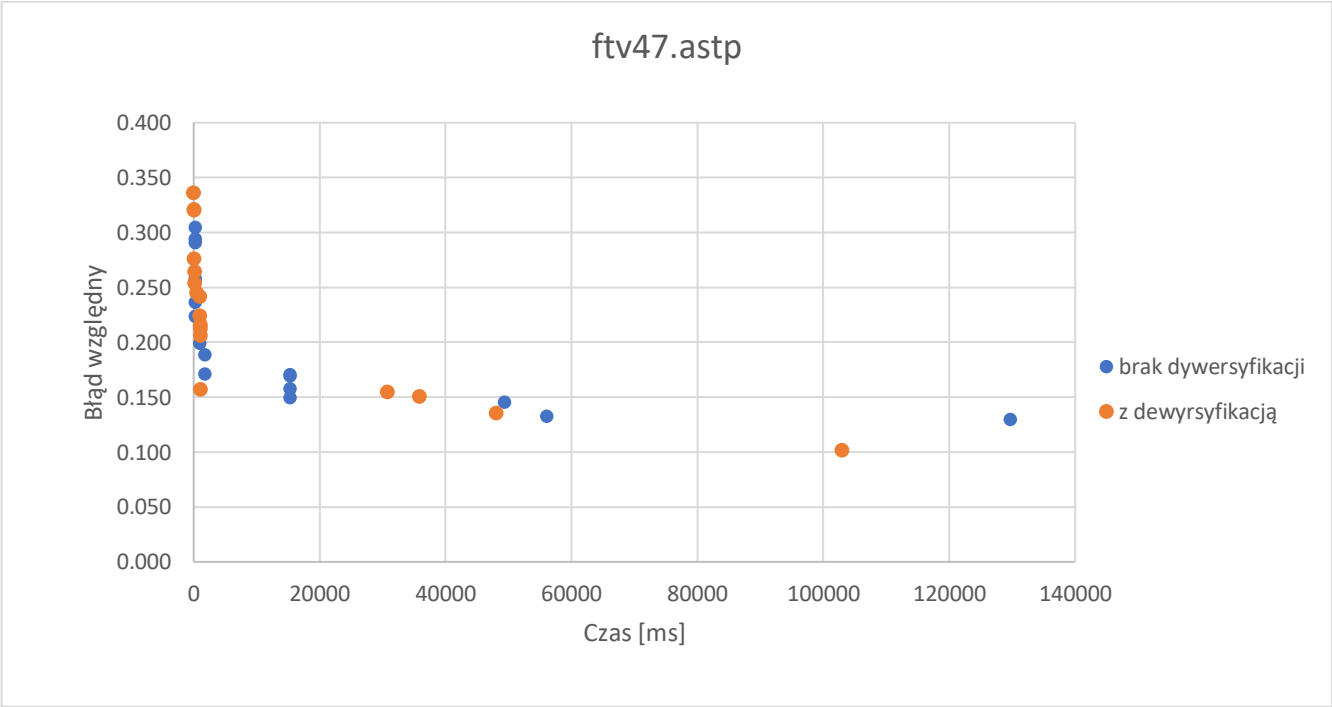
b) Wynik w postaci tabeli dla algorytmu z włączoną dywersyfikacją

Droga	2374	2347	2346	2267	2246	2227	2212	2206	2175	2160	2154	2143	2056	2051	2044	2017	**1957**
Czas[ms]	0	4	87	89	105	113	464	931	998	1098	1105	1109	1115	30810	35816	48089	**102925**
Błąd	0,337	0,322	0,321	0,276	0,265	0,254	0,245	0,242	0,225	0,216	0,213	0,207	0,158	0,155	0,151	0,136	**0,102**

Najlepsza ścieżka obliczona przez algorytm:

0 20 37 38 18 17 12 7 23 13 46 36 14 34 35 15 16 45 19 44 39 21 40 47 26 22 41 43 42 1 25 9 33 27 2 28 3 24 4 29 30 5 31 32 6 8 11 10 0

c) Przedstawienie wyniku w postaci wspólnego wykresu dla algorytmu z oraz bez dywersyfikacji



5.2.3. Wyniki dla pliku „rgb403.astp”

Plik zawiera przykład grafu z 403 wierzchołkami. Najkrótsza droga dla rozpatrywanego problemu to 2465.

a) Wynik w postaci tabeli dla algorytmu z wyłączona dywersyfikacją

Droga	3535	3534	3530	3529	3528	3527	3522	3521	3514	3512	3506	3502	3501	3499	3498	3495	3486	3481	3480	3479	3466	3462	3460	3458	3450
Czas[ms]	0	20	43	66	89	116	258	286	317	345	373	404	456	548	663	738	805	835	870	908	931	959	999	1054	1126
Błąd względny	0,434	0,434	0,432	0,432	0,431	0,431	0,429	0,428	0,426	0,425	0,422	0,421	0,420	0,419	0,419	0,418	0,414	0,412	0,412	0,411	0,406	0,404	0,404	0,403	0,400
Droga	3438	3433	3432	3431	3421	3413	3406	3402	3400	3398	3392	3385	3384	3366	3365	3358	3351	3349	3346	3344	3337	3330	3324	3318	**3317**
Czas[ms]	1159	1185	1218	1246	1270	1293	1318	1342	1364	1743	2626	2973	9067	9127	18167	27282	95811	245869	245893	299065	299087	299112	299136	553483	**553524**
Błąd względny	0,395	0,393	0,392	0,392	0,388	0,385	0,382	0,380	0,379	0,378	0,376	0,373	0,373	0,366	0,365	0,362	0,359	0,359	0,357	0,357	0,354	0,351	0,348	0,346	**0,346**

Najlepsza ścieżka obliczona przez algorytm:

0 267 46 23 14 62 13 205 204 24 36 286 273 83 33 376 68 64 105 19 18 194 380 397 84 257 43 34 295 50 56 394 225 58 8 6 235 121 2 386 47 112 322 272 28 11 101 10 55 59 76 310 29 65 5 260 35 160 371 281 176 31 52 40 39 78 226 147 79 69 245 81 22 21 82 247 54 85 26 12 86 75 27 15 1 30 96 66 60 44 87 67 94 88 353 263 90 72 57 91 25 116 92 51 49 37 249 256 120 392 351 293 93 145 143 365 165 187 95 303 71 97 70 349 73 99 389 359 369 4 355 115 114 393 102 80 7 41 89 20 118 261 269 278 122 119 168 103 117 251 113 317 104 203 384 383 197 146 144 38 391 154 106 340 339 45 363 289 301 255 212 109 275 210 265 326 258 111 223 214 192 189 123 17 136 327 124 284 290 125 74 216 200 198 126 323 305 215 309 127 191 338 234 228 224 240 219 350 130 341 32 274 357 328 370 395 132 243 288 291 315 133 148 325 134 306 319 195 280 279 236 98 167 320 139 292 329 294 227 381 140 173 344 141 259 142 199 229 352 149 347 401 354 342 266 150 300 190 100 138 297 151 296 298 63 337 239 77 156 390 153 346 377 155 378 398 233 324 362 157 334 400 158 241 330 343 159 276 348 379 308 244 238 313 128 193 185 250 368 213 277 161 335 399 162 53 174 253 181 316 230 382 246 388 186 396 332 220 217 331 180 209 373 183 184 175 152 333 282 177 221 196 248 164 208 166 16 131 285 170 129 268 375 387 9 201 364 182 307 254 366 48 262 178 372 356 312 218 311 171 367 211 270 345 271 107 61 163 3 283 179 202 237 264 358 304 222 232 374 137 172 207 110 361 188 299 42 336 321 360 314 231 318 108 169 242 135 385 302 402 287 252 206 0

b) Wynik w postaci tabeli dla algorytmu z włączoną dywersyfikacją

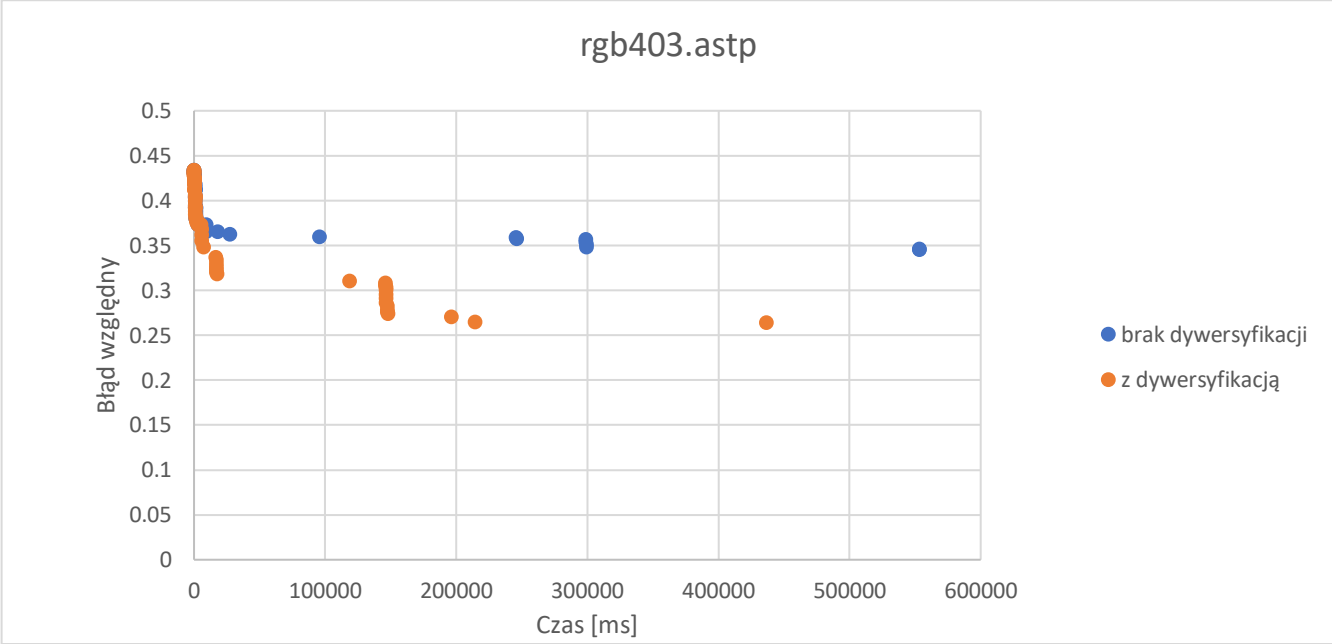
Droga	3535	3534	3530	3529	3528	3527	3522	3521	3514	3512	3506	3502	3501	3499	3498	3495
Czas[ms]	0	23	55	80	109	143	175	210	247	281	311	345	382	432	476	522
Błąd względny	0,434	0,434	0,432	0,432	0,431	0,431	0,429	0,428	0,426	0,425	0,422	0,421	0,420	0,419	0,419	0,418
Droga	3486	3481	3480	3479	3466	3462	3460	3458	3450	3438	3433	3432	3431	3421	3413	3406
Czas[ms]	566	628	676	723	768	818	865	899	932	972	1007	1065	1116	1151	1191	1229
Błąd względny	0,414	0,412	0,412	0,411	0,406	0,404	0,404	0,403	0,400	0,395	0,393	0,392	0,392	0,388	0,385	0,382
Droga	3402	3400	3398	3392	3385	3384	3372	3357	3340	3323	3295	3288	3277	3269	3261	3254
Czas[ms]	1266	1303	1713	1738	3008	5495	5656	5855	6004	7431	16765	16914	16967	17031	17150	17195
Błąd względny	0,380	0,379	0,378	0,376	0,373	0,373	0,368	0,362	0,355	0,348	0,337	0,334	0,329	0,326	0,323	0,320
Droga	3249	3230	3225	3218	3213	3206	3194	3182	3171	3161	3151	3146	3141	3131	3117	**3116**
Czas[ms]	17409	118717	146165	146228	146299	146376	146469	146558	146677	147249	147391	147508	148086	196501	214193	**436392**
Błąd względny	0,318	0,310	0,308	0,305	0,303	0,301	0,296	0,291	0,286	0,282	0,278	0,276	0,274	0,270	0,265	**0,264**

Najlepsza ścieżka obliczona przez algorytm:

0 237 109 130 312 35 333 267 386 354 342 337 77 31 309 184 244 80 14 69 153 75 81 22 41 170 334 161 148 325 222 185 379 318 383 259 125 298 373 355 6 140 66 74 216 200 221 300 70 99 145 340 126 10 398 141 116 110 159 20 115 189 158 1 151 338 246 345 299 49 331 323 201 118 399 18 132 306 168 286 215 400 367 76 15 58 8 73 88 62 36 314 310 206 30 164 169 27 11 186 53 305 37 248 348 105 19 297 268 16 210 202 87 56 396 370 139 4 377 92 347 352 171 63 71 360 353 401 344 181 346 104 349 52 40 39 174 250 249 280 279 361 51 265 275 313 378 266 33 364 303 122 341 78 226 156 42 68 131 214 192 114 393 79 96 176 258 127 177 135 154 335 372 21 188 173 190 100 291 95 271 350 336 321 362 123 230 382 205 357 326 273 330 290 155 143 392 120 199 262 236 227 144 208 240 219 43 384 368 149 146 60 44 241 112 61 391 317 133 162 48 2 172 26 84 93 197 320 111 369 366 292 329 332 296 381 97 302 89 304 5 117 150 295 90 72 198 311 55 119 29 207 223 319 239 175 82 247 220 389 187 50 387 9 231 288 343 254 23 225 194 380 113 272 28 283 277 263 253 209 278 224 233 301 351 257 375 191 91 94 385 67 38 212 103 397 394 264 390 211 46 213 356 47 107 234 228 57 142 129 255 12 178 138 388 294 83 252 371 327 183 17 180 359 232 24 13 64 365 32 328 293

137 289 402 287 324 124 339 166 134 243 54 242 179 284 108 256 85 235 121 358 281 182 136 395 238 229 374 363 204 285 282 86 101 217 203 316 193 157 269 160 3 270 322 196 45 218 34 261 245 102 276 152 59 251 274 98 167 315 163 25 106 165 376 308 260 65 128 195 147 307 7 0

c) Przedstawienie wyniku w postaci wspólnego wykresu dla algorytmu z oraz bez dywersyfikacji



6. Wnioski

Jak widać na powyższych wykresach czym dłużej algorytm się wykonuje tym pokazuje lepsze wyniki. Algorytm posiadający włączoną dywersyfikację pokazywał lepszy wynik dla każdego grafu. Algorytm Tabu Serach jest algorytmem heurystycznym co również można dostrzec w wynikach algorytmu. Jedynie dla przykładu br17.astp udało mu się znaleźć optymalną drogę natomiast dla dwóch pozostałych ftv47.astp i rgb403.astp rozwiązania posiadały błąd względny na poziomie 0,102 oraz 0,264 przy włączonej dywersyfikacji.

7. Kod źródłowy

Link do dysku z kodem i sprawozdaniem: https://drive.google.com/drive/folders/1qzOB3Xar0RW098CIi2xAjO4ppdD5h2lC .