POLITECHNIKA WROCŁAWSKA

PROJEKTOWANIE EFEKTYWNYCH ALGORYTMÓW
PROJEKT, WT 17:05
K00-58d

PROBLEM KOMIWOJAŻERA
ALGORYTM GENETYCZNY

Katarzyna Hajduk, 259189
ITE 2020, W4N

Prowadzący:
Dr inż. Jarosław Mierzwa

17 stycznia 2023

1. Wstęp teoretyczny

Zadanym problemem jest zaimplementowanie oraz dokonanie analizy efektywności algorytmu genetycznego dla asymetrycznego problemu komiwojażera (*ATSP*). Problem komiwojażera jest to zagadnienie optymalizacyjne polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie. Cykl Hamiltona jest to taki cykl w grafie, w którym każdy wierzchołek grafu został odwiedzony dokładnie raz w wyłączeniem pierwszego wierzchołka.

2. Założenia

Podczas realizacji zadania należy przyjąć następujące założenia:

- używane struktury danych powinny być alokowane dynamicznie (w zależności od aktualnego rozmiaru problemu),
- program powinien umożliwić wczytanie danych testowych z pliku - te pliki to: ftv47.atsp (1776), ftv170.atsp (2755) , rgb403.atsp (2465). W nawiasach podano najlepsze znane rozwiązanie dla danych zawartych w pliku - długość drogi.
- program musi umożliwiać wprowadzenia kryterium stopu jako czasu wykonania podawanego w sekundach,
- implementacje algorytmów należy dokonać zgodnie z obiektowym paradygmatem programowania, używanie „okienek” nie jest konieczne i nie wpływa na ocenę (wystarczy wersja konsolowa),
- kod źródłowy powinien być komentowany.

3. Implementacja algorytmu genetycznego

Algorytm genetyczny (ang. *Genetic algorithm*) jest algorytmem, który naśladuje naturalne procesy ewolucji. Ewolucja odbywa się w postaci niewielkich zmian jakości osobników oraz całej populacji, dążących do możliwie najlepszego spełnienia narzuconych warunków przez środowisko. Osobniki w kolejnych iteracjach algorytmu zostają poddane selekcji, krzyżowaniu oraz mutacji. Algorytm zatrzymuje się, gdy dojdzie do warunku stopu.

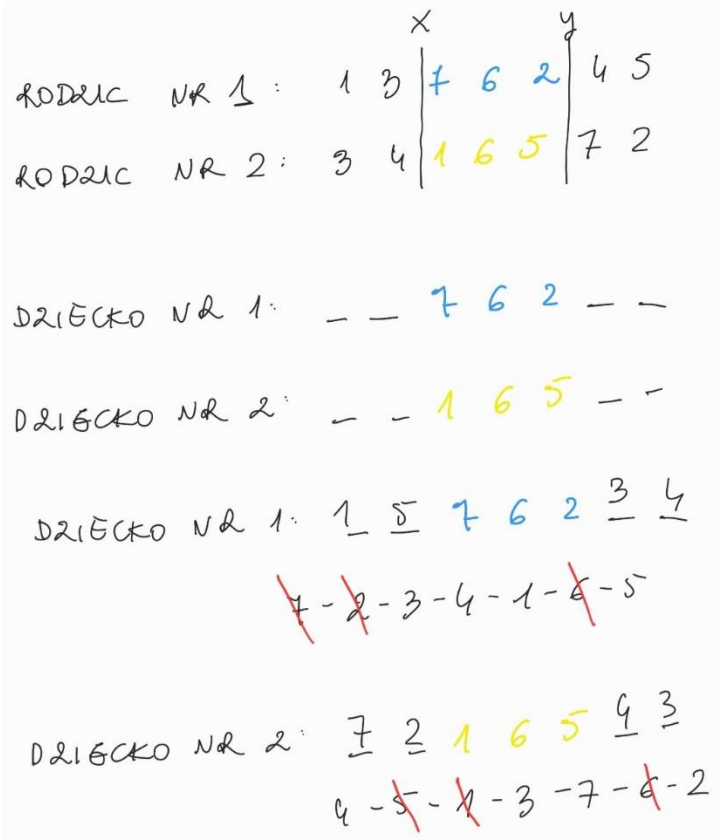
3.1. Selekcja turniejowa

Selekcja polega na wyborze z bieżącej populacji osobników, których poddamy krzyżowaniu. Wybór odbywa się przy pomocy selekcji turniejowej. Polega ona na tworzeniu turniejów kosztu z uczestnikami wybranymi z populacji. Z turnieju wybierany jest najlepszy osobnik, który trafia do puli osobników do krzyżowania. Liczba turniejów wynosi 10% populacji początkowej.

3.2. Krzyżowanie OX

Krzyżowanie polega na wymianie materiału genetycznego pomiędzy dwoma osobnikami. W wyniku tej operacji powstają całkiem nowe osobniki. Proces ten może zajść z danym prawdopodobieństwem.

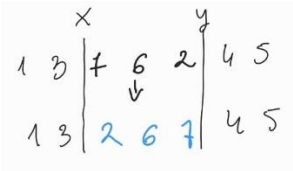
Krzyżowanie OX (ang. *Order Crossover*) polega na wybraniu dwóch punktów krzyżowania. Segment w ten sposób utworzony kopiuje się do potomka. Następnie przepisujemy wierzchołki z drugiego rodzica w porządku, rozpoczynając zaraz za skopiowanym segmentem. Jednocześnie omijamy miasta już wpisane. Drugiego potomka tworzymy zamieniając kolejność rodziców.



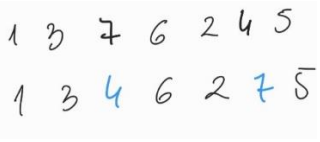
3.3. Mutacja

Mutacja polega na zamianie wartości wybranego genu osobnika. Celem mutacji jest wprowadzenie zmienności chromosomów. Proces ten może zajść z danym prawdopodobieństwem.

3.3.1. Mutacja Inverse



3.3.2. Mutacja Swap



4. Najważniejsze klasy w algorytmie GA

4.1. Implementacja czasu

Pomiar czasu zmierzyłam za pomocą QueryPerformanceCounter(). Zwracany stan licznika jest dzielony przez częstotliwość. Tak otrzymujemy pomiar w sekundach.

```
Long Long int SimulatedAnnealing::read_QPC() {
    LARGE_INTEGER count;
    QueryPerformanceCounter(&count);
    return((Long Long int)count.QuadPart);
}
```

4.2. Tworzenie populacji początkowej

Generowanie populacji początkowej polega na tworzeniu danej ilości losowych ścieżek.

```
void GeneticAlgorithm::generateFirstPopulation(vector<vector<int>> &firstPopulation, int populationSize) {
    vector<int> path;
    for (int i = 0; i < size; i++)
        path.push_back(i);

    for (int i = 0; i < populationSize; i++) {
        random_shuffle(path.begin(), path.end());
        firstPopulation.push_back(path);
    }
}
```

4.3. Selekcja turniejowa

Wybieram dwóch rodziców, przeprowadzając turniej po kosztach.

```
vector<vector<int>> GeneticAlgorithm::tournament(vector<vector<int>> population) {
    vector<vector<int>> parents;
    parents.resize(2);

    int tournamentSize = population.size() / 10;

    for (int i = 0; i < 2; i++) {
        bestCost = INT_MAX;
        for (int j = 0; j < tournamentSize; j++) {
            int index = rand() % (population.size() - 1);
            int cost = fitness[index];

            if (cost < bestCost) {
                bestCost = cost;
                bestPath = population[index];
            }
        }
        parents[i] = bestPath;
    }

    return parents;
}
```

4.4. Przeprowadzanie mutacji

Mutację możemy przeprowadzić na dwa sposoby – swap i inverse. Używam gotowych funkcji bibliotecznych.

```
vector<int> GeneticAlgorithm::mutationSwap(vector<int> path) {
    int first = rand() % path.size();
    int second;

    do {
        second = rand() % path.size();
    } while (second == first);

    swap((path)[first], (path)[second]);

    return path;
}
```

```
vector<int> GeneticAlgorithm::mutationInverse(vector<int> path) {
    int first = rand() % path.size();
    int second;

    do {
        second = rand() % path.size();
    } while (second == first);

    if (first < second)
        reverse(path.begin() + first, path.begin() + second + 1);
    else
        reverse(path.begin() + second, path.begin() + first + 1);

    return path;
}
```

4.5. Krzyżowanie osobników

Najpierw deklarujemy wszystkie potrzebne wartości.

```
pair<vector<int>, vector<int>> GeneticAlgorithm::crossoverOX(vector<int> firstPath, vector<int> secondPath) {
    vector<int> firstChild;
    vector<int> secondChild;
    int x, y;

    firstChild.resize(size, -1);
    secondChild.resize(size, -1);

    vector<int> tmpFirstChild;
    vector<int> tmpSecondChild;

    bool *firstMapping = new bool[size];
    bool *secondMapping = new bool[size];

    for (int i = 0; i < size; i++) {
        firstMapping[i] = false;
        secondMapping[i] = false;
    }

    do {
        x = rand() % (size - 1);
        y = rand() % (size - 1);
    } while (x == y || x > y);
```

Następnie wpisujemy wartości pomiędzy punkami krzyżowania oraz zapisujemy wartości, które już wystąpiły. Potem wpisujemy wartości wierzchołków po puncie krzyżowania, z wyłączeniem tych, które już wystąpiły.

```
for (int i = x; i < y; i++) {
    firstChild[i] = firstPath[i];
    secondChild[i] = secondPath[i];

    firstMapping[firstPath[i]] = true;
    secondMapping[secondPath[i]] = true;
}

for (int i = y; i < size; i++) {
    if (!firstMapping[secondPath[i]])
        tmpFirstChild.push_back(secondPath[i]);

    if (!secondMapping[firstPath[i]])
        tmpSecondChild.push_back(firstPath[i]);
}

for (int i = 0; i < y; i++) {
    if (!firstMapping[secondPath[i]])
        tmpFirstChild.push_back(secondPath[i]);

    if (!secondMapping[firstPath[i]])
        tmpSecondChild.push_back(firstPath[i]);
}
```

Na końcu wpisujemy odpowiednie wartości w odpowiednie miejsca.

```
    int actFirstPosition = 0;
    int actSecondPosition = 0;
    for (int i = y; i < size; i++) {
        if (firstChild.at(i) == -1) {
            firstChild[i] = tmpFirstChild[actFirstPosition];
            actFirstPosition++;
        }
        if (secondChild.at(i) == -1) {
            secondChild[i] = tmpSecondChild[actSecondPosition];
            actSecondPosition++;
        }
    }

    for (int i = 0; i < x; i++) {
        if (firstChild.at(i) == -1) {
            firstChild[i] = tmpFirstChild[actFirstPosition];
            actFirstPosition++;
        }
        if (secondChild.at(i) == -1) {
            secondChild[i] = tmpSecondChild[actSecondPosition];
            actSecondPosition++;
        }
    }

    delete[] firstMapping;
    delete[] secondMapping;

    return make_pair(firstChild, secondChild);
}
```

4.6. Sortowanie populacji

Populację sortujemy po koszcie, a następnie usuwamy osobników o najgorszym koszcie, tak, aby wielkość populacji była równa wielkości populacji początkowej.

```
void GeneticAlgorithm::sortPopulation(vector<vector<int>> &population, int populationSize) {
    sort(population.begin(), population.end(), [this](auto i, auto j) -> bool {
        return getPathCost(i) < getPathCost(j);
    });

    while (population.size() != populationSize) {
        population.pop_back();
    }
}
```

4.7. Główna pętla programu

Najpierw sprawdzamy warunek stopu. Następnie liczymy koszt populacji, przeprowadzamy turniej oraz krzyżujemy wybrane osobniki oraz dodajemy je do populacji. Następnie na całej populacji przeprowadzamy mutację z pewnym prawdopodobieństwem. Na końcu sortujemy populację oraz usuwamy nadmiarowe osobniki.

```
while (endTime < timeToStop) {
    fitnessCost(firstPopulation);
    parents = tournament(firstPopulation);

    children = crossoverProbability(parents[0], parents[1], crossover);
    firstPopulation.push_back(children.first);
    firstPopulation.push_back(children.second);

    for(int i = 0; i < population; i++){
        child = mutationProbability(firstPopulation[i], mutation, mutationType);
        firstPopulation.push_back(child);
    }

    sortPopulation(firstPopulation, population);

    endTime = ((read_QPC() - startTime) / frequency);
}
fitnessCost(firstPopulation);
```

5. Wyniki algorytmu

Dla trzech plików przyjmuję wielkość populacji początkowej = {100, 1000, 1500} oraz limit czasowy = 120s. Algorytm powtarzam 10 razy. Początkowo przyjmuję współczynnik krzyżowania 0.8 oraz współczynnik mutacji 0.01.

Następnie sprawdzam wpływ współczynnika mutacji na wyniki dla wartości = {0.02, 0.05, 0.10} dla najlepszej wielkości populacji, która wynosi 100.

5.1. Ftv47 – zmiana populacji

Sposób mutacji	Wielkość populacji	Kryterium stopu [s]	Średni koszt	Średni błąd [%]
Swap	100	120	2221	19.79927
	1000		2165	17.96073
	1500		2115	15.93933
Inverse	100		2407	26.21037
	1000		2212	19.47633
	1500		2167	17.6054

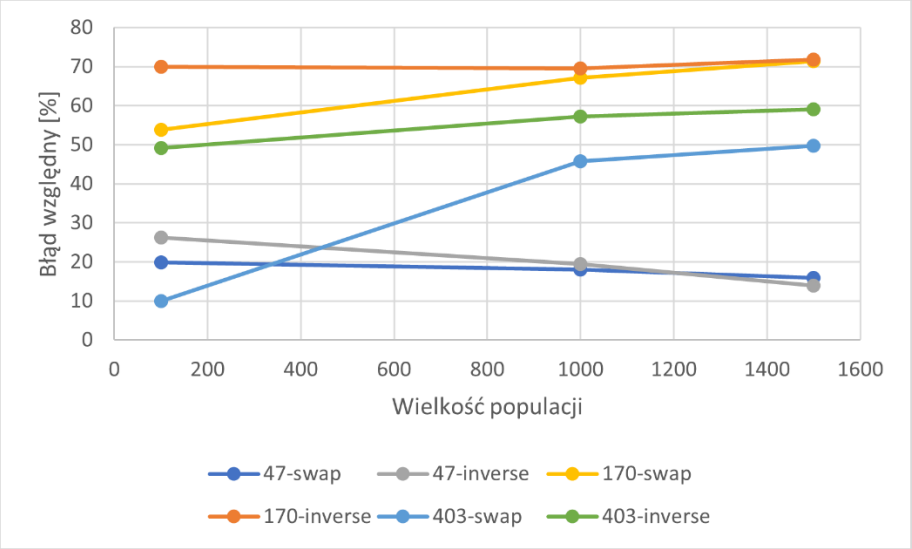
5.2. Ftv170 – zmiana populacji

Sposób mutacji	Wielkość populacji	Kryterium stopu [s]	Średni koszt	Średni błąd [%]
Swap	100	120	5968	53.79757
	1000		8390	67.1517
	1500		9629	71.38027
Inverse	100		9163	69.9021
	1000		9069	69.59873
	1500		9801	71.86595

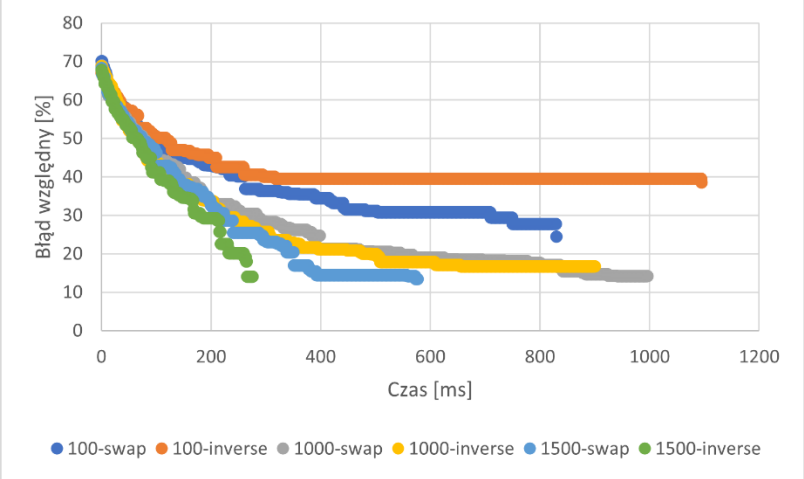
5.3. Rbg403 – zmiana populacji

Sposób mutacji	Wielkość populacji	Kryterium stopu [s]	Średni koszt	Średni błąd [%]
Swap	100	120	2738	9.93651
	1000		4541	45.70735
	1500		4901	49.70865
Inverse	100		4855	49.2259
	1000		5762	57.2225
	1500		6024	59.0803

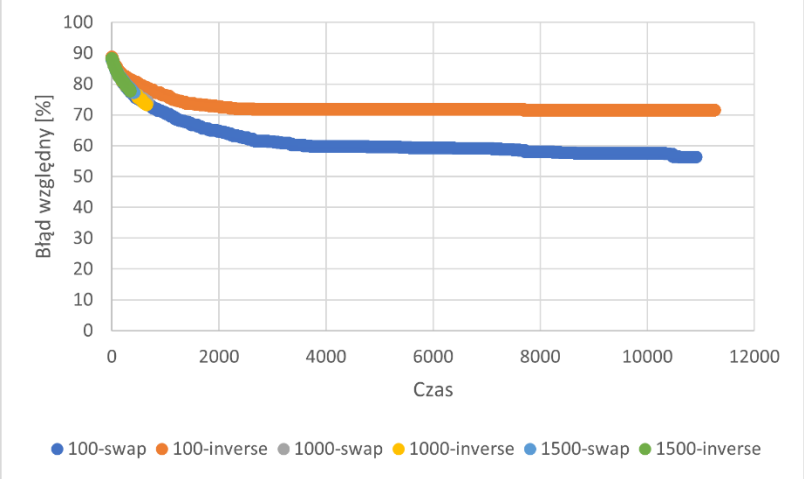
5.4. Porównanie algorytmu zależnego od wielkości populacji i rodzaju mutacji



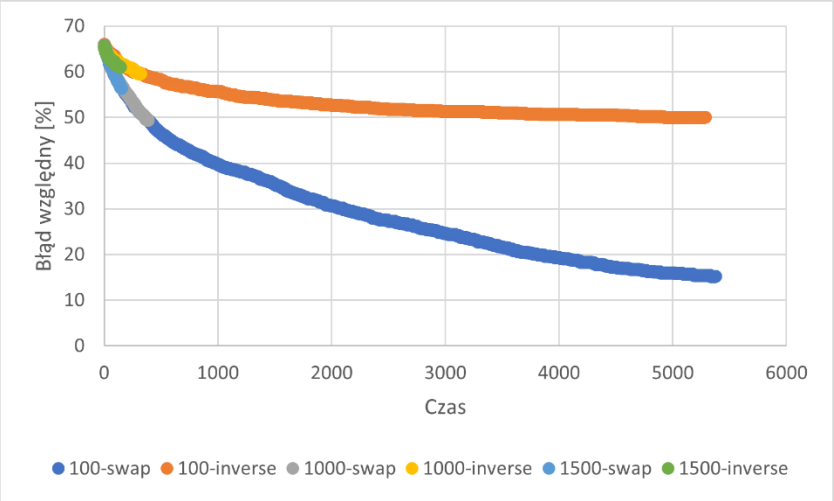
5.5. Wykres błędu względnego w funkcji czasu dla pliku ftv47.atsp – zmiana populacji



5.6. Wykres błędu względnego w funkcji czasu dla pliku ftv170.atsp – zmiana populacji



5.7. Wykres błędu względnego w funkcji czasu dla pliku rbg403.atsp – zmiana populacji



5.8. Ftv47 – zmiana współczynnika mutacji

Sposób mutacji	Współczynnik mutacji	Kryterium stopu [s]	Średni koszt	Średni błąd [%]
Swap	0.02	120	2055	13.5766
	0.05		2185	18.7185
	0.1		2320	23.4483
Inverse	0.02		2666	33.3833
	0.05		2452	27.5693
	0.1		2357	24.65

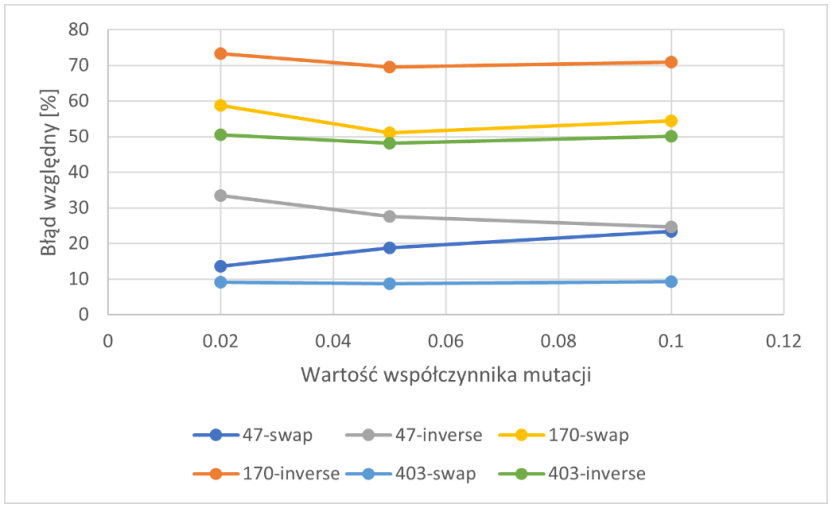
5.9. Ftv170 – zmiana współczynnika mutacji

Sposób mutacji	Współczynnik mutacji	Kryterium stopu [s]	Średni koszt	Średni błąd [%]
Swap	0.02	120	6679	58.7513
	0.05		5631	51.0744
	0.1		6043	54.4101
Inverse	0.02		10324	73.3146
	0.05		9026	69.4771
	0.1		9485	70.9541

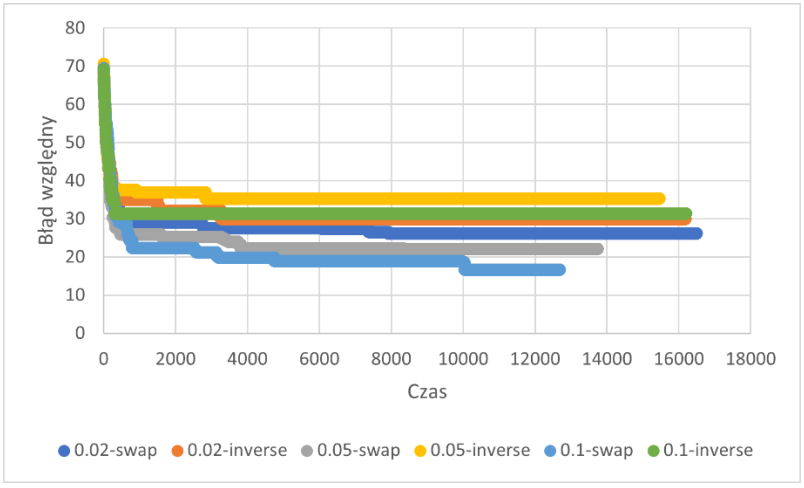
5.10. Rbg403 – zmiana współczynnika mutacji

Sposób mutacji	Współczynnik mutacji	Kryterium stopu [s]	Średni koszt	Średni błąd [%]
Swap	0.02	120	2714	9.17465
	0.05		2701	8.7375
	0.1		2719	9.22678
Inverse	0.02		4983	50.5318
	0.05		4746	48.0615
	0.1		4937	50.0709

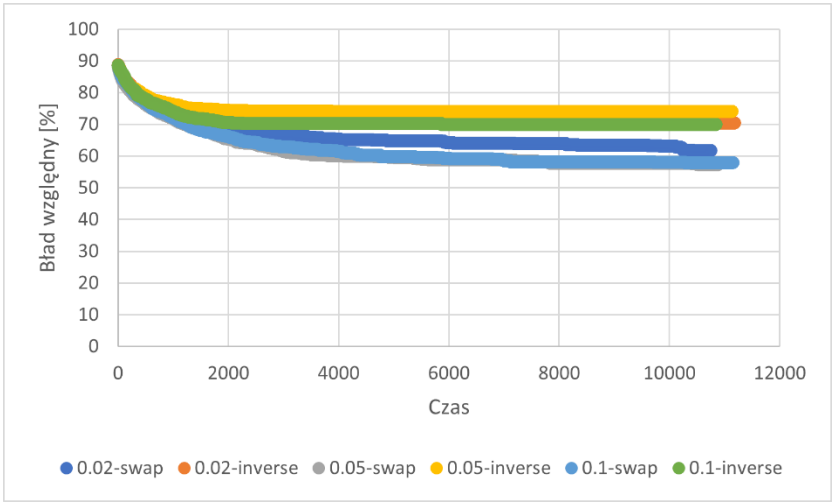
5.11. Porównanie algorytmu zależnego od współczynnika i rodzaju mutacji



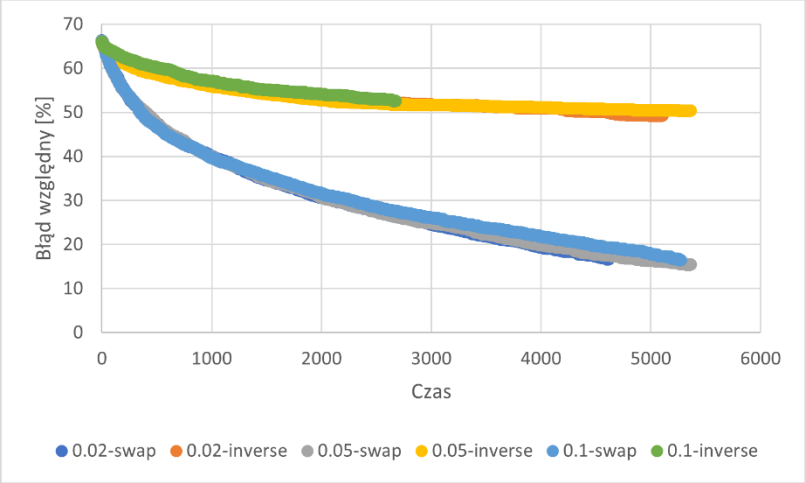
5.12. Wykres błędu względnego w funkcji czasu dla pliku ftv47.atsp – zmiana współczynnika mutacji



5.13. Wykres błędu względnego w funkcji czasu dla pliku ftv170.atsp – zmiana współczynnika mutacji



5.14. Wykres błędu względnego w funkcji czasu dla pliku rbg403.atsp – zmiana współczynnika mutacji



6. Najlepsze uzyskane wyniki dla GA

- ftv47.atsp

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

Koszt: 2041

- ftv170.atsp

152 -> 134 -> 118 -> 120 -> 121 -> 128 -> 132 -> 133 -> 6 -> 142 -> 143 -> 144 -> 141 -> 131 -> 127 -> 116 -> 119 -> 146 -> 145 -> 149 -> 161 -> 151 -> 13 -> 20 -> 46 -> 47 -> 48 -> 51 -> 50 -> 21 -> 29 -> 28 -> 27 -> 26 -> 15 -> 159 -> 16 -> 17 -> 68 -> 67 -> 167 -> 70 -> 87 -> 85 -> 73 -> 77 -> 1 -> 2 -> 4 -> 5 -> 169 -> 52 -> 54 -> 53 -> 43 -> 44 -> 45 -> 40 -> 156 -> 39 -> 38 -> 78 -> 82 -> 79 -> 80 -> 81 -> 7 -> 8 -> 14 -> 37 -> 75 -> 11 -> 12 -> 18 -> 170 -> 0 -> 110 -> 83 -> 71 -> 168 -> 72 -> 66 -> 65 -> 88 -> 153 -> 154 -> 89 -> 90 -> 91 -> 96 -> 97 -> 165 -> 163 -> 122 -> 124 -> 129 -> 111 -> 3 -> 9 -> 76 -> 19 -> 158 -> 36 -> 157 -> 33 -> 34 -> 35 -> 31 -> 32 -> 49 -> 112 -> 104 -> 99 -> 98 -> 95 -> 92 -> 166 -> 107 -> 106 -> 105 -> 126 -> 125 -> 136 -> 137 -> 138 -> 147 -> 148 -> 30 -> 41 -> 155 -> 42 -> 55 -> 58 -> 59 -> 60 -> 86 -> 93 -> 100 -> 101 -> 123 -> 162 -> 103 -> 114 -> 113 -> 115 -> 117 -> 102 -> 109 -> 108 -> 84 -> 69 -> 63 -> 64 -> 56 -> 57 -> 62 -> 61 -> 94 -> 164 -> 130 -> 135 -> 139 -> 140 -> 10 -> 74 -> 22 -> 23 -> 24 -> 25 -> 150 -> 160 -> 152

Koszt: 5739

- rbg403.atsp

288 -> 175 -> 291 -> 289 -> 244 -> 80 -> 10 -> 95 -> 55 -> 318 -> 271 -> 107 -> 61 -> 19 -> 377 -> 221 -> 199 -> 124 -> 1 -> 90 -> 72 -> 71 -> 176 -> 31 -> 344 -> 135 -> 270 -> 236 -> 227 -> 144 -> 253 -> 104 -> 9 -> 145 -> 366 -> 292 -> 216 -> 200 -> 266 -> 170 -> 334 -> 367 -> 76 -> 331 -> 378 -> 122 -> 265 -> 275 -> 325 -> 201 -> 129 -> 255 -> 13 -> 147 -> 220 -> 197 -> 242 -> 70 -> 264 -> 328 -> 83 -> 251 -> 280 -> 375 -> 371 -> 46 -> 195 -> 387 -> 349 -> 354 -> 342 -> 198 -> 50 -> 88 -> 62 -> 109 -> 164 -> 365 -> 245 -> 254 -> 223 -> 115 -> 91 -> 399 -> 224 -> 186 -> 53 -> 374 -> 234 -> 228 -> 337 -> 58 -> 277 -> 263 -> 180 -> 165 -> 193 -> 127 -> 192 -> 283 -> 8 -> 6 -> 160 -> 15 -> 68 -> 126 -> 119 -> 214 -> 301 -> 140 ->

57 -> 111 -> 369 -> 208 -> 276 -> 298 -> 320 -> 169 -> 27 -> 102 -> 209 -> 41 -> 338 -> 92 -> 347 -> 341 -> 178 -> 138 -> 297 -> 149 -> 146 -> 306 -> 73 -> 237 -> 231 -> 12 -> 252 -> 191 -> 114 -> 96 -> 94 -> 308 -> 5 -> 211 -> 44 -> 332 -> 37 -> 217 -> 49 -> 296 -> 206 -> 30 -> 312 -> 35 -> 141 -> 392 -> 120 -> 116 -> 38 -> 98 -> 133 -> 339 -> 132 -> 212 -> 130 -> 187 -> 139 -> 304 -> 394 -> 225 -> 372 -> 278 -> 18 -> 397 -> 260 -> 177 -> 248 -> 294 -> 257 -> 125 -> 74 -> 168 -> 85 -> 336 -> 321 -> 315 -> 167 -> 36 -> 172 -> 26 -> 158 -> 110 -> 326 -> 154 -> 262 -> 261 -> 78 -> 382 -> 243 -> 77 -> 156 -> 258 -> 246 -> 241 -> 112 -> 82 -> 249 -> 302 -> 89 -> 40 -> 39 -> 103 -> 311 -> 163 -> 25 -> 401 -> 29 -> 213 -> 28 -> 402 -> 287 -> 350 -> 159 -> 20 -> 128 -> 153 -> 75 -> 182 -> 42 -> 162 -> 48 -> 2 -> 205 -> 379 -> 202 -> 97 -> 285 -> 282 -> 317 -> 196 -> 0 -> 174 -> 185 -> 183 -> 17 -> 279 -> 151 -> 101 -> 33 -> 376 -> 106 -> 359 -> 268 -> 16 -> 47 -> 113 -> 395 -> 190 -> 100 -> 388 -> 348 -> 343 -> 81 -> 22 -> 21 -> 179 -> 368 -> 316 -> 351 -> 272 -> 356 -> 330 -> 290 -> 87 -> 56 -> 235 -> 121 -> 105 -> 391 -> 142 -> 335 -> 284 -> 329 -> 313 -> 7 -> 136 -> 386 -> 305 -> 215 -> 52 -> 286 -> 60 -> 99 -> 389 -> 239 -> 117 -> 157 -> 269 -> 222 -> 250 -> 361 -> 51 -> 123 -> 24 -> 307 -> 323 -> 309 -> 230 -> 226 -> 390 -> 150 -> 203 -> 295 -> 65 -> 314 -> 59 -> 373 -> 267 -> 327 -> 360 -> 353 -> 352 -> 385 -> 300 -> 108 -> 256 -> 247 -> 362 -> 23 -> 14 -> 69 -> 194 -> 324 -> 207 -> 45 -> 340 -> 240-> 219 -> 171 -> 63 -> 93 -> 84 -> 370 -> 358 -> 281 -> 43 -> 34 -> 364 -> 303 -> 398 -> 166 -> 67 -> 86 -> 11 -> 363 -> 204 -> 161 -> 148 -> 210 -> 232 -> 118 -> 79 -> 333 -> 181 -> 346 -> 189 -> 188 -> 299 -> 259 -> 345 -> 173 -> 319 -> 137 -> 54 -> 155 -> 143 -> 66 -> 273 -> 184 -> 4 -> 32 -> 274 -> 357 -> 322 -> 396 -> 293 -> 131 -> 355 -> 238 -> 229 ->393 -> 233 -> 380 -> 152 -> 310 -> 400 -> 64 -> 3 -> 381 -> 218 -> 384 -> 383 -> 134 -> 288

Koszt: 2691

7. Najlepsze uzyskane wyniki dla TS

- ftv47.atsp

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

Koszt: 1874

- ftv170.atsp

24 -> 25 -> 150 -> 160 -> 151 -> 152 -> 142 -> 141 -> 134 -> 131 -> 113 -> 164 -> 127 -> 126 -> 125 -> 119 -> 120 -> 121 -> 122 -> 123 -> 162 -> 101 -> 103 -> 117 -> 118 -> 124 -> 129 -> 128 -> 130 -> 135 -> 138 -> 139 -> 140 -> 6 -> 7 -> 8 -> 9 -> 10 -> 76 -> 74 -> 75 -> 11 -> 12 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23 -> 26 -> 27 -> 28 -> 30 -> 31 -> 33 -> 35 -> 34 -> 156 -> 40 -> 39 -> 38 -> 37 -> 49 -> 170 -> 73 -> 77 -> 2 -> 1 -> 0 -> 81 -> 80 -> 79 -> 82 -> 78 -> 72 -> 71 -> 60 -> 50 -> 51 -> 52 -> 53 -> 43 -> 55 -> 54 -> 58 -> 59 -> 61 -> 68 -> 67 -> 167 -> 70 -> 87 -> 85 -> 86 -> 83 -> 84-> 69 -> 66 -> 63 -> 64 -> 56 -> 57 -> 62 -> 65 -> 88 -> 153 -> 154 -> 89 -> 90 -> 91 -> 94 -> 96 -> 97 -> 99 -> 98 -> 95 -> 92 -> 93 -> 166 -> 108 -> 107 -> 106 -> 105 -> 165 -> 163 -> 100 -> 102 -> 104 -> 110 -> 109 -> 114 -> 115 -> 116 -> 136 -> 137 -> 147 -> 148 -> 149 -> 161 -> 14 -> 13 -> 17 -> 32 -> 158 -> 36 -> 157 -> 41 -> 155 -> 42 -> 45 -> 44 -> 46 -> 47 -> 48 -> 168 -> 3 -> 4 -> 5 -> 133 -> 169 -> 111 -> 112 -> 132 -> 144 -> 143 -> 146 -> 145 -> 15 -> 159 -> 16 -> 29 -> 24

Koszt: 3379

- rbg403.atsp

236 -> 220 -> 197 -> 23 -> 14 -> 62 -> 13 -> 205 -> 379 -> 372 -> 28 -> 251 -> 274 -> 293 -> 283 -> 8 -> 168 -> 135 -> 270 -> 19 -> 18 -> 402 -> 287 -> 350 -> 93 -> 203 -> 179 -> 368 -> 397 -> 394 -> 393 -> 335 -> 126 -> 55 -> 164 -> 3 -> 2 -> 61 -> 107 -> 386 -> 47 -> 112 -> 328 -> 257 -> 160 -> 365 -> 142 -> 150 -> 9 -> 273 -> 272 -> 278 -> 86 -

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

Koszt: 2349

8. Wnioski

Początkowo należało znaleźć optymalną wartość populacji początkowej. Po przeprowadzeniu eksperymentu widać, że wraz ze wzrostem tej wartości wzrasta również błąd względny. Wyjątkiem jest jedynie plik ftv47.atsp. Do dalszego testowania przyjęłam więc wartość 100.

Dobrze dostosowując współczynnik mutacji algorytmu można poprawić jego wynik. W ogólnym przypadku najlepsza okazała się wartość 0.05, gdyż to zazwyczaj dla niej wychodził najmniejszy błąd względny. Z wykresów można również zauważyć, że metoda mutacji swap jest wydajniejsza od inverse i przynosi lepsze wyniki.

Porównując algorytmy Tabu Search i Genetic Algoritm widać, że ten pierwszy daje korzystniejsze rezultaty. Oba algorytmy były testowane dla czasu 120s, jednak TS niekiedy znacznie wyprzedza GA.