Politechnika
Wrocławska,
Wydział Informatyki
i Telekomunikacji
Semestr V Rok III

Zadanie projektowe nr 2

Implementacja i analiza efektywności algorytmu Tabu Search i Symulowanego Wyżarzania dla problemu komiwojażera

Wykonał:
Patryk Ignasiak 263889
Gr. Piątek 11:15
Prowadzący:
dr inż. Jarosław Mierzwa
Przedmiot:
Projektowanie efektywnych
algorytmów – Projekt

Wrocław, 8 Grudnia 2023

Spis treści

1 Wstęp

Podczas realizacji drugiego projektu miało miejsce wdrożenie oraz analiza skuteczności algorytmów metaheurystycznych, takich jak:

- Tabu Search
- Symulowane Wyżarzanie

Badania zostały przeprowadzone w kontekście asymetrycznego problemu komiwojażera (ATSP). Ten specyficzny problem polega na znalezieniu minimalnego cyklu Hamiltona w grafie pełnym, skierowanym, gdzie wagi krawędzi są nieujemne. Minimalny cykl Hamiltona osiągany jest, gdy suma wag krawędzi w cyklu jest możliwie najmniejsza. Rozwiązanie tego problemu jest wyjątkowo trudne, zwłaszcza ze względu na liczbę krawędzi w grafie pełnym, skierowanym, która wynosi V(V-1). Dodatkowo, liczba możliwych kombinacji tworzących cykl Hamiltona wynosi (V-1)!. Analiza efektywności algorytmów metaheurystycznych w tym kontekście ma kluczowe znaczenie, ponieważ dostarcza alternatywne rozwiązania w sytuacjach, gdzie metody klasyczne mogą być niewystarczające.

Algorytmy metaheurystyczne są technikami optymalizacyjnymi stosowanymi do rozwiązywania trudnych problemów, które przekraczają możliwości klasycznych metod. Omawiane algorytmy, w tym Tabu Search i Symulowane Wyżarzanie, posiadają zdolność radzenia sobie z dużymi problemami optymalizacyjnymi, choć ich działanie jest obarczone pewnym stopniem losowości.

Działanie tych algorytmów opiera się na iteracyjnym przeszukiwaniu przestrzeni rozwiązań w poszukiwaniu optymalnego lub zbliżonego do optymalnego wyniku. Z racji na charakterystykę problemów optymalizacyjnych, algorytmy metaheurystyczne nie gwarantują znalezienia rozwiązania dokładnego, ale zapewniają podejście elastyczne i adaptacyjne do złożonych sytuacji.

2 Tabu Search

Przeszukiwanie z tabu jest metaheurystyką służącą do rozwiązywania problemów optymalizacyjnych. Opiera się ona na iteracyjnym przeszukiwaniu przestrzeni rozwiązań, korzystając z sąsiedztwa pewnych elementów tej przestrzeni oraz pamiętając ostatnie ruchy. Algorytm kontynuuje przeszukiwanie, dopóki nie zostanie spełniony warunek końcowy. Ruchy są zapisywane jako atrybuty przejścia, czyli parametry jednoznacznie opisujące wykonany ruch, na liście tabu. Obecność danego ruchu na liście tabu jest tymczasowa, zazwyczaj ograniczona do określonej liczby iteracji od ostatniego użycia, co oznacza, że dany ruch nie może być wykonany przez pewien czas, chyba że spełnia on kryterium aspiracji. Lista tabu ma za zadanie ograniczyć prawdopodobieństwo zapętleń w przeszukiwaniu i zmusić algorytm do badania nowych obszarów przestrzeni rozwiązań.

Przeszukiwanie z tabu wprowadza pewne opóźnienie w procesie zbiegania do rozwiązania optymalnego, ponieważ wymusza zamianę obecnego rozwiązania z najlepszym dostępnym sąsiadem, niezależnie od tego, czy sąsiad jest lepszy czy gorszy. Jednak pozytywnym aspektem tego podejścia jest zdolność do szybkiego wyjścia z obszaru przyciągania ekstremum lokalnego, które niekoniecznie musi być globalnie optymalne, czasami będąc znacznie oddalonym od optymalnego.

Warunkiem końcowym przeszukiwania z tabu może być określona liczba iteracji, brak zmiany globalnie najlepszego rozwiązania przez określoną liczbę kolejnych iteracji, co może stanowić motywację do wygenerowania nowego rozwiązania startowego i ponownego rozpoczęcia algorytmu z pewną wiedzą zdobytą wcześniej, lub osiągnięcie zadanej wartości funkcji celu.

2.1 Szczegółowy opis zastosowanego algorytmu

Krok 1 : Inicjalizacja algorytmu

- Wywołanie funkcji TabuSearch() z parametrami takimi jak macierz sąsiedztwa, rozmiar macierzy, warunek stopu, czyli po ilu sekundach algorytm powinien zakończyć działanie.
- Inicjalizacja ścieżki (metoda zachłanna, ale losowy wierzchołek początkowy)
- Obliczenie początkowego kosztu

Krok 2: Główna pętla

- Generowanie sąsiedztwa w zależności od wyboru definicji
- Dywersyfikacja jeśli wymagana
- Wybór najlepszego sąsiedztwa, które nie znajduje się na liście tabu
- Aktualizacja najlepszej ścieżki jeśli znaleziono lepszą
- Aktualizacja listy tabu
- Zwiększenie licznika dywersyfikacji jeśli nie poprawiono ścieżki

Krok 3: Zakończenie

- Warunkiem zakończenia jest przekroczenie kryterium stopu czyli jeśli czas trwania algorytmu jest większy niż kryterium stopu algorytm jest przerywany.

2.2 Kod algorytmu

Poniżej przedstawiam główny kod algorytmu 7Tabu Search, wykorzystane w nim funkcje zostaną omówione w punkcie 4 Omówienie pozostałych funkcji programu.

```
void TS::TabuSearch(vector<vector<int>> matrix, int size, int stop, int choice) {
	int diversification_threshold = 10; // maksymalna liczba do dywersyfikacji
	time_t startTime;

	startTime = time(NULL);
	srand(time(NULL));
	//inicjalizacja  ścieżek
	vector<int> path = generatePath(matrix, size);

	bestCost = calculateCost(matrix, size, path);
	int pathCost = bestCost;

	vector<vector<int>> tabu_list = { path };
	int diversification_count = 0;
	auto start = chrono::steady_clock::now();
	costs[0] = bestCost;
	int z = 1;
	vector<vector<int>> solution_neighborhood = getNeighborhoodSolutions(path);
	while (time(NULL) - startTime < stop) {
		if (choice == 2) {
```

```
                        vector<vector<int>> solution_neighborhood =
getNeighborhoodSolutions3Cities(path);
                }
                else if (choice == 3) {
                        vector<vector<int>> solution_neighborhood =
getNeighborhoodSolutionsInversion(path);
                }
                else {
                        vector<vector<int>> solution_neighborhood = getNeighborhoodSolutions(path);
                }

                // Dywersyfikacja
                if (diversification_count >= diversification_threshold) {
                        std::random_shuffle(solution_neighborhood.begin(), solution_neighborhood.end());
                        path = solution_neighborhood[0];
                        diversification_count = 0;
                }
                else {
                        path = solution_neighborhood[0];
                }
                pathCost = calculateCost(matrix, size, path);
                for (vector<int> candidate : solution_neighborhood) {
                        int candidateCost = calculateCost(matrix, size, candidate);
                        if (candidateCost < pathCost &&
                                std::find(tabu_list.begin(), tabu_list.end(), candidate) == tabu_list.end()) {
                                path = candidate;
                                pathCost = candidateCost;
                        }
                }
                //zamiana z najlepszym
                if (pathCost < bestCost) {
                        bestPath = path;
                        bestCost = pathCost;
                        auto end = chrono::steady_clock::now();
                        auto duration = end - start;

                        times[z] = chrono::duration_cast<chrono::microseconds>(duration).count();
                        costs[z] = pathCost;
                        z++;
                }

                tabu_list.push_back(path);
                if (tabu_list.size() > static_cast<size_t>(size)) {
                        tabu_list.erase(tabu_list.begin());
                }

                //Zwiększamy licznik do dywersyfikacji
                if (pathCost > bestCost) {
                        diversification_count++;
                }
                else {
                        diversification_count = 0;
                }
        }
}
```

2.3 Definicje sąsiedztwa

Podczas tworzenia algorytmu wykorzystałem trzy następujące definicje sąsiedztwa:

- Zamiana dwóch miast (swap)
- Zamiana pomiędzy trzema miastami (swap)
- Odwrócenie fragmentu ścieżki (reverse)

3 Symulowane wyżarzanie

Symulowane wyżarzanie to rodzaj algorytmu heurystycznego, który eksploruje przestrzeń alternatywnych rozwiązań problemu w poszukiwaniu najlepszych rezultatów. Metoda ta czerpie inspirację z procesu wyżarzania znanej z metalurgii, gdzie obróbka cieplna ma na celu uzyskanie pożądanych właściwości materiału.

Procedura wyżarzania w metalurgii polega na ogrzewaniu przedmiotu metalowego do określonej temperatury, utrzymaniu go w niej przez pewien czas, a następnie stopniowym schładzaniu. Ten proces ma na celu eliminację naprężeń, uzyskanie pożądanej struktury materiału oraz rekrystalizację. W zależności od potrzeb stosuje się różne parametry, takie jak temperatura, czas wyżarzania, czas schładzania itp.

W kontekście algorytmu symulowanego wyżarzania, inspiracja wyżarzaniem metalu przejawia się w podejściu do przeszukiwania przestrzeni rozwiązań. Rozkład Boltzmanna, opisujący prawdopodobieństwo stanu układu przy danej energii i temperaturze, jest wykorzystywany do opisu zachowań7 układów fizycznych. W kontekście algorytmu, prawdopodobieństwo akceptacji gorszych rozwiązań maleje wraz ze spadkiem temperatury, co pozwala na unikanie minimum lokalnych i poszukiwanie globalnych optimum.

Wykorzystanie analogii do wyżarzania w metalurgii oraz matematycznego modelu rozkładu Boltzmanna sprawia, że symulowane wyżarzanie stanowi efektywną metaheurystykę do rozwiązywania problemów optymalizacyjnych.

3.1 Szczegółowy opis zastosowanego algorytmu

Krok 1 : Inicjalizacja algorytmu

- Wywołanie funkcji z parametrami takimi jak macierz sąsiedztwa, rozmiar macierzy, temperatura początkowa, współczynnik schładzania oraz warunek stopu, czyli po ilu sekundach algorytm powinien zakończyć działanie.
- Inicjalizacja ścieżki (metoda zachłanna, ale losowy wierzchołek początkowy)
- Obliczenie początkowego kosztu

Krok 2: Główna pętla

- Generowanie sąsiedztwa poprzez zamianę dwóch losowych wierzchołków.
- Obliczenie kosztu dla nowej ścieżki.
- Akceptacja rozwiązania, czyli sprawdzenie czy jest lepsze od obecnego. Jeśli nie jest to sprawdzenie czy spełnia warunek:

(rand() / static_cast<double>(RAND_MAX)) < min(exp(-(currentCost - newCost) / T))

- Jeśli żaden z powyższych warunków nie zostanie spełniony to wracamy do poprzedniej ścieżki.
- A jeśli któryś był spełniony to sprawdzamy czy nowe rozwiązanie jest lepsze od najlepszego, jeśli tak to podmieniamy.
- Na koniec wykonujemy schładzanie według wybranego schematu.

Krok 3: Zakończenie

- Warunkiem zakończenia jest przekroczenie kryterium stopu czyli jeśli czas trwania algorytmu jest większy niż kryterium stopu algorytm jest przerywany.

3.2 Kod algorytmu

Postanowiłem przedstawić kod algorytmu tylko dla jednego sposobu schładzanie, gdyż różnica między nimi odności się tylko do jednej linijki w której jest schładzana temperatura.

```
void SA::annealingIns(vector<vector<int>> matrix, int size, int stop, double a, double T) {
	time_t startTime;

	startTime = time(NULL);
	srand(time(NULL));
	//inicjalizacja  ścieżek
	bestPath = new int[size];
	int* path = generatePath(matrix, size);

	bestCost = calculateCost(matrix, size, path);
	int temp;
	int currentCost = bestCost;
	int newCost;
	auto start = chrono::steady_clock::now();
	costs[0] = bestCost;
	int z = 1;
	while (time(NULL) - startTime < stop) {
		// Generowanie sąsiedztwa
		int v1 = rand() % size;
		int v2 = rand() % size;
		while (v1 == v2) {
			v2 = rand() % size;
		}
		// Zamiana wierzchołków
		temp = path[v1];
		path[v1] = path[v2];
		path[v2] = temp;

		newCost = calculateCost(matrix, size, path);
		//Akceptacja rozwiązania
		if (newCost < currentCost || (rand() / static_cast<double>(RAND_MAX)) < min(exp(-
(currentCost - newCost) / T))) {
			currentCost = newCost;

			if (currentCost < bestCost) {
				auto end = chrono::steady_clock::now();
				auto duration = end - start;

				times[z] =
chrono::duration_cast<chrono::microseconds>(duration).count();
```

```
                                costs[z] = currentCost;
                                z++;
                                for (int i = 0; i < size; i++) {
                                        bestPath[i] = path[i];
                                }
                                bestCost = currentCost;
                        }
                }
                else {
                        temp = path[v1];
                        path[v1] = path[v2];
                        path[v2] = temp;
                }
                //Schładzanie
                T *= a;
        }
        Tend = T;
}
```

3.3 Schematy schładzania

- Schemat z instrukcji:

$$T(i+1) = a * T(i)$$

- Schemat liniowy:

$$T(i+1) = \frac{T(i)}{a+bk}$$

- Schemat logarytmiczny:

$$T(i+1) = \frac{T(i)}{a+blogk}$$

Gdzie: 0 < a < 1, b = 0.05, k = nr. Iteracji pętli głównej algorytmu

4 Omówienie pozostałych funkcjonalności

4.1 calculateCost()

Funkcja ta jest obecna zarówno w symulowanym wyżarzaniu jak i tabu search, służy ona do obliczenia wartości danej ścieżki. Jedyna różnica jest taka, że ścieżka w tabu search przekazywana jest jako vector<int>.

```
int SA::calculateCost(vector<vector<int>> matrix, int size, int* path) {
        int sum = 0;
        for (int i = 0; i < size-1; i++) {
                sum += matrix[path[i]][path[i + 1]];
        }
        sum += matrix[path[size -1]][path[0]];
        return sum;
}
```

4.2 generatePath()

Funkcja generuje ścieżkę początkową za pomocą metody zachłannej ale z losowym wierzchołkiem startowym. Również jak poprzednia omawiana funkcja jest używana w obu algorytmach jednak tam ścieżka jest zapisywana jako vector<int>.

```
int* SA::generatePath(vector<vector<int>> matrix, int size) {
	int* path = new int[size];
	bool* visited = new bool[size] {false}; // Tablica do śledzenia, czy miasto zostało odwiedzone
	// Losowy wybór startowego miasta
	int firstCity = rand() % size;
	path[0] = firstCity;
	bestPath[0] = firstCity;
	visited[firstCity] = true;

	// Wybór kolejnych miast
	for (int i = 1; i < size; ++i) {
		int lastCity = path[i - 1];
		int nextCity = -1;
		int bCost = INT32_MAX;

		// Wybór najbliższego miasta spośród nieodwiedzonych
		for (int j = 0; j < size; ++j) {
			if (!visited[j] && matrix[lastCity][j] < bCost) {
				bCost = matrix[lastCity][j];
				nextCity = j;
			}
		}
		// Zaznaczenie miasta jako odwiedzonego
		visited[nextCity] = true;
		path[i] = nextCity;
		bestPath[i] = nextCity;
		bestCost += matrix[lastCity][nextCity];
	}
	bestCost += matrix[path[size-1]][0];
	return path;
}
```

4.3 swapPositions()

Funkcja zamienia miejscami dwa miasta w ścieżce.

```
std::vector<int> TS::swapPositions(const std::vector<int>& tour, int city1, int city2) {
	std::vector<int> tourCopy = tour;
	std::swap(tourCopy[city1], tourCopy[city2]);
	return tourCopy;
}
```

4.4 getNeighborhoodSolutions()

Funkcja generuje sąsiedztwa przy pomocy zamiany dwóch miast.

```
std::vector<std::vector<int>> TS::getNeighborhoodSolutions( std::vector<int>& tour) {
	std::vector<std::vector<int>> neighborhoodSolutions;

	for (size_t i = 1; i < tour.size() - 1; ++i) {
		for (size_t j = 1; j < tour.size() - 1; ++j) {
```

```
                        if (i == j) {
                                continue;
                        }
                        neighborhoodSolutions.push_back(swapPositions(tour, i, j));
                }
        }
        return neighborhoodSolutions;
}
```

4.5 swapPositions3Cities()

Funkcja służy do zamiany trzech miast.

```
std::vector<int> TS::swapPositions3Cities(const std::vector<int>& tour, int city1, int city2, int city3) {
        std::vector<int> tourCopy = tour;
        std::swap(tourCopy[city1], tourCopy[city2]);
        std::swap(tourCopy[city1], tourCopy[city3]);
        return tourCopy;
}
```

4.6 getNeighborhoodSolutions3Cities()

Funkcja generuje sąsiedztwa przy pomocy zamiany trzech miast.

```
std::vector<std::vector<int>> TS::getNeighborhoodSolutions3Cities(std::vector<int>& tour) {
        std::vector<std::vector<int>> neighborhoodSolutions;

        for (size_t i = 1; i < tour.size() - 1; ++i) {
                for (size_t j = i+1; j < tour.size() - 1; ++j) {
                        for (size_t k = j+1; k < tour.size() - 1; ++k) {
                                if (i == j == k || i == j || i == k || j == k) {
                                        continue;
                                }
                                neighborhoodSolutions.push_back(swapPositions3Cities(tour, i, j, k));
                        }
                }
        }
        return neighborhoodSolutions;
}
```

4.7 inversion()

Funkcja odwraca fragment ścieżki ograniczony dwoma miastami.

```
std::vector<int> TS::inversion(const std::vector<int>& tour, int city1, int city2) {
        std::vector<int> tourCopy = tour;
        std::reverse(tourCopy.begin() + city1, tourCopy.begin() + city2 + 1);
        return tourCopy;
}
```

4.8 getNeighborhoodSolutionsInversion()

Funkcja generuje sąsiedztwa przy pomocy odwracania fragmentów ścieżki.

```
std::vector<std::vector<int>> TS::getNeighborhoodSolutionsInversion(std::vector<int>& tour) {
        std::vector<std::vector<int>> neighborhoodSolutions;

        for (size_t i = 1; i < tour.size() - 1; ++i) {
                for (size_t j = i + 1; j < tour.size() - 1; ++j) {
                        if (i == j) {
                                continue;
```

```
                    }
                    neighborhoodSolutions.push_back(inversion(tour, i, j));
                }
            }
            return neighborhoodSolutions;
}
```

4.9 readFile()

Funkcja odpowiada za wczytanie pliku .xml z danymi do testów.

```
void Data::readFile(string filename) {
    // Otwórz plik XML
    std::ifstream file(filename);

    if (!file.is_open()) {
        std::cerr << "Nie można otworzyć pliku XML." << std::endl;
        return;
    }
    //policz wierzchołki
    std::string line;
    size = 0;
    while (std::getline(file, line)) {
        if (line.find("<vertex>") != std::string::npos) {
            size++;
        }
    }
    file.close();
   //utwórz macierz
    std::vector<std::vector<int>>tmpMatrix(size, std::vector<int>(size, INT32_MAX));
    this->matrix = tmpMatrix;
    //załaduj dane
    std::ifstream f(filename);
    int numOfRow = -1;
    while (std::getline(f, line)) {
        if (line.find("<vertex>") != std::string::npos) {
            numOfRow++;
        }
        if (line.find("<edge") != std::string::npos) {
            double edge;
            double power;
            int index;
            size_t startPos = line.find("\"") + 1;
            size_t endPos = line.find("e", startPos);
            std::string numberStr = line.substr(startPos, endPos - startPos);
            std::istringstream iss(numberStr);
            iss >> edge;
            startPos = line.find("+") + 1;
            endPos = line.find("\"", startPos);
            numberStr = line.substr(startPos, endPos - startPos);
            numberStr = line.substr(startPos, endPos - startPos);
            std::istringstream iss2(numberStr);
            iss2 >> power;
            edge = edge * pow(10, power);
            startPos = line.find(">") + 1;
            endPos = line.find("<", startPos);
```

```
            numberStr = line.substr(startPos, endPos - startPos);
            numberStr = line.substr(startPos, endPos - startPos);
            std::istringstream iss3(numberStr);
            iss3 >> index;
            matrix[numOfRow][index] = edge;
        }
    }

    f.close();
    cout << "Dane zostały wczytane!!!" << endl;
}
```

4.10 savePath() i savePathTS()

Obie funkcje służą do zapisania ścieżki do pliku. Różnicą jest sposób przekazywania ścieżki do funkcji.

```
void Data::savePath(int* path, string alg, string filename) {
    if (size == 1) {
        cout << "Brak danych do zapisania!!!" << endl;
        return;
    }
    //Tworzymy plik i zapisujemy rozmiar
    size_t num = filename.find(".");
    string name = alg + filename.substr(0, num) + ".txt";
    ofstream file(name);

    file << size << endl;

    file.close();
    //otwieramy plik do nadpisania
    //i zapisujemy  dane
    file.open(name, ios_base::app);
    for (int i = 0; i < size; i++) {
        file << path[i];
        file << endl;
    }
    file.close();
}void Data::savePathTS(std::vector<int> path, string alg, string filename) {
    if (size == 1) {
        cout << "Brak danych do zapisania!!!" << endl;
        return;
    }
    //Tworzymy plik i zapisujemy rozmiar
    size_t num = filename.find(".");
    string name = alg + filename.substr(0, num) + ".txt";
    ofstream file(name);

    file << size << endl;

    file.close();
    //otwieramy plik do nadpisania
    //i zapisujemy dane
    file.open(name, ios_base::app);

    for (int i = 0; i < size; i++) {
        file << path[i];
        file << endl;
```

```
    }
    file.close();
}
```

4.11 loadPath()

Funkcja służy do wczytania ścieżki z pliku i obliczenia długości drogi

```
void Data::loadPath(string filename) {
    // Otwórz plik
    std::ifstream file(filename);

    if (!file.is_open()) {
        std::cerr << "Nie można otworzyć pliku." << std::endl;
    }

    std::string line;

    bool wasFirst = false;
    cout << "Ścieżka: ";
    int first = -1;
    int prev, sum = 0;
    while (std::getline(file, line)) {
        if (wasFirst) {
            cout << line << " | ";
            int v;
            std::istringstream iss(line);
            iss >> v;
            if (first == -1) {
                first = v;
            }
            else {
                sum += matrix[prev][v];
            }
            prev = v;
        }
        wasFirst = true;
    }
    sum += matrix[prev][first];
    cout << "\nDroga: " << sum;
}
```

4.12 saveTestResult()

Funkcja zapisuje wyniki testów do plików .txt

```
void Data::saveTestResult(long long times[100], int costs[100], double a, int type) {
    number++;
    ofstream file(to_string(type)+"rezult"+ to_string(a) + ".txt");

    for (int i = 0; i < 100; i++) {
        if (costs[i] != 0) {
            file << times[i];
            file << ",";
            file << costs[i];
            file << endl;
        }
        else {
```

```
            break;
        }
    }
    file.close();
}
```

5 Omówienie wyników testów

5.1 Wstęp – omówienie sposobu testowania

Dla każdego z algorytmów było wykonywane 10 powtórzeń dla każdego przypadku. Jako przypadek można rozumieć:

- Dla symulowanego wyżarzania:
  - Współczynnik a - {0.85, 0.88, 0.92, 0.95, 0.999, 0.999999
  - Schemat schładzania
- Dla tabu search:
  - Definicja sąsiedztwa
- Dla obu:
  - Rozmiar pliku

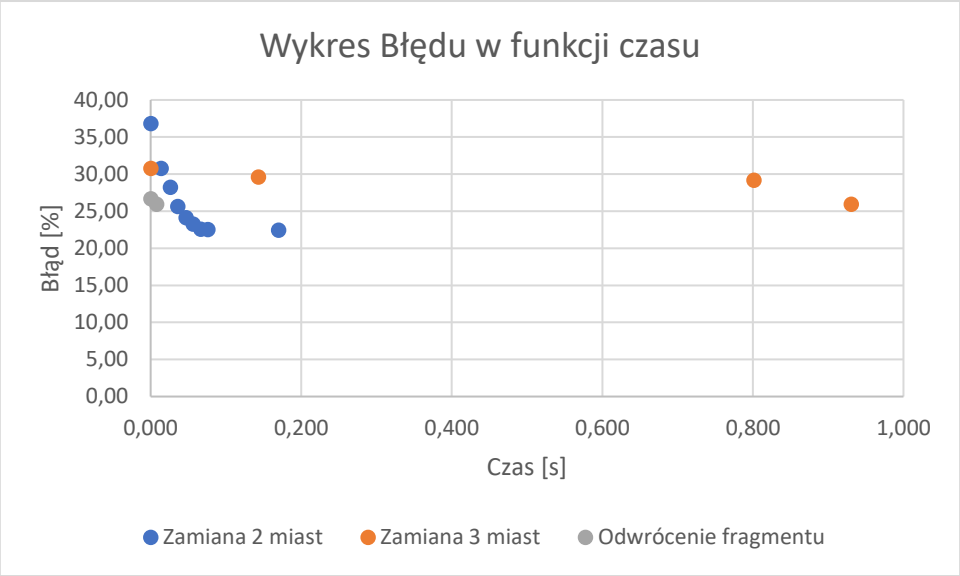
Warunek stopu w zależności od pliku:

- ftv55.xml – 30 sekund
- ftv170.xml – 60 sekund
- rbg358.xml – 90 sekund

5.2 Tabu search

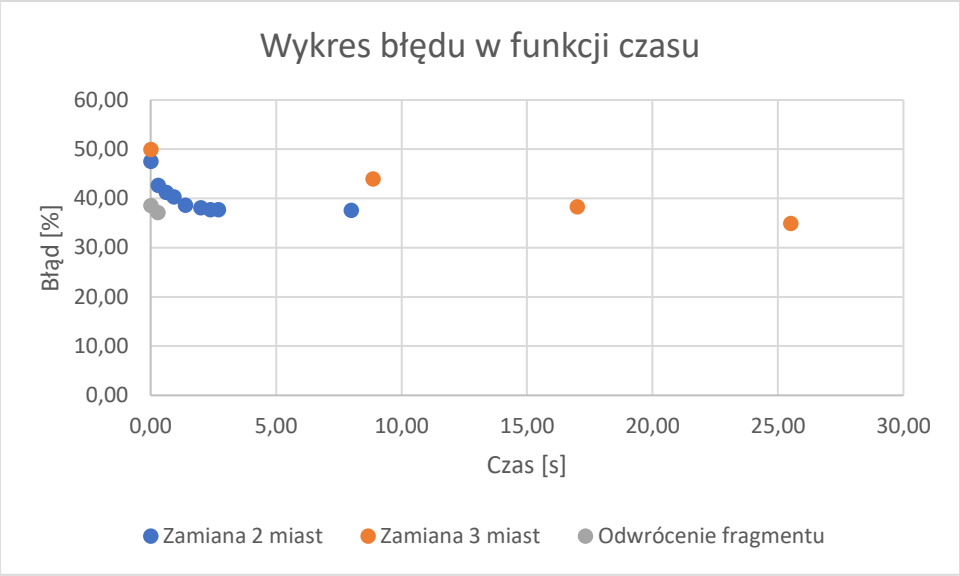
5.2.1 Plik ftv55.xml

Czas	Zamiana 2 miast	Zamiana 3 miast	Odwrócenie fragmentu
0	36,83883		
0		30,80274	
0			26,69571
0,007755			25,94897
0,013996	30,80274		
0,026302	28,2514		
0,035824	25,63783		
0,047114	24,14437		
0,05629	23,27318		
0,066534	22,58867		
0,075998	22,52645		
0,143253		29,62041	
0,169672	22,46422		
0,80105		29,18482	
0,930562		25,94897	



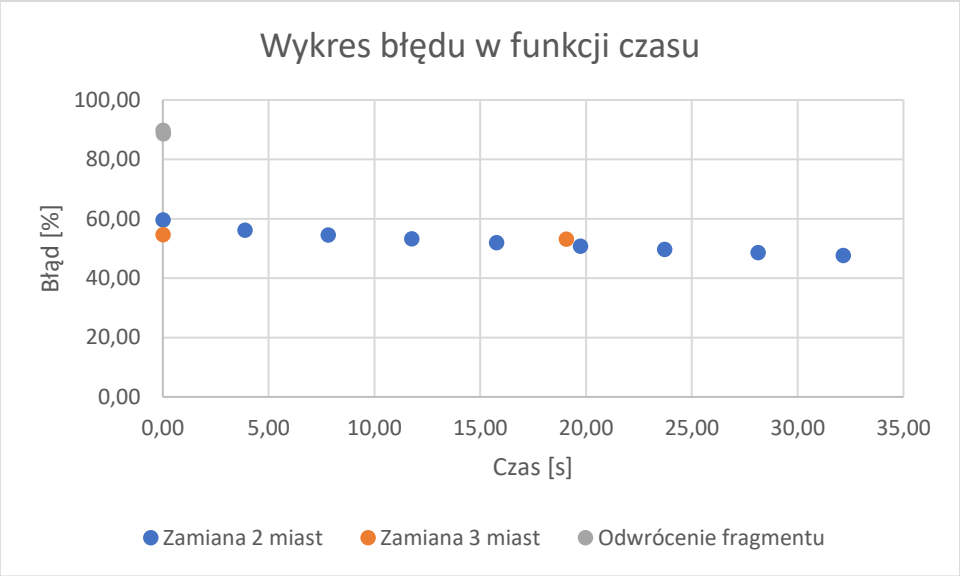
5.2.2 Ftv170.xml

Czas	Zamiana 2 miast	Zamiana 3 miast	Odwrócenie fragmentu
0,00	47,55		
0,00		49,98	
0,00			38,58
0,29			37,13
0,30	42,65		
0,62	41,27		
0,92	40,36		
1,38	38,66		
2,00	38,11		
2,38	37,75		
2,69	37,71		
7,99	37,60		
8,86		43,96	
17,00		38,29	
25,51		34,92	



5.2.3 Rbg358.xml

Czas	Zamiana 2 miast	Zamiana 3 miast	Odwrócenie fragmentu
0,00	59,67		
0,00		54,69	
0,00			89,85
0,02			88,65
3,88	56,15		
7,82	54,60		
11,76	53,31		
15,76	52,02		
19,07		53,14	
19,73	50,82		
23,71	49,70		
28,13	48,67		
32,15	47,64		



5.2.4 Wnioski

Algorytm tabu search jest algorytmem, który dzięki dywersyfikacji nie wpada w lokalne minima, tylko ma spore szanse na znalezienie globalnego. Jak możemy zauważyć na wykresach jeśli przez dłuższy czas nie ma poprawy to następuje dywersyfikacja i znajduje nowe minimum. Dodatkowo jak możemy zauważyć na powyższych wykresach najlepszą, dla tych przypadków, okazała się definicja sąsiedztwa z zamianą dwóch wierzchołków.

5.3 Symulowane wyżarzanie

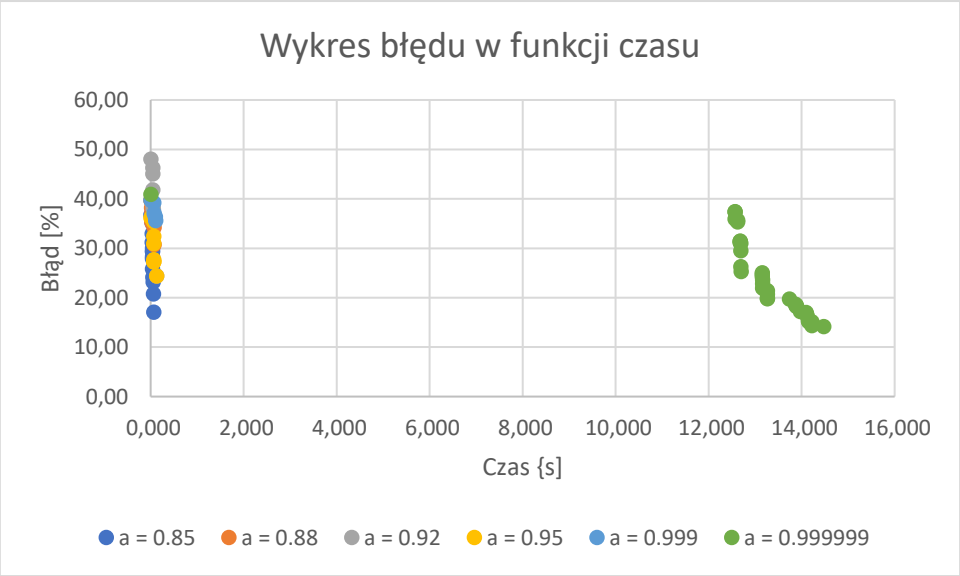
5.3.1 Plik ftv55.xml

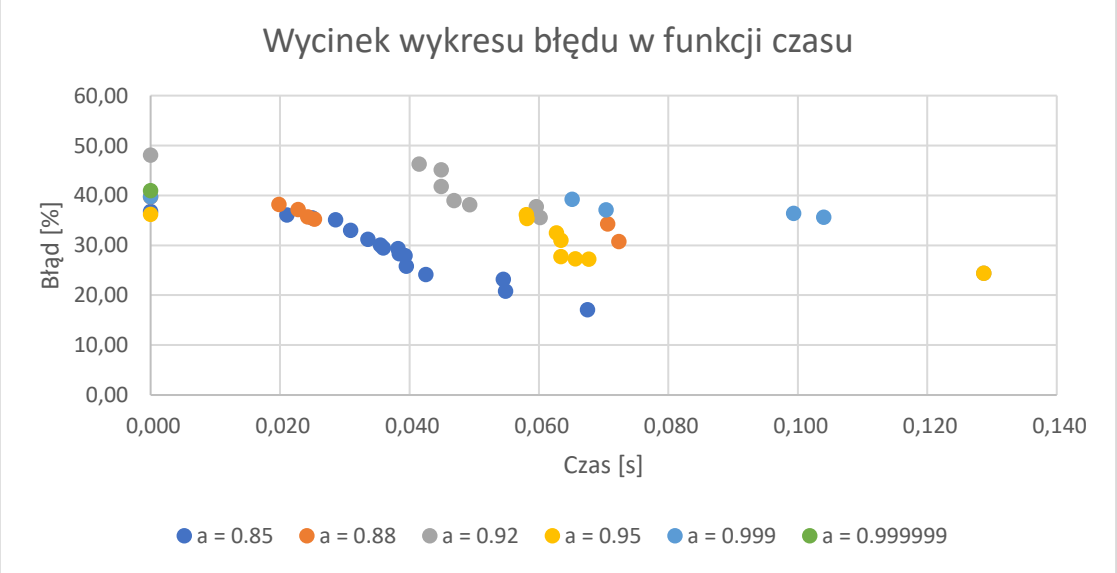
a) Schładzanie z instrukcji

Czas [s]	a = 0.85	a = 0.88	a = 0.92	a = 0.95	a = 0.999	a = 0.999999
0,000	36,75	39,74	48,07	36,19	39,74	40,92
0,020		38,18				
0,021	36,07					
0,023		37,19				
0,024		35,70				
0,025	35,51					
0,025		35,26				
0,029	35,07					
0,031	32,96					
0,034	31,22					
0,036	30,04					
0,036	29,48					
0,038	29,35					
0,038	28,30					
0,039	27,92					
0,039	25,81					
0,042			46,27			
0,043	24,13					

0,045			45,09			
0,045			41,79			
0,047			38,93			
0,049			38,12			
0,055	23,20					
0,055	20,77					
0,058				36,13		
0,058				35,39		
0,060			37,75			
0,060			35,57			
0,063				32,46		
0,063				30,97		
0,063				27,74		
0,065					39,24	
0,066				27,30		
0,068	17,10					
0,068				27,24		
0,070					37,13	
0,071		34,27				
0,072		30,72				
0,099					36,38	
0,104					35,63	
0,129			24,38			
0,129				24,38		
12,564						37,44
12,564						37,38
12,564						35,95
12,619						35,70
12,619						35,32
12,676						31,47
12,676						31,41
12,679						31,16
12,687						31,03
12,688						29,54
12,691						26,31
12,696						25,31
13,148						25,06
13,154						24,75
13,154						24,25
13,155						23,63
13,155						22,82
13,156						21,95
13,261						21,39
13,261						20,71
13,261						19,90
13,264						19,78

13,734						19,71
13,858						18,72
13,878						18,53
13,879						18,35
13,884						18,22
13,972						17,23
14,096						17,04
14,099						16,67
14,117						16,36
14,139						15,36
14,150						15,17
14,216						15,11
14,216						14,37
14,470						14,18

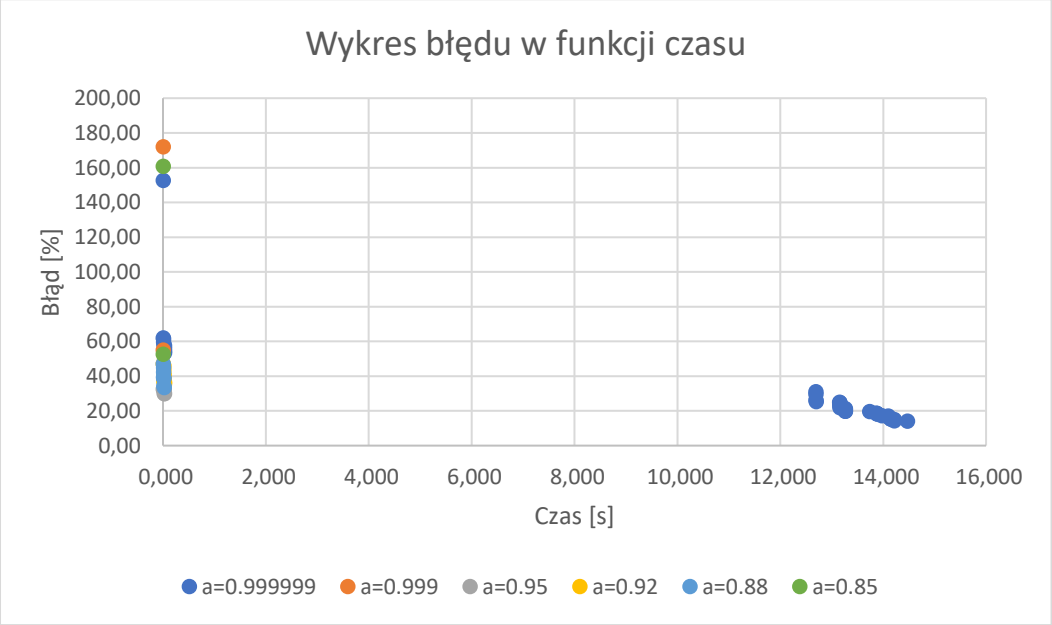


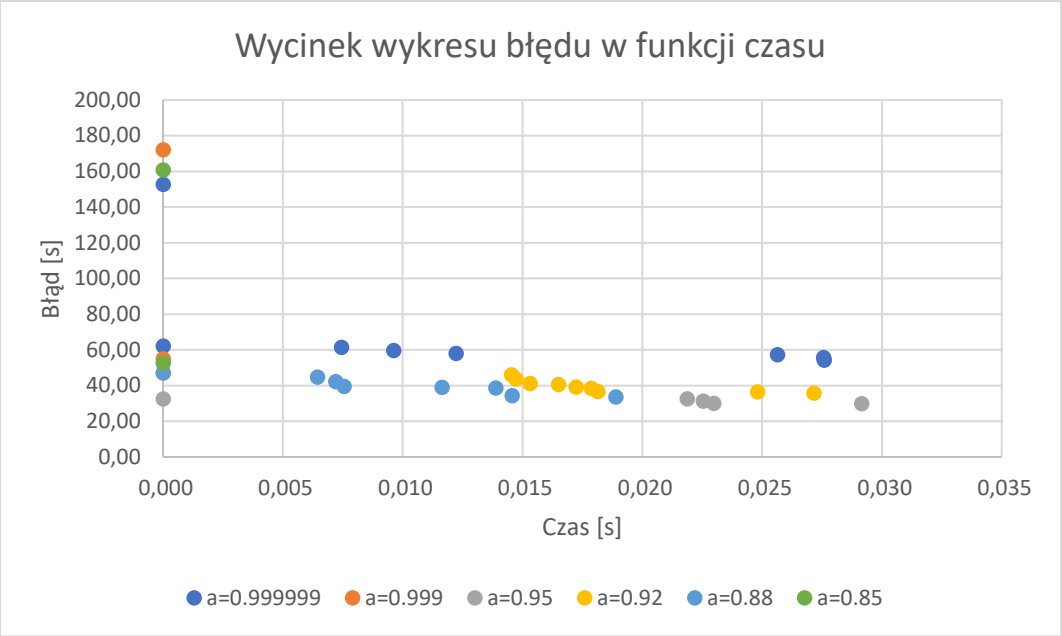


b) Schładzanie liniowe

Czas [s]	a=0.999999	a=0.999	a=0.95	a=0.92	a=0.88	a=0.85
0,000	152,67					
0,000		172,08				
0,000			32,65			
0,000				47,14		
0,000					47,14	
0,000						160,88
0,000		54,98				
0,000						52,74
0,000	62,13					
0,006					44,84	
0,007					42,35	
0,007	61,50					
0,008					39,55	
0,010	59,70					
0,012					39,12	
0,012	58,15					
0,014					38,62	
0,015				46,08		
0,015					34,45	
0,015				43,66		
0,015				41,23		
0,017				40,67		
0,017				39,18		
0,018				38,43		
0,018				36,75		
0,019					33,58	
0,022			32,59			
0,023			31,41			
0,023			30,04			
0,025				36,63		
0,026	57,40					
0,027				35,76		
0,028	55,72					
0,028	54,23					
0,029			29,91			
0,030	53,36					
12,687	31,03					
12,688	29,54					
12,691	26,31					
12,696	25,31					
13,148	25,06					
13,154	24,75					
13,154	24,25					
13,155	23,63					

13,155	22,82					
13,156	21,95					
13,261	21,39					
13,261	20,71					
13,261	19,90					
13,264	19,78					
13,734	19,71					
13,858	18,72					
13,878	18,53					
13,879	18,35					
13,884	18,22					
13,972	17,23					
14,096	17,04					
14,099	16,67					
14,117	16,36					
14,139	15,36					
14,150	15,17					
14,216	15,11					
14,216	14,37					
14,470	14,18					



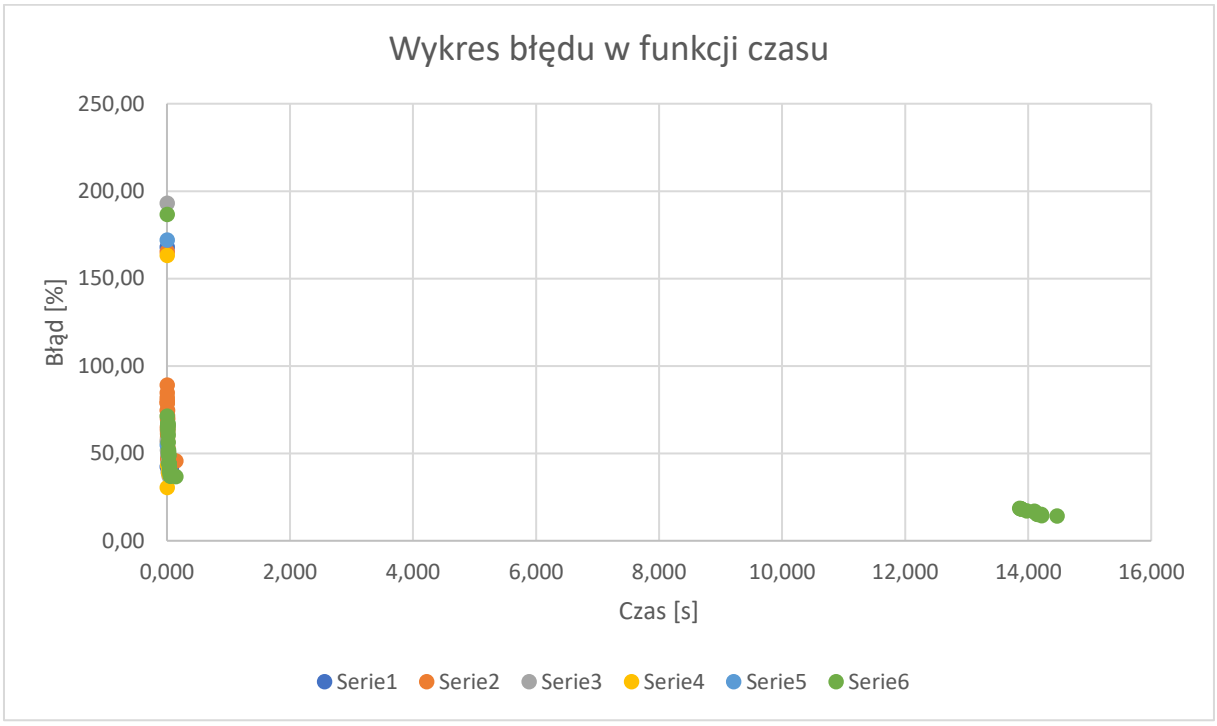


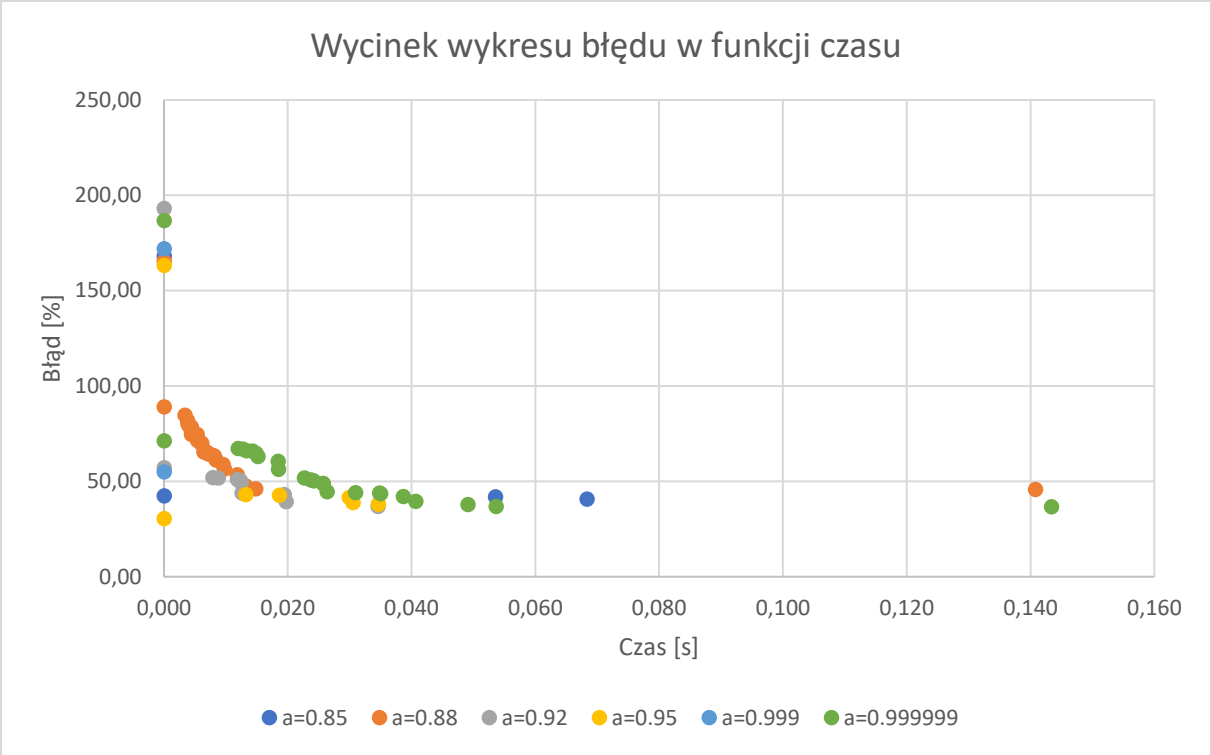
c) Schładzanie logarytmiczne

Czas [s]	a=0.85	a=0.88	a=0.92	a=0.95	a=0.999	a=0.999999
0,000	168,10					
0,000		165,42				
0,000			193,22			
0,000				163,25		
0,000					172,08	
0,000						186,75
0,000	42,41					
0,000				30,47		
0,000					54,98	
0,000						71,39
0,000		89,12				
0,000			57,21			
0,003		84,70				
0,004		81,90				
0,004		80,16				
0,004		78,98				
0,004		78,61				
0,004		74,75				
0,005		74,69				
0,005		71,52				
0,006		70,15				
0,006		69,59				
0,006		65,42				
0,007		65,24				
0,007		64,37				
0,008		64,18				
0,008			51,99			
0,008		63,37				

0,008		62,87				
0,008		61,26				
0,009			51,80			
0,010		58,89				
0,010		56,59				
0,012		53,48				
0,012		51,43				
0,012			51,06			
0,012						67,29
0,012			50,56			
0,012			50,50			
0,012			49,38			
0,013			44,03			
0,013						66,98
0,013		47,89				
0,013				43,10		
0,013		47,33				
0,013						66,17
0,014						65,92
0,015		46,21				
0,015						64,80
0,015						63,06
0,018						60,57
0,018						56,41
0,019				42,85		
0,019			43,10			
0,020			39,37			
0,023						51,93
0,024						50,81
0,024						50,31
0,026						48,94
0,026						48,88
0,026						44,65
0,030				41,67		
0,031				38,99		
0,031						44,09
0,035			36,88			
0,035				38,12		
0,035						44,03
0,035						43,66
0,039						42,10
0,041						39,68
0,049						38,00
0,054	41,92					
0,054						36,88
0,068	40,73					

0,141		45,77				
0,143						36,82
13,858						18,72
13,878						18,53
13,879						18,35
13,884						18,22
13,972						17,23
14,096						17,04
14,099						16,67
14,117						16,36
14,139						15,36
14,150						15,17
14,216						15,11
14,216						14,37
14,470						14,18

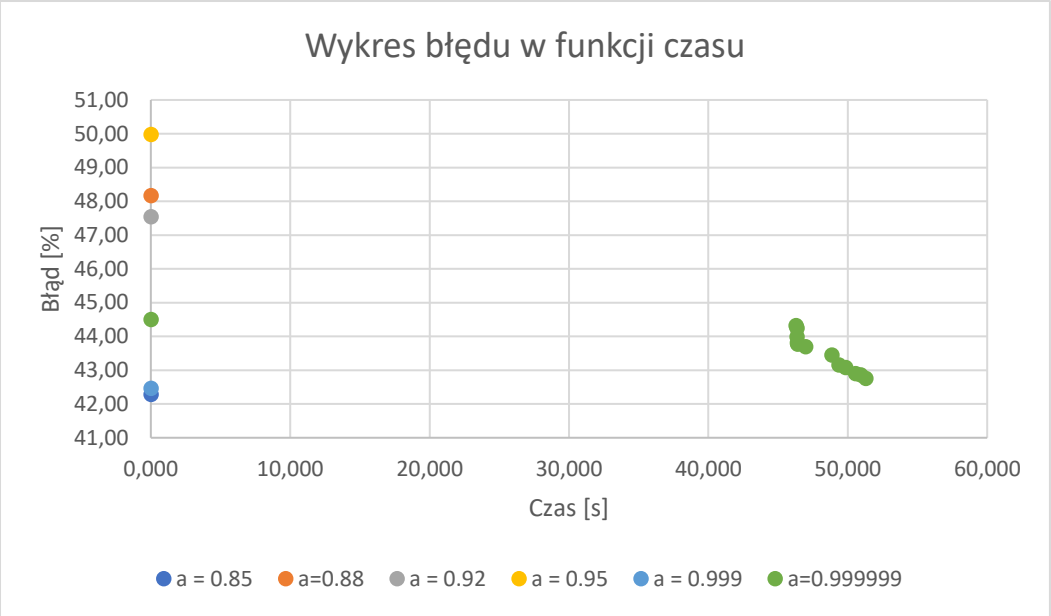


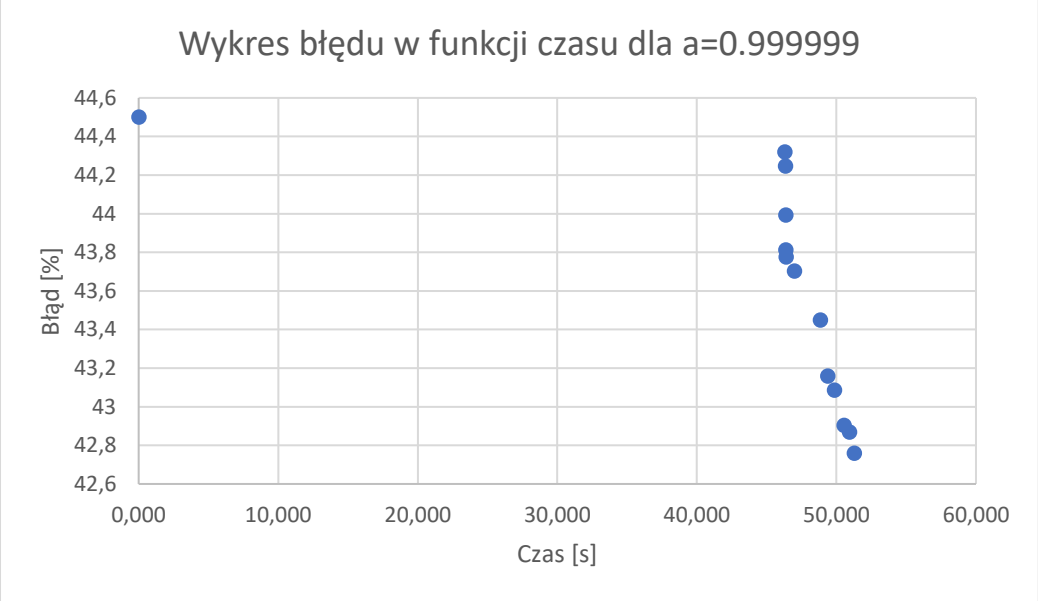


5.3.2 Plik ftv170.xml

a) Schładzanie z instrukcji

Czas [s]	a = 0.85	a=0.88	a = 0.92	a = 0.95	a = 0.999	a=0.999999
0,000	42,29					
0,000		48,17				
0,000			47,55			
0,000				49,98		
0,000					42,47	
0,000						44,50
46,291						44,32
46,353						44,25
46,362						43,99
46,365						43,81
46,394						43,77
46,976						43,70
48,851						43,45
49,369						43,16
49,849						43,09
50,543						42,90
50,921						42,87
51,278						42,76

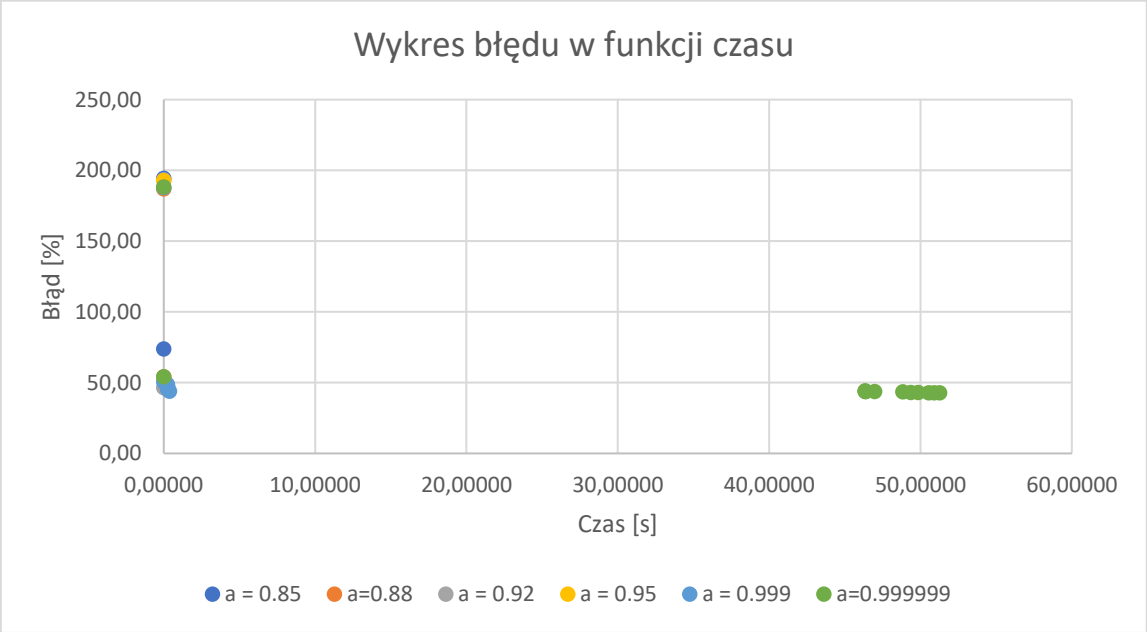


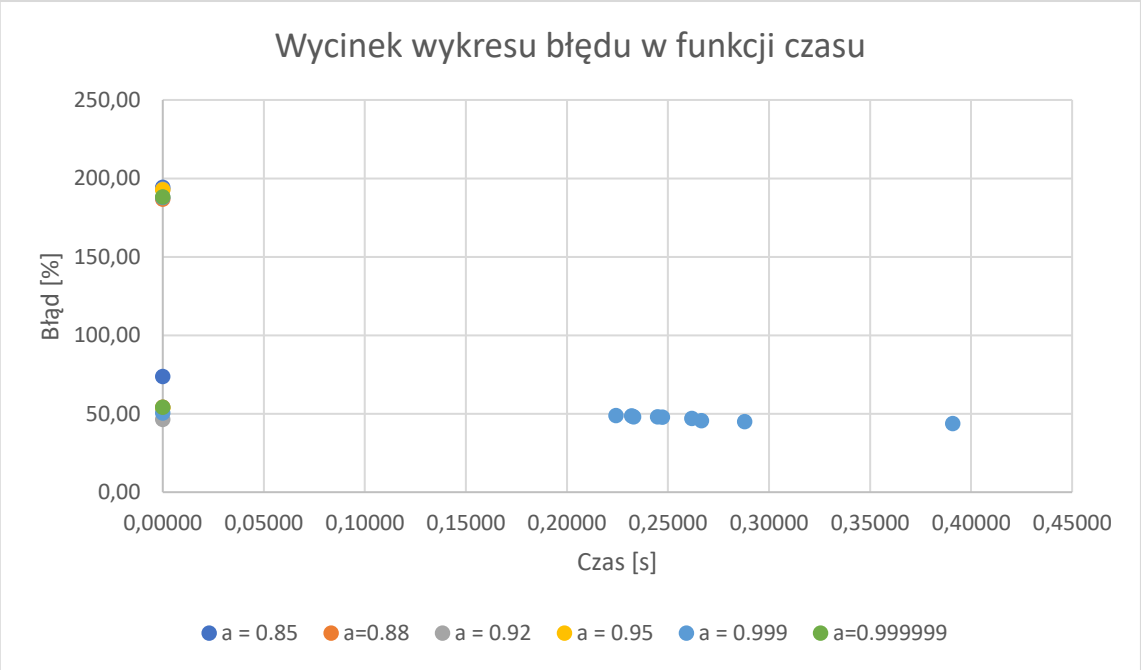


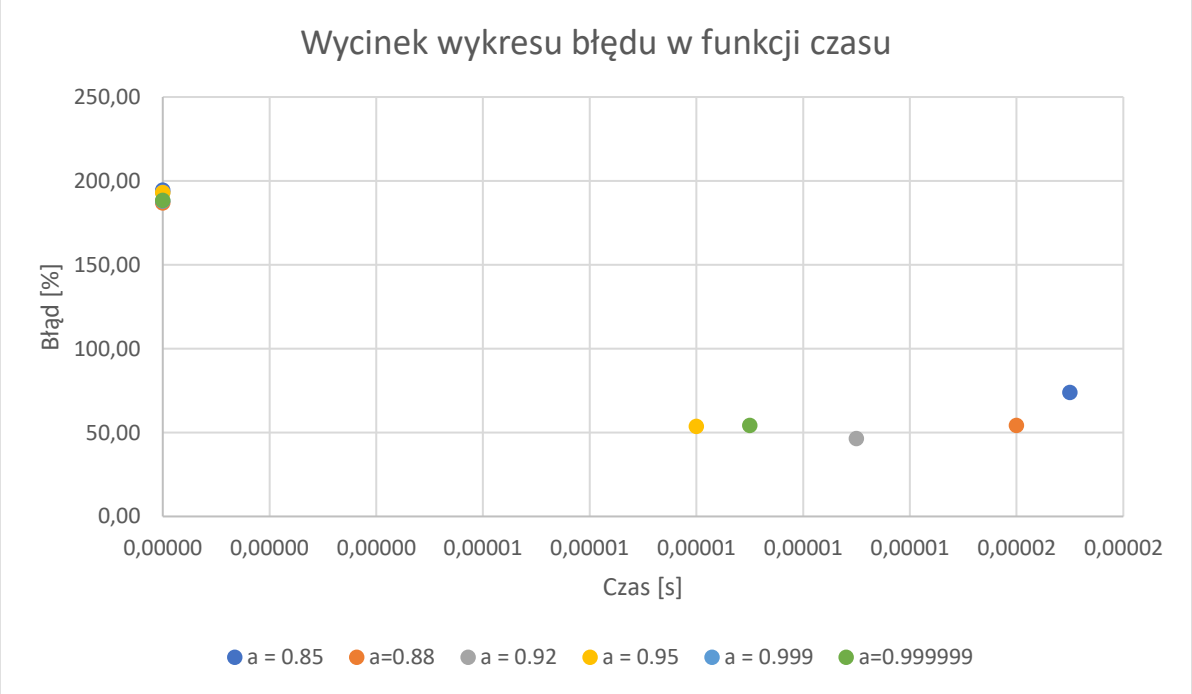
b) Schładzanie liniowe

Czas [s]	a = 0.85	a=0.88	a = 0.92	a = 0.95	a = 0.999	a=0.999999
0,00000	194,41					
0,00000		186,82				
0,00000			193,18			
0,00000				192,85		
0,00000					187,91	
0,00000						188,28
0,00001				53,61		
0,00001						54,16
0,00001			46,46			
0,00002		54,19				
0,00002	73,79					

0,00002					50,53	
0,22436					48,86	
0,23210					48,78	
0,23297					48,13	
0,24481					48,02	
0,24731					47,80	
0,26188					47,08	
0,26666					45,70	
0,28795					45,05	
0,39100					43,88	
46,35285						44,25
46,36185						43,99
46,36455						43,81
46,39410						43,77
46,97571						43,70
48,85052						43,45
49,36883						43,16
49,84867						43,09
50,54289						42,90
50,92106						42,87
51,27809						42,76



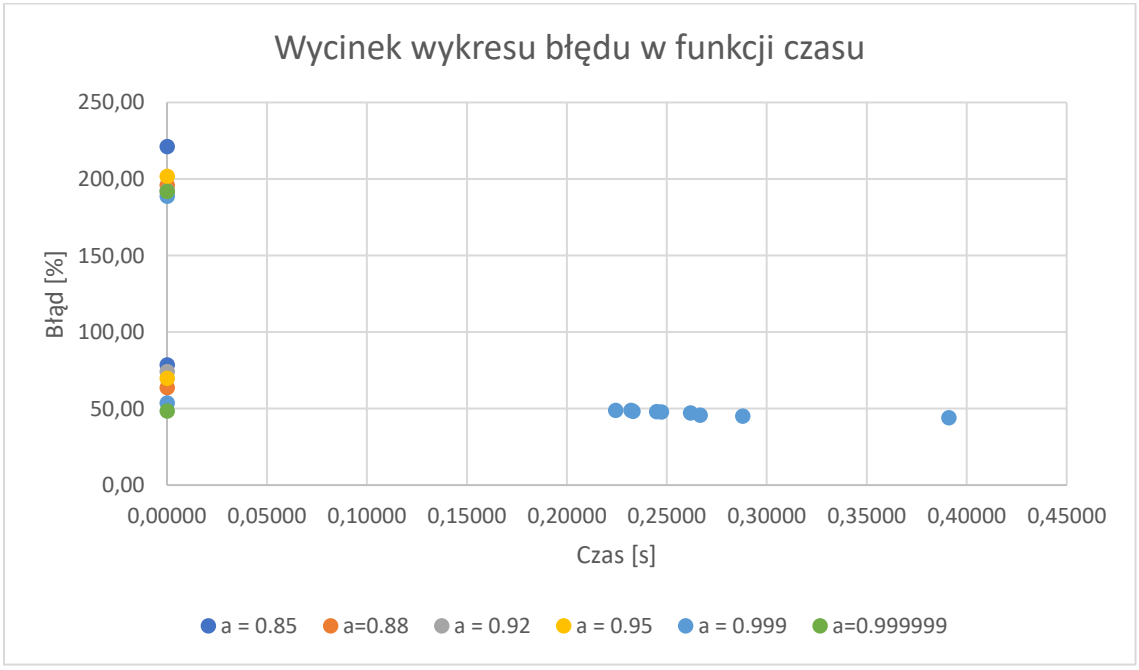


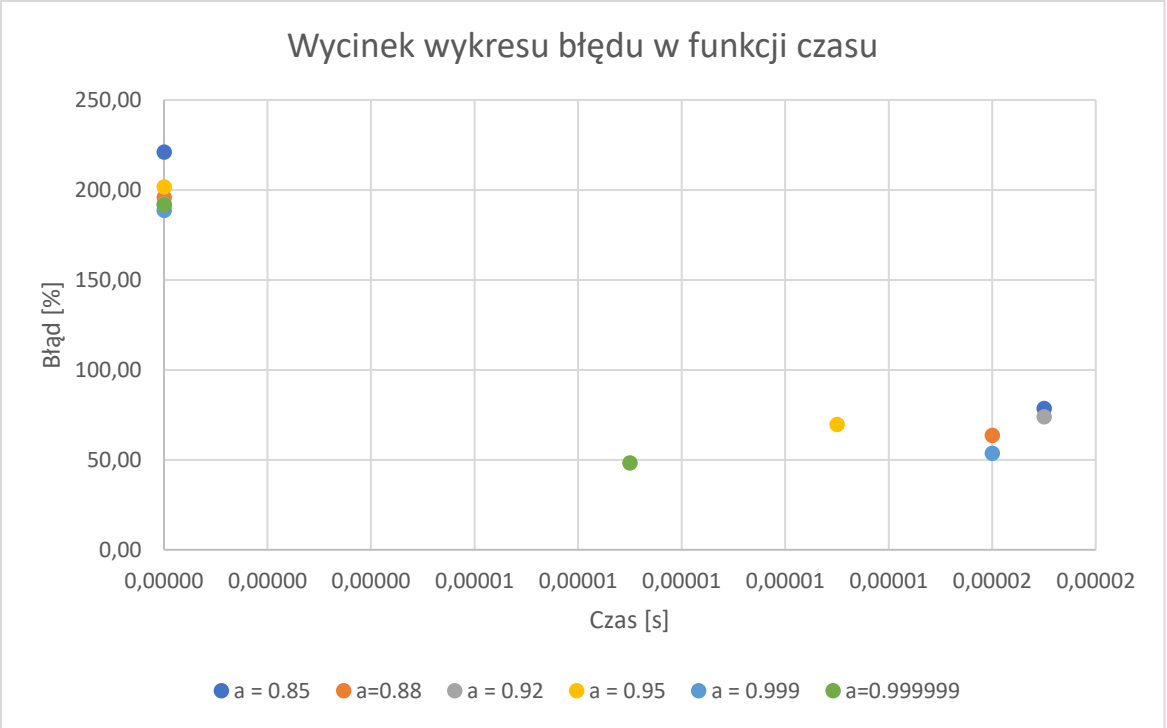


c) Schładzanie logarytmiczne

Czas [s]	a = 0.85	a=0.88	a = 0.92	a = 0.95	a = 0.999	a=0.999999
0,00000	221,16					
0,00000		195,75				
0,00000			192,30			
0,00000				201,63		
0,00000					188,78	
0,00000						191,76
0,00001						48,31
0,00001				69,76		

0,00002		63,59				
0,00002					53,68	
0,00002	78,58					
0,00002			74,05			
0,22436					48,86	
0,23210					48,78	
0,23297					48,13	
0,24481					48,02	
0,24731					47,80	
0,26188					47,08	
0,26666					45,70	
0,28795					45,05	
0,39100					43,88	
46,35285						44,25
46,36185						43,99
46,36455						43,81
46,39410						43,77
46,97571						43,70
48,85052						43,45
49,36883						43,16
49,84867						43,09
50,54289						42,90
50,92106						42,87
51,27809						42,76



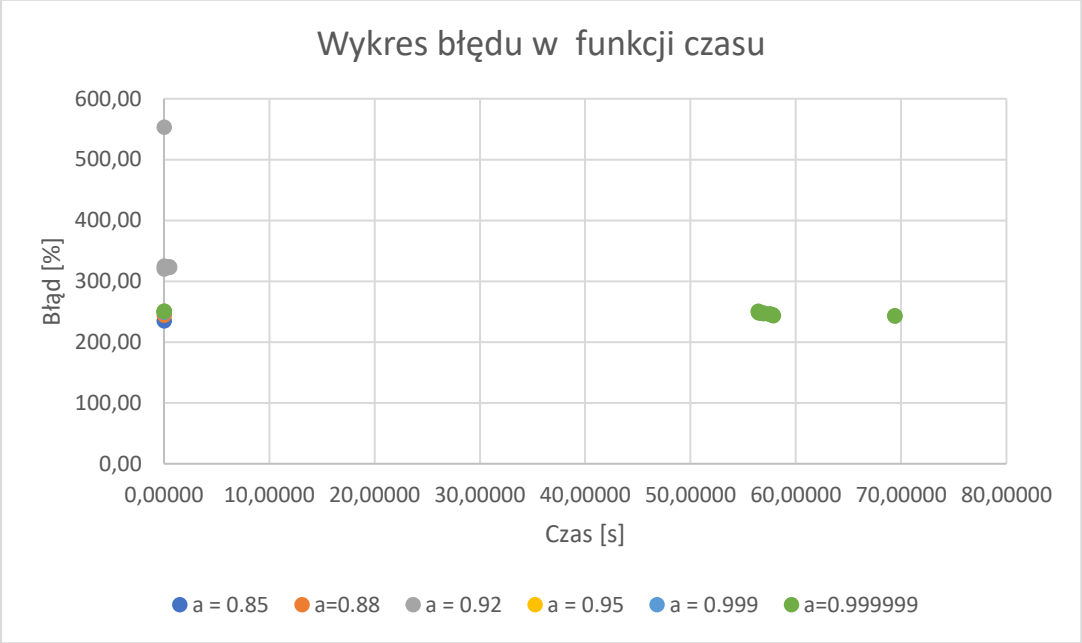


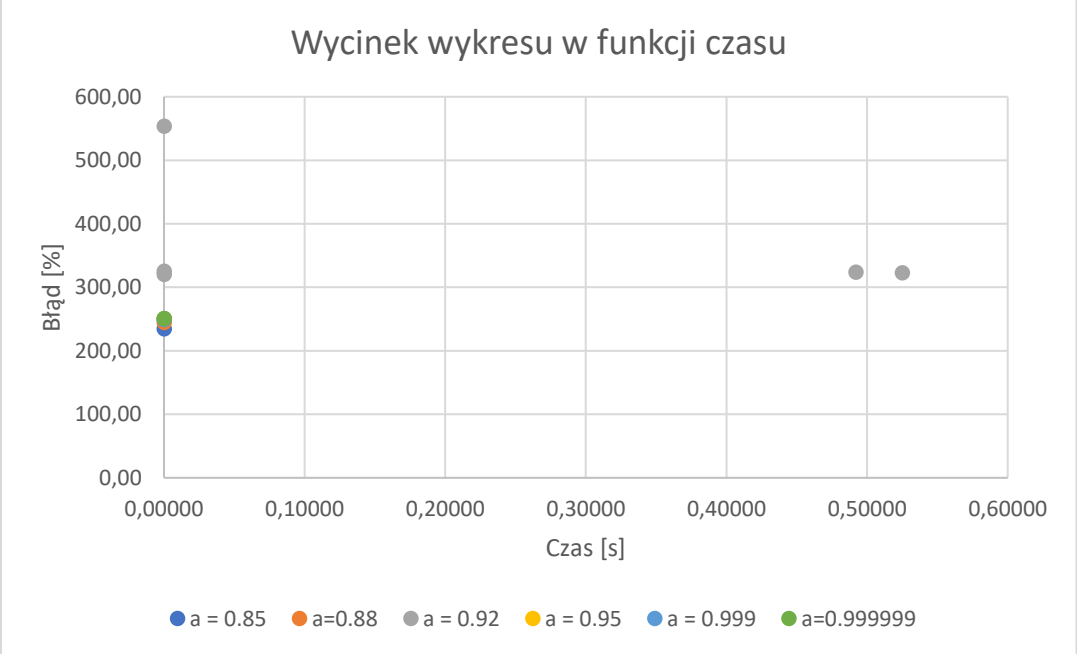
5.3.3 Plik rbg358.xml

a) Schładzanie z instrukcji

Czas [s]	a = 0.85	a=0.88	a = 0.92	a = 0.95	a = 0.999	a=0.999999
0,00000	234,82					
0,00000		244,54				
0,00000			553,65			
0,00000				250,13		
0,00000					249,44	
0,00000						250,99
0,00001			325,28			
0,49211			324,08			
0,52496			322,87			
0,61074			320,21			
56,43712						250,64
56,43722						250,21
56,46089						249,70
56,47495						248,41
56,57442						248,32
56,70782						248,15
56,84859						247,98
56,94444						247,29
56,94594						247,21
57,48691						246,43
57,51957						245,66
57,64667						245,40
57,78824						245,23
57,79102						244,54

57,84614						243,59
69,37299						243,42
69,39912						243,16



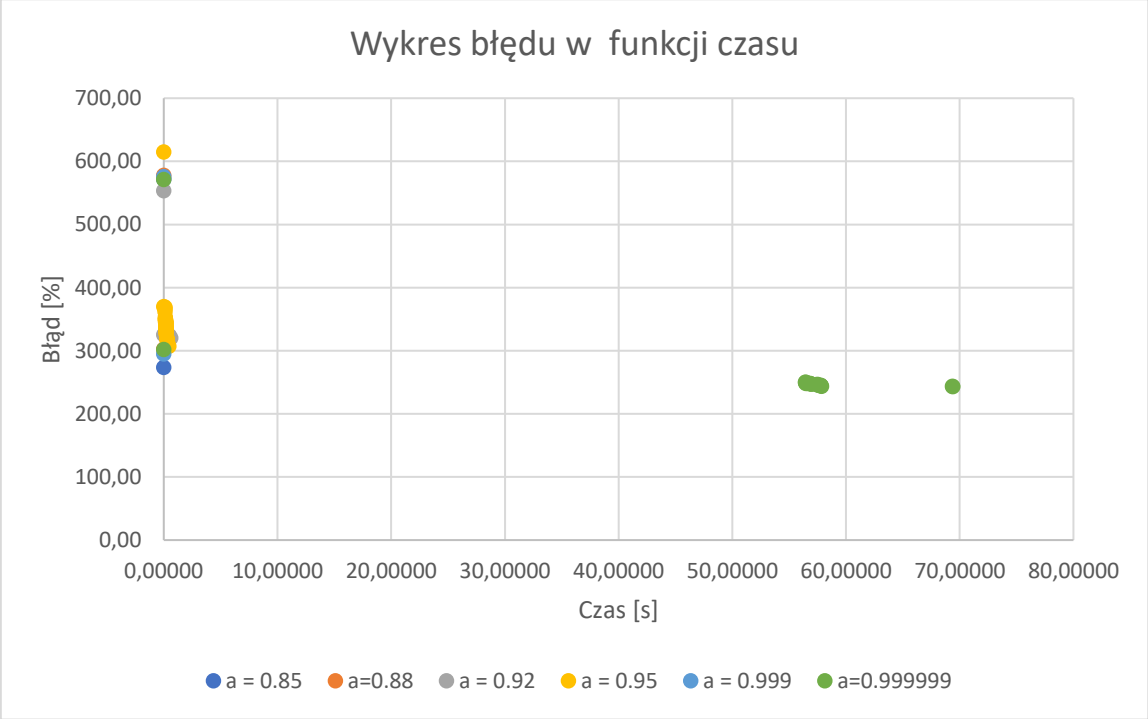


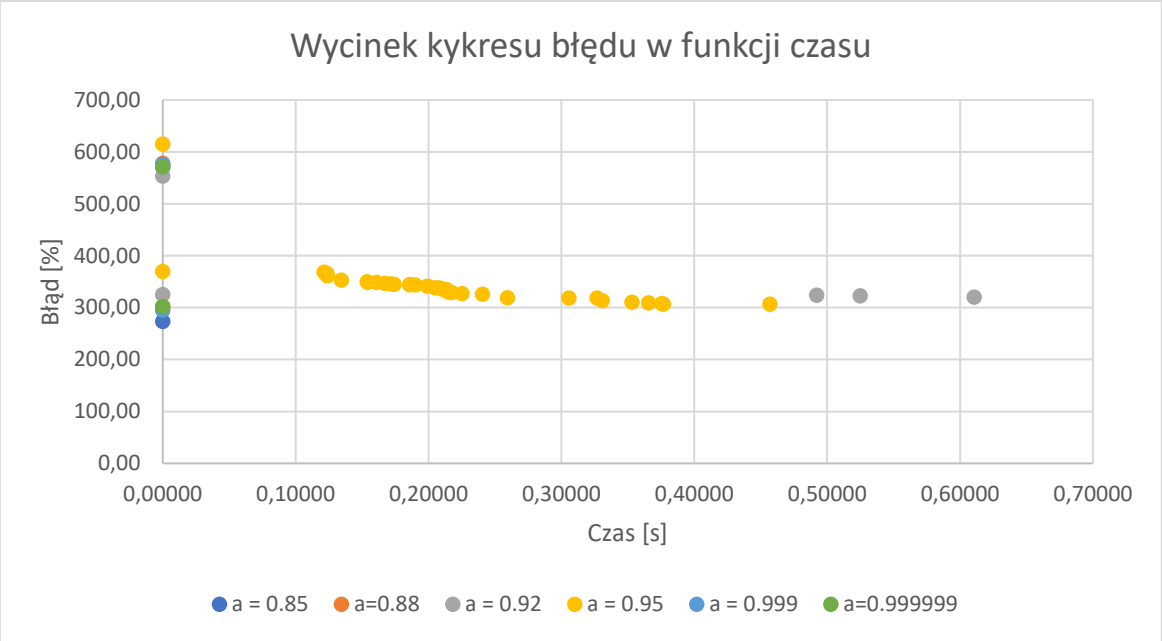
b) Schładzanie liniowe

Czas [s]	a = 0.85	a=0.88	a = 0.92	a = 0.95	a = 0.999	a=0.999999
0,00000	571,71					
0,00000		578,07				
0,00000			553,65			
0,00000				614,96		
0,00000					575,67	

0,00000						571,20
0,00001			325,28			
0,00001					295,10	
0,00001						302,15
0,00002	273,52					
0,00002				369,91		
0,00002		302,32				
0,12149				368,70		
0,12373				366,29		
0,12378				361,91		
0,13436				353,22		
0,15396				350,13		
0,15430				348,93		
0,16090				348,58		
0,16728				347,03		
0,17191				345,49		
0,17429				345,23		
0,18591				344,71		
0,18599				344,63		
0,19005				343,68		
0,19936				341,27		
0,20502				338,61		
0,20761				338,35		
0,21124				335,86		
0,21150				334,91		
0,21364				334,74		
0,21508				330,61		
0,21639				329,66		
0,21766				329,41		
0,22532				327,43		
0,24070				325,97		
0,25949				319,09		
0,30570				318,57		
0,32706				318,40		
0,33071				313,76		
0,35318				310,66		
0,36582				309,54		
0,37570				308,34		
0,37686				307,22		
0,45704				307,14		
0,49211			324,08			
0,52496			322,87			
0,61074			320,21			
56,43722						250,21
56,46089						249,70
56,47495						248,41

56,57442						248,32
56,70782						248,15
56,84859						247,98
56,94444						247,29
56,94594						247,21
57,48691						246,43
57,51957						245,66
57,64667						245,40
57,78824						245,23
57,79102						244,54
57,84614						243,59
69,37299						243,42
69,39912						243,16



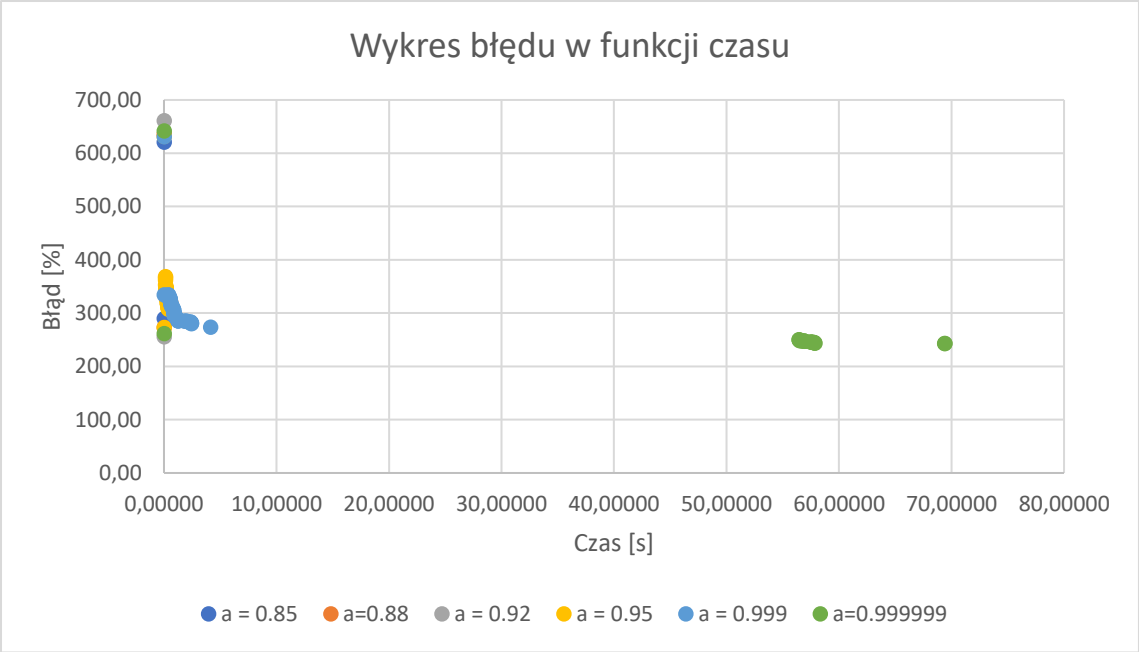


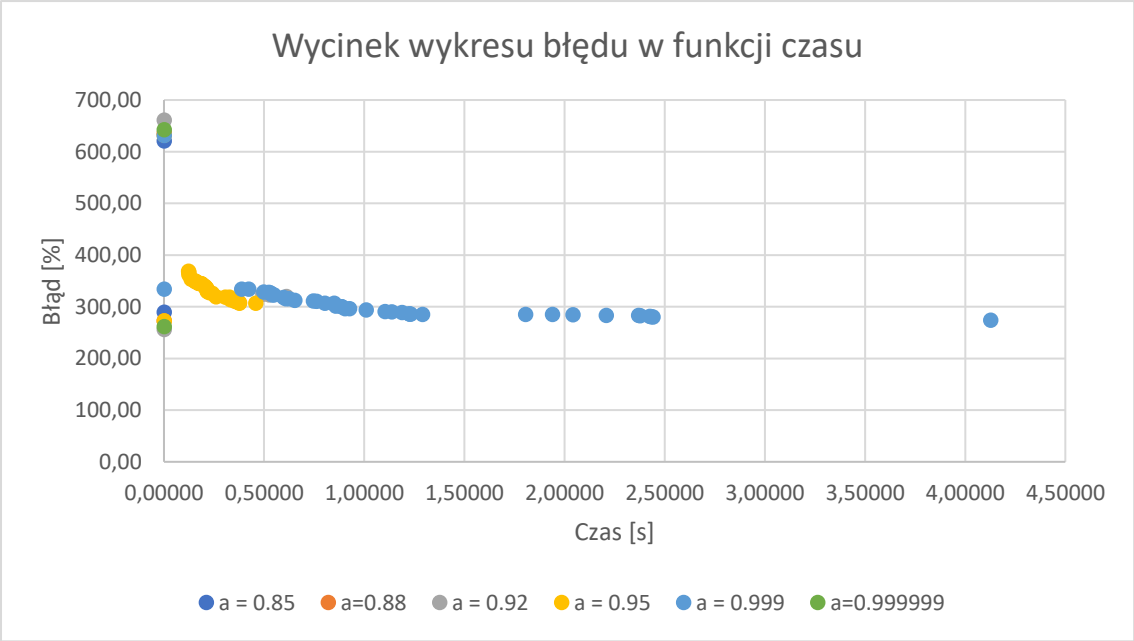
c) Schładzanie logarytmiczne

Czas	a = 0.85	a=0.88	a = 0.92	a = 0.95	a = 0.999	a=0.999999
0,00000	621,15					
0,00000		635,94				
0,00000			661,31			
0,00000				631,99		
0,00000					631,38	
0,00000						642,30
0,00001		273,17				
0,00001			255,89			
0,00001	289,77					
0,00002				273,17		
0,00002					334,48	
0,00002						261,56
0,12149				368,70		
0,12373				366,29		
0,12378				361,91		
0,13436				353,22		
0,15396				350,13		
0,15430				348,93		
0,16090				348,58		
0,16728				347,03		
0,17191				345,49		
0,17429				345,23		
0,18591				344,71		
0,18599				344,63		
0,19005				343,68		
0,19936				341,27		
0,20502				338,61		

0,20761				338,35		
0,21124				335,86		
0,21150				334,91		
0,21364				334,74		
0,21508				330,61		
0,21639				329,66		
0,21766				329,41		
0,22532				327,43		
0,24070				325,97		
0,25949				319,09		
0,30570				318,57		
0,32706				318,40		
0,33071				313,76		
0,35318				310,66		
0,36582				309,54		
0,37570				308,34		
0,37686				307,22		
0,38681					334,31	
0,42317					333,96	
0,45704				307,14		
0,49211			324,08			
0,49700					328,89	
0,52374					327,77	
0,52496			322,87			
0,53149					326,57	
0,53581					324,59	
0,54069					322,96	
0,54809					322,87	
0,59650					318,06	
0,60731					315,74	
0,61074			320,21			
0,62462					315,31	
0,65295					312,81	
0,74491					311,01	
0,76216					310,75	
0,80344					307,22	
0,84940					306,71	
0,85248					305,33	
0,85773					301,29	
0,88645					300,34	
0,88796					299,83	
0,90401					296,22	
0,92674					296,13	
1,00859					293,64	
1,10292					290,97	
1,13746					289,94	

1,18801					289,25	
1,22255					286,33	
1,22563					285,90	
1,23093					285,81	
1,28963					285,55	
1,80450					285,21	
1,93972					285,04	
2,04158					284,87	
2,20692					283,66	
2,36888					283,32	
2,37666					282,63	
2,42488					281,51	
2,43216					280,74	
2,44155					280,31	
4,12722					273,95	
56,43722						250,21
56,46089						249,70
56,47495						248,41
56,57442						248,32
56,70782						248,15
56,84859						247,98
56,94444						247,29
56,94594						247,21
57,48691						246,43
57,51957						245,66
57,64667						245,40
57,78824						245,23
57,79102						244,54
57,84614						243,59
69,37299						243,42
69,39912						243,16





5.3.4 Wnioski

Powyższe testy wykazały, że bardzo dużą rolę w procesie znajdywania rozwiązania stanowi odpowiedni dobór współczynników oraz warunku stopu. Analizując otrzymane wyniki zauważyłem, iż w zależności od współczynnika a, rozwiązania otrzymujemy w innym przedziale czasu, jest to związane z akceptacją rozwiązań. W zależności od schematu schładzania temperatura spada z inną prędkością np. dla schematu liniowego spada ona praktycznie liniowo więc przy mniejszych współczynnikach a, za szybko spada ona do bardzo małych wartości. Przez co bardzo mało gorszych rozwiązań jest akceptowanych. Inaczej to wygląda w schemacie logarytmicznym gdzie najpierw ta temperatura spada bardzo szybko, ale gdy jest już bliska zeru to praktycznie się wypłaszacza przez co nie notuje dużych popraw, ale przez akceptowanie sporej ilości rozwiązań generuje ich dużo.

5.4 Porównanie dwóch algorytmów

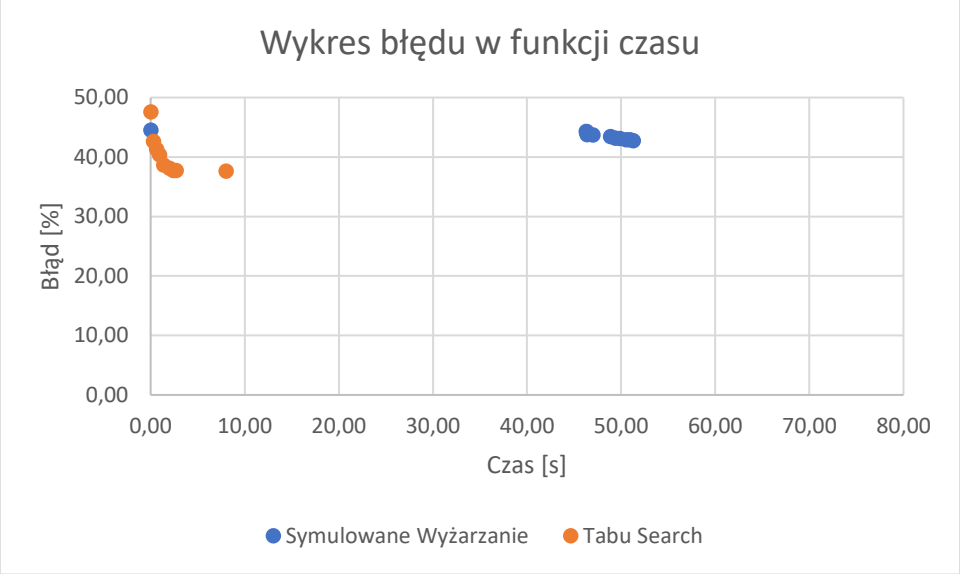
	ftv55.xml		ftv10.xml		rbg358.xml	
Czas [s]	Symulowane Wyżarzanie	Tabu Search	Symulowane Wyżarzanie	Tabu Search	Symulowane Wyżarzanie	Tabu Search
0,00	40,92	36,84	44,50	47,55	250,99	59,67
0,01		30,80				
0,03		28,25				
0,04		25,64				
0,05		24,14				
0,06		23,27				
0,07		22,59				
0,08		22,53				
0,17		22,46				
0,30				42,65		
0,62				41,27		
0,92				40,36		
1,38				38,66		
2,00				38,11		
2,38				37,75		
2,69				37,71		
3,88						56,15
7,82						54,60
7,99				37,60		
11,76						53,31
12,56	37,44					
12,56	37,38					
12,56	35,95					
12,62	35,70					
12,62	35,32					
12,68	31,47					
12,68	31,41					
12,68	31,16					
12,69	31,03					
12,69	29,54					
12,69	26,31					
12,70	25,31					
13,15	25,06					
13,15	24,75					
13,15	24,25					
13,15	23,63					
13,16	22,82					
13,16	21,95					
13,26	21,39					
13,26	20,71					
13,26	19,90					
13,26	19,78					

13,73	19,71					
13,86	18,72					
13,88	18,53					
13,88	18,35					
13,88	18,22					
13,97	17,23					
14,10	17,04					
14,10	16,67					
14,12	16,36					
14,14	15,36					
14,15	15,17					
14,22	15,11					
14,22	14,37					
14,47	14,18					
15,76						52,02
19,73						50,82
23,71						49,70
28,13						48,67
32,15						47,64
46,29			44,32			
46,35			44,25			
46,36			43,99			
46,36			43,81			
46,39			43,77			
46,98			43,70			
48,85			43,45			
49,37			43,16			
49,85			43,09			
50,54			42,90			
50,92			42,87			
51,28			42,76			
56,44					250,64	
56,44					250,21	
56,46					249,70	
56,47					248,41	
56,57					248,32	
56,71					248,15	
56,85					247,98	
56,94					247,29	
56,95					247,21	
57,49					246,43	
57,52					245,66	
57,65					245,40	
57,79					245,23	
57,79					244,54	
57,85					243,59	

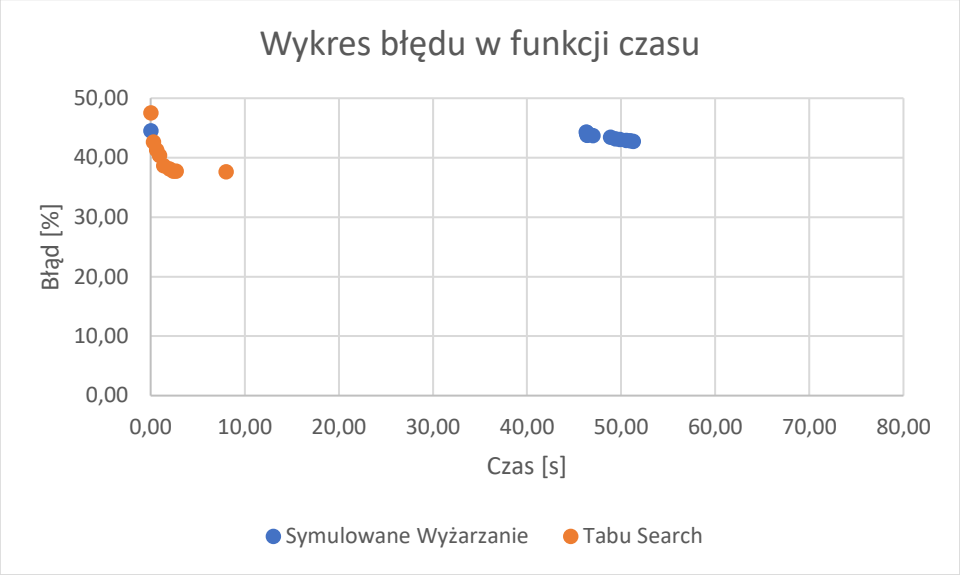
69,37					243,42	
69,40					243,16	

5.4.1 Plik ftv55.xml

5.4.2 Plik ftv170.xml



5.4.3 Plik rbg358.xml



5.4.4 Wnioski

Jak można zauważyć na powyższych wykresach algorytm tabu search znajduje najlepsze rozwiązanie szybciej niż symulowane wyżarzanie. Związane to jest z tym jaki współczynnik a został wybrany, ponieważ od tego zależy w którym momencie będzie najwięcej zmian. Z racji wyboru najlepszych rozwiązań do porównania wybrałem wolny ale dokładny współczynnik a = 0.999999. Analizując wykresy można również stwierdzić, iż dla większych problemów algorytm tabu search lepiej się spisał. Jednak testy wyżarzania odbywały się zaledwie dla 6 różnych parametrów, przez co nie możemy być w pełni pewni co byłoby dla innych.

6 Najlepsze uzyskane rozwiązania

Przedstawione przeze mnie pliki z najlepszymi rozwiązaniami będą się nieco różnić niż te prezentowane na powyższych wykresach. Związane jest to z moim błędem, ponieważ uruchamiając testy nie zauważyłem, że pliki z wynikami mi się nadpisują dla każdego przebiegu pętli a nie tylko dla lepszego. Z racji na czasochłonność testów nie byłem w stanie ich powtórzyć, zatem postanowiłem w między czasie opracowywania wyników poszukać lepszych rozwiązań. Dla niektórych udało mi się znaleźć nawet zadziwiająco dobre, a dla innych tragiczne.

6.1 Plik ftv55.xml

6.1.1 Algorytm Tabu search

Nazwa pliku: TS_ftv55.txt

Wynik: 1991

Ścieżka: 42 | 21 | 50 | 23 | 54 | 27 | 49 | 53 | 43 | 44 | 28 | 29 | 26 | 25 | 24 | 20 | 40 | 41 | 22 | 19 | 18 | 39 | 38 | 11 | 10 | 51 | 14 | 13 | 35 | 4 | 6 | 5 | 47 | 31 | 46 | 55 | 34 | 1 | 2 | 33 | 52 | 0 | 48 | 3 | 32 | 8 | 36 | 9 | 37 | 7 | 12 | 15 | 16 | 17 | 45 | 30

6.1.2 Algorytm symulowanego wyżarzania

Nazwa pliku: SA_ftv55.txt

Wynik: 1674

Ścieżka: 9 | 37 | 11 | 10 | 12 | 15 | 16 | 17 | 52 | 0 | 33 | 31 | 48 | 3 | 6 | 5 | 47 | 53 | 43 | 44 | 28 | 29 | 26 | 25 | 24 | 18 | 39 | 38 | 19 | 20 | 40 | 42 | 22 | 41 | 21 | 50 | 23 | 54 | 27 | 49 | 45 | 30 | 46 | 55 | 34 | 1 | 2 | 14 | 51 | 13 | 35 | 4 | 7 | 32 | 8 | 36

6.2 Plik ftv170.xml

6.2.1 Algorytm Tabu search

Nazwa pliku: TS_ftv170.txt

Wynik: 3636

Ścieżka: 119 | 120 | 121 | 122 | 123 | 162 | 102 | 103 | 117 | 118 | 124 | 129 | 128 | 130 | 131 | 113 | 164 | 127 | 126 | 125 | 136 | 137 | 138 | 135 | 139 | 140 | 6 | 7 | 8 | 9 | 10 | 76 | 74 | 75 | 11 | 12 | 18 | 19 | 20 | 21 | 22 | 16 | 26 | 27 | 28 | 29 | 30 | 31 | 33 | 34 | 156 | 40 | 39 | 38 | 37 | 170 | 168 | 73 | 77 | 1 | 2 | 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 | 101 | 104 | 110 | 109 | 114 | 115 | 116 | 146 | 145 | 144 | 143 | 147 | 148 | 149 | 161 | 160 | 151 | 152 | 142 | 141 | 132 | 112 | 169 | 111 | 3 | 4 | 5 | 133 | 134 | 14 | 13 | 17 | 32 | 158 | 36 | 157 | 41 | 155 | 42 | 45 | 44 | 46 | 47 | 48 | 49 | 35 | 23 | 24 | 25 | 150 | 15 | 159

6.2.2 Algorytm symulowanego wyżarzania

Nazwa pliku: SA_ftv170.txt

Wynik: 3936

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

6.3 Plik rbg358.xml

6.3.1 Algorytm Tabu search

Nazwa pliku: TS_rbg358.txt

Wynik: 1756

Ścieżka: 126 | 202 | 13 | 3 | 164 | 218 | 321 | 260 | 55 | 4 | 44 | 40 | 10 | 27 | 16 | 17 | 120 | 34 | 9 | 166 | 35 | 18 | 329 | 64 | 36 | 29 | 337 | 91 | 37 | 28 | 24 | 41 | 205 | 137 | 45 | 25 | 12 | 48 | 11 | 15 | 284 | 6 | 81 | 246 | 56 | 145 | 7 | 23 | 209 | 49 | 5 | 97 | 208 | 63 | 19 | 20 | 22 | 78 | 98 | 104 | 114 | 124 | 148 | 135 | 39 | 21 | 69 | 305 | 180 | 47 | 79 | 51 | 67 | 95 | 52 | 171 | 53 | 59 | 125 | 116 | 66 | 14 | 249 | 0 | 1 | 58 | 243 | 68 | 297 | 99 | 70 | 26 | 112 | 103 | 72 | 172 | 146 | 76 | 196 | 118 | 183 | 176 | 184 | 214 | 77 | 42 | 198 | 80 | 106 | 181 | 328 | 61 | 86 | 165 | 223 | 88 | 123 | 285 | 89 | 192 | 248 | 90 | 347 | 83 | 92 | 275 | 191 | 204 | 230 | 276 | 182 | 207 | 93 | 32 | 31 | 117 | 333 | 87 | 288 | 119 | 267 | 75 | 212 | 224 | 122 | 303 | 200 | 127 | 131 | 245 | 128 | 293 | 102 | 210 | 265 | 226 | 129 | 54 | 185 | 57 | 197 | 130 | 111 | 101 | 132 | 316 | 94 | 134 | 82 | 253 | 136 | 85 | 84 | 33 | 259 | 173 | 352 | 290 | 138 | 38 | 139 | 271 | 310 | 140 | 110 | 304 | 142 | 109 | 96 | 144 | 107 |

193 | 153 | 65 | 168 | 154 | 43 | 215 | 308 | 100 | 50 | 315 | 319 | 257 | 155 | 115 | 292 | 156 | 30 | 289 | 157 | 105 | 158 | 149 | 241 | 189 | 342 | 282 | 274 | 159 | 121 | 287 | 161 | 152 | 335 | 147 | 162 | 133 | 143 | 163 | 2 | 247 | 151 | 349 | 141 | 167 | 8 | 62 | 169 | 348 | 170 | 313 | 174 | 46 | 108 | 195 | 175 | 307 | 258 | 177 | 150 | 216 | 178 | 320 | 299 | 179 | 356 | 331 | 186 | 346 | 312 | 206 | 238 | 268 | 187 | 188 | 291 | 190 | 73 | 280 | 194 | 74 | 270 | 281 | 324 | 262 | 332 | 272 | 203 | 278 | 213 | 217 | 201 | 251 | 219 | 322 | 296 | 301 | 340 | 311 | 343 | 273 | 220 | 250 | 113 | 269 | 221 | 199 | 71 | 277 | 302 | 317 | 222 | 309 | 314 | 225 | 294 | 211 | 279 | 338 | 325 | 350 | 300 | 227 | 344 | 334 | 228 | 327 | 60 | 295 | 261 | 242 | 306 | 336 | 244 | 326 | 323 | 351 | 229 | 160 | 330 | 318 | 231 | 354 | 232 | 298 | 353 | 233 | 355 | 357 | 234 | 341 | 235 | 236 | 237 | 345 | 239 | 240 | 252 | 254 | 255 | 256 | 263 | 264 | 266 | 283 | 286 | 339

6.3.2 Algorytm symulowanego wyżarzania

Nazwa pliku: SA_rbg358.txt

Wynik: 1798

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

7 Bibliografia

- https://www.youtube.com/watch?v=wrkMM6a4S-U
- https://towardsdatascience.com/tabu-search-simply-explained-ee2852339d78
- https://docplayer.pl/106361694-Tabu-search-poszukiwanie-z-zakazami.html
- https://www2.imm.dtu.dk/courses/02719/tabu/4tabu2.pdf
- http://www.pi.zarz.agh.edu.pl/intObl/notes/IntObl_w2.pdf
- Wykłady – dr inż. Tomasz Kapłon