Politechnika Śląska
Wydział Matematyki Stosowanej
Kierunek Informatyka
Studia stacjonarne I stopnia

Projekt inżynierski

Algorytm symulowanego wyżarzania - zastosowanie w rozwiązywaniu zagadnień odwrotnych

Kierujący projektem:
dr inż. Adam Zielonka

Autor:
Kamil Kryus

Gliwice 2019

Projekt inżynierski:

Algorytm symulowanego wyżarzania - zastosowanie w rozwiązywaniu zagadnień odwrotnych

kierujący projektem: dr inż. Adam Zielonka

autor: Kamil Kryus

Podpis autora projektu

.....

Podpis kierującego projektem

.....

Oświadczenie kierującego projektem inżynierskim

Potwierdzam, że niniejszy projekt został przygotowany pod moim kierunkiem i kwalifikuje się do przedstawienia go w postępowaniu o nadanie tytułu zawodowego: inżynier.

Data Podpis kierującego projektem

Oświadczenie autora

Świadomy/a odpowiedzialności karnej oświadczam, że przedkładany projekt inżynierski na temat:
Algorytm symulowanego wyżarzania - zastosowanie w rozwiązywaniu zagadnień odwrotnych
został napisany przeze mnie samodzielnie.
Jednocześnie oświadczam, że ww. projekt:

- nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2000 r. Nr 80, poz. 904, z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym, a także nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
- nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów wyższej uczelni lub tytułów zawodowych.
- nie zawiera fragmentów dokumentów kopiowanych z innych źródeł bez wyraźnego zaznaczenia i podania źródła.

Podpis autora projektu

Kamil Kryus, nr albumu:246591, ... (podpis:) .....

Gliwice, dnia .....

Spis treści

Wstęp

Z problematyką wyznaczania optymalnego rozwiązania mamy do czynienia w wielu dziedzinach życia i nauki, np. minimalizując koszty inwestycji, maksymalizując zyski, szukając najkrótszego połączenia pomiędzy miastami. Szukając rozwiązania (zazwyczaj przybliżonego), zawsze dążymy do tego żeby było ono jak „najlepsze" (jak najbliższe dokładnemu) i zostało znalezione w rozsądnym czasie. W tym celu często korzysta się z algorytmów heurystycznych.

Metody heurystyczne są przybliżonymi metodami optymalizacyjnymi, ale otrzymane dzięki nim rezulaty są satysfakcjonujące. Otrzymując w ten sposób rozwiązanie możemy:

1. zaakceptować je, np. gdy dokładne rozwiązanie nie jest konieczne (np. kompresja obrazu),
2. zawęzić (istotnie) zakres i prowadzić dalsze poszukiwania.

Jednak aby metodę heurystyczną uznać za akceptowalną, chcemy żeby spełniała następujące wymagania:

- rozwiązanie jest możliwe do znalezienia przy rozsądnej liczby obliczeń (w przeliczeniu na ich koszt),
- otrzymane rozwiązanie powinno być bliskie optymalnemu,
- prawdopodobieństwo uzyskania złego rozwiązania powinno być „niskie".

1. Opis

Często w naukach technicznych możemy natrafić na zadania, które polegają na odtworzeniu niektórych parametrów modelu na podstawie danych będących wynikiem pewnych obserwacji. W odróżnieniu od tradycyjnych problemów, gdzie zaczynając od modelu i danych dochodzimy do rezultatów, w tego typu problemach dzieje się to odwrotnie. Tego typu problemy nazywa się problemami odwrotnymi.

Problemy odwrotne niestety są często źle postawione. Problemy, aby być zagadnieniami poprawnie postawionymi, muszą spełniać 3 wymagania:

1. rozwiązanie problemu musi istnieć,
2. każde rozwiązanie jest unikalne,
3. rozwiązanie zależy od danych oraz parametrów (np. małe zmiany w funkcjach wejścia powodują małe zmiany w rozwiązaniu).

Jednym z tego typów problemów (problemów odwrotnych) są odwrotne zagadnienia przewodnictwa ciepła. Przy posiadaniu niekompletnego opisu modelu matematycznego oraz funkcji opisującej rozkład temperatury w wybranych punktach, zadanie polega na rekonstrukcji niektórych brakujących parametrów modelu.

1.1. Cel

W pracy opisany i zaimplementowany zostanie algorytm symulowanego wyżarzania. Dla wybranych funkcji testowych zostały dobrane jego parametry, a finalnie zostanie on wykorzystany do rozwiązania odwrotnego zadania przewodnictwa ciepła.

W tym celu została stworzona w miarę możliwości uniwersalna aplikacja, która pozwala na znalezienie globalnego minimum kilku funkcji testowych wraz z zadanymi przez siebie parametrami oraz odtworzenie jednego z brakujących parametrów modelu matematycznego zadanego odwrotnego zadania przewodnictwa ciepła, przy podanym rozkładzie temperatur w wybranym punkcie.

2. Opis algorytmu symulowanego wyżarzania

Algorytm ten został stworzony wzorując się na zjawisku wyżarzania w metalurgii, które polega na nagrzaniu elementu stalowego do odpowiedniej temperatury, przetrzymaniu go w tej temperaturze przez pewien czas, a następnie powolnym jego schłodzeniu. Sam algorytm natomiast bazuje na metodach Monte-Carlo i w pewnym sensie może być rozważany jako algorytm iteracyjny.

Główną istotą i zarazem zaletą tego algorytmu jest wykonywanie pewnych losowych przeskoków do sąsiednich rozwiązań, dzięki czemu jest w stanie uniknąć wpadania w lokalne minimum. Algorytm ten najczęściej jest używany do rozwiązywania problemów kombinatorycznych, jak np. problemu komiwojażera.

2.1. Parametry

Początkowa konfiguracja

W tym kroku powinniśmy zainicjalizować naszą temperaturę pewną wartością oraz znaleźć początkowe, losowe rozwiązanie naszego problemu.

Temperatura

Temperatura jest zarówno czynnikiem iteracyjnym, jak i jest związana z funkcją prawdopodobieństwa zamiany „gorszego" rozwiązania na „lepsze". Zatem zakres temperatury powinien być taki, aby na początku działania naszego algorytmu dawał dużą możliwość zamian, a wraz z postępem procesu iteracyjnego te prawdopodobieństwo zamiany było bliskie zeru.

Końcowa temperatura

Jest to „bardzo niska” wartość. Temperatura osiągając taki poziom stanowi, iż proces wyżarzania się zakończył i rozwiązanie zostało znalezione. Wartość ta powinna być na tyle mała, by temperatura będąc niewiele większa prowadziła do bardzo niskiego prawdopodobieństwa, a jednocześnie nie wymagało to zbyt dużej ilości iteracji.

Powtarzanie pewną ilość razy dla zadanej temperatury

Wartość ta powinna być z góry ustalona i powinna dać nam możliwość sprawdzenia wielu sąsiadów obecnego rozwiązania, równocześnie nie powodując zbyt dużego obciążenia dla algorytmu. Parametr ten jednak jest najbardziej dostosowywalny ze względu na jego ”niezależność”.

Znajdowanie losowego sąsiada poprzedniego rozwiązania

Funkcja ta powinna nam pozwalać przejrzeć jak najszerszy zakres rozwiązań, a jednocześnie pozwolić na przeszukiwanie coraz to bliższych sąsiadów obecnie najlepszego rozwiązania, zatem warto uzależnić tą funkcję od stopnia zaawansowania procesu iteracyjnego.

Funkcja kosztu

Poprzez funkcję kosztu rozumiemy różnicę pomiędzy obecnie najlepszym rozwiązaniem, a nowym. Funkcja ta ma dodatkowe zastosowanie przy decydowaniu o zamianie gorszego rozwiązania na lepsze. Przy poszukiwaniu globalnego minimum wartość większa jest gorszym rozwiązaniem, dzięki czemu wynikiem tej funkcji jest zawsze liczba ujemna (przy decydowaniu o zamianie).

Prawdopodobieństwo zamiany P

Prawdopodobieństwo jest wykorzystywane przy decyzji zamiany nowego i gorszego rozwiązania, z wcześniejszym i lepszym.

Prawdopodobieństwo tej zamiany zależy od funkcji kosztu oraz obecnej temperatury. Prawdopodobieństwo zatem można przedstawić w następujący sposób:

$$P = \exp\left(\frac{\Delta E}{T}\right)$$

gdzie:

$$\Delta E - funkcja\ kosztu \tag{1}$$

T - obecna wartość temperatury

Prawdopodobieństwo to wraz ze spadkiem wartości funkcji kosztu maleje (gdyż jest zawsze ujemne), natomiast wyższa wartość temperatury zwiększa to prawdopodobieństwo. Decydując o tym czy powinniśmy zamienić nasze gorsze rozwiązanie z lepszym powinniśmy porównać obliczone prawdopodobieństwo z wartością losową zawierającą się w zakresie [0,1].

Chłodzenie temperatury

Szybkość chłodzenia temperatury nie powinna być zbyt duża, aby pozwolić algorytmowi na sprawdzenie jak największego zakresu możliwych rozwiązań, a jednocześnie niezbyt wolna, gdyż może to spowodować zbyt wolny spadek prawdopodobieństwa i zbyt częste akceptowanie gorszych (lub dużo gorszych) rozwiązań. W większości opracowań można spotkać ten proces, jako mnożnik temperatury w zakresie [0.8;0.99].

2.2. Kroki algorytmu

Algorytm ten można również przedstawić za pomocą listy kroków:

1. Zainicjalizuj początkową konfigurację
2. Dopóki temperatura > minimum, powtarzaj:
   (a) Powtórz zadaną ilość razy dla danej temperatury
      i. Znajdź losowo sąsiada poprzedniego rozwiązania
      ii. Sprawdź czy rozwiązanie jest lepsze od poprzedniego (funkcja kosztu)
         A. Jeżeli jest, zamień rozwiązania
         B. Jeżeli nie jest, zamień rozwiązania z pewnym prawdopodobieństwem P
   (b) Zmniejsz temperaturę

3. Funkcje testowe

Pomimo, iż algorytmy heurystyczne są dobrym wyborem wszędzie tam, gdzie ważny jest czas znalezienia rozwiązania, to przed skorzystaniem z danego algorytmu jesteśmy zmuszeni ustawić parametry algorytmu w taki sposób, by wynik był dostatecznie dokładny, a algorytm nie wykonywał niepotrzebnie obliczeń, zwłaszcza gdy większa dokładność nie jest nam potrzebna lub nie będzie stanowić większej różnicy w stosunku do już znalezionego wyniku. Dodatkową trudność stanowi ilość parametrów oraz to, iż każdy z nich może wpływać w inny sposób na złożoność obliczeniową oraz wynik, oraz parametry mogą być od siebie zależne. W opracowaniach naukowych rzadko kiedy można znaleźć wytyczne co do sposobu znalezienia odpowiednich parametrów do konkretnych problemów.

Starając się trzymać zasad dotyczących tworzenia dobrego algorytmu heurystycznego, przyjęliśmy kilka założeń, a następnie sukcesywnie poszukiwaliśmy odpowiednich wartości dla parametrów by (średni, 10-krotne powtórzenia) wynik był jak najlepszy, starając się zawężać zakres z czasem. Kiedy (średnie) wyniki były już zadowalające, sprawdzaliśmy jakość dobranych parametrów wykonując 100 razy algorytm z takimi samymi parametrami dla tego samego problemu, uzysując w prosty sposób procentową jakość algorytmu. W podsekcji "Dobieranie parametrów dla funkcji Rastrigina o 5 wymiarach" tabele przedstawią stopniowe dojście do parametrów dających zadowalające wyniki (zaokrąglone do 4 miejsca po przecinku), a następnie jakość tych parametrów dla danego problemu. Parametry dla innych problemów zostały zbadane w taki sam sposób i w tych sekcjach zostanie wspomniany jedynie wynik.

3.1. Funkcja kwadratowa dwóch parametrów

Jako pierwszą funkcję do testów przyjęliśmy funkcję kwadratową dwóch parametrów następującej postaci:

$$f(x, y) = x^2 + y^2$$

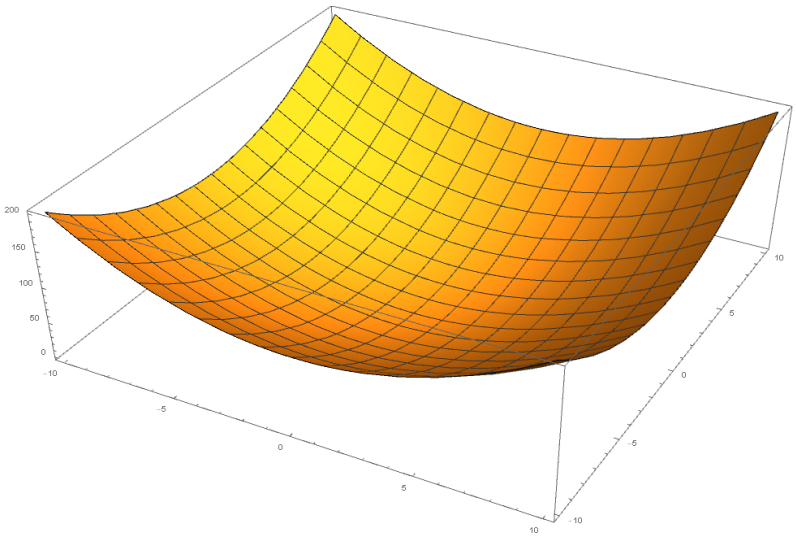
Funkcja ta jest funkcją parzystą i przyjmuje tylko wartości nieujemne. Na potrzeby projektu zakres dla tej funkcji został zawężony następująco:

$$x_i, y_i \in [-10, 10]$$

Posiada ona następujące globalne minimum:

$$f(0, 0) = 0$$

Funkcję prezentuje poniższy wykres:



Rysunek 1: Funkcja kwadratowa dwóch parametrów

3.1.1. Parametry dobrane dla funkcji kwadratowej dwóch parametrów

Poniższa tabela przedstawia parametry pozwalające na znalezienie rozwiązania z satysfakcjonującą jakością:

Parametr	**Wartość**
T_0	1
T_{end}	0.01
Chłodzenie	0.99
Iteracje	1
Skuteczność	100%

3.2. Funkcja Rastrigina

Funkcja Rastrigina jest funkcją ciągłą, skalowalną i multimodalną. Dzięki posiadaniu wielu minimum lokalnych, funkcja ta jest często stosowana w testowaniu algorytmów optymalizacyjnych. Przyjmuje ona następującą postać:

$$f(x) = An + \sum_{i=1}^{n}[x_i^2 - A\cos(2\pi x_i)]$$

gdzie:
A = 10,
n = ilość wymiarów

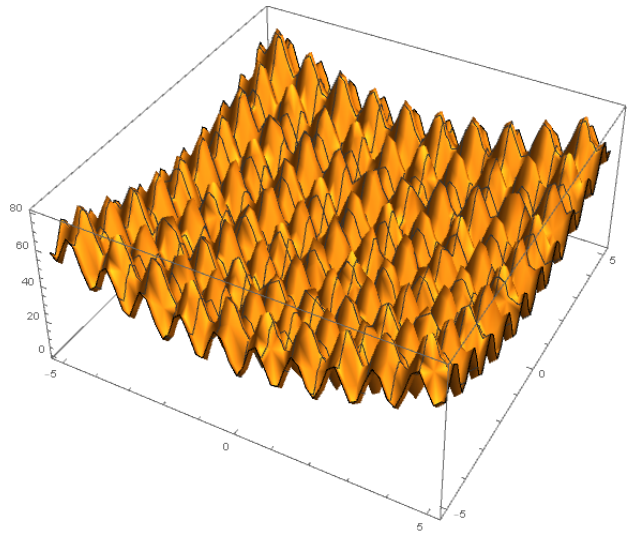
Wartości tej funkcji są nieujemne. Zakres wartości dla tej funkcji znajdziemy w przedziale:

$$x_i \in [-5.12, 5.12]$$

Posiada ona następujące globalne minimum:

$$f(0, ..., 0) = 0$$

By ujrzeć jej niektóre właściwości zaprezentowaliśmy jej wykres na poniższym obrazku:



Rysunek 2: Funkcja Rastrigina o 2 wymiarach

3.2.1. Dobieranie parametrów dla funkcji Rastrigina o 3 wymiarach

Po przeprowadzeniu testów zostały dobrane następujące parametry:

Parametr	**Wartość**
T_0	10
T_{end}	0.01
Chłodzenie	0.99
Iteracje	600
Skuteczność	100%

3.2.2. Dobieranie parametrów dla funkcji Rastrigina o 5 wymiarach

Przed rozpoczęciem testów przyjęto dwa założenia:

1. Końcowa temperatura została ustawiona na stałą wartość równą 0.01,
2. Stopień chłodzenia temperatury został ustawiony na 0.99.

W pierwszym etapie została sprawdzona temperatura w zakresie 2-10 oraz ilość iteracji w liczbie 100-1100.

Nr.	T_0	**Iteracje**	**Średnia rozwiązań**
1.	6	900	1,3201
2.	6	1100	1,3214
3.	8	1100	1,4151
4.	10	700	1,4182
5.	2	500	1,5137
6.	6	500	1,5151
7.	4	900	1,5182
8.	8	900	1,5191
9.	8	700	1,6145
10.	8	500	1,6169
11.	10	1100	1,6194
12.	10	900	1,7176
13.	2	1100	1,8087
14.	10	300	1,8099
15.	10	500	1,8106
16.	6	700	1,8174
17.	4	1100	1,8291
18.	2	700	1,9127
19.	6	300	2,0083
20.	2	900	2,2128
21.	4	500	2,2251
22.	4	700	2,3089
23.	2	300	2,3170
24.	8	300	2,4199
25.	4	300	2,5146
26.	10	100	2,8057
27.	6	100	3,0047
28.	2	100	3,0159
29.	8	100	3,2014
30.	4	100	3,2157

Powyższe wyniki nie dają zadowalających rezultatów. Można jednak zauważyć, iż w tym momencie badań temperatura nie ma aż takiego znaczenia, a większa ilość iteracji zdaje się dawać lepsze rezultaty. Zgodnie z założeniem, iż temperatura nie powinna być zbyt wysoka, postanowiliśmy dalej sprawdzać ten sam zakres temperatur i zwiększyć ilość iteracji około stukrotnie, co prezentuje następna tabela.

Nr.	T_0	**Iteracje (*1000)**	**Średnia rozwiązań**
1.	2	11	0,5162
2.	6	7	0,7157
3.	10	11	0,7165
4.	6	9	0,7189
5.	8	11	0,8120
6.	10	9	0,8208
7.	10	5	0,8244
8.	2	5	0,9171
9.	4	5	0,9184
10.	6	11	0,9273
11.	4	9	0,9278
12.	8	7	1,0155
13.	4	3	1,0177
14.	8	9	1,0224
15.	2	9	1,1158
16.	4	11	1,1203
17.	8	3	1,2153
18.	2	3	1,3114
19.	4	7	1,3136
20.	6	1	1,3147
21.	2	7	1,3152
22.	6	3	1,3194
23.	6	5	1,3202
24.	8	5	1,3222
25.	10	1	1,3252
26.	10	7	1,3270
27.	2	1	1,4161
28.	10	3	1,5143
29.	4	1	1,5184
30.	8	1	1,6182

Średnia rozwiązań najlepszego wyniku w tym teście wydawała się być obiecująca,

jednak sprawdzenie jakości takich parametrów zwróciło jakość równą 34% (przy stukrotnym uruchomieniu programu), co jest niskim wynikiem. Postanowiliśmy zwiększyć znowu zakres iteracji dziesięciokrotnie, co można zobaczyć w następnej tabeli.

Nr.	T_0	**Iteracje (* 1000)**	**Średnia rozwiązań**
1.	10	90	0,0252
2.	10	100	0,1215
3.	10	80	0,1216
4.	4	60	0,2178
5.	8	70	0,2200
6.	4	80	0,2206
7.	6	100	0,2208
8.	4	70	0,2209
9.	6	90	0,2242
10.	8	80	0,2257
11.	2	80	0,2266
12.	8	100	0,2279
13.	4	100	0,2337
14.	4	90	0,3147
15.	10	60	0,3199
16.	4	50	0,3222
17.	2	100	0,3224
18.	2	70	0,3247
19.	2	50	0,3248
20.	10	70	0,3249
21.	8	90	0,4161
22.	6	50	0,4199
23.	8	50	0,4257
24.	6	80	0,5203
25.	10	50	0,5257
26.	2	60	0,5264
27.	6	60	0,5308
28.	2	90	0,6167
29.	8	60	0,6255
30.	6	70	0,8194

Otrzymane wyniki sugerują, iż najlepsze wyniki dla podanych zakresów można

otrzymać przy temperaturze równej 10 i bardzo wysokich ilościach iteracji. W następnym kroku sprawdziliśmy jakość rozwiązań dla temperatury równej 10 i iteracji w zakresie przedstawionej w tabeli.

T_0	**Iteracje (*1000)**	**Jakość** [%]
10	10	29
10	20	42
10	30	45
10	40	57
10	50	69
10	60	66
10	70	68
10	80	74
10	90	80
10	100	81

Jakość rzędu 75-80% wydaje się być zadowalająca, dlatego uznaliśmy, że proces poszukiwania parametrów dla algorytmu symulowanego wyżarzania dla tego problemu został zakończony. Kolejna tabela przedstawia ostatecznie wybrane parametry dla tego problemu.

Parametr	**Wartość**
T_0	10
T_{end}	0.01
Chłodzenie	0.99
Iteracje	90000
Skuteczność	80%

3.3. Funkcja Rosenbrocka

Funkcja ta jest funkcją ciągłą, skalowalną i jednomodalną.

$$f(x) = \sum_{i=1}^{n-1} [100 \left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2]$$

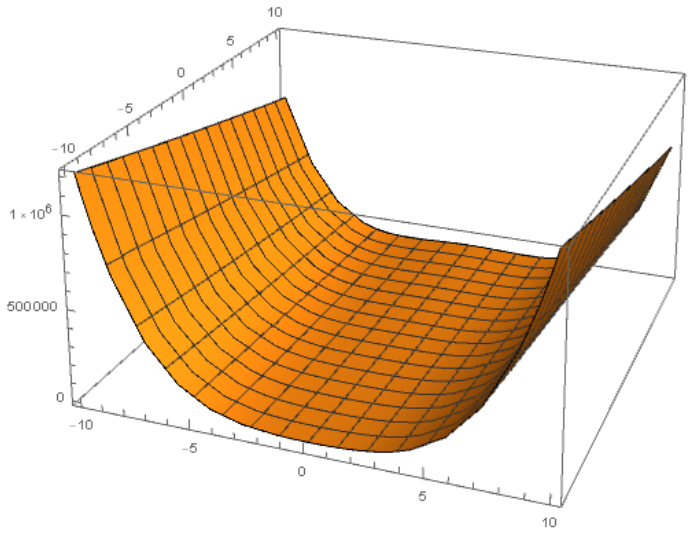
Funkcja ta również przyjmuje wyłącznie wartości nieujemne. Na potrzeby projektu wartości argumentów dla tej funkcji zostały zawężone do poniższego zakresu:

$$x_i \in [-10, 10]$$

Posiada ona następujące globalne minimum:

$$f(1, ..., 1) = 0$$

Poniższy wykres prezentuje jej wygląd w zadanym zakresie:



Rysunek 3: Funkcja Rosenbrocka o 2 wymiarach

3.3.1. Parametry dobrane dla funkcji Rosenbrocka o 3 wymiarach

Poniższa tabela przedstawia parametry, które pozwalały na uzyskanie satysfakcjonującego rozwiązania z dostatecznie „dużą” pewnością ich otrzymania:

Parametr	**Wartość**
T_0	10
T_{end}	0.01
Chłodzenie	0.99
Iteracje	500
Skuteczność	100%

4. Implementacja

Na realizację algorytmu symulowanego wyżarzania składa się implementacja kilku metod, które zostały przedstawione w roździale 2.1.2 i będą tutaj omówione i/lub zostanie przedstawiony ich kod.

4.1. Używane parametry i zmienne

Wraz z zainicjalizowaniem obiektu symulowanego wyżarzania, ustawianych jest kilka parametrów na wejście, a konkretniej problem do rozwiązania i parametry samego algorytmu. W moim programie nazywane są: **Function**, **Arguments**, **Arguments2**, **Iterations**, **BeginingTemperature**, **EndingTemperature**, **Cooling**, **SatisfactionSolutionValue**.

Function jest identyfikatorem referencji do obiektu problemu. Każdy problem musi dziedziczyć po klasie abstrakcyjnej „TestingFunction", co zapewnia uniwersalność stosowania algorytmu symulowanego wyżarzania oraz zapewnia nasz algorytm, iż implementacja samego problemu będzie posiadać pewne cechy (jak np. jawnie określoną ilość wymiarów).

Arguments jest właściwością w postaci tablicy liczb zmiennoprzecinkowych, które zawierają argumenty dla obecnie najlepszego rozwiązania problemu.

Arguments2 jest również tablicą liczb zmiennoprzecinkowych, jednak przechowuje ona wartości argumentów dla tymczasowego rozwiązania. Jest tego samego rozmiaru, co właściwość **Arguments**.

Iterations jest liczbą wewnętrznych iteracji. Tyle razy algorytm będzie szukał sąsiadów najlepszego rozwiązania, zanim obniży temperaturę.

BeginingTemperature jest to początkowa wartość temperatury, od której rozpoczyna się proces poszukiwań rozwiązania.

EndingTemperature jest liczbą, którą obniżana temperatura (zmienna **temperature**) musi osiągnąć, by zakończyć działanie algorytmu.

Cooling to liczba zmiennoprzecinkowa, w każdym globalnym kroku algorytmu zmienna **temperature** jest mnożona przez tą wartość. Jest ona mniejsza od 1, więc temperatura powoli się obniża.

SatisfactionSolutionValue jest minimalną liczbą, jaką rozwiązanie musi osiągnąć, aby wynik poszukiwania rozwiązania był dla satysfakcjonujący. Jest to zmienna opcjonalna, algorytm nadal będzie działać, gdy nie poda się jej wartości.

W programie również używam kilku pomocnicznych zmiennych:

temperature to zmienna, która przetrzymuje obecną liczbę stanowiącą temperaturę. Jest ona używana przy warunkach globalnej iteracji oraz przy funkcji prawdopodobieństwa. Wraz z postępem iteracji maleje.

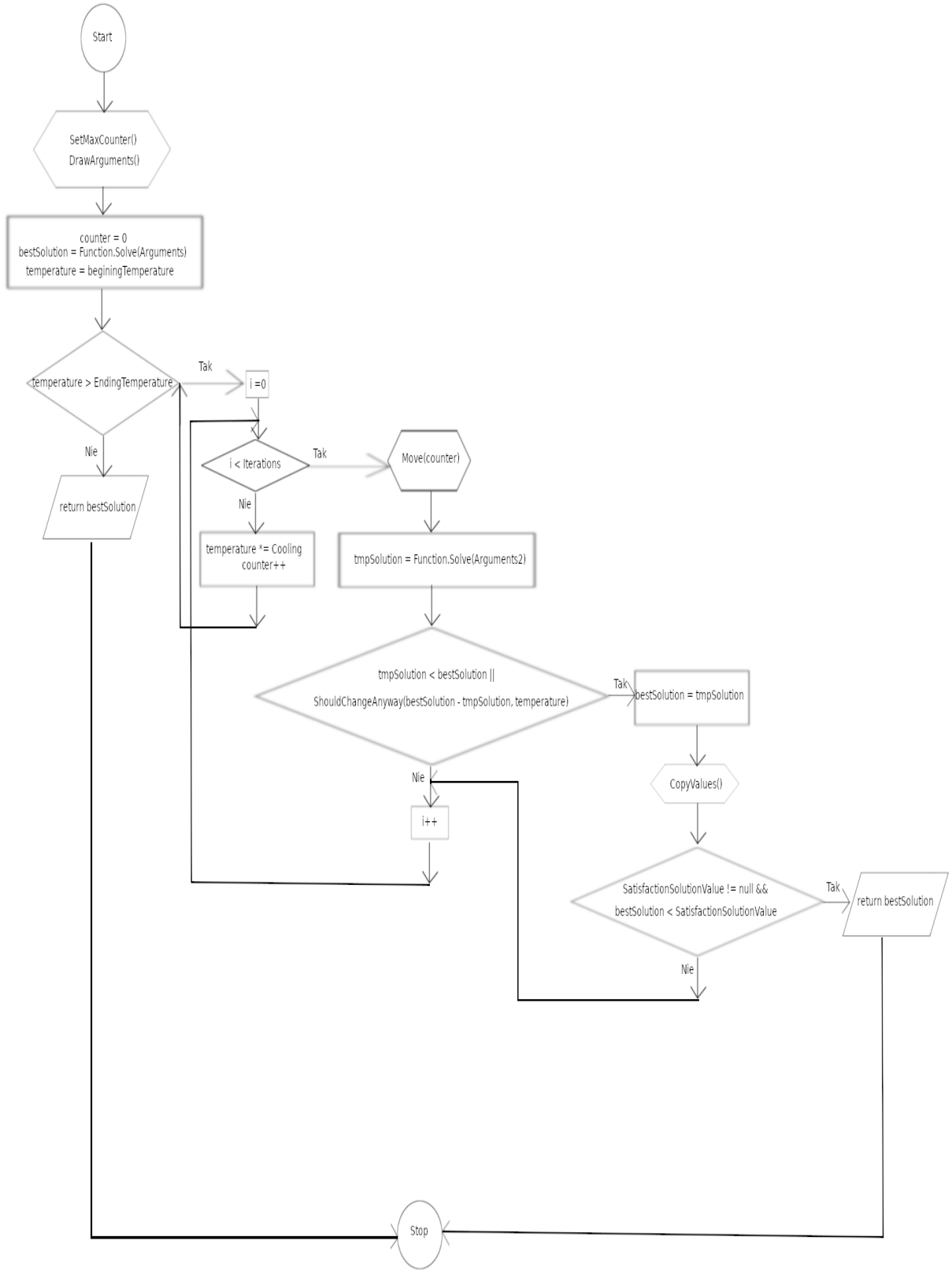
bestSolution to liczba zmiennoprzecinkowa, która jest obecnie najlepszym wynikiem rozwiązania. Finalnie będzie ona najlepszym rozwiązaniem całego problemu.

tmpSolution jest tymczasowym wynikiem rozwiązania (wynikiem rozwiązania problemu dla parametrów ze zmiennej **Arguments2**).

counter jest liczbą oznaczającą obecną, globalną iterację.

4.2. Schemat blokowy

Przygotowaliśmy również schemat blokowy reprezentujący poszczególne kroki i scenariusze w procesie poszukiwania rozwiązania zadanego problemu, co można zobaczyć na poniższym rysunku.



Rysunek 4: Schemat blokowy algorytmu symulowanego wyżarzania

4.3. Implementacja metod i algorytmu

Funkcja SetMaxCounter()

Metoda ta symuluje proces obniżania temperatury w celu obliczenia maksymalnej ilości globalnych iteracji.

```
private void SetMaxCounter()
{
    maxCounter = 0;
    double tmpTemperature = BeginingTemperature;
    while (tmpTemperature > EndingTemperature)
    {
        tmpTemperature *= Cooling;
        maxCounter++;
    }
}
```

Funkcja DrawArguments()

W metodzie tej losuję początkowe argumenty (właściwość **Arguments**) z przedziału zadanego w danym problemie.

Funkcja Move()

W tym kroku najpierw obliczany jest pozostały procent iteracji do ukończenia procesu. Następnie biorąc 80% pełnej puli możliwych wartości problemu, mnożymy ją przez pozostały procent iteracji (i przypisujemy do zmiennej value). Dalej, w pętli, każdemu argumentowi tymczasowego rozwiązania (właściwość **Arguments2**), przypisywana jest suma odpowiedniego argumentu najlepszego rozwiązania (aby był to sąsiad najlepszego rozwiązania) oraz losowa liczba z przedziału [-value, value]. Wraz z postępem iteracji zakres ten jest coraz węższy, ale rozwiązanie powinno też już być bliskie optymalnemu. Na końcu walidujemy nowy argument do podanego zakresu.

```
private void Move(int counter)
{
    double leftTemperatureCoolingTimes = maxCounter - counter;
```

```
    double leftPercent = leftTemperatureCoolingTimes / maxCounter;

    double domainValue = (Function.RightBound - Function.LeftBound);
    double value = (0.8 * domainValue) * (leftPercent);
    for (int i = 0; i < AmountOfArguments; i++)
    {
        double newValue = Arguments[i] +
RandomGenerator.Instance.GetRandomDoubleInDomain(-value, value);
        if (newValue < Function.LeftBound)
        {
            newValue = Function.LeftBound;
        }
        if (newValue > Function.RightBound)
        {
            newValue = Function.RightBound;
        }
        Arguments2[i] = newValue;
    }
}
```

Funkcja ShouldChangeAnyway()

Jest to prosta implementacja funkcji prawdopodobieństwa, o której mowa była w rozdziale 2.1.1.

Funkcja CopyValues()

Ze względu, iż język C# traktuje tablicę jako obiekt, to tablica jest typem referencyjnym i konieczne jest skopiowanie wartości ze zmiennej **Arguments2** do zmiennej **Arguments**.

Implementacja algorytmu

Opisane metody są wykorzystywane w poszczególnych krokach samego algorytmu. Po obliczeniu maksymalnej ilości iteracji, algorytm losuje pierwsze rozwiązanie. Następnie w pętli i następnej zagnieżdzonej pętli, szuka sąsiada obecnego najlepszego rozwiązania. Zamienia nowe rozwiązanie ze starym jeżeli zostały spełnione odpo-

wiednie warunki. Jeżeli nowe rozwiązanie spełnia kolejny warunek, to kończy program zwracając najlepsze rozwiązanie. Jeżeli nie, wychodzi z zagnieżdzonej pętli, zmniejsza zmienną odpowiedzialną za temperaturę i jest to koniec kroków w jednej pełnej, „globalnej" iteracji. Jeżeli program nie osiągnie satysfakcjonującego rozwiązania, a proces obniżania temperatury zakończy się, zwróci rozwiązanie, które udało mu się znaleźć kończąc tym samym program.

```
public double Solve()
{
    SetMaxCounter();
    int counter = 0;

    DrawArguments();
    double bestSolution = Function.Solve(Arguments);

    double temperature = BeginingTemperature;
    while (temperature > EndingTemperature)
    {
        for (int i = 0; i < Iterations; i++)
        {
            Move(counter);
            double tmpSolution = Function.Solve(Arguments2);

            if (tmpSolution < bestSolution ||
ShouldChangeAnyway(bestSolution - tmpSolution, temperature))
            {
                bestSolution = tmpSolution;
                CopyValues();
                if(SatisfactionSolutionValue != null &&
 bestSolution < SatisfactionSolutionValue)
                {
                    return bestSolution;
                }
            }
```

```
        }
        temperature *= Cooling;
        counter++;
    }
    return bestSolution;
}
```

5. Zastosowanie algorytmu w rozwiązywaniu odwrotnego zagadnienia przewodnictwa ciepła

Posiadając gotowy model matematyczny tradycyjnego problemu przewodnictwa ciepła możliwe jest zasymulowanie tego procesu i otrzymanie serii pomiarów temperatur. Obliczone pomiary temperatur można użyć do rozwiązania problemu odwrotnego.

5.1. Tradycyjny problem

5.1.1. Parametry

Proces obliczania rozkładu temperatur wymaga podania kilku parametrów, które zostaną teraz wyjaśnione.
Delegaty oznaczone f, g i h są opisem warunków brzegowych, odpowiednio rozkładem temperatury w czasie t_0 oraz rozkładem temperatur na początku i końcu procesu zależnym od czasu. W projekcie funkcje te przyjmują następującą postać:

$$f(x) = 0.5x^2 + 0.5$$

$$g(t) = t + 0.5$$

$$h(t) = t + 1$$

gdzie:
$x \in [0, a]$,
$t \in [0, T]$,
$a, T = 1$.

Liczby nx i nt są maksymalną liczbą węzłów, na którą dzielimy odpowiednie osie rozkładu temperatur (odpowiednio x i t). W projekcie przyjmują one następujące

wartości:

$$nx = 15$$

$$nt = 480$$

Pozostałe parametry wynikają z równania przewodnictwa ciepła:
c - ciepło właściwe,
rho - gęstość,
$lambda$ - współczynnik przewodności ciepła.

gdzie:
$c, rho, lambda = 1$

5.1.2. Obliczanie rozkładu temperatur

Następujący fragment kodu przedstawia proces obliczania rozkładu temperatur.

```
public double[][] Solve()
{
    double[][] temp = new double[this.nt + 1][];
    for (int i = 0; i < this.nt + 1; i++)
    {
        temp[i] = new double[this.nx + 1];
        for (int j = 0; j < this.nx + 1; j++)
        {
            temp[i][j] = 0;
        }
    }
    double hx = a / nx;
    if (this.tau / (hx * hx) >= 0.5)
    {
        Console.WriteLine("Niestabline");
        return null;
    }
    var x = new double[this.nx + 1];
    for (int i = 0; i < this.nx + 1; i++)
    {
```

```
        x[i] = i * hx;
    }
    for (int i = 0; i < this.nx + 1; i++)
    {
        temp[0][i] = this.f(x[i]);
    }
    for (int i = 0; i < nt + 1; i++)
    {
        temp[i][0] = this.g((i) * this.tau);
        temp[i][this.nx] = this.h((i) * this.tau);
    }
    var wsp = this.lambda * this.tau / (hx * hx * this.c * this.rho);
    for (int j = 1; j < this.nt + 1; j++)
    {
        for (int i = 1; i < this.nx; i++)
        {
            temp[j][i] = wsp * (temp[j - 1][i - 1] - 2.0 * temp[j - 1][i] +
 temp[j - 1][i + 1]) + temp[j - 1][i];
        }
    }
    return temp;
}
```

5.2. Problem odwrotny

Posiadając rozkład temperatur, pobraliśmy 10 jej pomiarów w 80 % maksymalnej iloświęzłów w równych odstępach czasu. Następująca tabela przedstawia temperatury wykorzystywane w obliczeniach:

Nr.	t_i
1.	0,82
2.	0,92
3.	1,02
4.	1,12
5.	1,22
6.	1,32
7.	1,42
8.	1,52
9.	1,62
10.	1,72

Posiadając model problemu tradycyjnego oraz obliczone pomiary temperatur, zadanie polegało na odtworzeniu jednego z warunków granicznych (h) za pomocą równania kwadratowego:

$$\overline{h}(t) = p^2 t + qt + s$$

w taki sposób, by obliczony na nowo rozkład temperatur przy pomocy nowej funkcji i pobrany zestaw danych jak najbardziej przypominał oryginalny. By odtworzona funkcja była równa pierwotnej, parametry powinny przyjąć następujące wartości:

$$p = 0$$

$$q = 1$$

$$s = 1$$

5.3. Wykorzystanie algorytmu heurystycznego

Do odnalezienia parametrów równania kwadratowego został użyty algorytm symulowanego wyżarzania. Algorytm w procesie iteracyjnym sprawdzał jakie wartości p, q i s pozwalały na uzyskanie jak najmniejszego błędu odtworzenia (liczonego jako sumę wartości bewzględnej różnic odpowiednich pomiarów temperatur), co przedstawia poniższy wzór:

$$\sum_{i=1}^{n} |x_i - z_i|$$

gdzie:
x_i - oryginalny i-ty pomiar
z_i - odtworzony i-ty pomiar
n - liczba pomiarów.

Poszukiwania parametrów zostały ograniczone do następującego przedziału:

$$p, q, s \in [-10, 10]$$

5.3.1. Implementacja rozwiązania problemu odwrotnego

Klasa odpowiadająca za problem odwrotny posiada metodę Solve(), która zwraca błąd odtworzenia funkcji. Funkcja ta przyjmuje dowolną ilość parametrów (ze względu na uniwersalność metody Solve() w problemach), gdzie tutaj pierwsze trzy argumenty są parametrami funkcji kwadratowej. W implementacji tej metody jest używanych kilka zmiennych/metod, które oznaczają:

$GetInverseProblemTemperatureMeasurements()$, pomocnicza metoda, która zwraca odpowiednie pomiary temperatur dla odtworzonego problemu tradycyjnego (z funkcją $\overline{h}$ jako odtworzona funkcja kwadratowa),
$Measurements$ - oryginalny zestaw pomiarów.

Implementacja metody Solve() ma następującą postać:

```
public override double Solve(params double[] values)
{
    this.p = values[0];
    this.q = values[1];
    this.s = values[2];

    double sum = 0;
```

```
    double[] tmpMeasurements = GetInverseProblemTemperatureMeasurements();
    for (int i = 0; i < Measurements.Length; i++)
    {
        sum += Math.Abs(Measurements[i] - tmpMeasurements[i]);
    }
    return sum;
}
```

Metoda ta jest wykorzystywana przez algorytm symulowanego wyżarzania do obliczania jak „najniższej" wartości błędu odtworzenia i to właśnie algorytm przekazuje wartości parametrów metodzie Solve().

5.3.2. Parametry algorytmu

Ze względu na czas oczekiwania znalezienia pojedynczego optymalnego rozwiązania problemu, ilość prób sprawdzających jakość dobranych parametrów została ograniczona do 5. Poniższa tabela przedstawia dobrane argumenty, które pozwalają na rozwiązanie problemu z satysfakcjonującym wynikiem błędu odtworzenia.

Nr.	T_0	T_{end}	**Iteracje**	**Chłodzenie**
1.	20	0.001	35000	0.99

5.4. Rezultaty

Oprócz prób odtworzenia funkcji granicznej dla dokładnych pomiarów, zostały również podjęte próby jej odtworzenia dla pomiarów, które posiadają kolejno 1, 2 i 5% zakres błędu. Każda wartość temperatury została obliczona w następujący sposób dla poszczególnych pomiarów:

$$t_{ip} = rand\left(t_i - \left(\frac{t_i p}{100}\right), t_i + \left(\frac{t_i p}{100}\right)\right)$$

gdzie:

p - wartość procentu zakresu błędu,

t_i - oryginalny i-ty pomiar,

t_{ip} - i-ty pomiar z p-procentowym zakresem błędu,

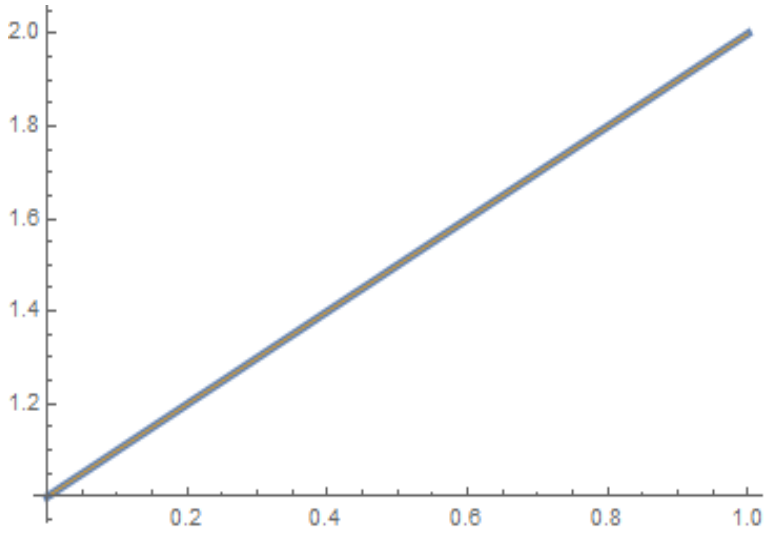
$rand$ - funkcja losująca liczbę z zakresu (od, do).

5.4.1. Dokładne pomiary temperatur

Następna tabela przedstawia wyniki osiągniete przy pomocy podanych parametrów (liczby zaokrąglono do 4 miejsca po przecinku) przy niezmienionych pomiarach temperatur.

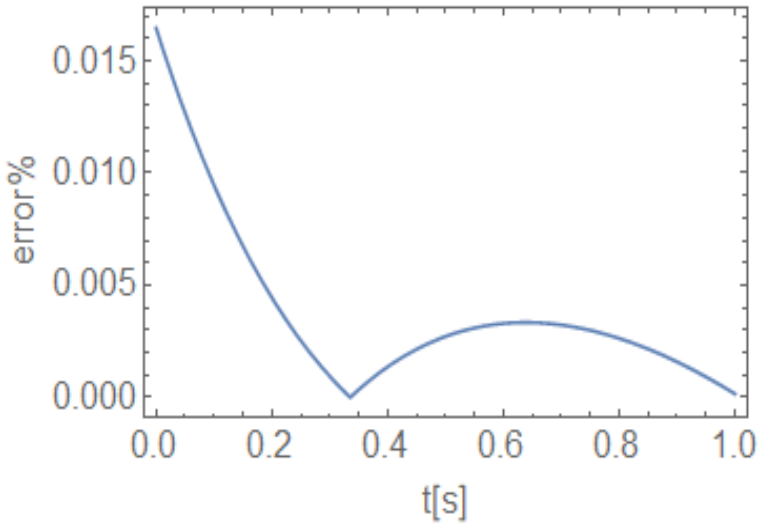
Nr.	**Błąd**	**P**	**Q**	**S**
1.	6,9416 * 10^{-4}	-0,0012	1,0015	0,9996
2.	9,4350 * 10^{-4}	0,0028	0,9971	1,0006
3.	6,1092 * 10^{-4}	-0,0013	1,0015	0,9996
4.	5,3948 * 10^{-4}	-0,0001	1,0005	0,9998
5.	8,7191 * 10^{-4}	-0,0026	1,0026	0,9995

Różnicę pomiędzy pierwotną funkcją graniczną, a odtworzoną (poprzez uśrednienie parametrów p, q i s) została również przedstawiona na wykresie (żółty kolor oznacza oryginał, niebieska odtworzoną, niebieska posiada większą grubość w celu zauważenia nachodzenia na siebie linii).



Rysunek 5: Porównanie początkowej funkcji granicznej z odtworzoną

Przygotowano również wykres pokazujący błąd bezwzględny obliczeń, co przedstawia poniższy rysunek:



Rysunek 6: Błąd bezwzględny obliczeń

5.4.2. 1% błąd pomiarowy temperatur

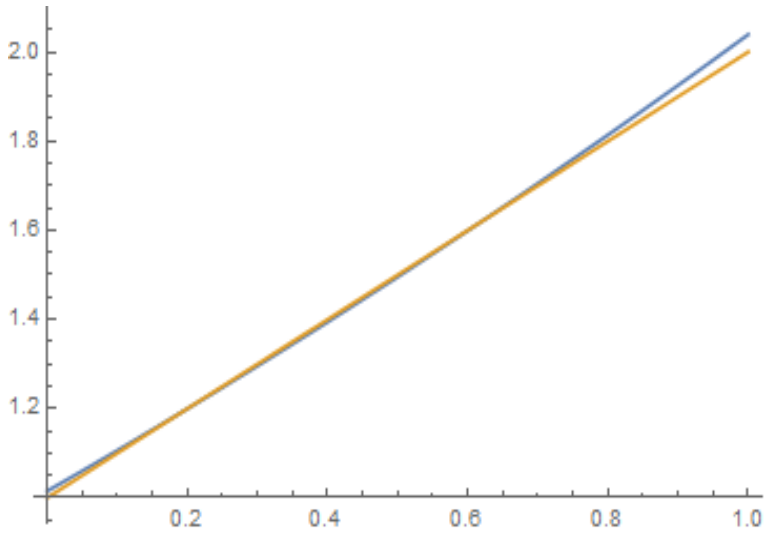
Pomiary wykorzystywane przy obliczeniach z 1% zakresem błędu pomiarowego wyglądają następująco:

Nr.	t_i
1.	0,8254
2.	0,9273
3.	1,0230
4.	1,1173
5.	1,2246
6.	1,3169
7.	1,4207
8.	1,5227
9.	1,6239
10.	1,7363

Przeprowadzenie procesu odtwarzania funkcji granicznej dało następujące rezultaty (zaokrąglone do 4 miejsca po przecinku):

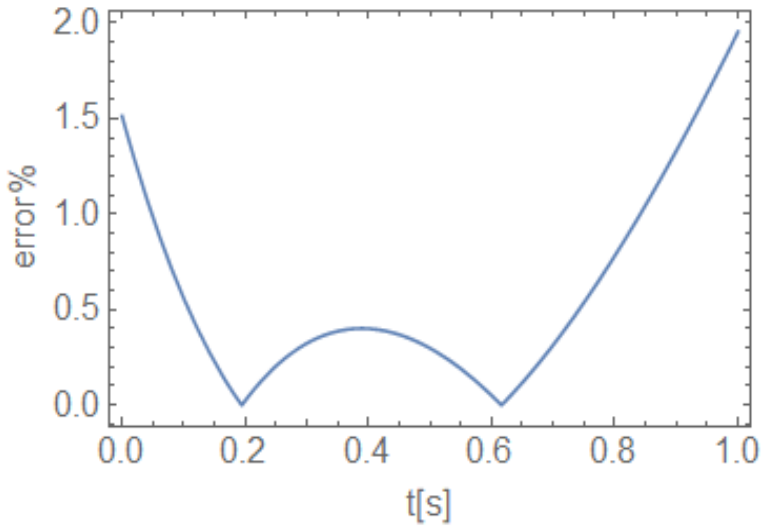
Nr.	**Błąd**	**P**	**Q**	**S**
1.	2,7209 * 10^{-2}	0,1190	0,8952	1,0189
2.	2,3787 * 10^{-2}	0,1152	0,9157	1,0140
3.	2,7362 * 10^{-2}	0,1500	0,8860	1,0063
4.	2,3056 * 10^{-2}	0,1173	0,8890	1,0178
5.	2,3617 * 10^{-2}	0,1304	0,9021	1,0187

Różnica pomiędzy oryginalną funkcją, a uzyskaną (poprzez uśrednienie parametrów) została przedstawiona na wykresie:



Rysunek 7: Porównanie początkowej funkcji granicznej z odtworzoną (z 1 % zakresem błędu pomiarów)

Następujący rysunek przedstawia błąd bezwzględny obliczeń:



Rysunek 8: Błąd bezwzględny obliczeń

5.4.3. 2% błąd pomiarowy temperatur

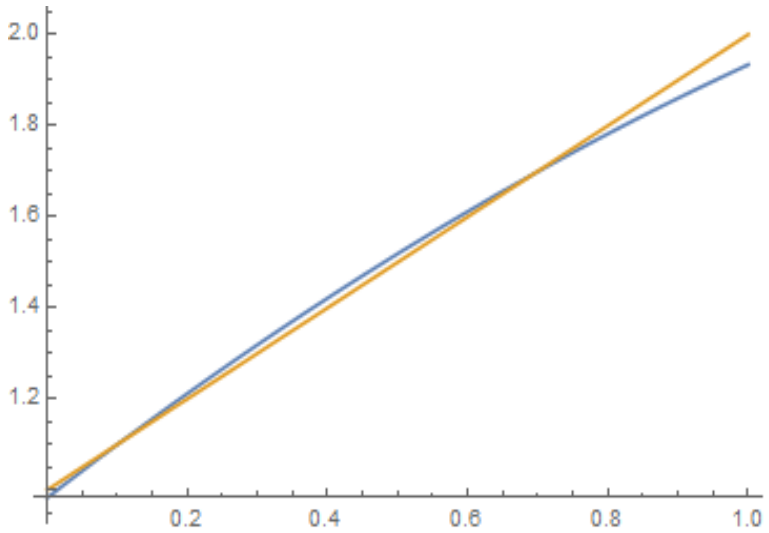
Przy obliczaniu parametrów odtwarzanej funkcji przy 2% zakresie błędu pomiarowego wykorzystano następujące pomiary temperatur:

Nr.	t_i
1.	0,8327
2.	0,9225
3.	1,0161
4.	1,1337
5.	1,2405
6.	1,3304
7.	1,4376
8.	1,5097
9.	1,6126
10.	1,6993

Obliczone parametry wraz z błędem prezentują się następująco (przy zaokrągleniu do 4 miejsca po przecinku):

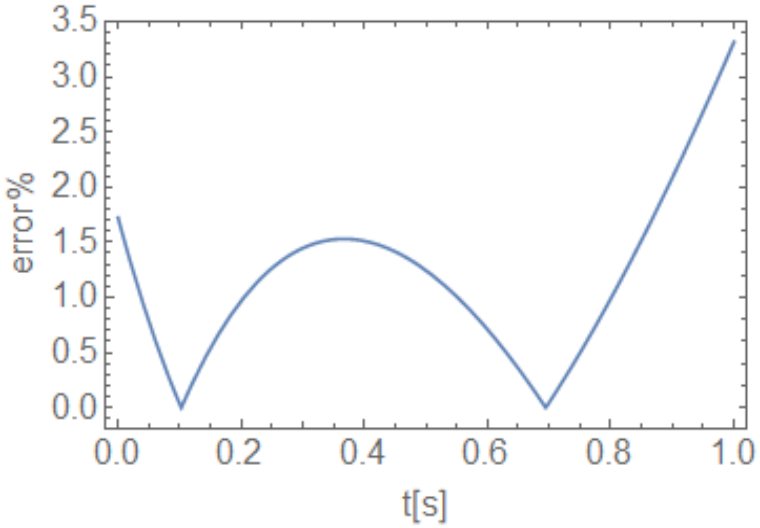
Nr.	**Błąd**	**P**	**Q**	**S**
1.	6,8528 * 10^{-2}	-0,3642	1,2993	0,9513
2.	6,5422 * 10^{-2}	-0,2570	1,1971	0,9992
3.	5,8742 * 10^{-2}	-0,2352	1,1853	0,9895
4.	6,0410 * 10^{-2}	-0,1531	1,1175	0,9937
5.	6,0850 * 10^{-2}	-0,1995	1,1643	0,9801

Różnicę pomiędzy funkcjami pierwotną i obliczoną (poprzez uśrednienie parametrów) przedstawia poniższy wykres:



Rysunek 9: Porównanie początkowej funkcji granicznej z odtworzoną (z 2 % zakresem błędu pomiarów)

Błąd bezwzględny obliczeń został przedstawiony poniżej:



Rysunek 10: Błąd bezwzględny obliczeń

5.4.4. 5% błąd pomiarowy temperatur

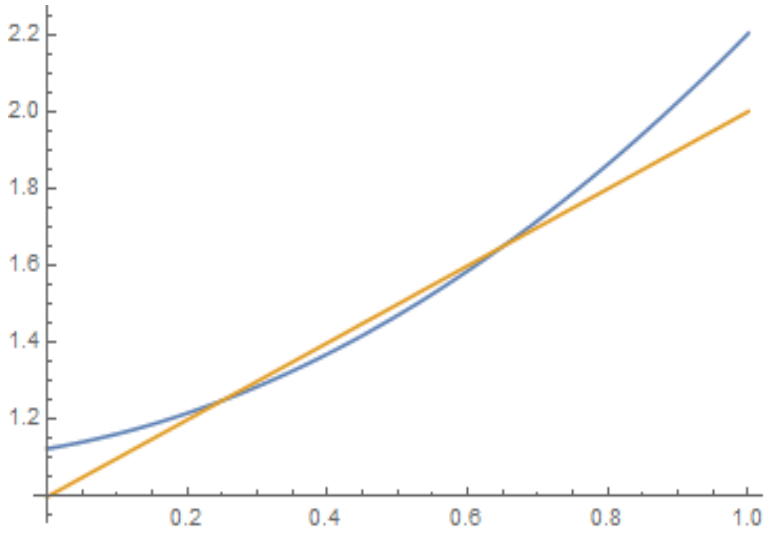
Pomiary wykorzystywane przy obliczeniach z 5% zakresem błędu pomiarowego wyglądają następująco:

Nr.	t_i
1.	0,8051
2.	0,9300
3.	1,0515
4.	1,1398
5.	1,2139
6.	1,2906
7.	1,3880
8.	1,5877
9.	1,6267
10.	1,7959

Wykorzystanie algorytmu dało następujące rezultaty (przy zaokrągleniu liczb do 4 miejsca po przecinku):

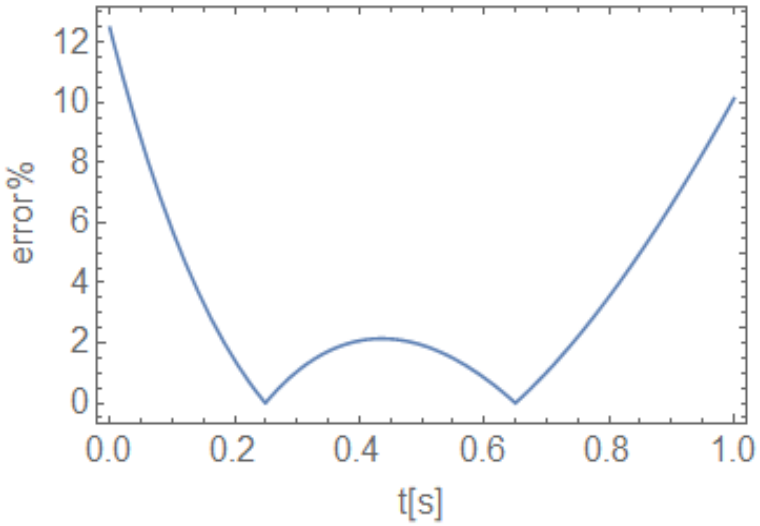
Nr.	**Błąd**	**P**	**Q**	**S**
1.	0,2031	0,8678	0,2034	1,1597
2.	0,2049	0,6563	0,4295	1,0985
3.	0,2030	0,7652	0,3167	1,1265
4.	0,2060	0,8133	0,2428	1,1277
5.	0,2041	0,7516	0,3426	1,1119

Wykresy obu funkcji zostały przedstawione poniżej:



Rysunek 11: Porównanie początkowej funkcji granicznej z odtworzoną (z 5 % zakresem błędu pomiarów)

Poniższy rysunek przedstawia błąd bewzględny obliczeń:



Rysunek 12: Błąd bezwzględny obliczeń

6. Narzędzia i technologie

W procesie tworzenia aplikacji zdecydowaliśmy się na użycie kilku rozwiązań, które pozwoliły na bezpieczną i przejrzystą pracę w kolejnych jego etapach.

6.1. Metodyka pracy

6.1.1. System kontroli wersji

System kontroli wersji posiada wiele zalet, m.in.: bezpieczeństwo, możliwość pracy w kilku miejscach/urządzeniach nad tym samym problemem, łatwą możliwość przywrócenia poprzedniej wersji, czy wreszcie, inspekcję jakości i poprawności kodu.
W moim projekcie skorzystałem z systemu kontroli Git, a repozytorium można znaleźć na portalu github.com.

6.1.2. Github Project Management

Pomimo, iż praca w pojedynkę nie wymagała ode mnie zaawansowanego zarządzania projektem i konieczności organizacji pracy, zdecydowałem się na użycie narzędzia pozwalającego na taką pracę. Podzielenie projektu na mniejsze zadania pozwoliło mi wydzielić poszczególne i odrębne sektory pracy, widzieć postępujący progres i łatwo odnaleźć się w aktualnie wykonywanym zadaniu. W tym celu skorzystałem z Github Project Management, który pozwala na proste zarządzanie zadaniami.

6.1.3. Środowisko programistyczne

Do implementacji projektu użyłem środowiska Microsoft Visual Studio Community 2017, które to zostało stworzone przez firmę Microsoft i pozwala na programowanie konsolowe oraz z graficznym interfejsem użytkownika (zarówno aplikacje desktopowe, jak i strony internetowe).
Dobra znajomość i przejrzystość tego środowiska programistycznego pozwoliła mi

skupić się na rozwiązywaniu problemu, omijając problem zapoznawania się z nowym narzędziem.

6.1.4. Mathematica

Mathematica jest programem opartym na systemie obliczeń symbolicznych oraz numerycznych. Program ten jest dość popularny wśród naukowców ze względu na wiele zalet, jak np. wydajność czy rozpięte możliwości wizualizacji danych. Mathematica jest programem komercyjnym, dlatego stworzenie wykresów do tego projektu oparłem na licencji wydziału Matematyki Stosowanej.

6.2. Użyte technologie

6.2.1. C#

Język programowania C# należy do obiektowych języków programowania, którego koncepcja opiera się na tworzeniu klas, które poprzez swoją zawartość (m.in. właściwości czy metody) mogą być reprezentowane poprzez obiekty i każde operacje są wykonywane poprzez nie. W projekcie korzystam z języka C# w wersji 7.0, która w momencie rozpoczęcia pracy była aktualna. Dobra znajomość tego języka pozwoliła mi nie zważać na problemy w znajomości składni czy funkcji i skupić się bezpośrednio na implementacji algorytmów, dobraniu odpowiednich parametrów dla poszczególnych funkcji testowych oraz lepszym przetestowaniu całej funkcjonalności.

6.2.2. Wolfram Language

Język ten służy głównie do programowania obliczeń matematycznych i programowania funkcjonalnego w programie Mathematica. Język ten, wraz z oprogramowaniem Mathematica, pozwalają m.in. na: operacje na macierzach, rozwiązywanie równań różniczkowych czy prezentowanie danych za pomocą wykresów. Z tej ostatniej funkcjonalności skorzystałem tworząc wykresy funkcji testowych.

7. Podsumowanie

Celem tej pracy inżynierskiej było stworzenie aplikacji, która pozwoliłaby rozwiązać zadany problem przewodnictwa ciepła poprzez zastosowanie algorytmu symulowanego wyżarzania. Stworzono więc program, który poprzez prosty interfejs graficzny pozwala na użycie tego algorytmu heurystycznego w kilku funkcjach testowych oraz zadanym odwrotnym problemie przewodnictwa ciepła i znalezienia ich optymalnego rozwiązania. Dodatkowo użytkownik jest w stanie zmodyfikować (domyślne i sugerowane) parametry algorytmu i sprawdzić jak wpłynie to na rezultat wykonywania programu.

Realizacja założeń projektu wymagała przeprowadzenia badań w kwestii odpowiedniego doboru parametrów dla poszczególnych problemów i ich jakości oraz przetestowania i zoptymalizowania samego działania algorytmu symulowanego wyżarzania. Dobrane parametry dla poszczególnych problemów pozwalają na stosunkowo szybkie i poprawne znalezienie optymalnego rozwiązania danego problemu, a wyniki przeprowadzonych testów rozwiązania odwrotnego zadania przewodnictwa ciepła na zbliżone odtworzenie jednego z brakujących parametrów modelu matematycznego.

Dalsze prace nad programem powinny rozwinąć projekt o możliwość zadania problemu tradycyjnego i odwrotnego przewodnictwa ciepła, stając się tym samym jeszcze bardziej użytecznym. Pomimo, iż algorytmy heurystyczne znacznie przyspieszają znalezienie optymalnego rozwiązania, to jednak bywa to proces czasochłonny, stąd informacja o estymowanym czasie do ukończenia procesie może być użyteczna dla użytkownika. Ze względu na zasoby czasowe i błąd ludzki, warto by było zautomatyzować proces przeprowadzania testów parametrów nowych problemów oraz dla już istniejących. Rozszerzenie projektu o dodatkowe algorytmy heurystyczne pozwoliłoby porównywać ze sobą jakość rozwiązań i złożoność pamięciową i czasową poszczególnych algorytmów, dzięki czemu w praktycznych celach można by dobierać odpowiedni algorytm do konkretnego problemu.

Literatura

[1] http://iswiki.if.uj.edu.pl/iswiki/images/2/20/AiSD_22._Symulowane_wy%C5%BCarzanie_%28problem_komiwoja%C5%BCera%29.pdf [dostęp: 20 sierpnia 2018]

[2] M. Duque-Anth, *Constructing efficient simulated annealing algorithms*, [w:] „Discrete Applied Mathematics", 1997 nr 77/2, s. 139-159 Dostępny w Internecie: https://www.sciencedirect.com/science/article/pii/S0166218X96001321 [dostęp 20 sierpnia 2018]

[3] http://wikizmsi.zut.edu.pl/uploads/archive/5/5e/20140303092223!OzWSI_I_S1_W1.pdf [dostęp: 25 sierpnia 2018]

[4] http://www.wikizmsi.zut.edu.pl/uploads/e/eb/OzWSI_I_S1_c1.pdf [dostęp: 25 sierpnia 2018]

[5] https://en.wikipedia.org/wiki/Test_functions_for_optimization#Test_functions_for_single-objective_optimization [dostęp: 30 października 2018]

[6] H. Abiyev, M. Tunay, *Optimization of High-Dimensional Functions through Hypercube Evaluation*, Dostępny w Internecie:https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4538776/ [dostęp: 13 listopada 2018]

[7] F. Rothlauf, *Optimization Problems*, [w:] *Design of Modern Heuristics: Principles and Application*, wyd. Springer-Verlag Berlin Heidelberg, 2011, s. 7-44, ISBN 978-3-540-72961-7 Dostępny w Internecie: https://pdfs.semanticscholar.org/b333/0f96d1a937fc2c63b3294729cfea30826134.pdf [dostęp: 27 listopada 2018]

[8] R. Martí, G. Reinelt, *Heuristic Methods*, [w:] *The Linear Ordering Problem, Exact and Heuristic Methods in Combinatorial Optimization*, wyd. Springer-Verlag Berlin Heidelberg, 2011, s. 17-40, ISBN 978-3-642-16728-7

[9] http://prac.im.pwr.edu.pl/∼plociniczak/lib/exe/fetch.php?media=odwrotne.pdf [dostęp: 10 grudnia 2018]

[10] https://www.math.unl.edu/∼scohn1/8423/wellposed.pdf [dostęp: 10 grudnia 2018]

[11] E. Hetmaniok, A. Zielonka, D. Słota, *Zastosowanie algorytmu selekcji klonalnej do odtworzenia warunku brzegowego trzeciego rodzaju*, [w:] „Zeszyty naukowe Politechniki Śląskiej", 2012 nr 2/1874

[12] E. Hetmaniok, D. Słota, A. Zielonka, *Application of the Ant Colony Optimization Algorithm for Reconstruction of the Thermal Conductivity Coefficient*, [w:] *Swarm and Evolutionary Computation*, wyd. Springer-Verlag Berlin Heidelberg, 2012, s. 240-248, ISBN 978-3-642-29352-8