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 itd. 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 można skorzystać z algorytmów heurystycznych.

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

1. zaakceptować je, np. gdy dokładne rozwiązanie nie jest konieczne np. w kompresji obrazu,
2. zawęzić (istotnie) zakres i prowadzić dalsze poszukiwania w oparciu o inny algorytm.

Metody heurystyczne uznaje się za akceptowalne, jeśli spełniają następujące wymagania:

- rozwiązanie jest możliwe do znalezienia przy „rozsądnej" liczbie obliczeń,
- 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 bezpośrednich problemów, gdzie zaczynając od modelu i danych dochodzimy do rezultatów, w tego typu problemach dzieje się to odwrotnie. Tego typu zadania nazywa się problemami odwrotnymi.

Problemy odwrotne niestety są często źle postawione. Problemy, aby być zagadnieniami poprawnie postawionymi, muszą spełniać następujące 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).

Przykładem tego typu problemów są odwrotne zagadnienia przewodnictwa ciepła. Przy niepełnym opisie modelu matematycznego, ale znając wartości funkcji w wybranych punktach (zazwyczaj pomiarowych), rozwiązanie zadania odwrotnego polega na rekonstrukcji 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 minimum globalnego funkcjonału, sprawdzona najpierw dla wybranych funkcji testowych, a następnie wykorzystana do odtworzenia brakujących parametrów opisujących model matematyczny przykładowego odwrotnego zadania przewodnictwa ciepła, przy zadanych wartościach temperatury w wybranym punkcie pomiarowym.

2. Opis algorytmu symulowanego wyżarzania

Algorytm ten został stworzony wzorując się na zjawisku wyżarzania metali, które polega na nagrzaniu elementu stalowego do ustalonej temperatury początkowej, 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, takich jak np. problem komiwojażera.

Zanim jednak przejdziemy do opisu samego algorytmu, powinniśmy najpierw sformułować rozważane zadanie optymalizacyjne.

2.1. Zadania optymalizacyjne

W dalszych rozważaniach skupimy się na wyznaczeniu minimum funkcji f określonej w $\mathbb{R}^n$.

$$f(x) \to \mathbb{R},$$

gdzie: $x = (x_1, x_2, ..., x_n) \in D \subset R^n$

Bez straty ogólności możemy założyć, że:
$D = [a_1, b_1] \times [a_2, b_2] \times ... \times [a_n, b_n]$

Wówczas zadanie minimalizacji funkcji f oznaczymy:

$$f(x) \to min.$$

2.2. Opis algorytmu

Początkowa konfiguracja

W tym kroku powinniśmy zainicjalizować temperaturę początkową (pewną zadaną wartość) oraz znaleźć początkowe, w tym wypadku losowo położone, rozwiązanie problemu.

Temperatura

Wraz z upływem czasu (kolejne iteracje) ulega zmianie i jest czynnikiem wpływającym na prawdopodobieństwo zamiany „gorszego" rozwiązania na „lepsze". Zatem zakres temperatury powinien być taki, aby na początku działania algorytmu dawał dużą możliwość zamian, a wraz z postępem procesu iteracyjnego prawdopodobieństwo zamiany było bliskie zeru.

Końcowa temperatura

Temperatura osiągając taki poziom stanowi, iż proces wyżarzania się zakończył i rozwiązanie zostało znalezione. Wartość ta powinna być na tyle mała, żeby jej „niewielka" zmiana nie miała praktycznego wpływu na zmianę rozwiązania (prawdopodobieństwo zmiany bliskie zeru), a jednocześnie jej za mała wartość nie prowadziła do zbyt dużej liczby iteracji.

Powtarzanie zadanej ilości iteracji dla stałej temperatury

Proces iteracyjny bez zmiany temperatury wykonuje się określoną (ustaloną jako parametr algorytmu) ilość razy.

Znajdowanie losowego sąsiada poprzedniego rozwiązania

Etap ten ma pozwolić przejrzeć jak najszerszy zakres rozwiązań, a jednocześnie pozwolić na przeszukiwanie coraz to bliższych sąsiadów obecnie „najlepszego" rozwiązania, zatem proces ten należy uzależnić od stopnia zaawansowania procesu iteracyjnego, tak że wraz ze wzrostem iteracji zawężeniu ulegnie zakres obszaru przeszukiwania.

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 „lepszym".

Prawdopodobieństwo tej zamiany zależy od funkcji kosztu oraz obecnej temperatury. Prawdopodobieństwo zamiany określone jest wzorem:

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

gdzie: ΔE - funkcja kosztu, T - obecna wartość temperatury.

Prawdopodobieństwo to wraz ze spadkiem wartości funkcji kosztu maleje (gdyż funkcja ta jest zawsze ujemna), natomiast wyższa wartość temperatury je zwiększa. Decydując o tym, czy powinniśmy zamienić nasze „gorsze" rozwiązanie z „lepszym", powinniśmy porównać obliczone prawdopodobieństwo z wartością losową z przedziału [0, 1] o rozkładzie równomiernym.

Chłodzenie temperatury

Szybkość chłodzenia temperatury nie powinna być „zbyt duża", aby pozwolić algorytmowi na sprawdzenie szerokiego 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, że ten proces określa mnożnik tempa spadku temperatury ustalony w zakresie [0.8;0.99].

Z powyższego opisu wynika, że algorytm ten zależy od czterech parametrów:

– temperatury początkowej (T_0),

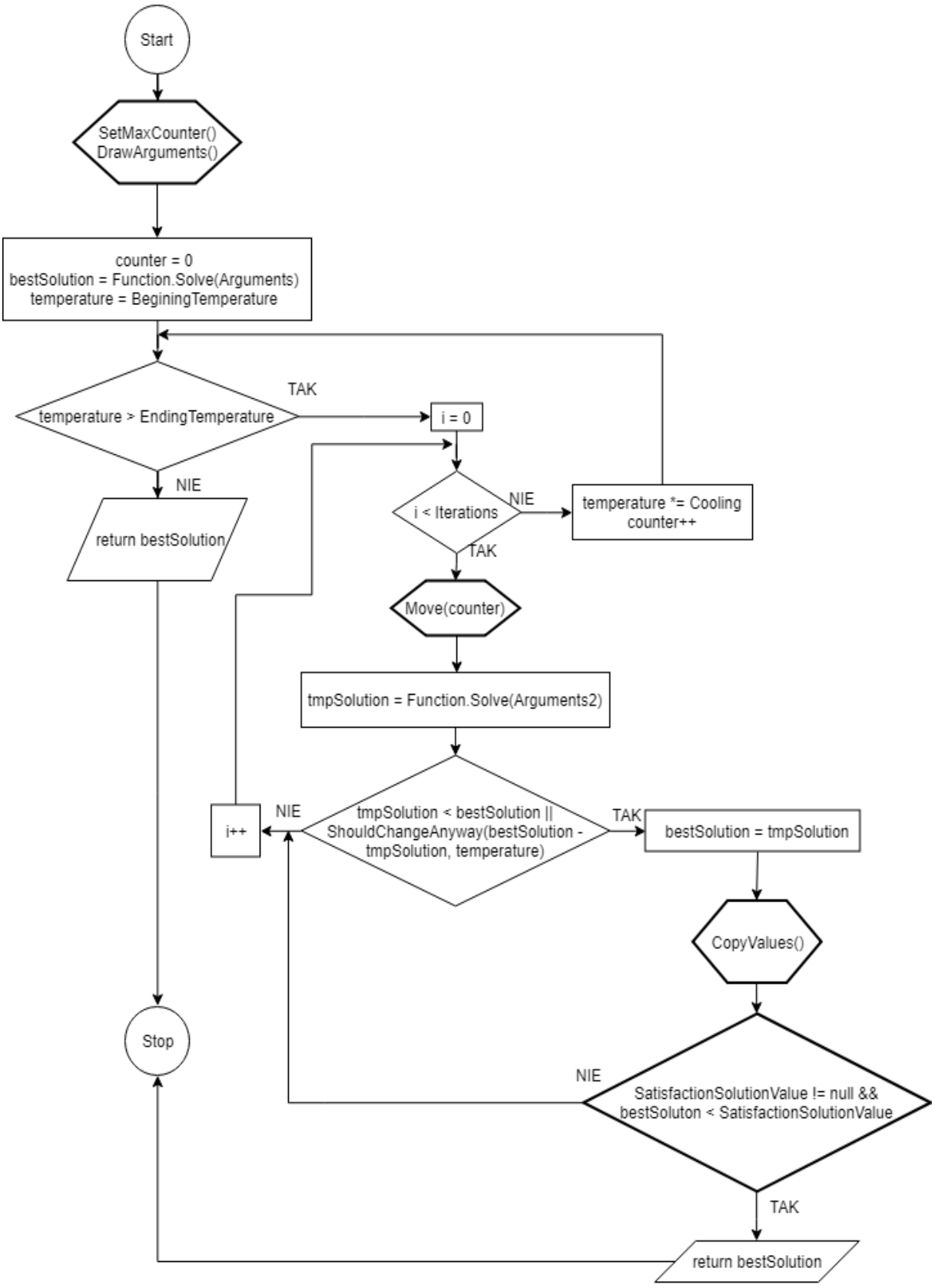
– temperatury końcowej (T_{end}),

– liczby wewnętrznych iteracji (It),

– parametru chłodzenia (k).

2.3. Schemat blokowy

Na rysunku 1 zamieszczony jest schemat blokowy reprezentujący poszczególne kroki i scenariusze w procesie poszukiwania rozwiązania zadanego problemu.



Rysunek 1: Schemat blokowy algorytmu symulowanego wyżarzania

2.4. Kroki algorytmu

Algorytm symulowanego wyżarzania można przedstawić schematycznie:

I Inicjalizacja: T_0, T_{end}, It, k

1. T = T_0
2. x_{best} = losowe współrzędne z dziedziny zadania

II Zasadnicza część algorytmu:

3. Dopóki T > T_{end}:
   (a) Wyznacz „sąsiada” x_{tmp} punktu x_{best} (wraz z liczbą iteracji szerokość zakresu wyznaczania „sąsiada” maleje)
      (a.a) Jeśli f(x_{best}) > f(x_{tmp}):
         A) $x_{best} = x_{tmp}$
      (a.b) w przeciwnym wypadku:
         A) ΔE = f(x_{best}) - f(x_{tmp})
         B) $P = \exp\left(\frac{\Delta E}{T}\right)$
         C) q = losowa liczba z przedziału [0, 1]
         D) Jeśli $P > q$, to:
            $x_{best} = x_{tmp}$
      (a.c) Wróć do (a) It razy
   (b) Schłodzenie: $T = kT$
   (c) Powrót do 3.

3. Narzędzia i technologie

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

3.1. Metodyka pracy

3.1.1. 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.

3.1.2. Github Project Management

Pomimo, iż praca realizowana przez jedną osobę 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.

3.1.3. 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.

3.1.4. 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.

3.1.5. Draw.io

Jest to aplikacja webowa, pozwalająca w prosty i przejrzysty sposób stworzyć darmowe reprezentacje wykresów, przepływów pracy czy schematów blokowych, oraz zarządzanie nimi.

3.2. Użyte technologie

3.2.1. 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.

3.2.2. 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.

4. Funkcje testowe

Pomimo, iż algorytmy heurystyczne są dobrym wyborem wszędzie tam, gdzie brak jest dodatkowych informacji o optymalizowanej funkcji, a jedynie znane są jej wartości, to przed użyciem danego algorytmu musimy dobrać parametry algorytmu w taki sposób, by proces optymalizacji prowadził do uzyskania dostatecznie dokładnych wyników, 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. W opracowaniach naukowych trudno znaleźć wytyczne co do sposobu wyznaczania odpowiednich parametrów, związane to jest z ich zależnością od rozwiązywanego problemu.

W pierwszej kolejności został przeprowadzony test dla szerokiego zakresu każdego z parametrów i że względu na dużą ilość prób, wykonano po 10 powtórzeń algorytmu dla tych samych jego parametrów. W ten sposób uzyskaliśmy „obiecujące" zestawy parametrów wyjściowych i następnie już dla nich zostały przeprowadzone kolejne próby, gdzie wykonano po 100 powtórzeń algorytmu dla tych samych, doprecyzowanych parametrów. W przypadku funkcji testowej Rastrigina w R^5 proces wyznaczania parametrów został opisany dokładnie, a w pozostałych przypadkach zostały zamieszczone wyselekcjonowane parametry algorytmu (wraz z ilością głównych iteracji oznaczoną symbolem It_m). Ze względu na chronologię, funkcje testowe zostaną przedstawione od najłatwiejszych, do bardziej wymagających.

4.1. Funkcja kwadratowa dwóch zmiennych

Jako pierwszą funkcję do testów przyjęliśmy funkcję kwadratową dwóch zmiennych postaci:

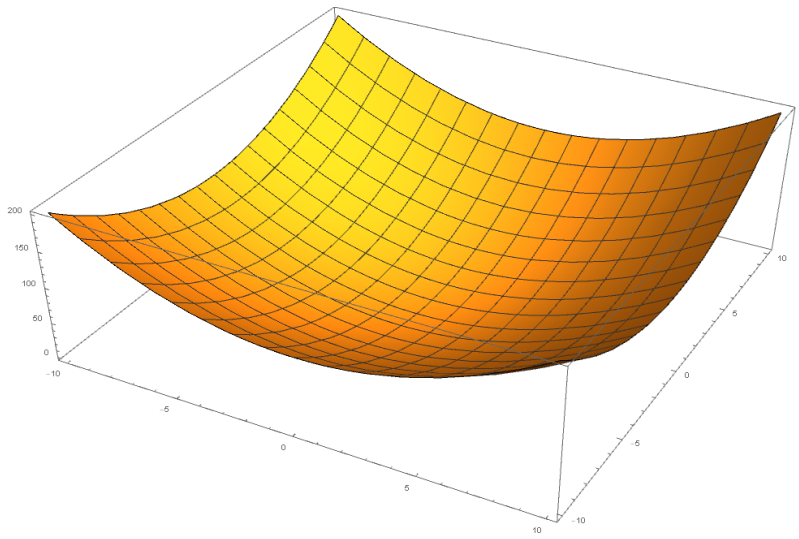
$$f(x, y) = x^2 + y^2, \qquad (x, y) \in R^2.$$

Wybór tak „łatwej” funkcji testowej padł ze względu na to, żeby upewnić się czy algorytm został poprawnie zaimplementowany, gdyż proces wyznaczania ekstremum tej funkcji jest „bardzo prosty” i nie wymaga dużego nakładu obliczeń.

Funkcja ta przyjmuje tylko wartości nieujemne i posiada minimum globalne w punkcie (0, 0). Na potrzeby testów dziedzina tej funkcji została zawężona:

$$(x, y) \in [-10, 10]^2$$

Funkcję $f(x, y)$ prezentuje poniższy rysunek:



Rysunek 2: Funkcja kwadratowa dwóch zmiennych

4.1.1. Parametry dobrane dla funkcji kwadratowej dwóch zmiennych

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

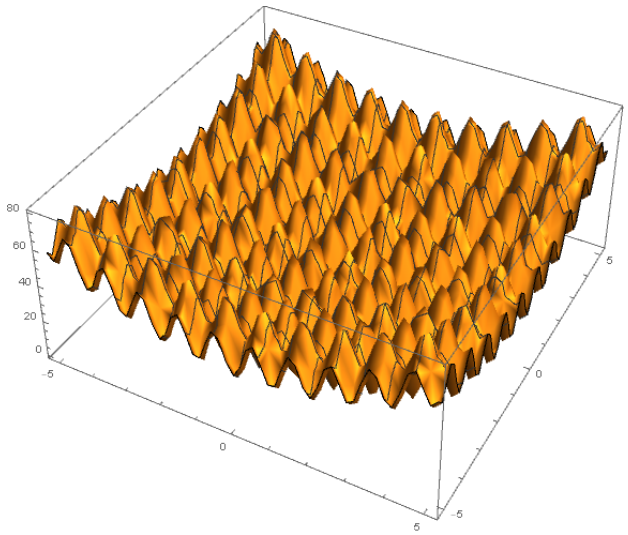
Parametr	Wartość
T_0	1
T_{end}	0.01
It	1
k	0.99
It_m	459
Skuteczność	100%

4.2. Funkcja Rastrigina

Funkcja Rastrigina jest funkcją ciągłą, skalowalną i multimodalną (jej wykres dla n=2, została przedstawiona na rysunku 2). Dzięki posiadaniu wielu minimum lokalnych, funkcja ta jest często stosowana w testowaniu algorytmów optymalizacyjnych. Funkcja Rastrigina określona jest wzorem:

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

gdzie: n = ilość wymiarów, x =$(x_1, ..., x_n)$, A = 10, $x_i \in [-5.12, 5.12]$.



Rysunek 3: Funkcja Rastrigina o 2 wymiarach

Wartości tej funkcji są nieujemne. Posiada ona następujące globalne minimum:

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

4.2.1. Dobieranie parametrów dla funkcji Rastrigina o 3 wymiarach

Po przeprowadzeniu testów zostały wyselekcjonowane następujące parametry, które prowadzą do uzyskania satysfakcjonujących rezultatów:

Parametr	**Wartość**
T_0	4
T_{end}	0.01
It	600
k	0.99
It_m	597
Skuteczność	99%

4.2.2. Dobieranie parametrów dla funkcji Rastrigina o 5 wymiarach

Przed rozpoczęciem testów przyjęto dwa założenia:

- końcowa temperatura została ustawiona na stałą wartość równą 0.01,
- stopień chłodzenia temperatury został ustawiony na 0.99.

W pierwszym etapie została sprawdzona temperatura dla wartości $T_0 \in \{2, 4, ..., 10\}$ oraz liczbie wewnętrznych iteracji $It \in \{100, 300, ..., 1100\}$.

Tabela 1: Wyniki testów parametrów dla podanych zakresów, gdzie T_0 jest temperaturą początkową, It jest ilością iteracji, a $\overline{f}_{min}$ średnią rozwiązań dla podanych parametrów uzyskaną w 10 próbach

T_0	It	$\overline{f}_{min}$	T_0	It	$\overline{f}_{min}$
10	1100	1,6194	6	1100	1,3214
10	900	1,7176	6	900	1,3201
10	700	1,4182	6	700	1,8174
10	500	1,8106	6	500	1,5151
10	300	1,8099	6	300	2,0083
10	100	2,8057	6	100	3,0047
8	1100	1,4151	4	1100	1,8291
8	900	1,5191	4	900	1,5182
8	700	1,6145	4	700	2,3089
8	500	1,6169	4	500	2,2251
8	300	2,4199	4	300	2,5146
8	100	3,2014	4	100	3,2157

Wyniki uzyskane dla powyższych zestawów parametrów 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.

Tabela 2: Wyniki testów parametrów dla podanych zakresów, gdzie T_0 jest temperaturą początkową, It jest ilością iteracji, a $\overline{f}_{min}$ średnią rozwiązań dla podanych parametrów uzyskaną w 10 próbach

T_0	$It(\times 1000)$	$\overline{f}_{min}$	T_0	$It(\times 1000)$	$\overline{f}_{min}$
10	11	0,7165	6	11	0,9273
10	9	0,8208	6	9	0,7189
10	7	1,3270	6	7	0,7157
10	5	0,8244	6	5	1,3202
10	3	1,5143	6	3	1,3194
10	1	1,3252	6	1	1,3147
8	11	0,8120	4	11	1,1203
8	9	1,0224	4	9	0,9278
8	7	1,0155	4	7	1,3136
8	5	1,3222	4	5	0,9184
8	3	1,2153	4	3	1,0177
8	1	1,6182	4	1	1,5184

Średnia rozwiązań najlepszego wyniku w tym teście wydawała się być obiecująca, jednak sprawdzenie jakości takich parametrów zwróciło jakość równą 34%, co nie jest satysfakcjonujące. Postanowiliśmy zwiększyć ponownie zakres iteracji dziesięciokrotnie, co można zobaczyć w następnej tabeli (tabela 3).

Tabela 3: Wyniki testów parametrów dla podanych zakresów, gdzie T_0 jest temperaturą początkową, It jest ilością iteracji, a $\overline{f}_{min}$ średnią rozwiązań dla podanych parametrów

T_0	$It(\times 1000)$	$\overline{f}_{min}$	T_0	$It(\times 1000)$	$\overline{f}_{min}$	T_0	$It(\times 1000)$	$\overline{f}_{min}$
10	100	0,1215	6	100	0,2208	2	100	0,3224
10	90	0,0252	6	90	0,2242	2	90	0,6167
10	80	0,1216	6	80	0,5203	2	80	0,2266
10	70	0,3249	6	70	0,8194	2	70	0,3247
10	60	0,3199	6	60	0,5308	2	60	0,5264
10	50	0,5257	6	50	0,4199	2	50	0,3248
10	40	0,5223	6	40	0,3182	2	40	0,4396
10	30	0,8243	6	30	0,4311	2	30	0,6237
10	20	0,3277	6	20	0,5213	2	20	0,4180
10	10	0,5312	6	10	0,7173	2	10	0,8205
8	100	0,2279	4	100	0,2337	0,1	100	0,5275
8	90	0,4161	4	90	0,3147	0,1	90	0,6247
8	80	0,2257	4	80	0,2206	0,1	80	0,6163
8	70	0,2200	4	70	0,2209	0,1	70	0,6202
8	60	0,6255	4	60	0,2178	0,1	60	0,8208
8	50	0,4257	4	50	0,3222	0,1	50	0,6103
8	40	0,1310	4	40	0,5243	0,1	40	0,7318
8	30	0,5167	4	30	0,5192	0,1	30	0,8203
8	20	0,8323	4	20	0,3261	0,1	20	1,1167
8	10	1,1295	4	10	0,7265	0,1	10	0,8112

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.

Tabela 4: Wyniki testów parametrów dla podanych zakresów, gdzie T_0 jest temperaturą początkową, It jest ilością iteracji, a $\overline{f}_{min}$ średnią rozwiązań dla podanych parametrów

T_0	$It(\times 1000)$	$\overline{f}_{min}$	T_0	$It(\times 1000)$	$\overline{f}_{min}$
10	100	0,1215	4	100	0,2337
10	90	0,0252	4	90	0,3147
10	80	0,1216	4	80	0,2206
10	70	0,3249	4	70	0,2209
10	60	0,3199	4	60	0,2178
10	50	0,5257	4	50	0,3222
10	40	0,5223	4	40	0,5243
10	30	0,8243	4	30	0,5192
10	20	0,3277	4	20	0,3261
10	10	0,5312	4	10	0,7265
8	100	0,2279	2	100	0,3224
8	90	0,4161	2	90	0,6167
8	80	0,2257	2	80	0,2266
8	70	0,2200	2	70	0,3247
8	60	0,6255	2	60	0,5264
8	50	0,4257	2	50	0,3248
8	40	0,1310	2	40	0,4396
8	30	0,5167	2	30	0,6237
8	20	0,8323	2	20	0,4180
8	10	1,1295	2	10	0,8205
6	100	0,2208	0,1	100	0,5275
6	90	0,2242	0,1	90	0,6247
6	80	0,5203	0,1	80	0,6163
6	70	0,8194	0,1	70	0,6202
6	60	0,5308	0,1	60	0,8208
6	50	0,4199	0,1	50	0,6103
6	40	0,3182	0,1	40	0,7318
6	30	0,4311	0,1	30	0,8203
6	20	0,5213	0,1	20	1,1167
6	10	0,7173	0,1	10	0,8112

T_0	$It(\times 1000)$	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
It	90000
k	0.99
It_m	688
Skuteczność	80%

4.3. Funkcja Rosenbrocka

Kolejnym, już ostatnim przykładem funkcji testowej jest funkcja Rosenbrocka. Funkcja ta jest funkcją ciągłą, skalowalną i jednomodalną.

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

Funkcja ta jest „bardzo płaska” w pobliżu ekstrema, dlatego jest nieliniowym

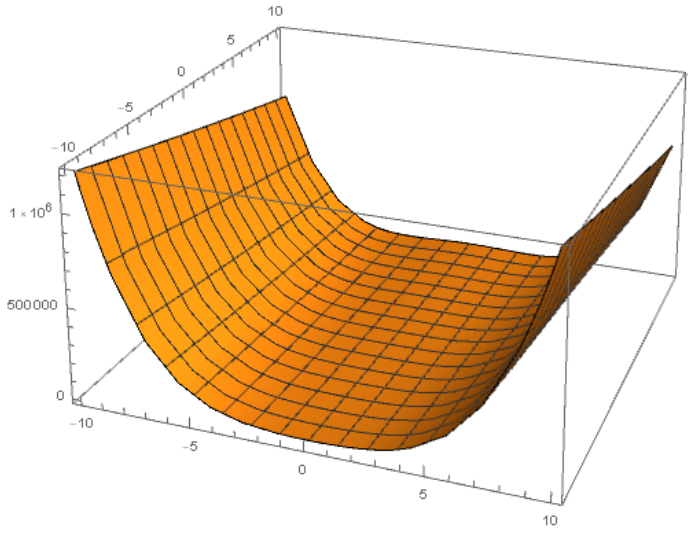
przykładem funkcji testowych. Funkcja ta przyjmuje wyłącznie wartości nieujemne. Na potrzeby projektu wartości argumentów dla tej funkcji zostały zawężone do poniższego zakresu:

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

Posiada ona następujące globalne minimum:

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

Poniższy wykres prezentuje jej wygląd w zadanym zakresie:



Rysunek 4: Funkcja Rosenbrocka o 2 wymiarach

4.3.1. 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	6
T_{end}	0.01
It	500
k	0.99
It_m	637
Skuteczność	99%

5. Implementacja

Implementacja algorytmu symulowanego wyżarzania składa się z kilku metod, które realizują opisany w rozdziale 2.4 jego przebieg. W rozdziale tym skupimy się na omówieniu tych metod oraz przedstawimy ich implementację w języku C#.

5.1. Używane parametry i zmienne

Wraz z zainicjalizowaniem obiektu symulowanego wyżarzania, należy ustawić parametry inicjalizujące go, a konkretniej: f - optymalizowaną funkcję, T_0 - temperaturę początkową, T_{end} - temperaturę końcową, It - ilość wewnętrznych iteracji, k - parametr chłodzący. W programie zostały one nazwane: **Function**, **Arguments**, **Arguments2**, **BeginingTemperature**, **EndingTemperature**, **Iterations**, **Cooling**, **SatisfactionSolutionValue**.

Function jest identyfikatorem referencji do obiektu reprezentującego problem. 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, która reprezentuje współrzędne dla obecnie najlepszego rozwiązania problemu ($x_{best} = (x_1, x_2, ..., x_n)$).

Arguments2 jest również tablicą liczb zmiennoprzecinkowych, jednak przechowuje ona współrzędne dla tymczasowego rozwiązania. Jest tego samego rozmiaru, co właściwość **Arguments** ($x_{tmp} = (x_1, x_2, ..., x_n)$).

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.

Iterations jest liczbą wewnętrznych iteracji (ilość powtórzeń algorytmu bez obniżania temperatury).

Cooling to liczba zmiennoprzecinkowa, w każdym głównym kroku algorytmu zmienna **temperature** jest mnożona przez tą wartość. Jest ona z przedziału (0, 1), więc temperatura powoli się obniża.

SatisfactionSolutionValue jest liczbą, jaką rozwiązanie musi osiągnąć, aby wynik poszukiwania rozwiązania był dla nas satysfakcjonujący. Jest to zmienna opcjonalna, algorytm nadal będzie działać, gdy nie poda się jej wartości. Dzięki niej, jeśli funkcjonał osiągnie wartość oczekiwaną, to przestaje działać algorytm.

W programie również używam kilku pomocnicznych zmiennych:

temperature to zmienna, która reprezentuje temperaturę. Jest ona używana w warunkach głównej iteracji oraz do wyznaczania funkcji prawdopodobieństwa. Wraz z postępem iteracji maleje.

bestSolution to liczba zmiennoprzecinkowa, która reprezentuje wartość obecnie najlepszego rozwiązania. Finalnie będzie ona wartością funkcjonału dla najlepszego rozwiązania całego problemu.

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

counter jest liczbą oznaczającą miarę głównej iteracji.

5.2. Implementacja

Metoda SetMaxCounter()

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

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

Metoda DrawArguments()

W metodzie tej losowane są początkowe argumenty (właściwość **Arguments**) z przedziału zadanego dla danego zagadnienia.

Metoda Move()

W tym miejscu najpierw obliczany jest pozostały procent iteracji do ukończenia procesu (znając parametry algorytmu można wyznaczyć ilość głównych iteracji). Następnie biorąc 80% szerokości zakresu wartości ze zmiennych (zmienna partOfTheDomain), mnożymy ją przez ilość pozostały do końca procesu procent iteracji (i przypisujemy do zmiennej value). Dalej, w pętli, każdej składowej tymczasowego rozwiązania (właściwość **Arguments2**), przypisywana jest suma odpowiedniej składowej najlepszego rozwiązania oraz losowej liczby z przedziału [-value, value]. Wraz z postępem iteracji zakres ten jest coraz węższy.

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

```
    double leftPercent = leftTemperatureCoolingTimes / maxCounter;

    double domainValue = (Function.RightBound - Function.LeftBound);
    double partOfTheDomain = 0.8;
    double value = (partOfTheDomain * 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;
    }
}
```

Metoda ShouldChangeAnyway()

Jest to prosta implementacja funkcji prawdopodobieństwa, o której mowa była w rozdziale 2.2.

Metoda 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 zagnieżdzonej pętli, szuka sąsiada obecnego najlepszego rozwiązania. Za-

mienia nowe rozwiązanie ze starym, jeżeli zostały spełnione odpowiednie warunki. Jeżeli nowe rozwiązanie spełnia kolejny warunek, to kończy program zwracając najlepsze rozwiązanie. W przeciwnym razie algorytm wychodzi z zagnieżdzonej pętli, zmniejsza zmienną odpowiedzialną za temperaturę i jest to koniec kroków w jednej pełnej, głównej 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;
}
```

6. Zastosowanie algorytmu w rozwiązywaniu odwrotnego zagadnienia przewodnictwa ciepła

Zanim przejdziemy do określenia zadania odwrotnego przewodnictwa ciepła, w pierwszej kolejności zdefiniujemy zadanie bezpośrednie (proste).

6.1. Zadanie bezpośrednie

Niech dane będzie równanie przewodnictwa ciepła postaci:

$$c\rho\frac{\partial u}{\partial t} = \lambda\frac{\partial^2 u}{\partial x^2}, \qquad x \in [0,a], t \in [0,T].$$

z zadanym warunkiem początkowym:

$$u(x,0) = f(x), \qquad x \in [0,a]$$

oraz z zadanymi warunkami brzegowymi I rodzaju:

$$u(0,t) = g(t), \qquad u(a,t) = h(t), \qquad t \in [0,T]$$

Do rozwiązania zadania bezpośredniego zastosujemy metodę różnic skończonych (schemat jawny). Rozważamy równanie przewodnictwa ciepła na siatce:

$$\Delta = \{x_i; x_i = a + ih, h = \frac{a}{n}, i = 0, ..., n\}$$

oraz w kolejnych chwilach czasu:

$$t_j = j\Delta\tau, \qquad gdzie\Delta\tau = \frac{T}{m}$$

Stosując ilorazy różnicowe, równanie przewodnictwa ciepła przyjmuje dyskretną postać:

$$c_i\rho_i\frac{u_i^{j+1} - u_i^j}{\Delta\tau} = \lambda\frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta h^2}$$

Przekształcając powyższy wzór otrzymujemy:

$$u_i^{j+1} = \frac{\lambda\Delta\tau}{c_i\rho_i}\left(\frac{u_{i+1}^j - 2u_i^j + u^j - 1_i}{\Delta h^2}\right) + u_i^j, i = 1, ..., n-1, j = 0, ..., m-1.$$

Jest to schemat różnicowy dla rozważanego zadania przewodnictwa ciepła, który pozwala wyznaczyć przybliżone wartości funkcji $u(x_i, t_j)$ w punktach siatki w wybranych chwilach czasu t_j.

6.2. Implementacja metody różnic skończonych

Do rozwiązania zadania bezpośredniego służy metoda Solve, której kod został przedstawiony poniżej:

```
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;
}
```

6.3. Sformułowanie zadania odwrotnego

Rozważane przez nas zadanie odwrotne polega na tym, że nie znamy funkcji $h(t)$ opisującej warunek brzegowy $u(a,t) = h(t)$. W zamian za to znamy wartości funkcji u (oznaczone $u_j^P(x_s)$) „pobrane" w punkcie pomiarowym $x_s \in (0,a)$ położonym w „pobliżu" brzegu a w pewnych chwilach czasu $t_j, j = 1,...n_t$. Wartości te mogą oznaczać np. pomiar temperatury. Zakładamy, że funkcja $h(t)$ jest postaci:

$$h(t) = pt^2 + qt + s$$

gdzie: p, q, s są szukanymi parametrami.

Dla wybranych wartości p, q, s rozwiązujemy zadanie bezpośrednie (określone w rozdziale 6.3.1) i w ten sposób odtworzymy wartości funkcji u w punkcie pomiarowym $\overline{u}(x_s)$.

Rozważane zadanie odwrotne sprowadza się do rozwiązania zadania optymalizacyjnego, polegającego na minimalizacji funkcjonału:

$$F(p,q,s) = \sum_{j=1}^{M} \prime \left(u_j^P(x_s) - \overline{u}_j(x_s) \right)^2 .$$

Minimalizując funkcjonał F wyznaczamy szukane wartości p, q, s, a tym samym otrzymujemy funkcję opisującą warunek brzegowy, co jest rozwiązaniem zadania odwrotnego. Do minimalizacji funkcjonału $F(p,q,s)$ wykorzystamy algorytm symulowanego wyżarzania.

6.3.1. Przykład numeryczny

Rozważmy zadanie dla $c = \rho = \lambda = 1$, $a = 1$, T= 1 z zadanym warunkiem początkowym:

$$u(x,0) = \frac{1}{2}x^2$$

oraz zadanym warunkiem brzegowym

$$u(0,t) = t$$

Brak informacji o funkcji opisującej warunek brzegowy na brzegu $x = 1$ rekompensują dane pomiarowe temperatury w punkcie $x_s = 0.8$ z krokiem $\Delta t = 0.1$.

Skadinąd wiemy, że rozwiązaniem dokładnym tak postawionego zadania jest funkcja $h(t) = t + 1$ oraz $u(x,t) = \frac{1}{2}x^2 + t$.

6.3.2. Implementacja rozwiązania problemu odwrotnego

Klasa odpowiadająca za rozwiązanie problemu odwrotnego posiada metodę Solve(), która zwraca błąd odtworzenia funkcji (wartość funkcjonału $F(p,q,s)$). Metoda ta przyjmuje dowolną ilość parametrów (ze względu na uniwersalność metody Solve() w przypadku innych rozwiązywanych problemów), gdzie tutaj akurat pierwsze trzy argumenty są parametrami szukanej funkcji kwadratowej. W implementacji tej metody jest używanych kilka zmiennych lub metod, które oznaczają:

$GetTemperatureMeasurements()$, pomocnicza metoda, która zwraca odpowiednie pomiary temperatur w punkcie pomiarowym (z funkcją $\overline{h}$ jako odtworzona funkcja kwadratowa),
$Measurements$ - dane pomiarowe.

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 = GetTemperatureMeasurements();
    for (int i = 0; i < Measurements.Length; i++)
    {
        sum += Math.Pow(Measurements[i] - tmpMeasurements[i], 2);
    }
    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().

6.3.3. Parametry algorytmu

Ze względu na czas obliczeń zadania odwrotnego, ilość prób sprawdzających jakość otrzymanych rezultatów została ograniczona do 5. Poniższa tabela przedstawia dobrane argumenty, które pozwalają na rozwiązanie problemu z „satysfakcjonującym" wynikiem błędu odtworzenia.

Parametr	**Wartość**
T_0	20
T_{end}	0.01
It	35000
k	0.99
It_m	757

6.4. Rezultaty

Oprócz prób rozwiązania zadania odwrotnego z danymi dokładnymi, zostały również podjęte próby dla danych zakłóconych błędem 1, 2 i 5%. Na potrzeby tych zadań temperatury pomiarowe zostały zakłócone losowym błędem o rozkładzie równomiernym.

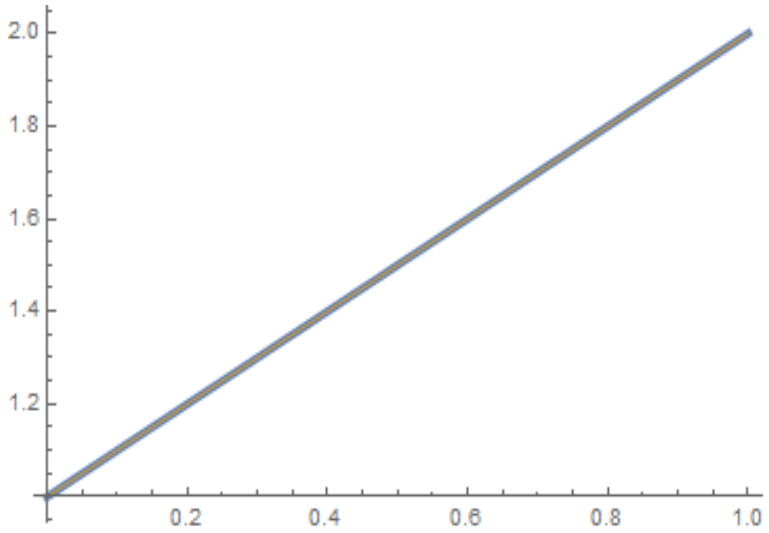
Tabela (XXX, tutaj będzie numer) przedstawia wyniki uzyskane dla podanych powyżej parametrów (liczby zaokrąglono do 4 miejsca po przecinku) dla danych niezakłóconych.

Tabela: Zestawienie wyników dla danych dokładnych (niezakłóconych)

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

gdzie Nr w tabeli oznacza numer próby odtwarzanego zadania odwrotnego (wykonano je 5 razy).

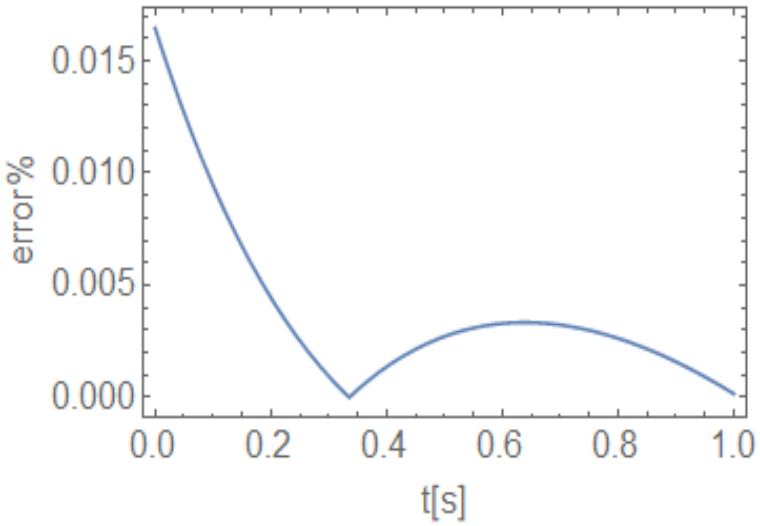
Różnicę pomiędzy funkcją dokładną, a opisującą odtworzony warunek brzegowy została przedstawiona na rysunku (żółty kolor oznacza funkcję dokładną, niebieska odtworzoną).



Rysunek 5: Porównanie dokładnej (linia żółta) i odtworzonej (linia niebieska) funkcji $h(t)$ opisującej warunek brzegowy

Przygotowano również wykres przedstawiający błąd względny uzyskanych rezul-

tatów, co przedstawia rysunek 6.



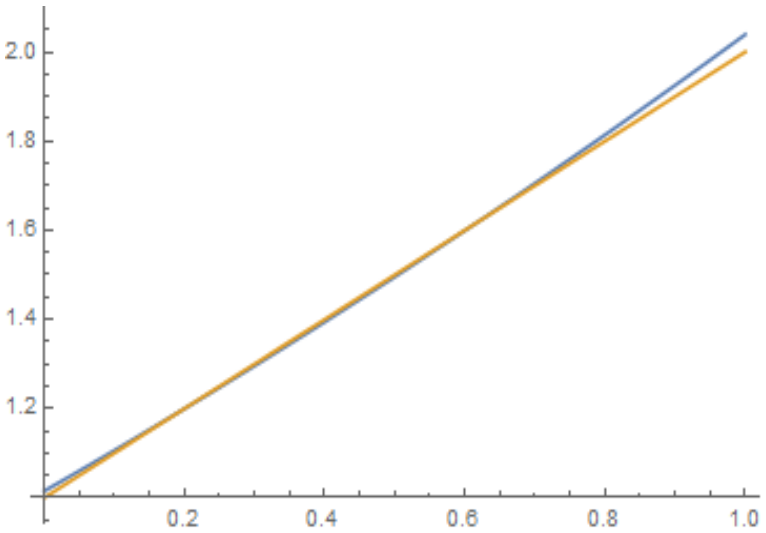
Rysunek 6: Błąd względny odtwarzanej funkcji $h(t)$ dla danych niezakłóconych błędem

W tabeli (XXX) zostały przedstawione wyniki uzyskane dla wybranych parametrów algorytmu (liczby zaokrąglono do 4 miejsca po przecinku) dla danych zakłóconych błędem 1%.

Tabela: Zestawienie wyników dla danych zakłóconych błędem 1%

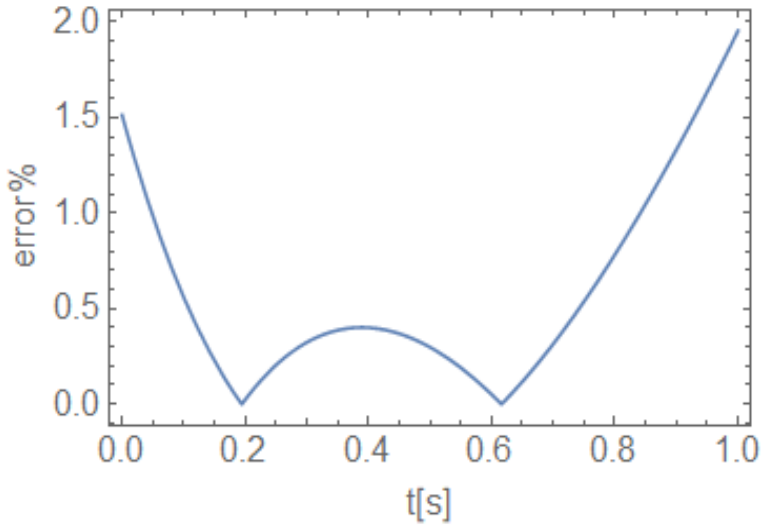
Nr.	**Błąd**	**p**	**q**	**s**
1.	2,7209 * 10^{-2}	0,1190	0,8952	1,0189
2.	2,3787 * 10^{-2}	0,1152	0,9157	1,0140
3.	2,7362 * 10^{-2}	0,1500	0,8860	1,0063
4.	2,3056 * 10^{-2}	0,1173	0,8890	1,0178
5.	2,3617 * 10^{-2}	0,1304	0,9021	1,0187

Porównanie funkcji dokładnej i odtwarzanej dla danych zakłóconych błędem 1% została przedstawiona na rysunku 7.



Rysunek 7: Porównanie dokładnej (linia żółta) i odtworzonej (linia niebieska) funkcji $h(t)$ dla danych zakłóconych błędem 1% opisującej warunek brzegowy

Błąd względny tego odtworzenia ilustruje rysunek 8.



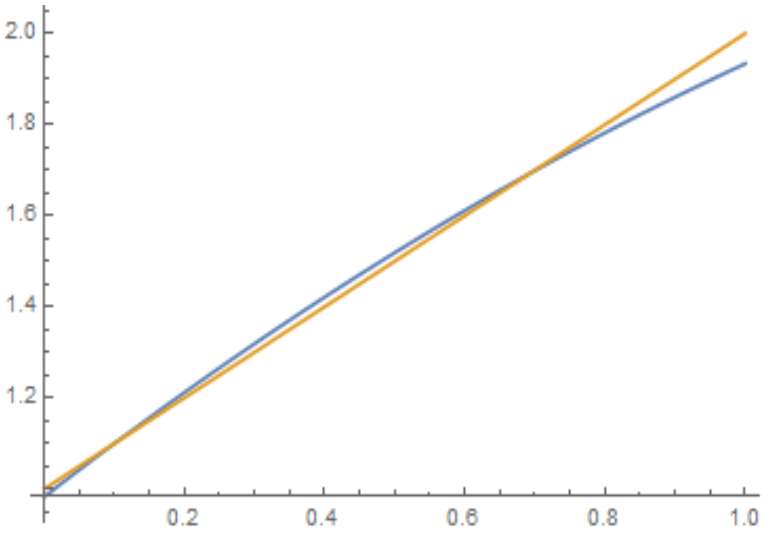
Rysunek 8: Błąd względny odtwarzanej funkcji $h(t)$ dla danych zakłóconych błędem 1%

Wyniki prób odtworzenia parametrów funkcji granicznej dla danych zakłóconych błędem 2% zostały przedstawione poniżej:

Tabela: Zestawienie wyników dla danych zakłóconych błędem 2%

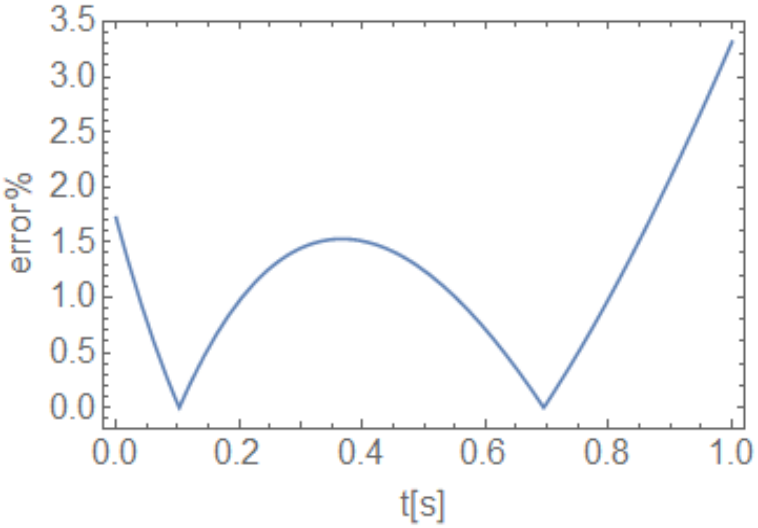
Nr.	**Błąd**	**p**	**q**	**s**
1.	6,8528 * 10^{-2}	-0,3642	1,2993	0,9513
2.	6,5422 * 10^{-2}	-0,2570	1,1971	0,9992
3.	5,8742 * 10^{-2}	-0,2352	1,1853	0,9895
4.	6,0410 * 10^{-2}	-0,1531	1,1175	0,9937
5.	6,0850 * 10^{-2}	-0,1995	1,1643	0,9801

Różnicę pomiędzy funkcjami dokładną i odtworzoną przedstawia poniższy wykres:



Rysunek 9: Porównanie dokładnej (linia żółta) i odtworzonej (linia niebieska) funkcji $h(t)$ dla danych zakłóconych błędem 2% opisującej warunek brzegowy

Rysunek 10 prezentuje błąd względny odtworzenia funkcji granicznej z danymi zakłóconymi błędem 2%.



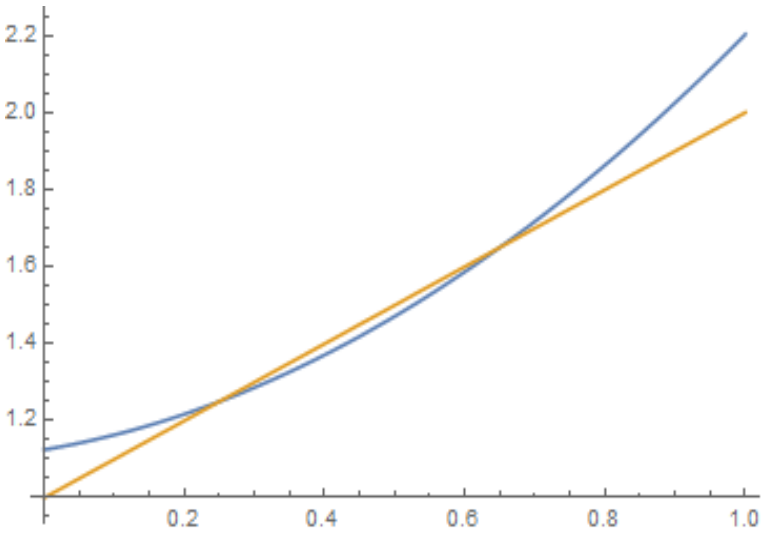
Rysunek 10: Błąd względny odtwarzanej funkcji $h(t)$ dla danych zakłóconych błędem 2%

Otrzymane rezultaty parametrów odtworzonej funkcji $h(t)$, przy danych pomiarowych zakłóconych błędem 5%, zostały przedstawione w tabeli (XXX).

Tabela: Zestawienie wyników dla danych zakłóconych błędem 5%

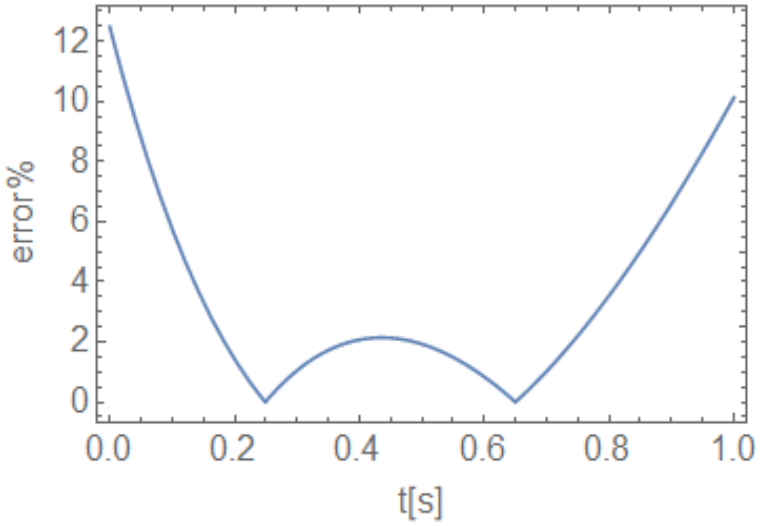
Nr.	**Błąd**	**p**	**q**	**s**
1.	0,2031	0,8678	0,2034	1,1597
2.	0,2049	0,6563	0,4295	1,0985
3.	0,2030	0,7652	0,3167	1,1265
4.	0,2060	0,8133	0,2428	1,1277
5.	0,2041	0,7516	0,3426	1,1119

Rysunek 11 przedstawia porównanie funkcji dokładnej oraz odtworzonej z danymi zakłóconymi błędem 5%.



Rysunek 11: Porównanie dokładnej (linia żółta) i odtworzonej (linia niebieska) funkcji $h(t)$ dla danych zakłóconych błędem 5% opisującej warunek brzegowy

Poniższy rysunek ilustruje błąd względny otrzymanych wyników:



Rysunek 12: Błąd względny odtwarzanej funkcji $h(t)$ dla danych zakłóconych błędem 5%

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 do wyznaczenia minimum dla kilku funkcji testowych oraz do rozwiązania odwrotnego zadania przewodnictwa ciepła określonego w rozdziale 6.3.1.

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 zadanego problemu, a wyniki przeprowadzonych testów rozwiązania odwrotnego zadania przewodnictwa ciepła świadczą, że zaproponowana metoda jest właściwym narzędziem do rozwiązywania tego typu zadań.

Literatura

[1] 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]

[2] 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]

[3] 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]

[4] 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

[5] 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

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

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

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

[9] K. Żydzik, T. Rak, *C# 6.0 i MVC 5. Tworzenie nowoczesnych portali internetowych*, Wyd. Helion, 2015, ISBN 978-83-283-0864-0

[10] J. Albahari, B. Albahari, *C# 7.0 in a Nutshell: The Definitive Reference*, Wyd. O'Reilly Media, 2017, ISBN 9781491987643

[11] 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]

[12] 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

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

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

[15] R. Grzymkowski, A. Zielonka, *Zastosowania teorii falek w zagadnieniach brzegowych*, Wyd. Pracownia Komputerowa Jacka Skalmierskiego, 2004, s. 86-89, ISBN 9788389105639