Politechnika Śląska
Wydział Matematyki Stosowanej
Kierunek Informatyka
Studia stacjonarne I stopnia

Projekt inżynierski

Porównanie wybranych algorytmów heurystycznych w rozwiązywaniu zagadnień odwrotnych

Kierujący projektem:
dr inż. Adam Zielonka

Autor:
Kamil Kryus

Gliwice 2018

Projekt inżynierski:

Porównanie wybranych algorytmów heurystycznych 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:
Porównanie wybranych algorytmów heurystycznych 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

Na problem znalezienia optymalnego rozwiązania możemy trafić w wielu dziedzinach życia i nauki, np. w matematyce szukając globalnego minimum/maximum lub szukając najkrótszego połączenia pomiędzy kilkoma miastami (problem komiwojażera). Szukając rozwiązań, zawsze chcemy by było ono jak najlepsze (lub dokładnie najlepsze) i zostało znalezione w rozsądnym czasie. W tym celu właśnie jest używana heurystyka.

Metody heurystyczne znacznie skracają czas wyszukiwania rozwiązania problemu, aczkolwiek często ich wynikiem jest jedynie wynik bardzo zbliżony do najlepszego. Pozwala jednak nam to na jedną z dwóch rzeczy:

1. zaakceptowanie takiego wyniku, gdy dokładne rozwiązanie nie jest konieczne (np. kompresja obrazu),
2. zawężenia zakresu i dalszych poszukiwań najlepszego rozwiązania.

Jednak aby metodę heurystyczną uznać za dobrą, musi ona spełniać 3 wymagania:

- rozwiązanie jest możliwe do znalezienia z rozsądnym wysiłkiem obliczeniowym,
- rozwiązanie powinno być bliskie optymalnemu,
- prawdopodobieństwo uzyskania złego rozwiązania powinno być niskie.

1. Opis

1.1. Opis problemu

Czasami w nauce, zwłaszcza w matematyce, możemy natrafić na zadanie, które będzie polegało na wyznaczeniu niektórych parametrów modelu, gdy posiadamy obserwowane wartości. W odróżnieniu od "zwyczajnych" 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 typów problemów pasujących do grupy problemów odwrotnych, jest odwrotne zagadnienie przewodnictwa ciepła. Przy posiadaniu niekompletnego modelu matematycznego oraz funkcji opisującej rozkład temperatury, zadanie polega na rekonstrukcji niektórych granicznych parametrów modelu.

1.2. Cel

Ze względu na dużą użyteczność algorytmów heurystycznych, postanowiłem rozwinąć swoją wiedzę na ich temat i dobrze poznać algorytm symulowanego wyżarzania i poprzez jego implementację, sprawdzić jego skuteczność i użyteczność w praktyce w przypadku kilku funkcji oraz przy rozwiązywaniu odwrotnego zagadnienia przewodnictwa ciepła, tworząc przy tym aplikację rozwiązującą konkretny problem.

Stworzenie aplikacji pozwalającej na obliczenie warunków granicznych w podanym problemie odwrotnym pozwoliłoby na uzupełnianie modelu matematycznego w prosty i szybki sposób. Ponadto uniwersalne podejście do implementacji danego algorytmu pozwala również na intuicyjne znajdowanie przybliżonej wartości globalnego minimum funkcji.

2. Opis algorytmów

2.1. Algorytm 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.

2.1.1. Parametry

Początkowa konfiguracja

W tym kroku powinniśmy zainicjalizować naszą temperaturę wysoką 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 wysoką możliwość zamian, a wraz z postępem iteracji te prawdopodobieństwo się zmniejszało i pod koniec było bliskie zeru.

Końcowa temperatura

Jest to bardzo mała 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.

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 ukończenia globalnych iteracji.

Funkcja kosztu

Poprzez funkcję kosztu rozumiemy różnicę pomiędzy obecnie najlepszym rozwiązaniem, a nowym. Funkcja ta ma dodatkowe zastosowanie przy decydowaniu o zamiany 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 zależy od funkcji kosztu oraz obecnej temperatury. Prawdopodobieństwo zatem można przedstawić w następujący sposób:

$$P = e^{\frac{\Delta E}{T}}$$

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].

2.1.2. 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

3.1. Funkcja kwadratowa dwóch parametrów

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

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

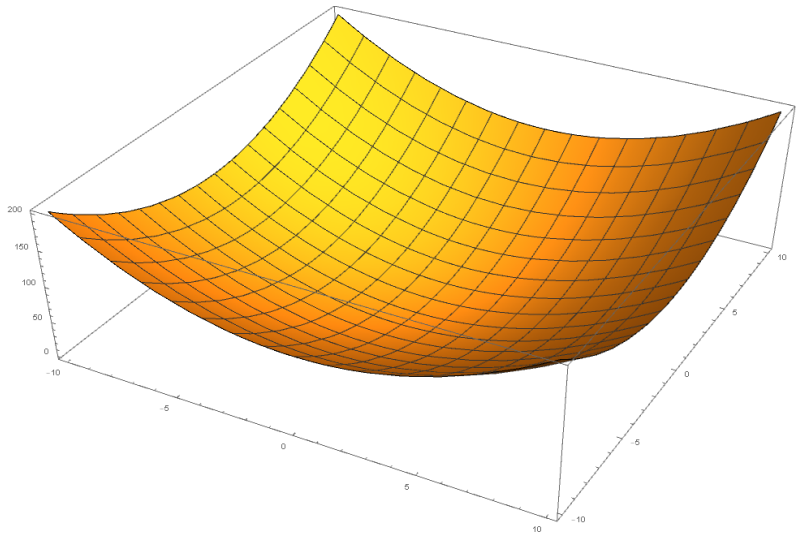
Funkcja ta jest funkcją parzystą i przyjmuje tylko wartości nieujemne. Na potrzeby projektu zakres dla tej funkcji został zawężony następująco:

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

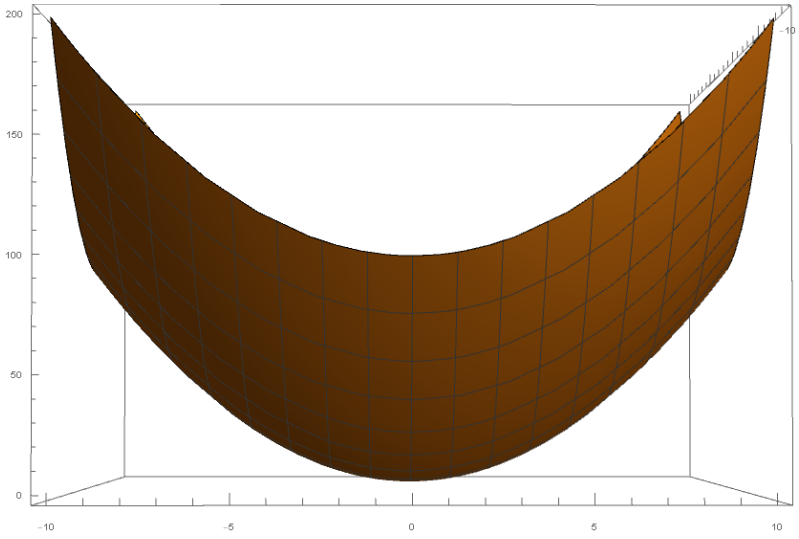
Posiada ona następujące globalne minimum:

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

Funkcję tę można zaprezentować na wykresie 2 wymiarowym, co prezentują następujące obrazki:



Rysunek 1: Funkcja kwadratowa dwóch parametrów



Rysunek 2: Funkcja kwadratowa dwóch parametrów

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

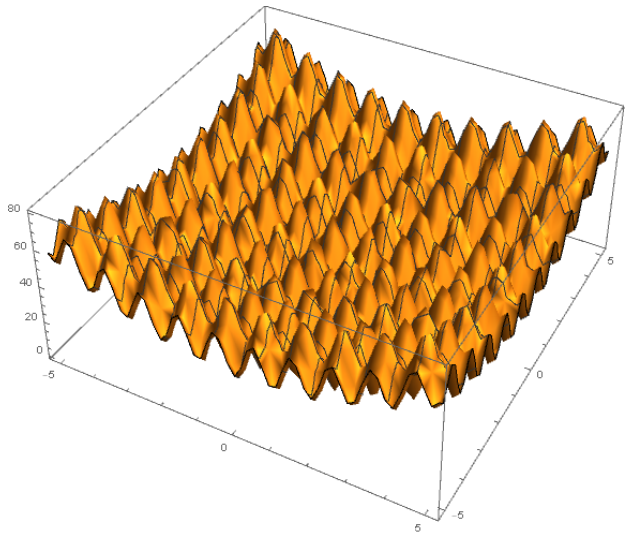
Wartości tej funkcji są nieujemne. Zakres wartości dla tej funkcji znajdziemy w przedziale:

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

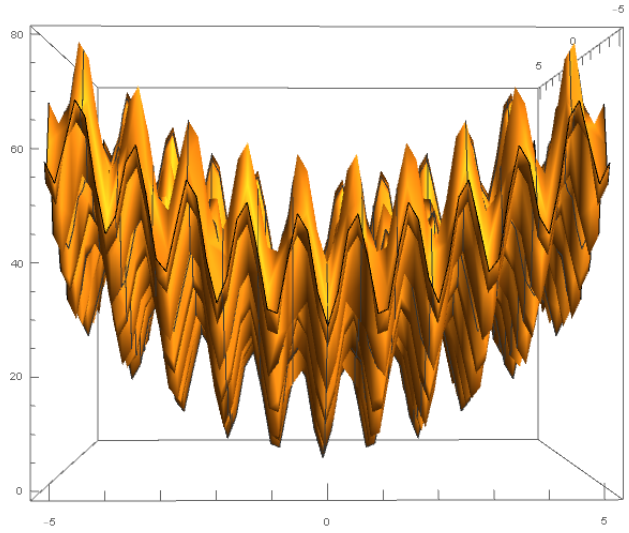
Posiada ona następujące globalne minimum:

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

By ujrzeć jej niektóre właściwości zaprezentowałem jej wykres w 2 wymiarach na poniższych obrazkach:



Rysunek 3: Funkcja Rastrigina o 2 wymiarach



Rysunek 4: Funkcja Rastrigina o 2 wymiarach

3.3. Funkcja Rosenbrocka

Funkcja ta jest funkcją ciągłą, skalowalną i jednomodalną.

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

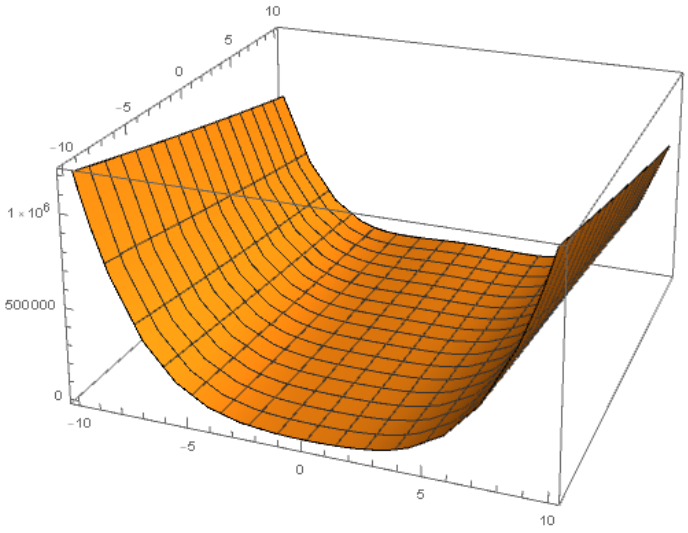
Funkcja ta również przyjmuje wyłącznie wartości nieujemnie. Na potrzeby projektu wartości argumentów dla tej funkcji zostały zawężone do poniższego zakresu:

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

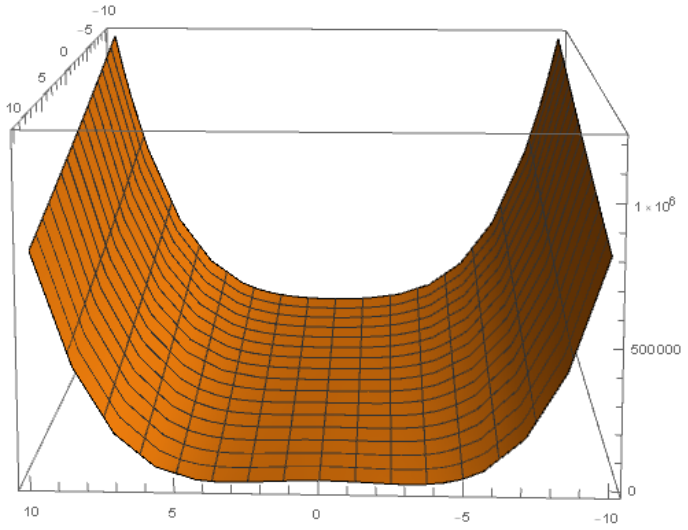
Posiada ona następujące globalne minimum:

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

Poniższe wykresy prezentują jej wygląd w zadanym zakresie:



Rysunek 5: Funkcja Rosenbrocka o 2 wymiarach



Rysunek 6: Funkcja Rosenbrocka o 2 wymiarach

4. Dobór parametrów

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, przyjąłem kilka założeń, a następnie sukcesywnie poszukiwałem 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, sprawdzałem 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 5 wymiarowej funkcji Rastrigina" tabele przedstawią stopniowe dojście do parametrów dających zadowalające wyniki, 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.

4.1. Dobieranie parametrów dla funkcji 5 wymiarowej funkcji Rastrigina

Przed rozpoczęciem testów przyjęto dwa założenia:

1. Końcowa temperatura została ustawiona na stałą wartość równą 0.001,
2. Stopień chłodzenia temperatury został ustawiony na 0.99.

Nr.	**T0**	**Iteracje**	**Rozwiązanie**
1.	500	300	1,09718505292704
2.	500	400	1,29513703551228
3.	100	500	1,29579688714116
4.	300	400	1,39516546905179
5.	100	400	1,59400704106305
6.	400	200	1,59460563439097
7.	400	400	1,59473245683331
8.	300	500	1,69368418646326
9.	100	200	1,69430049805511
10.	400	500	1,69488738622795
11.	200	500	1,79356181491557
12.	500	200	1,89311831496136
13.	300	100	1,89379034166828
14.	200	200	1,89405233085409
15.	400	300	1,99208845088035
16.	500	500	1,99250333315708
17.	300	300	1,99272286399703
18.	200	400	2,09157445324622
19.	100	300	2,19168206783873
20.	300	200	2,19179918169512
21.	500	100	2,19209302230291
22.	200	100	2,19237079270076
23.	400	100	2,19237108578013
24.	200	300	2,29096718181202
25.	100	100	2,98943206230488

Badanie zawarte w tabeli 1 pokazuje, iz parametry na takim poziomie nie maja az tak duzego znaczenia, jednak mozna zauwazyc, iz wieksza wartosc parametrow prowadzi do nieco lepszych srednich wynikow. Badanie skutecznosci dla najwyzszych parametrow (czyli 500 i 500) wynioslo 12%, co jest bardzo slabym wynikiem.

Nr.	**T0**	**Iteracje**	**Rozwiązanie**
1.	8000	6000	0,400456362305732
2.	10000	4000	0,400942457997776
3.	4000	8000	0,499468816488703
4.	4000	10000	0,499496708877431
5.	6000	6000	0,499881155483216
6.	8000	8000	0,500190776858622
7.	8000	10000	0,500669723808464
8.	2000	6000	0,500802825547873
9.	2000	4000	0,59876018888673
10.	2000	8000	0,598944859155338
11.	10000	8000	0,69839967735632
12.	6000	8000	0,698671405088412
13.	10000	10000	0,698922421188404
14.	4000	6000	0,797834332682657
15.	6000	10000	0,798499451192562
16.	4000	4000	0,798596850008959
17.	2000	10000	0,798896772938739
18.	8000	4000	0,79899550599823
19.	10000	6000	0,898177793387154
20.	6000	4000	0,898733662378301
21.	4000	2000	0,996463628047726
22.	8000	2000	0,997870530143807
23.	6000	2000	1,09703634442711
24.	2000	2000	1,09731890136792
25.	10000	2000	1,39564290923108

10k i 10k 46% 5k - 10k -¿ 46% 5k i 20k -¿ 58% 5k i 30k -¿ 72%

Nr.	**T0**	**Iteracje**	**Rozwiązanie**
1.	aaaaaaa aaaaaaa	aaaaaaa aaaaaaa	oko
2.	aaaaaaa aaaaaaa	aaaaaaa	oko
3.	aaaaaaa	aaaaaaaaaaaaaa	oko
4.	aaaaaaa	aaaaaaaaaaaaaa	oko
5.	aaaaaaa	aaaaaaaaaaaaaa	oko
6.	aaaaaaa	aaaaaaaaaaaaaa	oko
7.	aaaaaaa	aaaaaaaaaaaaaa	oko
8.	aaaaaaa	aaaaaaaaaaaaaa	oko
9.	aaaaaaa	aaaaaaaaaaaaaa	oko
10.	aaaaaaa	aaaaaaaaaaaaaa	oko
11.	aaaaaaa	aaaaaaaaaaaaaa	oko
12.	aaaaaaa	aaaaaaaaaaaaaa	oko
13.	aaaaaaa aaaaaaa	aaaaaaa aaaaaaa	oko
14.	aaaaaaa aaaaaaa	aaaaaaa	oko
15.	aaaaaaa	aaaaaaaaaaaaaa	oko
16.	aaaaaaa	aaaaaaaaaaaaaa	oko
17.	aaaaaaa	aaaaaaaaaaaaaa	oko
18.	aaaaaaa	aaaaaaaaaaaaaa	oko
19.	aaaaaaa	aaaaaaaaaaaaaa	oko
20.	aaaaaaa	aaaaaaaaaaaaaa	oko
21.	aaaaaaa	aaaaaaaaaaaaaa	oko
22.	aaaaaaa	aaaaaaaaaaaaaa	oko
23.	aaaaaaa	aaaaaaaaaaaaaa	oko
24.	aaaaaaa	aaaaaaaaaaaaaa	oko
25.	aaaaaaa	aaaaaaaaaaaaaa	oko
26.	aaaaaaa	aaaaaaaaaaaaaa	oko
27.	aaaaaaa	aaaaaaaaaaaaaa	oko
28.	aaaaaaa	aaaaaaaaaaaaaa	oko
29.	aaaaaaa	aaaaaaaaaaaaaa	oko
30.	aaaaaaa	aaaaaaaaaaaaaa	oko
31.	aaaaaaa	aaaaaaaaaaaaaa	oko
32.	aaaaaaa	aaaaaaaaaaaaaa	oko
33.	aaaaaaa	aaaaaaaaaaaaaa	oko

Nr.	**T0**	**Iteracje**	**Rozwiązanie**
1.	aaaaaaa aaaaaaa	aaaaaaa aaaaaaa	oko
2.	aaaaaaa aaaaaaa	aaaaaaa	oko
3.	aaaaaaa	aaaaaaaaaaaaaa	oko
4.	aaaaaaa	aaaaaaaaaaaaaa	oko
5.	aaaaaaa	aaaaaaaaaaaaaa	oko
6.	aaaaaaa	aaaaaaaaaaaaaa	oko
7.	aaaaaaa	aaaaaaaaaaaaaa	oko
8.	aaaaaaa	aaaaaaaaaaaaaa	oko
9.	aaaaaaa	aaaaaaaaaaaaaa	oko
10.	aaaaaaa	aaaaaaaaaaaaaa	oko
11.	aaaaaaa	aaaaaaaaaaaaaa	oko
12.	aaaaaaa	aaaaaaaaaaaaaa	oko

Nr.	T0	Iteracje	Rozwiązanie
1.	aaaaaaa aaaaaaa	aaaaaaa aaaaaaa	oko
2.	aaaaaaa aaaaaaa	aaaaaaa	oko
3.	aaaaaaa	aaaaaaaaaaaaaa	oko
4.	aaaaaaa	aaaaaaaaaaaaaa	oko
5.	aaaaaaa	aaaaaaaaaaaaaa	oko
6.	aaaaaaa	aaaaaaaaaaaaaa	oko
7.	aaaaaaa	aaaaaaaaaaaaaa	oko
8.	aaaaaaa	aaaaaaaaaaaaaa	oko
9.	aaaaaaa	aaaaaaaaaaaaaa	oko
10.	aaaaaaa	aaaaaaaaaaaaaa	oko
11.	aaaaaaa	aaaaaaaaaaaaaa	oko
12.	aaaaaaa	aaaaaaaaaaaaaa	oko

Ostatecznie wybrane parametry dla tego problemu:

pocz temp

konc temp

cooling

iteracje

Badanie skutecznosci

skutecznosc:

4.2. Dobieranie parametrów dla funkcji 3 wymiarowej funkcji Rastrigina

Ostatecznie wybrane parametry dla tego problemu:

pocz temp

konc temp
cooling
iteracje

skutecznosc:

4.3. Parametry dobrane dla funkcji kwadratowej dwóch parametrów

Ostatecznie wybrane parametry dla tego problemu:
pocz temp
konc temp
cooling
iteracje

skutecznosc:

4.4. Parametry dobrane dla funkcji Rosenbrocka

Ostatecznie wybrane parametry dla tego problemu:
pocz temp
konc temp
cooling
iteracje

skutecznosc:

5. Implementacja

Moja implementacja algorytmu symulowanego wyżarzania składą się z kilku etapów, przedstawionych w roździale 2.1.2, które zostaną omówione i/lub zostanie przedstawiona ich implementacja.

Używane parametry i zmienne

Wraz z zainicjalizowaniem obiektu symulowanego wyżarzania, ustawiam mu 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 zmienną, która przetrzymuje obiekt 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, która przetrzymuje w sobie tablicę argumentów dla obecnie najlepszego rozwiązania problemu.

Arguments2 za to jest tablicą argumentów dla tymczasowego rozwiązania. Jest tego samego rozmiaru, co zmienna **Arguments**.

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

BeginingTemperature - jest to wartość, od której rozpoczyna się proces poszukiwań.

EndingTemperature jest liczbą, która informuje o zakończeniu iteracji. Gdy

wartość zmiennej **temperature** spadnie do tego poziomu, proces poszukiwania się zakończył.

Cooling, w każdym kroku 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 jest zmienną używaną w "globalnej" iteracji oraz przy sprawdzaniu funkcji prawdopodobieństwa. Z postępem iteracji maleje ona.

bestSolution 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 iterację.

Funkcja SetMaxCounter()

W funkcji tej symuluję proces obniżania temperatury aby obliczyć maksymalną ilość globalnych iteracji.

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

```
            tmpTemperature *= Cooling;
            maxCounter++;
        }
    }
```

Funkcja DrawArguments()

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

Funkcja Move()

W tym kroku najpierw obliczam jaki procent iteracji pozostał do ukończenia procesu. Następnie biorąc 80% pełnej puli możliwych wartości problemu, mnożę ją przez pozostały procent iteracji (i przypisuję do zmiennej value). Dalej w pętli każdemu argumentowi tymczasowego rozwiązania (zmienna **Arguments2**), przypisuję sumę odpowiedniej liczby z argumentów najlepszego rozwiązania (żeby był to sąsiad najlepszego rozwiązania) oraz losową liczbę z przedziału [-value, value]. Wraz z postępem iteracji zakres ten jest coraz węższy, ale rozwiązanie powinno też już być bliskie szukanemu. Na końcu upewniam się, iż nowa wartość nie wychodzi poza ustalony przez problem zakres.

```
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, tablica jest typu referencyjnego i konieczne jest skopiowanie wartości ze zmiennej **Arguments2** do zmiennej **Arguments**.

Implementacja algorytmu

Algorytm ten w kolejnych punktach wykonuje kroki, które zostały właśnie opisane. Po obliczeniu maksymalnej ilości iteracji, losuje pierwsze rozwiązanie. Następnie w pętli i następnej zagnieżdzonej pętli, szuka sąsiada obecnego najlepszego rozwiązania. Zamienia nowe rozwiązanie ze starym jeżeli zostały spełnione odpowiednie warunki i kończy program zwracając najlepsze rozwiązanie, jeżeli spełnia warunek. Jeżeli nie, wychodzi z zagnieżdzonej pętli, zmniejsza zmienną odpowiedzialną za temperaturę i jest to koniec kroków w jednej pełnej, „globalnej" iteracji.

```
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 algorytmów w rozwiązywaniu odwrotnego zagadnienia przewodnictwa ciepła

7. Narzędzia i technologie

7.1. Metodyka pracy

7.1.1. 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.

7.1.2. 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.

7.1.3. 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.

7.1.4. 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.

7.2. Użyte technologie

7.2.1. 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.

7.2.2. 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.

8. Podsumowanie

8.1. Dalsze kierunki rozwoju

8.2. Źródła

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4538776/

F. Rothlauf, Design of Modern Heuristics, Natural Computing Series, DOI 10.1007/978-3-540-72962-4 2, © Springer-Verlag Berlin Heidelberg 2011

R. Mart'ı and G. Reinelt, The Linear Ordering Problem, Exact and Heuristic Methods in Combinatorial Optimization 175, DOI: 10.1007/978-3-642-16729-4 2, c Springer-Verlag Berlin Heidelberg 2011

http://prac.im.pwr.edu.pl/ plociniczak/lib/exe/fetch.php?media=odwrotne.pdf

https://www.math.unl.edu/ scohn1/8423/wellposed.pdf

Literatura

[1] Jakaś pozycja literatury

[2] Jakaś pozycja literatury