PORR Projekt

Algorytm świetlika (FA) i roju świetlików (GSO) do rozwiązania problemu plecakowego.

Cezary Dubiel, 251387
Tomasz Korzeniowski, 265753

16 stycznia 2018

1 Zadanie

Zdefiniować i rozwiązać dwa zadania o różnej złożoności. Zrealizować zadania w wersji sekwencyjnej i równoległej. Opisać przyśpieszenie obliczeń w zależności od stopnia zrównoleglenia (porównać obie metody - FA i GSO). Przedstawić graficznie zbieżność obu algorytmów w wersji sekwencyjnej - wartości funkcji celu w kolejnych iteracjach.

2 Podstawy teoretyczne

2.1 Problem plecakowy

Problem plecakowy polega na zapakowaniu maksymalnie cennego zbioru przedmiotów do plecaka, tak by nie przekroczyć jego ładowności. Można wyróżnić dwa podejścia do liczby pakowanych przedmiotów. W ogólnym podejściu zakładamy, że posiadamy wiele $(1, 2, \ldots)$ egzemplarzy każdego z N różnych przedmiotów. W bardziej rzeczywistych przypadkach posiadamy dokładnie jeden egzemplarz każdego z N różnych przedmiotów. W tym projekcie zajmiemy się drugim podejściem.

Zakładamy, że mamy do dyspozycji $n \in N$ przedmiotów. Każdy z tych przedmiotów jest warty c, a także posiada swoją wagę m. Naszym zadaniem jest zdecydowanie czy i które z przedmiotów załadujemy, tak by ich sumaryczna masa nie przekroczyła ładowności plecaka M_{max}. Model matematyczny problemu plecakowego można przedstawić następująco

$$\max \sum_{i}^{N} c_i x_i \tag{1}$$

$$\sum_{i}^{N} m_i x_i \leqslant M_{max} \tag{2}$$

$$x_i \in \{0, 1\} \tag{3}$$

Zatem poszukujemy rozwiązania w postaci binarnego wektora x długości N:

$$x_i = \begin{cases} 1, & \text{gdy i-ty przedmiot zostanie zapakowany do plecaka} \\ 0, & \text{w przeciwnym przypadku} \end{cases}$$

Jeśli przyjmiemy, że posiadamy sześć przedmiotów, które możemy spakować, a także znamy $C = [c_i]_{1\times N}$, $M = [m_i]_{1\times N}$ oraz wektor $x = (0, 1, 0, 0, 1, 1)$ to wiemy, że do plecaka został załadowany przedmiot drugi, piąty oraz szósty.

2.2 Algorytm świetlika

Algorytm świetlika (FA - ang. *firefly algorithm*) jest heurystyką inspirowaną zachowaniem społecznym świetlików (robaczków świętojańskich). Jego głównym zastosowaniem są zadania optymalizacji ciągłej z ograniczeniami. Może jednak być wykorzystywany także w problemach optymalizacji kombinatorycznej.

Fenomen świetlików polega na ich bioluminescencyjnej komunikacji. Każdy z nich cechuje się pewną atrakcyjnością, proporcjonalną do jasności z jaką świecą. Świetlik o większej jasności przyciąga inne osobniki (przyjmujemy obupłciowość świetlików) o mniejszej jasności. Zakładamy też, że świetliki nie umierają.

Na początku działania algorytmu losowana jest populacja świetlików. Każdy z nich reprezentuje pewne rozwiązanie problemu, czyli poszukiwany wektor x. Jego jasnością jest wartość funkcji celu dla danego rozwiązania. Następnie poszukiwany jest najlepszy ze świetlików (pod względem jasności - wartości funkcji celu), a pozostałe poruszają się w jego kierunku. Ruch i-tego świetlika zależy od wartości funkcji atrakcyjności β_i

$$\beta_i = \beta_0 e^{-\gamma r_{ij}^2} \tag{4}$$

gdzie β_0 (maksymalna atrakcyjność) oraz γ (współczynnik absorpcji) są ustalonymi na wstępie parametrami algorytmu, a r_{ij}^2 to odległość między świetlikami i oraz j.

Ruch świetlika i przyciąganego do bardziej atrakcyjnego świetlika j jest następujący

$$x_i = x_i + \beta_i(x_j - x_i) + u_i \tag{5}$$

gdzie u_i jest wektorem liczb losowych wygenerowanych z rozkładem równomiernym z przedziału [0,1]. Jeśli świetlik nie znajdzie się w sąsiedztwie innego świetlika o większej jasności wówczas wykonywane jest jego losowe przesunięcie u_i.

Algorytm świetlika można przedstawić w postaci pseudokodu

Algorytm 1 Algorytm świetlika (FA)

```
Require: C, M, k, β_i, γ
  wylosuj populację świetlików
  while nie osiągnięto warunku stopu do
    for i=1 to k do
      for j=1 to k do
        if f(x_j) > f(x_i) then
          oblicz odległość r_ij
          zmodyfikuj atrakcyjność β_i
          wygeneruj losowy wektor u_i
          utwórz nowe rozwiązanie zgodnie z (5)
        end if
      end for
    end for
    wygeneruj losowy wektor u_b
    przesuń najlepszego świetlika x_b = x_b + u_b
  end while
```

$f(x_j)$ oznacza wartość funkcji celu dla świetlika x_j. Warunkiem stopu algorytmu może być zadana liczba iteracji. Do sprawdzenia zbieżności algorytmu warunkiem stopu będzie brak zmian rozwiązania (na pewnym poziomie) przed zadaną liczbę iteracji.

2.3 Algorytm roju świetlików

Algorytm roju świetlików (GSO - ang. *glowworm swarm optimization*), w przeciwieństwie do FA, oparty jest na zachowaniach społecznych nieskrzydlatych świetlików. Przejawia się to zdolnością do zmiany intensywności emisji lucyferny, czyli wydawaniem blasków o różnym natężeniu. Od algorytmu świetlika, GSO różni się także ze względu na możliwość podziału świetlików na podgrupy. Pozwala to osiągać wiele lub nawet wszystkie optima lokalne.

Różna jest także reprezentacja sąsiedztwa świetlików. Zmienna sąsiedztwa r_d jest związana z zakresem radialnym r_s, tak aby $0 < r_d \leqslant r_s$. Wynika z tego, że świetlik i bierze pod uwagę świetlika j jako sąsiada, jeśli j jest wewnątrz obszaru sąsiedztwa rozwiązania i oraz poziom lucyferny jest wyższy dla j-tego niż i-tego.

Algorytm rozpoczyna się od losowego rozmieszczenia świetlików, a także przypisaniu im wielkość lucyferny równą 0. Przebieg algorytmu składa się z 3 faz głównych. W fazie pierwszej następuje modyfikacja poziomu lucyferny, czyli wyznaczenie wartości funkcji celu. W oryginalnej wersji algorytmu odejmuje się część lucyferny w celu symulacji mechanizmu rozkładu tej substancji w czasie. Ze względu na to, że przedmioty przynoszą stały zysk wynikający z ich zapakowania, zrezygnujemy z takiego pomniejszania.

Kolejnym krokiem jest faza ruchu. Wyznaczane jest prawdopodobieństwo przemieszczenia świetlika i w kierunku j w chwili t zgodnie ze wzorem

$$p_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)} \tag{6}$$

$$N_i(t) = \{j : d_{ij}(t) < r_d(t); l_i(t) < l_j(t)\} \tag{7}$$

gdzie $N_i(t)$ to zbiór sąsiadów świetlika i, a $d_{ij}(t)$ odległość między świetlikiem i oraz j.

Następnie świetliki są przemieszczane zgodnie z

$$x_i(t+1) = x_i(t) + s\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|} \tag{8}$$

gdzie $s > 0$ oznacza rozmiar kroku przemieszczenia.

Ostatnią fazą jest modyfikacja zakresu sąsiedztwa.

$$r_d(t+1) = \min\left\{r_s, \max\left\{0, r_d(t) + \beta(n_t - |N_i(t)|)\right\}\right\} \tag{9}$$

gdzie β to pewna stała, a n_t jest parametrem używanym do sterowania liczbą świetlików w sąsiedztwie.

Algorytm 2 Algorytm roju świetlików (GSO)

Require: C, M
  wylosuj populację świetlików
  while nie osiągnięto warunku stopu **do**
    zmodyfikuj poziom lucyferny każdego świetlika
    przemieść każdego świetlika (7)
    for każdego świetlika $j \in N_i(t)$ **do**
      wyznacz prawdopodobieństwo przemieszczenia (6)
      przemieść świetliki (8)
      zmodyfikuj zakres sąsiedztwa (9)
    end for
  end while

3 Wersja równoległa - pamięć wspólna

Implementacja algorytmów została wykonana w języku C++ 11. Do implementacji i testów wykorzystano zintegrowane środowisko programistyczne Code Blocks. Do realizacji wersji równoległej algorytmów z pamięcią wspólną wykorzystano środowisko OpenMP.

3.1 Sprzęt

Procesor	Intel Core i7-4770K 3.5 GHz, 4 rdzenie, 8 wątków (korzystanie z hiper-wątków)
Pamięć RAM	16 GB
System operacyjny	Windows 7

3.2 Wyniki

Dla każdego z algorytmów wykonano 10 symulacji rozwiązania problemów o różnych złożonościach (liczbie możliwych elementów do spakowania). Symulacje te zostały wykonane w wersji sekwencyjnej oraz równoległej dla czterech i ośmiu wątków. Większej liczby wątków nie ma sensu rozważać, ponieważ używany sprzęt nie wystarczy do większego zrównoleglenia obliczeń.

Ze względu na zależność kolejnych kroków wykonania algorytmu (wyznaczanie atrakcyjności świetlików, ich poruszanie, zmiana liderów/sąsiadów) od rozwiązań wyznaczonych w następujących po sobie iteracjach nie można zrównoleglić pętli głównych. Zostały zrównoleglone wszystkie pętle wewnętrzne wykonywane w powyższych krokach.

3.2.1 Przyśpieszenie obliczeń

Obliczenia były prowadzone przy założeniach, że rozpatrujemy populację dwustu świetlików, a maksymalna liczba iteracji jaką może wykonać algorytm jest stała i równa tysiąc. Zestawienie średnich czasów wykonania programu i osiągniętego przyśpieszenia w zależności od zrównoleglenia prezentują tabele 1 i 2.

Tabela 1: Średni czas wykonania algorytmu świetlika (FA) w sekundach.

złożoność	sekwencyjnie		4 wątki		8 wątków	
	czas	wariancja	czas	wariancja	czas	wariancja
24	0.024	2,06e-05	0,079	7,8e-04	0,072	4,37e-04
120	0,248	0,003	0,213	0,028	0,201	0,013
600	1,02	0,136	0,893	0,221	0,764	0,097
960	2,766	0,684	1,309	0,349	1,146	0,214

Tabela 2: Średni czas wykonania algorytmu roju świetlików (GSO) w sekundach.

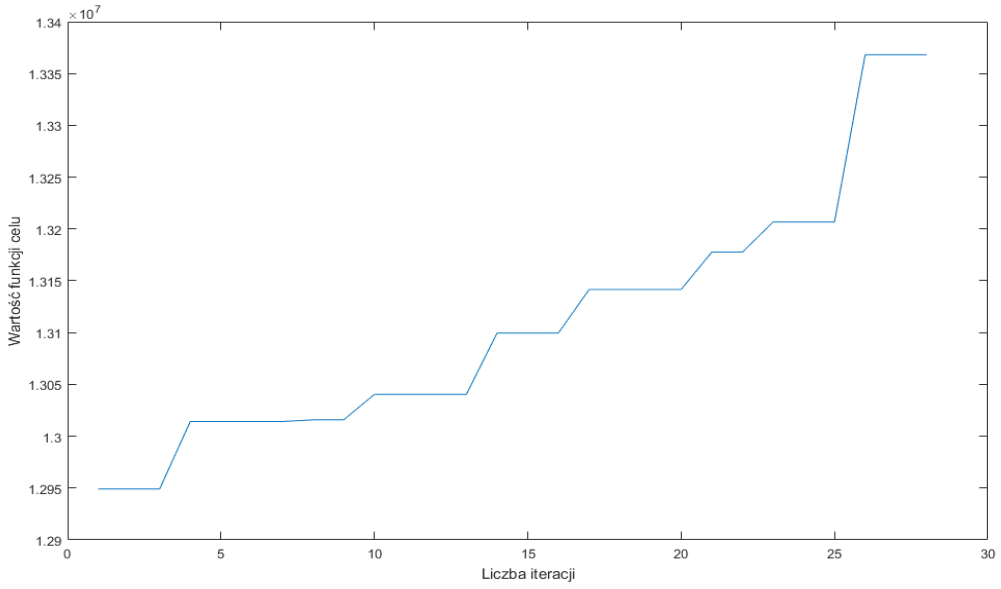
złożoność	sekwencyjnie		4 wątki		8 wątków	
	czas	wariancja	czas	wariancja	czas	wariancja
24	0,013	6,36e-06	0,035	4,51e-05	0,037	5,05e-05
120	0,141	9.3e-04	0,13	7.8e-04	0,122	6,9e-04
600	0,545	0,039	0,433	0,025	0,0389	0,02
960	1,547	0,207	0,895	0,072	0,7	0,043

Na podstawie tabeli 3 można zauważyć, że dla problemów o małej złożoności nastąpiło spowolnienie działania algorytmów zrównoleglonych w stosunku do wersji sekwencyjnej. Liczba operacji związanych z zarządzaniem nowymi wątkami przerasta złożoność problemu. Dopiero dla większych przypadków można zaobserwować przyśpieszenie.

Tabela 3: Przyśpieszenie wykonania algorytmów FA i GSO

złożoność	FA		GSO	
	4 wątki	8 wątków	4 wątki	8 wątków
24	0,303	0,332	0,375	0,355
120	1,106	1,239	1,093	1,16
600	1,142	1,336	1,259	1,403
960	2,113	2,414	1,729	2,225

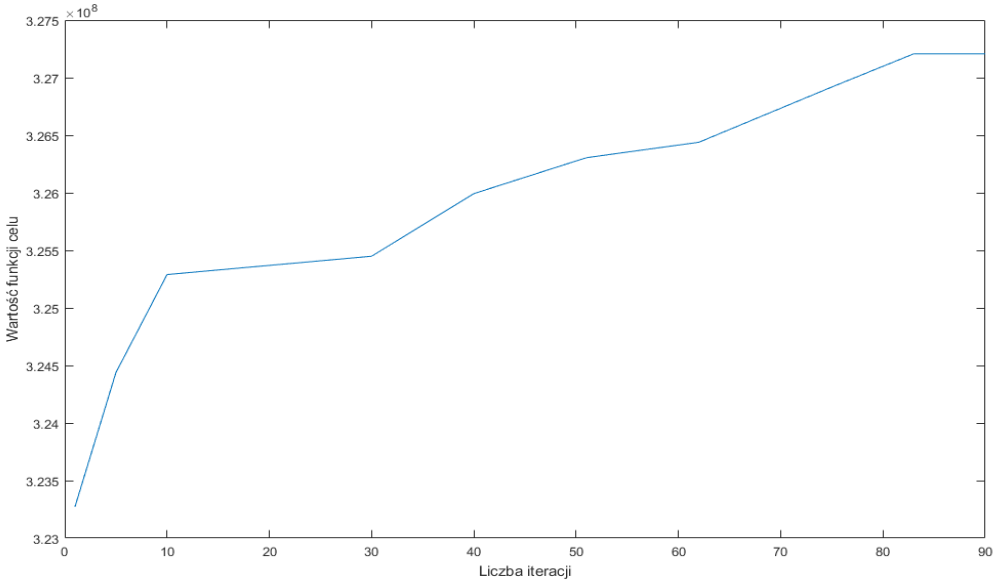
3.2.2 Zbieżność algorytmów



Wykres 1: Zbieżność algorytmu świetlika (FA) dla 24 elementów.

W każdym z testów zbieżności algorytmu znane było najlepsze rozwiązanie problemu plecakowego, na podstawie którego wyznaczony był ϵ. Jest to parametr wykorzystywany do wyznaczania zbieżności oznaczający procentową wartość o jaką może różnić się rozwiązanie znalezione przez algorytm w stosunku do rozwiązania optymalnego. Na potrzeby testów został on ustalony $\epsilon = 0.04$. Przez zbieżność algorytmu rozumiemy dostatecznie bliskie rozwiązanie w stosunku do rozwiązania optymalnego, które nie zmienia się przez określoną liczbę iteracji.

Na podstawie wykresów 1 i 2 można zauważyć, że liczba iteracji potrzebnych do rozwiązania problemu na założonym poziomie dokładności rośnie wraz z rozmiarem problemu. Ponadto dla problemów o dużych rozmiarach (600 elementów) charakterystyczny jest większy początkowy wzrost wartości funkcji celu w stosunku do dalszych iteracji.



Wykres 2: Zbieżność algorytmu świetlika (FA) dla 600 elementów.

Algorytm roju świetlików (GSO) nie nadaje się naszym zdaniem do problemów dyskretnych. Początkowo wylosowane najlepsze rozwiązanie pozostaje nim aż do końca działania algorytmu. Świetliki poruszają się wokół lokalnych maksimów, lecz liderzy (najlepsze lokalne rozwiązania) mają zbyt małe możliwości poruszania się.

4 Wersja równoległa - pamięć lokalna

Do realizacji wersji równoległej algorytmów z pamięcią lokalną wykorzystano środowisko OpenMPI. Ze względu na problemy z zestawieniem połączenia między komputerami osobistymi, obliczenia zostały wykonane w laboratorium wydziałowym - lab011.

W ramach testów został przyjęty model komunikacji między komputerami, w którym istnieje jeden węzeł główny oraz wiele węzłów wykonujących zadania. Węzeł główny odpowiedzialny jest za wczytanie danych o rozwiązywanym problemie, przesłaniu ich do węzłów wykonujących, a na koniec odebranie od nich wyników wykonanego algorytmu. Węzły obliczeniowe wykonują ten sam algorytm - FA lub GSO (w zależności od parametru podanego przez węzeł główny).

Algorytmy wykonywane w węzłach obliczeniowych wykonują się w wersji sekwencyjnej (brak zrównolegleń w ramach algorytmu). Zrównoleglenie polega na wielokrotnym wykonaniu tego samego algorytmu na wielu procesorach w tym samym czasie. Do wyznaczenia przyśpieszenia porównany będzie czas wykonania w wersji rozproszonej na k procesorach oraz czas wykonania na jednym procesorze k-krotnie.

Czas wykonania powinien być niewiele większy niż pojedyncze wykonanie wersji sekwencyjnej algorytmu. Do porównania wydajności, liczba świetlików oraz liczba iteracji algorytmu została przyjęta taka sama jak w wersji równoległej z pamięcią wspólną. Kryterium zbieżności nie było badane, ponieważ wyniki nie będą się różnić od już znanych.

4.1 Wyniki

Dla każdego z algorytmów wykonano 10 symulacji rozwiązywania problemów o różnych złożonościach. Średnie czasy wykonania algorytmów zostały przedstawione w tabelach 4 i 5.

Algorytm roju świetlików okazuje się wydajniejszy od algorytmu świetlika niemal dwukrotnie.

Tabela 4: Średni czas wykonania algorytmu świetlika (FA) w sekundach.

złożoność	sekwencyjnie		4 wątki		8 wątków	
	czas	wariancja	czas	wariancja	czas	wariancja
24	6,176	0,013	6,72	0,013	7,211	0,005
120	54,813	2,117	60,525	1,487	65,327	0,83
600	359,226	120,37	387,454	16,39	423,545	23,65
960	566,036	43,172	621,353	116,949	677,225	32,494

Tabela 5: Średni czas wykonania algorytmu roju świetlików (GSO) w sekundach.

złożoność	sekwencyjnie		4 wątki		8 wątków	
	czas	wariancja	czas	wariancja	czas	wariancja
24	6,847	0,519	8,309	0,027	9,091	0,036
120	36,86	4,257	42,641	1,11	45,489	1,444
600	183,17	198,822	200,584	26,962	220,316	12,953
960	288,149	118,902	321,373	115,337	351,474	80,838

Uzyskane przyśpieszenie zestawione w tabeli 6 pokazuje, że korzystając z 4 węzłów obliczeniowych można otrzymać niemal 3.5 krotne przyśpieszenie wykonania algorytmów. W przypadku 8 węzłów przyśpieszenie jest około 6.5 krotne. Przyśpieszenie uzyskane przez oba algorytmy jest podobne, z lekką przewagą algorytmu FA.

Tabela 6: Przyśpieszenie wykonania algorytmów FA i GSO

złożoność	FA		GSO	
	4 wątki	8 wątków	4 wątki	8 wątków
24	3,676	6,852	3,296	6,025
120	3,623	6,712	3,458	6,482
600	3,709	6,785	3,653	6,651
960	3,644	6,668	3,586	6,559

5 Wnioski

Testy rozwiązań zostały przeprowadzone w dwóch różnych środowiskach oraz innym sprzęcie obliczeniowym. Mimo to, można wskazać lepsze z rozwiązań - wersję równoległą z pamięcią lokalną. Wynika to z większego przyśpieszenia obliczeń pomimo wolniejszego sprzętu.

Sposób zrównoleglenia także istotnie wpływa na wydajność badanych algorytmów. W przypadku OpenMP zrównoleglenie jest bardzo drobne i stosowane do pętli wewnętrznych kolejnych funkcji algorytmów. Dla OpenMPI zrównoleglamy wykonanie całego algorytmu, dzięki czemu w tym samym czasie otrzymujemy wiele rozwiązań.

Wyniki jakościowe algorytmów są dość zbliżone do siebie, natomiast wydajniejszy jest algorytm GSO.

Literatura

[1] `http://www-users.mat.uni.torun.pl/~henkej/knapsack.pdf`

[2] http://pe.org.pl/articles/2014/6/37.pdf

[3] http://home.agh.edu.pl/~slukasik/pub/021_Lukasik_KAEiOG2011(presentation).pdf

[4] https://eti.pg.edu.pl/documents/176546/25263566/SZ_wyklad2.pdf