Problem kolorowania grafu dla algorytmów mrówkowych

Algorytmy metaheurystyczne

Józef Piechaczek 244761

2020-06-13

1 Algorytm mrówkowy

Algorytm mrówkowy to metaheurystyka populacyjna używana do rozwiązywania trudnych kombinatorycznych problemów optymalizacyjnych. Algorytm zainspirowany jest rzeczywistym zachowaniem mrówek szukających pożywienia. Mrówki początkowo poruszają się w sposób losowy. W momencie, gdy mrówka znajduje pożywienie, wraca ona do swojej kolonii, pozostawiając na ścieżce feromon - substancję, która ma sugerować innym mrówkom, że dana trasa prowadzi do pożywienia. W momencie, gdy feromon znajduje się już na ziemi, mrówki potrafią go wyczuć i preferują wybór ścieżki z większym natężeniem zapachu. Jeśli dana trasa była krótsza, natężenie feromonu na danym odcinku jest większe - wynika to z faktu, że mrówka pokonuje trasę w krótszym czasie. Feromon po pewnym czasie paruje - co jest zjawiskiem pozytywnym, gdyż pozwala na eksplorację innych miejsc. Ogólny algorytm mrówkowy można przedstawić w następujący sposób:

Algorytm 1 ACO

```
while not termination_condition do
  generateSolution()
  deamonActions()
  updatePheromone()
end while
```

2 Problem kolorowania grafu

Niech $G = (V, E)$ będzie grafem prostym nieskierowanym o zbiorze wierzchołków V i zbiorze krawędzi E. *k-kolorowanie* grafu G to taka funkcja $f : V \longrightarrow C$, gdzie $|C| = k$. *k-kolorowanie* grafu G jest poprawne wtedy, gdy $\forall (u, v) \in E, f(u) \neq f(v)$, czyli incydentne wierzchołki mają różne kolory. Liczba chromatyczna $\chi(G)$ zdefiniowana jest jako takie minimalne k, że istnieje poprawne *k-kolorowanie* grafu G. Problem kolorowania grafu, polega na znalezieniu takiego *k-kolorowania* grafu G, że $k = \chi(G)$. Jest on NP-trudny. Istnieje algorytm, który potrafi przybliżyć liczbę chromatyczną, działający w czasie $O(|V|(loglog|V|)^2/log(|V|)^3)$ [1]. Ponieważ problem jest NP-trudny, a algorytmy aproksymacyjne nie są zbyt efektywne do tego problemu często używa się heurystyk.

3 Metaheurystyczne podejścia do problemu kolorowania grafu

W algorytmach bazujących na ACO posłużymy się "komputerowymi mrówkami", które będą komunikowały się ze sobą w celu zbudowania jak najlepszego rozwiązania. Mrówki będą poruszać się po wierzchołkach grafu podejmując lokalnie decyzje o wyborze ścieżki, wybierając z większym prawdopodobieństwem ścieżki o wyższej wartości feromonu.

Rozwiązania zaproponowane w rozpatrzonych pracach [2–5] można podzielić na dwie główne kategorie: metody konstrukcyjne oraz metody ulepszające. Metody konstrukcyjne używają mrówek w celu utworzenia jak najlepszego, akceptowalnego rozwiązania. Metody ulepszające korzystają z innego algorytmu w celu utworzenia akceptowalnego rozwiązania, a następnie korzystają z mrówek, aby zminimalizować liczbę konfliktów/kolorów. W tej pracy skupię się na metodach konstrukcyjnych.

4 Zarys metody konstrukcyjnej

W metodach konstrukcyjnych informację o feromonie pomiędzy dwoma wierzchołkami przechowywać będziemy w macierzy $T_{n\times n}$, gdzie $n = |V|$, która początkowo zawiera domyślne natężenia feromonu.

Mrówki umieszczane są na wierzchołkach w zadanym statycznym porządku. Przykładami takich porządków są [4]:

- RANDOM - wierzchołki uporządkowane losowo
- LF - wierzchołki posortowane malejąco względem ich stopnia
- SL - porządek $v_1, \ldots, v_n$ taki, że każdy wierzchołek v_i ma najmniejszy stopień w podgrafie indukowanym przez wierzchołki $v_1, \ldots, v_i$.

Mrówki, aby nie kolorować jednego wierzchołka wielokrotnie, przechowują informacje odnośnie dozwolonych/niedozwolonych wierzchołków.

Jedna iteracja algorytmu trwa, dopóki każda mrówka znajdzie akceptowalne rozwiązanie. W pojedynczym kroku iteracji, mrówka musi podjąć decyzję, jaki wierzchołek powinien zostać pokolorowany jako następny. Podczas iteracji k, gdy mrówka znajduje się w wierzchołku v, wybiera wierzchołek u

z prawdopodobieństwem $p_{v,u}^k(T)$ [3]:

$$p_{v,u}^k(T) = \begin{cases} \frac{(T_{v,u})^\alpha (\eta_{v,u})^\beta}{\sum_{l \in N_i^k} (T_{v,l})^\alpha (\eta_{v,l})^\beta} & , u \text{ jest dozwolony} \\ 0 & , u \text{ jest niedozwolony} \end{cases} \quad (1)$$

gdzie $T_{v,u}$ oznacza natężenie feromonu pomiędzy wierzchołkami v i u, $\eta_{v,u}$ oznacza wartość komponenty dla v i u, N_i^k jest zbiorem dozwolonych kolejnych wierzchołków, a α i β zadanymi parametrami.

Po wykonaniu kroku mrówka koloruje wylosowany wierzchołek na akceptowalny kolor, zapamiętuje wierzchołek jako już pokolorowany oraz aktualizuje listę dozwolonych/niedozwolonych wierzchołków. W zależności od algorytmu wykonywana jest również lokalna aktualizacja natężenia feromonu.

W momencie gdy wszystkie mrówki znajdą akceptowalne rozwiązanie następuje globalna aktualizacja natężenia feromonu. Aby uniknąć zatrzymania w lokalnym optimum może również nastąpić wietrzenie feromonu w danej iteracji.

Algorytm powtarza się do momentu wystąpienia warunku końcowego.

5 Konkretny algorytm

Najważniejszym elementem algorytmu jest wybranie sposobu obliczania wartości komponenty. Sposoby te bazują na innych algorytmach, takich jak DSATUR [2–4], RLF [2–5], czy XRLF [2]. Postanowiłem opracować algorytm bazujący na metodzie DSATUR, w oparciu o prace [3, 4].

5.1 DSATUR

DSATUR to zachłanny algorytm, używający dynamicznego porządkowania wierzchołków. Algorytm korzysta z pojęcia stopnia saturacji wierzchołka, zdefiniowanego jako liczba różnych kolorów w sąsiedztwie wierzchołka. Algorytm zaczyna pracę w wierzchołku o najwyższym stopniu, który kolorowany jest pierwszym kolorem. Algorytm następnie przechodzi do wierzchołka o najwyższym stopniu saturacji. Jeśli jest wiele takich wierzchołków wybierany jest wierzchołek o najwyższym stopniu. Dalsze remisy są rozwiązywane losowo. Wierzchołek do którego przechodzimy kolorowany jest na pierwszy możliwy kolor. Algorytm powtarzamy, aż cały graf zostanie pokolorowany.

Zdefiniujmy teraz konkretny algorytm, oparty na ogólnej metodzie zadanej w punkcie 4.

Mrówki zostaną rozdysponowane losowo na wierzchołki grafu. Mrówka w danej iteracji będzie przechowywać informację mówiącą, które wierzchołki nie są jeszcze pokolorowane. Wierzchołki te będą wierzchołkami dozwolonymi. Początkowe wartości feromonu zdefiniowane są w następujący sposób:

$$T_{v,u} = \begin{cases} 0, & (v,u) \in E \\ 1, & (v,u) \notin E \end{cases}$$

Jako wartość komponenty przyjmujemy:

$$\eta_{v,u} = dsat(u)$$

Mrówka w danym kroku iteracji, z prawdopodobieństwem q_0 zadanym jako parametr, wybierać będzie najlepszy możliwy wierzchołek, tj. wierzchołek o najwyższym iloczynie $W_{v,u} = (T_{v,u})^{\alpha}(\eta_{v,u})^{\beta}$. Z prawdopodobieństwem $(1 - q_0)$ mrówka wybierze wierzchołek zgodnie z regułą (1) z dozwolonych wierzchołków.

Po wybraniu wierzchołka mrówka usunie go z lity dozwolonych, pokoloruje go na pierwszy możliwy kolor, a następnie dokona lokalnej aktualizacji natężenia feromonu w następujący sposób:

$$T_{v,u} = (1-p)T_{v,u} + T^0_{v,u} \tag{2}$$

gdzie v to wierzchołek z którego przechodzimy, u to wierzchołek na który przechodzimy, $T^0_{v,u}$ oznacza początkową wartość feromonu pomiędzy wierzchołkami v i u, a p to współczynnik trwałości feromonu.

W momencie, gdy każda mrówka ukończy kolorowanie, tj. gdy zbiór dozwolonych wierzchołków każdej z mrówek będzie pusty, następuje aktualizacja najlepszego rozwiązania oraz globalna aktualizacja natężenia feromonu zgodnie ze wzorem

$$(\forall (v_i, v_{i+1}) \in P^*)(T_{v_i,v_{i+1}} = \epsilon T_{v_i,v_{i+1}} + T^0_{v_i,v_{i+1}}) \tag{3}$$

,gdzie ϵ jest parametrem określającym trwałość feromonu podczas globalnej aktualizacji, a P^* oznacza ciąg wierzchołków pokonanych przez mrówkę, która uzyskała najlepszy wynik.

Jeśli w algorytmie nastąpiła stagnacja, tj. przez i_e iteracji nie nastąpiła aktualizacja najlepszego wyniku, następuje wietrzenie feromonu zgodnie ze wzorem:

$$T_{v,u} = (1 - p)T_{v,u} \tag{4}$$

Co więcej jeśli przez i_r iteracji nie nastąpiła zmiana wyniku, algorytm zaczyna pracę od nowa. Wartości i_e i i_r są parametrami programu. Algorytm powtarza się przez i_{max} iteracji.

Parametry programu podsumowane są w poniższej tabeli:

Parametr	Dziedzina	Opis
$nants$	$[1, 100]$	określa rozmiar kolonii mrówek
q_0	$[0, 1]$	określa prawdopodobieństwo wybrania najlepszego wierzchołka w danym kroku iteracji
α	$[0, \infty]$	określa ważność feromonu
β	$[0, \infty]$	określa ważność wartość komponenty
p	$[0, 1]$	określa trwałość feromonu
ϵ	$[0, 1]$	określa trwałość feromonu podczas globalnej aktualizacji
i_e	$[1, \infty]$	określa ilość iteracji bez lepszego wyniku potrzebną do wykonania wietrzenia feromonu
i_r	$[1, \infty]$	określa ilość iteracji bez lepszego wyniku potrzebną do zresetowania algorytmu
i_{max}	$[1, \infty]$	określa maksymalną liczbę iteracji

Tabela 1: Parametry programu

Pseudokod algorytmu:

Algorytm 2

ants = initAnts()
for i in 1:nants **do**
  while $(\exists ant)(ant.uncoloredV \neq \emptyset)$ **do**
    for ant in ants **do**
      if $ant.uncoloredV \neq \emptyset$ **then**
        if rand() $< q_0$ **then**
          $vnext = x \in ant.uncoloredV$, gdzie $W_{ant.current,x}$ jest maksymalne
        else
          $vnext = x \in ant.uncoloredV$, wylosowane zgodnie z (1)
        end if
        $ant.solution[vnext]$ = pierwszy możliwy kolor
        aktualizacja feromonu zgodnie z (2)
        $ant.uncoloredV = ant.uncoloredV - \{ant.vnext\}$
        $ant.current = vnext$
      end if
    end for
  end while
  $best\_color$ = najlepsze kolorowanie
  $best\_cost$ = najlepszy koszt
  $best\_path$ = ścieżka najlepszego kolorowania
  aktualizacja feromonu zgodnie z (3)
  ants = initAnts()
  if to samo rozwiązanie od i_e iteracji **then**
    wietrzenie feromonu zgodnie z (4)
  end if
  if to samo rozwiązanie od i_r iteracji **then**
    uruchom algorytm od nowa, zapamiętując najlepszy wynik
  end if
end for
return $best\_color$, $best\_cost$

6 Testy

Algorytm został zaimplementowany w języku Julia. Początkowe testy wykonywałem na grafach o niewielkiej liczbie wierzchołków (< 50), grafach wielodzielnych (multipartite graph) oraz grafach zbliżonych do nich. Algorytm bezproblemowo odnajdywał optimum w niewielkiej liczbie iteracji. W celu utrudnienia kolejne testy rozpatrzyłem na grafach DIMAC [8], których liczby chromatyczne są trudne do oszacowania oraz mających wiele krawędzi. Wyniki przedstawia następująca tabela (opt oznacza najniższą odkrytą do tej pory liczbę chromatyczną, LF oznacza algorytm zachłanny bazujący na porządku LF) :

$\|V\|$	$\|E\|$	$nants$	q_0	α	β	p	ϵ	i_e	i_r	i_{max}	ACODSAT	LF	opt
500	62624	40	0.5	2	4	0.5	0.35	5	10	5	67	71	??
500	62624	40	0.5	2	4	0.5	0.35	5	10	15	66	71	??
250	31336	40	0.5	2	4	0.5	0.35	5	10	5	38	41	??
250	31336	40	0.5	2	4	0.5	0.35	5	10	15	37	41	??
250	14849	40	0.5	2	4	0.5	0.35	5	10	5	68	70	65
250	14849	40	0.5	2	4	0.5	0.35	5	10	15	67	70	65
500	58862	40	0.5	2	4	0.5	0.35	5	10	5	128	134	122
500	58862	40	0.5	2	4	0.5	0.35	5	10	15	127	134	122
1000	238267	40	0.5	2	4	0.5	0.35	5	10	5	248	259	234
1000	238267	40	0.5	2	4	0.5	0.35	5	10	15	247	259	234
500	62624	40	0.5	4	4	0.7	0.8	2	5	5	67	71	??
500	62624	40	0.5	4	4	0.7	0.8	2	5	15	67	71	??
250	31336	40	0.5	4	4	0.7	0.8	2	5	5	38	41	??
250	31336	40	0.5	4	4	0.7	0.8	2	5	15	37	41	??
250	14849	40	0.5	4	4	0.7	0.8	2	5	5	68	70	65
250	14849	40	0.5	4	4	0.7	0.8	2	5	15	68	70	65
500	58862	40	0.5	4	4	0.7	0.8	2	5	5	129	134	122
500	58862	40	0.5	4	4	0.7	0.8	2	5	15	127	134	122

7 Obserwacje i wnioski

Algorytm oszacował liczbę chromatyczną lepiej niż standardowy algorytm zachłanny. Testy podzieliłem na dwie części: w pierwszej części znajdują się parametry powodujące że natężenie feromonu gra mniejszą rolę (eksploracja), w drugiej większą (eksploatacja). Algorytm sprawdził się lepiej dla pierwszego przypadku, choć różnice były znikome. Algorytm testowałem dla

różnych wartości parametrów. Parametry umieszczone w tabeli okazały się najkorzystniejsze, zatem są one jednocześnie parametrami rekomendowanymi. Główną trudnością algorytmu jest jego wysoka złożoność obliczeniowa. Choć standardowy algorytm DSATUR ma teoretyczną złożoność obliczeniową $O(n^2)$, w naszym wypadku występuje jednak szereg innych czynników, wpływających na wyższą złożoność. Czynnikami tymi są na przykład konieczność budowania macierzy natężenia feromonu oraz ciągłej jej aktualizacji czy skomplikowany proces wyboru kolejnego wierzchołka.

Algorytmy omówione w cytowanych pracach [2–5] różnią się głównie metodą obliczania wartości komponenty oraz sposobem umieszczania mrówek na wierzchołkach. W poszukiwaniu rozwiązania o najniższym koszcie, rozważyłem metody losowego rozmieszczania mrówek (RANDOM) oraz według stopni wierzchołków (LF). Algorytm oparty na RANDOM okazał się nieznacznie lepszy. Rozważyłem również użycie metody RLF do obliczania wartości komponenty, jednak na danych testowych osiągała one wartości niewiele gorsze niż DSATUR, choć w ogólnym przypadku zazwyczaj bywa odwrotnie. Argumentem przemawiającym za metodą DSATUR była również niższa złożoność obliczeniowa - algorytm RLF ma złożoność wynoszącą $O(n^3)$ [7], co staje się zauważalne przy rozpatrywaniu grafów o dużej liczbie wierzchołków.

Literatura

[1] M. Halldórsson, "A still better performance guarantee for approximate graph coloring," *Information Processing Letters*, vol. 45, pp. 19–23, 01 1993.

[2] S. Ahn, S. Lee, and T. Chung, "Modified ant colony system for coloring graphs," pp. 1849 – 1853 vol.3, 01 2004.

[3] M. Bessedik, R. Laib, A. Boulmerka, and H. Drias, "Ant colony system for graph coloring problem," pp. 786– 791, 12 2005.

[4] C. D and A. Hertz, "Ants can colour graphs," *Journal of the Operational Research Society*, vol. 48, pp. 295–305, 03 1997.

[5] E. Salari and K. Eshghi, "An aco algorithm for graph coloring problem," vol. 3, p. 5 pp., 01 2005.

[6] M. Adegbindin, A. Hertz, and M. Bellaiche, "A new efficient rlf-like algorithm for the vertex coloring problem," *Yugoslav Journal of Operations Research*, vol. 26, pp. 3–3, 01 2016.

[7] F. Leighton, "A graph coloring algorithm for large scheduling problems," *Journal of Research of the National Bureau of Standards*, vol. 84, 11 1979.

[8] P. K. Daniel Porumbel, Jin Kao Hao, "Dimacs graphs: Benchmark instances and best upper bounds." `http://cedric.cnam.fr/~porumbed/graphs/`, 2011. [Online; accessed 14-06-2020].