Problem izomorfizmu podgrafu

Paciorek Agata Pelc Jakub Korecki Tomasz

1. Wstęp

W ramach projektu z przedmiotu *Algorytmy decyzyjne i teoria złożoności* zajmowaliśmy się problemem izomorfizmu podgrafu. Jest on uogólnieniem problemu maksymalnej kliki oraz rozważań na temat istnienia cyklu Hamiltona w grafie, należy do klasy problemów NP – zupełnych.

Izomorfizm podgrafu znajduje zastosowanie głównie w dziedzinie informatyki chemicznej, gdzie wykorzystywany jest do znalezienia podobieństwa w strukturze związków chemicznych. Ponadto stosowany jest w bioinformatyce oraz matematycznym modelowaniu sieci społecznych.

2. Opis problemu

Mówimy, że dwa grafy $G_A = (V_A, E_A)$ i $G_B = (V_B, E_B)$ są *izomorficzne*, co zapisujemy $G_A \cong G_B$, jeżeli istnieje przekształcenie wzajemnie jednoznaczne $f: V_A \to V_B$ takie, że $(v, w)\ \epsilon\ E_A$ wtedy i tylko wtedy, gdy $\big(f(v), f(w)\big)\ \epsilon\ E_B$, tj. takie wzajemnie jednoznaczne przekształcenie wierzchołków grafu G_A na wierzchołki grafu G_B, które zachowuje relacje przylegania wierzchołków.

Problem izomorfizmu podgrafu zdefiniowany jest następująco: dla podanych grafów G_A oraz G_B określić, czy istnieje podgraf $G' = (V', E')$ taki, że $V' \subseteq V_B$ oraz $E' \subseteq E_B$ oraz spełniona jest równość $G_A \cong G'$.

Najbardziej znane algorytmy potrafiące rozwiązać problem izomorfizmu podgrafu to algorytm J. R. Ullmanna oraz algorytm VF2 (P. Foggia, M. Vento).

3. Opis algorytmu

Zaimplementowany przez nas algorytm to klasyczny przykład algorytmu genetycznego. Na początku losowana jest populacja początkowa o określonej wielkości, która w kolejnych iteracjach reprodukowana jest z zastosowaniem odpowiednich metod selekcji, krzyżowania oraz mutacji z odpowiednim prawdopodobieństwem. Algorytm kończy działanie, gdy warunek stopu zostanie spełniony lub w przypadku wygenerowania idealnego osobnika. Poniżej zaprezentowany został pseudokod naszego algorytmu:

```
algorytm_genetyczny() {
      populacja = populacja_początkowa;
      wykonuj:
            populacja = nastepna_generacja(populacja);
      aż (warunek stopu lub znaleziono rozwiązanie);
}

nastepna_generacja(populacja) {
      wykonuj:
            chromosom_1 = metoda_selekcji.wybierz();
            chromosom_2 = metoda_selekcji.wybierz();
            metoda_krzyżowania.krzyżuj(chromosom_1, chromosom_2);
            metoda_mutacji.mutuj(chromosom_1);
            metoda_mutacji.mutuj(chromosom_2);
            nowa_populacja.dodaj(chromosom_1);
            nowa_populacja.dodaj(chromosom_2);
      aż (nowa_populacja jest pełna);
}
```

3.1. Reprezentacja chromosomów

Wierzchołki obu grafów są numerowane ($V[0], V[1]$ …). Chromosom jest permutacją wierzchołków większego grafu. Przykład: Załóżmy, że $G_A = (V_A, E_A)$ jest grafem o mniejszej liczbie wierzchołków, natomiast $G_B = (V_B, E_B)$ jest grafem o większej liczbie wierzchołków. Jeżeli na 3 pozycji (element o indeksie 2) w chromosomie znajduje się liczba 20, to oznacza, iż w tym konkretnym osobniku wierzchołkowi $V_A[2]$ w mniejszym grafie został przyporządkowany wierzchołek $V_B[20]$ w grafie większym.

3.2. Funkcja fitness

Zastosowana przez nas funkcja oceniająca składa się z dwóch części. Istotniejsza z nich iteruje po wszystkich krawędziach mniejszego grafu i dla każdej z nich sprawdza, czy istnieje krawędź pomiędzy wierzchołkami większego grafu, które zostały przyporządkowane do jej incydentnych wierzchołków.

Druga część funkcji iteruje po wszystkich wierzchołkach mniejszego grafu i dla każdego z nich sprawdza, czy odpowiadający mu wierzchołek w grafie większym ma stopnie wejściowy i wyjściowy co najmniej takie jak on.

3.3. Metody selekcji

Ich zadaniem jest wybieranie z obecnej populacji odpowiednich osobników, które umieszczone zostaną w nowej populacji. Wybór ten musi być uzależniony od wartości funkcji oceniającej: im wyższa wartość tej funkcji zostanie przyporządkowana danemu wierzchołkowi, tym większa szansa, iż ten konkretny wierzchołek znajdzie się w nowej populacji. Zastosowane przez nas metody to:

- *ruletkowa* – liczymy sumę F_{sum} wszystkich wartości funkcji oceniającej oraz wkład każdego osobnika w sumę: $p(x_i) = F(x_i)/F_{sum}$. Następnie wartości $p(x_i)$ traktujemy jako rozkład

prawdopodobieństwa i dokonujemy losowania zgodnie z tym rozkładem,

- *turniejowa* – z populacji losujemy k osobników (gdzie $k = rozmiar\_populacji/8$). Następnie z tego zbioru wybieramy najlepszego osobnika.

3.4. Metody krzyżowania

Krzyżowanie to podstawowy operator stosowany w algorytmach genetycznych. My zastosowaliśmy trzy takie operatory:

- *cykliczne (ang. cycle crossover)* – tworzy potomstwo, którego każda pozycja w chromosomie jest skopiowana z odpowiedniej pozycji jednego z rodziców. Kopiowanie rozpoczynamy od pierwszego elementu w chromosomie. Dokonany wybór implikuje następny, jeśli nie chcemy powielić dwóch takich samych elementów w jednym chromosomie. Po wyczerpaniu takiego cyklu w podzbiorze elementów, wybieramy któregokolwiek z rodziców do skopiowania kolejnej wolnej pozycji i powtarzamy ciąg decyzyjny,
- *z zachowaniem porządku* (ang. *order crossover*) – losowane są dwa punkty w chromosomach rodziców. Do pierwszego potomka kopiowany jest fragment pomiędzy nimi z pierwszego rodzica. Fragment ten jest uzupełniany, począwszy od drugiego punktu krzyżowania, elementami z drugiego rodzica, które nie są jeszcze obecne w potomku (także zaczynając od drugiego punktu krzyżowania. Po dojściu do końca chromosomu uzupełniany jest jego początek. Drugi potomek powstaje przez symetryczne zastosowanie tej metody,
- *z częściowym odwzorowaniem* (ang. *partially maped crossover, PMX*) – losowane są dwa punkty w chromosomach rodziców. Odcinek pomiędzy nimi definiuje odwzorowanie: element z danej pozycji w jednym rodzicu zostanie zastąpiony w potomstwie

elementem z tej samej pozycji w drugim rodzicu (dotyczy to wszystkich wystąpień tych elementów, także spoza wylosowanego odcinka). Pozostałe elementy kopiowane są bez zmian z rodzica do potomka.

3.5. Metody mutacji

Odwzorowując ewolucję występującą w świecie rzeczywistym, algorytmy genetyczne również stosują mutacje, które powodują losowe zaburzenie struktury chromosomu. Oto zaimplementowane przez nas metody mutacji:

- *swap* – losujemy dwa elementy chromosomu i zamieniamy ich pozycje,
- *inversion* – losujemy dwie pozycje w chromosomie i odwracamy kolejność elementów zawartych pomiędzy nimi,
- *scramble* - losujemy dwie pozycje w chromosomie i mieszamy kolejność elementów zawartych pomiędzy nimi,
- *insertion* – losujemy element chromosomu i wstawiamy go na inną wylosowaną pozycję.

3.6. Warunki stopu

Do wyboru mamy dwa warunki stopu. Pierwszy to ilość iteracji, natomiast drugi to czas działania algorytmu. Niezależnie od wyboru algorytm zatrzymuje się natychmiast, gdy znajdzie poprawne rozwiązanie.

4. Opis aplikacji

Aplikacja została stworzona w języku Java, z zastosowaniem biblioteki JUNG oraz Swing. Graficzny interfejs użytkownika dostarcza użytkownikowi bardzo intuicyjną obsługę. Interfejs podzielony został na kilka części:

konfiguracyjna (generowanie grafu i parametry sterujące algorytmem), dwa panele, na których wyświetlane są grafy oraz log rejestrujący zdarzenia.

Możemy generować grafy skierowane i nieskierowane o zadanej ilości wierzchołków. Dodatkowo wyświetlać grafy możemy za pomocą 5 layoutów dostarczonych wraz z biblioteką JUNG:

- ISOMLayout
- KKLayout
- FRLayout
- CircleLayout
- SpringLayout

Działanie algorytmu możemy konfigurować na wiele sposobów. Należy podać liczebność populacji, rodzaj krzyżowania, rodzaj mutacji, rodzaj selekcji, warunek zakończenia algorytmu (oraz odpowiednią dla niego wartość: ilość iteracji lub ilość sekund), ustalić wagę dla dwóch wspomnianych wcześniej funkcji oceniających oraz prawdopodobieństwo wystąpienia krzyżowania i mutacji. Ponadto możemy wybrać, czy chcemy otrzymać wykres ilustrujący zmianę minimalnej, średniej i maksymalnej wartości funkcji oceniającej w kolejnych populacjach.