Dokumentacja wstępna

POP

Michał Brzeziński, Michał Sadlej

Zadanie: Sieć ograniczenia

Przy użyciu Algorytmu Ewolucyjnego zaprojektować sieć teleinformatyczną minimalizującą liczbę użytych systemów teletransmisyjnych o różnej modularności m, dla $(m = 1, m > 1, m >> 1)$.

Sieć opisana za pomocą grafu $G = (N,E)$, gdzie N jest zbiorem węzłów, a E jest zbiorem krawędzi.

Funkcja pojemności krawędzi opisana jest za pomocą wzoru $f_e(o) = \lceil o/m \rceil$.

Zbiór zapotrzebowań D, pomiędzy każdą parą węzłów opisuje macierz zapotrzebowań i jest dany.

Dla każdego zapotrzebowania istnieją co najmniej 2 predefiniowane ścieżki.

Sprawdzić jak wpływa na koszt rozwiązania agregacja zapotrzebowań, tzn. czy zapotrzebowanie realizowane jest na jednej ścieżce (agregacja), czy dowolnie na wszystkich ścieżkach w ramach zapotrzebowania (pełna dezagregacja).

Dobrać optymalne prawdopodobieństwo operatorów genetycznych oraz liczność populacji.

Dane pobrać ze strony http://sndlib.zib.de/home.action, dla sieci {\it polska}.

linki do danych:

https://sndlib.put.poznan.pl/download/sndlib-networks-native/polska.txt
https://sndlib.put.poznan.pl/home.action

link do opisu danych:

https://sndlib.put.poznan.pl/html/docu/io-formats/native/network.description.html

Wnioski

- Każde połączenie ma podstawową modułowość o danym koszcie z możliwością zainstalowania lepszej modułowości za odpowiednią dopłatą

- Rozwiązanie musi spełniać warunki z części DEMANDS

- ADMISSIBLE PATHS określa potencjalne ścieżki połączeń w DEMANDS

- Modularność - “pojemność” łącza

- Sieć opisana zbiorem węzłów i krawędzi

- Funkcja pojemności krawędzi - ceil(o/m)
- o - ilość danych do przesłania

- np
    - Jednostki w zadaniu nie zostały podane. Dla jasności przykładu użyto Mbps

    - potrzebne przesłanie o = 25 Mbps

    - modularność m = 10 Mbps

    - f_e(o) = ceil(25/10) = 3

    - potrzebna ilosć systemów transmicyjnych do przesłania wiadomości = 3

- Prawdopodobnie agregacja jest łatwiejsza do implementacji ale podatniejsza na przeciążenia przy skalowaniu w górę

Przykład

- Istnieją połączenia sieci Szczecin-Poznań-Wrocław i Bydgoszcz-Poznań-Wrocław
- Szczecin i Bydgoszcz muszą się połączyć z Wrocławiem i przesyłać 10 Mbps
- Mogą to robić wyłącznie przesyłając dane przez Poznań
- Jeśli modułowość krawędzi Poznań-Wrocław jest mniejsza niż 20 Mbps to nie będzie to możliwe

Założenia

- Modułowość można rozszerzyć, dopłacając.
- Nie można od razu wybudować rozszerzenia bez bazowej instalacji
- Rozwiązanie musi spełniać warunki z części DEMANDS

Realizacja

- gen został zaimplementowany jako słownik o podanej strukturze

```
ADMISSABLE_PATHS = {                    # <- gen
    "Demand_0_1" : {                    # <- chromosom
        "Path_0": int(sent_amount)      # <- allel
        .
        .
        .
        'Path_6 ': int(sent_amount)
    },
    .
    .
    .
    "Demand_10_11" : {
        "Path_0": int(sent_amount)
        .
        .
        .
    },
}
```

-

- Stworzyliśmy klasę EvolutionAlgorithm posiadającą wszystkie metody, by umożliwić łączenie modułów różnych algorytmów w jeden

- generatory
    - rand_split_without_aggregation - rozdzielający demand po wszystkich możliwych ścieżkach

    - rand_split_for_aggregation - rozdzielający wartośc z demand na dwie różne ścieżki z admissable paths po równo

- mutacje
    - mutate_for_aggregation - zabierający najmniejszą wartość przypisaną do admissable path i dodającą ją do innego admissable path danego demand

    - mutate_without_aggregation - odejmuje losową wartośc z allelu i dodaje ją do innego allelu danego chromosomu

    - mutate_with_switch - zamieniający losowo dwie wartośći allelów jednego chromosomu

- krzyżowanie
    - cross_for_aggregation - wybiera którego rodzica chromosom weźmie na podstawie, tego które wykorzystują mniej zasobów (na podstawie ilości ścieżek i ilości linków w ścieżce)

    - cross_without_aggregation - allele są liczone na podstawie średniej obu rodziców

- rodzice są wybierani poprzez tournament selection

- funkcja kosztu evaluate_cost, zlicza
    - “setup cost” dla każdego linku
    - koszty rozbudowania o moduły potrzebne by rozwiązanie z genu działało
    - funkcje kary za każdy link gdzie rozwiązanie w genie przekroczyło pojemność największego modułu

- nie jest to jeszcze w pełni zaimplementowane, ale wszystkie te opcje i moduły będzie możliwość wybrania w run_evolution_algorithm()

Eksperymenty

- Sprawdzenie wydajności i wyników algorytmu którego generator i funkcja krzyżowania może generować dane faworyzujące agregację albo dezagregację w każdym układzie.
    - Na przykład jak radzi sobie algorytm z generatorem agregacyjnym i krzyżowaniem dezagregacyjnym w porównaniu z algorytmem z generatorem i krzyżowaniem agregacyjnym

- Eksperymenty związane z wykorzystywaniem rozbudowy. Na przykład wprowadzenie kary lub nagrody w funkcji kosztów za rozbudowywanie struktur ponad bazową wersję

- Porównanie algorytmu priorytetyzującego najkrótsze połączenia i algorytmu który na to nie uważa

- Porównanie algorytmu genetycznego i ewolucyjnego różniczkowego

Wyniki

Wyniki aktualnych eksperymentów zostały umieszczone w pliku analisys.pdf w repozytorium