POLITECHNIKA WARSZAWSKA WYDZIAŁ EITI INSTYTUT INFORMATYKI

Dokumentacja ALHE SNDLib sieć Polska

Kacper Kula & Wojciech Sitek
26 stycznia 2021

1 ZADANIE PROJEKTOWE

Przy użyciu Algorytmu Ewolucyjnego zaprojektować sieć teleinformatyczną minimalizującą liczbę użytych systemów teletransmisyjnych o różnej modularności m (m = 1, m > 1 i 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, określonego w zadaniu. Zbiór zapotrzebowań D, pomiędzy każdą parą węzłów opisuje macierz zapotrzebowań i jest dany. Dla każdego zapotrzebowania istnieją co najmniej 3 predefiniowane ścieżki. Sprawdzić, jak wpływa na koszt rozwiązania agregacja zapotrzebowań, tzn. czy zapotrzebowanie realizowane jest na jednej ścieżce (pełna 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 polska.

2 WYJAŚNIENIE POJĘĆ

2.1 MODULARNOŚĆ

Modularność jest to ...

2.2 ALGORYTM EWOLUCYJNY

Zgodnie z wykładami prof. Jarosława Arabasa, Algorytm Ewolucyjny opisuje się za pomocą algorytmu, ukazanego w rozdziale 6.2. Na algorytm składają się następujące funkcjonalności:

1. inicjalizacja populacji,

2. główna pętla algorytmu ewolucyjnego,
3. mutacja (ang. *mutation*),
4. krzyżowanie (ang. *crossover*),
5. selekcja (ang. *selection*, oznaczane jako *select*),
6. warunek zatrzymania,
7. funkcja celu.

3 Założenia projektu

Projekt wykonywany jest w ramach przedmiotu Algorytmy Heurystyczne (ALHE) w semestrze zimowym 2020 na Wydziale EiTI Politechniki Warszawskiej. Prowadzącym projekt jest dr inż. Stanisław Kozdrowski.

Implementacja projektu jest wykonywana w języku Python. Algorytm ewolucyjny posiada własną implementację i nie jest zaczerpnięty z bibliotek zewnętrznych języka Python. Biblioteki narzędziowe języka Python, którymi się wspomagano, to między innymi biblioteki:

- xml - do przeprowadzenia parsowania pliku XML do obiektów Python (rozdział 4)
- tqdm - ukazywania paska postępu
- random - do generacji liczb pseudolosowych (użyto ziarna (ang. *seed*))
- logging - do logowania informacji pomocniczych w czasie działania programu
- json - do zapisu i odczytu plików JSON.

W projekcie zaimplementowano możliwość uruchomienia programu dla sieci w pełni zagregowanej lub sieci w pełni rozproszonej (dezagregacja, ang. *distributed*). Dodano także możliwość sterowania modularnością algorytmu.

4 Przygotowanie danych

Dla sieci „polska", pobrano plik XML oraz TXT ze wszystkimi informacjami dla danych dotyczących terytorium Polski. Następnie, przeprowadzono analizę budowy plików oraz znaczenia poszczególnych terminów. Wyeliminowano z programu dane nieistotne dla rozwiązywanego problemu. Analiza znaczenia terminów przeprowadzona jest w rozdziale 2.

Następnie, zbudowano parser, konwertujący wybrane części pliku XML do obiektów języka Python. Budowa słowników i list, w których były przechowywane informacje o sieci, została przedstawiona na poniższym przykładzie:

```
nodes = ['Gdansk', 'Bydgoszcz', ...]

link_keys = [
    (0, 1),
    (0, 2),
    (1, 3),
]

links_array = [
    Link_0_1_data,
    Link_0_2_data,
    ...,
    Link_1_3_data,
    ...,
]

Link_a_b_data = {
    'setupCost': 156.0,
    'capacity0': 155.0,
    'capacity1': 622.0,
    'cost0': 156.0,
    'cost1': 468.0,
}

demand_array = [
    Demand_0_1_data,
    Demand_0_2_data,
    ...
]

Demand_a_b_data = {
    'demand': 195.00,
    'admissiblePaths': [
        [(0,2), (1,2)],
        [(0,10), (1,10)],
        [(0,2), (2,9), (7,9), (1,7)],
        ...
    ]
}

demand_keys = [
    (0, 1),
    (0, 2),
    (1, 2),
```

]

5 MOŻLIWOŚĆ KONFIGURACJI PARAMETRÓW

Zaimplementowano możliwość konfiguracji parametrów, dostępnych w czasie działania programu ze wszystkich plików programu, w słowniku *settings*. W pliku *config/config.txt* istnieje możliwość ustalenia własnych wartości parametrów, bez konieczności znajomości języka Python. Sposób definiowania parametrów to: <klucz>=<wartość>, np. MODULARITY=5.

W pliku *config.txt*, prawda definiowana jest jako 1, fałsz jako 0. Później wartość parametru jest konwertowana na typ *bool* w języku Python.

W języku Python zapisane są ustawienia domyślne, które są nadpisywane przez ustawienia pliku *config.txt*. Można ustawić następujące parametry:

- LAMBDA - liczba chromosomów w każdej kolejnej populacji,
- MI - liczba chromosomów w populacji początkowej,
- CROSSOVER_PROB - prawdopodobieństwo krzyżowania,
- CROSSOVER_POINTS_COUNT - liczba punktów krzyżowania (domyślnie 2),
- MUTATION_PROB - prawdopodobieństwo mutacji,
- TARGET_FITNESS - docelowa wartość funkcji celu (jeden z warunków wystarczających zatrzymania),
- MAX_GENERATIONS - maksymalna liczba pokoleń (jeden z warunków wystarczających zatrzymania),
- MAX_STALE_GENERATIONS - maksymalna liczba niepoprawiających się pokoleń (jeden z warunków wystarczających zatrzymania),
- DISTRIBUTED - określa czy algorytm jest rozproszony (1), albo zagregowany (0),
- MODULARITY - modularność algorytmu,
- TOURNAMENT_COMPETITION_COUNT - wielkość szranek selekcji turniejowej,
- SEED - wartość ziarna generacji liczb pseudolosowych (podczas testowania ustawiona na 74),
- SHOW_LOG_ON_CONSOLE - czy propagować logi programu na konsolę użytkownika,
- MUTATION_FREQUENCY - jaka część populacji (od 0 do 1) ma zostać poddana mutacji. Wartość -1 powoduje wyłączenie tej funkcji i wtedy tylko 1 gen populacji zostaje zmutowany.

6 Charakterystyka implementacji algorytmu

6.1 Inicjalizacja populacji

Standardowa procedura inicjalizacji losuje wartości chromosomów z całej przestrzeni przeszukiwań i zwraca uzyskaną populację.

6.2 Główna pętla algorytmu

Zgodnie z działaniem algorytmu ewolucyjnego, główna pętla algorytmu, umieszczona w pliku *src/main.py*, wywołuje funkcje z pozostałych plików programu i steruje przebiegiem pętli. Algorytm pętli głównej został ukazany poniżej:

```
createInitPopulation(MI)
t = 0
while not stop_condition(t, stale_generations_count, lowest_fitness):
    pop = []
    for i in range(settings["LAMBDA"]):
        a = random.uniform(0, 1)
        if a < settings["CROSSOVER_PROB"]:
            chromosome = mutation(crossover(select(populations[t], k=2)))
            pop += chromosome
        else:
            chromosome = mutation(select(populations[t], k=1))
            pop += chromosome
    populations += pop
    t += 1
```

6.3 Mutacja

Jeżeli wartość parametru MUTATION_FREQUENCY jest równa -1, wtedy mutacja wykonywana jest na jednym, wybranym losowo z rozkładem jednostajnym, genie z populacji. Jeżeli wartość tego parametru jest w granicach od 0 do 1, taki procent liczby genów zostaje wylosowany ze zwracaniem z populacji i poddany mutacji. Mutacja polega na zmianie wylosowanego genu na wartość, wylosowaną ze wszystkich dostępnych wartości.

6.4 Krzyżowanie

Krzyżowanie jest wielopunktowe, zawsze z dwóch osobników powstaje jeden skrzyżowany. Liczbę punktów krzyżowania określa się w parametrze CROSSOVER_POINTS_COUNT, domyślnie równym 2.

6.5 Selekcja

Selekcja jest turniejowa. Z rozkładem jednostajnym bez zwracania losowane są różne chromosomy do szranek. Wielkość szranek jest określona w parametrze TOURNAMENT_COMPETITION_COUNT,

domyślnie równym 2.

6.6 Warunek zatrzymania

Algorytm zatrzymuje się, jeżeli wydarzy się co najmniej jedno z trzech zdarzeń:

- osiągnięta została maksymalna liczba pokoleń (MAX_GENERATIONS)
- osiągnięta została maksymalna liczba niepolepszających pokoleń (MAX_STALE_GENERATIONS)
- osiągnięta została pożądana wartość funkcji celu (TARGET_FITNESS)

6.7 Funkcja celu

Jest obliczana zgodnie z zasadami, opisanymi w zadaniu. Chromosom zagregowany składa się z liczb od 0 do 5, oznaczających indeksy dozwolonych ścieżek dla kolejnych zapotrzebowań. Chromosom rozproszony zaś, składa się z liczb od 0 do 1 dla każdej dozwolonej ścieżki dla kolejnych zapotrzebowań. Później finalna wartość procentowa przepływu daną dozwoloną ścieżką jest wartością znormalizowaną w ramach każdego zapotrzebowania.

7 Analiza wyników

Zgodnie z tabelą 7.1, przeprowadzono testy, aby sprawdzić dla różnych modularności i agregacji lub rozproszenia, jak zależą wartości parametrów algorytmu od prezentowanych wyników.

Mi,Lambda,Mut,Cross,Stale	Rozproszony	Modularność	Wynik
1000;1000;0,7;0,9;500	Agregacja	1	34624.0
1000;1000;0,7;0,9;500	Agregacja	1	22444,00
1000;1000;0,5;0,9;500	Agregacja	1	22444,00
1000;1000;0,95;0,95;500	Agregacja	1	22444,00
2000;2000;0,95;0,95;400	Agregacja	1	22444,00
20000;1500;0,95;0,95;300	Agregacja	1	22444,00
600;300;0,95;0,5;300	Agregacja	1	23438,00
600;300;0,95;0,95;600	Agregacja	1	22444,00
600;300;0,95;0,95;600	Agregacja	5	4494,00
600;300;0,95;0,95;600	Agregacja	5	4494,00
700;500;1;1;600	Agregacja	5	4493,00
700;500;1;1;600	Agregacja	100	231,00
700;500;1;0,8;600	Agregacja	100	231,00
50;100;1;1;600	Agregacja	100	230,00
50;100;1;0,5;600	Agregacja	100	245,00
50;100;0,5;1;600	Agregacja	100	230,00
50;100;0,5;0,8;600	Agregacja	100	230,00
50;100;0,5;0,8;600	Rozproszony	1	24012,00
1000;100;0,5;0,8;30	Rozproszony	1	25934,00
1000;200;0,5;0,8;30	Rozproszony	1	24862.00
50;40;0,5;0,8;30	Rozproszony	1	29184,00
50;40;1;1;150	Rozproszony	1	28993,00
50;100;0,9;0,9;80	Rozproszony	5	5410,00
1000;200;0,9;0,9;30	Rozproszony	5	5394,00
1000;200;0,9;0,9;30	Rozproszony	100	313,00
50;100;0,9;0,9;60	Rozproszony	100	332,00
200;50;0,9;0,9;250	Rozproszony	100	318,00

Tablica 7.1: Wyniki testów.