Algorytmy Genetyczne - projekt - telefon zaufania

Tomasz Maczek

Algorytmy Genetyczne
WFiIS

Styczeń 2024

1 Opis problemu

Jako temat projektu dostaliśmy za zadanie problemu z zakresu harmonogramowania. Jesteśmy odpowiedzialni za organizację pracy telefonu zaufania, przy którym dyżurują przez całą dobę, 7 dni w tygodniu psychologowie. Dostajemy ich dostępność w pliku wejściowym i na podstawie tego mamy wygenerować plik wyjściowy z harmonogramem.

Szczegóły problemu:

- W pliku wejściowym `dostepnosc.txt` każda linia jest w formacie **Imię Start-Koniec Typ**, gdzie:
  - **Imię** - imię psychologa,
  - **Start** - godzina o której może zacząć,
  - **Koniec** - godzina o której może skończyć,
  - **Typ** - **TAK** lub **EWENT** oznaczające odpowiednio czy jest w tym czasie w pełni dostępny czy jedynie w razie konieczności.
- Plik wyjściowy, `output.txt` wypisuje dane w formacie **Imię Start-Koniec** chronologicznie.
- Przedziały czasowe składają się z 10 minutowych bloków.

- Naszym celem jest jak największe zapełnienie grafiku psychologami, którzy są dostępni w danej godzinie tak aby było jak najmniej zmian w grafiku.

2 Implementacja

Program został napisany w języku `Python` z wykorzystaniem biblioteki `PyGad`. Problem został rozłożony na dni: dla każdego dnia tygodnia tworzony jest osobno grafik, który jest zapisywany w jednym pliku.
Dla każdego dnia na początku z użyciem biblioteki `Pandas` plik wejściowy zostaje wczytany do dataframe'a. Dla każdej unikalnej osoby w dniu zostaje dopasowany `person_index` liczony od 1, wartości `start` i `koniec` obliczone jako ilość minut od północy podzielona przez 10, oraz `dostepnosc` - 1 dla **TAK** i 0.5 dla **EWENT**.
Zostaje też stworzona tablica `dostepnosc` wymiarów ilość osób w dniu na ilość 10-minutowych bloków w dniu (144). Na jej podstawie tworzona jest populacja startowa.

2.1 Kodowanie

Chromosom jest rozmiaru 144 - ilość 10 minutowych bloków w 24 godzinach. Przyjmuje on wartości całkowite zawarte w `person_index` - przypisanie indeksu osoby do odpowiedniej pozycji chromosomu interpretujemy, że jest ona przypisana w tym czasie. Wartość 0 jest zarezerwowana dla braku rezerwacji - gdy nie ma dostępnego żadnego psychologa.

2.2 Populacja startowa

Aby usprawnić początkowe działanie algorytmu stworzono funkcję `custom_population` tworzącą na postawie argumentu `dostepnosci` (opisany wcześniej) populację startową o rozmiarze `size`. Populacja ta jest tworzona przez losowanie dla każdego bloku wartości z maksymalnie zapełnionego grafika osoby z tablicy `dostepnosci`. Na początku w tej populacji rozwiązania są bardzo słabej jakości, jednak wszystkie są możliwe do zastosowania, tj. nie ma przypisanych osób w niemożliwych dla nich przedziałach czasowych. Jest to też główne źródłó losowości w alogorytmie.

2.3 Funkcja dostosowania

Funkcja dostosowania zaimplementowana jest w `fitness_func`. Liczy ona dwa parametry zadane w treści projektu:

- `f` - zwiększa o 1 gdy w przedziale przypisany jest w pełni dostępny psycholog, o 0.5 gdy psycholog jest ewentualnie dostępny. Maksymalnie może przyjąć wartość 144.
- `g` - ilość zmian między psychologami w ciągu dnia. Gdy psycholog kończy zmianę bez następnego go zastępującego również liczymy to jako zmiana. Nie liczymy granic dnia jako zmian, więc maksymalnie w dniu mogą być 143 zmiany, gdy co 10 minut jest kto inny w grafiku.

Implementacja w `PyGad` pozwala nam na zwracanie dwóch wartości funkcji dopasowania, tutaj to jest bezpośrednio `f` oraz `144 - g` - tym większe im mniej zmian mamy.

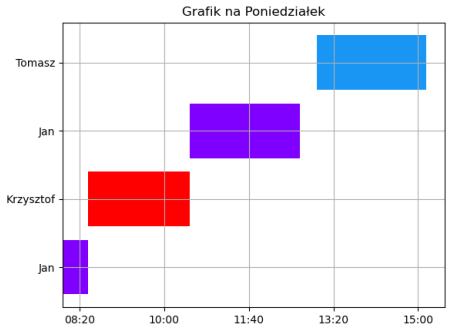
2.4 Parametry algorytmu genetycznego

Biblioteka `PyGad` pozwala nam na wygodne wybranie wielu parametrów algorytmu genetycznego. Tutaj podane są przekazane w tym wypadku:

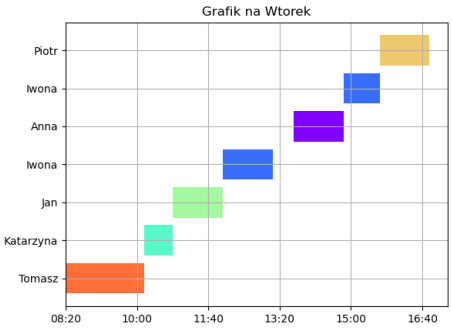
- ilość generacji = 150 - lekko zawyżona, ale dla bezpieczeństwa,
- ilość rodziców = 8,
- funkcja dopasowania opisana wyżej,
- populacja startowa opisana wyżej,
- operator selekcji **nsga2** - pozwala on na rozwiązywanie problemów optymalizacji wieloparametrowej,
- typ krzyżowania **uniform** - każdy bit jest wybierany od jednego z rodziców z równym prawdopodobieństwem,
- prawdopodobieństwo krzyżowania = 0.8,
- mutacja **losowa**,
- ilość mutowanych genów = 10,
- typ genu: całkowitoliczbowy.

3 Przykładowe wyniki

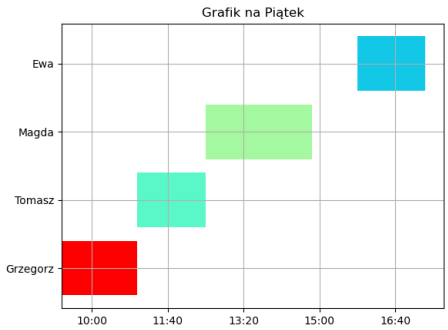
Program był testowany na stworzonym pliku `tutajnazwa`, na podstawie jego stworzone zostały przykładowe wykresy. Ta sama osoba jest reprezentowana jednym kolorem. Wykresy są tworzone również podczas uruchomienia programu. Poza tym tworzony jest plik `output.txt` z harmonogramem.



Rysunek 1: Przykładowy grafik na Poniedziałek



Rysunek 2: Przykładowy grafik na Wtorek



Rysunek 3: Przykładowy grafik na Piątek

4 Wnioski

- Z podanymi parametrami, szczególnie dzięki metodzie krzyżowania, algorytm bardzo szybko uzyskuje sensowne wyniki.
- Ustawianie dużego procentu lub ilości genów do mutacji miało raczej tendencję do psucia wyników, wrzucając komuś 10 minutową zmianę w miejscu, gdzie ktoś już ma blok czasowy. Z tego powodu mutacja została dość mocno ograniczona w tej implementacji.
- Powyższy problem można by próbować rozwiązać tworząc własną metodę mutacji, która mutowała by na postawie możliwości czasowych osób zmieniając duże bloki czasowe naraz. Biblioteka `PyGad` pozwala na tworzenie takich rozwiązań.
- Przez niski wpływ mutacji rozwiązania obliczone na tej samej populacji początkowej będą bardzo mocno podobne do siebie. Można temu prosto zaradzić tworząc za każdym razem nową populację startową.
- Wyniki zobrazowane na wykresach spełniają nasze oczekiwania. Nie są one idealne, co można by poprawić przez ulepszenie algorytmu genetycznego lub tworząc dodatkową funkcję do drobnych poprawek, jednak autor postanowił zostawić wyniki dokładnie takie jak zwraca algorytm.