Algorytmy rozwiązujące sudoku

Stanisław Olszewski

21 czerwca 2024

1 Wstęp

Program umożliwia użytkownikowi rozwiązywanie sudoku przy użyciu algorytmu genetycznego lub algorytmu roju cząsteczek. Dostępny jest plik zawierający pół miliona przykładowych sudoku do rozwiązania. Użytkownik może również wprowadzić własne sudoku, modyfikując plik `load_sudoku.py`.

Sudoku są rozwiązywane za pomocą odpowiednich funkcji fitness, które maksymalizują wartość funkcji w przypadku algorytmu genetycznego lub minimalizują ją dla algorytmu roju cząsteczek.

Program również oferuje możliwość wizualizacji rozwiązań. Pozwala to użytkownikowi na odczytanie sudoku w prosty i czytelny sposób. Poprawnie położone cyfry wyświetlane są w kolorze zielonym, a powtarzające się w kolorze czerwonym.

2 Instrukcja

Aby poprawnie uruchomić program należy użyć polecenia `python main.py` w folderze z projektem. Następnie będziemy mieli do wyboru następujące opcje:

- Sudoku użytkownika lub z przykładowe z datasetu
- Liczba prób rozwiązania
- Wybór algorytmu
- Czy generować wykres

```
> python main.py
1 - Sample sudoku
2 - User's sudoku?
1
Sample sudoku loaded with 47 missing values
Enter number of tries
10
1 - GA
2 - PSO
1
Draw plot? y/n
y
```

Rysunek 1: Przykładowe uruchomienie

3 Program

3.1 Dataset

Do mojego programu dołączyłem plik z 0.5 miliona przykładowych sudoku pochodzący z datasetu Kaggle. Pierwotnie zawierał on milion sudoku, lecz zdecydowałem się skrócić go o połowę aby zaoszczędzić miejsce na repozytorium.

3.2 Algorytm genetyczny

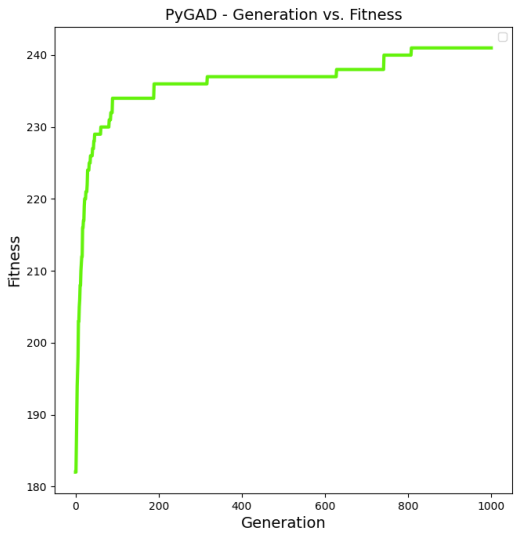
Po wielu próbach i testach udało mi się dobrać optymalne parametry algorytmu. Chromosom składa się się z genów, które są liczbami całkowitymi w zakresie 1-9. Jego długość jest dynamiczna i jest równa ilości brakujących miejsc w sudoku. Liczba generacji to 1000, a generacja składa się ze 100 chromosomów. Dokładniejsze parametry możemy znaleźć w pliku `ga.py`. Algorytm o takich parametrach jest w stanie rozwiązać większość sudoku.

Funkcja fitness najpierw łączy sudoku z pustymi polami z potencjalnym rozwiązaniem, a następnie oblicza wynik dla rozwiązania.

Obliczanie wyniku dla każdego rzędu, kolumny i kwadratu 3x3 dodaje 9 i odejmuje liczbę powtórzonych cyfr. Przykładowo jeśli w rzędzie powtórzy się tylko cyfra 5 do wyniku zostanie dodane 8. Maksymalny wynik jaki może osiągnąć funkcja to 243, który oznacza poprawnie rozwiązane sudoku.

```python
def fitness_func(ga_instance, solution, solution_idx):
    combined = combine(sudoku, solution)
    score = 0
    for i in range(len(sudoku)):
        score += 9 - (col_numof_missing_and_repeated(combined, i)[0])
    for i in range(len(sudoku)):
        score += 9 - (row_numof_missing_and_repeated(combined, i)[0])
    for x in [0, 3, 6]:
        for y in [0, 3, 6]:
            score += 9 - (square_numof_missing_and_repeated(combined, x, y)[0])
    return score
```

Rysunek 2: Funkcja fitness GA



Rysunek 3: Wykres fitness GA

Na 10 przykładowych uruchomień udało się rozwiązać sudoku 6 razy, a średni czas wykonania wynosił 7.2 sekundy.

3.3 Rój cząsteczek

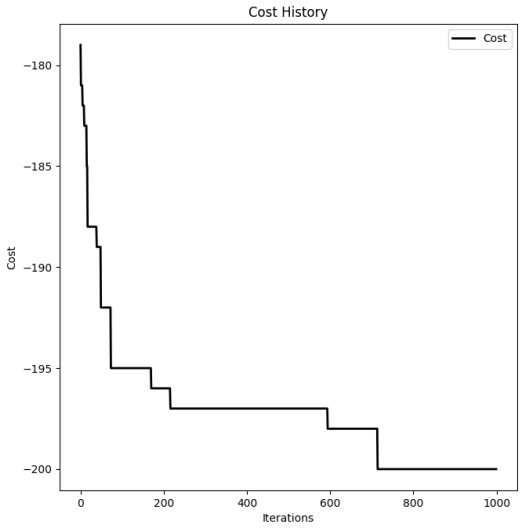
W tym przypadku funkcja fitness wygląda bardzo podobnie. Różnica polega na tym, że funkcja próbuje znaleźć minimum, zatem końcowy wynik jest negowany.

Problemem, który napotkałem przy użyciu roju cząsteczek jest to, że operuje on na liczbach zmiennoprzecinkowych. Aby go ominąć postanowiłem zaokrąglać liczby z przedziału 0.51-9.49 do najbliższych liczb całkowitych.

Pomimo takiego rozwiązania i wielu prób z doborem odpowiednich parametrów nie udało się mi się zoptymalizować algorytmu, aby był w stanie poprawnie rozwiązać sudoku.

```python
def optimize_fn(solutions):
    result = []
    for solution in solutions:
        combined = combine(sudoku, [round(x) for x in solution])
        score = 0
        for i in range(len(combined)):
            score += 9 - (col_numof_missing_and_repeated(combined, i)[0])
        for i in range(len(combined)):
            score += 9 - (row_numof_missing_and_repeated(combined, i)[0])
        for x in [0, 3, 6]:
            for y in [0, 3, 6]:
                score += 9 - (square_numof_missing_and_repeated(combined, x, y)[0])
        result.append(-score)
    return result
```

Rysunek 4: Funkcja fitness PSO



Rysunek 5: Wykres fitness PSO

Podczas 10 prób nie udało się ani razu rozwiązać sudoku, a średni czas wykonania wynosił 17.5 sekundy.

4 Podsumowanie

Podsumowując, w ramach projektu udało mi się stworzyć algorytm umożliwiający rozwiązywanie sudoku przy pomocy algorytmu genetycznego. Niestety jego dokładność nie jest idealna, jednak pozwala na rozwiązanie większości łamigłówek, lecz nie zawsze udaje się to przy pierwszym podejściu.

Zaadaptowanie algorytmu roju cząsteczek do tego zadania okazało się zaś niepowodzeniem. Nie był on w stanie rozwiązać sudoku, pomimo modyfikacji parametrów i funkcji fitness.

5 Bibliografia

- Dataset [Czerwiec 2024]
- ChatGPT [Czerwiec 2024]
- PyGAD [Czerwiec 2024]
- PySWARMS [Czerwiec 2024]