Inteligencja obliczeniowa - projekt II

Artur Jakubowski

Czerwiec 2024

Contents

Part I
Przedstawienie projektu

1 Ogólne założenia projektu

Projekt skupia się na stworzeniu jak najbardziej optymalnego planu dla pracowników szpitala, biorąc pod uwagę parametry takie jak specjalizacja danego pracownika, wymagane umiejętności do wykonania zadania, czas pracy pracownika w konkretnych dniach tygodnia, czas wykonywania zadania oraz pensja pracownika.

2 Bazy danych

2.1 Workers

Jest to lista zapisana w formacie '.json', w której znajduje się lista wszystkich pracowników szpitala. Baza została wygenerowana losowo, a jej wielkość jest zależna od parametrów wejściowych. Każde rekord w tej bazie zawiera:

- Imię danego pracownika
- Jego profesję
- Dni tygodnia zawierające godziny, w których dany pracownik jest dostępny
- Pensję godzinową pracownika

Każdy rekord ma następującą strukturę:

```
{
        "workerName": "Mr. Nicholas Wolf",
        "profession": "Hospital Administrator",
        "workHours": {
            "0": "02:00-10:00",
            "1": "02:00-10:00",
            "2": "02:15-10:15",
            "3": "00:45-08:45",
            "4": "03:00-11:00",
            "5": "07:00-15:00",
            "6": "02:00-10:00"
        },
        "hourlySalary": 90
    }
```

2.2 Schedule

Schedule zawiera posegregowaną listę (według dni tygodnia) wszystkich zadań przewidzianych na dany dzień. Baza została wygenerowana automatycznie. Każde zadanie zawiera pole:

- Czas trwania danego zadania
- Wymaganą rolę do jego ukończenia
- Nazwę danego zadania

Każdy rekord ma następującą strukturę:


```
{
            "day": 1,
            "events": [
                {
                    "time": "09:00-10:30",
                    "role": "Physiotherapist",
                    "activity": "Rehabilitation Session"
                },
                {
                    "time": "11:00-12:30",
                    "role": "Radiology Technician",
                    "activity": "Imaging Procedure"
                },
                {
                    "time": "13:00-14:00",
                    "role": "Dietitian",
                    "activity": "Nutritional Consultation"
                },
                {
                    "time": "15:00-16:00",
                    "role": "Hospital Executive",
                    "activity": "Administrative Meeting"
                }
            ]
        },
```


2.3 Professions

Professions zawiera listę wszystkich dostępnych profesji pracowników oraz powiązane z nimi typy zadań jakie mogą wykonywać. Baza została wygenerowana automatycznie na podstawie dostępnych profesji z bazy 'Workers'.

Fragment bazy danych wygląda następująco:


```
{
    "Physician": ["Doctor", "Surgeon", "General Practitioner"],
    "Nurse": ["Registered Nurse", "Licensed Practical Nurse", "Nurse Practitioner"],
    "Surgeon": ["Doctor", "Physician", "General Surgeon"],
    "Anesthesiologist": ["Doctor", "Physician", "Anesthesia Nurse"],
    "Pharmacist": ["Clinical Pharmacist", "Hospital Pharmacist", "Pharmacy
        Technician"],
    "Radiologist": ["Medical Imaging Technologist", "Radiology Technician"],
    "Medical Technologist": ["Clinical Laboratory Scientist", "Medical Laboratory
        Technician"],
    "Respiratory Therapist": ["Respiratory Technician", "Respiratory Nurse"],
    ...
```

3 Szczegółowy opis działania projektu

Projekt wykorzystuję algorytm genetyczny z biblioteki pygad, aby znaleźć jak najbardziej optymalny plan zadań dla szpitala. Program w pierwszej kolejności stara się przyporządkować jakiegokolwiek pracownika do danego zadania, a następnie sprawdza:

- Czy jego **kwalifikacje pozwalają mu wykonać zadanie** (pozwala mu na to jego specjalizacja)
- **Czy pracownik pracuje w szpitalu w konkretnych godzinach**
- **Czy pracownik, już nie ma zaplanowanego innego zadania** w podanych godzinach
- **Program dąży do tego aby zminimalizować wydatki dla szpitala** (wybierając w pierwszej kolejności pracowników, których pensja jest najmniejsza)

Plan jest generowany dla wybranego tygodnia, a lista wydarzeń w danym tygodniu jest pobierana z pliku 'schedule.json'

3.1 Opis funkcji fitness

Funkcja fitness składa się z dwóch części. Pierwsza przechowuje całkowity wynik jaki otrzymuje program po przydzieleniu wszytskich zadań oraz iteruje po wszystkich zadaniach.

Pierwsza część:

```
1     def checkSolution(self, solution):
2
3         totalReward = STARTING_REWARD
4
5         for index, workerNumber in enumerate(solution):
6             worker =
                  projectTypes.Worker.convertFromDictionary(self.workers[int(workerNumb
7             totalReward += self.checkSolutionQuality(worker,
                  self.schedule[index])
8
9         return totalReward
```

Druga część ma za zadanie obliczyć wartość nagrody dla każdego pojedynczego zadania bazując na czy program wybrał odpowiedniego pracownika jeśli chodzi o czas pracy, już zaplanowane zadania, jego profesję oraz bierze pod uwagę pensję pracownika

Druga część:

```
def checkSolutionQuality(self, worker: projectTypes.Worker, event: dict) -> int:
        reward = STARTING_TASK_REWARD

        eventWorkTime = projectTypes.WorkTime.convertFromString(event["time"])

        if (not worker.checkHoursAvaibility(event["day"], eventWorkTime)):
            reward -= WRONG_HOURS_PENALTY
        if (not worker.checkProfession(event["role"])):
            reward -= WRONG_PROFESSION_PENALTY
        if (worker.isBusy(event["day"], eventWorkTime)):
            reward -= ALREADY_BUSY_PENALTY

        worker.assignWorkTime(event["day"], eventWorkTime)

        reward -= worker.hourlySalary * SALARY_MULTIPLIER

        return reward
```

3.2 Wizualizacja planu

Po zakończeniu działania algorytmu genetycznego, wynikiem jest tablica, która zawiera indentyfikatory poszczególnych pracowników przypisanych do poszczególnych zadań:

```
    def initializePygad(self):
        self.pygad = pygad.GA(
            num_generations=200,
            num_parents_mating=3,
            fitness_func=self.fitnessFunc,
            sol_per_pop=50,
            num_genes=len(self.schedule),
            init_range_low=0,
            init_range_high=2,
            gene_space=[workerIndex for workerIndex in range(len(self.workers))],
        )

        self.pygad.run()

        self.solution, self.solution_fitness, _ = self.pygad.best_solution()

        self.pygad.plot_fitness(save_dir="answers.png")

        print(f"\n---------Results:---------\n")
        print(f"Solution:\n{self.solution}\n")
        print(f"FINAL FITNESS: {self.solution_fitness}")

        self.drawPlan()
```

W ostatnim etapie tworzenia wykresu przypisujemy odpowiendie wartości do osi X i Y, oraz dodajemy opisy zarówno aktywności, jak i profesji poszczególnych pracowników:

```
    def drawPlan(self):
        dataFrames = []

        for eventIndex, workerIndex in enumerate(self.solution):
            dataFrames.append(
                dict(Task=f"{self.schedule[eventIndex]['activity']}",
                     Start=f"2024-01-0{self.schedule[eventIndex]['day']
                        + 1}
                        {self.schedule[eventIndex]['time'].split('-')[0]}",
                     Finish=f"2024-01-0{self.schedule[eventIndex]['day']
                        + 1}
                        {self.schedule[eventIndex]['time'].split('-')[1]}",
                     Name=f"{self.workers[int(workerIndex)]['workerName']}\n({self.w


        df = pd.DataFrame(dataFrames)


        fig = px.timeline(df, x_start="Start", x_end="Finish",
            y="Name", text="Task")
        fig.update_yaxes(autorange="reversed")
        fig.show()
```

4 Testowanie działania algorymtmu

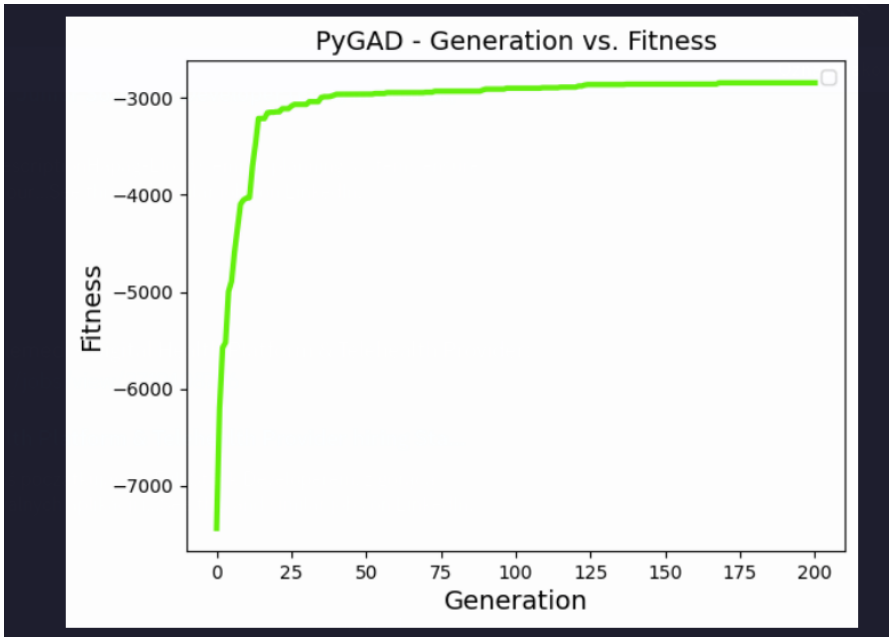
4.1 Testowanie ma małej ilości danych

Ten test polegał na sprawdzeniu działania algorytmu na stosunkowo małych danych (28 zadań, na wszytstkie 7 dni).

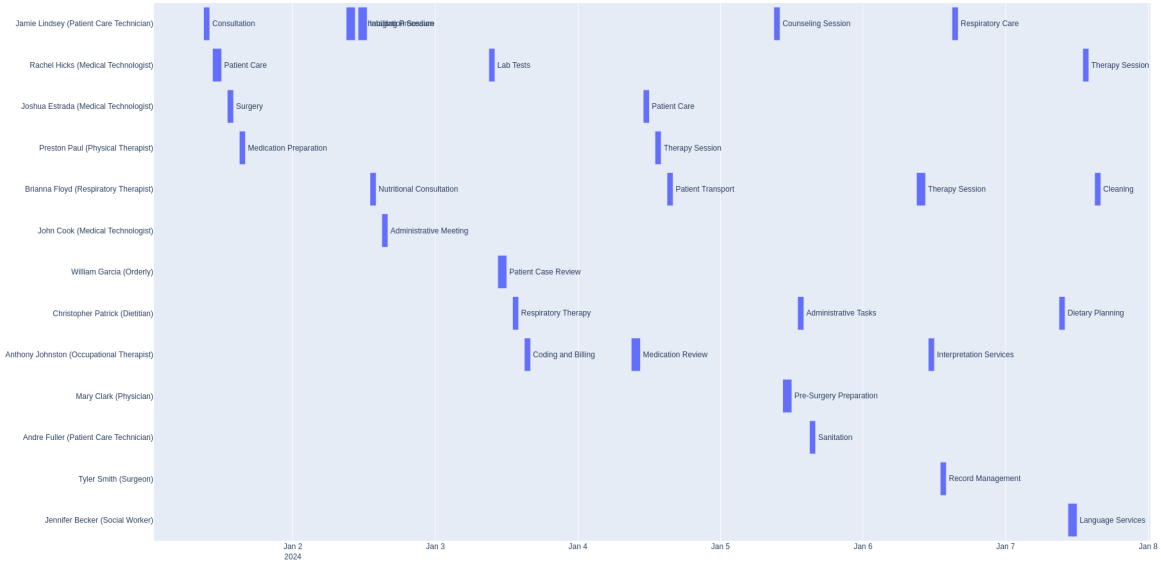
Wyniki algorytmu:

```
---------Results:---------

Solution:
[39. 45. 13. 37. 39. 39.  3. 42. 45. 33.  4. 21. 21. 13. 37.  3. 39. 47.
  4. 19.  3. 21. 24. 39.  4. 32. 45.  3.]

FINAL FITNESS: -2842.0
```

Wykres funckji fitness:



Wizualizacja planu:



4.2 Podsumowanie wyników testu

Na małej ilości danych algorytm poradził sobie bardzo dobrze, żadna osoba nie ma kilku nakładających się na siebie zadań, kązde zadanie zostało przyporządkowane konkretnej osobie, żadna osoba nie wykonuje także dwóch różnych zadań w tym samym czasie oraz program odpowiednio dobiera profesje poszczególnych pracowaników do konkretnych zadań.

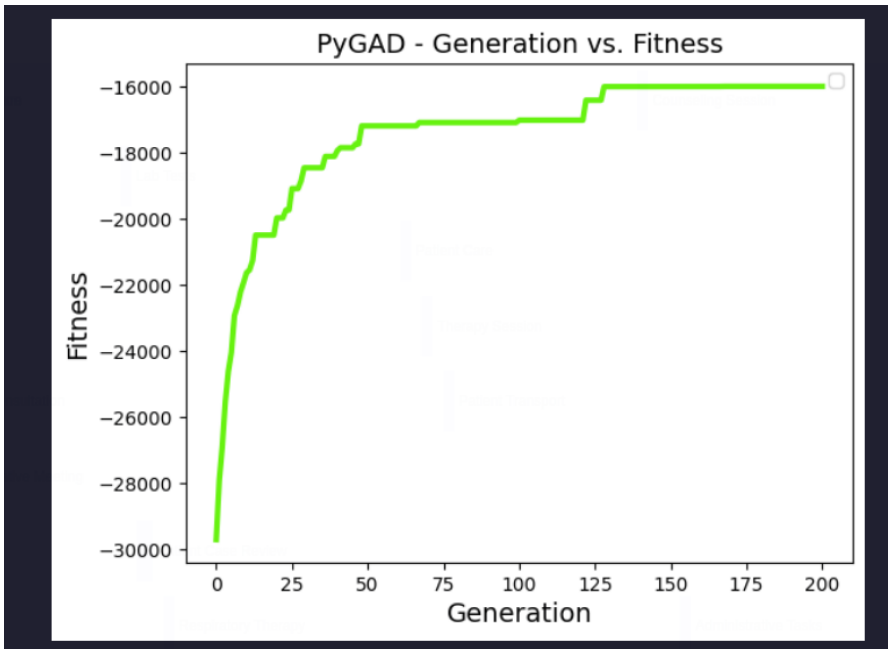
4.3 Testowanie działania algorytmu na dużej ilości danych

Test ten polegał na przetestowaniu działania algorytmu na dużej ilośći danych (około 100 zadań na wszystkie 7 dni)

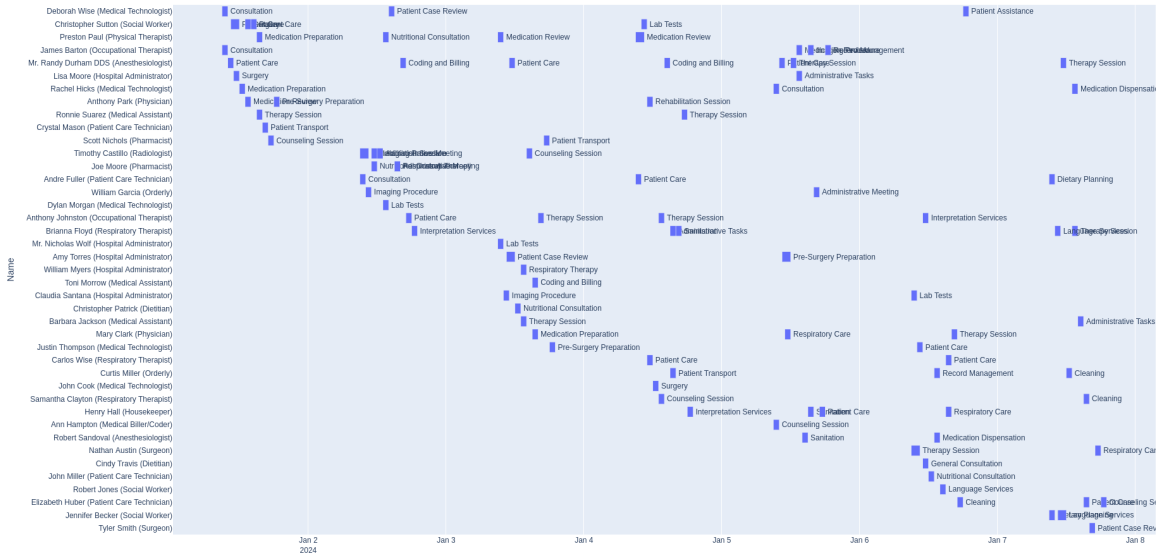
Wyniki algorytmu:

```
---------Results:---------

Solution:
[48. 35. 35. 37. 15.  5.  6. 45. 23. 35. 28.  2.  9. 23. 36. 36. 37. 30.
 19. 33. 30. 36. 10. 48. 30.  5. 21.  3. 16. 29. 22. 25. 37.  1.  5.  4.
 12. 36. 47. 21.  9. 20. 37. 26. 21. 43. 19. 35. 23. 42.  8.  5.  3.  3.
 28. 44. 40. 29.  6. 44. 45.  5. 47.  5. 15. 18. 15. 33. 44. 15. 41. 21.
 43. 44.  1. 20. 34.  0. 18. 38. 26. 47. 27. 48. 32. 32.  3.  8. 19.  3.
  5. 43. 45. 12. 27. 24. 41. 27.]

FINAL FITNESS: -15981.5
```

Wykres funckji fitness:



Wizualizacja planu:



4.4 Podsumowanie wyników z dużej bazy danych

Algorytm poradził sobie bardzo dobrze także z większą ilością danych, jednak tym razem z racji znacznie większej ilości zadań algorytm znacznie częściej wykorzystał tych samych pracowników do wykonywania różnych zadań i ogólnie skorzystał z znacznie większej ilości pracowników. Jednak algorytm nadal skonstruował ogólnie dobry plan, znajduje się w nim tylko jeden przypdaek gdzie zadania danego pracownika się na siebie nakładają i robią to tylko w częściowo (1 - zadanie 11:00-12:30, 2 - zadanie 12:00-13:00). Algorymt jednak znacznie wydłużył swój czas wykonywania z kilku sekund do około minuty.

5 Parametry użycie do przeprowadzenia testów

Parametry po odpowiednich testach wyglądają następująco:

```
1
2  def initializePygad(self):
3          self.pygad = pygad.GA(
4              num_generations=200,
5              num_parents_mating=3,
6              fitness_func=self.fitnessFunc,
7              sol_per_pop=50,
8              num_genes=len(self.schedule),
9              init_range_low=0,
10             init_range_high=2,
11             gene_space=[workerIndex for workerIndex in
                   range(len(self.workers))],
12         )
```

- **Number of generation = 200** - Zwiększenie ilości generacji znacznie zwiększa czas wykonywania, a dla dużej bazy danych wystarczające jest około 150 iteracji aby osiągnąć zadowalający wynik, także do testów dla reprodukowalnych wyników zostało przyjęta wortść równa 200
- **Number of paremnst mating = 3** - Ilość rodziców, którzy się krzyżują jest stosunkowo niska, poniważ zwiększona ich ilość wpływa końcowo negatywnie na wartość funkcji fitness. Jest to najoptymalniejsza znaleziony parametr dla większości przypadków.
- **Number of solutions per population = 50** - Zwiększenie wielkości populacji może w bardziej skomplikowanych przypadkach podwyższyć wartość funkcji fitness jednak dla wybranej bazy danych (100 rekordów), wartość ta okazała się dobrym kompromisem pod względem czasu wykonania programu, a jakością końcowych wyników. Dla zwiększonej bazy danych zadań lub pracowników, należałoby zwiększyć tą wartość.

6 Podsumowanie projektu

Projekt spełnił sowje początkowe założenia i dobrze radzi sobie z przedstwaionym problemem, a jego końcowe wyniki są stosunkowo optymalne dla przedstawionych problemów, jednak wraz ze zwiększaniem ilośći zadań jak i pracowników, zancząco rośnie czas wykonania algorytmu. Podsumowując algorytm potrafi skonstruować plan bliski optymalnego, bazując na dostępnych pracownikach jak i zadaniach podzielonych na konkretne dni.